Traducido del "?" al español - www.onlinedoctranslator.com

Guía del programador de GnuCOBOL

Para la versión 3.2 - Final [05 de febrero de 2025 a las 00:20 GMT.], para 3.2, 3.3 y 4.0 (solo parcial y tentativa).

Gary L. Cutler (cutlergl@gmail.com**) 2009 - 2014.**
Vincent B. Coen (vbcoen@gmail.com**) para todas las actualizaciones 2014 - 2025.**

Este manual documenta la compilación GnuCOBOL 3.2 - Final, 05 de febrero de 2025 a las

00:20 GMT. Documento: Copyright 2009-2014 Gary L. Cutler y FSF (Free Software

Foundation).
Actualizaciones: Copyright 2014-2025 Vincent B. Coen
y FSF. Contribuciones: Eugenio Di Lorenzo 2024 - 2025.

Los autores y propietarios de los derechos de autor del lenguaje de programación COBOL utilizado en este documento:

FLOW-MATIC (marca registrada de Sperry Rand Corporation) Programación para Univac(R) I y II. Sistemas de automatización de datos con derechos de autor de 1958 y 1959 de SperryRand Corporation; IBM commercial translate form F28-8013, copyrighted 1959 by IBM; FACT DSI27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, han autorizado específicamente el uso de este material en su totalidad o en parte de las especificaciones COBOL. Dicha autorización se extiende a la reproducción y uso de las especificaciones COBOL en manuales de programación o publicaciones similares.

Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU [FDL], Versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation; con Sección “Introducción” invariable, sin textos de portada y sin textos de contraportada. Se incluye una copia de la licencia en la sección titulada “Licencia de Documentación Libre de GNU”.

Se concede permiso para realizar y distribuir copias textuales de este manual siempre que se conserve el aviso de derechos de autor y este aviso de permiso en todas las copias.

Se concede permiso para copiar y distribuir versiones modificadas de este manual bajo las condiciones para la copia textual, siempre que todo el trabajo derivado resultante se distribuya bajo los términos de un aviso de permiso idéntico a este.

Se concede permiso para copiar y distribuir traducciones de este manual a otro

idioma, bajo las condiciones anteriores para las versiones modificadas, excepto que este aviso de permiso pueda figurar en una traducción aprobada por la Free Software Foundation.

Tabla de contenido

Guía del programador de GnuCOBOL 1

1 Introducción 3

1.1 Fuentes de referencia adicionales 3
1.2 Introducción a COBOL 4
1.2.1 ¿Por qué deberías aprender COBOL? 4
1.2.2 Productividad del programador 6
1.3 Entonces, ¿qué es GnuCOBOL? 7

2 Fundamentos de COBOL 9

2.1 El lenguaje COBOL: conceptos básicos 9
2.1.1 Palabras reservadas del idioma 9
2.1.2 Palabras definidas por el usuario 10
2.1.3 Insensibilidad a mayúsculas y minúsculas 10
2.1.4 Legibilidad de los programas 11
2.1.5 Divisiones Organizan Programas 13
2.1.6 Cuadernos de notas 13
2.1.7 Datos estructurados 13
2.1.8 Archivos 14
2.1.9 Manejo de mesas 17
2.1.10 Ordenar y fusionar datos 18
2.1.11 Manipulación de cadenas 18
2.1.12 Funciones de formato de pantalla 20
2.1.12.1 Una pantalla de muestra 21
2.1.12.2 Paleta de colores y atributos de vídeo 21
2.1.13 Características del generador de informes 24
2.1.14 Inicialización de datos 25
2.1.15 Convenciones de diagramas de sintaxis 26
2.1.16 Formato de las líneas fuente del programa 27
2.1.17 Estructura del programa 30
2.1.18 Comentarios 32
2.1.19 Literales 33
2.1.19.1 Literales numéricos 33
2.1.19.2 Literales alfanuméricos 33
2.1.19.3 Constantes figurativas 34
2.1.20 Puntuación 35
2.1.21 Interfaz con otros entornos 35
2.1.22 Referencias de la tabla 37
2.1.23 Calificación de nombres de datos 38
2.1.24 Modificadores de referencia 39
2.1.25 Expresiones aritméticas 40
2.1.26 Expresiones booleanas 43
2.1.27 Expresiones condicionales 44
2.1.27.1 Nombres de condiciones 44
2.1.27.2 Condiciones de la clase 45
2.1.27.3 Condiciones de la señal 47

2.1.27.4 Condiciones de estado del interruptor ... 48
2.1.27.5 Condiciones de relación ... 49
2.1.27.6 Condiciones combinadas ... 51
2.1.27.7 Condiciones Negadas ... 52
2.1.28 Uso de los puntos ... 53
2.1.29 Uso de construcciones VERBO/FINAL DEL VERBO ... 54
2.1.30 Acceso simultáneo a archivos ... 56
2.1.30.1 Intercambio de archivos ... 56
2.1.30.2 Bloqueo de registros ... 58

3 CDF - Instalación de dirección del compilador ... 61

3.1 >>LLAMADA-CONVENCION ... 62
3.2 COPIAR ... 63
3.3 REEMPLAZAR ... 65
3.4 >>DEFINIR ... 67
3.5 >>SI ... 68
3.6 >>CONFIGURAR ... 70
3.7 >>FUENTE ... 72
3.8 >>GIRAR ... 73
3.9 >>El ... 74
3.10 >>PANTALLA ... 75
3.11 >>PAGINA ... 76
3.12 >>LISTADO ... 77
3.13 >>SEGUNDOS INTERCIPALES ... 78
3.14 $ Directivas ... 79

4 DIVISIÓN DE IDENTIFICACIÓN ... 81

5 DIVISIÓN DE MEDIO AMBIENTE ... 85

5.1 SECCIÓN DE CONFIGURACIÓN ... 86
5.1.1 ORDENADOR FUENTE ... 87
5.1.2 COMPUTADORA OBJETO ... 88
5.1.3 NOMBRES ESPECIALES ... 90
5.1.3.1 Alfabeto-Nombre-Cláusula ... 94
5.1.3.2 Cláusula de definición de clase ... 96
5.1.3.3 Cláusula de definición de conmutación ... 97
5.1.3.4 Cláusula de caracteres simbólicos ... 98
5.1.4 REPOSITORIO ... 99
5.2 SECCIÓN DE ENTRADA-SALIDA ... 101
5.2.1 SELECCIONAR ... 102
5.2.1.1 ORGANIZACIÓN SECUENCIAL ... 109
5.2.1.2 ORGANIZACIÓN LINEA SECUENCIAL ... 110
5.2.1.3 ORGANIZACIÓN RELATIVA ... 112
5.2.1.4 ORGANIZACIÓN INDEXADA ... 114
5.2.2 MISMA ÁREA DE REGISTRO ... 116
5.2.3 ARCHIVOS MÚLTIPLES ... 117

6 DIVISIÓN DE DATOS ... 119

6.1 Principios de definición de datos ... 120
6.2 SECCIÓN DE ARCHIVO ... 122
6.2.1 Descripción de archivo/ordenación ... 123
6.2.2 ARCHIVO-SECCIÓN-Elemento-de-datos ... 126

6.3 SECCIÓN DE TRABAJO Y ALMACENAMIENTO 128
6.4 SECCIÓN DE ALMACENAMIENTO LOCAL 130
6.5 SECCIÓN DE ENLACE 132
6.6 SECCIÓN DE INFORME 135
6.6.1 Definiciones de grupos de informes 138
6.6.2 SECCIÓN DEL INFORME Elementos de datos 140
6.7 SECCIÓN DE PANTALLA 142
6.8 Elementos de datos especiales 145
6.8.1 01-Constantes de nivel 145
6.8.2 Elementos de datos de nivel 66 149
6.8.3 Elementos de datos de nivel 77 150
6.8.4 Elementos de datos de nivel 78 151
6.8.5 Elementos de datos de nivel 88 153
6.9 Cláusulas de descripción de datos 154
6.9.1 CUALQUIER LONGITUD 154
6.9.2 CUALQUIER NUMÉRICO 155
6.9.3 AUTO 156
6.9.4 SALTO AUTOMÁTICO 157
6.9.5 TERMINACIÓN DEL COCHE 158
6.9.6 COLOR DE FONDO 159
6.9.7 ANTES DE TIEMPO 160
6.9.8 BASADO 161
6.9.9 BIP 162
6.9.10 CAMPANA 163
6.9.11 BLANCO 164
6.9.12 EN BLANCO CUANDO ES CERO 165
6.9.13 PARPADEAR 166
6.9.14 COLUMNA 167
6.9.15 CONSTANTE 169
6.9.16 POR DEFECTO 170
6.9.17 CHEQUE VACÍO 171
6.9.18 BORRAR 172
6.9.19 EXTERNO 173
6.9.20 FALSO 174
6.9.21 COLOR DE PRIMER PLANO 175
6.9.22 DE 176
6.9.23 LLENO 177
6.9.24 GLOBAL 178
6.9.25 GRUPO INDICA 179
6.9.26 DESTACAR 180
6.9.27 JUSTIFICADO 181
6.9.28 LÍNEA IZQUIERDA 183
6.9.29 COMPROBACIÓN DE LONGITUD 184
6.9.30 LÍNEA 185
6.9.31 MÁS BAJO 187
6.9.32 LUZ BAJA 188
6.9.33 SIGUIENTE GRUPO 189
6.9.34 NO-ECHO 190
6.9.35 OCURRE 191
6.9.36 SOBRELINEA 194
6.9.37 IMAGEN 195
6.9.38 PRESENTE CUANDO 202
6.9.39 INMEDIATO 203

6.9.40 PROTEGIDO 204
6.9.41 REDEFINE 205
6.9.42 CAMBIOS DE NOMBRES 206
6.9.43 REQUERIDO 207
6.9.44 VIDEO INVERSO 208
6.9.45 MISMO QUE 209
6.9.46 SEGURO 211
6.9.47 LA SEÑAL ES 212
6.9.48 TAMAÑO 213
6.9.49 FUENTE 214
6.9.50 NOMBRES ESPECIALES. 215
6.9.51 SUMA DE 216
6.9.52 SINCRONIZADO 217
6.9.53 SE ACABÓ EL TIEMPO 219
6.9.54 A 220
6.9.55 TIPO 221
6.9.56 Definición de tipo 225
6.9.57 SUBRAYAR 226
6.9.58 SUPERIOR 227
6.9.59 USO 228
6.9.60 USANDO 238
6.9.61 VALOR 239

7 DIVISIÓN DE PROCEDIMIENTOS 243

7.1 DIVISIÓN DE PROCEDIMIENTOS UTILIZANDO 244
7.2 DIVISIÓN DE PROCEDIMIENTOS ENCADENAMIENTO 246
7.3 DIVISIÓN DE PROCEDIMIENTOS DEVOLUCIÓN 248
7.4 DIVISIÓN DE PROCEDIMIENTOS Secciones y Párrafos 249
7.5 DECLARATIVOS 250
7.6 Cláusulas comunes en sentencias ejecutables 253
7.6.1 AL FINAL + NO AL FINAL 253
7.6.2 CORRESPONDIENTE 254
7.6.3 CLAVE INVÁLIDA + CLAVE NO INVÁLIDA 255
7.6.4 CON EXCEPCIÓN + SIN EXCEPCIÓN 256
7.6.5 EN DESBORDAMIENTO + NO EN DESBORDAMIENTO 256
7.6.6 ERROR DE TAMAÑO + ERROR DE TAMAÑO NO 257
7.6.7 REDONDEADO 257
7.7 Registros especiales 260
7.8 Sentencias GnuCOBOL 263
7.8.1 ACEPTAR 263
7.8.1.1 ACEPTAR DESDE LA CONSOLA 263
7.8.1.2 ACEPTAR DESDE LA LÍNEA DE COMANDOS 264
7.8.1.3 ACEPTAR DEL MEDIO AMBIENTE 266
7.8.1.4 ACEPTAR elemento de datos 267
7.8.1.5 ACEPTAR DESDE FECHA/HORA 277
7.8.1.6 ACEPTAR DESDE Screen-Info 278
7.8.1.7 ACEPTAR DESDE Runtime-Info 279
7.8.1.8 ACEPTAR OMITIDO 280
7.8.1.9 ACEPTAR DESDE EL ESTADO DE EXCEPCIÓN 281
7.8.2 AGREGAR 282
7.8.2.1 AÑADIR 282
7.8.2.2 AGREGAR DONACIONES 284
7.8.2.3 AGREGAR CORRESPONDIENTE 286

7.8.3 ASIGNAR ... 287
7.8.4 ALTERAR ... 289
7.8.5 LLAMAR ... 290
7.8.6 CANCELAR ... 294
7.8.7 CERCA ... 295
7.8.8 COMPROMETERSE ... 296
7.8.9 CALCULAR ... 297
7.8.10 CONTINUAR ... 300
7.8.11 BORRAR ... 301
7.8.12 MOSTRAR ... 302
7.8.12.1 MOSTRAR EN EL dispositivo ... 302
7.8.12.2 MOSTRAR EN LA LÍNEA DE COMANDOS ... 303
7.8.12.3 MOSTRAR EN EL NOMBRE DEL ENTORNO ... 304
7.8.12.4 MOSTRAR elemento de datos ... 305
7.8.12.5 Elemento de datos DISPLAY (Microsoft v1-v2) ... 312
7.8.13 DIVIDIR ... 313
7.8.13.1 DIVIDIR EN ... 313
7.8.13.2 DIVIDIR EN DAR ... 315
7.8.13.3 DIVIDIR DANDO ... 317
7.8.14 ENTRADA ... 319
7.8.15 EVALUAR ... 321
7.8.16 EXAMINAR ... 325
7.8.17 ANEXO ... 327
7.8.18 SALIDA ... 329
7.8.19 GRATIS ... 332
7.8.20 GENERAR ... 333
7.8.21 REGRESAR ... 334
7.8.22 IR A ... 335
7.8.22.1 Sencillo IR A ... 335
7.8.22.2 IR A DEPENDE DE ... 336
7.8.23 SI ... 338
7.8.24 INICIALIZAR ... 339
7.8.25 INICIADO ... 343
7.8.26 INSPECCIONAR ... 344
7.8.27 GENERAR JSON ... 349
7.8.28 ANÁLISIS JSON ... 350
7.8.29 UNIR ... 351
7.8.30 MOVER ... 354
7.8.30.1 Movimiento simple ... 354
7.8.30.2 MOVER CORRESPONDIENTE ... 355
7.8.31 MULTIPLICAR ... 356
7.8.31.1 MULTIPLICA POR ... 356
7.8.31.2 MULTIPLICA LA DONACIÓN ... 357
7.8.32 ABIERTO ... 358
7.8.33 LLEVAR A CABO ... 360
7.8.33.1 Procedimiento REALIZAR ... 360
7.8.33.2 RENDIMIENTO en línea ... 362
7.8.33.3 DIVERSO ... 363

7.8.34 LEER ... 365
7.8.34.1 LECTURA secuencial ... 365
7.8.34.2 LECTURA Aleatoria ... 367
7.8.35 LISTO PARA TRAZAR ... 369
7.8.36 LIBERAR ... 370

7.8.37 RESTABLECER TRAZA ... 371
7.8.38 DEVOLVER ... 372
7.8.39 VOLVER A ESCRIBIR ... 373
7.8.40 RETROCESO ... 375
7.8.41 BUSCAR ... 376
7.8.42 BUSCAR TODO ... 377
7.8.43 COLOCAR ... 379
7.8.43.1 Establecer entorno ... 379
7.8.43.2 Puntero de programa SET ... 380
7.8.43.3 ESTABLECER DIRECCIÓN ... 381
7.8.43.4 Índice SET ... 382
7.8.43.5 CONFIGURAR ARRIBA/ABAJO ... 383
7.8.43.6 Nombre de la condición SET ... 384
7.8.43.7 Interruptor SET ... 385
7.8.43.8 ESTABLECER ATRIBUTO ... 386
7.8.43.9 ESTABLECER ÚLTIMA EXCEPCIÓN ... 387
7.8.44 CLASIFICAR ... 388
7.8.44.1 Ordenación basada en archivos ... 388
7.8.44.2 Tabla SORT ... 392
7.8.45 COMENZAR ... 393
7.8.46 DETENER ... 395
7.8.47 CADENA ... 397
7.8.48 SUSTRAER ... 399
7.8.48.1 RESTA DE ... 399
7.8.48.2 RESTA DAR ... 400
7.8.48.3 RESTA CORRESPONDIENTE ... 401
7.8.49 REPRIMIR ... 402
7.8.50 TERMINAR ... 403
7.8.51 TRANSFORMAR ... 404
7.8.52 DESCUBRIR ... 405
7.8.53 DESCONECTAR ... 406
7.8.54 ESCRIBIR ... 410
7.8.55 GENERAR XML ... 413
7.8.56 ANÁLISIS XML ... 424

8 Funciones ... 425

8.1 Funciones intrínsecas ... 425
8.1.1 abdominales ... 429
8.1.2 ACOS ... 430
8.1.3 ANUALIDAD ... 431
8.1.4 SAL ... 433
8.1.5 ATAN ... 434
8.1.6 UN POCO DE ... 435
8.1.7 DE BIT A CHAR ... 436
8.1.8 LONGITUD DE BYTE ... 437
8.1.9 CARBONIZARSE ... 438
8.1.10 FECHA Y HORA COMBINADAS ... 439
8.1.11 CONCAT ... 440
8.1.12 CONCATENAR ... 441
8.1.13 CONTENIDO-DURACIÓN ... 442
8.1.14 CONTENIDO-DE ... 443
8.1.15 PORQUE ... 444
8.1.16 MONEDA-SÍMBOLO ... 445

8.1.17 FECHA ACTUAL 446
8.1.18 FECHA-DE-ENTERO 447
8.1.19 FECHA-A-AAAAMMDD 448
8.1.20 DÍA-DE-ENTERO 449
8.1.21 DÍA-A-AAAADDD 450
8.1.22 Y 451
8.1.23 ARCHIVO DE EXCEPCIÓN 452
8.1.24 EXCEPCIÓN-UBICACIÓN 453
8.1.25 DECLARACIÓN DE EXCEPCIÓN 454
8.1.26 ESTADO DE EXCEPCIÓN 455
8.1.27 EXP 457
8.1.28 EXP10 458
8.1.29 FACTORIAL 459
8.1.30 FECHA ACTUAL FORMATADA 460
8.1.31 FECHA FORMATADA 461
8.1.32 FECHA Y HORA FORMATADA 462
8.1.33 TIEMPO FORMATADO 463
8.1.34 FRACCIÓN-PARTE 464
8.1.35 HEXAGONAL DE 465
8.1.36 HEXAGONAL A CHAR 466
8.1.37 ALGEBRAICO MÁS ALTO 467
8.1.38 ENTERO 468
8.1.39 ENTERO DE FECHA 469
8.1.40 ENTERO DEL DÍA 470
8.1.41 ENTERO DE FECHA FORMATADA 471
8.1.42 PARTE ENTERA 472
8.1.43 LONGITUD 473
8.1.44 LONGITUD-AN 474
8.1.45 COMPARACIÓN LOCAL 475
8.1.46 FECHA LOCAL 476
8.1.47 HORA LOCAL 477
8.1.48 HORA LOCAL DESDE SEGUNDOS 478
8.1.49 REGISTRO 479
8.1.50 REGISTRO10 480
8.1.51 MINÚSCULAS 481
8.1.52 ALGEBRAICO MÁS BAJO 482
8.1.53 MÁXIMO 483
8.1.54 SIGNIFICAR 484
8.1.55 MEDIANA 485
8.1.56 RANGO MEDIO 486
8.1.57 MÍNIMO 487
8.1.58 CONTRA 488
8.1.59 MÓDULO-IDENTIFICADOR-DE-LLAMADA 489
8.1.60 MODULO-FECHA 490
8.1.61 MÓDULO-FORMATIZADO-FECHA 491
8.1.62 ID DEL MÓDULO 492
8.1.63 RUTA DEL MÓDULO 493
8.1.64 MÓDULO-FUENTE 494

8.1.65 MÓDULO-TIEMPO 495
8.1.66 PUNTO DECIMAL MONETARIO 496
8.1.67 SEPARADOR DE MILES MONETARIOS 497
8.1.68 PUNTO DECIMAL NUMÉRICO 498
8.1.69 SEPARADOR DE MILLARES NUMÉRICOS 499

8.1.70 NUMERO VAL 500
8.1.71 NUMVAL-C 501
8.1.72 NUMVAL-C-2 502
8.1.73 NUMVAL-F 503
8.1.74 PALABRA 504
8.1.75 PALABRA MÁXIMA 505
8.1.76 PALABRAS-MIN 506
8.1.77 PI 507
8.1.78 VALOR ACTUAL 508
8.1.79 ALEATORIO 510
8.1.80 RANGO 512
8.1.81 movimiento rápido del ojo 513
8.1.82 CONTRARRESTAR 514
8.1.83 SEGUNDOS DESDE EL TIEMPO FORMATADO 515
8.1.84 SEGUNDOS PASADOS DE LA MEDIANOCHE 516
8.1.85 FIRMAR 517
8.1.86 PECADO 518
8.1.87 raíz cuadrada 519
8.1.88 DESVIACIÓN ESTÁNDAR 520
8.1.89 LONGITUD DE CARACTERES ALMACENADOS 521
8.1.90 SUSTITUTO 522
8.1.91 CASO SUSTITUTO 523
8.1.92 SUMA 524
8.1.93 BRONCEARSE 525
8.1.94 FECHA DE EXAMEN AAAAMMDD 526
8.1.95 DÍA DE EXAMEN AAAADDD 527
8.1.96 FECHA Y HORA CON FORMATO DE PRUEBA 528
8.1.97 PRUEBA-NUMVAL 529
8.1.98 PRUEBA-NUMVAL-C 530
8.1.99 PRUEBA-NUMVAL-F 531
8.1.100 RECORTAR 532
8.1.101 MAYÚSCULAS 533
8.1.102 DIFERENCIA 534
8.1.103 CUANDO SE COMPILA 535
8.1.104 AÑO-A-AAAA 536
8.1.105 BOOLEANO DE ENTERO 537
8.1.106 CHAR-NACIONAL 538
8.1.107 VISUALIZACIÓN DE 539
8.1.108 ARCHIVO DE EXCEPCIÓN-N 540
8.1.109 EXCEPCION-UBICACION-N 541
8.1.110 ENTERO DE BOOLEANO 542
8.1.111 NACIONAL-DE 543
8.1.112 ESTÁNDAR-COMPARAR 544
8.2 Subrutinas del sistema integradas 546
8.2.1 C$LLAMADO POR 548
8.2.2 C$CHDIR 549
8.2.3 C$COPIA 550
8.2.4 C$BORRAR 551
8.2.5 C$INFORMACIÓN DEL ARCHIVO 552
8.2.6 C$GETPID 553
8.2.7 C$JUSTIFICAR 554
8.2.8 C$HECHO 555
8.2.9 C$NARG 556

8.2.10 C$PARAMSIZE 557
8.2.11 C$IMPRIMIBLE 558
8.2.12 C$dormir 559
8.2.13 C$TOBAJADO 561
8.2.14 C$TOUPPER 562
8.2.15 SONIDO DE ALARMA CBL 563
8.2.16 CBL Y 564
8.2.17 SONIDO DE CAMPANA CBL 565
8.2.18 CBL CAMBIAR DIR 566
8.2.19 ARCHIVO DE COMPROBACIÓN CBL EXISTENTE 567
8.2.20 CBL CERRAR ARCHIVO 568
8.2.21 ARCHIVO DE COPIA CBL 569
8.2.22 CBL CREAR DIRECTORIO 570
8.2.23 CBL CREAR ARCHIVO 571
8.2.24 CBL ELIMINAR DIRECTORIO 572
8.2.25 CBL ELIMINAR ARCHIVO 573
8.2.26 Ecualizador CBL 574
8.2.27 PROCESO DE ERROR CBL 575
8.2.28 PROCEDIMIENTO DE SALIDA DE CBL 577
8.2.29 ARCHIVO DE LAVADO CBL 579
8.2.30 Horquilla CBL GC 580
8.2.31 CBL GC GETOPT 582
8.2.32 CBL GC ORGANIZADO 584
8.2.33 CBL GC NANOSUÉTER 587
8.2.34 CBL GC IMPRIMIBLE 588
8.2.35 DESCARGA DE SCR DE GC DE CBL 589
8.2.36 RESTAURACIÓN DEL SCR DEL GC DEL CBL 591
8.2.37 CONJUNTO SCR CBL GC TAMAÑO 592
8.2.38 IDENTIFICACIÓN DE ESPERA DE GC DE CBL 593
8.2.39 CBL OBTENGA CSR POS 594
8.2.40 CBL OBTENER DIRECTORIO ACTUAL 595
8.2.41 CBL OBTENER TAMAÑO SCR 596
8.2.42 IMP CBL 597
8.2.43 CBL NIMP 598
8.2.44 CBL NOR 599
8.2.45 CBL NO 600
8.2.46 CBL OC GETOPT 601
8.2.47 CBL OC ALOJADO 602
8.2.48 CBL OC NANOSLEEP 603
8.2.49 ARCHIVO ABIERTO CBL 604
8.2.50 CBL O 605
8.2.51 CBL LEER ARCHIVO 606
8.2.52 CBL LEER CARACTER KBD 607
8.2.53 ARCHIVO DE RENOMBRAMIENTO DE CBL 608
8.2.54 ERROR DE TIEMPO DE EJECUCIÓN DE CBL 609
8.2.55 CBL SET CSR POS 610
8.2.56 CBL REBAJADOR 611
8.2.57 CBL TOPPERS 612

8.2.58 ARCHIVO DE ESCRITURA CBL 613
8.2.59 Cable XOR 614
8.2.60 Extfh 615
8.2.61 SISTEMA 644
8.2.62 X"91" 645

8.2.63 X"E4" ... 647
8.2.64 X"E5" ... 647
8.2.65 X"F4" ... 648
8.2.66 X"F5" ... 649

9 Uso del generador de informes ... 651

9.1 Léxico RWCS ... 651
9.2 La anatomía de un informe ... 652
9.3 La anatomía de una página de informe ... 652
9.4 Cómo RWCS crea páginas de informes ... 653
9.5 Jerarquía de control ... 654
9.6 Un ejemplo ... 656
9.6.1 Datos ... 656
9.6.2 Programa ... 658
9.6.3 Páginas de informes generados ... 662
9.7 Jerarquía de control (revisada) ... 668
9.8 Convertir el formato de página FÍSICO en formato LÓGICO ... 670

10 Interfaz con el sistema operativo ... 673

10.1 Compilación de programas ... 673
10.1.1 cobc - El compilador GnuCOBOL ... 673
10.1.2 Opción cobc -Xref un ejemplo ... 684
10.1.3 Listado de referencias cruzadas mediante cobxref ... 686
10.1.4 Variables de entorno de tiempo de compilación ... 687
10.1.5 Variables de compilación predefinidas ... 688
10.1.6 Localización de cuadernos ... 690
10.1.7 Archivos de configuración del compilador ... 691
10.2 Programas en ejecución ... 698
10.2.1 Ejecución directa ... 698
10.2.2 Ejecución de bibliotecas cargables dinámicamente ... 698
10.2.2.1 cobcrun - Ejecución desde la línea de comandos ... 698
10.2.2.2 Subprogramas cargados dinámicamente ... 699
10.2.3 Variables de entorno de tiempo de ejecución ... 700
10.2.3.1 Instrucciones generales ... 702
10.2.3.2 Entorno general ... 703
10.2.3.3 Entorno de llamada ... 705
10.2.3.4 E/S de archivo ... 706
10.2.3.5 Pantalla de E/S ... 711
10.2.3.6 Informe de E/S ... 713
10.2.3.7 Variables de entorno de E/S de archivo y/o archivo de diccionario ... 714
10.2.4 Argumentos del programa ... 715
10.3 Truncamiento binario ... 715

11 Subprogramación ... 719

11.1 Tipos de subprogramas ... 719
11.2 Subprogramas independientes, contenidos y anidados ... 719
11.3 Puntos de entrada alternativos ... 720
11.4 Subprogramas dinámicos y estáticos ... 721
11.5 Flujo de ejecución de subprogramas ... 722
11.5.1 Flujo de ejecución de subrutinas ... 722
11.5.2 Flujo de ejecución de funciones definidas por el usuario ... 723
11.6 Compartir datos entre programas que llaman y programas llamados ... 725

11.6.1 Argumentos del subprograma ... 725
11.6.1.1 Consideraciones sobre el programa de llamadas ... 725
11.6.1.2 Consideraciones del programa llamado ... 726
11.6.2 Elementos de datos GLOBALES ... 726
11.6.3 Elementos de datos externos ... 727
11.7 Subprogramas recursivos ... 728
11.8 Combinando programas GnuCOBOL y C ... 730
11.8.1 Requisitos de la biblioteca de tiempo de ejecución de GnuCOBOL ... 730
11.8.2 Diferencias en la asignación de cadenas entre GnuCOBOL y C ... 730
11.8.3 Correspondencia de tipos de datos C con GnuCOBOL USAGE ... 731
11.8.4 Programas principales de GnuCOBOL que llaman a subprogramas de C ... 732
11.8.5 Programas principales de C que llaman a subprogramas de GnuCOBOL ... 733

12 Sugerencias de estilos de programación ... 737

12.1 Marcar cambios en los programas ... 737
12.2 Convenciones de codificación y nomenclatura de elementos de datos ... 738
12.3 Subíndices de tabla versus indexación de tabla ... 739
12.4 Convenciones de nomenclatura y uso de los cuadernos ... 741
12.5 DIVISIÓN DE PROCEDIMIENTOS Secciones versus párrafos ... 741
12.6 Calcular versus sumar-restar-multiplicar-dividir ... 742

13 Programación para XFD ... 745

13.1 GnuCobol utiliza SQL para archivos ... 745

Apéndice A Glosario de términos ... 755

Apéndice B ReservadoLista de palabras ... 763

Apéndice C Listas de palabras agrupadas por característica y función ... 765

C.1 Palabras reservadas comunes ... 765
C.2 Palabras extra (obsoletas) sensibles al contexto ... 784
C.3 Registros internos ... 784

Apéndice D Lista de funciones intrínsecas ... 787

Apéndice E Rutinas del sistema ... 791

Apéndice F Nombres de sistemas ... 793

F.1 Nombres del sistema: dispositivo ... 793
F.2 Nombres del sistema: característica ... 793
F.3 Nombres del sistema: switch ... 793

Apéndice G Excepciones ... 795

Apéndice H Licencia de Documentación Libre de GNU ... 799

Apéndice I Resumen de los cambios en el documento ... 805

Índice **809**

Guía del programador de GnuCOBOL

Este manual documenta la compilación de GnuCOBOL 3.2 - Final del 5 de febrero de 2025 a las 00:20 GMT.

1 Introducción

Este documento describe la sintaxis, la semántica y el uso del lenguaje de programación COBOL tal como lo implementa GnuCOBOL.

Los principales desarrolladores originales de GnuCOBOL fueron Keisuke Nishida y Roger While. Desde entonces, muchos miembros de la comunidad han participado en su desarrollo.

Este documento está pensado para servir como una referencia completamente funcional y una guía de usuario ligera, adecuada tanto para aquellos lectores que están aprendiendo COBOL por primera vez como una herramienta de entrenamiento ligera, como para aquellos que ya están familiarizados con otro dialecto de COBOL.

De esta guía se ha extraído un manual independiente que contiene únicamente los detalles básicos de la implementación de GnuCOBOL y está diseñado para programadores COBOL con experiencia. Ese documento (Referencia rápida de GnuCOBOL) no contiene material didáctico.

Otro manual que se basa en este manual (Guía del programador) es la Referencia del programador. En esta versión se eliminará todo el material de formación, por lo que será solo una referencia detallada. Hay algunos problemas que superar, ya que actualmente se utiliza el mismo texto para la Guía y será necesario cambiarlo en algunas áreas para eliminar todo el material de formación que cree nuevos documentos de texto que se utilicen como entrada para crear este manual. Este manual está destinado a programadores Cobol experimentados que requieran los detalles del lenguaje Cobol en relación con GnuCOBOL junto con detalles de programación para GnuCOBOL utilizando específicamente las diversas plataformas en las que puede ejecutarse.

Este manual se sitúa entre la Guía del programador y la (Referencia rápida de GnuCOBOL), que cubre los esqueletos del lenguaje Cobol aplicado a GnuCOBOL.

Otros documentos que conviene leer son el gnucobol.pdf que se encuentra en el directorio doc de las fuentes del compilador y el archivo NEWS que se suministra con el código fuente del compilador GnuCOBOL, en el directorio de nivel superior. Allí encontrará las últimas características del lenguaje COBOL que se han añadido, algunas de las cuales pueden no estar incluidas en este documento debido a limitaciones de tiempo. Si encuentra alguna, infórmelo como un error para la Guía del programador para que pueda ser corregido.

Otro documento que profundiza en el compilador y que es de lectura obligada es la sección de preguntas frecuentes disponible a través de GnuCOBOL.Manuales y guías (https://gnucobol.sourceforge.io/# Preguntas frecuentes), aunque podría ser un poco más ordenado para facilitar la lectura y encontrar la información necesaria, pero incluye un índice.

————————————————————————

Fin del Capítulo 1 — Introducción

1.1 Fuentes de referencia adicionales

Para aquellos que deseen aprender COBOL por primera vez, Gary puede recomendar encarecidamente los siguientes recursos.

Si te gusta tener un libro en tus manos, te recomiendo encarecidamente Structured COBOL de Murach.por Mike Murach, Anne Prince y Raul Menendez (2000) - ISBN 9781890774059. Mike Murach y sus diversos colaboradores han escrito excelentes libros de texto sobre COBOL durante décadas. Es un libro excelente para quienes están familiarizados con los conceptos de programación en otros lenguajes, pero no están familiarizados con COBOL.

Otro a considerar es COBOL Unleashed de Jon Wessler, et al (con muchos otros colaboradores) de 1998 - ISBN 9780672312540 - 1.000 páginas, que quizás esté más orientado a la

mainframe pero con mucho que ver con otras plataformas, incluso GnuCOBOL, lo que Vincent encuentra muy útil a veces.

Uno más es Beginning COBOL for Programmers Paperback 20 de marzo de 2014 de Michael Coughlan - ISBN 978-1430262534 que bien podría valer la pena leer. Hay muchos otros.

¿Prefieres un tutorial basado en la web? Prueba la Universidad de Limerick (Irlanda) - COBOL sitio web (http://www.csis.ul.ie/cobol/).

Además está la Preguntas frecuentes sobre GnuCOBOL (https://gnucobol.sourceforge.io/faq/ índice.html)—que ya ha superado las 1.400 páginas—disponible en formato HTML o descargable.
Archivo .pdf.

Junto con cada versión de las fuentes del compilador se incluye el archivo NEWS. Contiene actualizaciones actualizadas sobre el compilador y elementos adicionales del lenguaje COBOL que posiblemente aún no estén incluidos en este manual.

1.2 Introducción a COBOL

Si ya conoces un lenguaje de programación distinto a COBOL, lo más probable es que ese lenguaje sea Java, C o C++. Te darás cuenta de que COBOL es muy diferente a ellos; a veces las diferencias son buenas y otras no. Lo que debes recordar sobre COBOL es lo siguiente: fue diseñado para resolver problemas empresariales.

COBOL, que se presentó por primera vez al público en programación en 1959, fue el primer lenguaje de programación que se estandarizó (en 1960). Esto significaba que un programa COBOL que cumpliera con los estándares escrito en la computadora "A" fabricada por la empresa "B" podría compilarse y ejecutarse en la computadora "X" fabricada por la empresa "Y" con muy pocos cambios, si es que se hacía alguno. Esto puede no parecer algo tan importante hoy en día, pero fue un cambio radical con respecto a todos los lenguajes de programación que lo precedieron e incluso a muchos de los que lo siguieron.

El nombre COBOL lo dice todo: COBOL es un acrónimo que significa "(CO)mmon (B)usiness (O)riented (L)anguage" (lenguaje orientado a los negocios). Observe que la palabra "común" aparece antes que todas las demás. La palabra "negocio" aparece en segundo lugar. Ahí radica la clave del éxito de COBOL.

1.2.1 ¿Por qué deberías aprender COBOL?

A pesar de las declaraciones de los expertos del sector, el lenguaje de programación COBOL no está muerto, aunque los lenguajes más nuevos y denominados "modernos" como Java, C#, .NET, Ruby on Rails, etc., parecen haberse convertido en los lenguajes de elección en el mundo de la tecnología de la información. Estos lenguajes se han vuelto populares porque abordan los siguientes requisitos deseados para la programación "moderna":

1. Se ajustan a los principios de la programación orientada a objetos (POO). Esto es deseable por una razón importante: facilita la "reutilización del código", mejorando así la productividad de los programadores al permitirles reutilizar código previamente escrito (y depurado) en nuevas aplicaciones. Por una razón u otra, COBOL se percibe como débil en este sentido. No lo es (especialmente hoy), como veremos en la siguiente sección, pero la percepción es importante.
2. Estos lenguajes no se limitan a los ordenadores mainframe, como se cree que ocurre con COBOL. Algunos, como .NET y Ruby, ni siquiera están disponibles en los mainframes. Los lenguajes de programación "modernos" fueron diseñados y pensados para su uso en toda la variedad de plataformas informáticas, desde los ordenadores de bolsillo (es decir, los teléfonos inteligentes) hasta los superordenadores más gigantescos.

3. Existen varias implementaciones de COBOL excelentes disponibles comercialmente para sistemas que no son mainframe (Micro Focus Visual COBOL, AccuCOBOL, NetCOBOL y Elastic COBOL, por nombrar solo algunas), incluidos los sistemas Windows y UNIX/Linux. Sin embargo, no son baratas.

4. Las universidades adoran los lenguajes "modernos". En Estados Unidos, el 73% de las universidades no tienen ni un solo curso de COBOL en sus planes de estudio. Al parecer, COBOL ya no es lo suficientemente "cool" como para que los estudiantes llenen un aula.

El hecho de que COBOL tradicionalmente no admita objetos, clases y similares no significa que su enfoque "procedimental" de la informática no sea valioso; después de todo, ejecuta el 70 % de las transacciones comerciales del mundo y lo hace utilizando más de 200 mil millones de líneas de código (según las investigaciones):

- Utilizando programas que, en su mayor parte, son mucho más autodocumentados de lo que sería el caso con cualquier otro lenguaje de programación.
- Proporciona una precisión aritmética de 31 dígitos sin esfuerzo, con un rendimiento que se acerca al de los programas en lenguaje ensamblador bien escritos. No crea que esto no es de vital importancia para los bancos, las empresas de inversión y cualquier empresa interesada en el seguimiento de los ingresos, los gastos y las ganancias (obvio, como TODOS).
- Se integra bien con infraestructuras que no son COBOL, como XML, SOA, MQ, casi cualquier DBMS, plataformas de procesamiento de transacciones, instalaciones de gestión de colas y otros lenguajes de programación.
- Al poder ejecutarse en casi tantas plataformas informáticas diferentes como Java, no es posible ejecutar programas COBOL en un teléfono inteligente, pero sí en computadoras de escritorio, estaciones de trabajo, midframes/servidores, mainframes y supercomputadoras.

Los gerentes de TI y los líderes empresariales de hoy se enfrentan a un dilema desafiante: ¿cómo mantener la enorme base de código COBOL que aún hace funcionar sus negocios cuando el mundo académico prácticamente ha abandonado el lenguaje que necesitan que su gente use para que las cosas sigan funcionando? El problema se agrava por el hecho de que los programadores que son expertos en COBOL se están jubilando y se llevan consigo sus conocimientos. En algunos mercados, esto parece estar teniendo un efecto inflacionario en el costo de los recursos (programadores COBOL), cuya oferta es cada vez menor. La presión para actualizar las aplicaciones para que utilicen interfaces gráficas de usuario más actualizadas también se percibe como una razón para abandonar COBOL en favor de lenguajes compatibles con GUI como Java.

Las empresas están abordando el desafío COBOL de diferentes maneras:

1. Al emprender los denominados "proyectos de modernización", en los que las aplicaciones existentes se reescriben en lenguajes "modernos" o se reemplazan directamente con paquetes comprados. La mayoría de estas empresas utilizan estas actividades como excusa para abandonar los mainframes "costosos" en favor de soluciones de "sistemas abiertos" (presumiblemente) menos costosas (midframes/servidores).
2. Muchas veces, estas empresas descubren que el costo de la ingeniería de sistemas/redes, la gestión operativa y el monitoreo y la gestión de riesgos (es decir, la recuperación ante desastres) de las infraestructuras necesarias para soportar aplicaciones verdaderamente críticas para la misión es tan alto que la solución "menos costosa" realmente no lo es; en estos casos, el mainframe puede seguir siendo la mejor opción, dejando así a COBOL en juego y a las empresas buscando otra solución para al menos una parte de su base de aplicaciones.
3. Formación de sus propios programadores COBOL. Dado que las universidades y las escuelas técnicas han perdido interés en hacerlo, muchas empresas han asumido la tarea de "cultivar su propia" nueva generación de programadores COBOL. El miedo a quedar encasillados en una tecnología de nicho es un factor que inhibe a muchos de los programadores actuales a ofrecerse voluntariamente a dicha formación.

4. Al trasladar la interfaz de usuario al escritorio, estos esfuerzos implican ejecutar lenguaje moderno.Clientes front-end en los escritorios de los usuarios (o portátiles o teléfonos inteligentes, etc.) con programas COBOL que proporcionan funcionalidad de servidor en plataformas mainframe o midframe, proporcionando todo el "trabajo pesado" de bases de datos y archivos en el back-end. Soluciones como esta brindan a los usuarios

las interfaces de usuario que desean/necesitan mientras siguen aprovechando las fortalezas de COBOL en sistemas mainframe o midframe heredados (posiblemente) de menor tamaño.

Es probable que sea cierto que un profesional de TI ya no puede permitirse el lujo de permitir que COBOL sea la única herramienta en su caja de herramientas, pero con una base de código masiva aún en producción ahora y en el futuro previsible, agregar COBOL a un currículum vitae (CV) y/o resumen multilingüe (sí
— son diferentes) no es algo malo en absoluto. Conocer COBOL tan bien como el lenguaje del momento te convertirá en la persona más inteligente en la sala cuando se discuta la migración deSe pasa del entorno "heredado" actual a una implementación "moderna".

Descubrirás que COBOL es un lenguaje fácil de aprender y MUCHO MÁS FÁCIL de dominar que muchos de los lenguajes "modernos".

La razón principal por la que estás leyendo esto es que has descubierto GnuCOBOL, otra implementación de COBOL además de las mencionadas anteriormente. La característica distintiva de GnuCOBOL frente a las demás es que GnuCOBOL es de código abierto GRATUITO y, por lo tanto, su obtención y uso son GRATUITOS. Está mejorado y respaldado por la comunidad. Más adelante en este documento (verSección 1.3 [Entonces, ¿qué es GnuCOBOL?], página 7), comenzará a aprender más sobre las capacidades de esta implementación de COBOL.

1.2.2 Productividad del programador

A lo largo de la historia de la programación informática, la búsqueda de nuevas formas de mejorar la productividad de los programadores ha sido una consideración importante. Aparte de los aficionados, la programación es una actividad que se realiza por dinero y las empresas detestan gastar más de lo absolutamente necesario; incluso las agencias gubernamentales intentan gastar en proyectos la menor cantidad de dinero que sea absolutamente necesaria.

La cantidad de programación necesaria para llevar a cabo una tarea determinada, incluida la repetición del trabajo que se debe hacer para solucionar los errores detectados durante las pruebas (las pruebas se definen a veces en broma como el tiempo durante el cual una aplicación está realmente en producción, lo que permite a los usuarios descubrir los problemas), es la medida de la productividad del programador. Todo lo que reduzca ese esfuerzo reducirá, por tanto, el tiempo dedicado a dichas actividades, lo que reducirá el coste de las mismas. Cuando se reduce el coste de la programación, aumenta la productividad del programador.

En ocasiones, la búsqueda de una mayor productividad de los programadores (y, por lo tanto, de una reducción de los gastos de programación) ha adoptado la forma de introducir nuevas características en los lenguajes de programación, o incluso nuevos lenguajes en su conjunto. En ocasiones, ha dado lugar a nuevas formas de utilizar los lenguajes existentes.

Si bien muchos avances tecnológicos y de procedimiento han logrado mejoras evolutivas en la productividad del programador, cada uno de los tres eventos siguientes ha sido responsable de mejoras revolucionarias:

- El desarrollo de los llamados lenguajes de programación de "alto nivel", que permiten a un programador especificar en una sola declaración del lenguaje una acción que habría requerido muchas más declaraciones separadas en un lenguaje de programación anterior. La estandarización de dichos lenguajes, haciéndolos utilizables en una amplia variedad de computadoras y sistemas operativos, fue un aspecto clave de este desarrollo. COBOL fue un desarrollo pionero en esta área, siendo un descendiente directo del primer lenguaje de alto nivel (FLOW-MATIC, desarrollado por la teniente naval estadounidense Grace Hopper) y el primero en ser estandarizado.
- El establecimiento de técnicas de programación que facilitan la lectura de los programas y, por lo tanto, su comprensión. Estas técnicas no sólo reducen la cantidad de trabajo de

repaso necesario simplemente para que un programa funcione según lo previsto, sino que también reducen la cantidad de tiempo que un programador necesita para estudiar un programa existente con el fin de saber cómo adaptarlo mejor a los requisitos cambiantes del negocio. El principal desarrollo en esta área fue la programación estructurada.

Este enfoque de programación, introducido a finales de los años 70, dio lugar a nuevos lenguajes de programación (PASCAL, ALGOL, PL/1, etc.) diseñados en torno a él. Con el estándar ANSI 85, COBOL adoptó los principios propugnados por los expertos en programación estructurada, así como todos los lenguajes diseñados estrictamente en torno a él.

- El establecimiento de técnicas de programación Y la introducción de capacidades de lenguaje de programación para facilitar la reutilización del código del programa. Cualquier cosa que apoye la reutilización del código puede tener un profundo impacto en la cantidad de tiempo que lleva desarrollar nuevas aplicaciones o hacer cambios significativos en las existentes. En los últimos años, la programación orientada a objetos (POO) ha sido el "modelo" de la industria de la reutilización del código. Al permitir que la lógica del programa y las estructuras de datos que manipula la lógica se encapsulen en módulos fácilmente almacenables y recuperables (y por lo tanto "reutilizables") llamados clases, los lenguajes orientados a objetos como Java, C++ y C# se han convertido en los favoritos del mundo académico. Dado que los estudiantes se están formando en estos lenguajes y solo en ellos, en general, no es de extrañar que
—Hoy en día— los lenguajes de programación orientados a objetos son los favoritos de la industria.

La realidad, sin embargo, es que los buenos programadores han estado practicando la reutilización de código durante más de medio siglo. Hasta hace poco, los programadores COBOL contaban con algunas de las mejores herramientas de reutilización de código disponibles: lo hacían con cuadernos de copia y subprogramas en lugar de clases, métodos y atributos, pero los resultados netos han sido similares. Con los estándares COBOL2002 y COBOL 2014, el lenguaje de programación COBOL se ha vuelto tan "orientado a objetos" como los lenguajes "modernos", al tiempo que conserva la capacidad de soportar, modificar, compilar y ejecutar programas COBOL "heredados".

Si bien GnuCOBOL admite algunas de las construcciones de programación orientada a objetos definidas por los estándares COBOL2002 y COBOL2014, admite todos los aspectos del estándar ANSI 85 y, por lo tanto, cumple completamente con las necesidades de los puntos 1 y 2 anteriores. Con su conjunto de características admitidas (consulteSección 1.3 [Entonces, ¿qué es GnuCOBOL?], página 7), proporciona importantes capacidades de productividad para el programador.

1.3 Entonces, ¿qué es GnuCOBOL?

GnuCOBOL es un compilador y entorno de ejecución COBOL de código abierto y gratuito, escrito con el lenguaje de programación C, que es de uso gratuito y se puede utilizar en todas las formas, incluso con fines comerciales; no hay restricciones. GnuCOBOL se distribuye normalmente en forma de código fuente y luego debe compilarse para el sistema operativo de su computadora utilizando el compilador y cargador de C del sistema. Si bien originalmente se desarrolló para los sistemas operativos UNIX y Linux, GnuCOBOL también se ha compilado con éxito para computadoras que ejecutan OSX (una implementación de OpenBSD) y Windows utilizando las características de emulación de UNIX de herramientas como Cygwin y MinGW. Consulte tambiénSitio web de GNU (https://savannah.gnu.org/projects/gnucobol) Para obtener más información, también se pueden encontrar implementaciones predefinidas para Windows enhttps://www. http://arnoldtrembley.com/GnuCOBOL.htm.

El enfoque MinGW es uno de los favoritos del autor de este manual porque crea un compilador GnuCOBOL y una biblioteca de ejecución que requieren que solo haya una única DLL MinGW disponible para el compilador GnuCOBOL, la biblioteca de ejecución y los programas de usuario. Esa DLL se puede distribuir libremente según los términos de la

Licencia Pública General de GNU. Una compilación MinGW de GnuCOBOL cabe fácilmente en una unidad flash de 128 MB y se ejecuta desde ella sin necesidad de instalar nada.software en la computadora Windows que lo usará. Sin embargo, se sacrifican algunas funciones del lenguaje, como compartir archivos entre programas GnuCOBOL que se ejecutan simultáneamente y bloquear registros en ciertos tipos de archivos, ya que las rutinas del sistema operativo subyacentes necesarias para implementarlas no están disponibles para Windows y MinGW no las proporciona. La versión actual de MinGW está disponible en el enlace de descarga junto con varias

Otras plataformas en GnuCOBOLhttps://sourceforge.net/projects/gnucobol/files/ gnu-cobol, sitio web de descarga.

GnuCOBOL también se ha creado como una aplicación verdaderamente nativa de Windows que utiliza el paquete Visual Studio Community de Microsoft, que se puede descargar gratuitamente, para proporcionar el compilador y el enlazador/cargador de C. Este enfoque no se presta bien a una distribución "portátil".

El compilador GnuCOBOL genera código C a partir de sus programas fuente COBOL; ese código C luego se compila y vincula automáticamente utilizando el compilador C de su sistema (normalmente, pero no limitado a, gcc).

GnuCOBOL es totalmente compatible con gran parte del estándar ANSI 85 para COBOL (la única exclusión importante es el Módulo de Comunicaciones) y también es compatible con algunos de los componentes de los estándares COBOL2002 y COBOL2014, como la SECCIÓN DE PANTALLA (verSección 6.7 [SECCIÓN DE PANTALLA], página 142), SORT basado en tablas (verSección 7.8.44.2 [Tabla SORT], página 392) y funciones definidas por el usuario. Hay otras y se agregan más casi todas las semanas.

2 Fundamentos de COBOL

Este capítulo describe la sintaxis, la semántica y el uso del lenguaje de programación COBOL tal como se implementa en la versión actual de GnuCOBOL. En el resto de este documento, el lenguaje se escribe como COBOL para facilitar la lectura; sin embargo, el nombre del compilador conserva la combinación de mayúsculas y minúsculas de GnuCOBOL.

Este documento pretende servir como una referencia con todas las funciones y una guía de usuario adecuada tanto para aquellos lectores que están aprendiendo COBOL por primera vez como herramienta de capacitación, como para aquellos que ya están familiarizados con algunos dialectos del lenguaje COBOL.

Existe un manual independiente que contiene únicamente los detalles de la gramática COBOL tal como se implementa en GnuCOBOL, que está diseñado estrictamente para programadores COBOL experimentados y que se extrae de esta guía. NO contiene ningún material de capacitación.

Estos manuales adicionales son: Referencia rápida de GnuCOBOL que contiene sólo la semántica/gramática de COBOL en un documento breve, mientras que el otro, Programas de muestra de GnuCOBOL, muestra ejemplos detallados de programas COBOL con indicación de la sintaxis utilizada en cada programa.

Para cada implementación del compilador GnuCOBOL, también se deben leer los archivos NEWS suministrados para cualquier actualización de último momento junto con los archivos README e INSTALL para construir el compilador.

2.1 El lenguaje COBOL: conceptos básicos

2.1.1 Palabras reservadas del idioma

Los programas COBOL consisten en una secuencia de palabras y símbolos. Las palabras, que consisten en secuencias de letras (mayúsculas y/o minúsculas), dígitos, guiones ('-') y/o guiones bajos ('_'), pueden tener un significado específico predefinido para el compilador o pueden ser inventadas por el programador para sus propios fines.

La especificación del lenguaje GnuCOBOL define más de 1130 palabras reservadas, palabras a las que el compilador les asigna un significado especial. Esta lista y número se aplican a la lista predeterminada que cubre muchas implementaciones. Es posible limitar la lista a una implementación específica mediante -std=xyz[-strict] o anular manualmente la reserva de palabras si se utilizan en fuentes existentes como palabras definidas por el usuario.

Los programadores pueden usar una palabra reservada como parte de una palabra que están creando ellos mismos, pero no pueden crear su propia palabra como un duplicado exacto (sin tener en cuenta las mayúsculas y minúsculas) de una palabra reservada de COBOL. Tenga en cuenta que una palabra reservada incluye todas las clases, como funciones intrínsecas, nombres mnemotécnicos y rutinas del sistema. La lista de palabras reservadas se puede cambiar agregando o eliminando palabras específicas para una compilación determinada o, de manera predeterminada, mediante el uso del comando de dirección -std= (di- alect) y –conf= (archivo de configuración del usuario). Vea los archivos de configuración específicos que se guardan de manera predeterminada en
/usr/local/share/gnucobol/config. Además, si se utiliza la opción 'FUNCIÓN TOTALMENTE INTRÍNSICA', se agregarán más de 100 palabras reservadas. Estas se pueden modificar para que coincidan con los requisitos de una empresa o un equipo de proyecto, pero tenga en cuenta que se actualizan cuando se crea una nueva versión del compilador, por lo que puede ser más prudente crear su propia configuración basándose en una existente, pero con un nombre diferente.

Además, puede agregar y/o eliminar palabras reservadas agregando una de estas opciones a

cobc para agregar -freserved=word o, para eliminar, -fnot-reserved=word.
-freserved=palabra:alias para crear un alias para una palabra, así como -fnot-register=palabra o -fregister=palabra para eliminar o agregar, una palabra de registro especial.

Consulte el Apéndice B para obtener una lista completa de palabras reservadas de GnuCOBOL y los Apéndices C a F (para listas de palabras agrupadas).

Para cualquier versión de GnuCOBOL, también puede listar el conjunto completo actual de palabras reservadas ejecutando cobc con --list-reserved, --list-intrinsic, --list-system y también con los mnemónicos --list-. Nuevamente, esto está sujeto a variaciones según el uso del comando de línea --std.

2.1.2 Palabras definidas por el usuario

Cuando escriba programas GnuCOBOL, deberá crear una variedad de palabras para representar varios aspectos del programa, los datos del programa y el entorno externo en el que se ejecutará el programa. Esto incluirá nombres internos por los que se hará referencia a los archivos de datos, nombres de elementos de datos y nombres de procedimientos lógicos ejecutables como nombres de secciones y párrafos.

Las palabras definidas por el usuario pueden estar compuestas por los caracteres de la 'A' a la 'Z' (mayúsculas y/o minúsculas), del '0' al '9', guiones ('-') y guiones bajos ('_'). Las palabras definidas por el usuario no pueden comenzar ni terminar con guiones o guiones bajos.

Otros lenguajes de programación proporcionan al programador una capacidad similar para crear sus propias palabras (nombres) para partes de un programa; COBOL es algo inusual en comparación con otros lenguajes en el sentido de que las palabras definidas por el usuario pueden comenzar con un dígito.

Con excepción de los nombres de procedimientos lógicos, que pueden consistir únicamente en dígitos, las palabras definidas por el usuario deben contener al menos una letra.

El tamaño máximo de una palabra definida por el usuario en Cobol es de 31 caracteres según el estándar COBOL 2014, pero para ayudar a que otros compiladores lo admitan, se puede ampliar con el uso de -std (COBOL85 e ibm-strict tienen 30) para aumentar el límite a 63 caracteres. Se debe señalar que exceder el límite estándar restringirá seriamente la capacidad de transferir cualquier código escrito para GnuCOBOL a otra marca de compilador sin cambiar todas esas palabras definidas por el usuario a 30 o 31. Todo el arte de escribir con Cobol consiste en minimizar la necesidad de cambiar cualquier código a lo largo de los años en que sus programas estarán en uso.

Hay muchos ejemplos de programas escritos desde la década de 1960 que todavía están en funcionamiento en todo el mundo y el número de líneas de código Cobol se estima en 200 mil millones.

Por ejemplo, este autor (Vincent Coen) tiene un código que se remonta a principios de los años 60 con ciertos cambios a lo largo de los años, y aún sigue en pleno funcionamiento; basta con echar un vistazo al área Contrib y consultar cobxref, una herramienta de listado de referencias cruzadas de Cobol (también en Sourceforge), dectrans (traductor de decisiones), flightlog, un libro de registro de pilotos (también en Sourceforge) y también en Sourceforge, ACAS (sistema de contabilidad de computadoras Applewood), este solo con el código original, que se remonta a 1967.

Por supuesto, muchas, si no la mayoría de estas aplicaciones, han sufrido muchos cambios, actualizaciones, etc. a lo largo de los años, pero esto demuestra cuánto tiempo han sobrevivido los programas escritos en Cobol y se han seguido utilizando a tiempo completo durante unos 60 años.

La cuestión es que, al escribir en Cobol, siempre debes considerar si el código será transferible a otro sistema o compilador en los próximos años, ¡pero sin pasarse de la raya!

2.1.3 Insensibilidad a mayúsculas y minúsculas

Todas las implementaciones de COBOL permiten el uso de letras mayúsculas y minúsculas en la codificación de programas. GnuCOBOL es completamente insensible a las mayúsculas y

minúsculas utilizadas al escribir palabras reservadas o nombres definidos por el usuario. Por lo tanto, AAAAA, aaaaa, Aaaaa y AaAaA son todas la misma palabra en lo que respecta a GnuCOBOL.

La única vez que importa el uso de mayúsculas y minúsculas es dentro de cadenas de caracteres entre comillas, donde los valores de los caracteres serán exactamente como están codificados.

Por convención a lo largo de este documento, las palabras reservadas de COBOL se mostrarán completamente en MAYÚSCULAS, mientras que aquellas palabras creadas por un programador se representarán mediante tokens en mayúsculas o minúsculas.

Esta no es una mala práctica para usar en programas reales, ya que conduce a programas en los que es mucho más fácil distinguir las palabras reservadas de las definidas por el usuario.

2.1.4 Legibilidad de los programas

Los críticos de COBOL se centran con frecuencia en la verbosidad del lenguaje, y a menudo citan el caso de un programa denominado "Hola mundo" como "prueba" de que programar en COBOL es mucho más tedioso que en lenguajes más "modernos". Este tedio se cita como un impacto tan significativo en la productividad del programador que, en su opinión, COBOL no puede desaparecer lo suficientemente rápido.

A continuación se muestran dos aplicaciones "Hola mundo" diferentes, una escrita en Java y la segunda en GnuCOBOL. Primero, la versión en Java:

```
Clase HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola
        mundo!");
    }
}
```

Y aquí está el mismo programa, escrito en GnuCOBOL:

```
DIVISIÓN DE IDENTIFICACIÓN.
PROGRAMA-ID. HolaMundo.
DIVISIÓN DE PROCEDIMIENTO.
    MOSTRAR "¡Hola mundo!".
```

Ambos programas anteriores podrían haberse escrito en una sola línea, si así se hubiera deseado, y ambos lenguajes permiten que un programador utilice (o no) la sangría como crea conveniente para mejorar la legibilidad del programa. Hasta ahora parece que hay un empate.

Veamos cuánto más "prolijo" es COBOL que Java. Contemos los caracteres de los dos programas. El programa Java tiene 95 (sin contar los retornos de carro ni las sangrías). El programa COBOL tiene 89 (de nuevo, sin contar los retornos de carro ni las sangrías). Técnicamente, podrían haber sido solo 65 porque el encabezado IDENTIFICATION DIVISION. es en realidad opcional. Claramente, "Hola mundo" no parece más conciso en Java que en COBOL.

Veamos un problema diferente. Seguramente un programa que le pide al usuario que ingrese un número entero positivo, genera la suma de todos los números enteros positivos desde 1 hasta ese número y luego imprime el resultado será MUCHO más corto y MUCHO más fácil de entender cuando se codifica en Java que en COBOL, ¿cierto?

Tú puedes ser el juez. Primero, la versión de Java:

```
importar java.util.Scanner;
clase pública sumofintegers
{
    public static void main(String[] arg) {
        System.out.println("Ingrese un entero
        positivo"); Scanner scan=new
        Scanner(System.in);
        int n=scan.nextInt();
        int suma=0;
        para (int i=1;i<=n;i++) {
```

```
        suma+=i;
    }
    System.out.println("La suma es "+suma);
}
```

```
}
```

Y ahora la versión COBOL:

```
DIVISIÓN DE IDENTIFICACIÓN.
PROGRAMA-ID. SumaDeEnteros.
DIVISIÓN DE DATOS.
SECCIÓN DE TRABAJO Y
ALMACENAMIENTO. 01norte
          BINARIO-LARGO.
01 yo     BINARIO-LARGO.
01 suma VALOR BINARIO-LARGO 0.
PROCEDIMIENTO DIVISIÓN.
MOSTRAR "Ingrese un entero positivo"
ACEPTAR n
REALIZAR VARIACIÓN DE i DE 1 EN 1 HASTA QUE i > n
      AGREGAR i A la
suma FINALIZAR
MOSTRAR "La suma es " suma.
```

Mi familiaridad con COBOL puede estar prejuiciando mi opinión, pero no me parece que el código Java sea más simple que el código COBOL. En caso de que esté interesado en el recuento de caracteres, el código Java tiene 278 (sin contar los caracteres de sangría). El código COBOL es 298 (274 sin el encabezado IDENTIFICATION DIVISION.).

A pesar de lo que has visto aquí, cuanto más compleja sea la lógica de programación que se implementa, más conciso parecerá el código Java, incluso en comparación con el estándar COBOL de 2002. Sin embargo, esa concisión tiene un precio: la legibilidad del código del programa. Los programas Java (o C o C++ o C#) generalmente son inteligibles solo para programadores capacitados. Sin embargo, los programas COBOL pueden ser bastante comprensibles para los no programadores. En realidad, esto es un efecto secundario de la "verbosidad" del lenguaje, donde las declaraciones COBOL usan palabras naturales en inglés para describir sus acciones. Esta legibilidad inherente me ha resultado útil muchas veces a lo largo de mi carrera cuando he tenido que aprender procesos comerciales (o legales) oscuros leyendo el código del programa COBOL que los respalda.

Los lenguajes "modernos", como Java, también tienen su propia infraestructura de sobrecarga "estándar" que debe codificarse para escribir la lógica necesaria en el programa. Tomemos como ejemplo las sentencias public static void main(String[] arg) e import java.util.Scanner;. Los críticos tienden a olvidarse de esto cuando critican a COBOL por su "sobrecarga" estructural.

Cuando se desarrolló por primera vez, la sintaxis de fácil lectura de COBOL lo hizo profundamente diferente de todo lo que se había visto antes. Por primera vez, fue posible especificar la lógica de una manera que fuera, al menos hasta cierto punto, comprensible incluso para los no programadores. Tomemos como ejemplo el siguiente código escrito en FORTRAN, un lenguaje desarrollado solo un año antes de COBOL:

```
EXT = PRECIO * CANTIDAD
INVTOT = INVTOT + EXT
```

Con su limitación original en la longitud de los nombres de las variables (nombres de uno a seis caracteres compuestos por una letra seguida de hasta cinco letras y/o dígitos), su regla implícita era que las variablesSe creaban automáticamente como reales (coma flotante) a menos que su nombre comenzara con una letra del rango IN, y su uso de notación algebraica para expresar acciones que se estaban realizando, FORTRAN no era un lenguaje particularmente legible, incluso para los programadores. Compare esto con el código COBOL equivalente:

```
MULTIPLICA el precio POR la cantidad DANDO el
importe extendido AGREGA el importe extendido AL
```

total de la factura

Claramente, incluso un no programador podría al menos entender conceptualmente lo que estaba sucediendo. Con el tiempo, lenguajes como FORTRAN desarrollaron nombres de variables más robustos y COBOL introdujo una capacidad sintáctica más basada en fórmulas para operaciones aritméticas, pero FORTRAN nunca fue tan legible como COBOL.

Debido a su legibilidad inherente, preferiría que me asignaran una tarea para hacercambios significativos en un programa COBOL sobre el que no sé nada que me pidan hacer lo mismo con un programa C, C++, C# o Java.

Aquellos que argumentan que es demasiado aburrido / derrochador / requiere mucho tiempo / insultante (elija una opción) tener que codificar un programa COBOL "desde cero" claramente ignoran los siguientes hechos:

- Muchos sistemas tienen herramientas de desarrollo de programas disponibles para facilitar la tarea de codificar programas; aquellas herramientas que se concentran en COBOL son capaces de proporcionar plantillas para gran parte del texto "general" de cualquier programa.
- Los buenos programadores han mantenido —durante décadas— sus propios programas "plantilla" esqueléticos para una variedad de tipos de programas; simplemente cargue una plantilla en un editor de texto y tendrá un buen comienzo para el programa.
- Según la leyenda, en realidad solo se ha escrito UN programa en COBOL, y todos los programas que se han "escrito" a partir de entonces fueron simplemente derivados de ese. Aunque esto está claramente pensado como una broma (probablemente) de mal gusto, se acerca mucho a la simple verdad de que muchos programadores "reutilizan" programas COBOL existentes cuando crean otros nuevos. Ciertamente, no hay nada que impida que esto suceda con programas escritos en otros lenguajes, pero parece que sucede más en los talleres COBOL. Es irónico que la "reutilización del código" sea uno de los argumentos que se utilizan para justificar la existencia de los lenguajes "modernos".

2.1.5 Divisiones Organizan Programas

Los programas COBOL están estructurados en cuatro áreas principales de codificación, cada una con su propio propósito. Estas cuatro áreas se conocen como divisiones.

Cada división puede constar de una variedad de secciones y cada sección consta de uno o más párrafos. Un párrafo consta de oraciones, cada una de las cuales consta de una o más afirmaciones.

Esta estructura jerárquica de componentes de programas estandariza la composición de todos los programas COBOL. Gran parte de este manual describe las distintas divisiones, secciones, párrafos y declaraciones que pueden formar parte de cualquier programa COBOL.

2.1.6 Cuadernos de notas

Declaración COPY (verSección 3.2 [COPIA], página 63) Un Copybook es un segmento de código de programa que puede ser utilizado por varios programas simplemente haciendo que esos programas utilicen la instrucción COPY para importar ese código. Este código puede definir archivos, estructuras de datos o código de procedimiento.

Los lenguajes de programación actuales tienen una declaración (normalmente, esta declaración se denomina "import", "include" o "#include") que realiza esta misma función. Sin embargo, lo que hace que la función de copia de libros de COBOL sea diferente de la función "include" de los lenguajes más nuevos es el hecho de que la declaración COPY puede editar el código fuente importado a medida que se copia. Esta capacidad hace que las bibliotecas de copia de libros sean extremadamente valiosas para hacer que el código sea reutilizable. Consulte también la sección 3. Comandos de la función de dirección del compilador COPY y REPLACE.

2.1.7 Datos estructurados

Un área contigua de almacenamiento dentro del espacio de memoria de un programa a la que se puede hacer referencia, por nombre, en un programa COBOL se denomina elemento de datos. Otros lenguajes de programación utilizan el término variable, propiedad o atributo para describir lo mismo.

COBOL introdujo el concepto de datos estructurados. El principio de los datos estructurados en COBOL se basa en la idea de poder agrupar elementos de datos relacionados y asignados de forma contigua en un único elemento de datos agregado, denominado Elemento de grupo. Por ejemplo, un elemento de grupo de 35 caracteres llamado "Nombre del empleado" podría constar de un "Apellido" de 20 caracteres seguido de un "Nombre" de 14 caracteres y una "Inicial del segundo nombre" de 1 carácter.

En COBOL, un elemento de datos que no está formado por otros elementos de datos se denomina elemento elemental. En el ejemplo anterior, "Apellido", "Nombre" e "Inicial del segundo nombre" son elementos elementales.

2.1.8 Archivos

Una de las fortalezas de COBOL es la amplia variedad de archivos de datos a los que puede acceder. Los programas GnuCOBOL, al igual que los creados con otras implementaciones de COBOL, necesitan que se les describa la estructura de los archivos que leerán o escribirán. La característica de más alto nivel de la estructura de un archivo se define especificando la organización del archivo, de la siguiente manera:

ORGANIZACIÓN LINEA SECUENCIAL

Se trata de archivos con la estructura interna más simple de todas. Su contenido es estructurado.Se puede considerar simplemente como una serie de registros de datos de tamaño idéntico o diferente, cada uno de ellos terminado por un carácter delimitador especial de fin de registro. Un carácter de avance de línea ASCII (hexadecimal 0A) es el carácter delimitador de fin de registro utilizado por cualquier compilación GnuCOBOL de UNIX o pseudo-UNIX (MinGW, Cygwin, OSX). Una compilación verdaderamente nativa de Windows utilizaría una secuencia de retorno de carro y avance de línea (hexadecimal 0D0A).

Los registros se deben leer o escribir en estos archivos de manera puramente secuencial. La única forma de leer (o escribir) el registro número 100 sería haber leído (o escrito) primero los registros número 1 a 99.

Cuando un programa GnuCOBOL escribe en el archivo, la secuencia delimitadora se agregará automáticamente a cada registro de datos a medida que se escribe en el archivo. Una instrucción WRITE (verSección 7.8.54 [ESCRIBIR], página 410) a este tipo de archivo se realizará como si se especificara una cláusula BEFORE ADVANCING 1 LINE en WRITE, si no se codifica ninguna cláusula ADVANCING.

Cuando se lee el archivo, el sistema de ejecución de GnuCOBOL eliminará la secuencia delimitadora final de cada registro. Los datos se rellenarán (a la derecha) con espacios si los datos recién leídos son más cortos que el área descrita para los registros de datos en el programa. Si los datos son demasiado largos, se truncarán y se perderá el exceso.

Estos archivos no deben definirse para contener campos de datos binarios exactos porque el contenido de esos campos podría tener inadvertidamente la secuencia de fin de registro como parte de sus valores; esto confundiría al sistema de tiempo de ejecución al leer el archivo e interpretaría ese valor como una secuencia de fin de registro real.

Las siguientes variables de entorno pueden tener un efecto en el comportamiento del procesamiento de LINE SEQUENTIAL, por lo que pueden requerir cambios en la configuración para:

COB_LS_FIJOC
OB_LS_NULOS
COB_LS_DIVISI

ÓN

Al utilizar formatos compatibles con Micro Focus:
COB-MF-LS-NULLS
COB_MF_LS_INSTAB
COB_MF_LS_DIVIDIR
COB_MF_LS_VALIDAR

Consulte las secciones 10.2.3.4 y 10.2.3.7 para obtener más información.

LINEA AVANZANDO

Se trata de archivos con una estructura interna similar a la de un archivo secuencial de líneas. Estos archivos se definen (sin una especificación ORGANIZATION explícita) mediante la cláusula LINE ADVANCING en su sentencia SELECT (consulteSección 5.2.1 [SELECCIONAR], página 102).

Cuando un programa GnuCOBOL escribe en este tipo de archivo, se agregará automáticamente una secuencia delimitadora de fin de registro a cada registro de datos a medida que se escribe en el archivo. Una operación de ESCRITURA en este tipo de archivo se realizará como si se hubiera especificado una cláusula AFTER ADVANCING 1 LINE en la operación de ESCRITURA, si no se codifica ninguna cláusula ADVANCING.

Al igual que los archivos secuenciales de líneas, estos archivos no se deben definir para contener campos de datos binarios exactos porque el contenido de esos campos podría tener inadvertidamente la secuencia de fin de registro como parte de sus valores; esto confundiría al sistema de tiempo de ejecución al leer el archivo, e interpretaría ese valor como una secuencia de fin de registro real.

ORGANIZACIÓN SECUENCIAL

Estos archivos también tienen una estructura interna simple. Su contenido está estructurado simplemente como una secuencia de caracteres de datos de longitud arbitraria. Esta secuencia de caracteres se tratará como una serie de registros de longitud fija simplemente dividiendo lógicamente la secuencia de caracteres en segmentos de longitud fija, cada uno tan largo como el tamaño máximo de registro definido en el programa. No hay caracteres delimitadores especiales de fin de registro en el archivo y cuando un programa GnuCOBOL escribe en el archivo, no se agrega ninguna secuencia delimitadora a los datos.

Los registros de este tipo de archivo tienen todos la misma longitud física, excepto posiblemente el último registro del archivo, que puede ser más corto que los demás. Si se definen registros lógicos de longitud variable en el programa, el espacio ocupado por cada registro físico del archivo ocupará el espacio descrito por la descripción del registro más larga del programa.

Entonces, si un archivo contiene 1275 caracteres de datos y un programa define la estructura de ese archivo como que contiene registros de 100 caracteres, entonces el contenido del archivo constará de doce (12) registros de 100 caracteres y un registro final que contiene solo 75 caracteres.

Parecería que debería ser posible localizar y procesar cualquier registro en el archivo directamente simplemente calculando su posición inicial de carácter en función del tamaño de registro definido por el programa. Sin embargo, aun así, los registros deben leerse o escribirse en estos archivos de manera puramente secuencial. La única forma de leer (o escribir) el registro número 100 sería haber leído (o escrito) primero los registros número 1 a 99.

Cuando se lee el archivo, los datos se transfieren al programa exactamente como existen en el archivo. En caso de que se lea un registro corto como último registro, ese registro se rellenará (hacia la derecha) con espacios.

Se debe tener cuidado de que los programas que lean un archivo de este tipo describan registros cuya longitud sea exactamente la misma que la utilizada por el programa que creó el archivo. Por ejemplo, a continuación se muestra el contenido de un archivo SECUENCIAL creado por un programa que escribió en él cinco registros de 6 caracteres. Los valores 'A', 'B', . . . reflejan los registros que se escribieron en el archivo:

```
'¡Aaaaaa!'
```

```
'BBBBBB'
'CC'
'DDDDD
D'
```

'EEEEEE'

Ahora, supongamos que otro programa lee este archivo, pero describe registros de 10 caracteres en lugar de 6. Estos son los registros que leerá el programa:

'¡AAAAAAAAAA
AA!"BBCCYYD
D'
'DDDDEEEEEE'

Puede haber ocasiones en las que esto sea exactamente lo que estaba buscando. Sin embargo, la mayoría de las veces, este no es un comportamiento deseable. Sugerencia: use un libro de copias para describir los diseños de registros de cualquier archivo; esto garantiza que varios programas que accedan a ese archivo "verán" los mismos tamaños y diseños de registros al codificar una declaración COPY (consulteSección 3.2 [COPIA], página 63) para importar los diseños de registros en lugar de codificarlos manualmente.

Estos archivos pueden contener campos de datos binarios exactos. Debido a que no existe una secuencia de caracteres que constituya un delimitador de fin de registro, el contenido de los campos de registro es irrelevante para el proceso de lectura.

ORGANIZACIÓN RELATIVA

El contenido de estos archivos consiste en una serie de registros de datos de longitud fija precedidos por un encabezado de registro de cuatro bytes. El encabezado de registro contiene la longitud de los datos, en bytes. El recuento de bytes no incluye el encabezado de registro de cuatro bytes.

Los registros de este tipo de archivo tienen todos la misma longitud física. Si se definen registros lógicos de longitud variable en el programa, el espacio ocupado por cada registro físico en el archivo ocupará el máximo espacio posible y el campo de longitud del registro lógico contendrá la cantidad de bytes de datos del registro que están realmente en uso.

Esta organización de archivos se definió para permitir un procesamiento secuencial o aleatorio. Con un archivo RELATIVO, es posible leer o escribir el registro 100 directamente, sin tener que haber leído o escrito primero los registros 1 a 99. El sistema de ejecución de GnuCOBOL utiliza el tamaño máximo de registro definido por el programa para calcular una posición relativa de bytes en el archivo donde comienzan el encabezado y los datos del registro, y luego transfiere los datos necesarios hacia o desde el programa.

Cuando el archivo se escribe mediante un programa GnuCOBOL, no se añade ninguna secuencia delimitadora a los datos, sino que se agrega un campo de longitud de registro al comienzo de cada registro físico.

Cuando se lee el archivo, los datos se transfieren al programa exactamente como existen en el archivo.

Se debe tener cuidado de que los programas que lean un archivo de este tipo describan registros cuya longitud sea exactamente la misma que la utilizada por los programas que crearon el archivo. ¡No terminará bien si la biblioteca de tiempo de ejecución de GnuCOBOL interpreta una cadena de caracteres ASCII de cuatro bytes como la longitud de un registro cuando transfiere datos del archivo al programa!

Sugerencia: use un libro de copias para describir los diseños de registros de cualquier archivo; esto garantiza que múltiples programas que accedan a ese archivo "verán" los mismos tamaños y diseños de registros al codificar una declaración COPY (verSección 3.2 [COPIA], página 63) para importar los diseños de registros en lugar de codificarlos manualmente.

Estos archivos pueden contener campos de datos binarios exactos. El contenido de

los campos de registro es irrelevante para el proceso de lectura, ya que no existe un delimitador de fin de registro.

ORGANIZACIÓN INDEXADA

Esta es la estructura de archivos más avanzada disponible para los programas GnuCOBOL. No es posible describir la estructura física de dichos archivos porque dicha estructura variará según la función de administración avanzada de archivos que se haya incluido en el programa.

La compilación de GnuCOBOL que utilizará (Berkeley Database [BDB], VBISAM, etc.) se centrará en la estructura lógica del archivo.

Se almacenarán varias estructuras para un archivo INDEXADO. La primera será un componente de datos, que puede considerarse similar a la estructura interna de un archivo relativo. Sin embargo, no se puede acceder directamente a los registros de datos por su número de registro, como sucedería con un archivo relativo, ni se pueden procesar secuencialmente por su secuencia física en el archivo.

Las estructuras restantes serán uno o más componentes de índice. Un componente de índice es una estructura de datos que (de alguna manera) permite que el contenido de un campo, llamado clave primaria, dentro de cada registro de datos (un número de cliente, un número de empleado, un código de producto, un nombre, etc.) se convierta en un número de registro de modo que el registro de datos para cualquier valor de clave primaria dado se pueda leer, escribir y/o eliminar directamente. Además, la estructura de datos de índice se define de tal manera que permita que el archivo se procese secuencialmente, registro por registro, en secuencia ascendente de los valores de campo de clave primaria. Si esta estructura de índice existe como una estructura de árbol de búsqueda binaria (b-tree), una estructura hash elaborada o algo más es bastante irrelevante para el programador - el comportamiento de la estructura será como se acaba de describir. El mecanismo real utilizado dependerá del paquete de gestión de archivos avanzado que se incluyó en su implementación de GnuCOBOL cuando se construyó.

El sistema de ejecución no permitirá que se escriban dos registros en un archivo indexado con el mismo valor de clave principal.

Existe la posibilidad de definir un campo adicional, conocido como clave alternativa. Los campos de clave alternativa se comportan como claves primarias, lo que permite el acceso directo y secuencial a los datos de registro en función de los valores de los campos de clave alternativa, con una excepción. Esa excepción es el hecho de que se puede permitir que las claves alternativas tengan valores duplicados, según cómo se describa el campo de clave alternativa al compilador de GnuCOBOL.

Puede haber cualquier cantidad de claves alternativas, pero cada campo de clave conlleva una penalización de espacio en disco y de tiempo de ejecución. A medida que aumenta la cantidad de campos de claves alternativas, tomará cada vez más tiempo escribir o modificar registros en el archivo.

Estos archivos pueden contener campos de datos binarios exactos. El contenido de los campos de registro es irrelevante para el proceso de lectura, ya que no existe un delimitador de fin de registro.

Todos los archivos se describen inicialmente en un programa GnuCOBOL mediante una declaración SELECT (verSección 5.2.1 [SELECCIONAR], página 102). Además de definir un nombre con el que se hará referencia al archivo dentro del programa, la declaración SELECT especificará el nombre y la ruta con la que el sistema operativo conocerá el archivo junto con sus atributos de organización, bloqueo y compartición.

Una descripción del archivo en la SECCIÓN DE ARCHIVO (verSección 6.2 [SECCIÓN DE ARCHIVO], página 122) voluntaddefine la estructura de los registros dentro del archivo, incluyendo si son posibles o no registros de longitud variable y, de ser así, cuál podría ser la longitud mínima y máxima. Además, la entrada de descripción del archivo puede especificar tamaños de bloque de E/S del archivo.

2.1.9 Manejo de mesas

Otros lenguajes de programación tienen matrices; COBOL tiene tablas. Básicamente, son lo mismo. Existen dos instrucciones especiales en el lenguaje COBOL (SEARCH y SEARCH ALL) que facilitan la búsqueda de datos en una tabla.

SEARCH busca una tabla secuencialmente, deteniéndose sólo cuando se encuentra una entrada de tabla que coincide con una de cualquier cantidad de condiciones de búsqueda, o cuando se han verificado todas las entradas de tabla con los criterios de búsqueda y ninguna coincide con ninguno de esos criterios.

SEARCH ALL realiza una búsqueda extremadamente rápida en una tabla ordenada por un campo clave contenido en cada entrada de la tabla. El algoritmo utilizado para dicha búsqueda es una búsqueda binaria. El algoritmo garantiza que solo se deba verificar una pequeña cantidad de entradas en la tabla para encontrar una entrada deseada o para determinar que la entrada deseada no existe en la tabla. Cuanto más grande sea la tabla, más efectiva será esta búsqueda. Por ejemplo, una búsqueda binaria de una tabla que contiene 32,768 entradas localizará una entrada en particular o determinará que la entrada no existe al observar no más de quince (15) entradas. El algoritmo se explica en detalle en la documentación de la declaración SEARCH ALL (consulteSección 7.8.42 [BUSCAR TODO], página 377).

Finalmente, COBOL tiene la capacidad de realizar ordenaciones in situ de los datos que se encuentran en una tabla.

2.1.10 Ordenar y fusionar datos

El lenguaje COBOL incluye una potente sentencia SORT que puede ordenar grandes cantidades de datos según estructuras de claves de complejidad arbitraria. Estos datos pueden tener su origen en el programa o pueden estar contenidos en uno o más archivos externos. Los datos ordenados pueden escribirse automáticamente en uno o más archivos de salida o pueden procesarse registro por registro en la secuencia ordenada.

Una sentencia complementaria (MERGE) puede combinar el contenido de varios archivos, siempre que todos ellos estén ordenados previamente de manera similar según la misma estructura de clave. La salida resultante constará del contenido de todos los archivos de entrada, fusionados y ordenados según la(s) estructura(s) de clave común. La salida generada por una sentencia MERGE puede escribirse automáticamente en uno o más archivos de salida o puede procesarse internamente por el programa.

También existe una forma especial de la declaración SORT sólo para ordenar los datos que residen en una tabla.

Esto es particularmente útil si desea utilizar BUSCAR TODO en la tabla.

2.1.11 Manipulación de cadenas

Ha habido lenguajes de programación diseñados específicamente para el procesamiento de cadenas de texto y ha habido lenguajes de programación diseñados con el único propósito de realizar cálculos numéricos de alta potencia. La mayoría de los lenguajes de programación se encuentran en algún punto intermedio.

COBOL no es una excepción, aunque incluye algunas capacidades de manipulación de cadenas muy potentes; de hecho, GnuCOBOL tiene incluso más capacidades de manipulación de cadenas que muchas otras implementaciones de COBOL. A continuación se resumen las capacidades de procesamiento de cadenas de GnuCOBOL:

Concatenar dos o más cadenas

- Función intrínseca CONCATENAR (ver Sección 8.1.12 [CONCATENAR], página 441).
- Instrucción STRING (verSección 7.8.47 [STRING], página 397).

Conversión de una hora o fecha numérica a una cadena de caracteres formateada

- Función intrínseca LOCALE-TIME (verSección 8.1.47 [HORA LOCAL], página 477).
- FECHA LOCAL intrínseco función (ver Sección 8.1.46 [LOCAL-FECHA], página 476).

Convierte un valor binario en su carácter correspondiente en el conjunto de caracteres del programa

- Función intrínseca CHAR (verSección 8.1.9 [CHAR], página 438). Agregue 1 al argumento antes de invocar la función; la descripción de la función intrínseca CHAR

presenta una técnica que utiliza la declaración MOVE que logrará lo mismo sin la necesidad de agregar 1 al valor del argumento numérico primero.

Convertir una cadena de caracteres a minúsculas

- Función intrínseca en MINÚSCULAS (verSección 8.1.51 [MINÚSCULAS], página 481).
- Subrutina de sistema incorporada C$TOLOWER (verSección 8.2.13 [C$TOLOWER], página 561).
- Subrutina de sistema incorporada CBL_TOLOWER (verSección 8.2.56 [CBL PARA BAJAR], página 611).

Convertir una cadena de caracteres a mayúsculas

- Función intrínseca en MAYÚSCULAS (verSección 8.1.101 [MAYÚSCULAS], página 533).
- Subrutina de sistema incorporada C$TOUPPER (verSección 8.2.14 [C$TOUPPER], página 562).
- Subrutina de sistema incorporada CBL_TOUPPER (verSección 8.2.57 [CBL TOUPPER], página 612).

Convertir una cadena de caracteres en solo caracteres imprimibles

- Subrutina de sistema incorporada C$PRINTABLE (verSección 8.2.11 [C$PRINTABLE], página 558).

Convierte un carácter a su valor numérico en el conjunto de caracteres del programa

- Función intrínseca ORD (verSección 8.1.74 [ORD], página 504). Restar 1 del resultado; la descripción de la función intrínseca ORD presenta una técnica que utiliza la declaración MOVE que logrará lo mismo sin la necesidad de sumar 1 al valor del argumento numérico primero.

Contar las ocurrencias de subcadenas en una cadena más grande

- Declaración INSPECT (ver Sección 7.8.26 [INSPECCIONAR], página 344) con el Cláusula de recuento.

Decodificar una cadena numérica formateada a un valor numérico

- Función intrínseca NUMVAL (verSección 8.1.70 [NUMVAL], página 500).
- Función intrínseca de NUMVAL-C (verSección 8.1.71 [NUMVAL-C], página 501).

Determinar la longitud de una cadena o un elemento de datos capaz de almacenar cadenas

- Función intrínseca LONGITUD (verSección 8.1.43 [DURACIÓN], página 473).
- Función intrínseca BYTE-LENGTH (verSección 8.1.8 [BYTE-LENGTH], página 437).

Extraer una subcadena de una cadena en función de su posición de carácter inicial y su longitud

- Uso de un modificador de referencia en el campo de cadena - VerSección 2.1.24 [Referencia Modificadores], página 39.

Formatear un elemento numérico para la salida, incluidos separadores de miles (',' en EE. UU.), símbolos de moneda ('$' en EE. UU.), puntos decimales, símbolos de crédito/débito, caracteres de signo iniciales o finales

- Declaración MOVE (verSección 7.8.30 [MOVE], página 354) con edición de símbolos de imagen aplicada al campo receptor:

Justificación (izquierda, derecha o centrada) de un campo de cadena

- Subrutina de sistema incorporada C$JUSTIFY (ver Sección 8.2.7 [C$JUSTIFY], página 554).

Sustitución monoalfabética de uno o más caracteres en una cadena con caracteres diferentes

- Declaración INSPECT (ver Sección 7.8.26 [INSPECCIONAR], página 344) con el MUDADO.

- Sentencia TRANSFORM (verSección 7.8.51 [TRANSFORMAR], página 404).
- SUSTITUIR función intrínseca (verSección 8.1.90 [SUSTITUTO], página 522).
- Función intrínseca del CASO SUSTITUTO (verSección 8.1.91 [CASO SUSTITUTO], página 523).

Analizar una cadena, dividiéndola en subcadenas en función de una o más secuencias de caracteres delimitadores1

- Instrucción UNSTRING (verSección 7.8.53 [UNSTRING], página 406).

Eliminación de espacios iniciales o finales de una cadena

- Función intrínseca TRIM (verSección 8.1.100 [TRIM], página 532).

Sustitución de una sola subcadena por otra de la misma longitud, en función de la posición y la longitud del carácter inicial de la subcadena

- Declaración MOVE (verSección 7.8.30 [MOVE], página 354) con un modificador de referencia en el campo "recepción" (verSección 2.1.24 [Modificadores de referencia], página 39).

Sustitución de una o más subcadenas en una cadena con subcadenas de reemplazo de la misma longitud, independientemente de dónde ocurran

- Declaración INSPECT (verSección 7.8.26 [INSPECCIONAR], página 344) con un REEMPLAZO cláusula.
- SUSTITUIR función intrínseca (verSección 8.1.90 [SUSTITUTO], página 522).
- Función intrínseca del CASO SUSTITUTO (verSección 8.1.91 [CASO SUSTITUTO], página 523).

Sustitución de una o más subcadenas en una cadena con subcadenas de reemplazo de una longitud potencialmente diferente, independientemente de dónde ocurran

- SUSTITUIR función intrínseca (verSección 8.1.90 [SUSTITUTO], página 522).
- Función intrínseca del CASO SUSTITUTO (verSección 8.1.91 [CASO SUSTITUTO], página 523).

2.1.12 Funciones de formato de pantalla

El estándar COBOL2002 formaliza extensiones del lenguaje COBOL que permiten la definición y el procesamiento de pantallas basadas en texto, como es una función típica en los ordenadores mainframe y midframe, así como en muchos sistemas de punto de venta (es decir, "cajas registradoras"). GnuCOBOL implementa prácticamente todas las características de manejo de pantallas descritas por COBOL2002.

Estas funciones permiten mostrar campos en posiciones específicas de fila/columna, asignar varios colores y atributos de video a los campos de la pantalla y presionar teclas de función específicas (F1, F2,
...) para que sean detectables. Todo esto se lleva a cabo a través de los auspicios de la SECCIÓN DE PANTALLA (verSección 6.7 [SECCIÓN DE PANTALLA], página 142) y formatos especiales de la declaración ACCEPT (verSección 7.8.1 [ACEPTAR], página 263) y la declaración DISPLAY (verSección 7.8.12 [PANTALLA], página 302).

El estándar COBOL2002, y por lo tanto GnuCOBOL, sólo cubre pantallas de interfaz de usuario textual (TUI) (aquellas compuestas por caracteres ASCII presentados utilizando una variedad de recursos visuales).tributos) y no las capacidades de procesamiento y diseño de pantalla de interfaz gráfica de usuario (GUI) más avanzadas incorporadas en la mayoría de los sistemas operativos modernos. Hay paquetes basados en subrutinas disponibles que pueden realizar presentaciones GUI completas, la mayoría de los cuales pueden ser llamados por programas GnuCOBOL, con una inversión moderada de tiempo de investigación (Tcl/Tk, por ejemplo), pero actualmente ninguno está incluido con GnuCOBOL.

[1]Estos delimitadores pueden ser caracteres individuales, cadenas de varios caracteres o múltiples apariciones consecutivas de cualquiera de ellos.

2.1.12.1 Una pantalla de muestra

Una pantalla de muestra producida por un programa GnuCOBOL:

La pantalla anterior fue generada por el compilador interactivo de GnuCOBOL o GCic. Consulte la sección "GCic" en Programas de ejemplo de GnuCOBOL para obtener la fuente y la lista de referencias cruzadas de este programa.

Las pantallas se definen en la sección de pantallas de la división de datos. Una vez definidas, las pantallas se utilizan en tiempo de ejecución mediante las instrucciones ACCEPT y DISPLAY.

2.1.12.2 Paleta de colores y atributos de vídeo

GnuCOBOL admite las siguientes especificaciones de atributos visuales en la SECCIÓN DE PANTALLA (verSección 6.7 [SECCIÓN DE PANTALLA], página 142):

Color — Se pueden especificar ocho (8) colores diferentes tanto para el color de fondo (pantalla) como para el color de primer plano (texto) de cualquier posición de fila/columna en la pantalla. Los colores se especifican por número, aunque un libro de texto que se incluye con todas las distribuciones de GnuCOBOL (screenio.cpy) define nombres COB-COLOR-xxxxxx para los distintos colores, de modo que se puedan especificar como un nombre más significativo en lugar de un número. Los ocho colores, por número, con los nombres de constantes definidos en screenio.cpy, son los siguientes:

Negro	COB-COLOR-NEGRO
Azul	COB-COLOR-AZUL
Verde	COB-COLOR-VERDE
Cian	COB-COLOR-CIAN
Rojo	COB-COLOR-ROJO
Magenta	COB-COLOR-MAGENTA

Amarillo COB-COLOR-AMARILLO

Blanco COB-COLOR-BLANCO

Colores extendidos

Esta es una función de video que depende del paquete curses integrado en su versión de GnuCOBOL. Los resultados reales de los códigos de color extendidos dependen de las capacidades de la terminal y de la biblioteca screenio subyacente (y, para PDCurses, también de la configuración del parámetro COB LEGACY).

Con la librería PDcurses en un sistema operativo Windows, se pueden utilizar códigos de colores adicionales, del 8 al 15, tanto para el primer plano como para el fondo. En ese caso, en un programa GnuCOBOL se puede utilizar una paleta con todos los códigos de colores siguientes (códigos de colores clásicos del 0 al 7 y códigos de colores extendidos del 8 al 15):

```
*> Colores clásicos
78 COB-COLOR-NEGRO          VALOR  0.
78 COB-COLOR-AZUL           VALOR  1.
78 COB-COLOR-VERDE          VALOR  2.
78 COB-COLOR-CIAN           VALOR  3.
78 COB-COLOR-ROJO           VALOR  4.
78 COB-COLOR-MAGENTA        VALOR  5.
*> 78 COB-COLOR-AMARILLO   VALOR  6.
78 COB-COLOR-MARRÓN         VALOR  6.  *> Valor anterior para AMARILLO
78 COB-COLOR-BLANCO         VALOR  7.

*> Colores extendidos
78 COB-COLOR-GRIS           VALOR  8.
78 COB-COLOR-AZUL CLARO     VALOR  9.
78 COB-COLOR-VERDE CLARO    VALOR 10.
78 COB-COLOR-CIAN CLARO     VALOR 11.
78 COB-COLOR-ROJO CLARO     VALOR 12.
78 COB-COLOR-MAGENTA CLARO  VALOR 13.
78 COB-COLOR-ROSA           VALOR 13.  *> mismo código que LIGHT-
                                       MAGENTA
78 COB-COLOR-AMARILLO       VALOR 14.
78 COB-COLOR-CLARO-BLANCO   VALOR 15.
```

Antes de utilizar esta paleta con códigos de color clásicos y extendidos, es necesario configurar el parámetro "COB LEGACY = 1" utilizando uno de los siguientes modos:

1. En el archivo de configuración de tiempo de ejecución (consulte COB_LEGACY en 10.2.3.5. Pantalla E/S).
2. Establecer la variable de entorno a nivel del sistema operativo.
3. Establecer la variable de entorno en el archivo fuente COBOL de la siguiente manera:

ESTABLECER EL ENTORNO "COB LEGACY" A "1" En este último modo, la declaración GnuCobol debe realizarse al comienzo del programa y no se puede cambiar mientras el programa se esté ejecutando.

Utilizando un código de color extendido del 8 al 15 sin especificar el parámetro COB LEGACY = 1, obtendrá el color "clásico" correspondiente (un código de color del 0 al 7) y el siguiente atributo:

1. Usando FOREGROUND-COLOR y un código de color del 8 al 15: el personaje será RESALTO.

2. Usando BACKGROUND-COLOR y un código de color del 8 al 15: el personaje PARPADEARÁ.

La biblioteca Ncurses en el sistema operativo Unix no admite colores extendidos en este momento.

Resumiendo.

Sin especificar la variable de entorno COB LEGACY = 1, obtendrá lo siguiente:

1. Usando FOREGROUND-COLOR y un código de color del 8 al 15: el personaje será RESALTO (y ese es el nuevo color).

2. Usando BACKGROUND-COLOR y un código de color del 8 al 15: el personajeserá el código de color correspondiente del 0 al 7 y PARPADEARÁ. (el carácter parpadea en la pantalla)

Al especificar el parámetro COB LEGACY = 1 obtendrá lo siguiente:

3 Usando FOREGROUND-COLOR y un código de color del 8 al 15: el carácter del primer plano será el nuevo color.

4. Usando BACKGROUND-COLOR y un código de color del 8 al 15: el fondo del personaje será el nuevo color.

La biblioteca Ncurses en los sistemas operativos *nix no admite colores extendidos en este momento. [*nix incluye Linux, OSX, etc.]

Brillo del texto

Hay tres niveles de brillo posibles admitidos para el texto: luz tenue (tenue), normal y luz brillante (brillante). No todas las implementaciones de GnuCOBOL admiten los tres (algunas tratan la luz tenue como normal). El factor decisivo para determinar si se admiten dos o tres niveles depende de la versión del paquete curses que se esté utilizando. Se trata de un paquete de utilidad screen-IO que se incluye en la biblioteca de tiempo de ejecución de GnuCOBOL cuando se crea el software de GnuCOBOL.

Como regla general, las implementaciones de Windows admiten dos niveles, mientras que las de Unix admiten los tres.

Parpadeo Esta también es una función de video que depende del paquete curses integrado en su versión de GnuCOBOL. Si el parpadeo está habilitado en ese paquete, el texto que se muestra en los campos definidos en la sección de pantalla como parpadeante alternará infinitamente entre la configuración más brillante posible (resaltado) y una configuración "invisible" donde el color del texto coincide con el color de fondo del campo. Una compilación de Windows, que generalmente usa el paquete "pcurses", usará un color de fondo más brillante de lo normal para indicar "parpadeo".

Vídeo inverso

Este atributo de video simplemente intercambia los colores de primer plano y de fondo y las opciones de visualización.

Delineación de campo

Es posible, si el paquete curses que se utiliza lo permite, dibujar bordes en los bordes superior, izquierdo y/o inferior de un campo.

Entrada segura

Si se desea, los campos de pantalla utilizados como campos de entrada pueden definirse como campos "seguros", donde cada unoEl carácter ingresado (sin importar lo que se haya escrito realmente) aparecerá como un asterisco (*). El carácter real

cuya tecla se presionó aún se almacenará en el

Sin embargo, este campo no está disponible en el programa. Esto es muy útil para los campos de contraseña o número de cuenta.

Personaje de aviso

Los campos de entrada pueden tener cualquier carácter utilizado como carácter de relleno. Estos caracteres de relleno proporcionan una indicación visual del tamaño del campo de entrada y se transformarán automáticamente en espacios cuando el programa procese el campo de entrada. Si no se define ningún carácter de este tipo para un campo de entrada, se asumirá un guión bajo ('_').

A continuación se muestra un ejemplo de la paleta de colores de GnuCOBOL, que muestra todas las combinaciones posibles de los distintos atributos de vídeo. Este ejemplo se preparó en un Macintosh con OSX Mavericks (10.9). El parpadeo funciona: la captura de pantalla muestra las cosas en medio del parpadeo, cuando los colores del texto y del fondo son momentáneamente los mismos. Desafortunadamente, solo hay dos intensidades de pantalla disponibles (al igual que en Windows, la configuración de "luz baja" es la misma que la predeterminada).

La paleta de colores y las opciones de vídeo de GnuCOBOL::

Las filas de cada bloque están numeradas con el color de fondo, mientras que las columnas están numeradas con el color de primer plano.

Consulte la sección "Colores" en Programas de ejemplo de GnuCOBOL, para obtener una lista de fuentes y referencias cruzadas del programa (Colors.cbl) que produjo la pantalla anterior.

2.1.13 Características del generador de informes

GnuCOBOL incluye una implementación del Sistema de control de redacción de informes (Report Writer Control System, RWCS). El módulo de redacción de informes está totalmente implementado a partir de la versión 3.0. Se trata de una característica adicional estandarizada y opcional del lenguaje COBOL que automatiza gran parte de la mecánica involucrada en la

generación de informes impresos mediante:

1. Controlar la paginación de informes, incluyendo:

   A. La producción automática de un aviso único en la primera página del informe (encabezado del informe).
   B. La producción de cero o más líneas de encabezado en la parte superior de cada página del informe (encabezado de página).
   C. La producción de cero o más líneas de pie de página en la parte inferior de cada página del informe (pie de página).
   D. La numeración automática de páginas impresas.
   E. El formato de las líneas del informe que conforman el cuerpo principal del informe (detalle).
   F. Conciencia total de dónde el "lápiz" está a punto de "escribir" en la página actual, forzando automáticamente una expulsión a una nueva página, junto con la generación automática de un pie de página para cerrar la página anterior y/o un encabezado de página para comenzar la nueva.
   G. La producción de un aviso único al final de la última página de un informe (pie del informe).
2. Realizar acciones de informes especiales en función del hecho de que los datos que se utilizan para generar el informe se han ordenado según uno o más campos clave:
   A. Supresión automática de la presentación de uno o más campos de datos del grupo de detalles cuando los valores de los campos duplican los del grupo de detalles generado anteriormente. Los campos de este tipo se denominan campos de indicación de grupo.
   B. Provoca automáticamente que los campos de indicación de grupo de detalles suprimidos vuelvan a aparecer si se imprime un grupo de detalles en una página nueva.
   C. Reconocer cuándo se controlan los campos del informe: campos vinculados a los que se usaron como declaración SORT (consulteSección 7.8.44 [SORT], página 388) claves — han cambiado. Esto se conoce como interrupción de control. El RWCS puede realizar automáticamente las siguientes acciones de informe cuando se produce una interrupción de control:
      - Producir un pie de página, conocido como pie de control, después de las líneas de detalle que compartían el mismo valor anterior para el campo de control.
      - Producir un encabezado, conocido como encabezado de control, antes de las líneas de detalle que comparten el mismo valor nuevo para el campo de control.
3. Realice el resumen de datos de la siguiente manera:
   A. Generar automáticamente subtotales en los pies de control y/o informes, resumiendo los valores de cualquier campo en el grupo de detalle.
   B. Generación automática de totales de cruces de pies en grupos de detalles. Estos serían sumas de dos o más valores presentados en el grupo de detalles.

La SECCIÓN DEL INFORME (verSección 6.6 [SECCIÓN DEL INFORME], página 135) la documentación explora la descripción de los informes y la DIVISIÓN DE PROCEDIMIENTOS (verCapítulo 7 [PROCEDIMIENTO DIVISIÓN], página 243) capítulo documenta las distintas declaraciones de lenguaje que realmente producen informes. Antes de leerlos, puede que le resulte útil leerCapítulo 9 [Escritor de informes Uso], página 651, que se dedica a juntar las piezas para usted.

2.1.14 Inicialización de datos

Hay tres formas en las que se inicializan los datos de la división de datos.

1. Cuando se ejecuta por primera vez un programa o subprograma, gran parte de los datos en su división de datos se inicializarán de la siguiente manera:

- Los elementos de datos alfanuméricos y alfabéticos (es decir, texto) se inicializarán en ESPACIOS.
- Los elementos de datos numéricos se inicializarán con un valor de CERO.

- Elementos de datos con un VALOR explícito (verSección 6.9.61 [VALOR], página 239) La cláusula en su definición se inicializará con ese valor específico.

Cada una de las distintas secciones de la división de datos tiene sus propias reglas sobre cuándo ocurrirán las acciones descritas anteriormente; consulte la documentación de esas secciones para obtener información adicional.

Estas reglas de inicialización predeterminadas pueden variar sustancialmente de una implementación de COBOL a otra. Por ejemplo, es bastante común que el almacenamiento de división de datos se inicialice con todos los ceros binarios, excepto para aquellos elementos de datos donde hay cláusulas VALUE. Tenga cuidado al trabajar con aplicaciones desarrolladas originalmente para otra implementación de COBOL para asegurarse de que las reglas de inicialización predeterminadas de GnuCOBOL no resulten disruptivas.

2. Un programador puede utilizar la declaración INITIALIZE (verSección 7.8.24 [INICIALIZAR], página 339) para inicializar cualquier grupo o elemento de datos elemental en cualquier momento. Esta declaración proporciona muchas más opciones de inicialización que las simples reglas mencionadas anteriormente.
3. Cuando la declaración ALLOCATE (verSección 7.8.3 [ASIGNAR], página 287) se utiliza para asignar un elemento de datos o simplemente para asignar un área de almacenamiento de un tamaño especificado en ALLOCATE, esa asignación puede ocurrir con o sin inicialización, según las necesidades del programador.

2.1.15 Convenciones de diagramas de sintaxis

La sintaxis del lenguaje GnuCOBOL se describirá en diagramas de sintaxis especiales utilizando las siguientes técnicas de descripción sintáctica:

PALABRA RESERVADA OBLIGATORIA
~~~~~~~~~~~~~~~~~~~~~~~~

Las palabras reservadas del lenguaje COBOL aparecerán en MAYÚSCULAS. Cuando aparecen subrayadas, como en este caso, son palabras reservadas obligatorias.

PALABRA RESERVADA OPCIONAL

Cuando las palabras reservadas aparecen sin subrayar, como en este caso, son opcionales; dichas palabras reservadas están disponibles en la sintaxis del lenguaje únicamente para mejorar la legibilidad; su presencia o ausencia no tiene ningún efecto sobre el programa.

ABREVIATURA
~~~~

Cuando sólo se subraya una parte de una palabra reservada, indica que la palabra puede codificarse en su forma completa o puede abreviarse a la parte que está subrayada.

artículos sustituibles

Los términos genéricos que representan elementos sustituibles definidos por el usuario se mostrarán completamente en minúsculas en los diagramas de sintaxis. Cuando se haga referencia a dichos elementos en el texto, aparecerán como elementos sustituibles.

Cláusula de sintaxis compleja

Los elementos que aparecen en mayúsculas y minúsculas mixtas dentro de un diagrama sintáctico representan cláusulas complejas de otros elementos sintácticos que pueden aparecer en esa posición. Algunas sintaxis COBOL se vuelven bastante complicadas y el uso de una convención como esta reduce significativamente la complejidad de un diagrama sintáctico. Cuando se hace referencia a dichos elementos en el texto, aparecerán como cláusula sintáctica compleja.

[ ] Los metacaracteres entre corchetes en los diagramas de sintaxis documentan la sintaxis del lenguaje que es opcional. Los caracteres [] en sí no deben codificarse. Si un diagrama de sintaxis contiene 'a [b] c', los elementos de sintaxis 'a' y 'c' son obligatorios, pero el elemento 'b' es opcional.

| Los metacaracteres de barra vertical en los diagramas de sintaxis documentan opciones simples. El carácter | en sí no debe codificarse. Si un diagrama de sintaxis contiene 'a|b|c', se debe seleccionar exactamente uno de los elementos 'a', 'b' o 'c'.

{ xxxxxx }
{ aaaaaa }
{ zzzzzz }
Una lista vertical de elementos, delimitada por múltiples caracteres entre llaves, es otra forma deque significa una elección entre una serie de elementos en la que se debe seleccionar exactamente uno de ellos. Este formato se utiliza para mostrar opciones cuando una o más de las selecciones son más complejas que una sola palabra, o cuando hay demasiadas opciones para presentarlas horizontalmente con los metacaracteres '|'.

| xxxxxx |
| aaaaaa |
| zzzzzz |
Una lista vertical de elementos, delimitada por múltiples barras verticales, significa una elección entre una serie de elementos donde se pueden seleccionar una o más de las opciones.

... La secuencia de metacaracteres ... significa que el elemento de sintaxis que lo precede inmediatamente puede repetirse. La secuencia ... en sí no debe codificarse. Si un diagrama de sintaxis contiene un b... c, el elemento de sintaxis 'a' debe ir seguido de al menos un elemento 'b' (posiblemente más) y la secuencia completa debe terminar con un elemento de sintaxis 'c'.

{ } Los metacaracteres de llaves ('{' y '}') se pueden utilizar para agrupar una secuencia de elementos de sintaxis de modo que se los pueda tratar como una sola entidad. Los caracteres {} en sí no se deben codificar. Estos se utilizan normalmente en combinación con los metacaracteres '|' o '...'.

* () - + =: "'<,>./
Cualquiera de estos caracteres que aparecen en un diagrama de sintaxis debe interpretarse de manera literal y son caracteres que deben codificarse (cuando esté permitido) en la declaración cuyo formato se describe. Tenga en cuenta que un carácter "." es un carácter literal que debe codificarse en una declaración, mientras que un símbolo "..." es la secuencia de metacaracteres descrita anteriormente.

2.1.16 Formato de las líneas fuente del programa

Antes del estándar COBOL2002, las instrucciones de código fuente en los programas COBOL se estructuraban en torno a tarjetas perforadas de 80 columnas. Esto significa que cada línea de código fuente en un programa COBOL constaba de cinco "áreas" diferentes, definidas por su número de columna.

A partir del estándar COBOL2002, ahora existe un segundo modo para las declaraciones de código fuente COBOL: en este modo de operación, las declaraciones COBOL pueden tener cada una hasta 255 caracteres de longitud, sin requisitos específicos sobre qué debe aparecer en qué columnas.

Por supuesto, en consonancia con la larga tradición de COBOL de mantener la compatibilidad con estándares anteriores, los programadores (y, por supuesto, los compiladores COBOL compatibles) pueden trabajar en ambos modos. Incluso es posible cambiar de uno a otro en el mismo programa. Los términos Modo de formato fijo y Modo de formato libre se utilizan para referirse a estos dos modos de formateo del código fuente.

El compilador GnuCOBOL (cobc) admite ambos modos de formato de línea fuente y utiliza de manera predeterminada el modo de formato fijo sin ninguna otra información.

Se puede indicar al compilador que funcione en cualquier modo de cualquiera de las cuatro maneras siguientes:

1. Usando un interruptor de opción del compilador — use el interruptor -fixed para comenzar en Modo de Formato Fijo (recuerde que este es el valor predeterminado) o el interruptor -free para comenzar en Modo de Formato Libre.
2. Puede utilizar las cláusulas SOURCEFORMAT AS FIXED y SOURCEFORMAT AS FREE de la directiva CDF >>SET (consulteSección 3.6 [>>SET], página 70) dentro de su código fuente para cambiar al modo de formato fijo o libre, respectivamente.
3. Puede utilizar las cláusulas >>FORMAT IS FIXED y FORMAT IS FREE de la directiva >>DEFINE CDF (consulteSección 3.4 [>>DEFINIR], página 67) dentro de su código fuente para cambiar al modo de formato fijo o libre, respectivamente.
4. Puede utilizar la directiva >>SOURCE CDF (verSección 3.7 [>>FUENTE], página 72) para cambiar al modo de formato libre (>>EL FORMATO DE FUENTE ES LIBRE) o al modo de formato fijo (>>EL FORMATO DE FUENTE ES FIJO).

Usando los métodos 2 a 4 anteriores, puede alternar entre los dos formatos a voluntad.

Las tres últimas opciones anteriores son todas equivalentes; las tres son compatibles con GnuCOBOL, por lo que se puede mantener la compatibilidad del código fuente con una amplia variedad de otras implementaciones de COBOL. Con las tres, si el compilador está actualmente en modo de formato fijo, el >> debe comenzar en la columna 8 o más allá, siempre que ninguna parte de la directiva se extienda más allá de la columna 72. Si el compilador está actualmente en modo de formato libre, el >> puede aparecer en cualquier columna, siempre que ninguna parte de la directiva se extienda más allá de la columna 255.

Dependiendo del modo de formato de origen en el que se encuentre el compilador, deberá seguir varias reglas para el modo de formato vigente en ese momento. Estas reglas se presentan en los próximos párrafos.

La siguiente explicación presenta los distintos componentes de cada registro de línea fuente de GnuCOBOL cuando el compilador está funcionando en modo de formato fijo. Recuerde que este es el modo predeterminado para el compilador de GnuCOBOL.

1-6 Área de números de secuencia

Históricamente, en la época en que se utilizaban tarjetas perforadas para enviar el código fuente de un programa COBOL a un compilador COBOL, esta parte de una declaración COBOL estaba reservada para un número de secuencia de seis dígitos. Si bien los compiladores COBOL ignoran el contenido de esta área, existía para que un programa perforado en tarjetas de 80 caracteres pudiera, si la baraja de tarjetas se dejaba caer al suelo, pasar por una máquina clasificadora de tarjetas y restaurarse a su secuencia correcta. Por supuesto, esto no es necesario hoy en día; a decir verdad, no lo ha sido durante mucho tiempo.

VerSección 12.1 [Marcar cambios en los programas], página 737, para discutir el uso valioso que puede dársele hoy al área del número de secuencia.

7 Área indicadora

La columna 7 sirve como indicador en el que aparecerá uno de los cinco valores posibles: espacio, D (o d), - (guión), / o *. Los significados de estos caracteres son los siguientes:

espacio — No tiene ningún significado especial: este es el carácter normal que aparecerá en esta área.

D/d — La línea contiene una declaración GnuCOBOL válida que normalmente se trata como un comentario a menos que el programa se esté compilando en modo de depuración.

* — La línea es un comentario.

/ La línea es un comentario que también forzará la expulsión de una página en el listado de compilación. Si bien GnuCOBOL considerará esta línea como un comentario, no insertará ningún listado generado.

- La línea es una continuación de la línea anterior. Son necesarias únicamente cuando se divide en varias líneas un literal alfanumérico (cadena de caracteres entre comillas), una palabra reservada o una palabra definida por el usuario.

8-11 Área A

Los encabezados de DIVISIÓN, SECCIÓN y sección de párrafo deben comenzar en el Área A, al igual que los números de nivel 01, 77 en las entradas de descripción de datos y los encabezados de descripción de archivo FD y SD y SORT.

12-72 Área B

Todos los demás componentes del lenguaje de programación COBOL están codificados en estas columnas.

73-80 Área del nombre del programa

Esta es otra área obsoleta de las instrucciones COBOL. Esta parte de cada instrucción también se remonta a la época en que los programas se perforaban en tarjetas; se esperaba que el nombre del programa (o al menos los primeros 8 caracteres del mismo) se perforaran aquí para que, si una baraja de código fuente COBOL que se soltaba contenía más de un programa, esa práctica máquina clasificadora de tarjetas pudiera usarse para separar primero las tarjetas por nombre de programa y luego ordenarlas por número de secuencia. Los compiladores COBOL actuales (incluido GnuCOBOL) simplemente ignoran todo lo que esté más allá de la columna 72.

VerSección 12.1 [Marcar cambios en los programas], página 737, para discutir el uso valioso que puede dársele hoy al área del nombre del programa.

2.1.17 Estructura del programa

Sintaxis completa del programa GnuCOBOL

```
[ DIVISIÓN DE IDENTIFICACIÓN. ]
   ~~~~~~~~~~~~~~~~~~~~~~~
   ID-PROGRAMA|ID-FUNCIÓN.    nombre-1 [Opciones-del-programa].
   ~~~~~~~~~~~
~~~~~~~~~~~~[ DIVISIÓN DE
MEDIO AMBIENTE. ]
   ~~~~~~~~~~~ ~~~~~~~~
[SECCIÓN DE CONFIGURACIÓN.]
   ~~~~~~~~~~~~~ ~~~~~~~
[ORDENADOR FUENTE.          Compilación-Especificación-Computadora.          ]
   ~~~~~~~~~~~~~~~
[ COMPUTADORA-OBJETO.       Especificación de computadora de ejecución.      ]
   ~~~~~~~~~~~~~~~
[ REPOSITORIO.              Prototipo-Especificación ..............]
   ~~~~~~~~~~
[ NOMBRES ESPECIALES.       Especificación de configuración del programa.    ]
   ~~~~~~~~~~~~~
[SECCIÓN DE ENTRADA-SALIDA.]
   ~~~~~~~~~~~~ ~~~~~~~
[ CONTROL DE ARCHIVOS.      Descripción general del archivo .... ]
   ~~~~~~~~~~~~
[ CONTROL IO.               Especificación de almacenamiento en búfer de archivos  ]
   ~~~~~~~~~~~
[ DIVISIÓN DE DATOS. ]
   ~~~~~~~~~~~~~
[SECCIÓN DE ARCHIVO.        Descripción detallada del archivo .. ]
   ~~~~~~~~~~~~
[SECCIÓN DE TRABAJO Y ALMACENAMIENTO. Definición de datos permanentes  ]
   ~~~~~~~~~~~~~~~ ~~~~~~~
[SECCIÓN DE ALMACENAMIENTO LOCAL. ..................................................... Definición de datos temporales
                                                                                 ]
   ~~~~~~~~~~~~~ ~~~~~~~
[SECCIÓN DE ENLACE.         Descripción del argumento del subprograma ...]
   ~~~~~~~ ~~~~~~~~
[SECCIÓN INFORME.           Descripción del informe ..........]
   ~~~~~~ ~~~~~~~~
[SECCIÓN DE PANTALLA.       Definición del diseño de la pantalla ]
   ~~~~~~ ~~~~~~~~
   DIVISIÓN DE PROCEDIMIENTOS [ { USANDO                          Subprograma-Argumento...  } ]
   ~~~~~~~~~ ~~~~~~~~~      { ~~~~~~                                                  }
                            { ENCADENAMIENTO Argumento del programa principal      }
                              ~~~~~~~~~
                        [     DEVOLVIENDO identificador-1 ] .
[DECLARATIVAS.]               ~~~~~~~~~~
   ~~~~~~~~~~~~~
[ Rutina del controlador de eventos  ]
```

[FIN DE LOS DECLARATIVOS.]

```
    ~~~ ~~~~~~~~~~~~~~
    Lógica general del
    programa
 [ Subprograma anidado ...      ]
 [ FIN DEL PROGRAMA | FUNCIÓN nombre-1 ]
    ~~~ ~~~~~~~ ~~~~~~~~
```

———

Cada programa consta de hasta cuatro Divisiones (agrupaciones principales de secciones, párrafos y codificación descriptiva o procedimental que se relacionan con un propósito común), denominadas Identificación, Entorno, Datos y Procedimiento.

1. No todas las divisiones son necesarias en cada programa, pero deben especificarse en el orden en que se muestran cuando se utilizan.
2. Los siguientes puntos se refieren a la división de identificación
   - El encabezado DIVISIÓN DE IDENTIFICACIÓN siempre es opcional.
3. Los siguientes puntos pertenecen a la división de medio ambiente:
   - Si ambas secciones opcionales de esta división están codificadas, deben codificarse en la secuencia que se muestra.
   - Cada una de estas secciones consta de una serie de párrafos específicos (ORDENADOR-FUENTE y ORDENADOR-OBJETO, por ejemplo). Cada uno de estos párrafos cumple una función específica. Si no se requiere ningún código para la función de uno de los párrafos, se puede omitir el párrafo completo.
   - Si ninguno de los párrafos dentro de una de las secciones está codificado, se puede omitir el encabezado de sección.
   - Los párrafos dentro de cada sección solo pueden codificarse en esa sección, pero pueden codificarse en cualquier orden.
   - Si ninguna de las secciones dentro de la división de medio ambiente está codificada, el encabezado DIVISIÓN DE MEDIO AMBIENTE puede omitirse.
4. Los siguientes puntos pertenecen a la división de datos:
   - La división de datos consta de seis secciones opcionales; cuando se utilizan, esas secciones deben codificarse en el orden que se muestra en el diagrama de sintaxis.
   - Cada una de estas secciones consta de código que cumple una función específica. Si no se requiere código para la función de una de esas secciones, se puede omitir la sección completa, incluido el encabezado.
   - Si ninguna de las secciones dentro de la división de datos está codificada (una circunstancia altamente improbable, pero teóricamente posible), el encabezado DIVISIÓN DE DATOS en sí puede omitirse.
5. Los siguientes puntos pertenecen a la división del procedimiento:
   - Al igual que con las demás divisiones, la división de procedimientos puede constar de secciones y esas secciones pueden, a su vez, constar de párrafos. Sin embargo, a diferencia de las demás divisiones, los nombres de las secciones y los párrafos los define el programador, y puede que no haya ninguno definido si así lo desea el programador.
   - Cada rutina de manejo de eventos será una sección independiente dedicada a capturar un evento de tiempo de ejecución en particular. Si no hay secciones de este tipo codificadas, se pueden omitir las líneas DECLARATIVES. y END DECLARATIVES.
6. Un solo archivo de código fuente COBOL puede contener:
   - Una parte de un programa; estos archivos se conocen como cuadernos de copias

- Un solo programa. En este caso, la instrucción END PROGRAM o END FUNCTION es opcional.

- Múltiples programas, separados entre sí por instrucciones END PROGRAM o END FUNCTION. El programa final en un archivo de código fuente de este tipo no necesita tener una instrucción END PROGRAM o END FUNCTION.

7. El subprograma 'B' puede estar anidado dentro del programa 'A' incluyendo el código fuente del programa B al final de la división de procedimientos del programa A sin una instrucción END PROGRAM A. o END FUNCTION A. intermedia. Por ahora, eso es todo lo que se dirá sobre la anidación. VerSección 11.2 [Subprogramas independientes, contenidos y anidados], página 719, para obtener más información.

8. Independientemente de cuántos programas componen un único archivo fuente GnuCOBOL, solo se generará un único programa ejecutable de salida a partir de ese archivo fuente cuando se compile el archivo.

2.1.18 Comentarios

La siguiente información describe cómo se pueden incorporar comentarios en el código fuente del programa GnuCOBOL para proporcionar documentación.

Tipo de comentario	Modo fuente: descripción
Líneas en blanco	**FIJADO**—Se pueden insertar líneas en blanco según se desee. **GRATIS**—Se pueden insertar líneas en blanco según se desee.
Línea completacomentarios	**FIJADO**— Una línea fuente completa será tratada como un comentario (y será ignorada por el compilador) codificando un asterisco ('*') en la columna siete (7). **GRATIS**— Una línea de código fuente completa se tratará como un comentario (y será ignorado por el compilador) codificando la secuencia '*>', comenzando en cualquier columna, como los primeros caracteres que no están en blanco en la línea.
Comentarios de línea completa con avance de página	**FIJADO**— Una línea de código fuente completa se tratará como un comentario codificando una barra ('/') en la columna siete (7). Muchos compiladores COBOLTambién se emitirá un salto de página en el listado del programa, de modo que la línea '/' se encuentre en la parte superior de una nueva página. El compilador GnuCOBOL no admite este comportamiento de salto de página. El compilador interactivo GnuCOBOL, o GCic, admite esto ¡Comportamiento de avance de página cuando genera listados de fuentes de programas! VerSección "GCic" en Programas de ejemplo de GnuCOBOL, para el listado de fuentes y referencias cruzadas (producido por GCic) de este programa — puede ver el efecto de '/' allí. **GRATIS**— No existe un modo de fuente libre equivalente a '/'.
Comentarios de líneas parciales	**FIJADO**— Cualquier texto que aparezca después de la secuencia de caracteres '*>' en una línea de origen se considerará un comentario. El '*' debe aparecer en la columna siete (7) o en las

siguientes.

GRATIS— Cualquier texto que siga a la secuencia de caracteres '*>' en una línea de origen se considerará un comentario. El asterisco (*) puede aparecer en cualquier columna.

Comentarios que pueden tratarse como código, normalmente con fines de depuración.

FIJADO— Al codificar una 'D' en la columna 7 (mayúscula o minúscula), una línea fuente de GnuCOBOL que de otro modo sería válida será tratada como un comentario por el compilador.

GRATIS— Al especificar la secuencia de caracteres '>>D' (mayúsculas) o minúsculas) como los primeros caracteres que no están en blanco en una línea de origen, una línea de origen GnuCOBOL que de otro modo sería válida será tratada como un comentario por el compilador.

Las declaraciones de depuración se pueden compilar especificando -fdebugging-line en el compilador GnuCOBOL o medianteagregando la cláusula WITH DEBUGGING MODE al párrafo SOURCE-COMPUTER.

2.1.19 Literales

Los literales son valores constantes que no cambian durante la ejecución de un programa. Existen dos tipos fundamentales de literales: numéricos y alfanuméricos.

2.1.19.1 Literales numéricos

Un literal numérico

- Números enteros como 1, 56, 2192 o -54.
- Valores de punto fijo no enteros, como 1,317 o -2,95.
- Valores de punto flotante que utilizan la notación 'Enn', como 9.92E25, que representa 9.92 x 1025 (10 elevado a la 25.ª potencia) o 5.7E-14, que representa 5.7 x 10−14 (10 elevado a la -14.ª potencia). Tanto la mantisa (el número antes de la 'E') como el exponente (el número después de la 'E') pueden especificarse explícitamente como positivos (con un '+'), negativos (con un '-') o sin signo (y, por lo tanto, implícitamente positivos). Un valor literal de punto flotante debe estar dentro del rango −1,7 x 10308 a +1,7 x 10308 con no más de 15 dígitos decimales de precisión.
- Literales numéricos hexadecimales
- Literales con terminación nula
- Cadena C sin formato que utiliza caracteres "L".
- Binario usando B#0 o 1.
- Octal usando O#0 - 7. (Esa es la letra 'O').
- Número hexadecimal que utiliza H# o X# '0' - 'F'.
- Literales booleanos (estándar) Carácter B".
- Literales booleanos (hexadecimales) BX" carácter hexadecimal ".
- Literales Nacionales (Estándar) Carácter "N" o carácter "NC".
- Literales Nacionales (Hexadecimales) NX "carácter".

2.1.19.2 Literales alfanuméricos

Un literal alfanumérico

Un literal alfanumérico no es válido para su uso en expresiones aritméticas a menos que primero se convierta a su equivalente computacional numérico; hay tres funciones intrínsecas de conversión numérica integradas en GnuCOBOL que pueden realizar esta conversión: NUMVAL (verSección 8.1.70 [NUMVAL], página 500), NUMVAL-C (verSección 8.1.71 [NUMVAL-C], página 501) y NUMVAL-F
(verSección 8.1.73 [NUMVAL-F], página 503).

Los literales alfanuméricos pueden tomar cualquiera de las siguientes formas:

- Una secuencia de caracteres encerrados entre comillas simples (''')

- Un literal formado de acuerdo con las mismas reglas que para un literal de cadena (arriba), pero con el prefijo de la letra 'Z' (mayúscula o minúscula) constituye un literal de cadena delimitado por ceros. Estos literales se diferencian de los literales de cadena ordinarios en que se terminarán explícitamente con un byte de valor hexadecimal 00. Estos literales alfanuméricos delimitados por ceros
- Un literal alfanumérico hexadecimal

Los literales alfanuméricos que son demasiado largos para caber en una sola línea pueden continuar en la siguiente línea de una de dos maneras:

1. Si está utilizando el modo de formato fijo, el literal alfanumérico se puede ejecutar hasta la columna 72 inclusive. Luego, el literal se puede continuar en la siguiente línea en cualquier lugar después de la columna 11 codificando otra comilla o apóstrofo (lo que se haya utilizado para comenzar el literal originalmente). La línea de continuación también debe tener un guion (-)

```
         1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012345

      01 DEMOSTRACIÓN DE VALOR LITERAL LARGO        PIC X(60) VALOR "Este es un l largo
     -                                                 "ong literal que
     -                                                 " debe ser continuo
     -                                                 "ed."
```

2. Independientemente de si el compilador está funcionando en modo fijo o de formato libre, GnuCOBOL permite dividir los literales alfanuméricos en fragmentos separados. Estos fragmentos tienen sus propios caracteres de comillas o apóstrofes iniciales y finales y se "unen" en el momento de la compilación mediante el uso de '&'.

```
         1         2         3         4         5         6         7
123456789012345678901234567890123456789012345678901234567890123456789012345

      01 DEMOSTRACIÓN DE VALOR LITERAL LARGO PIC X(60) VALOR "Esto es un" &
                                                "literal largo que debe" &
                                                "continuará.".
```

Si su programa utiliza el modo de formato libre, habrá menos necesidad de continuar con literales alfanuméricos largos porque las declaraciones pueden tener hasta 255 caracteres.

Los literales numéricos se pueden dividir en líneas, al igual que los literales alfanuméricos, utilizando cualquiera de las técnicas anteriores, y tanto las palabras reservadas como las definidas por el usuario también se pueden dividir en líneas (utilizando la primera técnica). La continuación de los literales numéricos y las palabras reservadas o definidas por el usuario se proporciona simplemente para proporcionar compatibilidad con versiones y programas COBOL más antiguos, pero no se debe utilizar con programas nuevos, ya que solo hace que los programas tengan un aspecto feo.

2.1.19.3 Constantes figurativas

Las constantes figurativas son palabras reservadas que pueden utilizarse como literales en cualquier lugar donde el valor de las constantes figurativas pueda interpretarse como una secuencia arbitrariamente larga de los caracteres en cuestión. Cuando se requiere una longitud específica, como sería el caso de un argumento para un subprograma, no se puede utilizar una constante figurativa. Por lo tanto, los siguientes son usos válidos de las constantes figurativas:

```
05 RELLENO                PIC 9(10) VALOR CEROS.
   - - -
MOVER ESPACIOS A Nombre del empleado
```

Pero esto no es:

```
LLAMAR "SUBPGM" USANDO ESPACIOS
```

Las siguientes son las constantes figurativas de GnuCOBOL y sus respectivos valores equivalentes.

CERO Esta constante figurativa tiene un valor numérico 0 (cero). CEROS y CEROS son sinónimos de CERO.

ESPACIO Esta constante figurativa tiene un valor de uno o más caracteres de espacio. SPACES es sinónimo de SPACE.

CITA Esta constante figurativa tiene un valor de uno o más caracteres de comillas dobles ("). QUOTES es sinónimo de CITA.

BAJO-VALOR

Esta constante figurativa tiene un valor de uno o más de los caracteres que ocupan la posición más baja en la secuencia de ordenación del programa, tal como se define en OBJECT-COMPUTER (verSección 5.1.2 [OBJECT-COMPUTER], página 88) párrafoo —si no se hizo tal especificación— en cualquier conjunto de caracteres predeterminado que esté usando el programa (normalmente, se trata del conjunto de caracteres ASCII). LOW-VALUES es un sinónimo de LOW-VALUE.

Cuando el conjunto de caracteres en uso es ASCII sin modificaciones de secuencia de intercalación, el valor de la constante figurativa LOW-VALUES es el carácter ASCII NUL. Sin embargo, como los conjuntos de caracteres se pueden redefinir, no debe confiar en este hecho. En su lugar, utilice la constante figurativa NULL.

ALTO VALOR

Esta constante figurativa tiene un valor de uno o más de los caracteres que ocupan la posición más alta en la secuencia de clasificación del programa, tal como se define en laPárrafo OBJECT-COMPUTER o —si no se hizo tal especificación— en el conjunto de caracteres predeterminado que esté usando el programa (normalmente, se trata del conjunto de caracteres ASCII). HIGH-VALUES es un sinónimo de HIGH-VALUE.

NULO Un carácter compuesto enteramente de bits cero (independientemente de la secuencia de clasificación del programa).

Los programadores pueden crear sus propias constantes figurativas a través de los CARACTERES SIMBÓLICOS(verSección 5.1.3.4 [Cláusula de caracteres simbólicos], página 98) cláusula de los NOMBRES ESPECIALES (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) párrafo.

2.1.20 Puntuación

Una coma (',')

El uso de caracteres de coma puede causar confusión en un compilador COBOL si se utiliza la cláusula DECIMAL POINT IS COMMA en SPECIAL-NAMES (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90) párrafo, como podría ser el caso en Europa. La siguiente declaración, que llama a una subrutina pasándole dos argumentos (las constantes numéricas 1 y 2):

```
LLAMAR A "SUBRUTINA" USANDO 1,2
```

¿Será — con PUNTO DECIMAL ESCOMA en efecto — en realidad se puede interpretar como una llamada a una subrutina con 1 argumento (el literal numérico no entero cuyo valor es 1 y 2 décimas). Por este motivo, es mejor siempre ir seguido de una coma con un espacio.

El carácter de punto ('.')

Las reglas sobre dónde y cuándo se necesitan los períodos en la división de procedimientos son algo complicadas. VerSección 2.1.28 [Uso de los períodos], página 53, para más detalles.

2.1.21 Interfaz con otros entornos

A través de la instrucción CALL, los programas COBOL pueden invocar otros programas COBOL que actúen como subprogramas. Esto es bastante similar a las capacidades de

enlace entre programas que ofrecen otrosEn el caso de GnuCOBOL, la función CALL es lo suficientemente potente como para adaptarse al punto en que un programa GnuCOBOL puede comunicarse con el sistema operativo, la administración de bases de datos y las API de bibliotecas de tiempo de ejecución, incluso si no fueron escritas en COBOL. VerSección 11.8.4

[Programas principales de GnuCOBOL que llaman a subprogramas de C], página 732, como ejemplo de cómo un programa GnuCOBOL podría invocar un subprograma en lenguaje C, pasando información de ida y vuelta entre los dos.

El hecho de que GnuCOBOL admita una interfaz bidireccional completa con programas en lenguaje C significa que, incluso si no puede acceder directamente a una API de biblioteca, siempre podrá hacerlo a través de un pequeño programa "envoltorio" en C que sea LLAMADO por un programa GnuCOBOL.

2.1.22 Referencias de la tabla

COBOL utiliza paréntesis para especificar los subíndices utilizados para hacer referencia a las entradas de la tabla (las tablas en COBOL son lo que otros lenguajes de programación denominan matrices).

Por ejemplo, observe la siguiente estructura de datos que define una cuadrícula de caracteres de 4 columnas por 3 filas:

```
01    RED.
        05 LA CUADRÍCULA OCURRE 3 VECES.
            10 LA CUADRÍCULA-COLUMA OCURRE 4 VECES.
                15 CARACTERES DE CUADRÍCULA       FOTO X(1).
```

Si la estructura contiene la siguiente cuadrícula de caracteres:

```
ABCDEF
GHIJKL
```

Entonces GRID-CHARACTER (2, 3) hace referencia a 'G' y GRID-CHARACTER (3, 2) hace referencia a "Yo".

Los subíndices se pueden especificar como literales numéricos (enteros), elementos de datos numéricos (enteros), datos elementos creados con cualquiera de los enteros sin imagen USO (verSección 6.9.59 [USO], página 228) especificaciones, elementos de datos de ÍNDICE DE USO o expresiones aritméticas que resultan en un valor entero distinto de cero.

En los ejemplos anteriores, se utiliza una coma como carácter separador entre los dos valores de subíndice; los puntos y comas (;) también son caracteres separadores de subíndices válidos, ¡al igual que los espacios! El uso de una coma o un punto y coma como separador en una situación de este tipo es técnicamente opcional, pero por convención la mayoría de los programadores COBOL utilizan uno u otro. No se recomienda el uso de ningún carácter separador (excepto un espacio), aunque sea sintácticamente correcto, ya que esta práctica puede dar lugar a un código poco amigable para el programador. No es demasiado difícil leer y comprender GRID-CHARACTER(2 3), pero es otra historia completamente diferente cuando se intenta comprender GRID-CHARACTER(I + 1 J / 3) (en lugar de GRID-CHARACTER(I + 1, J / 3)). El compilador lo acepta, pero demasiado de esto me haría doler la cabeza.

2.1.23 Calificación de nombres de datos

COBOL permite que los nombres de datos se dupliquen dentro de un programa, siempre que se puedan hacer referencias a esos nombres de datos de tal manera que esas referencias sean únicas a través de un proceso conocido como calificación.

Para ver la calificación en funcionamiento, observe los siguientes segmentos de dos registros de datos definidos en un programa COBOL:

```
01 EMPLEADO.                          01  C L I E N T E .
    05 DIRECCIÓN POSTAL.                  05 DIRECCIÓN POSTAL.
        10  CALLE          FOT X(35).        10  CALLE          FOT X(35).
                            O                                    O
        10  CIUDAD         FOT X(15).        10  CIUDAD         FOT X(15).
                            O                                    O
        10  ESTADO         FOT X(2).         10  ESTADO         FOT X(2).
                            O                                    O
        10  CÓDIGO                            10  CÓDIGO
            POSTAL.                               POSTAL.
            15 CÓDIGO       FOT 9(5).             15 CÓDIGO      FOT 9(5).
            POSTAL-5        O                     POSTAL-5       O
            15 RELLENO      FOT X(4).             15 RELLENO     FOT X(4).
                            O                                    O
```

Ahora, abordemos el problema de configurar la parte CIUDAD de la DIRECCIÓN POSTAL de un EMPLEADO como "Filadelfia". Claramente,MOVE 'Filadelfia' A CIUDAD no puede funcionar porque el compilador no podrá determinar a cuál de los dos campos CIUDAD se refiere.

En un intento de corregir el problema, podríamos calificar la referencia a CIUDAD como MOVER 'Filadelfia' A CIUDAD DE DIRECCIÓN POSTAL.

Lamentablemente, eso también es insuficiente porque aún no especifica de manera suficiente a qué CIUDAD se hace referencia. Para identificar realmente qué CIUDAD específica desea, tendría que codificar MOVE 'Philadelphia' TO CITY OF MAILING-ADDRESS OF EMPLOYEE.

Ahora no puede haber confusión en cuanto a qué CIUDAD se está modificando. Afortunadamente, no es necesario ser tan específico; COBOL permite omitir los niveles de calificación intermedios e innecesarios. Esto permite que MOVE 'Philadelphia' TO CITY OF EMPLOYEE haga bien su trabajo.

Si necesita calificar una referencia a una tabla, hágalo codificando algo como identificador-1 DE identificador-2 (subíndice(s)).

La palabra reservada IN puede usarse en lugar de OF.

2.1.24 Modificadores de referencia

Sintaxis del modificador de referencia (formato 1)

identificador-1 [OF|IN identificador-2] [ ] (inicio:[longitud]) (subíndice...)

~~ ~~

Y O

Sintaxis del modificador de referencia (formato 2)

referencia de función intrínseca (inicio:[longitud])

Yo

El estándar COBOL 1985 introdujo el concepto de un modificador de referencia para facilitar referencias a sólo una parte de un elemento de datos; GnuCOBOL admite plenamente la modificación de referencias.

El valor inicial indica la posición del carácter inicial al que se hace referencia (los valores de posición del carácter comienzan con 1, no 0 como es el caso en algunos lenguajes de programación) y la longitud especifica cuántos caracteres se desean.

Si no se especifica ninguna longitud, se utilizará un valor equivalente a las posiciones de caracteres restantes desde el inicio.

Se asumirá hasta el final del identificador-1 o hasta el final del valor devuelto por la función.

Tanto el inicio como la longitud pueden especificarse como literales numéricos enteros, elementos de datos numéricos enteros o expresiones aritméticas con un valor entero.

A continuación se muestran algunos ejemplos:

APELLIDO DEL CLIENTE (1:3)
: Hace referencia a los primeros tres caracteres de APELLIDO-CLIENTE.

APELLIDO-CLIENTE (4:)
: Hace referencia a todas las posiciones de caracteres de APELLIDO-CLIENTE desde la cuarta en adelante.

FUNCIÓN FECHA ACTUAL (5:2)
: Hace referencia al mes actual como un número de 2 dígitos en formato de caracteres. VerSección 8.1.17 [FECHA ACTUAL], página 446, para obtener más información.

Dígitos hexadecimales (Nibble + 1:1)
: Suponiendo que Nibble es un elemento de datos numéricos con un valor en el rango de 0 a 15, y Hex-Digits es un elemento PIC X(16) con un valor de 0123456789ABCDEF, esto convierte ese valor numérico en un dígito hexadecimal.

Entrada a la mesa (6) (7:5)
: Hace referencia a los caracteres 7 a 11 (5 caracteres en total) en la sexta aparición de Table-Entry.

La modificación de referencia se puede utilizar en cualquier lugar donde un identificador sea legal, incluso sirviendo como campo receptor de declaraciones como MOVE (verSección 7.8.30 [MOVE], página 354), CADENA (verSección 7.8.47 [STRING], página 397) y ACEPTAR (verSección 7.8.1 [ACEPTAR], página 263), por nombrar algunos.

2.1.25 Expresiones aritméticas

Sintaxis de expresiones aritméticas

```
Expresión-unaria-1 { **      } Expresión unaria 2
                   {   *|/ }
                   {   +|- }
```

Sintaxis de expresión unaria

```
{ [ +|- ] { ( Expresión-Aritmética-1 )                } }
{         { [ LONGITUD DE ] { identificador-1      } } }
{         {    ~~~~~~ ~~    { literal-1             } } }
{         {                 { Referencia de función } } }
{Expresión aritmética-2                                 }
```

Las expresiones aritméticas se forman utilizando cuatro categorías de operaciones: exponenciación, multiplicación y división, suma y resta, y especificación de signos.

En expresiones complejas compuestas de múltiples operadores y operandos, se aplica una precedencia de operación según la cual aquellas operaciones que tienen una precedencia mayor se calculan primero antes que las operaciones con una precedencia menor.

Como es el caso en casi cualquier otro lenguaje de programación, el programador siempre tiene la libertad de usar pares de paréntesis para encerrar subexpresiones de expresiones complejas que se deben evaluar antes que otras subexpresiones en lugar de dejar que la precedencia del operador dicte la secuencia de evaluación.

En orden de precedencia más alto a más bajo, a continuación se presenta un análisis de cada categoría de operación:

Nivel 1 (más alto): especificación de signos unarios (+ y - con un solo argumento)

El operador unario "menos" (-) devuelve la negación aritmética de su único argumento, devolviendo efectivamente como su valor el producto de su argumento y -1.

El operador unario "más" (+) devuelve el valor de su único argumento, devolviendo efectivamente como su valor el producto de su argumento por +1.

Nivel 2 — Exponenciación (**)

El valor del argumento izquierdo se eleva a la potencia indicada por el argumento derecho. Se permiten potencias no enteras.

Nivel 3 — Multiplicación (*) y división (/)

El operador * calcula el producto de los argumentos izquierdo y derecho, mientras que el operador / calcula el valor del argumento izquierdo dividido por el valor del argumento derecho. Si el argumento derecho tiene un valor de cero, la evaluación de la expresión se terminará prematuramente antes de que se genere un valor. Esto puede provocar un error del programa en tiempo de ejecución.

Una secuencia de múltiples operaciones de tercer nivel (A * B / C, por ejemplo) se evaluará en estricta secuencia de izquierda a derecha si no se utilizan paréntesis

para controlar el orden de evaluación.

Nivel 4 — Suma (+) o resta (-)

El operador + calcula la suma de los argumentos izquierdo y derecho, mientras que el operador - calcula el valor del argumento derecho restado del valor del argumento izquierdo.

Una secuencia de múltiples operaciones de 4.º nivel (A - B + C, por ejemplo) se evaluará en estricta secuencia de izquierda a derecha si no se utilizan paréntesis para controlar el orden de evaluación.

Las reglas sintácticas de COBOL, que permiten un guión (-) en los nombres de elementos de datos, pueden generar cierta ambigüedad.

```
01   do      FOTO  9  VALOR  5.
01   D       FOTO  9  VALOR  2.
01   CD      FOTO  9  VALOR  7.
01   I       FOTO  9  VALOR  0.
---
CALCULAR I=C-D+1
```

El COMPUTE (verSección 7.8.9 [COMPUTE], página 297) La declaración evaluará la expresión aritmética C-D+1 y luego guardará ese resultado en I.

¿Qué valor se almacenará en I? ¿El número 4, que es el resultado de restar el valor de D (2) al valor de C (5) y luego sumar 1? ¿O será el número 8, que es el valor de sumar 1 al valor del dato CD (7)?

La respuesta correcta es 8: ¡el valor del elemento de datos CD más 1! Ojalá que ese fuera el resultado esperado.

El compilador GnuCOBOL en realidad siguió la siguiente lógica de toma de decisiones al generar código para la declaración COMPUTE:

1. ¿Hay un elemento de datos denominado CD definido? Si es así, utilice su valor para la secuencia de caracteres CD.
2. Si no hay ningún elemento de datos CD, ¿existen los elementos de datos C y D? Si no es así, la instrucción COMPUTE es errónea. Sin embargo, si los hay, se generará un código para restar el valor de D de C y sumar 1 al resultado.

Si hubiera habido al menos un espacio a la izquierda y/o a la derecha del -, no habría habido ninguna ambigüedad: el compilador se habría visto obligado a utilizar los elementos de datos individuales C y D.

Para evitar cualquier posible ambigüedad, así como para mejorar la legibilidad del programa, se considera una buena práctica de programación COBOL codificar siempre al menos un espacio tanto a la izquierda como a la derecha de cada operador en expresiones aritméticas, así como el signo = en un COMPUTE.

A continuación se muestran algunos ejemplos de cómo la precedencia de las operaciones afecta los resultados de las expresiones aritméticas (todos los ejemplos utilizan literales numéricos para simplificar la discusión).

Expresión	**Resultado**	**Notas**
3 * 4 + 1	13	* tiene prioridad sobre +

4 * 2 ** 3 - 10	22	2**3 es 8 (** tiene precedencia sobre *), por 4 es 32, menos 10 es 22.

(4 * 2) ** 3 - 10	502	Los paréntesis permiten una aplicación recursiva de la reglas de expresión aritmética, que permiten alterar efectivamente la precedencia de las operaciones. 4 por 2 es 8 (el uso de paréntesis "triunfa" sobre el operador de exponenciación, por lo que la multiplicación ocurre primero); 8 ** 3 es 512, menos 10 es 502.
5/2,5 + 7 * 2 - 1,15	14,85	Entero y los operandos no enteros pueden ser libremente entremezclado

Por supuesto, los operandos de expresiones aritméticas pueden ser elementos de datos numéricos (cualquier USO exceptoPUNTERO o PUNTERO DE PROGRAMA) así como literales numéricos.

2.1.26 Expresiones booleanas

Sintaxis de expresiones booleanas

Sintaxis general de expresiones booleanas

[Operando-1] { Operador booleano }

Operando-2B-NOT operando-3

Operadores booleanos	Significado
B-MAYÚS-L	Operación IZQUIERDA
CAMBIO B-LC	Operación de desplazamiento circular hacia la izquierda
B-MAYÚS-R	Operación Shift Right
Cambio B-RC	Operación de desplazamiento circular a la derecha

Operadores booleanos binarios	Significado
BANDA	Operación AND (conjunción booleana)
B-O	Operación OR inclusiva (disyunción inclusiva booleana)
B-XOR	Operación OR exclusiva (disyunción booleana exclusiva)

Operador booleano unario	Significado
B-NO	Operación de negación

——

Un operador booleano especifica el tipo de operación booleana que se realizará en uno o dos operandos, para un operador unario, un operador binario o un operador de desplazamiento respectivamente. Los operadores booleanos son los siguientes:

Los operadores de desplazamiento existían en otros dialectos antes de que se agregaran al estándar:

B-MAYÚS-L-> B-IZQUIERDA + B-MAYÚS-R-> B-DERECHA;

que también es compatible con GnuCOBOL (si no usa un dialecto estricto que no tenga estas palabras reservadas). Operan sobre el valor binario (no se realiza empaquetado ni desempaquetado).

Los operadores binarios se pueden utilizar en cálculos o expresiones. Ambos lados pueden ser un elemento numérico o literal (incluidas las notaciones hexadecimales especiales;

La evaluación de una expresión booleana se realizará de la siguiente manera:

a) [... paréntesis...]
b) La precedencia de operaciones en el mismo nivel de inclusividad, es: 1º Negación (B-NOT)
2do Conjunción (B-AND)
3º Disyunción exclusiva (B-XOR)
4ª Disyunción inclusiva (B-OR)

La precedencia de las operaciones de desplazamiento booleano (B-SHIFT-L, B-SHIFT-R, B-SHIFT-LC y B-SHIFT-RC) es la misma que la de la operación anterior, si la hubiera. Si la operación de desplazamiento booleanono está precedida por otra operación, la precedencia de esa operación es la misma que B-AND.

c) Cuando la secuencia de evaluación no se especifica mediante paréntesis, la evaluación de operaciones con la misma precedencia se realizará de izquierda a derecha.

2.1.27 Expresiones condicionales

Las expresiones condicionales son expresiones que identifican las circunstancias en las que un programa puede realizar una acción o dejar de realizarla. Por lo tanto, las expresiones condicionales producen un valor VERDADERO o FALSO.

Hay siete tipos de expresiones condicionales, como se analiza en las siguientes secciones.

2.1.27.1 Nombres de condiciones

Éstas son las condiciones más sencillas de todas. Observe el código siguiente:

```
05  TALLA DE CAMISA        Imagen 99V9.
    88  DIMINUTO           VALOR  0 A 12,5
    88  XS                 VALOR  13 AL 13.5.
    88  S                  VALOR  14,  14.5.
    88  METRO              VALOR  15,  15.5.
    88  yo                 VALOR  16,  16.5.
    88  SG                 VALOR  17,  17.5.
    88  XXL                VALOR  18,  18.5.
    88  XXXL               VALOR  19,  19.5.
    88 MUY GRANDE          VALOR  20 AL 99,9.
```

Los nombres de condición TINY, XS, S, M, L, XL, XXL, XXXL y VERY-LARGE tendrán VERDADERO o Valores FALSOS basados en los valores dentro de su elemento de datos principal (TALLA DE CAMISA).

Un programa que quiera probar si el valor actual de TALLA DE CAMISA puede clasificarse como XL podría tener esa decisión codificada como una condición combinada (el tipo más complejo de expresión condicional), como:

```
SI TALLA DE CAMISA = 17 O TALLA DE CAMISA = 17.5

- o -

SI LA TALLA DE LA CAMISA = 17 O 17.5
```

O simplemente podría utilizar el nombre de condición XL de la siguiente manera:

```
SI XL
```

2.1.27.2 Condiciones de la clase

Sintaxis de clase-condición

```
identificador-1 ES [ NO ] { NUMÉRICO    }
                ~~~       { ~~~~~~~     }
                          { ALFABETICO  }
                          { ~~~~~~~~~~  }
                          { ALFABETICO-MINUSCULO }
                          { ~~~~~~~~~~~~~~~~~~~~ }
                          { ALFABETICO-MAYÚS }
                          { ~~~~~~~~~~~~~~~~~~~~ }
                          { OMITIDO     }
                          { ~~~~~~~     }
                          { nombre-clase-1 }
```

Las condiciones de clase evalúan el tipo de datos que están almacenados actualmente en un elemento de datos.

1. La prueba de clase NUMÉRICA considera que solo los caracteres '0', '1', . . . , '9' son numéricos; solo un elemento de datos que no contenga nada más que dígitos pasará una prueba de clase NUMÉRICA. Los espacios, puntos decimales, comas, signos de moneda, signos más, signos menos y cualquier otro carácter excepto los caracteres numéricos no pasarán las pruebas de clase NUMÉRICA.
2. La prueba de la clase ALFABÉTICA considera únicamente letras mayúsculas, minúsculas y espacios como de naturaleza alfabética.
3. Las condiciones de clase ALFABÉTICO-MINÚSCULAS y ALFABÉTICO-MAYÚSCULAS consideran como aceptables únicamente los espacios y el respectivo tipo de letras para aprobar dicha prueba de clase.
4. La opción NO invierte el valor VERDADERO/FALSO de la condición.
5. Tenga en cuenta que lo que constituye una "letra" (o mayúscula/minúscula también, para ese caso) puede verse influenciado mediante el uso de las especificaciones de CLASIFICACIÓN DE CARACTERES en la COMPUTADORA-OBJETO (verSección 5.1.2 [OBJECT-COMPUTER], página 88) párrafo.
6. Sólo los elementos de datos cuyo USO (verSección 6.9.59 [USO], página 228) se define explícita o implícitamente como DISPLAY y se puede utilizar en condiciones de clase NUMÉRICA o ALFABETICA.
7. Algunas implementaciones de COBOL no permiten el uso de elementos de grupo o elementos PIC A con condiciones de clase NUMÉRICAS y el uso de elementos PIC 9 con condiciones de clase ALFABETICAS. GnuCOBOL no tiene tales restricciones.
8. La condición de clase OMITTED se utiliza cuando es necesario que un subprograma determine si se le pasó o no un argumento en particular. En tales condiciones de clase, el identificador-1 debe ser un elemento de sección de enlace definido en la cláusula USING del encabezado PROCEDURE DIVISION del subprograma. ConsulteSección 7.1 [DIVISIÓN DE PROCEDIMIENTOS DE USO], página 244, para obtener información adicional.

La opción class-name-1 le permite probar una clase definida por el usuario. A continuación se muestra un ejemplo. Primero, suponga que se trata de los siguientes NOMBRES ESPECIALES (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90) definición de la clase definida por el usuario 'Hexadecimal':

```
NOMBRES ESPECIALES.
```

LA CLASE hexadecimal es '0' HASTA '9', 'A' HASTA 'F', 'a' HASTA 'f'.

Ahora observe el siguiente código, que ejecutará el procedimiento 150-Process-Hex-Value si Entered-Value no contiene nada más que dígitos hexadecimales válidos:

```
SI el valor ingresado es
    hexadecimal EJECUTAR 150-
    Procesar valor hexadecimal
FIN-SI
```

2.1.27.3 Condiciones de la señal

Sintaxis de signo-condición

```
identificador-1 ES [ NO ] { POSITIVO }
                   ~~~    { ~~~~~~~~ }
                          { NEGATIVO }
                          { ~~~~~~~~ }
                          { CERO     }
                            ~~~~
```

Las condiciones de signo evalúan el estado numérico de un elemento de datos definido con una IMAGEN (verSección 6.9.37 [IMAGEN], página 195) y/o USO (verSección 6.9.59 [USO], página 228) que admite valores numéricos.

1. Una condición de clase POSITIVA o NEGATIVA será VERDADERA solo si el valor del identificador-1 es estrictamente mayor o menor que cero, respectivamente.
2. Una condición de clase CERO se puede pasar solo si el valor del identificador-1 es exactamente cero.
3. La opción NO invierte el valor VERDADERO/FALSO de la condición.

2.1.27.4 Condiciones de estado del interruptor

En el párrafo NOMBRES ESPECIALES, se puede asociar un nombre de interruptor externo con uno o más nombres de condición. Estos nombres de condición se pueden utilizar para probar el estado de encendido/apagado del interruptor externo.

Aquí están las secciones de código relevantes en un programa llamado testprog, que está diseñado simplemente para anunciar si SWITCH-1 está activado:

```
---
AMBIENTED I V I S I Ó N
.
NOMBRES ESPECIALES.
    EL ESTADO DEL INTERRUPTOR 1 ENCENDIDO ES El interruptor 1 está encendido.
---
DIVISIÓN DE PROCEDIMIENTOS.
---
    SI el interruptor 1 está encendido
        MOSTRAR "El interruptor
    1 está encendido" FINALIZAR SI
---
```

A continuación se muestran dos sesiones de ventana de comandos diferentes (la izquierda en un sistema Unix/Cygwin/OSX y la derecha en un sistema Windows) que activarán el interruptor y luego ejecutarán el programa testprog. Observe cómo se muestra el mensaje que indica que el programa detectó que se configuró el interruptor en ambos ejemplos:

```
$COB_SWITCH_1=ENCENDIDO        C:>CONFIGURAR  I N T E R R U P T O R  C O B _ 1 = E N C E N D I D O
$ exportar COB_SWITCH_1        C:>progprueba
$./testprog                    El interruptor 1 está encendido
El interruptor 1 está encendido C:>
$
```

2.1.27.5 Condiciones de relación

Sintaxis de relación-condición

```
{ identificador-1          } ES [ NO ] RelOp { identificador-2          }
{ literal-1                }       ~~~       { literal-2                }
{ expresión-aritmética-1 }                   { expresión-aritmética-2 }
{ nombre-índice-1          }                 { nombre-índice-2          }
```

Sintaxis RelOp

```
{ IGUAL A                      }
{ ~~~~~ IGUAL                  }
{ ~~~~~~ MAYOR                 }
{ QUE                          }
{ ~~~~~~~                      }
{ MÁS QUE                      }
{                 O IGUAL    A }
{ ~~~~~~~         ~~ ~~~~~     }
{ MENOS QUE                    }
{ ~~~~                         }
{ MENOS QUE O  IGUAL A         }
{ ~~~~         ~~  ~~~~~       }
{ =                            }
{ >                            }
{ >=                           }
{ <                            }
{ <=                           }
{ <>                           }
```

Estas condiciones evalúan cómo dos valores diferentes se "relacionan" entre sí.

1. Al comparar un valor numérico con otro, el USO (verSección 6.9.59 [USO], página 228) y el número de dígitos significativos en cada valor son irrelevantes ya que la comparación se realiza utilizando los valores algebraicos reales.
2. Al comparar cadenas, la comparación se realiza en base a la secuencia de ordenación del programa. Cuando los dos argumentos de cadena tienen una longitud diferente, se supone que el más corto se rellena (a la derecha) con una cantidad suficiente de espacios para que las dos cadenas tengan la misma longitud. Las comparaciones de cadenas se realizan carácter por carácter, de izquierda a derecha, hasta que se puede establecer el valor VERDADERO/FALSO para la prueba de relación. Los caracteres se comparan de acuerdo con su posición relativa en la SECUENCIA DE ORDENACIÓN del programa (como se define en NOMBRES ESPECIALES (verSección 5.1.3 [NOMBRES ESPECIALES], página 90)), no de acuerdo con los valores de patrón de bits que los caracteres tienen almacenados.
3. De manera predeterminada, la SECUENCIA DE COMPARACIÓN del programa se basará completamente en los valores de patrones de bits de los distintos caracteres.
4. No existe diferencia funcional entre utilizar la versión verbosa (ES IGUAL A, ES MENOS).

QUE, . . .) frente a la versión simbólica (=, <, . . .) de los operadores de relación reales. Lo mismo se aplica a (NO ES IGUAL A frente a la versión simbólica (<>).

Como ejemplo podemos escribir

```
Si AAA <> 0 es lo mismo que si AAA no = 0
si AAA no <> 0 es lo mismo que si AAA = 0 - (aunque el significado
es un poco oscuro).
```

2.1.27.6 Condiciones combinadas

Sintaxis de condición combinada

```
[ ( ] Condición-1 [ ) ] { AND } [ ( ] Condición-2 [ ) ]
                        { ~~~ }
                        { O   }
                        { ~~  }
```

Una condición combinada es aquella que calcula un valor VERDADERO/FALSO a partir de los valores VERDADERO/FALSO de otras dos condiciones (que podrían ser condiciones combinadas).

1. Si alguna de las condiciones tiene un valor VERDADERO, el resultado de la operación OR entre las dos dará como resultado un valor VERDADERO. La operación OR entre dos condiciones FALSAS dará como resultado un valor FALSO.
2. Para que AND dé como resultado VERDADERO, ambas condiciones deben tener un valor VERDADERO. En todas las demás circunstancias, AND produce un valor FALSO.
3. Al encadenar varias condiciones similares con el mismo operador (OR/AND) y los argumentos izquierdo o derecho tienen sujetos comunes, es posible abreviar el código del programa. Por ejemplo:

   SIESTADO DE CUENTA = 1 O ESTADO DE CUENTA = 2 O ESTADO DE CUENTA = 7

   Podría abreviarse como:

   SIESTADO DE LA CUENTA = 1 O 2 O 7
4. Así como la multiplicación tiene prioridad sobre la suma en expresiones aritméticas, AND tiene prioridad sobre OR en condiciones combinadas. Utilice paréntesis para cambiar esta prioridad, si es necesario. Por ejemplo:

   FALSO Y FALSO O VERDADERO Y VERDADERO
   Se evalúa como VERDADERO

   (FALSO Y FALSO) O (VERDADERO Y VERDADERO)
   Se evalúa como VERDADERO (ya que AND tiene precedencia sobre OR): esto es idéntico al ejemplo anterior

   (FALSO Y (FALSO O VERDADERO)) Y VERDADERO
   Se evalúa como FALSO

2.1.27.7 Condiciones Negadas

Sintaxis de condición negada

NO Condición-1
~~~

———————————————————————————————————————

Una condición se puede negar anteponiéndole el operador NOT.

1. El operador NOT tiene la mayor precedencia de todos los operadores lógicos, al igual que un signo menos unario (que "niega" un valor numérico) es el operador aritmético de mayor precedencia.
2. Se deben utilizar paréntesis para indicar explícitamente la secuencia en la que se evalúan y procesan las condiciones si no se desea la precedencia predeterminada. Por ejemplo:

   NO ES VERDADERO Y FALSO Y NO FALSO
   Se evalúa como FALSO Y FALSO Y VERDADERO, lo que se evalúa como FALSO.

   NO (VERDADERO Y FALSO Y NO FALSO)
   Se evalúa como NO (FALSO), que se evalúa como VERDADERO.

   NO ES VERDADERO Y (FALSO Y NO FALSO)
   Se evalúa como FALSO Y (FALSO Y VERDADERO) que se evalúa como FALSO
~~~

2.1.28 Uso de los puntos

Todas las implementaciones de COBOL distinguen entre oraciones y declaraciones en la división de procedimientos. Una declaración es una única instrucción COBOL ejecutable. Por ejemplo, todas estas son declaraciones:

```
MOVER ESPACIOS A Dirección de
empleado AGREGAR 1 A Contador de
registros
MOSTRAR "Contador de registros=" Contador de registros
```

Algunas sentencias COBOL tienen un ámbito de aplicación asociado a ellas, en el que una o más sentencias pueden considerarse parte de la sentencia en cuestión o estar relacionadas con ella. Un ejemplo de una situación de este tipo podría ser el siguiente, en el que el interés de un préstamo se calcula y se muestra a un tipo de interés del 4 % si el saldo del préstamo es inferior a 10 000 $ y del 4,5 % en caso contrario. (ADVERTENCIA: el siguiente código contiene un error):

```
SI Saldo del préstamo < 10000
    MULTIPLICA el saldo del préstamo POR 0,04 DANDO
interés DE LO CONTRARIO
    MULTIPLICA el saldo del préstamo POR 0,045 DANDO
Interés MOSTRAR "Monto de interés = " Interés
```

En este ejemplo, la declaración IF en realidad tiene un alcance que puede incluir dos conjuntos de declaraciones asociadas: un conjunto que se ejecutará cuando se ejecute la declaración IF (verSección 7.8.23 [IF], página 338) condición es VERDADERA, y otra si es FALSA.

Lamentablemente, hay un problema con lo anterior. Un ser humano que mire ese código probablemente inferiría que el DISPLAY (verSección 7.8.12 [PANTALLA], página 302) debido a su falta de sangría, se debe ejecutar independientemente del valor VERDADERO/FALSO de la condición IF. Desafortunadamente, el compilador (cualquier compilador COBOL) no lo verá de esa manera porque realmente no le importa qué tipo de sangría se use, si es que se usa alguna. De hecho, cualquier compilador COBOL estaría igualmente feliz de ver el código escrito de esta manera:

```
SI Saldo del préstamo < 10000 MULTIPLICA Saldo
del préstamo POR 0,04 DANDO Interés DE LO
CONTRARIO MULTIPLICA
Saldo del préstamo POR 0,045 DANDO Interés
MOSTRAR "Monto de interés = " Interés
```

¿Cómo entonces informamos al compilador que la declaración DISPLAY está fuera del alcance de la función? ¿SI?

Ahí es donde entran las oraciones.

Una sentencia COBOL se define como cualquier secuencia de instrucciones de longitud arbitraria, seguida de un punto (.). El punto es lo que pone fin al alcance de un conjunto de instrucciones. Por lo tanto, nuestro ejemplo debería haberse codificado de la siguiente manera:

```
SI Saldo del préstamo < 10000
    MULTIPLICA el saldo del préstamo POR 0,04 DANDO
interés DE LO CONTRARIO
    MULTIPLICA el saldo del préstamo por 0,045 y obtienes el interés.
MOSTRAR "Monto de interés = " Interés
```

Vea el período al final del segundo MULTIPLICAR (verSección 7.8.31 [MULTIPLICAR], página 356)? Eso es lo que termina el alcance del IF, haciendo así que la ejecución de la declaración DISPLAY sea completamente independiente del estado VERDADERO/FALSO del IF.

2.1.29 Uso de construcciones VERBO/FINAL DEL VERBO

Antes del estándar COBOL de 1985, utilizar un punto era la única forma de señalar el final del alcance de una declaración.

Lamentablemente, esto provocó algunos problemas. Eche un vistazo a este código:

```
Si A = 1
    Si B = 1
        PANTALLA "A y B = 1"
ELSE *> ¡Este ELSE tiene un problema!
    Si B = 1
        MOSTRAR "A NO = 1 PERO B = 1"
    DE LO CONTRARIO
        MOSTRAR "NI A NI B = 1".
```

El problema con este código es que la sangría, tan importante para mejorar la legibilidad de un programa, puede proporcionar una visión errónea del flujo lógico. Un ELSE siempre se asocia con el IF encontrado más recientemente; esto significa que el ELSE enfatizado se asociará con la declaración IF B = 1, no con la declaración IF A = 1, como parecería implicar la sangría.

Este tipo de problema llevó a que se añadiera una solución provisional al lenguaje COBOL: el SIGUIENTE FRASE cláusula:

```
Si A = 1
    Si B = 1
        MOSTRAR "A & B = 1"
    DE LO CONTRARIO
        SIGUIENTE FRASE
DEMÁS
    Si B = 1
        MOSTRAR "A NO = 1 PERO B = 1"
    DE LO CONTRARIO
        MOSTRAR "NI A NI B = 1".
```

LA SIGUIENTE FRASE informa al compilador que si la condición B = 1 es falsa, el control debe recaer en la primera declaración que sigue al siguiente período.

Con el estándar COBOL de 1985 se introdujo una solución mucho más elegante. Cualquier verbo COBOL (la primera palabra reservada de una declaración) que necesitara algo así podía usar una construcción END-verb para finalizar su alcance sin interrumpir el alcance de ninguna otra declaración en la que pudiera haber estado. Cualquier compilador COBOL 85 habría permitido la siguiente solución a nuestro problema:

```
Si A = 1
    Si B = 1
        MOSTRAR "A & B = 1"
    FIN-SI
DEMÁS
    Si B = 1
        MOSTRAR "A NO = 1 PERO B = 1"
    DE LO CONTRARIO
        MOSTRAR "NI A NI B = 1".
```

Esta nueva función hizo que el período quedara casi obsoleto, ya que nuestro segmento de programa probablemente estaría codificado así hoy:

```
Si A = 1
```

```
        Si B = 1
           MOSTRAR  "A y B = 1"
        DEMÁS
           MOSTRAR  "A = 1 PERO B NO      =  1"
        FIN-SI
    DEMÁS
        Si B = 1
           MOSTRAR  "A NO = 1 PERO B      =  1"
        DEMÁS
           MOSTRAR  "NI A NI B            =  1"
        FIN-SI
    FIN-SI
```

COBOL (incluido GnuCOBOL) todavía requiere que cada párrafo de división de procedimiento contenga al menos una oración si hay algún código ejecutable en ese párrafo, pero un estilo de codificación popular ahora es simplemente codificar un solo punto justo antes del final de cada párrafo.

El estándar para el lenguaje COBOL muestra que las distintas cláusulas verbales END son opcionales porque el uso de un punto como terminador de ámbito sigue siendo legal.

Si va a portar código existente a GnuCOBOL, encontrará que es una solución útil.facilidad capaz de cumplir con cualquier lenguaje y estándares de codificación que ese código probablemente use. Sin embargo, si está creando nuevos programas GnuCOBOL, le recomendaría enfáticamente que use las estructuras de verbos END en esos programas.

2.1.30 Acceso simultáneo a archivos

La manipulación de archivos de datos es una de las grandes fortalezas del lenguaje COBOL. COBOL cuenta con funciones integradas para abordar la posibilidad de que varios programas intenten acceder al mismo archivo simultáneamente. El acceso simultáneo de varios programas se gestiona de dos formas: compartiendo archivos y bloqueando registros.

No todas las implementaciones de GnuCOBOL admiten opciones de uso compartido de archivos y bloqueo de registros. El hecho de que lo hagan o no depende del sistema operativo para el que se crearon y de las opciones de creación que se usaron cuando se generó la implementación específica de GnuCOBOL.

2.1.30.1 Intercambio de archivos

GnuCOBOL controla el acceso simultáneo a archivos en el nivel más alto a través del concepto de compartición de archivos, que se aplica cuando un programa intenta abrir un archivo. Esto se logra mediante una rutina del sistema operativo UNIX llamada fcntl. Ese módulo no es compatible actualmente con Windows y no está presente en el paquete de emulación Unix MinGW. Las compilaciones de GnuCOBOL creadas utilizando un entorno MinGW no podrán soportar controles de compartición de archivos; los archivos siempre se compartirán en dichos entornos. Una compilación de GnuCOBOL creada utilizando el entorno Cygwin en Windows tendría acceso a fcntl y, por lo tanto, soportará la compartición de archivos. Por supuesto, las compilaciones Unix reales de GnuCOBOL, así como las compilaciones OSX, no deberían tener problemas porque fcntl debería estar disponible.

Cualquier limitación impuesta a una OPEN exitosa (verSección 7.8.32 [ABIERTA], página 358) permanecerá vigente hasta que su programa emita un CIERRE (verSección 7.8.7 [CERRAR], página 295) contra el archivo o el programa termina.

El intercambio de archivos se controla mediante el uso de una cláusula SHARING:

```
COMPARTIENDO CON { TODOS LOS DEMÁS }
                 ~~~~~~~~~~~~~~~~~~
                 { ~~~~~~~~~~~~~~~~ }
                 {NINGUNA OTRA      }
                 { ~~~~~~~~~~~~~~~~ }
                 { SOLO LECTURA     }
                   ~~~~ ~~~~~~~~~~~~
```

Esta cláusula se puede utilizar en la declaración SELECT del archivo (verSección 5.2.1 [SELECCIONAR], página 102), en la declaración OPEN (verSección 7.8.32 [ABIERTA], página 358) que inicia el uso del archivo por parte de su programa, o ambos. Si se especifica una opción SHARING en ambos lugares, las especificaciones realizadas en la declaración OPEN tendrán prioridad sobre las de la declaración SELECT.

Aquí están los significados de las tres opciones:

TODOS LOS DEMÁS Cuando su programa abre un archivo con esta opción de uso compartido activada, no se aplicarán restricciones a otros programas que intenten ABRIR el archivo después de que lo haya hecho su programa. Este es el modo de uso compartido predeterminado.

NINGÚN OTRO Cuando su programa abre un archivo con esta opción de uso compartido activada, su programa anuncia que no está dispuesto a permitir que ningún otro programa tenga acceso al archivo mientras esté usando ese archivo; los intentos de ABRIR realizados en otros programas fallarán con un estado de archivo de 37 (PERMISO DENEGADO) hasta que usted CIERRE (consulteSección 7.8.7 [CERRAR], página 295) el archivo.

SOLO LECTURA Abrir un archivo con esta opción de uso compartido indica que está dispuesto a

permitir que otros programas ABRIR el archivo para su entrada mientras lo tiene abierto. Si intentan cualquier otra opción de ABRIR, fallará con un estado de archivo de 37. Por supuesto, su programa puede fallar si otra persona llegó al archivo primero y lo abrió con una opción de uso compartido que impuso limitaciones de uso compartido de archivos.

Si la selección de un archivo está codificada con una cláusula FILE STATUS, el programa podrá detectar los fallos de OPEN (incluidos los inducidos por fallos de compartición) y será posible una recuperación sin problemas (o al menos una finalización sin problemas). Sin embargo, si no se ha codificado ninguna cláusula de este tipo, se emitirá un mensaje de tiempo de ejecución y se finalizará el programa.

2.1.30.2 Bloqueo de registros

El bloqueo de registros es posible gracias a un software de gestión de archivos avanzado integrado en la implementación de GnuCOBOL que está utilizando. Este software proporciona un único punto de control para el acceso a los archivos (normalmente, archivos INDEXADOS POR ORGANIZACIÓN). Uno de esos paquetes de tiempo de ejecución capaces de hacer esto es el paquete Berkeley Database (BDB), un paquete que se utiliza con frecuencia en las compilaciones de GnuCOBOL para admitir archivos indexados.

Las distintas sentencias de E/S que puede ejecutar su programa pueden imponer limitaciones al acceso de otros programas que se ejecutan simultáneamente al registro del archivo al que acaban de acceder. Estas limitaciones se imponen sintácticamente colocando un bloqueo en el registro mediante una cláusula LOCK. Los demás registros del archivo siguen estando disponibles, siempre que las limitaciones de uso compartido de archivos impuestas en el momento en que se abrió el archivo no impidieran el acceso a todo el archivo.

1. Si la compilación de GnuCOBOL que está utilizando fue configurada para utilizar el paquete Berkeley Database (BDB) para E/S de archivos indexados, el bloqueo de registros estará disponible mediante la variable de entorno de ejecución DB_HOME.
2. Si selecciona SELECT (verSección 5.2.1 [SELECCIONAR], página 102) declaración o archivoABIERTO (verSección 7.8.32 [ABIERTA], página 358) especifica COMPARTIRSIN OTRA, se desactivará el bloqueo de registros.
3. Si el SELECT del archivo contiene una cláusula LOCK MODE IS AUTOMATIC, cada vez que se lea un registro del archivo, dicho registro se bloqueará automáticamente. Otros programas pueden acceder a otros registros dentro del archivo, pero no a un registro bloqueado.
4. Si el SELECT del archivo contiene una cláusula LOCK MODE IS MANUAL, se colocan bloqueos en los registros solo cuando una declaración READ ejecutada contra el archivo incluye una cláusula LOCK (esta cláusula se analizará en breve).
5. Si se especifica la cláusula LOCK ON en el SELECT del archivo, los bloqueos (ya sea adquiridos de forma automática o manual) seguirán acumulándose a medida que se lean más y más registros, hasta que se liberen explícitamente. Esto se conoce como bloqueo de múltiples registros.

   Los bloqueos adquiridos a través del bloqueo de registros múltiples permanecen vigentes hasta que el programa que mantiene el bloqueo...

   - ... termina, o . . .
   - ... ejecuta una instrucción CLOSE (verSección 7.8.7 [CERRAR], página 295) contra el archivo, o . . .
   - ... ejecuta una instrucción UNLOCK (verSección 7.8.52 [DESBLOQUEAR], página 405) contra el archivo, o . . .
   - ... ejecuta una declaración COMMIT (verSección 7.8.8 [COMMIT], página 296) o . . .
   - ... ejecuta una instrucción ROLLBACK (verSección 7.8.40 [ROLLBACK], página 375).
6. Si no se especifica la cláusula LOCK ON, entonces la siguiente instrucción de E/S que ejecute su programa, excepto START (verSección 7.8.45 [INICIO], página 393), liberará el bloqueo. Esto se conoce como bloqueo de registro único.
7. Una cláusula LOCK, que puede codificarse en una LECTURA (verSección 7.8.34 [LEER], página 365), REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373) o la declaración WRITE (verSección 7.8.54 [ESCRIBIR], página 410) se ve así:

```
{ IGNORANDO BLOQUEO  }
{ ~~~~~~~~ ~~~~      }
{CON [ SIN ] BLOQUEO }
{            ~~    ~~~~ }
{ CON CERRADURA GUARDADA }
```

{ ~~~~ ~~~~ }

```
{ CON BLOQUEO IGNORAR }
{         ~~~~~~ ~~~~ }
{ CON ESPERA       }
          ~~~~
```

La opción WITH [NO] LOCK es la única disponible para las instrucciones REWRITE o WRITE. Los significados de las distintas opciones de bloqueo de registros son los siguientes:

IGNORAR BLOQUEO
CON BLOQUEO IGNORAR
: Estas opciones (que son sinónimas) informan a GnuCOBOL que cualquier bloqueo mantenido por otros programas debe ser ignorado.

CON BLOQUEO Se denegará el acceso al registro a otros programas.

CON NOCERRAR
: El registro no se bloqueará. Esta es la opción predeterminada para todas las declaraciones.

CON CERRADURA GUARDADA
: Cuando el bloqueo de un solo registro está en efecto, cuando se accede a un nuevo registro, se liberan los bloqueos mantenidos para registros anteriores. Al utilizar esta opción, no solo se bloquea el registro al que se accede recientemente (como lo haría WITH LOCK), sino que también se conservarán los bloqueos de registros anteriores. Una lectura posterior sin la opción KEPT LOCK liberará todos los bloqueos "mantenidos", al igual que la declaración UNLOCK.

CON ESPERA Esta opción informa a GnuCOBOL que el programa está dispuesto a esperar a que se libere un bloqueo mantenido (por otro programa) en el registro que se está leyendo.

Sin esta opción, cualquier intento de leer un registro bloqueado se cancelará inmediatamente y se devolverá un estado de archivo de 51.

Con esta opción, el programa esperará un tiempo preconfigurado para que se libere el bloqueo. Si el bloqueo se libera dentro del tiempo de espera preconfigurado, la lectura será exitosa. Si el tiempo de espera preconfigurado expira antes de que se libere el bloqueo, se cancelará el intento de lectura y se emitirá un estado de archivo 51.

———————————————————————

Fin del Capítulo 2: Fundamentos de COBOL

3 CDF - Instalación de dirección del compilador

La función de dirección del compilador, o CDF, es un medio para controlar la compilación de programas Gnu-COBOL. CDF proporciona un mecanismo para configurar o restablecer dinámicamente ciertos parámetros.Cambios de compilador, introducción de código fuente nuevo de una o más bibliotecas de código fuente, modificación dinámica del código fuente y procesamiento condicional o ignorar por completo las instrucciones fuente. Esto se logra mediante una serie de instrucciones y directivas CDF especiales que aparecerán en el código fuente del programa.

Cuando el compilador está operando en modo de formato fijo, todas las declaraciones CDF deben comenzar en la columna siete (7) o más allá.

Hay dos tipos de declaraciones CDF admitidas en GnuCOBOL: declaraciones de manipulación de texto y directivas del compilador.

Las instrucciones de manipulación de texto de CDF COPY y REPLACE se utilizan para introducir código nuevo en los programas, ya sea con o sin cambios, o pueden utilizarse para modificar instrucciones existentes en el programa. Las instrucciones de manipulación de texto siempre terminan con un punto.

Las directivas CDF, denotadas por la presencia de una secuencia de caracteres >> como parte del nombre de la declaración, influyen en el proceso de compilación del programa.

Las directivas del compilador nunca terminan con un punto.

La opción de línea de comandos del compilador -D ofrece control adicional (verSección 10.1.1 [cobc - El compilador GnuCOBOL], página 673).

3.1 >>CONVOCATORIA-CONVENCION

CDF>>Sintaxis de CONVENCIÓN DE LLAMADA

```
>>LLAMADA-CONVENCION          { COBOL }
~~~
```

3.2 COPIA

Sintaxis de la instrucción COPY de CDF

```
{ COPIAR  }     nombre del
{ ~~~~    }     cuaderno
{ INCLUIR }
{ ~~~~~~~ }

      [ IN|OF nombre-biblioteca ]
        ~~ ~~
      [ SUPRIMIR IMPRESIÓN ]
        ~~~~~~~~
      [ REEMPLAZO { Cláusula-Frase | Cláusula-Cadena }...  ]      .
        ~~~~~~~~~
```

Sintaxis de frase-cláusula COPY de CDF

```
{ ==pseudo-texto-1== } POR { ==pseudo-texto-2== }
{ identificador-1   } ~~ { identificador-2   }
{ literal-1         }    { literal-2         }
{ palabra-1         }    { palabra-2         }
```

Sintaxis de cláusula de cadena COPY de CDF

```
[ PRINCIPAL|FINAL ] ==palabra parcial 1== POR ==palabra parcial 2==
  ~~~~~~~ ~~~~~~~~                          ~~
```

1. Las instrucciones COPY se utilizan para importar libros de copias (verSección 2.1.6 [Cuadernos], página 13) en un programa.
2. Las instrucciones COPY se pueden usar en cualquier lugar dentro de un programa COBOL donde el código contenido en el libro de copias sería sintácticamente válido.
3. La cláusula opcional SUPPRESS (con o sin la palabra reservada PRINTING opcional) es válida sintácticamente, pero no funcional. Se admite para facilitar la compatibilidad con el código fuente escrito para otras versiones de COBOL.
4. No hay diferencia entre el uso de la palabra IN y la palabra OF: usa la que prefieras.
5. Un punto es absolutamente obligatorio al final de cada declaración COPY, incluso si la declaración ocurre dentro del alcance de otra en la que un punto podría parecer disruptivo, como dentro del alcance de un IF (verSección 7.8.23 [IF], página 338). Sin embargo, este punto obligatorio al final de la declaración no afecta el alcance de la declaración en la que se encuentra la COPIA.
6. Tanto el pseudo-texto-2 como la palabra-parcial-2 pueden ser nulos.

7. Se ubican todas las instrucciones COPY y se insertan los contenidos de los libros de copias correspondientes en el código fuente del programa antes de que comience el proceso de compilación real. Si un libro de copias contiene una instrucción COPY, se repetirá el proceso de inserción del libro de copias para resolver la COPY incorporada. Esto continuará hasta que no queden instrucciones COPY sin resolver. En ese momento, comenzará la compilación real del programa.
8. VerSección 10.1.6 [Ubicación de los cuadernos de copias], página 690, para conocer las reglas específicas sobre cómo el compilador ubica los libros de copias.
9. La cláusula opcional REPLACING permite realizar uno o más de los siguientes tipos de reemplazos de texto:

Cláusula-frase

Reemplazo de una o más palabras reservadas completas, identificadores definidos por el usuario o literales; los siguientes puntos se aplican a esta opción:

- Esta opción no se puede utilizar para reemplazar parte de una palabra, identificador o literal.
- Todo lo que precede al BY se denominará aquí cadena de búsqueda.
- Las cadenas de búsqueda de un solo elemento se pueden especificar codificando el identificador-1, el literal-1 o la palabra-1 que se va a reemplazar.
- Se pueden especificar cadenas de búsqueda de varios elementos mediante la opción ==pseudo-text-1==. Por ejemplo, para reemplazar todas las apariciones de UPON PRINTER, debe especificar ==UPON PRINTER==.
- La cadena de reemplazo, que sigue al BY, se puede especificar utilizando cualquiera de las cuatro opciones.
- Si la cadena de reemplazo es una frase de varios elementos o se debe eliminar por completo, debe utilizar la opción ==pseudo-text-2==. Si pseudo-text-2 es nulo (en otras palabras, el texto de reemplazo se especifica como ====), se eliminarán todas las apariciones encontradas de la cadena de búsqueda.

Cláusula de cadena

Con esta función, puede reemplazar secuencias de caracteres que aparecen al principio (consulte LEADING) o al final (consulte TRAILING) de palabras reservadas o definidas por el usuario. Por ejemplo, para cambiar todas las palabras de la forma "0100-xxxxxx" a "020-xxxxxx", codifique LEADING ==0100-== BY ==020-==. Para simplemente eliminar todos los prefijos "0100-" de las palabras, codifique LEADING ==0100-== BY ====.

3.3 REEMPLAZAR

Sintaxis de la instrucción CDF REPLACE (formato 1)

```
REEMPLAZAR [ TAMBIÉN ] { Cláusula-frase | Cláusula-cadena }...    .
~~~~~~~     ~~~~
```

Sintaxis de la instrucción CDF REPLACE (formato 2)

```
REEMPLAZAR [ ÚLTIMO ] APAGADO .
~~~~~~~     ~~~~     ~~~
```

Sintaxis de cláusula-frase REPLACE de CDF

```
{ ==pseudo-texto-1== } POR { ==pseudo-texto-2== }
                        ~~
```

Sintaxis de cláusula de cadena CDF REPLACE

```
[ PRINCIPAL|FINAL ] ==palabra parcial 1== POR ==palabra parcial 2==
  ~~~~~~~ ~~~~~~~~                        ~~
```

1. La declaración REPLACE proporciona un mecanismo para cambiar la totalidad o parte de una o más declaraciones GnuCOBOL.
2. Un punto es absolutamente obligatorio al final de cada declaración REPLACE (en cualquier formato), incluso si la declaración ocurre dentro del alcance de otra donde un punto podría parecer disruptivo (como dentro del alcance de un IF (verSección 7.8.23 [IF], página 338) declaración; sin embargo, el período no afectará el alcance de la declaración en la que se produce el REEMPLAZO.
3. Los siguientes puntos se aplican al formato 1 de la declaración REPLACE:
   A. El formato 1 de la declaración REPLACE se puede utilizar para realizar cambios en el código fuente del programa de la misma manera que la opción REPLACING de la declaración COPY, a través de estas opciones:

      Cláusula-frase
      Reemplazar una o más palabras reservadas completas, identificadores definidos por el usuario o literales; los siguientes puntos se aplican a esta opción:
      - Esta opción no se puede utilizar para reemplazar parte de una palabra, identificador o literal.
      - Todo lo que precede al BY se denominará aquí cadena de búsqueda.
      - Las cadenas de búsqueda en REPLACE siempre se especifican utilizando la opción ==pseudo-text-1==. Por ejemplo, para reemplazar

todas las apariciones de UPON PRINTER, debe especificar ==UPON PRINTER==.

- La cadena de reemplazo, que sigue al BY, se especifica utilizando el Opción ==pseudo-texto-2==. Si el pseudo-texto-2 es nulo (en otras palabras, el texto de reemplazo se especifica como ====), se eliminarán todas las apariciones encontradas de la cadena de búsqueda.

Cláusula de cadena

Con esta función, puede reemplazar secuencias de caracteres que aparecen al principio (consulte LEADING) o al final (consulte TRAILING) de palabras reservadas o definidas por el usuario. Por ejemplo, para cambiar todas las palabras del formato "0100-xxxxxx" a "020- xxxxxx", codifique LEADING ==0100-== BY ==020-==. Para simplemente eliminar todos los prefijos "0100-" de las palabras, codifique LEADING ==0100-== BY ====.

B. Una vez que se encuentra una declaración REPLACE de Formato 1 en el archivo fuente que se está compilando actualmente, el Modo Reemplazar se activa y los cambios especificados por esa declaración se realizarán automáticamente en todas las declaraciones fuente posteriores que el compilador lea del archivo.

C. El modo Reemplazar permanece vigente (y continúa realizando cambios en el código fuente) hasta que se encuentra otro REEMPLAZAMIENTO de Formato 1, se llega al final del archivo fuente del programa que se está compilando actualmente o se encuentra una declaración REEMPLAZAMIENTO de Formato 2.

D. Cuando se encuentra una sentencia REPLACE de formato 1 con la palabra clave ALSO sin que el modo de reemplazo esté activo en ese momento, el efecto será como si no se hubiera especificado ALSO. Si el modo de reemplazo ya estaba en vigencia, el efecto será "empujar" las especificaciones de cambio actuales a la parte superior de una pila y agregar las especificaciones de la nueva sentencia a las que ya estaban en vigencia.

E. Cuando se encuentra un REEMPLAZAMIENTO de Formato 1 sin la palabra clave ALSO, cualquier especificación de cambio apilada, si la hubiera, se descartará y las especificaciones de cambio actualmente en vigor, si las hubiera, se reemplazarán por las de la nueva declaración.

F. Cuando se llega al final del archivo fuente que se está compilando actualmente, el modo de reemplazo se desactiva y se descartan todas las especificaciones de reemplazo apiladas; la compilación del próximo archivo fuente (si hay alguna) comenzará con el modo de reemplazo inactivo y sin especificaciones de cambio en la pila.

4. Los siguientes puntos se aplican al formato 2 de la declaración REPLACE:

A. Si el modo Reemplazar está actualmente inactivo, se ignorará la declaración REPLACE de Formato 2.

B. Si el modo de reemplazo está activo actualmente, una REPLACE OFF. desactivará el modo de reemplazo y descartará cualquier especificación de reemplazo en la pila. El compilador funcionará de ahora en adelante como si nunca se hubiera encontrado una REPLACE, hasta que se encuentre otra REPLACE de formato 1.

C. Si el modo de reemplazo está activo actualmente, un REPLACE LAST OFF. reemplazará las especificaciones de reemplazo actuales con las que se extrajeron de la parte superior de la pila. Si no había especificaciones de reemplazo en la pila, el efecto será como si se hubiera codificado un REPLACE OFF.

3.4 >>DEFINIR

CDF>>Sintaxis de la directiva DEFINE

```
>>DEFINE [CONSTANTE] cdf-variable-1 COMO { OFF                    }
~~~~~~~~   ~~~~~~~~                     { ~~~                    }
                                        { literal-1 [ ANULAR ] }
                                        {             ~~~~~~~~   }
                                        { PARÁMETRO [ ANULAR ] }
                                          ~~~~~~~~~   ~~~~~~~~
```

Utilice la directiva >>DEFINE CDF para crear variables CDF y (opcionalmente) asignarles valores literales o de variables de entorno.

1. La palabra reservada AS es opcional y puede incluirse o no, a criterio del programador. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. CDFLas variables definidas de esta manera se vuelven indefinidas una vez que se ejecuta END PROGRAM o END FUNCTION.
   La directiva se encuentra en la fuente de entrada.
3. La directiva >>DEFINE CDF es una forma de crear variables CDF que pueden ser procesadas por otras instrucciones CDF como >>IF (verSección 3.5 [>>IF], página 68). La directiva >>SET CDF (verSección 3.6 [>>SET], página 70) proporciona otra forma de crearlos.
4. Los nombres de las variables CDF siguen las reglas de los nombres definidos por el usuario estándar de GnuCOBOL y no pueden duplicar ninguna palabra reservada de CDF. Los nombres de las variables CDF pueden duplicar palabras reservadas de COBOL, siempre que no se especifique la opción CONSTANT, pero no se recomiendan dichos nombres.
5. La opción CONSTANT es válida únicamente junto con el literal-1. Cuando se especifica CONSTANT, la variable CDF que se crea se puede utilizar dentro del código COBOL normal como si fuera un valor literal. Sin esta opción, la variable CDF solo se puede referenciar en otras instrucciones CDF. La opción OFF se utiliza para crear una variable sin asignarle ningún valor. Después de una directiva DEFINE en la que se especifica la frase OFF, no se debe utilizar el nombre de variable de compilación-1 excepto en una condición definida [donde se evalúa como no definida].
6. La opción PARAMETER se utiliza para crear una variable cuyo valor es el de la variable de entorno del mismo nombre. Tenga en cuenta que esta asignación de valor se produce en el momento de la compilación, no en el momento de la ejecución del programa.
7. En ausencia de la opción OVERRIDE, la variable cdf-1 no debe haberse definido aún. Cuando se especifica la opción OVERRIDE, se creará la variable cdf-1 con el valor especificado, si aún no se había definido. Si ya se había definido, se redefinirá con el nuevo valor.

3.5 >>SI

CDF>>Sintaxis de la directiva IF

```
>>IF CDF-Expresión condicional-1
~~~~       [ Líneas fuente del programa 1 ]

[ >>ELIF CDF-Expresión condicional-2
   ~~~~~~ [ Líneas-Fuente-del-Programa-2 ] ]...

[ >>DE LO CONTRARIO
   ~~~~~~ [ Líneas-fuente-del-programa-3 ] ]

>>FIN-SI
~~~~~~~~
```

Sintaxis de expresión condicional CDF

```
{ cdf-variable-1 } ES [ NO ] { DEFINIDA                    }
{  literal-1       }      ~~~  {  ~~~~~~~                  }
                               {  COLOCAR                  }
                               {  ~~~                      }
                               {  CDF-RelOp { variable-cdf-2  } }
                               {            { literal-2       } }
```

Sintaxis CDF-RelOp

```
>=    o    MAYOR    QUE  O IGUAL A
           QUE           ~~ ~~~~~
           ~~~~~~~
>     o    MAYOR    QUE
           QUE
           ~~~~~~~
<=    o   MENO  QUE O     IGUAL A
             S
           ~~~~         ~~  ~~~~~
<     o   MENO  QUE
             S
           ~~~~
```

La directiva CDF >>IF hace que el compilador GnuCOBOL procese o ignore declaraciones de origen COBOL, declaraciones de manipulación de texto CDF y directivas CDF dependiendo del valor de una o más expresiones condicionales basadas en variables CDF.

1. Las palabras reservadas IS, THAN y TO son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.

2. Cada directiva >>IF debe terminar con una directiva >>END-IF.
3. Puede haber cualquier número de cláusulas >>ELIF después de un >>IF, incluido cero.
4. No puede haber más de una cláusula >>ELSE después de una >>IF. Cuando se utiliza >>ELSE, debe seguir a la >>IF y a todas las cláusulas >>ELIF.
5. El compilador solo puede procesar una de las instrucciones del bloque Program-Source-Lines-n que se encuentran dentro del alcance de >>IF . . . >>END-IF. Se decidirá cuál de ellas (si es que hay alguna) se procesará de la siguiente manera:
   A. Cada expresión condicional CDF-n se evaluará, a su vez, en la secuencia en la que se codifica en la declaración >>IF y en cualquier cláusula >>ELIF que pueda estar presente hasta que una de ellas se evalúe como VERDADERA. Una vez que una de ellas se evalúe como VERDADERA, el bloque de código Program-Source-Lines-n que corresponde a la expresión condicional CDF-n VERDADERA será el que se procese. Todos los demás dentro del ámbito >>IF->>END-IF se ignorarán.
   B. Si ninguna CDF-Conditional-Expression evalúa como VERDADERO, y hay una cláusula >>ELSE, el bloque de declaraciones Program-Source-Lines-3 que sigue a la cláusula >>ELSE será procesado por el compilador y todos los demás dentro del ámbito >>IF->>END-IF serán ignorados.
   C. Si ninguna CDF-Conditional-Expression-n evalúa como VERDADERO y no hay ninguna cláusula >>ELSE, entonces el compilador no procesará ninguno de los bloques de instrucciones Program-Source-Lines-n dentro del ámbito >>IF->>END-IF.
   D. Si el bloque de instrucción Program-Source-Lines-n> seleccionado para procesamiento está vacío, no se produce ningún error; simplemente no se generará ningún código desde la estructura >>IF->>END-IF.
6. Un bloque Program-Source-Lines-n puede contener cualquier código COBOL o CDF válido.
7. Los siguientes puntos se aplican a cualquier CDF-Conditional-Expression-n:
   A. La opción DEFINED prueba si la variable cdf-1 ha sido definida, pero aún no se le ha asignado un valor (>>DEFINE ... OFF); use la opción NOT para probar si la variable no está definida.
   B. La opción SET prueba si a la variable cdf-1 se le ha asignado un valor, ya sea a través de un >>Declaración SET o mediante >>DEFINE sin la opción OFF.
   C. Se pueden comparar entre sí dos variables CDF, dos literales o una única variable CDF y un único literal mediante un operador relacional. A diferencia de la declaración IF estándar de Gnu-COBOL (consulteSección 7.8.23 [IF], página 338), no se pueden realizar comparaciones múltiples con AND u OR juntas; puede anidar un segundo >>IF dentro del primero, sin embargo, para simular un AND y un OR se puede simular mediante la opción >>ELIF.
   D. El símbolo <> significa NO IGUAL A.

3.6 >>CONFIGURAR

CDF>>Sintaxis de la directiva SET

```
>>SET { [ CONSTANTE ] cdf-variable-1 literal-1 ]                                  }
~~~~~ {      ~~~~~~~~~                                                            }
      { FORMATO DE ORIGEN FIJO | LIBRE | VARIABLE | XOPEN | XCARD | CRT | TERMINAL | COBOLX }
          { ~~~~~~~~~~~~~~  ~~~~~ ~~~~ ~~~~~~~~ ~~~~~ ~~~~~ ~~~ ~~~~~~~~ ~~~~~~ }
        { NOFOLDCOPYNAME                                                           }
        { ~~~~~~~~~~~~~~~                                                          }
        { FOLDCOPYNAME COMO SUPERIOR|MINUSCULAR                                    }
           ~~~~~~~~~~~~       ~~~~~ ~~~~~
```

La directiva CDF >>SET proporciona un medio alternativo para realizar las acciones de las directivas >>DEFINE y >>SOURCE, así como un medio para controlar los interruptores -free, -fixed y -ffold-copy del compilador desde dentro del código fuente del programa.

1. La palabra reservada AS es opcional (sólo en las cláusulas SOURCEFORMAT y FOLDCOPYNAME) y puede incluirse o no, a discreción del programador. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. CDFLas variables definidas de esta manera se vuelven indefinidas una vez que se ejecuta END PROGRAM o END FUNCTION.
   La directiva se encuentra en la fuente de entrada.
3. La opción FOLDCOPYNAME proporciona el equivalente a especificar el modificador -ffold-copy=xxx del compilador, donde xxx es UPPER o LOWER.
4. ElLa opción NOFOLDCOPYNAME desactiva el efecto de la instrucción >>SET FOLDCOPYNAME o del modificador -ffold-copy=xxx del compilador.
5. Si se utiliza la opción CONSTANT, también se debe utilizar el literal 1. Esta opción proporciona otro medio para definir constantes que se pueden utilizar en cualquier parte del programa en la que se pueda especificar un literal.
6. Las opciones restantes de la directiva CDF >>SET proporcionan una funcionalidad equivalente a la >>DEFINE [sin embargo, como esta forma ya no está en los estándares de Cobol y está clasificada como arcaica y será marcada como tal, se debe utilizar >>DEFINE para todos los programas nuevos], y las directivas >>SOURCE, de la siguiente manera:

   >>SET cdf-variable-1
   >>DEFINE cdf-variable-1 COMO DESACTIVADA

   >>SET cdf-variable-1 COMO literal-1
   >>DEFINE cdf-variable-1 COMO literal-1

   >>ESTABLECER CONSTANTE cdf-variable-1 literal-1
   >>DEFINIR CONSTANTE cdf-variable-1 literal-1

   >> ESTABLECER FORMATO DE FUENTE COMO FIJO
   >>EL FORMATO DE LA FUENTE ES FIJO

   >> ESTABLECER FORMATO DE FUENTE COMO LIBRE
   >>EL FORMATO FUENTE ES GRATUITO

   >> ESTABLECER FORMATO DE FUENTE COMO VARIABLE
   >>EL FORMATO DE LA FUENTE ES VARIABLE

>> ESTABLECER FORMATO DE ORIGEN COMO XOPEN
>>EL FORMATO DE LA FUENTE ES XOPEN

>>ESTABLECER FORMATO DE FUENTE COMO XCARD
>>EL FORMATO FUENTE ES XCARD

>>ESTABLECER FORMATO DE FUENTE COMO CRT
>>EL FORMATO DE LA FUENTE ES CRT

>> ESTABLECER FORMATO DE FUENTE COMO TERMINAL
>>EL FORMATO DE LA FUENTE ES TERMINAL

>> ESTABLECER FORMATO DE ORIGEN COMO COBOLX
>>EL FORMATO FUENTE ES COBOLX

>>SET XFD literal-1
[Ver capítulo 13]

>>Directiva de microenfoque SET
[hacer]

3.7 >>FUENTE

CDF>>Sintaxis de la directiva SOURCE

>>EL FORMATO DE ORIGEN ES { FIJO | LIBRE | VARIABLE | XOPEN | XCARD | CRT | TERMINAL | COBOLX }
~~~~~~~~ ~~~~~ ~~~~ ~~~~~~~~ ~~~~~ ~~~~~ ~~~ ~~~~~~~~ ~~~~~~

———————————————————————————————————————

La directiva >>SOURCE CDF pone al compilador en modo de formato de código fuente FIJO o LIBRE. Esto, en efecto, proporciona otro mecanismo para controlar el cambio a libre del compilador y -interruptor fijo.

1. Las palabras reservadas FORMAT y IS son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. ARREGLADO: El código fuente está dividido en: columnas 1 a 6, el área del número de secuencia; columna 7, el área del indicador; columnas 8 a 72, el área de texto del programa; y columnas 72 a 80, el área de referencia.
3. GRATIS: El área de texto del código fuente comienza en la columna 1 y continúa hasta el final de la línea (hasta un máximo de 255 caracteres).
4. VARIABLE: Idéntico al formato FIJO anterior, excepto que se extiende hasta la columna 256 (en MF y algunos otros, es 252).
5. XOPEN: formato libre X/Open. El área de texto del programa puede comenzar en la columna 1 a menos que haya un indicador presente, y las líneas pueden contener hasta 80 caracteres. El indicador para depurar líneas es D en lugar de D o d.
6. XCARD: formato xCard de ICOBOL. Formato variable con margen derecho establecido en la columna 255 en lugar de 250.
7. CRT: formato libre ICOBOL (CRT). Similar al formato X/Open mencionado anteriormente, con líneas que contienen hasta 320 caracteres e indicadores de línea de depuración de un solo carácter (D o d).
8. TERMINAL: Formato de terminal ACUCOBOL-GT. Similar al formato CRT anterior, con el indicador para depurar líneas como \D en lugar de D o d. Este formato es compatible en su mayor parte con el formato de fuente de terminal VAX COBOL.
9. COBOLX: Este formato es similar al formato CRT mencionado anteriormente, excepto que el área de indicadores siempre está presente en la columna 1; el área de texto del programa comienza en la columna 2 y se extiende hasta el final del registro. Las líneas pueden contener hasta 255 caracteres.
10. Tenga en cuenta que con los formatos de origen XOPEN, CRT, TERMINAL y COBOLX, faltan espacios.No se insertan dentro de literales alfanuméricos continuos que se truncan antes del margen derecho.
11. Puede cambiar entre los distintos modos de formato según lo desee y el efecto tendrá efecto inmediatamente.
12. También puede utilizar la directiva >>SET CDF para realizar esta función.
13. Si el compilador ya está en el modo especificado, esta declaración no tendrá efecto.
~~~~~~~~

3.8 >>GIRAR

CDF>>Sintaxis de la directiva TURN

```
>>TURN { nombre-excepción-1 [ nombre-archivo-1 ]...        }...
~~~~~~
    { APAGADO                          }
    { ~~~                              }
{ COMPROBANDO [ CON UBICACIÓN ] }
      ~~~~~~~~ ~~            ~~~~~~~~
```

La directiva (des)activará las comprobaciones de excepciones.

3.9 >>El

CDF>>Sintaxis de la directiva D

>>El
~~~

——

La directiva elimina todas las líneas de depuración flotantes si el modo de depuración no está activo. De lo contrario, ignorará la parte de la línea que contiene la directiva.
~~~

3.10 >>PANTALLA

CDF>>Sintaxis de la directiva DISPLAY

>>MOSTRAR texto fuente [VCS = cadena de versión]

La directiva es una extensión v1.0 y mostrará mensajes durante la compilación.

3.11 >>PAGINA

CDF>>Sintaxis de la directiva PAGE

>>PAGINA [texto del comentario]
~~~~~~

———————————————————————————————————————

La directiva PAGE especifica la expulsión de la página y proporciona documentación para el listado de fuentes.

1. Comentario: El texto puede contener cualquier carácter del conjunto de caracteres codificados estándar de la computadora, excepto los caracteres de control.
2. El texto del comentario no se verifica en cuanto a su sintaxis y solo sirve como documentación.
3. Si se está produciendo un listado de fuentes, una directiva PAGE provocará la expulsión de la página seguida delistado de la directiva PAGE.
4. Si no se elabora un listado la directiva no tendrá efecto.
~~~~~~

3.12 >>LISTADO

CDF>>Sintaxis de la directiva LISTING

```
>>LISTADO   {EN}
~~~~~~~~~   {APAGADO}
```

La directiva permite des-(activar) el listado de programas.

3.13 >>SEGUNDOS INTERCIPALES

CDF>>Sintaxis de la directiva LEAP-SECONDS

>>SEGUNDOS INTERCIPALES
~~~~~~~~~~~~~~

La directiva CDF >>LEAP-SECONDS se reconoce sintácticamente pero por lo demás no es funcional.

——

Permite más de 60 segundos por minuto.
~~~~~~~~~~~~~~

3.14 $ Directivas

Sintaxis de la directiva CDF $

Directivas $ (Dólar) - Activas.

Estas directivas están activas y tienen la misma función que las que comienzan con >>:

$DEFINIR
$PANTALLA ACTIVADA|DESACTIVADA
$SI
$ELIF
$DE LO CONTRARIO
$ELSE-SI
$FIN
$CONFIGURAR

Se recomienda utilizar únicamente las directivas estándar en lugar de las directivas MF (cuando sea posible), ya que tienen mayores posibilidades de ser portables.

Directivas $ (Dólar) - No activas.
Estos NO están activos y producirán un mensaje de advertencia:

$MOSTRAR VCS...

Reconocido pero por lo demás

ignorado. @OPTIONS opciones-texto

Se aceptan directivas adicionales de Micro-Focus:

ADDRSV | ADD-RSV literal-1
ADDSYN | ADD-SYN literal-1 = literal-2
ASIGNAR "EXTERNO" | "DINÁMICO"
ATADO
Llamar a literal-1
COMP1 | COMP-1 "BINARIO" | "FLOTADOR"
COPIANOMBREPLEGAR |DOBLAR-COPIAR-NOMBRE "SUPERIOR"
| "INFERIOR" MAKESYN | HACER-SYN
SIN LÍMITES | NO-ATADO
NOPLEGABLECOPIANO | NOFOLD-COPIA-NOMBRE | NOMBRE DE COPIA SIN DOBLAR
ANULAR literal-1 = literal-2
ELIMINAR literal-1
FORMATO FUENTE | FORMATO FUENTE "FIJO" | "LIBRE" | "VARIABLE"
RANGO SS "2"
NOSOTROSRANGO | SIN RANGO

———————————————————————————————————————

Ofrece soporte para las directivas del compilador MF.

————————————————————————

Fin del Capítulo 3 — CDF - Compiler Directing Facility

4 DIVISIÓN DE IDENTIFICACIÓN

DIVISIÓN DE IDENTIFICACIÓN Sintaxis

```
[ { IDENTIFICACIÓN } DIVISIÓN. ]
[ { ~~             } ~~~~~~~~~   ]

{ PROGRAMA-ID.  } { nombre del programa     } .
{ ~~~~~~~~~~~~~ } { literal-1               } [ AS { literal-2 } ] [
Cláusula de tipo ].
{ FUNCIÓN-ID. } { literal-3      } [ COMO          literal-4    ] .
  ~~~~~~~~~~~~   { nombre-de-función } .

[ { OPCIONES. }                                                  ]
[     ~~~~~~~                                                    ]
[ [ LA ARITMÉTICA ES NATIVA. ]                                   ]
[ [ ~~~~~~~~~~~~   ~~~~~    ]                                   ]
[                                                                ]
[ [EL MODO REDONDEADO PREDETERMINADO ES {LEJOS DE CERO} } ] ]
[ [ ~~~~~~~~ ~~~~~~~~        {MÁS CERCANO-A-CERO} ] ] [ [
                             {MÁS CERCANO-INCLUSO     } ] ]
[ [                          {MÁS CERCANO A CERO      } ] ]
[ [                          {PROHIBIDO               } ] ]
[ [                          {HACIA MAYOR             } ] ]
[ [                          {HACIA-MENOR             } ] ]
[ [                          {TRUNCAMIENTO            }.]   ]
[                                                                ]
[ [ LA CONVENCIÓN DE ENTRADA ES {COBOL                           } ] ]
[ [ ~~~~~~~~~~~~~~~~~~                                           {EXTERNO} ]      ]
[ [                          {LLAMADA ESTÁNDAR}.]                                  ]

[ AUTOR.            [   comentario-     ]...]
  ~~~~~~              entrada-1.
[ FECHA DE          [   comentario-     ]...]
  COMPILACIÓN.        entrada-2.
  ~~~~~~~~~~~~~~
[ FECHA DE          [   comentario-     ]...]
  MODIFICACIÓN.       entrada-3.
  ~~~~~~~~~~~~~~
[ FECHA DE          [   comentario-     ]...]
  ESCRITURA.          entrada-4.
  ~~~~~~~~~~~~~
[ INSTALACIÓN.      [   comentario-     ]...]
                      entrada-5.
  ~~~~~~~~~~~~~
[ OBSERVACIONES.    [   comentario-     ]...]
                      entrada-6.
  ~~~~~~~~
[ SEGURIDAD.        [   comentario-     ]...]
                      entrada-7.
  ~~~~~~~~~
```

EL AUTOR, FECHA DE COMPILACIÓN, FECHA DE MODIFICACIÓN, FECHA DE ESCRITURA, INSTALACIÓN,

OBSERVACIONES

Los párrafos SECURITY y SECURITY son compatibles con GnuCOBOL únicamente para brindar compatibilidad con programas escritos para los estándares ANS1974 (o anteriores). A partir del estándar ANS1985, estas cláusulas se han vuelto obsoletas y no se deben utilizar en programas nuevos.

———————————————————————————————————————

Sintaxis de la cláusula de tipo PROGRAM-ID

ES [COMÚN] [PROGRAMA INICIAL | RECURSIVO]
~~~~~~ ~~~~~~~ ~~~~~~~~~

La división de identificación proporciona una identificación básica del programa dándole un nombre y definiendo opcionalmente algunas características de alto nivel a través de los ocho párrafos predefinidos que se pueden especificar.

1. Los párrafos que se muestran arriba pueden codificarse en cualquier secuencia.
2. Las palabras reservadas AS, IS y PROGRAM son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
3. Una cláusula de tipo puede codificarse solo cuando se especifica PROGRAM-ID. Si se codifica una, COMÚN, COMÚN INICIAL o COMÚN RECURSIVO debe serespecificado.
4. Si bien el encabezado DIVISIÓN DE IDENTIFICACIÓN o DIVISIÓN DE ID real es opcional, los párrafos ID-PROGRAMA / ID-FUNCIÓN no lo son; sin embargo, solo se puede codificar uno u otro.
5. En la definición del nombre de PROGRAM-ID (o FUNCTION-ID) no debe utilizar nombres idénticos a los de las funciones de la biblioteca 'libc'. Las funciones de la biblioteca 'libc' tienen nombres en minúsculas o en mayúsculas y minúsculas. El uso de nombres siempre en mayúsculas, para PROGRAM-ID y FUNCTION-ID evita cualquier posibilidad de superposición u otro uso incorrecto. Del mismo modo, el uso de nombres que comiencen con 'CBL' o 'COB' puede entrar en conflicto con las rutinas de función predefinidas y, nuevamente, podría terminar con la rutina incorrecta que se está llamando y, por último, no anteponga el nombre con un guión bajo ( ) ya que estos se utilizan con el uso de rutinas internas. Tenga en cuenta que esto también es cierto para cualquier biblioteca C en el espacio del programa, por lo que incluye curses, gmp, posiblemente libm, libxml2, JSON, BDB, ...
6. El parámetro -Wobsolete del compilador hará que el compilador GnuCOBOL emita mensajes de advertencia si se utilizan estas (o cualquier otra sintaxis obsoleta) en un programa.
7. Si se especifica, literal-1 debe ser un literal alfanumérico real y no puede ser una constante figurativa.
8. Los párrafos PROGRAM-ID y FUNCTION-ID sirven para identificar el programa en el entorno externo (es decir, el sistema operativo). Si no hay una cláusula AS presente, el identificador del programa servirá como esa identificación externa. Si hay una cláusula AS especificada, ese literal especificado servirá como identificación externa. En el resto de este documento, esa "identificación externa" se denominará el nombre del punto de entrada principal.
9. El INICIAL,Las palabras COMUNES y RECURSIVAS se utilizan únicamente dentro de subprogramas que funcionan como subrutinas. Sus propósitos son los siguientes:
   A. COMMON debe usarse únicamente dentro de subprogramas que sean subprogramas anidados. AUn subprograma anidado declarado como COMÚN puede ser llamado desde cualquier programa anidado en el archivo fuente que se está compilando, no solo desde aquellos "encima" de él en la estructura de anidación.
   B. La cláusula RECURSIVE, si existe, hará que el compilador genere un código objeto diferente para el subprograma que le permitirá invocarse a sí mismo y regresar correctamente al programa que lo invocó.

   Las funciones definidas por el usuario (es decir, FUNCTION-ID) siempre son recursivas.
~~~~~~

C. La cláusula INITIAL, si se especifica, garantiza que el subprograma estará en su estado inicial (es decir, compilado) cada vez que se ejecute, no solo la primera vez.

———————————————————————

Fin del Capítulo 4 — DIVISIÓN DE IDENTIFICACIÓN

5 DIVISIÓN DE MEDIO AMBIENTE

DIVISIÓN DE MEDIO AMBIENTE Sintaxis

```
DIVISIÓN DE MEDIO AMBIENTE.
[SECCIÓN DE CONFIGURACIÓN.]
[ORDENADOR FUENTE.        Compilación-Especificación-Computadora.        ]
[ COMPUTADORA-OBJETO.     Especificación de computadora de ejecución.    ]
[ NOMBRES ESPECIALES.     Especificación de configuración del programa.  ]
[ REPOSITORIO.            Prototipo-Especificación ..............]
[SECCIÓN DE ENTRADA-SALIDA.]
[ CONTROL DE ARCHIVOS.    Descripción general del archivo ....]
[ CONTROL IO.             Especificación de almacenamiento en búfer de archivos  ]
```

Esta división define el entorno informático externo en el que se ejecutará el programa. Esto incluye la definición de los archivos que el programa puede tener.

- Si ambas secciones opcionales de esta división están codificadas, deben codificarse en la secuencia que se muestra.
- Los párrafos dentro de las secciones pueden codificarse en cualquier orden.
- Estas secciones consisten en una serie de párrafos específicos y predefinidos (ORIGEN-COMPUTADORA y OBJETO-COMPUTADORA, por ejemplo), cada uno de los cuales cumple una función específica. Si no se requiere código para la función de uno de los párrafos, se puede omitir el párrafo completo.
- Si alguno de los párrafos dentro de una de las secciones está codificado, el encabezado de la sección en sí debe estar codificado.
- Si ninguno de los párrafos dentro de una de las secciones está codificado, se puede omitir el encabezado de sección.
- Si ninguna de las secciones dentro de la división de medio ambiente está codificada, el encabezado DIVISIÓN DE MEDIO AMBIENTE puede omitirse.

5.1 SECCIÓN DE CONFIGURACIÓN

SECCIÓN DE CONFIGURACIÓN Sintaxis

```
SECCIÓN DE CONFIGURACIÓN.
~~~~~~~~~~~~~~ ~~~~~~~
[ ORDENADOR-FUENTE. Compilación-Ordenador-Especificación .    ]
~~~~~~~~~~~~~~~~
[ COMPUTADORA-OBJETO. Especificación-Computadora-Ejecución .        ]
~~~~~~~~~~~~~~~~
[ NOMBRES ESPECIALES.                                    Especificación de configuración
del programa.        ]
~~~~~~~~~~~~~~
[ REPOSITORIO.    Prototipo-Especificación..............  ]
~~~~~~~~~~~
```

——

Esta sección define el sistema informático en el que se compila y ejecuta el programa y también especifica cualquier configuración ambiental especial o características de compatibilidad.

1. Los cuatro párrafos de esta sección pueden especificarse en cualquier orden, pero si no es en este orden, se emitirá una advertencia.
2. La sección de configuración no está permitida en un subprograma anidado. Un programa anidado hereda los valores de la sección de configuración de su programa principal.
3. Si un programa no requiere ninguna de las características proporcionadas por la sección de configuración, se puede omitir todo el encabezado SECCIÓN DE CONFIGURACIÓN del programa.

5.1.1 ORDENADOR FUENTE

Sintaxis de ORDENADOR FUENTE

```
ORDENADOR-FUENTE. nombre-ordenador [ CON MODO DE DEPURACIÓN ].
~~~~~~~~~~~~~~~~                     ~~~~~~~~~~ ~~~~
```

Este párrafo define la computadora en la que se compila el programa y proporciona una forma en la que se puede activar el código de depuración integrado en el programa.

1. La palabra reservada WITH es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Este párrafo no está permitido en un subprograma anidado. Un programa anidado hereda el Configuración de la COMPUTADORA FUENTE de su programa padre.
3. El valor especificado para computer-name es irrelevante, siempre que sea una palabra COBOL válida que no coincida con ninguna palabra reservada de GnuCOBOL. El valor computer-name puede incluir espacios. No es necesario que coincida con el nombre de computadora utilizado con el párrafo OBJECT-COMPUTER, si lo hubiera.
4. La cláusula DEBUGGING MODE, si está presente, informará al compilador que las líneas de depuración (aquellas con una 'D' en la columna 7 si el Modo de fuente fija está vigente, o aquellas con el prefijo >>D si el Modo de fuente libre está vigente) —normalmente tratadas como comentarios— deben compilarse.
5. Incluso sin la cláusula DEBUGGING MODE, todavía es posible compilar líneas de depuración. Las líneas de depuración también se pueden compilar especificando el modificador -fdebugging-line en el compilador GnuCOBOL.

5.1.2 COMPUTADORA OBJETO

Sintaxis de OBJETO-COMPUTADORA

```
COMPUTADORA-OBJETO.   [ nombre-de-computadora ]
~~~~~~~~~~~~~~~
[EL TAMAÑO DE LA MEMORIA ES entero-1 PALABRAS|CARACTERES ]
  ~~~~~~ ~~~~                  ~~~~~ ~~~~~~~~~~
[ LA SECUENCIA DE COMPARACIÓN DEL PROGRAMA ES nombre-alfabeto-1 ]
          ~~~~~~~~~
[ SEGMENTO-LIMITE ES entero-2 ]
  ~~~~~~~~~~~~~
[ LA CLASIFICACIÓN DE CARACTERES ES { nombre-local-1       } ]
                ~~~~~~~~~~~~~~~   { LOCAL              }
                                  { ~~~~~~             }
                            { USUARIO PREDETERMINADO         }
                                { ~~~~~~~~~~~~~~  }
                              { SISTEMA PREDETERMINADO }
                                   ~~~~~~~~~~~~~~
```

Las cláusulas MEMORY SIZE y SEGMENT-LIMIT se reconocen sintácticamente pero por lo demás no son funcionales.

———————————————————————————————————————

Este párrafo describe la computadora en la que se ejecutará el programa.

1. El nombre de la computadora, si se especifica, debe seguir inmediatamente al nombre del párrafo OBJECT-COMPUTER. Las cláusulas restantes pueden codificarse en cualquier secuencia.
2. Las palabras reservadas CHARACTER, IS, PROGRAM y SEQUENCE son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
3. El valor especificado para el nombre de la computadora, si lo hay, es irrelevante siempre que sea una palabra COBOL válida que no coincida con ninguna palabra reservada de GnuCOBOL. El nombre de la computadora puede incluir espacios. No es necesario que coincida con el nombre de la computadora utilizado con el párrafo SOURCE-COMPUTER, si lo hay.
4. El párrafo OBJECT-COMPUTER no está permitido en un subprograma anidado. Un programa anidado hereda la configuración OBJECT-COMPUTER de su programa principal.
5. La cláusula COLLATING SEQUENCE le permite especificar una secuencia de ordenación de caracteres personalizada que se utilizará cuando se comparen valores alfanuméricos entre sí. Los datos se seguirán almacenando en el conjunto de caracteres nativo de la computadora, pero la secuencia lógica en la que se ordenan los caracteres para fines de comparación se puede modificar con respecto a la definida por el conjunto de caracteres nativo de la computadora. El nombre-alfabeto-1 que especifique debe definirse en SPECIAL-NAMES (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90) párrafo.
6. Si no se especifica ninguna cláusula COLLATING SEQUENCE, se utilizará la secuencia de clasificación implícita en el conjunto de caracteres nativo de la computadora (normalmente ASCII).
7. La cláusula opcional CLASSIFICATION se puede utilizar para especificar una configuración regional para el entorno en el que se ejecutará el programa, con el fin de influir en las asignaciones de mayúsculas y minúsculas de caracteres para MAYÚSCULAS

(consulteSección 8.1.101 [MAYÚSCULAS], página 533) y MINÚSCULAS (verSección 8.1.51 [MINÚSCULAS], página 481) funciones intrínsecas y la clasificación de caracteres para el ALFABETICO, ALFABETICO-MINUSCULO y ALFABETICO-MAYUSCULO

Pruebas de clase. Las definiciones de estas clases se toman de la especificación de la convención cultural (LC_CTYPE) de la configuración regional especificada.

Los significados de las cuatro especificaciones regionales son los siguientes:

A. *nombre-local-1*hace referencia a una CONFIGURACIÓN LOCAL (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) definición.
B. La palabra clave LOCALE se refiere a la configuración regional actual (vigente en el momento en que se ejecuta el programa).
C. La palabra clave USER-DEFAULT hace referencia a la configuración regional predeterminada especificada para el usuario que actualmente ejecuta este programa.
D. La palabra clave SYSTEM-DEFAULT denota la configuración regional predeterminada especificada para la computadora en la que se ejecuta el programa.

8. La ausencia de una cláusula CLASSIFICATION hará que la clasificación de caracteres se realice de acuerdo con las reglas del conjunto de caracteres nativos de la computadora (ASCII, EBCDIC, etc.).

5.1.3 NOMBRES ESPECIALES

Sintaxis de NOMBRES ESPECIALES

```
NOMBRES ESPECIALES.
~~~~~~~~~~~~~
  [ CONVENCIÓN DE LLAMADA entero-1 ES nombre-mnemónico-1 ]
     ~~~~~~~~~~~~~~~~
  [ LA CONSOLA ES
  CRT ]
     ~~~~~~~        ~~~
  [ EL ESTADO DEL CRT ES identificador-1 ]
     ~~~ ~~~~~~
  [ EL SIGNO MONEDA ES literal-1 ]
     ~~~~~~~~~ ~~~~
  [ CURSOR ES identificador-2 ]
     ~~~~~~
  [EL PUNTO DECIMAL ES COMA ]
     ~~~~~~~~~~~~~~        ~~~~~
  [ EL ESTADO DEL EVENTO ES identificador-3 ]
     ~~~~~ ~~~~~~
  [ LOCALE nombre-local-1 ES literal-2 ]...
     ~~~~~~
  [ EL SIGNO NUMÉRICO ESTÁ SEPARADO AL FINAL ]
     ~~~~~~~~ ~~~~        ~~~~~~~~
~~~~~~~~~~ [ EL CONTROL DE PANTALLA ES
  identificador-4 ]
     ~~~~~~ ~~~~~~~~
  [ nombre-dispositivo-1 ES nombre-

  mnemónico-2 ]... [ nombre-

  característica-1 ES nombre-

  mnemónico-3 ]... [ Cláusula-

  alfabética ]...

  [ Cláusula de definición

  de clase ]... [ Cláusula

  de definición de

  cambio ]...

  [ Cláusula de caracteres simbólicos ]...
  .
```

Alfabeto-Nombre-Cláusula, Cláusula de definición de clase,
*Cláusula de definición de conmutación*y cláusula de caracteres simbólicos

Se discuten en detalle en las siguientes cuatro secciones.

El párrafo NOMBRES ESPECIALES proporciona un medio para especificar varias opciones de configuración del programa y del entorno operativo.

1. Las distintas cláusulas que pueden especificarse dentro del párrafo NOMBRES ESPECIALES pueden codificarse en cualquier orden.
2. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.

3. El párrafo SPECIAL-NAMES no está permitido en un subprograma anidado. Un programa anidado hereda la configuración SPECIAL-NAMES de su programa principal.
4. Sólo la cláusula final especificada dentro de este párrafo deberá terminar con un punto.
5. La cláusula CALL-CONVENTION permite que un entero decimal, que representa una serie de configuraciones de interruptores ON/OFF, se asocie con un nombre mnemónico que luego puede codificarse en una declaración CALL (verSección 7.8.5 [LLAMADA], página 290). Las configuraciones de interruptor definidas por este mnemónico controlarán entonces cómo se manejará el enlace a una subrutina invocada por la declaración CALL que hace referencia al mnemónico-nombre-1.
6. La cláusula CONSOLE IS CRT, si se especifica, hará que una declaración DISPLAY que carezca de una cláusula UPON explícita se trate como una declaración de elemento de datos DISPLAY (consulteSección 7.8.12.4 [DIS- [PLAY elemento de datos], página 305), y cualquier declaración ACCEPT que carezca de una cláusula FROM se tratará como una declaración de elemento de datos ACCEPT (consulteSección 7.8.1.4 [ACEPTAR elemento de datos], página 267).
7. Si no se especifica la cláusula CRT STATUS, se asignará un identificador COB-CRT-STATUS implícito (con PICTURE 9(4)) con el fin de recibir estados ACCEPT de pantalla. Si se especifica CRT STATUS, entonces el identificador-1 debe definirse en el programa como un campo PICTURE 9(4).
8. La cláusula CURRENCY SIGN se puede utilizar para redefinir el carácter que se utilizará como signo de moneda en una IMAGEN (verSección 6.9.37 [IMAGEN], página 195) cláusula. El signo monetario predeterminado es el de dólar ('$'). Puede especificar cualquier carácter excepto 0-9, AZ, az, +, -, ,, ., *,
   /, ;, (,), =, \\, comillas ('"') o espacio.
9. La cláusula CURSOR IS permite especificar un elemento de datos de 4 o 6 caracteres en el que se encuentra la posición del cursor en la pantalla en el momento en que se satisface una instrucción ACCEPT en pantalla. El valor se devolverá como rrcc o rrrccc, según la longitud del identificador-2 especificado, donde rr y rrr representan el número de fila (que comienza en cero) y cc y ccc representan el número de columna (que también comienza en cero). No hay ningún elemento de datos predeterminado asignado para estos datos si no se especifica la cláusula CURSOR IS, y es responsabilidad del programador definir el identificador-2 si se especifica la cláusula. Si el identificador-2 se refiere a un elemento de datos que contiene una posición de pantalla válida al comienzo de una instrucción ACCEPT, y esa posición corresponde a un campo de entrada, esa posición se utiliza como la posición inicial del cursor. Esta posición puede estar al comienzo de un campo de entrada o en algún desplazamiento dentro del campo de entrada.
10. La cláusula DECIMAL POINT IS COMMA invierte la definición de los caracteres , y . cuando se utilizan como símbolos de edición de IMÁGENES y dentro de literales numéricos. Esto puede tener efectos secundarios no deseados: consulteSección 2.1.20 [Puntuación], página 35.
11. La cláusula LOCALE se puede utilizar para asociar nombres de configuración regional definidos por el sistema operativo externo (literal-2) con un nombre interno (nombre-regional-1) al que se puede hacer referencia dentro del programa. Los nombres de configuración regional están definidos por el sistema operativo o el compilador de C que GnuCOBOL utilizará en su computadora.
12. La siguiente es la lista de posibles códigos de configuración regional, por ejemplo, que estarían disponibles en una computadora Windows que ejecute una versión de GnuCOBOL creada utilizando el emulador Unix MinGW y el compilador GNU C (gcc):

    A de ZA, ET, AE, BH, DZ, EG, IQ, JO, KW, LB, LY, MA, OM, QA, SA, SY, TN, YE, CL, IN, Circa AZ, Latín AZ

B ba_R, be_BY, bg_BG, bn_IN bo_BT, bo_CN, br_FR, bs_Cyrl_BA, bs_Latn_BA

C ca_ES, cs_CZ, cy_GB

D de_DK, de_AT, de_CH, de_DE, de_LI, de_LU, dsb_DE, dv_MV

E el_GR, en 029, en AU, en BZ, en CA, en GB, en IE, en IN, en JM, en MY en NZ, en_PH, en_SG, en TT, en US, en ZA, en ZW, es AR, es BO, es CL, es CO, es CR, es_DO, es EC, es ES, es GT, es HN, es MX, es NI, es PA, es_PE, es PR, es PY, es SV, es US, es UY es VE, et EE, eu ES

F para IR, para FI, para fil PH, para FO, para BE, para CA, para CH, para FR, para LU, para MC, para NL

G ga_IE, gbz AF, gl ES, gsw FR, gu IN

H ha_Latn NG, he IL, hi IN, hr BA, hr HR, hu HU, hy AM

I id_ID, ig NG, ii CN, es IS, it CH, it IT, iu Cans CA, iu Latn CA

J Yo JP

K ka_GE, kh_KH, kk KZ, kl GL, kn IN, ko KR, kok IN, ky KG

L lb_LU, lo LA, lt LT, lv LV

M mi_NZ, mk MK, ml IN, mn Cyrl_MN, mn Mong_CN moh CA, sr IN, ms BN, ms_MY, mt_MT

N nb_NO, ne NP, nl BE, nl NL, nn NO, ns ZA

O oc_FR, o IN

P pa_EN, pl_PL, ps AF, pt BR, pt PT

Q qut_GT, quz_BO, quz_EC, quz PE

R rm_CH, ro RO, ru RU, rw RW

S sa_IN, sah_RU, se FI, se NO se SE, si LK, sk SK, sl SI, sma NO, sma SE, smj NO, smj SE, smn FI, sms FI, sq AL, sr Cyrl BA, sr Cyrl CS, sr Latin BA, sr Latin CS, sv SE, sw SE, sr KE, sr KE

T ta_IN, te IN, tg Cyrl TJ, th th_tk TM, tmz Latin DZ, tn ZA, tr IN, tr TR, tt RU

U ug_CN, uk UA, ur PK, uz Cyrl UZ, uz Latn UZ

V Nosotros VN

W cuando DE, wo SN

X xh_ZA

Y tu_de

Z de_China, de Hong Kong, de Moscú, de Singapur, de Taiwán y de Singapur

13. La especificación NUMERIC SIGN TRAILING SEPARATE hace que todos los elementos de datos USAGE DISPLAY numéricos con signo se creen como si la cláusula SIGN IS TRAILING SEPARATE CHARACTER estuviera incluida en sus definiciones.
14. La cláusula device-name-1 IS mnemotecnia-name-2 le permite especificar un nombre alternativo (device-name-1) para uno de los mnemotecnia-name-2 de nombre de dispositivo incorporados en GnuCOBOL. La lista de nombres de dispositivos incorporados en GnuCOBOL y el dispositivo físico asociado con ese nombre son los siguientes:

    CONSOLA Es la pantalla (modo pantalla) del sistema PC o Unix.

    SISTEMA
    STDIN
    SISTEMA Estos dispositivos (todos son sinónimos) representan la entrada estándar del sistema (tubería 0). En un sistema PC o UNIX, normalmente es el teclado. El

contenido de

Se puede entregar un archivo a un programa GnuCOBOL para acceder a él a través de uno de estos nombres de dispositivo agregando la secuencia '0< nombre_de_archivo' al final del comando de ejecución del programa.

LISTA
SYSL DE
SALIDA
ESTÁND
AR DE
IMPRES
ORA
SYSLST
SYSOUT Estos dispositivos (todos son sinónimos) representan la salida estándar del sistema (tubería 1). En un sistema PC o UNIX, normalmente es la pantalla. La salida enviada a uno de estos dispositivos por un programa GnuCOBOL se puede enviar a un archivo agregando la secuencia '1> filename' al final del comando de ejecución del programa.

ESTÉREO
SYSERR Estos dispositivos (son sinónimos) representan la salida de error estándar del sistema (tubería 2). En un sistema PC o UNIX, normalmente es la pantalla. La salida enviada a uno de estos dispositivos por un programa GnuCOBOL se puede enviar a un archivo agregando la secuencia '2> filename' al final del comando de ejecución del programa.

15. La cláusula feature-name-1 IS mnemotecnia-name-3 permite asignar nombres mnemotécnicos a los nombres de función de posición de hasta 13 canales de impresora (es decir, posicionamiento de página vertical) Cnn (nn=01-12) y CSP. Una vez que se ha asignado un nombre mnemotécnico a una posición de canal, las instrucciones del formato WRITE record-name AFTER ADVANCING mnemotecnia-name-3 pueden codificarse para escribir el registro de impresión especificado en la posición de canal asignada a mnemotecnia-name-3.

    Las impresoras que admiten el posicionamiento de canales son, por lo general, impresoras de línea de tipo mainframe. Al escribir en impresoras que no admiten el posicionamiento de canales, se enviará un avance de página a la impresora.

    La opción de posicionamiento CSP significa "Sin espaciado". Las pruebas en una compilación MinGW de Gnu-COBOL muestran que esto también da como resultado la emisión de un salto de página.

5.1.3.1 Alfabeto-Nombre-Cláusula

Sintaxis de cláusulas alfabéticas de NOMBRES ESPECIALES

```
ALFABETO nombre-alfabeto-1 ES { ASCII              }
~~~~~~~~                      { ~~~~~              }
                              { EBCDIC             }
                              { ~~~~~~~            }
                              { NATIVO             }
                              { ~~~~~~~            }
                              { ESTÁNDAR-1         }
                              { ~~~~~~~~~~~        }
                              { ESTÁNDAR-2         }
                              { ~~~~~~~~~~~        }
                              { Cláusula literal... }
```

ALFABETO DE NOMBRES ESPECIALES Sintaxis de cláusula literal

```
literal-1 [ { HASTA|HASTA literal-2 } ]
            { ~~~~ ~~~~~~~~          }
            { {TAMBIÉN literal-3}...   }
              ~~~~
```

La cláusula ALPHABET relaciona alphabet-name-1 con un conjunto de códigos de caracteres especificado o una secuencia de clasificación, incluida una que usted define utilizando la opción literal-1.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Las palabras reservadas THRU y THROUGH son intercambiables.
3. GnuCOBOL considera que ASCII, STANDARD-1 y STANDARD-2 son intercambiables.
4. NATIVE especifica el conjunto de caracteres predeterminado del sistema.
5. Los siguientes puntos se aplican al uso de las especificaciones literales-n para componer un conjunto de caracteres personalizado:
   A. Los valores literales n son números enteros o caracteres alfanuméricos entrecomillados. Estos representan un solo carácter en el conjunto de caracteres NATIVO, ya sea por su valor de texto real (carácter alfanumérico entrecomillado) o por posición ordinal en el conjunto de caracteres NATIVO (número entero).
   B. La secuencia en la que se definen los caracteres en esta cláusula especifica el orden relativo que deben tener dichos caracteres cuando se realizan comparaciones utilizando este alfabeto.
   C. Las posiciones de los caracteres en esta lista no afectan los valores de almacenamiento binarios reales utilizados para los caracteres. Los valores binarios seguirán siendo los del conjunto de caracteres NATIVO.
   D. Puede especificar cualquiera de las constantes figurativas ESPACIO, ESPACIOS, CERO,

CEROS, CEROS, COMILLA, COMILLAS, VALOR ALTO, VALORES ALTOS, VALOR BAJO o VALORES BAJOS para cualquiera de los
literal-1, especificaciones literal-2 o literal-3.

6. Una vez que haya definido un nombre de alfabeto, dicho nombre de alfabeto se puede usar en especificaciones en cláusulas CODE-SET, COLLATING SEQUENCE o SYMBOLIC CHARACTERS en otras partes del programa.

5.1.3.2 Cláusula de definición de clase

Sintaxis de cláusula de definición de clase de NOMBRES ESPECIALES

```
CLASE nombre-clase-1 ES { literal-1 [ HASTA|HASTA literal-2 ] }...
~~~~~                                ~~~~ ~~~~~~~
```

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Las palabras reservadas THRU y THROUGH son intercambiables.
3. Tanto el literal-1 como el literal-2 deben ser literales alfanuméricos de longitud 1.
4. Los literales especificados en esta cláusula definen los caracteres posibles que se pueden encontrar en el valor de un elemento de datos para ser considerado parte de la clase.
5. Por ejemplo, lo siguiente define una clase llamada Hexadecimal, cuya definición especifica los únicos caracteres que pueden estar presentes en un elemento de datos alfanuméricos si ese elemento de datos va a ser parte de la clase Hexadecimal:

```
LA CLASE Hexadecimal ES DEL '0' AL '9'
                         'A' A 'F'
                     'a' HASTA 'f'
```

6. Una vez que se ha definido la clase Hexadecimal, el código del programa podría usar una declaración como IF input-item IS Hexadecimal para determinar si el valor de los caracteres en un elemento de datos es válido de acuerdo con esa clase.

5.1.3.3 Cláusula de definición de conmutación

Sintaxis de cláusula de definición de cambio de NOMBRES ESPECIALES

nombre-interruptor-1 [IS nombre-mnemónico-1]

[EN ESTADO ES condición-nombre-1]
~~

[EL ESTADO DESACTIVADO ES condición-nombre-2]
~~~

———————————————————————————————————————

La cláusula switch-definition asocia un nombre de condición con un interruptor de ejecución en tiempo de ejecución para que el estado de ese interruptor pueda probarse desde dentro de un programa.

1. Las palabras reservadas IS y STATUS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Los nombres válidos para switch-name-1 son SWITCH-n (n = 0-36).
3. Si el programa se compila con el modificador -fsyntax-extension, los nombres de modificador SWn (n = 0-15) también son válidos; corresponden a SWITCH-0 a SWITCH-15, respectivamente, así como a SWITCH-16 a SWITCH-36, SWITCH 0 a SWITCH 26 y SWITCH A a SWITCH Z.
4. En el momento de la ejecución, cada conmutador se asociará con una variable de entorno de tiempo de ejecución COB_SWITCH_n, donde n tendrá el valor '0' a '15'. Cualquiera de estas dieciséis variables de entorno que tengan el valor ON (independientemente del valor en mayúsculas o minúsculas) se considerará como activada. Cualquiera de estas dieciséis variables de entorno que no tengan ningún valor o un valor distinto de ON se considerará desactivada.
5. Cada conmutador especificado debe tener al menos uno de los siguientes: IS mnemotécnico-nombre-1,Opción ON STATUS o OFF STATUS definida para él, de lo contrario no habrá forma de referenciar el interruptor desde dentro de un programa GnuCOBOL.
6. La sintaxis IS mnemotécnica-nombre-1 proporciona un medio para configurar el interruptor en ON o Valor OFF a través de la instrucción SET (verSección 7.8.43 [SET], página 379).
7. La sintaxis ON STATUS y OFF STATUS proporciona una forma de asociar un nombre de condición con el estado de encendido o apagado del interruptor, de modo que el estado se pueda probar en el momento de la ejecución a través de la declaración IF (consulteSección 7.8.23 [IF], página 338).
~~~

5.1.3.4 Cláusula de caracteres simbólicos

Sintaxis de cláusulas de caracteres simbólicos de NOMBRES ESPECIALES

PERSONAJES SIMBÓLICOS
~~~~~~~~

```
{ carácter-simbólico-1...                    ES|SON entero-1...     }...

[ EN nombre-alfabeto-1 ]
  ~~
```

Esta cláusula puede utilizarse para definir sus propias constantes figurativas.

1. Las palabras reservadas ARE, CHARACTERS y IS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Debe haber exactamente tantos valores enteros-1 especificados como haya *carácter simbólico 1*nombres.
3. Cada nombre de carácter simbólico se asociará con el carácter entero 1 correspondiente en el alfabeto nombrado en la cláusula IN. Los valores enteros seleccionan caracteres del alfabeto por su posición ordinal y no por su valor numérico; por lo tanto, un entero de 15 seleccionará el decimoquinto carácter en el alfabeto especificado, independientemente del valor numérico real del patrón de bits que constituye ese carácter.
4. Si no se especifica ningún nombre-alfabeto-1, se asumirá el conjunto de caracteres nativos del sistema.
5. Los dos ejemplos de código siguientes definen el mismo conjunto de nombres de constantes figurativas para cinco caracteres de control ASCII (suponiendo que ASCII es el conjunto de caracteres nativo del sistema). Los dos ejemplos son idénticos en sus efectos, aunque la forma en que se definen las constantes figurativas es diferente.

   Individualmente

   CARACTERES SIMBÓLICOS NUL ES 1
   SOH ES 2
   LA LLAMADA ES 8
   DC1 es 18
   DC2 ES 19

   Respectivamente

   CARACTERES SIMBÓLICOS NUL SOH BEL DC1 DC2
   SON 1 2 8 18 19
~~~~~~~~

5.1.4 REPOSITORIO

Sintaxis del REPOSITORIO

```
REPOSITORIO.
~~~~~~~~~~

FUNCIÓN { { nombre-función-intrínseca-1 } ...} INTRÍNSECO
~~~~~~~~ {      TODO INTRÍNSECO                    } ~~~~~~~~~~
                ~~~ ~~~~~~~~~~
          {     función-prototipo-nombre-1 [ AS literal-1 ] } ...
                                             ~~
PROGRAMA  {                                  nombre-prototipo-programa-1 [ AS literal-2 ] } ... .
~~~~~~~                                      ~~
```

El párrafo REPOSITORIO permite especificar los nombres de los prototipos de programas y funciones. También permite la declaración de nombres de funciones intrínsecas que pueden utilizarse sin especificar la palabra FUNCIÓN.

1. El párrafo REPOSITORIO no está permitido en un subprograma anidado. Un programa anidado hereda la configuración REPOSITORIO de su programa principal.
2. La cláusula INTRINSIC le permite marcar una o más (o TODAS) funciones intrínsecas integradas como utilizables sin la necesidad de codificar la palabra clave FUNCTION delante de los nombres de las funciones.
3. Como alternativa al uso de la cláusula ALL INTRINSIC, puede compilar sus programas Gnu-COBOL utilizando el modificador -fintrinsics=ALL.
4. La opción function-prototype-name-1 es necesaria para especificar el nombre de una función definida por el usuario que utilizará su programa. Opcionalmente, si lo desea, puede especificar un nombre de alias con el que hará referencia a esa función definida por el usuario. Si lo desea, también puede utilizar la cláusula AS para proporcionar un nombre de alias para una función intrínseca incorporada.
5. El siguiente ejemplo
   - permite especificar todas las funciones intrínsecas sin el uso de la palabra clave FUNCTION,
   - nombra dos funciones definidas por el usuario llamadas MY-FUNCTION-1 y MY-FUNCTION-2 que serán utilizadas por el programa y
   - especifica los nombres de alias SIGMA para la función intrínseca DESVIACIÓN ESTÁNDAR y MF2 para MI-FUNCION-2.

```
REPOSITORIO.
      FUNCIÓN TOTALMENTE INTRÍNSECA
      FUNCION MI-FUNCION-1
      FUNCIÓN MI-FUNCIÓN-2 COMO "MF2" FUNCIÓN
      DESVIACIÓN ESTÁNDAR COMO "SIGMA".
```

Otro ejemplo más completo extraído de las preguntas frecuentes:

```
          >>IDENTIFICACIÓN DE
FUENTE LIBRE      DIVISIÓN.
PROGRAMA-ID.      prog.
AMBIENTE          DIVISIÓN.
```

```
CONFIGURACIÓN    SECCIÓN.
REPOSITORIO.
      PROGRAMA prueba-de-
      recursión PROGRAMA
      prueba-de-cancelación.
DATOS
                       DIVISIÓN.S
ECCIÓN DE TRABAJO-
ALMACENAMIENTO.
01    En un            FOTO 9 VALOR 0.
PROCEDIMIENTO    DIVISIÓN.
      LLAMAR   prueba de recursión
      USANDO num MOSTRAR        "<"
      LLAMAR   cancelar prueba
      LLAMAR   cancelar
      pruebaCANCELAR
      cancelar-prueba
      LLAMADA cancelar
      pruebaMOSTRAR  "<"
      .
FIN DEL PROGRAMA prog.

IDENTIFICACIÓN   DIVISIÓN.
ID DEL PROGRAMA.     prueba de recursión
RECURSIVO. DATOS     DIVISIÓN.
ENLACE               SECCIÓN.
01    incógnita      FOTO 9.
PROCEDIMIENTO    DIVISIÓN USANDO x.
      AGREGAR   1  Para x
      MOSTRAR   x NO AVANZAR SI
                x = 1
                LLAMAR 'recursion-test' USANDO x
      FIN-SI.
FIN DEL PROGRAMA         prueba de recursión.


IDENTIFICACIÓN   DIVISIÓN.
ID DEL PROGRAMA.     cancelar-
prueba.DATOS
                       DIVISIÓN.SEC
CIÓN DE TRABAJO-
ALMACENAMIENTO.
01    incógnita      FOTO 9 VALOR 1.
PROCEDIMIENTO
                DIVISIÓN.MOSTR
      AR        x NO AVANZAR
      AGREGAR   1 a x.
FIN DEL PROGRAMA         cancelar-prueba.
```

Una nota especial sobre las funciones definidas por el usuario— debido a que debe nombrar una función definida por el usuario que su programa utilizará en el párrafo REPOSITORIO, siempre puede hacer referencia a esa función desde la división de procedimientos de su programa sin necesidad de

utilizar la palabra clave FUNCTION.

5.2 SECCIÓN DE ENTRADA-SALIDA

SECCIÓN DE ENTRADA-SALIDA Sintaxis

```
[SECCIÓN DE ENTRADA-SALIDA.]
   ~~~~~~~~~~~~ ~~~~~~~ [
CONTROL DE ARCHIVOS. ]
   ~~~~~~~~~~~~~
      [ Sentencia SELECT...     ]

[ CONTROL IO. ]
   ~~~~~~~~~~~
      [Declaración MULTIPLE-

      FILE][Declaración del mismo

      registro]
```

La sección ENTRADA-SALIDA proporciona la definición de cualquier archivo al que accederá el programa, así como el control del proceso de almacenamiento en búfer de E/S contra esos archivos a través de los párrafos CONTROL DE ARCHIVO y CONTROL DE E/S, respectivamente.

1. Como muestra el diagrama, hay tres tipos de enunciados que pueden aparecer en los dos párrafos de esta sección. Si ninguno de los enunciados está codificado en un párrafo en particular, se puede omitir el párrafo en sí; de lo contrario, es obligatorio.
2. Si no se codifica ninguno de los párrafos, se puede omitir el encabezado SECCIÓN DE ENTRADA-SALIDA; de lo contrario, normalmente se requiere.
3. Si el archivo de configuración del compilador (verSección 10.1.7 [Archivos de configuración del compilador], página 691) si está utilizando tiene la verificación de sintaxis relajada establecida en 'sí', los párrafos CONTROL DE ARCHIVO y CONTROL DE E/S pueden especificarse sin que se haya codificado el encabezado SECCIÓN DE ENTRADA-SALIDA.
4. Si ambos tipos de declaraciones están codificados en el párrafo IO-CONTROL, el orden en que se codifican dichas declaraciones es irrelevante.

5.2.1 SELECCIONAR

Sintaxis de la instrucción SELECT

```
SELECCIONAR [ [ NO ] OPCIONAL ] nombre-archivo-1
~~~~~~      ~~~  ~~~~~~~~
[ ASIGNAR { A   } [{ EXTERNO }] [{ DISCO | DISCO        }] [{ identificador-1 }] ]
  ~~~~~~ { USANDO }  { ~~~~~~~~ }     { ~~~~ ~~~~       }    { palabra-1        }
                     { DINÁMICO  }                    { MOSTRAR         }{ literal-1    }
                       ~~~~~~~~      { ~~~~~~~~         }
                                     { TECLADO          }
                                     { ~~~~~~~~~        }
                                     { LINEA AVANZANDO }
                                       { ~~~~ ~~~~~~~~~ }
                                     { IMPRESORA   }
                                     { ~~~~~~~~         }
                                     {ALEATORIO         }
                                     { ~~~~~~~          }
                                     { CINTA            }
                                       ~~~~
[ LA SECUENCIA DE COMPAÑAMIENTO
ES nombre-alfabeto-1 ]
  ~~~~~~~~~
[ ARCHIVO | ORDENAR ] EL ESTADO ES identificador-2 [ identificador-3 ] ]
  ~~~~ ~~~~   ~~~~~~
[ EL MODO DE BLOQUEO ES { MANUAL | AUTOMÁTICO       } ]
  ~~~~          { ~~~~~~ ~~~~~~~~~~                 }
        { EXCLUSIVO [ CON { BLOQUEO EN MÚLTIPLES REGISTROS } ] }
                    ~~~~~~~~~~        { ~~~~ ~~ ~~~~~~~~ ~~~~~~~ }
                                    {BLOQUEO EN REGISTRO          }
                                    { ~~~~ ~~ ~~~~~~              }
                                    { RETROCESO                   }
                                    { ~~~~~~~~~                   }
[ Cláusula de ORGANIZACIÓN ]
  ~~~~~~~~~~~~
[ Cláusula de ORGANIZACIÓN ]
  ~~~~~~~~~~~~
[EL DELIMITADOR DE REGISTRO ES ESTÁNDAR-1]
  ~~~~~~ ~~~~~~~~~~
                         ~~~~~~~~~~
[ RESERVA entero-1 ZONAS ]
  ~~~~~~~
[ COMPARTIENDO CON { TODOS LOS DEMÁS } ]
  ~~~~~~~      { ~~~          }
             {NINGUNA OTRA       }
               { ~~           }
              { SOLO LECTURA }
                ~~~~ ~~~~
```

Las cláusulas COLLATING SEQUENCE, RECORD DELIMITER, RESERVE y ALL OTHER se reconocen sintácticamente pero por lo demás no son funcionales.

La declaración SELECT crea una definición de un archivo y vincula esa definición COBOL al entorno del sistema operativo externo.

1. Las palabras reservadas AREAS, IS, MODE, OTHER, SEQUENCE, TO, USING y WITH son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Después de nombre-de-archivo-1, las distintas cláusulas pueden codificarse en cualquier secuencia.
3. Un punto debe seguir a la última cláusula codificada.
4. La cláusula OPTIONAL, que se debe utilizar únicamente para archivos que se utilizarán para proporcionar datos de entrada al programa, indica que el archivo puede o no estar disponible en tiempo de ejecución. Los intentos de ABRIR un archivo OPTIONAL cuando el archivo no existe recibirán un valor de estado de archivo especial no fatal (consulte el estado 05 en la lista de valores de estado de archivo a continuación) que indica que el archivo no está disponible; un intento posterior de LEER ese archivo devolverá una condición AT END (fin del archivo). Opcionalmente, los archivos pueden designarse como NO OPCIONALES, si se desea. Esto es útil cuando se especifica el modificador -foptional-file del compilador, que automáticamente convierte todos los archivos en OPCIONALES excepto aquellos declarados explícitamente como NO OPCIONALES.
5. El valor file-name-1 que especifique será el nombre con el que hará referencia al archivo dentro de su programa. Este nombre debe formarse de acuerdo con las reglas para nombres definidos por el usuario (consulteSección 2.1.2 [Palabras definidas por el usuario], página 10).
6. La cláusula opcional ASSIGN especifica cómo —en tiempo de ejecución, cuando se abre el archivo nombre-1— un dispositivo lógico (STDIN, STDOUT) o un archivo en cualquier parte de uno de los sistemas de archivos montados actualmente se asociará con el archivo nombre-1, de la siguiente manera:
   A. La cláusula ASSIGN tiene tres componentes:

      Tipo EXTERNO, DINÁMICO o ninguno

      Dispositivo la lista de opciones de dispositivos

      Locador se muestra como una elección entre identificador-1, palabra-1 y literal-1.
   B. Se asumirá ASSIGN TO DISC file-name-1 si no hay una cláusula ASSIGN en un SELECT.
   C. Si se codifica una cláusula ASSIGN sin un dispositivo, se asumirá el dispositivo DISC.
   D. Si se codifica una cláusula de localizador, el archivo COBOL nombre-archivo-1 se adjuntará a un archivo de datos dentro de cualquier sistema de archivos que esté montado y disponible para el programa en ejecución en el momento en que se abra el nombre-archivo-1. La forma en que se identifica ese archivo varía, según el localizador especificado, de la siguiente manera:
      a. Si se codifica literal-1, el valor del literal servirá como cadena de ubicación de archivo que identificará el archivo de datos.
      b. Si se codifica el identificador 1, el valor del identificador servirá como cadena de ubicación de archivo que identificará el archivo de datos.
      c. Si se codifica la palabra-1 (una palabra sintácticamente válida que no duplica una palabra reservada o definida por el usuario) y el tipo es EXTERNO, entonces la palabra-1 en sí servirá como la cadena de ubicación de archivo que identificará el archivo de datos. Sin embargo, si no se especificó un tipo EXTERNO, el compilador creará un elemento de datos PIC X(1024) llamado palabra-1 dentro del programa; el contenido de ese elemento de datos en el momento en que el programa abre el nombre-archivo-1 servirá entonces como la cadena de ubicación de archivo que identificará el archivo de datos.
      d. Las cadenas de ubicación de archivos se discutirán en breve.

E. Si no se codifica ningún localizador, el archivo nombre-1 se adjuntará a un dispositivo lógico o a un archivo según el dispositivo especificado (o implícito), de la siguiente manera:
   a. DISC o DISK asumirán que hay un archivo adjunto en un archivo llamado file-name-1 en cualquier directorio que esté actual en el momento en que se abre el archivo.

b. DISPLAY asumirá una conexión al dispositivo lógico STDOUT; estos archivos solo deben usarse para salida.
c. KEYBOARD asumirá una conexión al dispositivo lógico STDIN; estos archivos solo deben usarse para entrada.
d. PRINTER asumirá una conexión al dispositivo/puerto lógico LPT1; estos archivos solo deben usarse para salida.
e. RANDOM o TAPE se comportarán exactamente como lo hace DISC. Estos dos dispositivos adicionales se proporcionan para facilitar la compilación de código fuente COBOL a partir de otras implementaciones COBOL.

F. El dispositivo LINE ADVANCING requiere que se especifique un localizador; estos archivos solo se deben usar para salida. Un archivo de avance de línea COBOL permitirá que los caracteres de control de carro, como saltos de línea y saltos de página, se escriban en el archivo del sistema operativo adjunto, a través de la cláusula ADVANCING de la instrucción WRITE (consulteSección 7.8.54 [ESCRIBIR], página 410).

G. Las cadenas de ubicación de archivo se utilizan (en tiempo de ejecución) para identificar la ruta y el nombre del archivo de datos que se debe adjuntar a file-name-1 cuando se abre ese archivo.

H. Si el archivo de configuración del compilador (verSección 10.1.7 [Archivos de configuración del compilador], página 691) que utilizó para compilar el programa tenía un valor de mapeo de nombre de archivo de sí, el sistema de ejecución de GnuCOBOL primero intentará identificar una variable de entorno definida actualmente cuyo valor servirá como la ruta y el nombre de archivo del archivo de datos, de la siguiente manera:

a. Si el archivo de configuración del compilador (verSección 10.1.7 [Archivos de configuración del compilador], página 691) (verSección 10.1.7 [Archivos de configuración del compilador], página 691) que utilizó para compilar el programa especificó mf como el valor de la cláusula de asignación, entonces la cadena de localizador de archivos se interpretará de acuerdo con las reglas de Microfocus COBOL, es decir, todo lo anterior al último '-' en la cadena de localizador de archivos se ignorará; los caracteres después del último '-' se tratarán como la base de un nombre de variable de entorno. Si no hay ningún carácter '-' en la cadena de localizador de archivos, entonces toda la cadena de localizador de archivos servirá como base de un nombre de variable de entorno. Este es el comportamiento predeterminado para cada archivo de configuración excepto ibm.
b. Si, por otro lado, el archivo de configuración del compilador (verSección 10.1.7 [Com- Archivos de configuración de piler], página 691) que utilizó para compilar el programa especificó ibm como el valor de la cláusula de asignación, entonces la cadena de localizador de archivos se interpretará de acuerdo con las reglas de IBM COBOL, es decir, se espera que la cadena de localizador de archivos tenga el formato S-xxx o AS-xxx, en cuyo caso xxx se tratará como la base de un nombre de variable de entorno. Si no hay ningún carácter '-' en la cadena de localizador de archivos, entonces toda la cadena de localizador de archivos servirá como base de un nombre de variable de entorno.
c. Una vez que se ha determinado el nombre de una variable de entorno base (llamémosla bbbb), el sistema de ejecución buscará la primera de las siguientes variables de entorno que exista, en esta secuencia:

DD_yyyy
yyyyyyyy

```
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyyyyyyy
yyy
```

Los sistemas Windows no distinguen entre mayúsculas y minúsculas en lo que respecta a las variables de entorno, por lo que no hay diferencia entre los dos primeros cuando se utiliza una implementación de GnuCOBOL creada para Windows/MinGW o Windows nativo.

Si se encuentra una variable de entorno, su valor servirá como ruta y nombre de archivo al archivo de datos.

I. Si no se encontró ninguna variable de entorno o el archivo de configuración (verSección 10.1.7 [Archivos de configuración del compilador], página 691) utilizado para compilar el programa tenía un valor de asignación de nombre de archivo de NO, entonces el valor de la cadena de localización de archivo servirá como ruta y nombre de archivo al archivo de datos.

J. Las rutas y los nombres de archivo se pueden especificar en un absoluto (C:\\Data\\datafile.dat, /Data/datafile.dat, . . .) o en relación con el directorio actual (Data\\datafile.dat, Data/datafile.dat, . . .). Si no se incluye ningún nombre de directorio (datafile.dat), el archivo debe estar en el directorio actual.

7. La cláusula FILE STATUS o SORT STATUS (ambas son equivalentes y solo se debe especificar una u otra, si es que se debe especificar alguna) se utiliza para especificar el nombre de un elemento de datos numérico de dos dígitos en el que se guardará un código de estado de E/S después de cada verbo de E/S que se ejecute contra el archivo. Esto en realidad no asigna el elemento de datos; debe definir el elemento usted mismo en algún lugar de la división de datos. Tenga en cuenta que la siguiente lista está completa a partir de la v3.2, pero se pueden agregar más y todas las pruebas deben incluir una para los valores distintos de cero como un cajón de sastre. Consulte el archivo del libro de copias filestat.cpy (que puede no incluir todos estos códigos de estado enumerados). Consulte la versión de muestra más abajo. Los posibles códigos de estado que se pueden devolver a un elemento de datos FILE STATUS son los siguientes:

00	Éxito
02	Éxito (clave de registro duplicada escrita)
04	Éxito (incompleto)
05	Éxito (archivo opcional no encontrado)
06	Múltiples registros (en LS)
07	Éxito (Sin unidad)
09	Éxito LS Datos erróneos
10	Se alcanzó el final del archivo si se lee hacia adelante o se alcanzó el principio del archivo si se lee hacia atrás
14	Afuera de rango clave
21	Clave inválida
22	La clave ya existe
23	Clave no encontrada
24	Violación de límite de clave
30	Error permanente de E/S
31	Nombre de archivo inconsistente
34	Violación de límites
35	Archivo no encontrado
37	Permiso denegado
38	Cerrado con cerradura
39	Atributo conflictivo
41	Archivo ya abierto
42	Archivo no abierto

43 Lectura no realizada

44 Desbordamiento de registros

46 Error de lectura

47 ENTRADA ABIERTA denegada (permisos insuficientes para leer el archivo)

48 ABRIR SALIDA denegada (permisos insuficientes para escribir en el archivo)

49 OPEN IO denegado (permisos insuficientes para leer y/o escribir archivos)

51 Registro bloqueado

52 Fin de página

57 Mala especificación de LINAGE (linaje de E/S)

61 Error al compartir archivos

71 Mal personaje

91 Archivo no disponible - [No implementado]

8. Ejemplo de copia del libro de miembro, le sugerimos que copie este contenido en el formato FileStat-Msgs.cpy y lo guarde en su carpeta de copia del libro.

```
*>*********************************************************
*> Autor: Gary L. CutlerCutlerGL@gmail.com                *
*>                                                        *
*> Este libro de copias define una declaración EVALUATE capaz de *
*> traducir códigos FILE-STATUS de dos dígitos a un mensaje. *
*>                                                        *
*> Utilice la opción REEMPLAZAR para COPIAR para cambiar los *
   nombres de
*> los identificadores MSG y STATUS para                  *
*> los nombres que necesita su programa.                  *
*> chg 09/08/23 vbc por estados faltantes                 *
*> Chg 11/12/23 vbc por estados faltantes, nota enmendada *
   arriba
*>                    para permitir la expansión del cuaderno. *
*>*********************************************************
      EVALUAR ESTADO
          CUAN 00 MOVE 'Éxito                    ' A gluta
           DO      R                                mato
                                                    mon
                                                    osódi
                                                     co
          CUAN 02 MOVE 'Éxito duplicado           ' A gluta
           DO      R                                mato
                                                    mon
                                                    osódi
                                                     co
          CUAN 04 MOVE 'Éxito incompleto          ' A gluta
           DO      R                                mato
                                                    mon
                                                    osódi
                                                     co
          CUAN 05 MOVE 'Éxito opcional, faltante'   A gluta
           DO      R                                mato
                                                    mon
                                                    osódi
```

```
                                                                  co
cuando 06 mover "Múltiples registros LS"                       a  mensaje
CUANDO 07 MOVER 'Éxito Sin Unidad                              '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
cuando 09 mover "Éxito LS Datos erróneos"                      a  mensaje
CUANDO 10 MOVER 'Fin del archivo                               '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 14 MOVER 'Fuera de rango clave                          '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 21 MOVER 'Clave no válida                               '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 22 MOVER 'La clave existe                               '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 23 MOVER 'La clave no existe                            '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 24 MOVER 'Violación de límite clave                     '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 30 MOVER 'Error permanente                              '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 31 MOVER 'Nombre de archivo                                A gluta
                inconsistente                                  '  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 34 MOVER 'Violación de límites                          '  A gluta
                                                                  mato
                                                                  mon
                                                                  osódi
                                                                  co
CUANDO 35 MOVER 'Archivo no encontrado                            A gluta
                                                               '  mato
                                                                  mon
                                                                  osódi
                                                                  co
```

CUANDO	37	MOVER	'Permiso denegado '	A	glutamato monosódico

CUANDO	38	MOVER	'Cerrado con candado '	A	glutamato monosódico
CUANDO	39	MOVER	'Atributo de conflicto '	A	glutamato monosódico
CUANDO	41	MOVER	'Ya abierto '	A	glutamato monosódico
CUANDO	42	MOVER	'No abierto '	A	glutamato monosódico
CUANDO	43	MOVER	'Leer no terminado '	A	glutamato monosódico
CUANDO	44	MOVER	'Desbordamiento de registros '	A	glutamato monosódico
CUANDO	46	MOVER	'Error de lectura '	A	glutamato monosódico
CUANDO	47	MOVER	'Entrada denegada '	A	glutamato monosódico
CUANDO	48	MOVER	'Salida denegada '	A	glutamato monosódico
CUANDO	49	MOVER	'E/S denegada '	A	glutamato monosódico
CUANDO	51	MOVER	'Registro bloqueado '	A	glutamato monosódico
CUANDO	52	MOVER	'Fin de página '	A	glutamato

```
                                                                mon
                                                                osódi
                                                                co
        CUAN  57  MOVE  Linaje de E/S                    '  A  gluta
         DO         R                                           mato
                                                                mon
                                                                osódi
                                                                co
        CUAN  61  MOVE  'Error al compartir              '  A  gluta
         DO         R    archivos                               mato
                                                                mon
                                                                osódi
                                                                co
        CUAN  71  MOVE  'Mal carácter                    '  A  gluta
         DO         R                                           mato
                                                                mon
                                                                osódi
                                                                co
        CUAN  91  MOVE  'Archivo no disponible           '  A  gluta
         DO         R                                           mato
                                                                mon
                                                                osódi
                                                                co
        CUANDOO T R O
                MOVE "Estado de archivo desconocido       " AMSG
    FIN-EVALUAR.
```

9. Un código de muestra que define y utiliza este libro de copias en un programa fuente de formato libre.

En la sección de trabajo y almacenamiento:

```
 01   Áreas de trabajo WS.
      03   Mensaje de evaluación de WS   foto  x(25)        espacios de valor.
      03   Respuesta de Fs               foto  xx.
      03   WS-Sin cuidado                foto  incógnita.

 -----

 01   Mensajes de error.
*> Todo el sistema
      03   SL002                 pic x(31) valor "SL002 Nota de error y presione enter".
```

En la División de Procedimiento:

```
reescribir CLIENTE-Registro clave no válida
          mostrar "Reescribir registro de cliente" en la línea ws-23-lines
          col 1 mostrar CUSTOMER-FILE-STATUS en la línea WS-23-Lines
          col 22 mover CUSTOMER-FILE-STATUS a FS-Reply
          ejecutar mensaje de evaluación
          Mostrar ws-Eval-Msg en la línea ws-23-lines col 25
          pitido SL002            en la línea ws-lines col 1
          aceptar  WS-No-Cuidado      en la línea ws-lines col 33
```

espacio de exhibición en la línea ws-23-lines col 1 con borrado eos fin-reescritura

*>

```
 Evaluar-Mensaje              Sección.
*>================================
*>
 Copiar "FileStat-Msgs.cpy" reemplazando MSG por ws-Eval-Msg
                                          ESTADO por fs-reply.
*>
 Eval-Msg-Salir.   Sección de salida.
*>************     ************
*>
```

10. La cláusula SHARING define las condiciones bajo las cuales el programa estará dispuesto (o no) a permitir que otros programas que se ejecuten al mismo tiempo accedan al archivo. VerSección 2.1.30.1 [Compartir archivos], página 56, para más detalles.
11. La cláusula LOCK define cómo se gestionará el acceso simultáneo al archivo registro por registro. VerSección 2.1.30.2 [Bloqueo de registros], página 58, para más detalles.
12. Para obtener detalles de sintaxis de la cláusula ORGANIZATION, consulte el siguiente grupo de párrafos.
13. Una declaración SELECT sin una ORGANIZATION codificada explícitamente se manejará como si se hubiera especificado la siguiente cláusula ORGANIZATION:

    LA ORGANIZACIÓN ES SECUENCIAL EL
    MODO DE ACCESO ES SECUENCIAL

5.2.1.1 ORGANIZACIÓNSECUENCIAL

Sintaxis de la cláusula ORGANIZACIÓN SECUENCIAL

[ORGANIZACIÓN | ORGANIZACIÓN ES] REGISTRO BINARIO SECUENCIAL

[EL MODO DE ACCESO ES SECUENCIAL]

Los archivos declarados como ORGANIZACIÓN SECUENCIAL consistirán en registros sin secuencias explícitas de caracteres delimitadores de final de registro; los registros en dichos archivos están "delineados" por un desplazamiento de bytes calculado (basado en la longitud máxima del registro) en el archivo.

1. Las palabras reservadas BINARY, IS, MODE y RECORD son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas ORGANIZACIÓN y ORGANIZACIÓN son intercambiables.
3. La frase ORGANIZACIÓN ES (y su alternativa internacionalizada, ORGANIZACIÓN ES) es opcional para proporcionar compatibilidad con aquellas (pocas) implementaciones de COBOL que consideran que ORGANIZACIÓN es opcional. La mayoría de las implementaciones de COBOL requieren la palabra ORGANIZACIÓN, por lo que debería utilizarse en programas nuevos.
4. Estos archivos no se pueden preparar con ningún software de edición de texto o procesamiento de texto estándar, ya que todos estos programas incorporarán caracteres delimitadores al final de los registros (utilice ORGANIZACIÓN ES LÍNEA SECUENCIAL en su lugar).
5. Estos archivos pueden contener datos de USO DE VISUALIZACIÓN o de USO DE CÁLCULO (de cualquier variedad) ya que ninguna secuencia de datos binarios puede interpretarse accidentalmente como un delimitador de fin de registro.
6. Si bien los registros en un archivo SECUENCIAL DE ORGANIZACIÓN pueden definirse como registros de longitud variable, el archivo se estructurará de tal manera que reserve espacio para cada registro igual al tamaño del registro más grande posible, según la descripción del archivo en la SECCIÓN DE ARCHIVO.
7. La cláusula ACCESS MODE SEQUENTIAL es opcional porque, si no está presente, se asumirá de todos modos para este tipo de archivo. La estructura interna de estos archivos es tal que solo se pueden procesar de manera secuencial; para leer el registro número 100 en un archivo de este tipo, por ejemplo, primero se deben leer los registros del 1 al 99.
8. Los archivos secuenciales se procesan utilizando las siguientes declaraciones:
   - CERRAR (verSección 7.8.7 [CERRAR], página 295)
   - COMPROMISO (verSección 7.8.8 [COMMIT], página 296)
   - ELIMINAR (verSección 7.8.11 [ELIMINAR], página 301)
   - FUSIONAR (verSección 7.8.29 [FUSIÓN], página 351)
   - ABIERTO (verSección 7.8.32 [ABIERTA], página 358)
   - LEER (verSección 7.8.34 [LEER], página 365)
   - REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373)
   - ORDENAR (verSección 7.8.44 [SORT], página 388)
   - DESBLOQUEAR (verSección 7.8.52 [DESBLOQUEAR], página 405)

- ESCRIBIR (verSección 7.8.54 [ESCRIBIR], página 410)

5.2.1.2 ORGANIZACIÓN LINEA SECUENCIAL

Sintaxis de la cláusula SECUENCIAL DE LÍNEA DE ORGANIZACIÓN

```
[ ORGANIZACIÓN | ORGANIZACIÓN ES ] LÍNEA SECUENCIAL
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~ [ EL
    MODO DE ACCESO ES SECUENCIAL ]
      ~~~~~~~ ~~~~~~~~~~~
    [ EL CARÁCTER DE RELLENO ES literal-1 | identificador-1 ]
      ~~~~~~~~
```

La cláusula PADDING CHARACTER se reconoce sintácticamente pero por lo demás no es funcional.

Los archivos declarados como LÍNEA DE ORGANIZACIÓN SECUENCIAL consistirán en registros terminados por un carácter delimitador de fin de registro o una secuencia de caracteres.

1. Las palabras reservadas CHARACTER, IS y MODE son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas ORGANIZACIÓN y ORGANIZACIÓN son intercambiables.
3. La frase ORGANIZACIÓN ES (y su alternativa internacionalizada, ORGANIZACIÓN ES) es opcional para brindar compatibilidad con aquellas (pocas) implementaciones de COBOL que consideran que esa palabra es opcional. La mayoría de las implementaciones de COBOL requieren la palabra ORGANIZACIÓN, por lo que debería usarse en programas nuevos.
4. Esta es la única ORGANIZACIÓN válida para los archivos que están asignados al dispositivo IMPRESORA.
5. Estos archivos pueden crearse con cualquier software de edición de texto o de procesamiento de textos estándar capaz de escribir archivos de texto. Dichos archivos NO DEBEN contener ningún dato de CÁLCULO DE USO o BINARIO (de ningún tipo), ya que dichos campos podrían contener accidentalmente secuencias de bytes que podrían interpretarse como un delimitador de fin de registro.
6. Se admiten formatos de registros de longitud fija y variable.
7. La secuencia delimitadora de fin de registro será X'0A' (un carácter ASCII de avance de línea) o X'0D0A' (una secuencia ASCII de retorno de carro + avance de línea). La primera se utiliza en implementaciones Unix de GnuCOBOL (incluidos Windows/MinGW, Windows/Cygwin y OSX).implementaciones) mientras que este último se utilizaría con implementaciones nativas de Windows.
8. Al leer un archivo SECUENCIAL DE LÍNEAS, los registros que excedan el tamaño implicado en la descripción del archivo en la SECCIÓN DE ARCHIVO se truncarán, mientras que los registros más cortos que ese tamaño se rellenarán hacia la derecha con ESPACIOS.
9. La cláusula ACCESS MODE SEQUENTIAL es opcional porque, si no está presente, se asumirá de todos modos para este tipo de archivo. La estructura interna de estos archivos es tal que los datos solo se pueden procesar de manera secuencial; para leer el registro número 100 en un archivo de este tipo, por ejemplo, primero se deben leer los registros del 1 al 99.
10. Los archivos asignados a IMPRESORA o CONSOLA deben especificarse como LÍNEA DE ORGANIZACIÓN SECUENCIAL.
11. Los archivos secuenciales de línea se procesan utilizando las siguientes instrucciones:
    - CERRAR (verSección 7.8.7 [CERRAR], página 295)
    - COMPROMISO (verSección 7.8.8 [COMMIT], página 296)
    - ELIMINAR (verSección 7.8.11 [ELIMINAR], página 301)

- FUSIONAR (verSección 7.8.29 [FUSIÓN], página 351)

- ABIERTO (verSección 7.8.32 [ABIERTA], página 358)
- LEER (verSección 7.8.34 [LEER], página 365)
- REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373)
- ORDENAR (verSección 7.8.44 [SORT], página 388)
- DESBLOQUEAR (verSección 7.8.52 [DESBLOQUEAR], página 405)
- ESCRIBIR (verSección 7.8.54 [ESCRIBIR], página 410)

5.2.1.3 ORGANIZACIÓNRELATIVO

Sintaxis de la cláusula ORGANIZATION RELATIVE

```
[ ORGANIZACIÓN | LA ORGANIZACIÓN ES ] RELATIVA
  ~~~~~~~~~~~~ ~~~~~~~~~~~~
                                        ~~~~~~~~
    [ EL MODO DE ACCESO ES {
    SECUENCIAL } ]
        ~~~~~~          { ~~~~~~~~~~ }
                      { DINÁMICO }
                      { ~~~~~~~~   }
                      {ALEATORIO     }
                        ~~~~~~
    [ LA CLAVE RELATIVA ES identificador-1 ]
      ~~~~~~~~
```

Estos archivos son archivos con una organización interna tal que los registros pueden procesarse de manera secuencial en función de su ubicación física en el archivo o de manera aleatoria permitiendo que los registros se lean, escriban o actualicen especificando el número de registro relativo en el archivo.

1. Las palabras reservadas IS, KEY y MODE son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas ORGANIZACIÓN y ORGANIZACIÓN son intercambiables.
3. La frase ORGANIZACIÓN ES (y su alternativa internacionalizada, ORGANIZACIÓN ES) es opcional para brindar compatibilidad con aquellas (pocas) implementaciones de COBOL que consideran que esa palabra es opcional. La mayoría de las implementaciones de COBOL requieren la palabra ORGANIZACIÓN, por lo que debería usarse en programas nuevos.
4. Los archivos RELATIVOS A LA ORGANIZACIÓN no se pueden asignar a la CONSOLA, PANTALLA, AVANCE DE LÍNEA o dispositivos de IMPRESORA.
5. La cláusula CLAVE RELATIVA es opcional sólo si se especifica MODO DE ACCESO SECUENCIAL.
6. Si bien un archivo RELATIVO A ORGANIZACIÓN puede definirse como un archivo que tiene registros de longitud variable, el archivo se estructurará de tal manera que reserve espacio para cada registro igual al tamaño del registro más grande posible según lo define la descripción del archivo en la SECCIÓN DE ARCHIVO.
7. MODO DE ACCESO SECUENCIAL, el MODO DE ACCESO predeterminado si no se especifica ninguno, indica que los registros del archivo se procesarán de manera secuencial, de acuerdo con su secuencia física en el archivo.
8. MODO DE ACCESO ALEATORIO significa que los registros se procesarán en secuencia aleatoria especificando su número de registro en el archivo cada vez que se lea o escriba el archivo.
9. ACCESOMODE DYNAMIC indica que el programa puede alternar entre el modo SECUENCIAL y el ALEATORIO durante la ejecución. El archivo comienza inicialmente en modo SECUENCIAL cuando se abre por primera vez, pero el programa puede usar la instrucción START (consulteSección 7.8.45 [INICIO], página 393) para cambiar entre acceso secuencial y aleatorio.
10. El elemento de datos CLAVE RELATIVA es un elemento de datos numérico que no se puede definir como un campo dentro de los registros de este archivo. Su propósito es devolver el

número de registro relativo actual de un archivo relativo que se está procesando en modo de acceso SECUENCIAL y servir como una clave que especifica el número de registro relativo que se leerá o escribirá al procesar un archivo relativo en modo de acceso ALEATORIO.

11. Los archivos relativos se procesan utilizando las siguientes declaraciones:
- CERRAR (verSección 7.8.7 [CERRAR], página 295)
- COMPROMISO (verSección 7.8.8 [COMMIT], página 296)
- ELIMINAR (verSección 7.8.11 [ELIMINAR], página 301)
- FUSIONAR (verSección 7.8.29 [FUSIÓN], página 351), MODO DE ACCESO ALEATORIO no permitido
- ABIERTO (verSección 7.8.32 [ABIERTA], página 358)
- LEER (verSección 7.8.34 [LEER], página 365)
- REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373)
- ORDENAR (verSección 7.8.44 [SORT], página 388), MODO DE ACCESO ALEATORIO no permitido
- INICIO (verSección 7.8.45 [INICIO], página 393)
- DESBLOQUEAR (verSección 7.8.52 [DESBLOQUEAR], página 405)
- ESCRIBIR (verSección 7.8.54 [ESCRIBIR], página 410)

5.2.1.4 ORGANIZACIÓN INDEXADA

Sintaxis de la cláusula ORGANIZATION INDEXED

```
[ ORGANIZACIÓN | LA ORGANIZACIÓN ESTÁ ] INDEXADA
  ~~~~~~~~~~~~ ~~~~~~~~~~~~
                                        ~~~~~~~
    [ EL MODO DE ACCESO ES {
    SECUENCIAL } ]
  ~~~~~~            { ~~~~~~~~~~      }
                        { DINÁMICO }
                        { ~~~~~~~~   }
                        {ALEATORIO      }
                          ~~~~~~
    [ LA CLAVE DEL REGISTRO ES { [ nombre-datos-1      ]
      ~~~~~~
                        { [ nombre-clave-registro-1 ]
                    [ =|{FUENTE ES} nombre-datos-2 ] ...   ]        }
                                  ~~~~~~
    [ LA CLAVE DE REGISTRO ALTERNATIVO ES { [ nombre-datos-3      ]
      ~~~~~~~~~~ ~~~~~~
                                  { [ nombre-clave-registro-2 ]
                            [ =|{FUENTE ES} nombre-de-datos-4 ] ...      ]        }
                                        ~~~~~~
                                    [ CON DUPLICADOS ] ]...
                                        ~~~~~~~~~~~
                                    [SUPRIMIR CUANDO TODO literal          ]
                                      ~~~~~~~~~~~~~~~~~~
                                    [SUPRIMIR CUANDO HAY ESPACIOS | CEROS ]
                                      ~~~~~~~~~~~~~~~~~~~~~      ~~~~~~
```

———

Los archivos indexados, al igual que los archivos relativos, pueden procesar sus registros de manera secuencial o aleatoria. Sin embargo, a diferencia de los archivos relativos, la ubicación real de un registro en un archivo indexado se calcula automáticamente en función de los valores de uno o más campos alfanuméricos dentro de los registros del archivo. Por ejemplo, un archivo indexado que contiene datos de productos puede utilizar el código de identificación del producto como clave de registro. Esto significa que puede leer, escribir o actualizar el registro A6G4328th o el registro Z8X7723th directamente, en función del valor de identificación del producto de esos registros.

1. Las palabras reservadas IS, KEY y MODE son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas ORGANIZACIÓN y ORGANIZACIÓN son intercambiables.
3. La frase ORGANIZACIÓN ES (y su alternativa internacionalizada, ORGANIZACIÓN ES) es opcional para brindar compatibilidad con aquellas (pocas) implementaciones de COBOL que consideran que esa palabra es opcional. La mayoría de las implementaciones de COBOL requieren la palabra ORGANIZACIÓN, por lo que debería usarse en programas nuevos.
4. Los archivos INDEXADOS DE ORGANIZACIÓN no se pueden asignar a CONSOLA, PANTALLA, TECLADO, AVANCE DE LÍNEA o IMPRESORA.
5. ACCESOMODO SECUENCIAL, el MODO DE ACCESO predeterminado si no se especifica ninguno,

indica que los registros del archivo se procesarán de manera secuencial con respecto a los valores de la CLAVE DE REGISTRO o la CLAVE DE REGISTRO ALTERNATIVA a la que se hizo referencia más recientemente en una declaración START (consulteSección 7.8.45 [INICIO], página 393).

6. MODO DE ACCESO ALEATORIO significa que los registros se procesarán en secuencia aleatoria accediendo al registro con una clave de registro específica o valores de clave de registro alternativos.
7. EL MODO DE ACCESO DINÁMICO permite que el archivo sea procesado ya sea en modo ALEATORIO o SECUENCIAL; el programa puede cambiar entre los dos modos según sea necesario. La instrucción START se utiliza para realizar el cambio entre modos.
8. La cláusula RECORD KEY define el campo dentro del registro que se utiliza para proporcionar el acceso principal a los registros dentro del archivo. No se permitirá que dos registros del archivo tengan el mismo valor de campo PRIMARY KEY. La cláusula SOURCE IS se utiliza con claves divididas.
9. La cláusula ALTERNATE RECORD KEY, si se utiliza, define un campo adicional dentro del registro que proporciona un medio alternativo para acceder directamente a los registros o un campo adicional mediante el cual se puede procesar secuencialmente el contenido del archivo. Tiene la opción de permitir que los registros tengan valores de clave alternativos duplicados, si es necesario.
10. Puede haber varias cláusulas ALTERNATE RECORD KEY, cada una de las cuales define una clave alternativa adicional para el archivo.
11. El uso de la cláusula SUPPRESS WHEN se utiliza cuando se requieren claves dispersas que pueden tomar la forma de un literal, espacios o ceros.
12. Los archivos indexados se procesan utilizando las siguientes declaraciones:
    - CERRAR (verSección 7.8.7 [CERRAR], página 295)
    - COMPROMISO (verSección 7.8.8 [COMMIT], página 296)
    - ELIMINAR (verSección 7.8.11 [ELIMINAR], página 301)
    - FUSIONAR (verSección 7.8.29 [FUSIÓN], página 351), MODO DE ACCESO ALEATORIO no permitido
    - ABIERTO (verSección 7.8.32 [ABIERTA], página 358)
    - LEER (verSección 7.8.34 [LEER], página 365)
    - REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373)
    - ORDENAR (verSección 7.8.44 [SORT], página 388), MODO DE ACCESO ALEATORIO no permitido
    - INICIO (verSección 7.8.45 [INICIO], página 393)
    - DESBLOQUEAR (verSección 7.8.52 [DESBLOQUEAR], página 405)
    - ESCRIBIR (verSección 7.8.54 [ESCRIBIR], página 410)

5.2.2 MISMA ÁREA DE REGISTRO

Sintaxis de IO-CONTROL MISMA ÁREA

```
MISMA ÁREA { ORDENAR-FUSIONAR } PARA nombre-archivo-1... .
  ~~~~ { ~~~~~~~~~~ }
       { CLASIFICAR    }
       { ~~~~          }
       { REGISTRO }
         ~~~~~~
```

Las cláusulas SAME SORT-MERGE y SAME SORT se reconocen sintácticamente pero por lo demás no son funcionales.

La cláusula SAME RECORD AREA le permite especificar que varios archivos deben compartir los mismos buffers de memoria de entrada y salida.

1. Las palabras reservadas AREA y FOR son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Esta declaración debe terminar con un punto.
3. Si bien codificar un solo nombre de archivo (el elemento repetido file-name-1) es sintácticamente válido, esta declaración no tendrá efecto en el programa a menos que se especifiquen al menos dos archivos.
4. El efecto de esta declaración será hacer que los archivos especificados compartan el mismo buffer de E/S en la memoria. Estos buffers a veces pueden llegar a ser bastante grandes y, al tener varios archivos compartiendo la misma memoria de buffer, puede reducir significativamente la cantidad de memoria que utiliza el programa (de esta manera, haciendo "espacio" para más código procedimental o datos). Si utiliza esta función, asegúrese de que no más de uno de los archivos especificados esté ABIERTO simultáneamente.

5.2.3 ARCHIVOS MÚLTIPLES

Sintaxis de IO-CONTROL MULTIPLE FILE

LA CINTA CONTIENE VARIOS ARCHIVOS
~~~~~~~~

```
{ nombre-archivo-1 [ POSICIÓN entero-1 ] }...
                   ~~~~~~~~
```

La cláusula MULTIPLE FILE TAPE está obsoleta y, por lo tanto, se reconoce pero no es funcional.

———————————————————————————————————————

———————————————————————

Fin del Capítulo 5 — DIVISIÓN DE MEDIO AMBIENTE
```

# 6 DIVISIÓN DE DATOS

**Sintaxis de DIVISIÓN DE DATOS**

```
DIVISIÓN DE DATOS.
~~~~
~~~~~~~~[SECCI
ÓN DE
ARCHIVO.
~~~~ ~~~~~~~
{ Archivo/Ordenar-Descripción [{ ARCHIVO-SECCIÓN-Elemento-de-datos }]...   }...
                                                                                ]
{   { 01-Nivel-Constante   }   }
{   { 78-Nivel-Constante   }   }
{ 01-Nivel-Constante   }
{ 78-Nivel-Constante   }
[SECCIÓN DE TRABAJO-ALMACENAMIENTO.
~~~~~~~~~~~~~~~~ ~~~~~~~
[{ SECCIÓN-DE-ALMACENAMIENTO-DE-TRABAJO-Elemento-de-datos }]...]
  { 01-Nivel-Constante   }
  { 78-Nivel-Constante
                                          }
[SECCIÓN DE ALMACENAMIENTO
LOCAL.
~~~~~~~~~~~~~~ ~~~~~~~
[{ Elemento de datos de la sección de almacenamiento local }]...   ]
  { 01-Nivel-Constante   }
  { 78-Nivel-Constante
                                          }
[SECCIÓN DE ENLACE.
~~~~~~~ ~~~~~~~
[{ SECCIÓN DE ENLACE-Elemento-de-datos }]...   ]
  { 01-Nivel-Constante   }
  { 78-Nivel-Constante
                                  }
[SECCIÓN INFORME.
~~~~~~ ~~~~~~~
{ Descripción-del-Informe [{ Definición-del-Grupo-de-Informes }]...   }...   ]
{   { 01-Nivel-Constante   }   }
{   { 78-Nivel-Constante   }   }
{ 01-Nivel-Constante   }
{ 78-Nivel-Constante   }
[SECCIÓN DE PANTALLA.
~~~~~~ ~~~~~~~
[{ PANTALLA-SECCIÓN-Elemento-de-datos }]...   ]
  { 01-Nivel-Constante   }
  { 78-Nivel-Constante   }
```

Todos los datos utilizados por cualquier programa COBOL deben definirse en una de las seis secciones de la división de datos, dependiendo del propósito de los datos.

1. Si no se describirán datos en una de las secciones de división de datos, se puede omitir el encabezado de esa sección.
2. Si no se necesitan secciones de división de datos, se puede omitir el encabezado DATA DIVISION.
3. Si se necesita más de una sección en la división de datos (una situación común), las secciones deben codificarse en la secuencia en que se presentan arriba.

## 6.1 Principios de definición de datos

Los elementos de datos de GnuCOBOL, al igual que los de otras implementaciones de COBOL, se describen de manera jerárquica. Esto se adapta al hecho de que los elementos de datos con frecuencia deben poder dividirse en elementos subordinados. Tomemos como ejemplo la siguiente disposición lógica de una parte de un elemento de datos llamado Empleado:

El elemento de datos Employee consta de dos elementos de datos subordinados: un elemento de datos Employee-Name y un elemento de datos Employment-Dates (es de suponer que también habrá muchos otros, pero no nos interesan por ahora). Como muestra el diagrama, cada uno de esos elementos de datos se divide, a su vez, en elementos de datos subordinados. Esta jerarquía de elementos de datos puede llegar a ser bastante profunda y GnuCOBOL, al igual que otras implementaciones de COBOL, puede manejar hasta 49 niveles de dichas estructuras jerárquicas.

Como se presentó anteriormente (verSección 2.1.7 [Datos estructurados], página 13), un elemento de datos que se descompone en otros elementos de datos se denomina elemento de grupo, mientras que uno que no se descompone se denomina elemento elemental.

COBOL utiliza el concepto de número de nivel para indicar el nivel en el que se encuentra un elemento de datos en una estructura de datos, como el ejemplo que se muestra arriba. Cuando se definen estos elementos de datos, se definen todos juntos con un número en el rango de 1 a 49 especificado delante de sus nombres. Con el paso de los años, se ha llegado a establecer una convención entre los programadores de COBOL según la cual los números de nivel siempre se codifican como números de dos dígitos; no es necesario especificarlos como números de dos dígitos, pero todos los ejemplos que ve en este documento adoptarán ese enfoque.

El elemento de datos en la parte superior, también denominado registro, siempre tiene un número de nivel de 01. Después de eso, puede asignar números de nivel como desee (01–02–03–04..., 01–05–10–15..., etc.), siempre que siga estas sencillas reglas:

1. Cada elemento de datos en el mismo nivel de un diagrama de jerarquía como el que ve aquí (si hiciera uno, lo que rara vez hará, o nunca, una vez que se acostumbre a este concepto) debe tener el mismo número de nivel.
2. Cada nuevo nivel utiliza un número de nivel que es estrictamente mayor que el utilizado en el nivel padre (el siguiente nivel superior).
3. Al describir jerarquías de datos, nunca se puede utilizar un número de nivel mayor que 49 (excepto 66, 77, 78 y 88, que tienen significados muy especiales (consulteSección 6.8 [Elementos de datos especiales], página 145).

Entonces, la definición de estos elementos de datos en un programa GnuCOBOL sería algo así:

```
01 Empleado
    05 Nombre del empleado
        10 Apellido
        10 Nombre de pila
        10 Inicial del segundo nombre
```

```
        10 Desde-Fecha
            15 años
            15 meses
            15 días
    10 hasta la fecha
            15 años
            15 meses
            15 días
```

La sangría queda a criterio exclusivo del programador para facilitar la lectura a los humanos (al compilador no le importa en absoluto). Históricamente, las implementaciones de COBOL que requerían programas fuente en modo de formato fijo requerían que el número de nivel 01 comenzara en el área A y que todo lo demás comenzara en el área B. GnuCOBOL solo requiere que toda la sintaxis de definición de datos se encuentre en las columnas 8 a 72. En el modo de formato libre, por supuesto, ni siquiera existen esas limitaciones.

¿Has notado que hay dos nombres de datos definidos para Año, Mes y Día? Eso es perfectamente legal, siempre que cada uno pueda calificarse de forma única para que sea distinto del otro. Tomemos como ejemplo los elementos Año. Uno se define como parte del elemento de datos Desde-Fecha mientras que el otro se define como parte del elemento de datos Hasta-Fecha. En COBOL, en realidad codificaríamos las referencias a estos dos elementos de datos como Año OF Desde-Fecha y Año OF Hasta-Fecha o Año IN Desde-Fecha y Año IN Hasta-Fecha (COBOL permite utilizar IN u OF). Dado que estas referencias nos aclararían cualquier confusión sobre a qué Año se puede hacer referencia, el compilador GnuCOBOL tampoco se confundirá.

El ejemplo de codificación que se muestra arriba está incompleto; solo describe los nombres de los elementos de datos y sus relaciones jerárquicas entre sí. Además, cualquier definición válida de elemento de datos también deberá describir qué tipo de datos se incluirán en un elemento de datos (¿numéricos? ¿alfanuméricos? ¿alfabéticos?), cuántos datos se pueden almacenar en el elemento de datos y una multitud de otras características.

Cuando se definen elementos de grupo, a los elementos subordinados se les puede asignar el "nombre" FILLER. Puede haber cualquier cantidad de elementos FILLER definidos dentro de un elemento de grupo. No se puede hacer referencia directa a un elemento de datos llamado FILLER; estos elementos se utilizan generalmente para especificar una parte no utilizada del almacenamiento total asignado a un elemento de grupo. Tenga en cuenta que es posible que el nombre del elemento de grupo en sí se especifique como FILLER si no es necesario hacer referencia directa a la estructura del grupo en sí.

6.2 SECCIÓN DE ARCHIVO

Sintaxis de la SECCIÓN DE ARCHIVO

```
[SECCIÓN DE ARCHIVO.
  ~~~~ ~~~~~~~
  { Archivo/Ordenar-Descripción [ { ARCHIVO-SECCIÓN-Elemento-de-datos } ]...      }...
                                                                  ]
  {                          { 01-Nivel-Constante      }       }
  {                          { 78-Nivel-Constante      }       }
  { 01-Nivel-Constante                                         }
  { 78-Nivel-Constante                                         }
```

Cada archivo al que se ha hecho referencia mediante una declaración SELECT (verSección 5.2.1 [SELECCIONAR], página 102) también debe describirse en la sección de archivo de la división de datos.

Los archivos destinados a usarse como archivos de trabajo de clasificación/combinación deben describirse con una Descripción de archivo de clasificación/combinación (SD), mientras que todos los demás archivos se describen con una Descripción de archivo (FD). Cada una de estas descripciones casi siempre estará seguida de al menos una descripción de registro.

6.2.1 Descripción de archivo/ordenación

Sintaxis de descripción de archivo/ordenación

```
FD|SD nombre-archivo-1 [ ES EXTERNO|GLOBAL ]
~~ ~~                    ~~~~~~~~ ~~~~~~
[ BLOQUE CONTIENE [ entero-1 A ] entero-2 CARACTERES|REGISTROS ]
  ~~~~~~                        ~~                 ~~~~~~~~~~ ~~~~~~~~
[ EL CONJUNTO DE CÓDIGOS ES nombre-alfabeto-1 ]
  ~~~~~~~~
[ DATOS { EL REGISTRO ES                        } identificador-1...        ]
  ~~~~ { ~~~~~~~         }
       { LOS REGISTROS SON }
           ~~~~~~~~
[ ETIQUETA { EL REGISTRO ES    } OMITIDO|ESTÁNDAR ]
  ~~~~~~ { ~~~~~~~         } ~~~~~~~~ ~~~~~~~~~~
         { LOS REGISTROS SON }
           ~~~~~~~~
[ LINAJE ES entero-3 | identificador-2 LÍNEAS
  ~~~~~~~
    [ LÍNEAS INFERIORES entero-4 | identificador-3 ]
                ~~~~~~~
    [ LÍNEAS EN LA PARTE SUPERIOR entero-5 | identificador-4 ]
                ~~~
    [ CON PIE EN entero-6 | identificador-5 ] ]
          ~~~~~~~~
[ REGISTRO { CONTIENE [ entero-7 A ] entero-8 CARACTERES   } ]
  ~~~~~~~                            ~~
           { VARÍA EN TAMAÑO                                }
                ~~~~~~~~
           {       [ DE [ entero-7 A ] entero-8 CARACTERES }
                                       ~~
           {              DEPENDE DEL identificador-6 ]     }
                          ~~~~~~~~~~
[EL MODO DE GRABACIÓN ES modo-grabación ]
  ~~~~~~~~~~
[ { EL INFORME ES                        } informe-nombre-1...       ]
  { ~~~~~~~       }
  { LOS INFORMES SON }
    ~~~~~~~~
[ EL VALOR DE nombre-implementador-1 ES literal-1 | identificador-7 ].
  ~~~~~ ~~
```

Las cláusulas BLOQUE CONTIENE, REGISTRO DE DATOS, REGISTRO DE ETIQUETA, MODO DE GRABACIÓN y VALOR DE
Se reconocen sintácticamente, pero son obsoletas y no funcionales. Estas cláusulas no deberían codificarse en programas nuevos.

---

1. Las palabras reservadas ARE, AT, CHARACTERS (sólo cláusula RECORD), CONTAINS, FROM, IN, IS, ON y WITH son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.

2. Los términos RECORD IS y RECORDS SON son intercambiables.
3. Los términos INFORME ES y INFORMES SON son intercambiables.
4. Solo los archivos destinados a usarse como archivos de trabajo para SORT (verSección 7.8.44 [ORDENAR], página 388) o FUSIONAR (verSección 7.8.29 [FUSIÓN], página 351) Las declaraciones deben codificarse con un SD; todas las demás deben definirse con un FD.
5. La secuencia en la que se definen los archivos a través de FD o SD, en comparación con la secuencia en la que se codificaron sus declaraciones SELECT, es irrelevante.
6. El nombre especificado como nombre-de-archivo-1 debe coincidir exactamente con el nombre especificado en el archivo.
   Sentencia SELECT.
7. La cláusula CODE-SET permite un alfabeto personalizado, definido en SPECIAL-NAMES (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) párrafo, que se asociará con un archivo. Esta cláusula es válida únicamente cuando se utiliza con archivos secuenciales o secuenciales de línea y su uso puede depender de la versión del compilador en uso, la plataforma utilizada y otros factores; consulte el archivo NEWS que se incluye con su versión del compilador para obtener más detalles.
8. La cláusula LINAGE solo se puede especificar en el FD de un archivo secuencial o secuencial de líneas. Si se utiliza con un archivo secuencial, la organización de ese archivo se cambiará implícitamente a secuencial de líneas. Los diversos componentes de la cláusula LINAGE definen el diseño de las páginas impresas de la siguiente manera:

   LINEAS EN LA PARTE SUPERIOR
   : Número de líneas no utilizadas (es decir, que se dejan en blanco) en la parte superior de cada página. El valor predeterminado, si no se especifica, es cero.

   LÍNEAS DE ABAJO
   : Número de líneas no utilizadas (es decir, que se dejan en blanco) en la parte inferior de cada página. El valor predeterminado, si no se especifica, es cero.

   LINAJE ES n LINEAS
   : Número total de líneas utilizadas/utilizables en la página.

   La suma de las tres especificaciones anteriores debe ser el número total de líneas posibles disponibles en una página impresa.

   PIE EN
   : Número de línea más allá del cual no se puede imprimir nada, excepto el pie de página que aparecerá en cada página. El valor predeterminado para esto, si no se especifica, es cero, lo que significa que no habrá pies de página. Este valor no puede ser mayor que el valor de LINAGE IS n LINES.
9. Esta estructura de página, una vez definida, se puede aplicar automáticamente mediante la declaración WRITE(verSección 7.8.54 [ESCRIBIR], página 410).
10. Si se especifica una cláusula LINAGE en un archivo FD, se creará el registro especial LINAGE-COUNTER para el archivo. Este elemento de datos creado automáticamente siempre contendrá el número de línea relativo actual en la página que se está preparando, que servirá como punto de partida para una declaración WRITE.
11. Las cláusulas RECORD CONTAINS y RECORD IS VARYING se ignoran (se emite un mensaje de advertencia) cuando se utilizan con archivos secuenciales de líneas. Con otras organizaciones de archivos, estas cláusulas mutuamente excluyentes definen la longitud de los registros de datos dentro del archivo. El elemento de datos especificado como identificador-6 debe definirse dentro de una de las descripciones de registro de nombre-de-archivo-1.

12. La cláusula REPORT IS anuncia al compilador que el archivo se dedicará al Sistema de control del escritor de informes (RWCS); la cláusula nombra uno o más informes, cada uno de los cuales será

Se describe en la sección de informes. Se aplican las siguientes reglas especiales cuando se utiliza la cláusula REPORT:

A. La cláusula solo se puede especificar en el FD de un archivo secuencial o secuencial de líneas. Si se utiliza con un archivo secuencial, la organización de ese archivo se cambiará implícitamente a secuencial de líneas.
B. El FD no puede ir seguido de descripciones de registros. Se deben incluir descripciones detalladas de los datos.impreso en el archivo se definirá en la SECCIÓN INFORME (verSección 6.6 [INFORME SECCIÓN], página 135).
C. Si también se especifica una cláusula LINAGE, se ignorarán los valores especificados para LINAGE IS y FOOTING AT. Se respetarán los valores de LINES AT BOTTOM y LINES AT TOP, si los hay.

13. Las siguientes reglas especiales se aplican únicamente para ordenar/fusionar archivos de trabajo:
    A. Los archivos de trabajo de clasificación/combinación deben asignarse a DISK (o DISC) en sus declaraciones SELECT.
    B. Las ordenaciones y fusiones se realizarán en la memoria, si la cantidad de datos que se están ordenando lo permite.
    C. Si se necesitan archivos de trabajo del disco debido a la cantidad de datos que se están ordenando o fusionando, se asignarán automáticamente al disco en una carpeta definida por:
        - La variable de entorno de ejecución TMPDIR (verSección 10.2.3 [Entorno de tiempo de ejecución] Variables ambientales], página 700)
        - La variable de entorno de ejecución de TMP
        - La variable de entorno de ejecución TEMP

        (en ese orden).
    D. Estos archivos de disco se eliminarán automáticamente al finalizar SORT o MERGE. También se eliminarán si el programa finaliza de forma anormal antes de que finalicen las operaciones SORT o MERGE. Si alguna vez necesita saberlo, los archivos temporales de trabajo de ordenación y fusión se llamarán cob*.tmp.
    E. Si especifica un nombre de archivo específico en el SELECT del archivo de trabajo de clasificación/combinación, se ignorará.
14. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el EXTERNO y Opciones GLOBALES.
15. implementor-name-1 puede tomar el valor: FILE-ID y se utiliza para especificar el nombre externo del archivo como literal-1 (solo si este nombre externo no se ha especificado en la cláusula SELECT correspondiente). Ejemplo:

```
SELECCIONAR archivo1 ASIGNAR A DISCO (sin un nombre externo...
                                                    ---
Departamento de Bomberos Archivo 1
    VALOR DEL ARCHIVO-ID "file1.txt".
```

6.2.2 ARCHIVO-SECCIÓN-Elemento-de-datos

Sintaxis de elemento de datos de sección de archivo

```
número de nivel [ identificador-1 | RELLENO ] [ ES GLOBAL | EXTERNO ]
                                  ~~~~~~~      ~~~~~~~ ~~~~~~~~~

[EN BLANCO CUANDO ES CERO ]
  ~~~~~      ~~~~

[ DERECHO JUSTIFICADO ]
  ~~~~

[ OCURRE [ entero-1 A ] entero-2 VECES
  ~~~~~~                ~~

        [ DEPENDE DEL identificador-2 ]
          ~~~~~~~~~~

        [ Identificador STEP-6 ]
        [ LA CLAVE ASCENDENTE|DESCENDENTE ES identificador-3 ]
          ~~~~~~~~~~ ~~~~~~~~~~~

        [ INDEXADO POR identificador-4 ] ]
          ~~~~~~~

[ LA IMAGEN ES una cadena de imágenes ]
  ~~~

[ REDEFINE identificador-5 ]
  ~~~~~~~~~~

[ EL SIGNO ESTÁ LIDERANDO|DETRÁS [ SEPARAR [ CARÁCTER ] ]
  ~~~~       ~~~~~~~~ ~~~~~~~~~      ~~~~~~~~~

[ SINCRONIZADO | SINCRONIZADO [ IZQUIERDA | DERECHA ] ]
  ~~~~             ~~~~                ~~~~ ~~~~~

[ EL USO ES uso-de-elemento-de-datos ] .[ ARCHIVO-SECCIÓN-Elemento-de-
datos ]...
  ~~~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

Cada descripción de archivo de ordenación (SD o FD) debe ir seguida de al menos un elemento de datos de nivel 01, excepto las descripciones de archivo que contienen la cláusula REPORT IS. Estos elementos de datos de nivel 01, a su vez, pueden desglosarse en elementos de grupo subordinados y elementales. Un elemento de datos de nivel 01 definido aquí en la sección de archivo también se conoce como Registro, incluso si es un elemento elemental, siempre que dicho elemento elemental carezca del atributo CONSTANT.

1. Las palabras reservadas BY, IS, KEY, ON y WHEN son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas SYNCHRONIZED y SYNCHRONISED son intercambiables. Ambas pueden abreviarse como SYNC.
3. La palabra reservada PICTURE puede abreviarse como PIC.
4. Como muestra el diagrama de sintaxis, la definición de un elemento de datos de la sección de archivo es recursiva, ya que puede haber cualquier cantidad de especificaciones de este tipo codificadas después de una definición de archivo o definición de archivo. La primera de

estas especificaciones debe tener un número de nivel de 01 y describirá un formato específico de registro de datos dentro del archivo. Las especificaciones que siguen a esa pueden tener un nivel de

números mayores que 01, en cuyo caso están definiendo un desglose jerárquico del registro. La definición de un registro finaliza cuando ocurre una de las siguientes situaciones:

Se encuentra otro elemento de nivel 01
: significa el inicio de otro diseño de registro para el archivo.

Se encuentra otro FD o SD
: marca la finalización de la descripción detallada del archivo y comienza otra.

Se encuentra un encabezado de división o sección
: También marca la finalización de la descripción detallada del archivo y también significa el final de la sección del archivo.

5. Cada descripción de elemento de datos de sección de archivo debe terminar con un punto.
6. Si hay múltiples descripciones de registros presentes para un FD o SD determinado, la que tenga la longitud más larga definirá el tamaño del búfer de registros en el que se ejecutará una declaración READ (consulteSección 7.8.34 [LEER], página 365) o una declaración RETURN (verSección 7.8.38 [DEVOLUCIÓN], página 372) entregará los datos leídos del archivo y desde el cual se ejecuta una instrucción WRITE (verSección 7.8.54 [ESCRIBIR], página 410) o la declaración RELEASE (verSección 7.8.36 [LIBERACIÓN], página 370) obtendrá los datos que se escribirán en el archivo.
7. Las diversas descripciones de registros de nivel 01 para una descripción de archivo comparten implícitamente ese búfer de registro común (por lo tanto, proporcionan diferentes formas de ver la estructura de datos que pueden existir dentro del archivo). Los búferes de registros se pueden compartir entre archivos utilizando la MISMA ÁREA DE REGISTRO (consulteSección 5.2.2 [MISMA ÁREA DE REGISTRO], página 116) cláusula.
8. Los únicos números de nivel válidos son 01-49, 66, 77, 78 y 88. Los números de nivel 66, 77, 78 y 88 tienen usos especiales: consulteSección 6.8 [Elementos de datos especiales], página 145, para más detalles.
9. No especificar un identificador-1 o FILLER inmediatamente después del número de nivel tiene el mismo efecto que si se hubiera especificado FILLER. No se puede hacer referencia directa a un elemento de datos denominado FILLER; estos elementos se utilizan generalmente para especificar una parte no utilizada del almacenamiento total asignado a un elemento de grupo o para describir un elemento de grupo cuyo contenido solo se referenciará utilizando los nombres de los elementos que pertenecen a él.
10. EXTERNO no se puede combinar con GLOBAL o REDEFINES.
11. Los buffers de datos de la sección de archivo (y, por lo tanto, todos los diseños de registros de nivel 01 definidos en la sección de archivo) se inicializan con todos los ceros binarios cuando el programa se carga en el almacenamiento.
12. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.3 SECCIÓN DE TRABAJO Y ALMACENAMIENTO

Sintaxis del elemento de datos de la sección de almacenamiento de trabajo

```
número de nivel [ identificador-1 | RELLENO ] [ ES GLOBAL | EXTERNO ]
                                  ~~~~~~      ~~~~~~   ~~~~~~~~
[ BASADO ]
  ~~~~~
[EN BLANCO CUANDO ES CERO ]
  ~~~~~      ~~~~
[ DERECHO JUSTIFICADO ]
  ~~~~
[ OCURRE [ entero-1 A ] entero-2 VECES
  ~~~~~~            ~~
      [ DEPENDE DEL identificador-2 ]
        ~~~~~~~~~
      [ LA CLAVE ASCENDENTE|DESCENDENTE ES identificador-3 ]
        ~~~~~~~~~ ~~~~~~~~~~
      [ INDEXADO POR identificador-4 ] ]
        ~~~~~~~
[ LA IMAGEN ES una cadena de imágenes ]
  ~~~
[ REDEFINE identificador-5 ]
  ~~~~~~~~~
[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ]
  ~~~~     ~~~~~~~ ~~~~~~~~   ~~~~~~~~
[ SINCRONIZADO | SINCRONIZADO [ IZQUIERDA | DERECHA ] ]
  ~~~~           ~~~~            ~~~~ ~~~~~
[ EL USO ES data-item-usage ]
  ~~~~~
[ EL VALOR ES [ TODOS ] literal-1 ].  [ SECCIÓN DE ALMACENAMIENTO DE TRABAJO - Elemento
de datos ]...
  ~~~~~      ~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

La sección de almacenamiento de trabajo se utiliza para describir elementos de datos que no forman parte de archivos, pantallas o informes y cuyos valores de datos persisten durante la ejecución del programa.

1. Las palabras reservadas BY, CHARACTER, IS, KEY, ON, RIGHT (JUSTIFIED), TIMES y WHEN son son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas SYNCHRONIZED y SYNCHRONISED son intercambiables. Ambas pueden abreviarse como SYNC.
3. La palabra reservada PICTURE puede abreviarse como PIC.
4. La palabra reservada JUSTIFICADO puede abreviarse como JUST.
5. Como muestra el diagrama de sintaxis, la definición de un elemento de datos de sección de almacenamiento de trabajo es recursiva, ya que puede haber cualquier cantidad de especificaciones de este tipo codificadas una después de la otra. La primera de estas

especificaciones debe tener un número de nivel de 01. Las especificaciones que siguen a esa pueden tener números de nivel mayores que 01, en cuyo caso están definiendo un elemento de datos de sección de almacenamiento de trabajo.

Desglose jerárquico de un registro. La definición de un registro finaliza cuando se produce una de las siguientes situaciones:

- Se encuentra otro elemento de nivel 01: esto significa el final de la definición de un registro y el comienzo de otro.
- Se encuentra un elemento de nivel 77: esto significa el final de la definición del registro y comienza la definición de un elemento de datos especial; consulteSección 6.8.3 [Elementos de datos de nivel 77], página 150, para obtener más información.
- Se encuentra un encabezado de división o sección: esto también marca la finalización de un registro y también significa el final de la sección de trabajo y almacenamiento.

6. Cada descripción de elemento de datos de sección de almacenamiento de trabajo debe terminar con un punto.
7. Los únicos números de nivel válidos son 01-49, 66, 77, 78 y 88. Los números de nivel 01 a 49 se utilizan para definir elementos de datos que pueden ser parte de una estructura jerárquica. El número de nivel 01 también se puede utilizar para definir una constante: un elemento con un valor inmutable especificado en el momento de la compilación.
8. Los niveles 66, 77, 78 y 88 tienen usos especiales: consulteSección 6.8 [Datos especiales Artículos], página 145, para más detalles.
9. No especificar un identificador-1 o FILLER inmediatamente después del número de nivel tiene el mismo efecto que si se hubiera especificado FILLER. No se puede hacer referencia directa a un elemento de datos denominado FILLER; estos elementos se utilizan generalmente para especificar una parte no utilizada del almacenamiento total asignado a un elemento de grupo o para describir un elemento de grupo cuyo contenido solo se referenciará utilizando los nombres de los elementos que pertenecen a él.
10. Los elementos de datos definidos dentro de la sección de almacenamiento de trabajo se inicializan automáticamente una vez, a medida que se carga en la memoria el programa en el que se definen los datos. Los subprogramas se pueden cargar en la memoria más de una vez (consulte la declaración CANCEL (consulteSección 7.8.6 [CAN- [CEL], página 294)), en cuyo caso la inicialización se producirá cada vez que se carguen. VerSección 2.1.14 [Inicialización de datos], página 25, para una discusión de las reglas de inicialización.
11. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.4 SECCIÓN DE ALMACENAMIENTO LOCAL

Sintaxis de elemento de datos de sección de almacenamiento local

```
número de nivel [ identificador-1 | RELLENO ] [ ES GLOBAL | EXTERNO ]
                                  ~~~~~~      ~~~~~~ ~~~~~~~~
[ BASADO ]
  ~~~~~
[EN BLANCO CUANDO ES CERO ]
  ~~~~~      ~~~~
[ DERECHO JUSTIFICADO ]
  ~~~~
[ OCURRE [ entero-1 A ] entero-2 VECES
  ~~~~~~          ~~
      [ DEPENDE DEL identificador-2 ]
        ~~~~~~~~~
      [ LA CLAVE ASCENDENTE|DESCENDENTE ES identificador-3 ]
        ~~~~~~~~~ ~~~~~~~~~~
      [ INDEXADO POR identificador-4 ] ]
        ~~~~~~~
[ LA IMAGEN ES una cadena de imágenes ]
  ~~~
[ REDEFINE identificador-5 ]
  ~~~~~~~~~
[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ]
  ~~~~     ~~~~~~~ ~~~~~~~~    ~~~~~~~~
[ SINCRONIZADO | SINCRONIZADO [ IZQUIERDA | DERECHA ] ]
  ~~~~           ~~~~              ~~~~ ~~~~~
[ EL USO ES data-item-usage ]
  ~~~~~
[ EL VALOR ES [ TODOS ] literal-1 ].  [Elemento de datos de la sección de almacenamiento
local]...
  ~~~~~      ~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

———————————————————————————————————————

La sección de almacenamiento local es similar a la de almacenamiento de trabajo, pero describe los datos dentro de un subprograma que se asignarán dinámicamente e inicializarán (automáticamente) cada vez que se ejecute el subprograma.Sección 2.1.14 [Inicialización de datos], página 25, para las reglas de inicialización de datos.

1. Las palabras reservadas BY, CHARACTER IS, KEY, ON, RIGHT (JUSTIFIED), TIMES y WHEN son son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas SYNCHRONIZED y SYNCHRONISED son intercambiables. Ambas pueden abreviarse como SYNC.
3. La palabra reservada PICTURE puede abreviarse como PIC.
4. La palabra reservada JUSTIFICADO puede abreviarse como JUST.
5. Como muestra el diagrama de sintaxis, la definición de un elemento de datos de una SECCIÓN DE ALMACENAMIENTO LOCAL es recursiva, ya que puede haber cualquier cantidad

de especificaciones de este tipo codificadas una tras otra. La primera de estas especificaciones debe tener un número de nivel de 01. Especificaciones que

De ello se deduce que se pueden tener números de nivel mayores que 01, en cuyo caso se está definiendo un desglose jerárquico de un registro. La definición de un registro finaliza cuando se produce una de las siguientes situaciones:

- Se encuentra otro elemento de nivel 01: esto significa el final de la definición de un registro y el comienzo de otro.
- Se encuentra un encabezado de división o sección: esto también marca la finalización de un registro y también significa el final de la sección de almacenamiento local.

6. Cada descripción de elemento de datos de SECCIÓN DE ALMACENAMIENTO LOCAL debe terminar con un punto.
7. Los únicos números de nivel válidos son 01-49, 66, 77, 78 y 88. Los números de nivel 01 a 49 se utilizan para definir elementos de datos que pueden ser parte de una estructura jerárquica. El número de nivel 01 también se puede utilizar para definir una constante: un elemento con un valor inmutable especificado en el momento de la compilación.
8. Los niveles 66, 77, 78 y 88 tienen usos especiales: consulteSección 6.8 [Datos especiales Artículos], página 145, para más detalles.
9. No especificar un identificador-1 o FILLER inmediatamente después del número de nivel tiene el mismo efecto que si se hubiera especificado FILLER. No se puede hacer referencia directa a un elemento de datos denominado FILLER; estos elementos se utilizan generalmente para especificar una parte no utilizada del almacenamiento total asignado a un elemento de grupo o para describir un elemento de grupo cuyo contenido solo se referenciará utilizando los nombres de los elementos que pertenecen a él.
10. El almacenamiento local no se puede utilizar en subprogramas anidados.
11. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.5 SECCIÓN DE ENLACE

Sintaxis de elemento de datos de sección de enlace

```
número de nivel [ identificador-1 | RELLENO ] [ ES GLOBAL | EXTERNO ]
                                  ~~~~~~~      ~~~~~~~ ~~~~~~~~~

[ CUALQUIER LONGITUD ]
  ~~~ ~~~~~~~

[ CUALQUIER NUMÉRICO ]
  ~~~

~~~~~~~~[
BASADO ]
  ~~~~~

[EN BLANCO CUANDO ES CERO ]
  ~~~~~        ~~~~

[ DERECHO JUSTIFICADO ]
  ~~~~

[ OCURRE [ entero-1 A ] entero-2 VECES
  ~~~~~~                ~~   ILIMITADO
                             ~~~~~~~~~~

      [ DEPENDE DEL
      identificador-3 ]
        ~~~~~~~~~~

      [ LA CLAVE ASCENDENTE|DESCENDENTE ES identificador-4 ]
        ~~~~~~~~~~ ~~~~~~~~~~~

      [ INDEXADO POR identificador-5 ] ]
        ~~~~~~~~

[ LA IMAGEN ES una cadena de imágenes ]
  ~~~

[ REDEFINE identificador-6 ]
  ~~~~~~~~~~

[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ]
  ~~~~       ~~~~~~~~ ~~~~~~~~~      ~~~~~~~~~

[ SINCRONIZADO | SINCRONIZADO [ IZQUIERDA | DERECHA ] ]
  ~~~~             ~~~~                ~~~~ ~~~~~~

[ EL USO ES uso-de-elemento-de-datos ] .[ ENLACE-SECCIÓN-Elemento-de-datos ]...
  ~~~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

La sección de enlace describe datos dentro de un subprograma que sirven como argumentos de entrada o resultados de salida del subprograma.

1. Las palabras reservadas BY, CHARACTER, IS, KEY, ON y WHEN son opcionales y pueden incluirse o no, a discreción del programador. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas SYNCHRONIZED y "SYNCHRONISED" son intercambiables. Ambas pueden abreviarse como SYNC.
3. La palabra reservada PICTURE puede abreviarse como PIC.

4. La palabra reservada JUSTIFICADO puede abreviarse como JUST.

5. Como muestra el diagrama de sintaxis, la definición de un elemento de datos de SECCIÓN DE ENLACE es recursiva, ya que puede haber cualquier cantidad de especificaciones de este tipo codificadas una tras otra. La primera de estas especificaciones debe tener un número de nivel de 01. Las especificaciones que siguen a esta pueden tener números de nivel mayores que 01, en cuyo caso están definiendo un desglose jerárquico de un registro. La definición de un registro finaliza cuando ocurre una de las siguientes situaciones:
   - Se encuentra otro elemento de nivel 01: esto significa el final de la definición de un registro y el comienzo de otro.
   - Se encuentra un encabezado de división o sección: esto también marca la finalización de un registro y también significa el final de la sección de enlace.
6. Cada descripción de elemento de datos de SECCIÓN DE ENLACE debe terminar con un punto.
7. Los únicos números de nivel válidos son 01-49, 66, 77, 78 y 88. Los números de nivel 01 a 49 se utilizan para definir elementos de datos que pueden ser parte de una estructura jerárquica. El número de nivel 01 también se puede utilizar para definir una constante: un elemento con un valor inmutable especificado en el momento de la compilación.
8. Los niveles 66, 77, 78 y 88 tienen usos especiales: consulteSección 6.8 [Datos especiales Artículos], página 145, para más detalles.
9. Se espera que:
   A. Una sección de enlace debe aparecer únicamente dentro de un subprograma. Sin embargo, el compilador no impedirá su uso en un programa principal.
   B. Todos los elementos de datos de nivel 01 descritos dentro de la sección de vinculación de un subprograma deben aparecer en una DIVISIÓN DE PROCEDIMIENTO USANDO (verSección 7.1 [DIVISIÓN DE PROCEDIMIENTOS DE USO], página 244) o como argumentos en una declaración ENTRY.
   C. Cada elemento de datos de nivel 01 descrito dentro de la sección de enlace de un subprograma debe corresponder a un argumento pasado en una declaración CALL (verSección 7.8.5 [LLAMADA], página 290) o un argumento en una llamada de función al subprograma.
10. No especificar un identificador-1 o FILLER inmediatamente después del número de nivel tiene el mismo efecto que si se especificara FILLER. No se puede hacer referencia directa a un elemento de datos denominado FILLER; estos elementos se utilizan generalmente para especificar una parte no utilizada del almacenamiento total asignado a un elemento de grupo o para describir un elemento de grupo cuyo contenido solo se referenciará mediante los nombres de los elementos que pertenecen a él. En la sección de enlace, los elementos de datos de nivel 01 no se pueden denominar FILLER.
11. No se asigna almacenamiento para los datos definidos en la sección de enlace; las descripciones de datos que se encuentran allí son simplemente áreas de almacenamiento que se pasarán al subprograma mediante un programa que realiza la llamada. Por lo tanto, cualquier discusión sobre la inicialización predeterminada de dichos datos es irrelevante. Sin embargo, es posible asignar manualmente elementos de datos de la sección de enlace que no sean argumentos del subprograma mediante la declaración ALLOCATE (consulteSección 7.8.3 [ASIGNAR], página 287). En tales casos, la inicialización se realizará según la documentación de esa declaración.
12. La palabra reservada UNBOUNDED se puede utilizar para especificar una tabla ilimitada (una tabla con un número máximo ilimitado de ocurrencias) y se puede hacer referencia a ella en cualquier lugar donde se pueda hacer referencia a una tabla.
13. Un grupo sin límites es un grupo que contiene al menos una tabla sin límites. Se puede especificar SÓLO en la sección de vínculos y debe ser de tipo alfanumérico o nacional.
14. Puede hacer referencia a grupos ilimitados en la sintaxis COBOL en cualquier lugar donde se

pueda hacer referencia a un grupo alfanumérico o nacional, con las siguientes excepciones:

No se pueden especificar grupos ilimitados como argumento BY CONTENT en una instrucción CALL. No se pueden especificar grupos ilimitados como data-name-2 en la División de procedimientos

Frase de RETORNO. No se pueden especificar grupos ilimitados como argumentos para funciones intrínsecas, excepto como argumento para la función intrínseca LENGTH.

15. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.6 SECCIÓN DE INFORME

SECCIÓN DE INFORME Sintaxis

```
[SECCIÓN INFORME.
  ~~~~~~ ~~~~~~~
  { Descripción-del-Informe [ { Definición-del-Grupo-de-Informes } ]...      }...    ]
  {                         { 01-Nivel-Constante        }        }
  {                         { 78-Nivel-Constante        }        }
  { 01-Nivel-Constante                                           }
  { 78-Nivel-Constante                                           }
```

Sintaxis de descripción de informe (RD)

```
Nombre del informe RD [ IS GLOBAL ]
~~                    ~~~~~~
[ EL CÓDIGO ES literal-1 | identificador-1 ]
  ~~~~
[ { EL CONTROL ES                    } { FINAL  }...    ]
    ~~~~~~~            ~~~~~
  { LOS CONTROLES SON } { identificador-2 }
    ~~~~~~~~~
[ PÁGINA [ { EL LÍMITE ES                    } ] [ { literal-2  } LÍNEAS ]
  ~~~~   { ~~~~~        }       { identificador-3 } ~~~~
         { LOS LÍMITES SON }
            ~~~~~~
      [ literal-3 | identificador-4 COLUMNAS|COLS ]
                                    ~~~~~~~ ~~~~ [
      EL ENCABEZADO ES literal-4 | identificador-5 ]
        ~~~~~~~
      [ EL PRIMER DETALLE ES literal-5 | identificador-6 ]
        ~~~~~ ~~ ~~~~~~
      [ ÚLTIMO CH|{ENCABEZADO DE CONTROL} ES literal-6 | identificador-7 ]
        ~~~~ ~~    ~~~~~~~ ~~~~~~~
      [ EL ÚLTIMO DETALLE ES literal-7 | identificador-8 ]
        ~~~~ ~~ ~~~~~~
      [ PIE ES literal-8 | identificador-9 ] ] .
        ~~~~~~~
```

En esta sección se describe el diseño de los informes impresos, así como muchos de los aspectos funcionales de la generación de informes que se producirán mediante el sistema de control de Report Writer. Es importante mantener el orden de estas cláusulas y garantizar que todos los campos definidos o referenciados con esta sección estén realmente definidos en la SECCIÓN DE TRABAJO Y ALMACENAMIENTO y no en otro lugar.

1. Las palabras reservadas ARE e IS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las frases CONTROL IS y CONTROLS ARE son intercambiables, al igual que las frases LÍMITE DE PÁGINA y Frases LÍMITES DE PAGINA.

3. La palabra reservada LINES puede abreviarse como LINE.
4. La palabra reservada COLUMNAS puede abreviarse como COLS.
5. Cada informe al que se hace referencia en una cláusula REPORT IS (verSección 6.2.1 [Archivo/Ordenar-Descripción], página 123) debe describirse con una descripción del informe (RD).
6. VerSección 6.9.24 [GLOBAL], página 178, para obtener información sobre la opción GLOBAL.
7. Por favor veaSección 2.1.13 [Características del generador de informes], página 24, si aún no lo ha leído. Le permitirá familiarizarse con la terminología de Report Writer que aparece a continuación.
8. Las siguientes reglas se aplican a la cláusula PAGE LIMITS:
   A. Si no se especifica ninguna cláusula PAGE LIMITS, se generará todo el informe como si constara de una sola página de longitud arbitraria.
   B. Todos los literales (literal-2 a literal-8) deben ser numéricos con valores enteros positivos distintos de cero.
   C. Todos los identificadores (identificador-2 a identificador-8) deben ser numéricos, sin editar y con valores enteros positivos distintos de cero.
   D. Cualquier valor especificado para literal-2 o identificador-2 definirá el número total de líneas disponibles en cualquier página del informe, sin contar los márgenes no utilizados en la parte superior y/o inferior de la página (definidos por los valores LINES AT TOP y LINES AT BOTTOM especificados en la cláusula LINAGE del FD al que está vinculado este RD — consulteSección 6.2.1 [Archivo/Ordenar-Descripción], página 123).
   E. Cualquier valor especificado para literal-3 o identificador-3 será ignorado.
   F. La cláusula HEADING define el primer número de línea en el que se puede presentar un encabezado de informe o un encabezado de página.
   G. La cláusula FIRST DETAIL define la primera línea en la que se puede presentar un grupo de detalles.
   H. La cláusula LAST CONTROL HEADING define la última línea en la que se puede presentar cualquier línea de un encabezado de control.
   I. La cláusula LAST DETAIL define la última línea en la que se puede presentar cualquier línea de un grupo de detalles.
   J. La cláusula FOOTING define la última línea en la que se puede presentar cualquier línea de un grupo de zapata de control.
   K. Las siguientes reglas establecen valores predeterminados para las distintas cláusulas PAGE LIMIT, asumiendo que hay una:

      TÍTULO por defecto es uno (1)

      TÍTULO DEL PRIMER DETALLE
      Se utiliza el valor

      ÚLTIMO ENCABEZADO DE CONTROL
      valor de ÚLTIMO DETALLE o, si éste no está presente, el valor de PIE o, si éste también está presente, el valor de LÍMITE DE PÁGINA

      ÚLTIMO DETALLE
      valor de PIE DE PÁGINA o, si no está presente, el valor de PIE DE PÁGINA

      LÍMITE valor del ÚLTIMO DETALLE o, si no está presente, el valor del LÍMITE DE PÁGINA

   L. Para que los valores especificados en una cláusula PAGE LIMIT sean válidos, se deben cumplir todos los siguientes requisitos:

ser verdad:

- PRIMER DETALLE ≤ ENCABEZADO
- ÚLTIMO ENCABEZADO DE CONTROL ≤ PRIMER DETALLE

- ÚLTIMO DETALLE ≤ ÚLTIMO ENCABEZADO DE CONTROL
- PIE ≤ ÚLTIMO DETALLE

9. Las siguientes reglas se aplican a la cláusula CONTROL:
   A. Si no hay una cláusula CONTROL, el informe no contendrá interrupciones de control; esto implicaque no puede haber grupos de informes de ENCABEZAMIENTO DE CONTROL o PIE DE CONTROL definidos para este RD.
   B. Incluya la palabra reservada FINAL si desea incluir un encabezado de control especial antes de que se genere la primera línea de detalle (ENCABEZADO DE CONTROL FINAL) o después de que se genere la última línea de detalle (PIE DE CONTROL FINAL).
   C. Si especifica FINAL, debe ser el primer salto de control nombrado en el RD.
   D. Cualquier especificación de identificador-9 incluida en la cláusula CONTROL hace referencia a nombres de datos definidos en cualquier sección de división de datos, excepto la sección de informe.
   E. Debe haber un grupo de informes de ENCABEZAMIENTO DE CONTROL y/o PIE DE CONTROL definido en la sección de informe para cada identificador-9.
   F. En el momento de la ejecución:
      - Cada vez que se ejecuta una declaración GENERATE (verSección 7.8.20 [GENERAR], página 333) se ejecuta contra un grupo de informes detallados definido para este RD, el RWCS verificará el contenido de cada elemento de datos del identificador-2; siempre que el valor de un identificador-9 haya cambiado desde el GENERATE anterior, una condición de interrupción de control estará en vigencia para ese identificador-2.
      - Una vez determinada la lista de cortes de control, se presentará el PIE DE CONTROL para cada identificador-2 que tenga un corte de control (si se define dicho grupo de informes).
      - A continuación, se presentará el ENCABEZADO DE CONTROL para cada identificador-2 que tenga un salto de control (si se define dicho grupo de informes).
      - Los grupos de informes CONTROL FOOTING y CONTROL HEADING se presentarán en la secuencia en la que aparecen enumerados en la cláusula CONTROL.
      - Sólo después de que se haya realizado este procesamiento se presentará el grupo de informes detallados especificado en GENERAR.
10. Cada RD tendrá asignado lo siguiente:
   A. El registro especial PAGE-COUNTER (verArtículo 7.7 [Registros Especiales], página 260), que contendrá el número de página del informe actual.
      - Este registro se establecerá en un valor de 1 cuando se ejecute una instrucción INITIATE (consulteSección 7.8.25 [INICIAR], página 343) se ejecuta para el informe y se incrementará en 1 cada vez que RWCS inicie una nueva página del informe.
      - Las referencias a PAGE-COUNTER dentro de la sección de informe se calificarán implícitamente con el nombre del informe al que pertenece el grupo de informes que hace referencia al registro.
      - Las referencias a PAGE-COUNTER en la división de procedimientos deben calificarse con el nombre de informe apropiado si hay varios RD definidos.
   B. El registro especial LINE-COUNTER, que contendrá el número de línea actual en la página actual.
11. El RD debe ir seguido de al menos una definición de grupo de informes de nivel 01.

6.6.1 Definiciones de grupos de informes

Sintaxis de definición de grupo de informes

```
01 [ identificador-1 ]

[ EL NÚMERO DE LÍNEA ES { entero-1 [ [ EN LA SIGUIENTE PÁGINA ] } ]
  ~~~~                                ~~~~ ~~~~
                    { +|MÁS entero-1                    }
                         ~~~~
                    {EN LA SIGUIENTE PAGINA             }
                          ~~~~ ~~~~
[ EL SIGUIENTE GRUPO ES { [ +|MÁS ] entero-2 } ]
  ~~~~ ~~~~~            ~~~~
            { SIGUIENTE | { SIGUIENTE PÁGINA } | PÁGINA }
                     ~~~~   ~~~~ ~~~~    ~~~~
[ EL TIPO ES { RH | { ENCABEZADO DEL INFORME }          } ]
  ~~~~         ~~   ~~~~~~ ~~~~~~~~
            { PH|{ENCABEZADO DE PÁGINA}                 }
               ~~   ~~~~ ~~~~~~~~
            { CH|{ENCABEZADO DE CONTROL} FINAL|identificador-2  }
               ~~   ~~~~~~~ ~~~~~~~~    ~~~~~
            { DE|DETALLE                                }
               ~~ ~~~~~~
            { CF|{PIE DE CONTROL} FINAL|identificador-2 }
               ~~   ~~~~~~~ ~~~~~~~~    ~~~~~
            { PF|{PIE DE PÁGINA}                        }
                ~~ ~~~~ ~~~~~~~~
            { RF|{PIE DEL INFORME}                      }
               ~~   ~~~~~~ ~~~~~~~~
.     [ INFORME-SECCIÓN-Elemento-de-datos ]...
```

La sintaxis que se muestra aquí documenta cómo se define un grupo de informes en un informe. Esta sintaxis es válida únicamente en la sección de informes y únicamente después de un RD.

1. Las palabras reservadas IS, NUMBER y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Los términos RH y ENCABEZADO DE INFORME son intercambiables, al igual que PH y ENCABEZADO DE PÁGINA, CH
   y ENCABEZAMIENTO DE CONTROL, DE y DETALLE, CF y PIE DE CONTROL, PF y PIE DE PÁGINA como así como RF y REPORT FOOTING.
3. El grupo de informes que se está definiendo será parte del RD codificado más recientemente.
4. El TIPO (verSección 6.9.55 [TIPO], página 221) La cláusula especifica el tipo de grupo de informes que se está definiendo.
5. El número de nivel utilizado para una definición de grupo de informes debe ser 01.
6. La especificación del identificador opcional 1 asigna un nombre a este grupo de informes para que se pueda hacer referencia al grupo mediante una declaración GENERATE o en un USE BEFORE REPORTING.
7. No se pueden nombrar dos grupos de informes en el mismo informe (RD) con el mismo identificador-1. Sin embargo, puede haber múltiples definiciones de identificador-1 en

diferentes informes. En tales casos, las referencias al identificador-1 deben estar calificadas por el nombre del informe.

8. Solo puede haber un encabezado de informe, un pie de informe, un encabezado de control final, un pie de control final, un encabezado de página y un pie de página definidos por informe.
9. Las declaraciones de grupos de informes deben ir seguidas de al menos un elemento de datos de REPORT-SECTION
   con un número de nivel en el rango 02-49.
10. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.6.2 SECCIÓN DEL INFORME Elementos de datos

Sintaxis de elemento de datos de sección de informe

```
número de nivel [ identificador-1 ]

[EN BLANCO CUANDO ES CERO ]
   ~~~~~          ~~~~
[ COLUMNA [ { EL NÚMERO ES   } ] [ +|MÁS ] entero-1 ]
   ~~~          { ~~~~~~~        }          ~~~~
              { LOS NÚMEROS SON }
                  ~~~~~~~
[ GRUPO
INDICADO ]
   ~~~~~ ~~~~~~~~
[ DERECHO JUSTIFICADO ]
   ~~~~
[ EL NÚMERO DE LÍNEA ES { entero-2 [ [ EN LA SIGUIENTE PÁGINA ] } ]
   ~~~~                { +|MÁS entero-2 ~~~~ ~~~~  }
                           ~~~~
                       {EN LA SIGUIENTE PAGINA                }
                             ~~~~ ~~~~
[ OCURRE [ entero-3 A ] entero-4 VECES
   ~~~~~~                  ~~
    [ DEPENDE DEL identificador-2 ]
        ~~~~~~~~~~
      [ PASO entero-5 ]
        ~~~~
      [ VARIAR identificador-3 DE { identificador-4 } POR { identificador-5 } ]
        ~~~~~~~~                  ~~~~ { entero-6       } ~~ { entero-7
                                  }[ LA IMAGEN ES una cadena de
imágenes ]
   ~~~
[ PRESENTECUANDO condición-nombre ]
   ~~~~~~~ ~~~~
[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ]
   ~~~~       ~~~~~~~ ~~~~~~~~      ~~~~~~~~
[ { FUENTE ES literal-1|identificador-6 [ REDONDEADO ]           } ]
      ~~~~~~                                ~~~~~~~
   { SUMA DE { identificador-7 }...  [ { REINICIAR EN FINAL|identificador-8 } ] }
   { ~~~     { literal-2       }          { ~~~~~    ~~~~~                }     }
   { EL VALOR ES [ TODOS ] literal-3                                        { Identificador UPON-9
                                        }                                   }
      ~~~~~          ~~~                      ~~~~
.    [ INFORME-SECCIÓN-Elemento-de-datos ]...
```

Las descripciones de elementos de datos que describen las líneas y los campos del informe que conforman la esencia de un grupo de informes siguen inmediatamente a la definición de ese grupo.

1. Las palabras reservadas IS, NUMBER, OF, ON, RIGHT, TIMES y WHEN (BLANK) son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.

2. La palabra reservada COLUMNA puede abreviarse como COL.
3. La palabra reservada JUSTIFICADO puede abreviarse como JUST.

4. La palabra reservada PICTURE puede abreviarse como PIC.
5. La FUENTE (verSección 6.9.49 [FUENTE], página 214), SUMA (verSección 8.1.92 [SUMA], página 524) y VALOR (verSección 6.9.61 [VALOR], página 239) Las cláusulas, válidas únicamente sobre un elemento elemental, son mutuamente excluyentes entre sí.
6. Los elementos de grupo (aquellos sin cláusulas PICTURE) se utilizan con frecuencia para describir líneas completas de un informe, mientras que los elementos elementales (aquellos con una cláusula picture) se utilizan con frecuencia para describir campos específicos de información en el informe. Cuando se utiliza esta convención de codificación, los elementos de grupo tendrán LINE (consulteSección 6.9.30 [LÍNEA], página 185) cláusulas y ninguna COLUMNA (verSección 6.9.14 [COLUMNA], página 167) mientras que los elementos elementales se especificarán al revés.
7. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.7 SECCIÓN DE PANTALLA

Sintaxis de elemento de datos de sección de pantalla

```
número de nivel [ identificador-1 | RELLENO ]
                                    ~~~~~~

[ AUTO | SALTO AUTOMÁTICO | TERMINACIÓN AUTOMÁTICA | TABULACIÓN ]
  ~~~~   ~~~~~~~~~~   ~~~~~~~~~~~~~~
         ~~~[ CAMPANA | PITIDO ]
  ~~~~   ~~~~
[ COLOR-DE-FONDO| COLOR-DE-FONDO ES entero-1 | identificador-2 ]
  ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~
[ COLOR-DE-PRIMERO|COLOR-DE-PRIMERO ES entero-3 | identificador-4 ]
  ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~ [ EN BLANCO { LÍNEA
| PANTALLA } ]
  ~~~~~   ~~~~ ~~~~~~~
[BORRAR { EOL|EOS                    } ]
  ~~~~~   ~~~ ~~~
[         { [HASTA EL FINAL DE ] {LÍNEA | PANTALLA } } ]
                ~~~          ~~~~   ~~~~~~
[ INICIAL ]
  ~~~~~~~
[ BLANCOCUANDO CERO ] [ DERECHO JUSTIFICADO ]
  ~~~~~        ~~~~       ~~~~
[ PARPADEO ] [ RESALTAR | BAJA LUZ ]
  ~~~~~      ~~~~~~~~~   ~~~~~~~~
[ VIDEO INVERSO | INVERSO | INVERTIDO ]
  ~~~~~~~~~~~~~   ~~~~~~~   ~~~~~~~~
[ COLUMNA | EL NÚMERO DE POSICIÓN ES [ { +|MÁS  } ] entero-2 | identificador-3
]
  ~~~        ~~~~~~~~                    ~~~~
[                                [ { -|MENOS } ]                          ]
                                       ~~~~~
[ CURSOR { identificador-10 } ]
  ~~~~~~
[ DE literal-1 | identificador-5 ]
  ~~~~
[ Identificador TO-5              ]
  ~~
[ USANDO identificador-5          ]
  ~~~~~
[ { EL VALOR ES [ TODOS ] literal-1  } ]
    ~~~~~       ~~~
[ COMPLETO | LONGITUD-VERIFICAR ] [ OBLIGATORIO | VACÍO-VERIFICAR ]
  ~~~~     ~~~~~~~~~~~~       ~~~~~~~~
           ~~~~~~~~~~~[SIN ECO | SIN ECO | APAGADO |
SEGURO ]
  ~~~~~~~   ~~~~~~~   ~~~   ~~~~~~
[ LÍNEA IZQUIERDA ] [ LÍNEA SUPERIOR ] [ SUBRAYADO ]
```

~~~~~~~~ ~~~~~~~~ ~~~~~~~~~

[EL NÚMERO DE LÍNEA ES [{ +|MÁS }] entero-4 | identificador-6]
~~~~~~~~

```
   ~~~                          ~~~~
[                       [ { -|MENOS } ]                          ]
                                ~~~~~
[ OCURRE entero-5
VECES ]
   ~~~~~~
[ LA IMAGEN ES una cadena de imágenes ]
   ~~~
[ PROMPT [ CARÁCTER ES literal-2 | identificador-7 ]
   ~~~~~~     ~~~~~~~~~
[ REQUERIDO | CHEQUE VACÍO ]
   ~~~~~~~~     ~~~~~~~~~~~
[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ] .
   ~~~~       ~~~~~~~ ~~~~~~~~      ~~~~~~~~
```

———

La sección de pantalla describe las pantallas que se mostrarán durante la E/S de terminal/consola.

1. Las palabras reservadas CHARACTER (cláusula SEPARATE), IS, NUMBER, RIGHT, TIMES y WHEN son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La palabra reservada COLUMNA puede abreviarse como COL.
3. La palabra reservada PICTURE puede abreviarse como PIC.
4. Los siguientes conjuntos de palabras reservadas son intercambiables:
   - AUTO, AUTO-SALTO y AUTOTERMINACIÓN
   - COLOR DE FONDO y COLOR DE FONDO
   - CAMPANA y PITIDO
   - COLOR DE PRIMER PLANO y COLOR DE PRIMER PLANO
   - COMPROBACIÓN COMPLETA Y DE LONGITUD
   - REQUERIDO y CHEQUE VACÍO
   - SEGURO y SIN ECO
5. Los elementos de datos definidos en la sección de pantalla describen los diseños de pantalla de entrada, salida o combinación que se utilizarán con la declaración de elemento de datos ACCEPT (consulteSección 7.8.1.4 [ACEPTAR elemento de datos], página 267) o la declaración del elemento de datos DISPLAY (verSección 7.8.12.4 [MOSTRAR elemento de datos], página 305) declaraciones. Estos diseños de pantalla pueden definir toda el área de pantalla disponible o cualquier subconjunto de ella.
6. El término área de pantalla disponible es confuso en aquellos entornos en los que se invocan sesiones de línea de comandos dentro de un entorno de interfaz gráfica de usuario, como será el caso en Windows, OSX y la mayoría de los sistemas Unix/Linux: estos entornos permiten que existan ventanas de sesión de línea de comandos con un número variable de filas y columnas de pantalla disponibles. Cuando diseñe pantallas GnuCOBOL, debe hacerlo teniendo en cuenta la geometría lógica de filas y columnas de pantalla en las que se ejecutará el programa.
7. Los elementos de datos con números de nivel 01 (Constantes), 66, 78 y 88 se pueden utilizar en la sección de pantalla; tienen la misma sintaxis, reglas y uso que en las otras secciones de división de datos.
8. Sin LINEA (verSección 6.9.30 [LÍNEA], página 185) o COLUMNA (verSección 6.9.14 [COLUMN], página 167) cláusulas, los campos de la sección de pantalla se mostrarán en la ventana de la consola comenzando en cualquier coordenada de línea/columna que se indique o implique en la declaración del elemento de datos ACCEPT o DISPLAY que presenta

el elemento de pantalla. Después de que se presenta un campo en la ventana de la consola, el siguiente campo se presentará inmediatamente después de ese campo.

9. Una cláusula LINE explícitamente establecida en la definición de un elemento de datos de la sección de pantalla anulará cualquier cláusula LINE incluida en la declaración ACCEPT o DISPLAY que presente ese elemento de datos en la pantalla. Lo mismo sucede con las cláusulas COLUMN.
10. Las teclas Tab y Back-Tab (Shift-Tab en la mayoría de los teclados) posicionarán el cursor de un campo a otro en la secuencia de línea/columna en la que aparecen los campos en la pantalla en el momento de la ejecución, independientemente de la secuencia en la que se definieron en la sección de pantalla.
11. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.
12. Consulte los detalles del elemento de datos ACEPTAR para obtener más información sobre el uso de CURSOR.
13. El uso de la frase TAB obliga al usuario a finalizar el verbo ACCEPT con una tecla de terminación. Este es el modo estándar para GnuCOBOL y otros compiladores, pero no para RM COBOL. TAB no se puede utilizar junto con las cláusulas AUTO / AUTO-SKIP y AUTOTERMINATE.

6.8 Elementos de datos especiales

6.8.1 01-Constantes de nivel

Sintaxis de constante de nivel 01

```
01 nombre-constante-1 CONSTANTE [ ES GLOBAL ]
                      ~~~~~~~~~   ~~~~~~
   { CO { literal-1                          } }
   { M  { expresión aritmética 1             } }
     O
   {    { { LONGITUD DE  } DE { identificad  } } }
          BYTE                  or-1
   {    { { ~~~~~~~~~~~  }    { nombre de uso } } }
   {    { { LONGITUD     }                    } }
   {        ~~~~~~                             }
   { DE   CDF-nombre-                          }
          variable-1
     ~~~~
```

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, TRABAJO-ALMACENAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, PANTALLA.

La constante de nivel 01 es uno de los cinco tipos de constantes de tiempo de compilación que se pueden declarar dentro de un programa. Los otros cuatro tipos son >>DEFINE Directiva CDF (verSección 3.4 [>>DE- BIEN], página 67) constantes, directiva >>SET CDF (verSección 3.6 [>>SET], página 70) constantes,Constantes de 78 niveles (verSección 6.8.4 [Elementos de datos de nivel 78], página 151, y expresión aritmética-1).

1. Las palabras reservadas AS, IS y OF son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. VerSección 6.9.24 [GLOBAL], página 178, para obtener información sobre la opción GLOBAL.
3. Este tipo particular de declaración de constante proporciona la capacidad de determinar la longitud de un elemento de datos o el tamaño de almacenamiento asociado con un USO numérico particular (verSección 6.9.59 [USO], página 228) tipo — algo que no es posible con los otros tipos de constantes.
4. Las constantes definidas de esta manera se vuelven indefinidas una vez que se encuentra un FIN DE PROGRAMA o FIN DE FUNCIÓN en la fuente de entrada.
5. Las descripciones de datos de este formato en realidad no asignan ningún almacenamiento: simplemente definen un nombre (constant-name-1) que puede usarse en cualquier lugar donde pueda usarse un literal numérico (consulte las opciones BYTE-LENGTH o LENGTH) o un literal del mismo tipo que literal-1.
6. No se puede hacer referencia al nombre constante-nombre-1 en una directiva CDF.
7. Se debe tener cuidado de que el nombre-constante-1 no duplique ningún otro nombre de elemento de datos que se haya definido en el programa, ya que las referencias a ese nombre de elemento de datos se referirán a la constante y no al elemento de datos. El compilador GnuCOBOL no emitirá una advertencia sobre esta condición.
8. El valor especificado para el nombre de uso puede ser cualquier USO que no utilice una IMAGEN (verSección 6.9.37 [IMAGEN], página 195) cláusula. Estas serían cualquiera de las

siguientes: BINARY-C-LONG, BINARY-CHAR, BINARY-DOUBLE, BINARY-LONG, BINARY-SHORT, COMP-1 (o COMPUTATIONAL-1), COMP-2 (o COMPUTATIONAL-2), FLOAT-DECIMAL-16,FLOAT-DECIMAL-34, FLOAT-LARGO, FLOAT-CORT, PUNTERO o PUNTERO DE PROGRAMA.

9. La cláusula BYTE-LENGTH producirá un valor numérico para el nombre-constante-1 idéntico al que devolvería la función intrínseca BYTE-LENGTH ejecutada contra el identificador-1 o un elemento de datos declarado con un USAGE de nombre-de-uso.
10. La cláusula LENGTH producirá un valor numérico para el nombre-constante-1 idéntico al que devolvería la función intrínseca LENGTH ejecutada contra el identificador-1 o un elemento de datos declarado con un USAGE de nombre-de-uso.

A continuación se muestran ejemplos de uso de la opción expresión-aritmética

```
78 wVALOR CONSTANTE 2 *          (23  + 3) + (10 / 2).
01 wCONST2 constante (23 + 3)**2.
01 wCONST3 constante (12 + wCONST2)**2 - wCONST.
78  wCONST4v a l o r        (12 + wCONST3)**2 - wCONST2.
```

A continuación se muestra otro ejemplo del uso de la opción expresión-aritmética.

```
PROGRAMA-ID. TESTCONST.
DIVISIÓN AMBIENTE. DIVISIÓN
DATOS.
SECCIÓN DE TRABAJO Y
ALMACENAMIENTO. 01 A
          CONSTANTE 2.
01 B      CONSTANTE ( (3 + 2) * A).
01 CON CONSTANTE (A ** B).
01 MIS DATOS PIC
X(CON).
DIVISIÓN DE
PROCEDIMIENTOS.
CON PANTALLA
  ACEPTAR omitido
  MOVER "12345678901234567890123456789 0" A MIS DATOS
    VISUALIZAR MIS DATOS
  ACEPTAR omitió
  GOBACK.
```

Aquí se muestra la lista de un programa GnuCOBOL que utiliza constantes de nivel 01 para mostrar la longitud (en bytes) de los distintos tipos de uso sin imágenes.

```
DIVISIÓN DE IDENTIFICACIÓN.
IDENTIFICACIÓN DEL
PROGRAMA. Longitudes de
uso. DIVISIÓN DE DATOS.
SECCIÓN DE TRABAJO-ALMACENAMIENTO.
01   Len-BINARIO-C-LARGO    CONSTANT CO  LONGIT  DE  BINARIO-C-LARGO.
                                E    MO    UD
01   Len-BINARIO-CARACTER   CONSTANT CO  LONGIT  DE  CARACTER BINARIO.
                                E    MO    UD
01   Len-BINARIO-DOBLE      CONSTANT CO  LONGIT  DE  BINARIO-DOBLE.
                                E    MO    UD
01   Len-BINARIO-LARGO      CONSTANT CO  LONGIT  DE  BINARIO-LARGO.
                                E    MO    UD
01   Len-BINARIO-CORTO      CONSTANT CO  LONGIT  DE  BINARIO-CORTO.
```

```
                                E       MO    UD
01   Len-COMP-1                 CONSTANT CO  LONGIT  DE  COMP-1.
                                E       MO    UD
01   Len-COMP-2                 CONSTANT CO  LONGIT  DE  COMP-2.
                                E       MO    UD
01   Len-FLOTANTE-              CONSTANT CO  LONGIT  DE  FLOTANTE-DECIMAL-
     DECIMAL-16                 E       MO    UD        16.
01   Len-FLOTANTE-              CONSTANT CO  LONGIT  DE  FLOTANTE-DECIMAL-
     DECIMAL-34                 E       MO    UD        34.
01   Len-FLOTANTE-LARGO         CONSTANT CO  LONGIT  DE  FLOTADOR-LARGO.
                                E       MO    UD
01   Len-FLOTADOR-CORTO         CONSTANT CO  LONGIT  DE  FLOTADOR-CORT.
                                E       MO    UD
```

```
01    Justo-PUNTERO                CONSTANTE  SEGÚN LA LONGITUD DEL PUNTERO.
01    Len-PROGRAMA-PUNTERO CONSTANTE COMO LONGITUD DEL PUNTERO
DEL PROGRAMA.DIVISIÓN DE PROCEDIMIENTOS.
000-Principal.
      PANTALLA "En este sistema, con esta compilación de GnuCOBOL,
      los USOS sin IMÁGENES "PANTALLA" tienen estas longitudes (en
      bytes): "PANTALLA" "
      MOSTRAR "BINARIO-C-LARGO:  " Len-BINARIO-C-
      LARGOMOSTRAR "BINARY-CHAR:    " Len-BINARY-
      CHAR PANTALLA "BINARY-DOUBLE:" Len-BINARIO-
      DOBLE PANTALLA "BINARIO-LARGO:  " Len-BINARIO-
      LARGO PANTALLA "BINARIO-CORT: "Pantalla corta
      Len-BINARIO" COMP-1:            " Len-COMP-1
      PANTALLA "COMP-2:                   "   Len-COMP-2
      MOSTRAR "FLOAT-DECIMAL-16: " Len-FLOAT-DECIMAL-
      16 MOSTRAR "FLOAT-DECIMAL-34: " Len-FLOAT-
      DECIMAL-34 MOSTRAR "FLOAT-LONG:       " Len-FLOAT-
      PANTALLA LARGA "FLOAT-CORTO:  " Len-FLOAT-SHORT
      DISPLAY "PUNTERO:                   " Len-PUNTER
      MOSTRAR"PUNTERO DEL PROGRAMA:"            Len-
      PROGRAMA-PUNTERO DETENER EJECUTAR.
```

La salida de este programa, en un sistema Windows 7 con una compilación MinGW de 32 bits de Gnu-COBOL es:

```
En este sistema, con esta compilación de GnuCOBOL, los
USOS sin IMÁGENES tienen estas longitudes (en bytes):

BINARIO-C-LARGO:     4
CARACTER BINARIO:    1
BINARIO-DOBLE:       8
BINARIO-LARGO:       4
BINARIO-CORTO:       2
COMP-1:              4
COMP-2:              8
FLOTANTE-DECIMAL-    8
16:
FLOTANTE-DECIMAL-    16
34:
FLOTADOR-LARGO:      8
FLOTADOR-CORTO:      4
PUNTERO:             4
PUNTERO DEL          4
PROGRAMA:
```

La salida de este programa, en un sistema Linux X64 que ejecuta cobc (GnuCOBOL) 3.1.2.0 es: En

este sistema, con esta compilación de GnuCOBOL, el
Los USOS sin IMÁGENES tienen estas longitudes (en bytes):

```
BINARIO-C-LARGO:     8
CARACTER             1
```

```
BINARIO:
BINARIO-DOBLE:          8
```

BINARIO-LARGO:	4
BINARIO-CORTO:	2
COMP-1:	4
COMP-2:	8
FLOTANTE-DECIMAL-16:	8
FLOTANTE-DECIMAL-34:	16
FLOTADOR-LARGO:	8
FLOTADOR-CORTO:	4
PUNTERO:	8
PUNTERO DEL PROGRAMA:	8

Descubre las diferencias entre 32 y 64 bits.

6.8.2 Elementos de datos de nivel 66

Sintaxis de elemento de datos de 66 niveles

```
66 identificador-1RENOMBRAN identificador-2 [ HASTA|HASTA identificador-3 ].
```

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

Un elemento de datos de 66 niveles reagrupa elementos definidos previamente al especificar agrupaciones alternativas, posiblemente superpuestas, de elementos de datos elementales.

1. Las palabras reservadas THRU y THROUGH son intercambiables.
2. Un elemento de datos de nivel 66 no puede cambiar el nombre de un elemento de datos de nivel 66, nivel 01, nivel 77 o nivel 88.
3. Puede haber varios elementos de datos de nivel 66 que cambien el nombre de los elementos de datos contenidos en la misma descripción de registro de nivel 01.
4. Todas las entradas de RENAMES asociadas con un registro lógico deben seguir inmediatamente a las de ese registro.última entrada de descripción de datos.

6.8.3 Elementos de datos de nivel 77

Sintaxis de elemento de datos de nivel 77

```
77 identificador-1 [ ES GLOBAL|EXTERNO ]
                     ~~~~~~ ~~~~~~~~
[ BASADO ]
  ~~~~~
[EN BLANCO CUANDO ES CERO ]
  ~~~~~       ~~~~
[ DERECHO JUSTIFICADO ]
  ~~~~
[ LA IMAGEN ES una cadena de imágenes ]
  ~~~
[ REDEFINE identificador-5 ]
  ~~~~~~~~~~
[ EL SIGNO ESTÁ AL IZQUIERDO | AL FINAL [ CARÁCTER SEPARADO ] ]
  ~~~~     ~~~~~~~ ~~~~~~~~     ~~~~~~~~
[ SINCRONIZADO | SINCRONIZADO [ IZQUIERDA | DERECHA ] ]
  ~~~~           ~~~~             ~~~~ ~~~~~~
[ EL USO ES data-item-usage ]
  ~~~~~
[ EL VALOR ES [ TODOS ] literal-1 ].
  ~~~~~       ~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

Esta sintaxis es válida en las siguientes secciones: ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

La intención de un elemento de nivel 77 es poder crear un elemento de datos elemental independiente.

1. Las palabras reservadas CHARACTER, IS, RIGHT (JUSTIFIED) y WHEN son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La palabra reservada JUSTIFIED puede abreviarse como JUST, la palabra reservada PICTURE puede abreviarse como PIC y las palabras reservadas SYNCHRONIZED y SYNCHRONISED pueden abreviarse como SYNC.
3. Los programas nuevos que requieran un elemento elemental independiente deben codificarse para utilizar un número de nivel de 01 en lugar de 77.
4. VerSección 6.9 [Cláusulas de descripción de datos], página 154, para obtener información sobre el uso de las distintas cláusulas de descripción de datos.

6.8.4 Elementos de datos de nivel 78

Sintaxis de constante de nivel 78

```
78 nombre-constante-1 EL VALOR ES
                      ~~~~~
  { entero-1                    } [ { +|-|*|/|** } { entero-3                    } ]
  { identificador-1             } [ { Y           } { identificador-3             } ]
  { literal-1                   } [ { ~~~         } { literal-3                   } ]
  { expresión-aritmética-1      } [ { O           } { expresión-aritmética-3      } ]
  { LONGITUD DE { identificador-2 } } [ { ~~ } { LONGITUD DE { identificador-4 } } ]
  { ~~~~~~   { literal-2      } } [                { ~~~~~~   { literal-4      } } ]

  { INICIO DE identificador-5        }
    ~~~~~
  { PRÓXIMO }
    ~~~~
```

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, TRABAJO-ALMACENAMIENTO, ALMACENAMIENTO LOCAL, ENLACE, PANTALLA

La constante de 78 niveles es uno de los cuatro tipos de constantes de tiempo de compilación que se pueden declarar dentro de un programa. Los otros tres tipos son >>DEFINE Directiva CDF (verSección 3.4 [>>DE- BIEN], página 67) constantes, directiva >>SET CDF (verSección 3.6 [>>SET], página 70) constantes y constantes de nivel 01 (verSección 6.8.1 [Constantes de nivel 01], página 145).

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Las constantes definidas de esta manera se vuelven indefinidas una vez que se encuentra un FIN DE PROGRAMA o FIN DE FUNCIÓN en la fuente de entrada.
3. Las descripciones de datos de este formato en realidad no asignan ningún almacenamiento: simplemente definen un nombre (constant-name-1) que puede usarse en cualquier lugar donde se pueda usar un literal del mismo tipo que literal-1.
4. No se puede hacer referencia al nombre constante-nombre-1 en una directiva CDF.
5. Se debe tener cuidado de que el nombre-constante-1 no duplique ningún otro nombre de elemento de datos que se haya definido en el programa, ya que las referencias a ese nombre de elemento de datos se referirán a la constante y no al elemento de datos. El compilador GnuCOBOL no emitirá una advertencia sobre esta condición.
6. Consulte 6.8.1 para ver ejemplos de uso de la opción expresión-aritmética.
7. NEXT proporciona el valor entero que representa el desplazamiento en el que el siguiente byte de almacenamiento sigue a la declaración de datos anterior.

El siguiente programa de ejemplo muestra algunas definiciones:

```
DIVISIÓN DE
IDENTIFICACIÓN.
PROGRAMA-ID. TEST78.
DIVISIÓN DE DATOS.
SECCIÓN DE TRABAJO-ALMACENAMIENTO.
VALOR 78 AAA (100 -1) + 100.
```

```
01 K                FOTO X(19).
```

```
01 UN ESPACIO DE VALOR PIC X(05) EXTERNO.
01 R PIC EXTERNO X(08) ESPACIO DE VALOR.
78 BBB VALOR LONGITUD DE A.
78 CCC VALOR LONGITUD DE '1234'.
78 DDD VALOR LONGITUD DE '1234567890' / LONGITUD DE A.
78 EEE VALOR LONGITUD DE '123'     * 3.
78 FFF VALOR INICIO DE R.
01 B EXTERNO.
    05 C      PIC X(26) VALOR 'A'.
    05 D      PIC X(21) VALOR 'A'.
    78 GGG                    VALORP R Ó X I M O .
    78 HH ...                 VALOR INICIO DE A.
    05METRO      Foto X(23)VALOR 'B'.
DIVISIÓN DE
      PROCEDIMIENTO.
      PANTALLA 'AAA= ' AAA
      PANTALLA 'BBB= ' BBB
      PANTALLA 'CCC= ' CCC
      PANTALLA 'DDD= '
      DDD PANTALLA 'EEE= '
      EEE PANTALLA 'FFF= '
      FFF PANTALLA 'GGG= '
      GGG PANTALLA 'HHH=
      ' HHH DETENER
      EJECUTAR.
```

6.8.5 Elementos de datos de nivel 88

Sintaxis de elemento de datos de 88 niveles

```
88 condición-nombre-1 { VALOR ES } {literal-1 [ HASTA|HASTA literal-2 ]}...
                      { ~~~~~~      }            ~~~~ ~~~~~~~~
                      { LOS VALORES SON }
                        ~~~~~~~

  [CUANDO SE ESTABLECE COMO FALSO ES literal-3 ] .
                  ~~~~~~
```

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

Los nombres de condición son elementos de datos booleanos (es decir, VERDADERO / FALSO) que reciben sus valores VERDADERO y FALSO según los valores del elemento de datos que no es de nivel 88 cuya definición siguen inmediatamente.

1. Las palabras reservadas ARE, IS, SET y TO son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas THRU y THROUGH son intercambiables.
3. Los nombres de condición siempre se definen subordinados a otro elemento de datos (que no sea de nivel 88). Ese elemento de datos debe ser un elemento elemental. Siempre que el elemento de datos principal asuma uno de los valores especificados en el VALOR del elemento de nivel 88 (consulteSección 6.9.61 [VALOR], página 239) cláusula, condition-name-1 tomará el valor de TRUE.
4. Los nombres de condiciones no ocupan ningún espacio de almacenamiento.
5. La cláusula opcional THROUGH permite especificar un rango de posibles valores VERDADEROS.
6. Siempre que el elemento de datos principal asuma cualquier valor excepto uno de los valores especificados en *condición-nombre-1*En la cláusula VALUE de 's, condition-name-1 tomará el valor FALSE.
7. Al ejecutar la instrucción SET condition-name-1 TO TRUE, el elemento de datos padre de condition-name-1 tomará el primer valor especificado en la cláusula VALUE de condition-name-1.
8. Al ejecutar la instrucción SET condition-name-1 TO FALSE, el elemento de datos padre de condition-name-1 tomará el valor especificado en la cláusula FALSE de condition-name-1. Si condition-name-1 no tiene una cláusula FALSE, la instrucción SET (consulteSección 7.8.43 [SET], página 379) generará un mensaje de error en el momento de la compilación.
9. VerSección 2.1.27.1 [Nombres de las condiciones], página 44, para obtener más información.

6.9 Cláusulas de descripción de datos

6.9.1 CUALQUIER LONGITUD

Sintaxis del atributo ANY LENGTH

CUALQUIER LONGITUD
~~~ ~~~~~~

——

Esta sintaxis es válida en las siguientes secciones: ENLACE

Los elementos de datos declarados con el atributo ANY LENGTH no tienen una longitud fija en tiempo de compilación. Dichos elementos solo se pueden definir en la sección de enlace de un subprograma, ya que solo pueden servir como descripciones de argumentos de subrutina. Estos elementos deben tener una IMAGEN (consulteSección 6.9.37 [IMAGEN], página 195) cláusula que especifica exactamente un símbolo A, X, U o 1.

1. CUALQUIER LONGITUD y BASADA (verSección 6.9.8 [BASED], página 161) no se pueden usar juntas en la misma descripción de elemento de datos. La cláusula ANY LENGTH especifica que la longitud del elemento de datos se determinará en tiempo de ejecución, el tipo se determina (y algún día se verifica con EC-PROGRAM-ARGS) mediante el símbolo de imagen.

   Se determinan verificando la definición del llamador, que por lo tanto DEBE ser un módulo GnuCOBOL o un programa C que use funciones de API para crear campos COBOL.
~~~

6.9.2 CUALQUIER NUMÉRICO

Sintaxis del atributo ANY NUMERIC

CUALQUIER NUMÉRICO
~~~ ~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: ENLACE

Los elementos de datos declarados con el atributo ANY NUMERIC no tienen una longitud de compilación fija. Dichos elementos solo se pueden definir en la sección de enlace de un subprograma, ya que solo pueden servir como descripciones de argumentos de subrutina. Estos elementos deben tener una IMAGEN (consulteSección 6.9.37 [IMAGEN], página 195) cláusula que especifica exactamente un símbolo 9. La cláusula ANY NUMERIC especifica que la longitud y el uso del elemento de datos se determinarán en tiempo de ejecución.

Se determinan verificando la definición del llamador, que por lo tanto DEBE ser un módulo GnuCOBOL o un programa C que use funciones de API para crear campos COBOL.

1. El CUALQUIER NUMÉRICO y BASADO (verSección 6.9.8 [BASED], página 161) Las cláusulas no se pueden utilizar juntas en la misma descripción de elemento de datos.
~~~

6.9.3 AUTO

Sintaxis del atributo AUTO

AUTO
~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Un campo cuya descripción incluye este atributo hará que el cursor avance automáticamente al siguiente campo con entrada habilitada de una pantalla si el campo está completamente lleno con datos de entrada.

1. El AUTO, AUTO-SKIP (verSección 6.9.4 [AUTO-SKIP], página 157) y AUTOTERMINAR (ver Sección 6.9.5 [TERMINACIÓN AUTOMÁTICA], página 158) Las cláusulas son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
~~~~

6.9.4 SALTO AUTOMÁTICO

Sintaxis del atributo AUTO-SKIP

SALTO AUTOMÁTICO
~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Un campo cuya descripción incluye este atributo hará que el cursor avance automáticamente al siguiente campo con entrada habilitada de una pantalla si el campo está completamente lleno con datos de entrada.

1. El AUTO (verSección 6.9.3 [AUTO], página 156), SALTO AUTOMÁTICO y TERMINACIÓN AUTOMÁTICA (verSección 6.9.5 [TERMINACIÓN AUTOMÁTICA], página 158) Las cláusulas son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
~~~~~~~~~

6.9.5 TERMINACIÓN DEL COCHE

Sintaxis del atributo AUTOTERMINATE

TERMINACIÓN DEL COCHE
~~~~~~~~~~~~~

———

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Un campo cuya descripción incluye este atributo hará que el cursor avance automáticamente al siguiente campo con entrada habilitada de una pantalla si el campo está completamente lleno con datos de entrada.

1. El AUTO (verSección 6.9.3 [AUTO], página 156), SALTO AUTOMÁTICO (verSección 6.9.4 [AUTO- SALTAR], página 157) y AUTOTERMINATE son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
~~~~~~~~~~~~~

6.9.6 COLOR DE FONDO

Sintaxis del atributo BACKGROUND-COLOR

COLOR DE FONDO|COLOR DE FONDO ES entero-1 | identificador-1
~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula se utiliza para especificar el color de fondo de la pantalla del elemento de datos de pantalla o el color de fondo de pantalla predeterminado de los elementos subordinados si se utiliza en un elemento de grupo.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Las palabras reservadas COLOR DE FONDO y COLOR DE FONDO son intercambiables.
3. Puede especificar los colores mediante un número (0-7) o utilizando los nombres de constantes proporcionados en el
   Libro de copias screenio.cpy (incluido con todas las distribuciones de código fuente de GnuCOBOL).
4. Los colores también se pueden especificar utilizando un identificador numérico no editado cuyo valor esté en el rango de 0 a 7.

Para los DISPLAY compuestos, los atributos siempre se aplican solo al elemento fuente anterior, pero lo siguiente también permite un cambio por variable o literal, es decir

```
Nombre para mostrar:"  FONDO-COLOR COB-AMARILLO
            NOMBRE-VAR      COLOR DE FONDO  NEGRO
            COB
PANTALLA FINAL
```

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~~~~~~~~~

6.9.7 ANTES DE TIEMPO

Sintaxis del atributo BEFORE TIME

```
ANTES DE TIEMPO    {identificador-1}
        ~~~~  {literal-1       }
```

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula es funcionalmente igual que la cláusula TIMEOUT. El valor del identificador-1 o literal-1 especifica el tiempo que debe transcurrir antes de que finalice automáticamente la instrucción ACCEPT cuando el usuario no ingresa ningún dato.

6.9.8 BASADO

Sintaxis del atributo BASED

BASADO
~~~~~

Esta sintaxis es válida en las siguientes secciones: ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

A los elementos de datos declarados con BASED no se les asigna almacenamiento en el momento de la compilación. En el momento de la ejecución, ALLOCATE (verSección 7.8.3 [ASIGNAR], página 287) o ESTABLECER DIRECCIÓN (verSección 7.8.43.3 [ESTABLECER DIRECCIÓN], página 381) se utilizan para asignar espacio y (opcionalmente) inicializar dichos elementos.

1. La BASE y CUALQUIER LONGITUD (verSección 6.9.1 [CUALQUIER LONGITUD], página 154) Las cláusulas no se pueden utilizar juntas en la misma descripción de elemento de datos.
2. La cláusula BASED sólo se puede utilizar en elementos de datos de nivel 01 y nivel 77.
~~~~~

6.9.9 BIP

Sintaxis del atributo BEEP

BIP
~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

1. El BEEP y el BELL (verSección 6.9.10 [BELL], página 163) Las cláusulas son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
2. Utilice esta cláusula para que se produzca un tono audible cuando se muestre el elemento de la pantalla.
~~~~

6.9.10 CAMPANA

Sintaxis del atributo BELL

CAMPANA
~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

1. El BEEP (verSección 6.9.9 [BEEP], página 162) y las cláusulas BELL son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
2. Utilice esta cláusula para que se produzca un tono audible cuando se muestre el elemento de la pantalla.
~~~~

6.9.11 BLANCO

Sintaxis del atributo BLANK

LÍNEA EN BLANCO|PANTALLA
~~~~~ ~~~~ ~~~~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula dejará en blanco toda la pantalla (PANTALLA EN BLANCO) o sólo la línea en la que se van a mostrar los datos (LÍNEA EN BLANCO).

1. Las áreas en blanco tendrán sus colores de primer plano y de fondo establecidos según los atributos del campo que contiene la cláusula BLANK.
2. Esta cláusula es útil cuando un elemento de una sección de pantalla se muestra encima de uno mostrado anteriormente.
~~~~~

6.9.12 EN BLANCO CUANDO ES CERO

Sintaxis del atributo BLANK-WHEN-ZERO

BLANCO CUANDOC E R O
~~~~~ ~~~~

———————————————————————————————————————

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

Esta cláusula hará que el valor de ese elemento se transforme automáticamente en espacios si alguna vez se MUEVE un valor de 0 al elemento.

1. La palabra reservada WHEN es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Esta cláusula solo se puede utilizar en un elemento de datos PIC 9 con un USO (verSección 6.9.59 [EE. UU.- EDAD], página 228) de PANTALLA.
~~~~~

6.9.13 PARPADEAR

Sintaxis del atributo BLINK

PARPADEAR
~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula BLINK modifica la apariencia visual del campo mostrado haciendo que el contenido del campo parpadee.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~

6.9.14 COLUMNA

Sintaxis de la cláusula COLUMNA (SECCIÓN DEL INFORME)

```
{ COLUMNA } [ NÚMERO      ] [ IZQUIERDA       ] [ ES ] { [ +|MÁS ] entero-1 }
{ ~~~~~~ } [ NÚMEROS ] [ ~~~~                 ] [ SON ] {        ~~~~     }
{ COL    }                [ DERECHA         ]                              { {entero-2 } ...
         }
  ~~~                       ~~~~~
{ COLUMNAS }              [ CENTRO ]
  ~~~~~~~                   ~~~~~~
{ COLUMNAS }
  ~~~~
```

Sintaxis de la cláusula COLUMNA (SECCIÓN DE PANTALLA)

```
EL NÚMERO DE COLUMNA ES [ { +|MÁS      } ] entero-2 | identificador-3
~~~                     ~~~~
                   [ { -|MENOS } ]
                       ~~~~~
```

Esta sintaxis es válida en las siguientes secciones: INFORME, PANTALLA

La cláusula COLUMN proporciona los medios para indicar en qué columna se debe presentar un campo en la ventana de la consola (sección de pantalla) o en un informe (sección de informe).

1. Las palabras reservadas ARE, IS, NUMBER y NUMBERS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La palabra reservada COLUMNA puede abreviarse como COL.
3. La ubicación de la línea de una sección de informe o de un campo de sección de pantalla estará determinada por la LÍNEA
   (verSección 6.9.30 [LÍNEA], página 185) cláusula.
4. El valor de entero-1 debe ser 1 o mayor.
5. Si se utiliza el identificador-1 para especificar una posición de columna absoluta o relativa, el identificador-1 debe definirse como un elemento numérico de cualquier USO (consulteSección 6.9.59 [USO], página 228) distinto de COMPUTATIONAL-1 o COMPUTATIONAL-2, sin editar símbolos. El valor del identificador-1 en el momento en que se presenta el elemento de datos de la pantalla debe ser 1 o mayor. Tenga en cuenta que el compilador aceptará un identificador COMPUTATIONAL-1 o COMPUTATIONAL-2, pero producirá resultados impredecibles en tiempo de ejecución.
6. La coordenada de la columna de un campo se puede indicar de forma absoluta (es decir, COLUMNA 5) o de forma relativa en función del final del campo presentado anteriormente (es decir, COLUMNA MÁS 1).
7. Si se desea, se puede utilizar el símbolo "+" en lugar de la palabra PLUS; sin embargo, si se utiliza el símbolo "+", debe haber al menos un espacio que lo separe del entero 1. Si no se incluye este espacio, el símbolo "+" se considerará simplemente parte del entero 1 y la cláusula COLUMN se considerará una especificación de columna absoluta en lugar de

relativa.

8. El uso del posicionamiento de columna relativo (COLUMN PLUS) tiene un comportamiento ligeramente diferente según la sección en la que se utiliza la cláusula, como se indica a continuación:

A. Cuando se utiliza en un elemento de datos de una sección de informe, COLUMN PLUS posicionará el inicio del valor del nuevo campo de manera tal que haya columnas en blanco con un entero 1 entre el final del campo anterior y el inicio de este campo.

   Si la descripción de un elemento de datos del informe incluye la FUENTE (verSección 6.9.49 [FUENTE], página 214), SUMA (verSección 8.1.92 [SUM], página 524) o VALOR (verSección 6.9.61 [VALOR], página 239) pero no tieneCláusula COLUMNA, se asumirá COLUMNA MÁS 1.

B. Cuando se utiliza en un elemento de datos de una sección de pantalla, COLUMN PLUS posicionará el nuevo campo de manera que comience exactamente un carácter entero o un carácter de identificador después del último carácter del campo anterior. Por lo tanto, COLUMN PLUS 1 no dejará posiciones en blanco entre el final del campo anterior y el comienzo de este.

   Si la descripción de un elemento de datos de pantalla incluye el FROM (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220), USANDO (verSección 6.9.60 [USO], página 238) o VALOR (verSección 6.9.61 [VALOR], página 239) pero no tiene cláusula COLUMN, el nuevo campo de pantalla comenzará en la coordenada de columna del último carácter del campo anterior.

6.9.15 CONSTANTE

Sintaxis del atributo CONSTANT

CONSTANTE
~~~~~~~~

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, TRABAJO-ALMACENAMIENTO, ALMACENAMIENTO LOCAL, ENLACE, PANTALLA

Esta opción significa que el elemento de datos de nivel 01 en cuya declaración se especifica CONSTANT se tratará como un nombre simbólico para un valor literal, utilizable dondequiera que se pueda usar un literal del tipo apropiado.

1. El valor de un elemento de datos definido como una constante no se puede modificar en tiempo de ejecución. De hecho, no es sintácticamente aceptable utilizar dicho elemento de datos como campo de destino de ninguna instrucción de división de procedimiento que almacene un valor.
2. VerSección 6.8.1 [Constantes de nivel 01], página 145, para obtener información adicional.
~~~~~~~~

6.9.16 POR DEFECTO

Sintaxis del atributo DEFAULT

POR DEFECTO
~~~~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula DEFAULT mostrará los datos existentes antes de permitir que el usuario los actualice. La cláusula DEFAULT es sinónimo de la cláusula UPDATE).
~~~~~~~

6.9.17 CHEQUE VACÍO

Sintaxis del atributo EMPTY-CHECK

CHEQUE VACÍO
~~~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula obliga al usuario a ingresar datos en el campo en el que están especificados (o en todos los campos subordinados con capacidad de entrada si se especifica EMPTY-CHECK en un elemento del grupo).

1. El CHECK VACÍO y REQUERIDO (verSección 6.9.43 [REQUERIDO], página 207) Las cláusulas son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
2. Para que tenga efecto, el usuario primero debe mover el cursor al campo que tiene esta cláusula en su definición.
3. La declaración del elemento de datos ACCEPT (verSección 7.8.1.4 [ACEPTAR elemento de datos], página 267) ignorará la tecla Enter y cualquier otra pulsación de tecla que haga que el cursor se mueva a otro elemento de la pantalla a menos que se hayan ingresado datos en el campo. Las teclas de función aún podrán finalizar la operación ACEPTAR.
4. Para que funcione, este atributo debe ser compatible con el paquete "curses" subyacente con el que se creó la implementación de GnuCOBOL. A partir de este momento, el paquete "PDCurses" (usado para compilaciones nativas de Windows o MinGW) no admite EMPTY-CHECK.
~~~~~~~~~~~

6.9.18 BORRAR

Sintaxis de la cláusula ERASE

BORRAR EOL|EOS
~~~~~ ~~~ ~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

ERASE borrará el contenido de la pantalla desde la ubicación donde se mostrará el elemento de datos de pantalla cuya descripción contiene esta cláusula, hasta el final de la pantalla (ERASE EOS)

1. Las áreas borradas tendrán sus colores de primer plano y de fondo establecidos según los atributos del campo que contiene la cláusula ERASE.
2. Esta cláusula es útil cuando un elemento de una sección de pantalla se muestra encima de uno mostrado anteriormente.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~

6.9.19 EXTERNO

Sintaxis del atributo EXTERNAL

EXTERNO
~~~~~~~~

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL

Esta cláusula marca una descripción del elemento de datos, FD o SD, consulteSección 6.2.1 [Archivo/Ordenar-Descripción], página 123,como compartible con otros programas ejecutados desde el mismo hilo de ejecución.

1. Al especificar la cláusula EXTERNAL en un FD o SD, la descripción del archivo se puede compartir entre todos los programas ejecutados desde el mismo hilo de ejecución, siempre que se codifique una cláusula EXTERNAL con la descripción del archivo en cada programa que la requiera. Esta compartición permite que el archivo se abra, lea y/o escriba y cierre en diferentes programas. Esta compartición se aplica también a las descripciones de registros subordinadas a la descripción del archivo.
2. Al especificar la cláusula EXTERNAL en la descripción de un elemento de datos, dicho elemento puede compartirse entre todos los programas ejecutados desde el mismo hilo de ejecución, siempre que el elemento de datos esté codificado (con una cláusula EXTERNAL) en cada programa que lo requiera.
3. Los siguientes puntos se aplican a la especificación de EXTERNO en la definición de un elemento de datos:
   A. La cláusula EXTERNAL sólo se puede especificar en el nivel 77 o 01.
   B. Un elemento EXTERNO debe tener un nombre de datos y ese nombre no puede ser RELLENO.
   C. EXTERNO no se puede combinar con BASADO (verSección 6.9.8 [BASED], página 161), GLOBAL (verSección 6.9.24 [GLOBAL], página 178) o REDEFINE (verSección 6.9.41 [REDEFINES], página 205).
~~~~~~~~

6.9.20 FALSO

Sintaxis de la cláusula FALSE

CUANDO SE ESTABLECE COMO FALSO ES literal-1
~~~~~

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

Esta cláusula, que solo puede aparecer en la definición de un nombre de condición de nivel 88, se utiliza para especificar el valor del elemento de datos que sirve como padre del nombre de condición de nivel 88 que obligará al nombre de condición a asumir un valor FALSO.

1. Las palabras reservadas IS, SET, TO y WHEN son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. VerSección 6.8.5 [Elementos de datos de nivel 88], página 153, o VerSección 2.1.27.1 [Nombres de las condiciones], página 44, para obtener más información.
~~~~~

6.9.21 COLOR DE PRIMER PLANO

Sintaxis del atributo FOREGROUND-COLOR

COLOR DE PRIMER PLANO|COLOR DE PRIMER PLANO ES entero-1 | identificador-1
~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula se utiliza para especificar el color del texto dentro de un elemento de datos de pantalla o el color de texto predeterminado de los elementos subordinados si se utiliza en un elemento de grupo.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Las palabras reservadas FOREGROUND-COLOR y FOREGROUND-COLOUR son intercambiables.
3. Puede especificar los colores mediante un número (0-7) o utilizando los nombres de constantes proporcionados en el libro de copias screenio.cpy (que se proporciona con todas las distribuciones de código fuente de GnuCOBOL).
4. Los colores también se pueden especificar utilizando un identificador numérico no editado cuyo valor esté en el rango de 0 a 7.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~~~~~~~~~

6.9.22 DE

Sintaxis de la cláusula FROM

DESDE literal-1 | identificador-5
~~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula se utiliza para especificar el elemento de datos del cual un campo de sección de pantalla obtendrá su valor cuando se muestra la pantalla, o un literal que especificará el valor de ese mismo campo.

1. El DE, HASTA (verSección 6.9.54 [TO], página 220), USANDO (verSección 6.9.60 [USO], página 238) y VALOR (verSección 6.9.61 [VALOR], página 239) Las cláusulas son mutuamente excluyentes en la definición de cualquier elemento de datos de la sección de pantalla.
~~~~

6.9.23 LLENO

Sintaxis del atributo FULL

LLENO
~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula FULL obliga al usuario a ingresar datos en el campo en el que está especificado (o en todos los campos subordinados con capacidad de entrada si se especifica en un elemento de grupo) suficientes para llenar cada posición de carácter del campo.

1. El control COMPLETO y de LONGITUD (verSección 6.9.29 [VERIFICACIÓN DE LONGITUD], página 184) Las cláusulas son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
2. Para que esta cláusula tenga efecto en el momento de ejecución, el usuario debe mover el cursor al campo que tiene esta cláusula en su definición.
3. La declaración del elemento de datos ACCEPT (verSección 7.8.1.4 [ACEPTAR elemento de datos], página 267) ignorará la tecla Enter y cualquier otra pulsación de tecla que mueva el cursor y que pueda hacer que el cursor se mueva.para pasar a otro elemento de la pantalla a menos que se haya ingresado la cantidad adecuada de datos en el campo. Sin embargo, las teclas de función aún podrán finalizar la operación ACEPTAR.
4. Para que este atributo sea funcional, debe ser compatible con el paquete "curses" subyacente con el que se creó la implementación de GnuCOBOL. A partir de este momento, el paquete "PDCurses" (usado para compilaciones nativas de Windows o MinGW) no es compatible con FULL.
~~~~

6.9.24 GLOBAL

Sintaxis del atributo GLOBAL

GLOBAL
~~~~~~

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, INFORME

Esta cláusula marca un elemento de datos, constante de nivel 01, FD (verSección 6.2.1 [Archivo/Ordenar-Descripción], página 123), SD (verSección 6.2.1 [Archivo/Ordenar-Descripción], página 123) o un RD (verSección 6.6 [SECCIÓN DEL INFORME], página 135) como compartible con cualquier subprograma anidado.

1. Al especificar la cláusula GLOBAL en la descripción de un archivo o un informe, esa descripción puede ser compartida entre un programa y cualquier subprograma anidado dentro de él, siempre que el FD, SD o RD esté codificado (con una cláusula GLOBAL) en cada subprograma anidado que lo requiera. Esta compartición permite que el archivo se abra, lea y/o escriba y cierre o que el informe se inicie o finalice en esos programas. Los programas compilados por separado no pueden compartir una descripción de archivo GLOBAL, pero pueden compartir una EXTERNAL (consulteSección 6.9.19 [EXTERNO], página 173) descripción del archivo. Esta compartición se aplica a las descripciones de registros subordinadas a la descripción del archivo y también a los grupos de informes subordinados al RD.
2. Al especificar la cláusula GLOBAL en la descripción de un elemento de datos, dicho elemento de datos puede compartirse entre un programa y cualquier subprograma anidado dentro de él, siempre que el elemento de datos esté codificado (con una cláusula GLOBAL) en cada programa que lo requiera.
3. Los siguientes puntos se aplican a la especificación de GLOBAL en la definición de un elemento de datos:
   A. La cláusula GLOBAL sólo se puede especificar en el nivel 77 o 01.
   B. Un elemento GLOBAL debe tener un nombre de datos y ese nombre no puede ser FILLER.
   C. GLOBAL no se puede combinar con EXTERNO (verSección 6.9.19 [EXTERNO], página 173), REDEFINE (verSección 6.9.41 [REDEFINE], página 205) o BASADO (verSección 6.9.8 [BASED], página 161).
~~~~~~

6.9.25 GRUPO INDICA

Sintaxis del atributo GROUP-INDICATE

GRUPOI N D I C A R
~~~~~ ~~~~~~~~

——

Esta sintaxis es válida en las siguientes secciones: INFORME

La cláusula GROUP INDICATE especifica que el elemento de datos en cuya definición aparece la cláusula se presentará solo en circunstancias muy limitadas.

1. Esta cláusula solo puede aparecer dentro de un grupo de informes DETAIL (consulteSección 6.9.55 [TIPO], página 221).
2. Cuando esté presente esta cláusula, el dato en cuestión se presentará únicamente en las siguientes circunstancias:
   A. En la primera presentación del grupo de detalle posterior al INICIADO (verSección 7.8.25 [INICIAR], página 343) del informe.
   B. En la primera presentación del grupo de detalles después de iniciar cada nueva página.
   C. En la primera presentación del grupo de detalle después de que se produzca cualquier interrupción de control.
~~~~~

6.9.26 DESTACAR

Sintaxis del atributo HIGHLIGHT

DESTACAR
~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula controla la intensidad del texto (COLOR DE PRIMER PLANO (verSección 6.9.21 [COLOR DEL PRIMER PLANO], página 175)) configurando esa intensidad en el valor más alto de tres posibles.

1. Esta cláusula, junto con LOWLIGHT (verSección 6.9.32 [LOWLIGHT], página 188), tienen como objetivo proporcionar un esquema de intensidad de tres niveles (LOWLIGHT... nada (Normal)... HIGHLIGHT).

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~~

6.9.27 JUSTIFICADO

Sintaxis del atributo JUSTIFICADO

DERECHO JUSTIFICADO
~~~~

———————————————————————————————————————

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

La presencia de una cláusula JUSTIFIED RIGHT en la definición de un elemento de datos altera la forma en que se almacenan los datos en el campo, pasando del comportamiento predeterminado "justificado a la izquierda, con espacios en blanco" al comportamiento "justificado a la derecha, con espacios en blanco". A menos que utilice alguno de los dialectos de IBM, NO tiene ningún efecto sobre el contenido inicial de una variable, por ejemplo:

01 UNA IMAGEN X(12) VALOR JUSTO CORRECTO 'ABC'.

Se mostrará el contenido como 'ABC' sin ninguna justificación.

1. La palabra reservada RIGHT es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. La palabra reservada JUSTIFICADO puede abreviarse como JUST.
3. Esta cláusula es válida únicamente para elementos de datos alfabéticos (PIC A) o alfanuméricos (PIC X).
4. La presencia o ausencia de esta cláusula influye en el comportamiento del MOVE (verSección 7.8.30 [MOVE], página 354) así como la declaración FROM (verSección 6.9.22 [DESDE], página 176), FUENTE (verSección 6.9.49 [FUENTE], página 214) y USANDO (verSección 6.9.60 [USO], página 238) cláusulas de descripción de elementos de datos.
5. Si el valor que se almacena en el campo tiene la misma longitud que el campo receptor, la presencia o ausencia de la cláusula JUSTIFIED RIGHT en la descripción de ese campo es irrelevante.
6. Los siguientes ejemplos ilustran el comportamiento de la presencia y ausencia de la cláusula JUSTIFIED RIGHT cuando el tamaño del campo es diferente al del valor que se está almacenando. En estos ejemplos, el símbolo b representa un espacio.

Cuando el valor es más corto que el tamaño del campo:

SinJUSTIFICADO	**Con**JUSTIFICADO
01 A FOTO X(6).MOVER 'ABC' A A	01 UNA FOTO X(6) JUSTIFICADA A LA DERECHA.MOVER 'ABC' A A
Resultado	**Resultado**
~~~~

ABCbbb *bbbb*abecedario

Yo

Cuando el valor es más largo que el tamaño del campo:

SinJUSTIFICADO	**Con**JUSTIFICADO
01 Una foto X(6).	01 UNA IMAGEN X(6) JUSTIFICADA
MOVER 'ABCDEFGHI' A A	A LA DERECHA. MUEVA 'ABCDEFGHI' A A
Resultado	**Resultado**
A B C D E F	DEFGHI

6.9.28 LÍNEA IZQUIERDA

Sintaxis del atributo LEFTLINE

LÍNEA IZQUIERDA
~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula LEFTLINE introducirá una línea vertical en el borde izquierdo de un campo de pantalla.

1. LA LÍNEA IZQUIERDA, LÍNEA SUPERIOR (verSección 6.9.36 [OVERLINE], página 194) y SUBRAYADO (ver Sección 6.9.57 [SUBRAYADO], página 226) Las cláusulas pueden usarse en cualquier combinación en la descripción de un solo campo.
2. Esta cláusula es esencialmente no funcional cuando se utiliza dentro del shell de comandos de Windows.(cmd.exe) entornos y programas en ejecución compilados utilizando una implementación de GnuCOBOL creada con "PDCurses" (como las compilaciones de Windows/MinGW).
3. El funcionamiento de esta cláusula en sistemas Cygwin o UNIX/Linux/OSX dependerá de las capacidades de los atributos de video de los controladores de salida del terminal y del software "curses" que se utilice.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~

6.9.29 COMPROBACIÓN DE LONGITUD

Sintaxis del atributo LENGTH-CHECK

COMPROBACIÓN DE LONGITUD
~~~~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula LENGTH-CHECK obliga al usuario a ingresar datos en el campo en el que están especificados (o en todos los campos subordinados con capacidad de entrada si se especifican en un elemento de grupo) suficientes para llenar cada posición de carácter del campo.

1. El COMPLETO (verSección 6.9.23 [COMPLETA], página 177) y las cláusulas LENGTH-CHECK son intercambiables y no pueden usarse juntas en la misma descripción de elemento de datos.
2. Para que esta cláusula tenga efecto en el momento de ejecución, el usuario debe mover el cursor al campo que tiene esta cláusula en su definición.
3. La declaración del elemento de datos ACCEPT (verSección 7.8.1.4 [ACEPTAR elemento de datos], página 267) ignorará la tecla Enter y cualquier otra pulsación de tecla que mueva el cursor y que pueda hacer que el cursor se mueva.para pasar a otro elemento de la pantalla a menos que se haya ingresado la cantidad adecuada de datos en el campo. Sin embargo, las teclas de función aún podrán finalizar la operación ACEPTAR.
4. Para que funcione, este atributo debe ser compatible con el paquete "curses" subyacente con el que se creó la implementación de GnuCOBOL. A partir de este momento, el paquete "PDCurses" (usado para compilaciones nativas de Windows o MinGW) no admite LENGTH-CHECK.
~~~~~~~~~~~~

6.9.30 LÍNEA

Sintaxis de la cláusula LINE (SECCIÓN DE INFORME)

```
{ LÍNEA    [ NÚMERO ]  [ ES  ] }  { entero-2 [ [EN    PRÓXI PÁGI ] }
                                                       MO   NA
{ ~~~~~~   [ NÚMEROS ] [ SON ] }  {                    ~~~~ ~~~~   }
{ LAS                          }  { +|PLUS entero-2                } ...
  LINEAS
  SON
```

Sintaxis de la cláusula LINE (SCREEN SECTION)

```
EL NÚMERO DE LÍNEA ES [ { +|MÁS       } ] entero-4 | identificador-6
~~~                   ~~~~
                [ { -|MENOS } ]
                      ~~~~~
```

Esta sintaxis es válida en las siguientes secciones: INFORME, PANTALLA

Esta cláusula proporciona un medio para indicar explícitamente en qué línea se debe presentar un campo en la ventana de la consola (sección de pantalla) o en un informe (sección de informe).

1. Las palabras reservadas IS, NUMBER y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Los siguientes puntos documentan el uso del formato 1 de la cláusula LINE:
   A. La ubicación de la columna de un elemento del informe estará determinada por la COLUMNA (verSección 6.9.14 [COLUMNA], página 167) cláusula.
   B. El valor de entero-1 debe ser 1 o mayor.
   C. El número de línea del informe en el que se presentará el elemento de datos que contiene esta cláusula junto con cualquier elemento de datos subordinado se puede indicar en forma absoluta (es decir, LÍNEA 5) o en forma relativa en función de la línea mostrada anteriormente (es decir, LÍNEA MÁS 1).
   D. Se puede utilizar el símbolo '+' en lugar de la palabra PLUS, si se desea; sin embargo, si se utiliza '+',Debe haber al menos un espacio que lo separe del entero 1. Si no se incluye este espacio, el '+' se tratará simplemente como parte del entero 1 y la cláusula LINE se tratará como una especificación de línea absoluta en lugar de relativa.
   E. La cláusula opcional NEXT PAGE especifica que, independientemente de si el grupo de informes que contiene esta cláusula cabe o no en la página del informe que se está generando actualmente, el grupo de informes se verá obligado a aparecer en una nueva página.
3. Los siguientes puntos documentan el uso del formato 2 de la cláusula LINE:
   A. La ubicación de la columna de un campo de sección de pantalla está determinada por la COLUMNA (verSección 6.9.14 [COLUMNA], página 167) cláusula.
   B. El valor de entero-1 debe ser 1 o mayor.
   C. Si se utiliza el identificador-1 para especificar una posición de columna absoluta o relativa, el

identificador-1

*1*debe definirse como un elemento numérico de cualquier USO (verSección 6.9.59 [USO],

página 228) distinto de COMPUTATIONAL-1 o COMPUTATIONAL-2, sin símbolos de edición. El valor del identificador-1 en el momento en que se presenta el elemento de datos de la pantalla debe ser 1 o mayor. Tenga en cuenta que el compilador aceptará un identificador COMPUTATIONAL-1 o COMPUTATIONAL-2, pero producirá resultados impredecibles en tiempo de ejecución.

D. El número de línea de pantalla en la que se mostrará el elemento de datos que contiene esta cláusula junto con cualquier elemento de datos subordinado se puede indicar de forma absoluta (es decir, LÍNEA 5) o de forma relativa en función de la línea mostrada anteriormente (es decir, LÍNEA MÁS 1).

E. Si se desea, se puede utilizar el símbolo "+" en lugar de la palabra PLUS; sin embargo, si se utiliza "+", debe haber al menos un espacio que lo separe del entero 1. Si no se incluye este espacio, el "+" se considerará simplemente parte del entero 1 y la cláusula LINE se considerará una especificación de línea absoluta en lugar de relativa.

F. Si la descripción de un elemento de datos de pantalla incluye el FROM (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220), USANDO (verSección 6.9.60 [USO], página 238) o VALOR (verSección 6.9.61 [VALOR], página 239) pero no tiene cláusula LINE, se asumirá la "línea de pantalla actual".

6.9.31 MÁS BAJO

Sintaxis del atributo LOWER

MÁS BAJO
~~~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula proporciona un medio para indicar explícitamente que los datos aceptados estarán en minúsculas.
~~~~~

6.9.32 LUZ BAJA

Sintaxis del atributo LOWLIGHT

LUZ BAJA
~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula LOWLIGHT controla la intensidad del texto (COLOR DE PRIMER PLANO) estableciendo esa intensidad en su valor más bajo de tres posibles.

1. Esta cláusula, junto con HIGHLIGHT (verSección 6.9.26 [HIGHLIGHT], página 180), tienen como objetivo proporcionar un esquema de intensidad de tres niveles (LOWLIGHT . . . nada (Normal) . . . HIGHLIGHT). En entornos como una consola de Windows donde solo se admiten dos niveles de intensidad, LOWLIGHT es lo mismo que omitir esta cláusula por completo.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~

6.9.33 SIGUIENTE GRUPO

Sintaxis de la cláusula NEXT-GROUP

```
EL SIGUIENTE GRUPO ES { [ +|MÁS ] entero-2   }
   ~~~~ ~~~~~                ~~~~
     { SIGUIENTE | { SIGUIENTE PÁGINA } | PÁGINA }
                     ~~~~   ~~~~ ~~~~    ~~~~
```

Esta sintaxis es válida en las siguientes secciones: INFORME

Esta cláusula define las reglas sobre dónde comenzará el siguiente grupo que se presentará en un informe, línea por línea, con respecto a la última línea del grupo en el que aparece esta cláusula.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Los términos SIGUIENTE, SIGUIENTE PÁGINA y PÁGINA son intercambiables.
3. AEl grupo de informes debe contener al menos una cláusula LINE NUMBER para poder contener también una Cláusula SIGUIENTE GRUPO.
4. Si el RD (verSección 6.6 [SECCIÓN DEL INFORME], página 135) en los que el grupo de informes que contiene una cláusula NEXT GROUP no contiene una cláusula PAGE LIMITS, solo se puede especificar la opción PLUS entero 1.
5. La opción PÁGINA SIGUIENTE no se puede utilizar en un PIE DE PÁGINA.
6. La opción SIGUIENTE GRUPO no se puede especificar ni en un ENCABEZADO DE INFORME ni en un ENCABEZADO DE PÁGINA.
7. Los efectos de NEXT GROUP se sumarán a cualquier espacio entre líneas definido por la cláusula LINE NUMBER del siguiente grupo presentado.

6.9.34 NO-ECHO

Sintaxis del atributo NO-ECHO

NO-ECHO | NO ECHO | OFF
~~~~~~~ ~~ ~~~~ ~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula NO-ECHO hará que todos los datos ingresados en el campo aparezcan en la pantalla como espacios. La cláusula OFF es la misma que la cláusula NO-ECHO (y lo mismo que NO ECHO).

1. Si NO-ECHO está presente, se ignora la cláusula PROMPT.
2. Si la configuración del dialecto -fno-echo-means-secure está activa, todos los datos ingresados en el campo aparecerán en la pantalla como asteriscos. En este caso, las cláusulas NO-ECHO y SECURE son intercambiables.
3. El NO-ECHO y SECURE (verSección 6.9.46 [SEGURO], página 211) Las cláusulas no pueden usarse juntas en la misma descripción de elemento de datos.
4. Esta cláusula solo se puede utilizar en un campo que permita la entrada de datos (un campo que contenga USING (verSección 6.9.60 [USO], página 238) o A (verSección 6.9.54 [TO], página 220) cláusula).

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~

6.9.35 OCURRE

Sintaxis de la cláusula OCCURS

FORMATO GENERAL.

Formato 1 (tabla fija):

OCURRIR entero-2 [VECES]
~~~~~~

Formato 2 (se produce según la tabla):

OCURRE [entero-1 A] { entero-2 [VECES] } DEPENDE DE identificador-1
~~~~~~ ~~ { SIN LÍMITES } ~~~~~~~~~~

[CLAVE ASCENDENTE|DESCENDENTE ES identificador-5 ...] ...
~~~~~~~~~~ ~~~~~~~~~~

[INDEXADO POR nombre-índice-1 ...]
~~~~~~~SECC

IÓN DE INFORME.

Formato 3

OCURRE [entero-1 A] { entero-2 [VECES] } [DEPENDE DE identificador-1]
~~~~~~ ~~ ~~~~~~~~~

[PASO entero-3]
~~~~

[VARIAR identificador-2 DE { identificador-3 } POR { identificador-4 }]
~~~~~~~ ~~~~ { entero-4 } ~~ { entero-5 }

SECCIÓN DE PANTALLA.

Formato 4

OCURRE entero-2 VECES
~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA pero ILIMITADO es SÓLO para VINCULACIÓN.

La cláusula OCCURS se utiliza para crear una estructura de datos llamada tabla, donde las entradas de esa estructura se repiten varias veces.

1. Las palabras reservadas BY (INDEXED), IS, KEY, ON y TIMES son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.

2. La palabra reservada UNBOUNDED se puede utilizar para especificar una tabla ilimitada (una tabla con un número máximo ilimitado de ocurrencias) y se puede hacer referencia a ella en cualquier lugar donde se pueda hacer referencia a una tabla.
3. Un grupo sin límites es un grupo que contiene al menos una tabla sin límites. Se puede especificar SÓLO en la sección de vínculos y debe ser de tipo alfanumérico o nacional.
4. Puede hacer referencia a grupos ilimitados en la sintaxis COBOL en cualquier lugar donde se pueda hacer referencia a un grupo alfanumérico o nacional, con las siguientes excepciones:

   No se pueden especificar grupos ilimitados como argumento BY CONTENT en una instrucción CALL. No se pueden especificar grupos ilimitados como data-name-2 en la frase RETURNING de división de procedimiento. No se pueden especificar grupos ilimitados como argumentos para funciones intrínsecas, excepto como argumento para la función intrínseca LENGTH.
5. El valor de entero-2 especifica cuántas entradas se asignarán en la tabla.
6. El siguiente es un ejemplo de cómo se podría definir una tabla:

   ```
   05 TRIMESTRAL-LOS INGRESOS OCURREN 4 VECES PIC 9(7)V99.
   ```

   Esto asignará lo siguiente:

   ```
   INGRESOS TRIMESTRALES
   (1)              INGRESOS
   TRIMESTRALES          (2)
   INGRESOS TRIMESTRALES
   (3)INGRESOS
   TRIMESTRALES (4)
   ```

   Cada ocurrencia se referencia utilizando la sintaxis de subíndice (un literal numérico, una expresión aritmética o un identificador numérico entre paréntesis) que se muestra arriba.
7. La cláusula OCCURS también puede utilizarse a nivel de grupo, en cuyo caso se repite toda la estructura del grupo, de la siguiente manera:

   ```
   05 GRP OCURRE 3 VECES.
       10 A      FOTO X(1).
       10 B      FOTO X(1).
       10 C      FOTO X(1).
   ```

   Esto permitiría referencias a cualquiera de los siguientes:

   ```
   GRP(1) incluye A(1), B(1) y C(1) GRP(2)
   incluye A(2), B(2) y C(2) GRP(3) incluye
   A(3), B(3) y C(3)
   ```

   o cada elemento A,B,C podría referenciarse de la siguiente manera:

   ```
   A(1) carácter n.° 1 de GRP(1)
   B(1) carácter n.° 2 de GRP(1)
   C(1) carácter n.° 3 de GRP(1)
   A(2) carácter n.° 1 de GRP(2)
   B(2) carácter n.° 2 de GRP(2)
   C(2) carácter n.° 3 de GRP(2)
   A(3) carácter n.° 1 de GRP(3)
   B(3) carácter n.° 2 de GRP(3)
   C(3) carácter n.° 3 de GRP(3)
   ```
8. La cláusula opcional DEPENDING ON se puede agregar a una operación OCCURS para crear una tabla de longitud variable. En tales casos, el valor de entero-1 especifica cuál será el número mínimo de entradas en la tabla, mientras que entero-2 especifica el máximo. Dichas tablas se

asignarán al tamaño máximo especificado como entero-2. En el momento de la ejecución, el valor de identificador-1 determinará cuántos elementos de la tabla son accesibles.

9. Consulte la documentación de SEARCH (verSección 7.8.41 [BÚSQUEDA], página 376), BUSCAR TODO (verSección 7.8.42 [BUSCAR TODO], página 377) y SORT (verSección 7.8.44 [ORDENAR], página 388) declaraciones para explicaciones de las cláusulas KEY e INDEXED BY.
10. El estándar COBOL dice que la cláusula OCCURS no se puede especificar en una entrada de descripción de datos que tenga un número de nivel 01, 66, 77 u 88, aunque es válida en elementos de datos descritos subordinados a un elemento de datos de nivel 01. GnuCOBOL admite, como una extensión disponible en varios compiladores, la cláusula OCCURS en los niveles 01 y 77. Dependiendo de -std, esto puede degradarse a una advertencia o no tener un diagnóstico. Por ejemplo, -std=ibm-strict y
-std=cobol2002.
11. Los siguientes puntos se aplican a un OCCURS utilizado en la sección de informe:
    A. La cláusula STEP opcional define un valor de incremento que se agregará a cualquier LÍNEA absoluta (verSección 6.9.30 [LÍNEA], página 185) o COLUMNA (verSección 6.9.14 [COLUMNA], página 167) especificaciones numéricas que pueden ser parte o estar subordinadas a la definición de este elemento de datos.
    B. La cláusula opcional VARYING define un identificador que puede usarse como subíndice para las múltiples ocurrencias de este o cualquier elemento de datos subordinado si se especifica la FUENTE (verSección 6.9.49 [FUENTE], página 214) o SUMA (verSección 8.1.92 [SUM], página 524) Las cláusulas de este elemento de datos o de los elementos de datos subordinados hacen referencia a entradas dentro de la tabla. El elemento de datos del identificador 2 se crea dinámicamente según sea necesario y no se puede hacer referencia a él fuera del alcance de la definición del elemento de datos del informe.
    C. Los dos ejemplos siguientes ilustran dos formas diferentes en las que un informe podría incluir cifras de ventas correspondientes a cuatro trimestres en sus líneas de detalle: una haciendo las cosas "a la manera difícil" y otra utilizando las capacidades avanzadas de OCCURS de STEP y VARYING. Ambos suponen que la definición de la siguiente tabla existe en el almacenamiento de trabajo:

```
05 LAS VENTAS OCURREN 4 VECES FOTO 9(7)V99.
```

Primero, el "camino difícil":

```
10 COL 7   FOT (7) 9,99 $ FUENTE VENTAS(1
   UM        O                        ).
   NA
10 COL 17  FOT (7) 9,99 $ FUENTE VENTAS(2
   UM        O                        ).
   NA
10 COL 27  FOT (7) 9,99 $ FUENTE VENTAS(3
   UM        O                        ).
   NA
10 COL 37  FOT (7) 9,99 $ FUENTE VENTAS(4
   UM        O                        ).
   NA
```

Y luego usando STEP y VARYING:

```
10 COL 7     OCURRE 4 VECES PASO 10 VARIACIÓN TRIMESTRE DE 1
             POR 1 FOTO $(7)9.99 FUENTE VENTAS(TRIMESTRE).
```

6.9.36 SOBRELINEA

Sintaxis del atributo OVERLINE

SOBRELINEA
~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula OVERLINE introducirá una línea horizontal en el borde superior de un campo de pantalla.

1. LA LÍNEA IZQUIERDA (verSección 6.9.28 [LEFTLINE], página 183), SOBRELINEADO y SUBRAYADO (ver Sección 6.9.57 [SUBRAYADO], página 226) Las cláusulas pueden usarse en cualquier combinación en la descripción de un solo campo.
2. Esta cláusula es esencialmente no funcional cuando se utiliza dentro del shell de comandos de Windows.(cmd.exe) entornos y programas en ejecución compilados utilizando una implementación de GnuCOBOL creada con "PDCurses" (como las compilaciones de Windows/MinGW).
3. El funcionamiento de esta cláusula en sistemas Cygwin o UNIX/Linux/OSX dependerá de las capacidades de los atributos de video de los controladores de salida del terminal y del software "curses" que se utilice.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~

6.9.37 IMAGEN

Sintaxis de la cláusula PICTURE

LA IMAGEN ES una cadena de imágenes
~~~

———————————————————————————————————————

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

La cláusula de imagen define la clase (numérica, alfabética o alfanumérica), el tamaño y el formato de los datos que puede contener el elemento de datos que se está definiendo. A veces, esta función se ve facilitada por la cláusula USAGE (consulteSección 6.9.59 [USO], página 228) cláusula, y en algunos casos se asumirá completamente por esa cláusula.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. La palabra reservada PICTURE puede abreviarse como PIC. La mayoría de los programadores prefieren utilizar esta última.
3. Una cláusula de imagen sólo puede especificarse en un elemento elemental.
4. Una cadena de imágenes es una secuencia de los símbolos especiales '$', '*', '+', ',', '-', '.', '/', '0' (cero),
   '1', '9', 'A', 'B', CR, DB, 'S', 'V', 'P', 'X' y 'Z'.
5. En general, cada símbolo de imagen representa un solo carácter almacenado o un solo dígito decimal. Hay algunas excepciones, que se analizarán cuando sea necesario.
6. Cuando una cadena de imágenes contiene una secuencia repetida de símbolos (PIC 9999/99/99), por ejemplo, la repetición se puede especificar utilizando un recuento de repeticiones entre paréntesis, como en PIC 9(4)/9(2)/9(2). El uso de recuentos de repeticiones es opcional y su uso (o no) depende totalmente del usuario.a discreción del programador. Muchos programadores utilizan la repetición para secuencias pequeñas (PIC XXX) y el conteo de repeticiones para secuencias más grandes (PIC 9(9).
7. Este primer conjunto de símbolos de imagen define el tipo de datos básico de un elemento de datos. Cada símbolo representa el valor de almacenamiento de un solo carácter.

   'A' Define el almacenamiento reservado para un solo carácter alfabético ('A'-'Z', 'a'-'z').

   'NORTE' Define el almacenamiento reservado para un único carácter en el conjunto de caracteres nacionales de la computadora. El soporte para conjuntos de caracteres nacionales en GnuCOBOL actualmente está implementado solo parcialmente, y el efecto de compilación y ejecución de usar el símbolo de imagen 'N' es el mismo que si se hubiera codificado X(2), con el efecto adicional de que dicho campo se calificará como un campo NACIONAL o NACIONAL-EDITADO en un INICIALIZAR (verSección 7.8.24 [INICIALIZAR], página 339) declaración.

   'INCÓGNITA'Define el almacenamiento reservado para un solo carácter alfanumérico (cualquier carácter). '9' Define el almacenamiento reservado para un solo carácter de dígito numérico ('0'-'9').

   Por lo general, solo se utiliza un tipo de cada uno de esos símbolos en la misma cláusula de imagen, pero eso no es un requisito. Los elementos de datos que, de los tres símbolos anteriores, solo utilizan símbolos de imagen "A" se conocen como elementos de datos alfabéticos, mientras que aquellos que utilizan símbolos de imagen "9" sin ningún símbolo "A" o
~~~

"X" (o aquellos que tienen un USO sin una IMAGEN) se conocen como elementos de datos numéricos. Todos los demás elementos de datos se conocen como elementos de datos alfanuméricos.

Si necesita asignar espacio para un elemento de datos cuyo formato es de dos letras seguidas de cincodígitos seguidos de tres letras, podría utilizar la cadena de imágenes AA99999AAA, A(2)9(5)A(3)

XXXXXXXXXX o X(10). No existe ninguna diferencia funcional entre los cuatro: ninguno de ellos proporciona ninguna funcionalidad que los otros no tengan. Los dos primeros probablemente permitan una mejor documentación del contenido de los campos esperados, pero no proporcionan ninguna capacidad de cumplimiento en tiempo de ejecución.

Sin embargo, en lo que respecta a la aplicación, tanto las cadenas de imágenes alfabéticas como las numéricas proporcionan capacidades de aplicación tanto en tiempo de compilación como en tiempo de ejecución. En el caso de la aplicación de la compilación, el compilador puede emitir mensajes de advertencia si intenta especificar un valor no numérico para un elemento de datos numéricos o si intenta MOVER (consulteSección 7.8.30 [MOVE], página 354) un elemento de datos no numérico a uno que sí lo es. Existen capacidades similares para elementos de datos alfabéticos. En tiempo de ejecución, puede utilizar una prueba de clase especial (consulteSección 2.1.27.2 [Condiciones de la Clase], página 45) para determinar si el contenido de un elemento de datos es completamente numérico o completamente alfabético.

8. '1' Define el almacenamiento de un solo bit que representa una condición booleana con estados de cero o 1, condición de encendido o apagado. Advertencia: el nivel de implementación de esta función es específico de la versión del compilador a partir de v3.1-RC-1.
9. Los siguientes símbolos de imagen se pueden utilizar con elementos de datos numéricos.

'PAG' Define una posición de dígito implícita que se considerará cero cuando se haga referencia al elemento de datos en tiempo de ejecución. Este símbolo se utiliza para permitir que los elementos de datos que contendrán valores muy grandes se asignen utilizando menos almacenamiento al suponer que existe una cierta cantidad de ceros finales (uno por "P") al final de los valores.

El símbolo "P" no está permitido junto con el símbolo "N".

El símbolo "P" sólo se puede utilizar al principio o al final de una oración con

imagen. "P" es un símbolo repetible.

Todos los cálculos y MOVE (verSección 7.8.30 [MOVE], página 354) las operaciones que involucran dicho elemento de datos se comportarán como si los ceros realmente estuvieran allí.

Por ejemplo, supongamos que necesita asignar un elemento de datos que contenga la cantidad de millones de dólares de ingresos brutos que su empresa tenga este año:

```
01 Ingresos Brutos PIC 9(9).
```

En este caso, se reservarán 9 caracteres de almacenamiento. Los valores 000000000 a 999999999 representarán los ingresos brutos. Pero, si solo se registran los millones (es decir, los últimos seis dígitos siempre serán 0), puede definir el campo como:

```
01 Ingresos Brutos PIC 9(3)P(6).
```

Siempre que se haga referencia a Ingresos brutos en los cálculos, o siempre que su valor se mueva a otro elemento de datos, el valor de Ingresos brutos se tratará como si fuera nnn000000, donde nnn es el valor real almacenado.

Si quisieras almacenar el valor 128 millones en ese campo, lo harías como si las "P" fueran "9":

```
MOVER 128000000 A Ingresos Brutos
```

UNA PANTALLA (verSección 7.8.12 [PANTALLA], página 302) de un elemento de datos que contiene símbolos "P" es un poco extraño. El valor mostrado será el

que esté realmente almacenado, pero el tamaño total del valor mostrado será como si los símbolos "P" hubieran sido "9". Por lo tanto, después de que la declaración anterior estableciera un valor para Ingresos brutos, una DISPLAY Ingresos brutos produciría un resultado de "128000000". Estos son los tres caracteres reales almacenados con seis ceros agregados.

'S' Este El símbolo, si se utiliza, debe ser el primer símbolo en el valor PICTURE. Una 'S' indica que el elemento de datos tiene signo, lo que significa que son posibles valores negativos para este elemento de datos. Sin una 'S', a cualquier valor negativo almacenado en este elemento de datos mediante una instrucción MOVE o aritmética se le quitará el signo negativo (convirtiéndose en el valor absoluto).

El símbolo "S" no está permitido junto con el símbolo "N".

El símbolo 'S' solo puede aparecer una vez en una cadena de imágenes. VerSección 6.9.47 [SIGNO IS], página 212, para una discusión más detallada sobre cómo se pueden almacenar valores negativos en un elemento de datos numéricos.

"V" Este El símbolo se utiliza para definir dónde se encuentra un punto decimal implícito (si lo hay) en un elemento numérico. Así como puede haber un solo punto decimal en un número, no puede haber más de una "V" en una IMAGEN. Los puntos decimales implícitos no ocupan espacio en el almacenamiento; solo especifican cómo se utilizan los valores. Por ejemplo, si el valor 1234 está almacenado en un campo definido como PIC 999V9, ese valor se trataría como 123.4 en cualquier declaración que lo haga referencia.

El símbolo "V" no está permitido junto con "N". El símbolo "V" solo puede aparecer una vez en una cadena de imágenes.

10. Cualquier símbolo de edición introducido después de este punto, si se codifica en la cláusula de imagen de un elemento de datos numérico, transformará ese elemento de datos de un elemento de datos numérico a un elemento de datos editado numéricamente. Los elementos de datos editados numéricamente se tratan como alfanuméricos y no pueden servir como subíndices de tabla ni como argumentos de origen en una declaración aritmética.
11. Los siguientes son los símbolos de edición de inserción fijos que se pueden especificar en una cadena de imágenes. Cada uno de estos símbolos de edición insertará un carácter especial en el valor del campo en la posición especificada en la cadena de imágenes. Estos símbolos de edición introducirán un carácter adicional en el tamaño total del campo por cada aparición del símbolo en la cadena de imágenes.

'B' El símbolo de edición 'B' introduce un espacio en blanco en el valor del campo para cada ocurrencia.

Se pueden codificar varios símbolos 'B'.

El siguiente ejemplo formateará un número de diez dígitos (presumiblemente un número de teléfono) en un diseño '### ### ####':

```
- - -
    05 Número de teléfono      Foto 9(3)B9(3)B9(4).
- - -
    MOVER 5185551212 A PANTALLA DE
    NÚMERO DE TELÉFONO Número de teléfono
```

Este código mostrará '518 555 1212'.

'0' El símbolo de edición '0' (cero) introduce un carácter "0" en el valor del campo para cada aparición en la cadena de imágenes.

Se pueden codificar varios símbolos "0". A continuación, se muestra un ejemplo:

```
- - -
```

```
05    Elemento de salida        Foto 909090909.
- - -
MOVE 12345 A PANTALLA del
elemento de salida Elemento de salida
```

Lo anterior mostrará '102030405'.

'/' El símbolo de edición '/' inserta un carácter "/" en el valor del campo para cada aparición en la cadena de imágenes.

Se pueden codificar varios símbolos '/'.

Este símbolo de edición se utiliza con mayor frecuencia para formatear fechas, de la siguiente manera:

```
---
    05   Año-Mes-Día          Foto 9(4)/9(2)/9(2).
---
    MOVER 20140207 A LA VISUALIZACIÓN Año-
    Mes-Día Año-Mes-Día
```

Este ejemplo muestra '2014/02/07'.

12. Los siguientes son los símbolos de formato numérico que se pueden especificar en una cadena de imágenes. Cada uno de estos símbolos de edición insertará caracteres especiales en el valor del campo para presentar los números en un formato "amigable". Estos símbolos de edición introducirán un carácter adicional en el tamaño total del campo por cada aparición del símbolo en la cadena de imágenes. Los campos numéricos cuya cláusula de imagen contenga estos caracteres no se pueden utilizar como campos de origen en ningún cálculo ni pueden servir como campos de origen para la transferencia de valores de datos a ningún elemento de datos que no sea un campo alfanumérico.

'.' El símbolo '.' inserta un punto decimal en el valor de un campo numérico. Cuando el contenido de un campo de envío de un elemento de datos numéricos se mueve a un elemento de datos de recepción cuya cláusula de imagen contiene el símbolo de edición '.', el punto decimal implícito ('V') o real en el elemento de datos de envío o literal, respectivamente, se alineará con el símbolo '.' en el campo de recepción. Luego, los dígitos se transfieren desde el campo de envío al campo de recepción hacia afuera desde la 'V' o '.' del campo de envío, truncando los dígitos de envío si no hay suficientes posiciones en el campo de recepción. Cualquier posición de dígito en el campo de recepción que no reciba dígitos del campo de envío, si las hay, se establecerá en 0.

El símbolo "." no está permitido junto con "N". Un

ejemplo puede resultar de ayuda:

```
---
05   Campo fuente    PIC 9(2)V9 VALOR 7.2.
05   Campo de destino    CIF 9(5).9(2).
---
MOVER 1234567.89 AL CAMPO DE DESTINO
PANTALLA Campo de
destinoMOVER 19 A LA
PANTALLA DE CAMPO DE
DESTINO Campo de
destino
MOVER Campo de origen A Campo de
destino MOSTRAR Campo de destino
```

El ejemplo mostrará tres resultados: '34567.89', '00019.00' y '00007.20'.

Ambas definiciones de elementos de datos parecen tener dos puntos decimales en sus cláusulas de imagen. En realidad no es así, porque el último carácter de cada definición de elemento de datos es siempre un punto, el punto que finaliza la definición.

',' El','El símbolo sirve como separador de miles. Muchas veces, verás números

grandes formateados con estos símbolos, por ejemplo, 123.456.789. Esto se puede lograr fácilmente agregando símbolos separadores de miles a una cadena de imágenes. Los símbolos separadores de miles que no sean necesarios se ignorarán, es decir, no se usarán.

El símbolo "," no está permitido junto con "N". A continuación, se muestra un ejemplo:

```
- - -
05   Mis-ganancias-de-la-lotería     Foto 9(3),9(3),9(3).
- - -
MOVER 12345 A Mis-Ganancias-De-Lotería
MOSTRAR Mis-Ganancias-De-Lotería
```

Esto produce 012.345

El valor '12.345' (muy decepcionante para mis planes de jubilación, pero unSe mostrará una buena demostración del separador de miles). Observe cómo, dado que la primera coma no era necesaria debido a la escasa cantidad que gané, se ignora.

Si lo desea, puede invertir los roles de los símbolos de edición '.' y "," especificando EL PUNTO DECIMAL ES COMA en NOMBRES ESPECIALES (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90)
párrafo.

```
- - -
05   Mis-ganancias-de-la-lotería     FOTO zzz,zzz,zz9.
- - -
MOVER 12345 A Mis-Ganancias-De-Lotería
MOSTRAR Mis-Ganancias-De-Lotería
```

Esto produce 12.345

Tenga en cuenta que no hay ceros a la izquierda cuando se utiliza la imagen Z.

13. Los siguientes son símbolos de inserción. Se utilizan para insertar un carácter adicional (dos en el caso de CR y DB) para indicar el signo (positivo o negativo) del valor numérico que se mueve al campo cuya cadena de imágenes contiene uno de estos símbolos, o el hecho de que el elemento de datos representa una cantidad de moneda (dinero). Solo se puede utilizar uno de los símbolos '+', '-', CR o DB en una cláusula de imagen. En este contexto, cuando se utiliza cualquiera de estos símbolos en una cadena de imágenes, deben estar al final. Los símbolos '+', '-' y/o de moneda también se pueden utilizar como símbolos de edición flotantes al principio de la cadena de imágenes, un tema que se tratará en el siguiente párrafo numerado.

'+' Si el valor del valor numérico introducido en el campo es positivo (0 o mayor), se insertará el carácter "+". Si el valor es negativo (menor que 0), se insertará el carácter "-".

El símbolo '+' no está permitido junto con 'N'.

'-' Si el valor del valor numérico introducido en el campo es positivo (0 o mayor), se insertará un espacio. Si el valor es negativo (menor que 0), se insertará un carácter '-'.

El símbolo '-' no está permitido junto con 'N'.

ES Este símbolo está codificado con los dos caracteres "C" y "R". Si el valor del valor numérico introducido en el campo es positivo (0 o mayor), se insertarán dos espacios. Si el valor es negativo (menor que 0), se insertarán los caracteres CR (crédito).

El símbolo CR no está permitido junto con 'N'.

base de datos Este símbolo se codifica con los dos caracteres "D" y "B". Si el valor del valor numérico introducido en el campo es positivo (0 o mayor), se insertarán dos espacios. Si el valor es negativo (menor que 0), se insertarán los caracteres DB (débito).

El símbolo DB no está permitido junto con 'N'.

'$' Independientemente del valor que se mueva al campo, este símbolo insertará el símbolo de moneda en el valor del elemento de datos en la posición donde aparece en la cadena de imágenes (verSección 5.1.3 [NOMBRES ESPECIALES], página 90).

El símbolo '$' no está permitido junto con 'N'.

14. Estos símbolos de edición se conocen como símbolos de reemplazo flotantes. Estos símbolos pueden aparecer en secuencias antes de cualquier símbolo de edición "9" en la cadena de imágenes de un elemento de datos numéricos. El uso de estos símbolos transforma ese elemento de datos numéricos en un elemento de datos editado numérico, que ya no se puede utilizar en cálculos o subíndices.
15. Cada uno de los siguientes símbolos se comporta como un '9', hasta que se agotan todos los dígitos del valor numérico y se van a insertar los ceros iniciales. En efecto, estos símbolos de edición definen lo que debe suceder con esos ceros iniciales.

'$' De aquellos símbolos de moneda que corresponden a posiciones de caracteres en las que residen ceros iniciales, el valor '0' del más a la derecha será reemplazado por el símbolo de moneda vigente para el programa (verSección 5.1.3 [NOMBRES ESPECIALES], página 90). Cualquier valor cero inicial restante que ocupe posiciones descritas por este símbolo será reemplazado por espacios.

El símbolo '$' no está permitido junto con 'N'.

Cualquier símbolo de moneda codificado a la derecha de un '.' será tratado exactamente como un '9'.

'*' Este símbolo se denomina símbolo de protección de verificación. Todos los símbolos de protección de verificación que corresponden a posiciones de caracteres en las que se encuentran ceros a la izquierda tendrán sus valores "0" reemplazados por "*".

El símbolo '*' no está permitido junto con 'N'.

Cualquier símbolo de supresión de verificación codificado a la derecha de un '.' será tratado exactamente como un '9'.

'+' De los símbolos "+" que corresponden a posiciones de caracteres en las que se encuentran ceros a la izquierda, el valor "0" del que se encuentra más a la derecha se reemplazará por un "+" si el valor en el elemento de datos es cero o mayor, o por un "-" en caso contrario. Los valores cero a la izquierda restantes que ocupen posiciones descritas por este símbolo se reemplazarán por espacios. No se pueden usar tanto "+" como "-" en la misma cadena de imágenes.

El símbolo '+' no está permitido junto con 'N'.

Cualquier símbolo '+' codificado a la derecha de un '.' será tratado exactamente como un '9'.

'-' De los símbolos "-" que corresponden a posiciones de caracteres en las que se encuentran ceros a la izquierda, el valor "0" del que se encuentra más a la derecha se reemplazará por un espacio si el valor en el elemento de datos es cero o mayor, o por un "-" en caso contrario. Los valores cero a la izquierda

restantes que ocupen posiciones descritas por este símbolo se reemplazarán por espacios. No se pueden usar tanto "+" como "-" en la misma cadena de imágenes.

El símbolo '-' no está permitido junto con 'N'.

Cualquier símbolo '-' codificado a la derecha de un '.' será tratado exactamente como un '9'.

'CON' Todos los símbolos 'Z' que corresponden a posiciones de caracteres en las que residen ceros iniciales tendrán sus valores '0' reemplazados por espacios.

Cualquier símbolo de supresión de cero codificado a la derecha de un '.' será tratado exactamente como un '9'.

'Z' y '*' no deben codificarse en la misma cadena de imágenes

'+' y '-' no deben estar codificados en la misma cadena de imágenes

Cuando se codifican varios símbolos flotantes, incluso si solo se utiliza uno de ellos, solo el último codificado se considerará flotante. Los símbolos flotantes principales se tratarán como fijos y aparecerán en la salida en la posición codificada. Por ejemplo, si un elemento de datos tiene una cadena de imagen PIC +$ZZZZ9.99 y se mueve un valor de 1 a ese campo en tiempo de ejecución, el valor resultante será (el símbolo b representa un espacio) +$bbbb1.00. Esto es coherente con muchas otras implementaciones de COBOL, donde el resultado también habría sido +$bbbb1.00.

6.9.38 PRESENTE CUANDO

Sintaxis de la cláusula PRESENTE-CUANDO

PRESENTE CUANDO nombre-de-la-condicion
~~~~~~~ ~~~~

Esta sintaxis es válida en las siguientes secciones: INFORME

Esta cláusula nombra un nombre de condición existente (verSección 2.1.27.1 [Nombres de las condiciones], página 44) que servirá como interruptor que controla la presentación o supresión de un grupo de informes.

1. Si el nombre de condición especificado tiene un valor de FALSO cuando se ejecuta una declaración GENERATE (verSección 7.8.20 [GENERAR], página 333) hace que se presente un grupo de informes, se suprimirá la presentación de ese grupo.
2. Si el nombre de la condición tiene un valor VERDADERO, se presentará el grupo.
3. VerSección 2.1.27.1 [Nombres de las condiciones], página 44, para obtener más información.
~~~~~~~

6.9.39 INMEDIATO

Sintaxis de la cláusula PROMPT

PROMPT [CARÁCTER ES literal-1 | identificador-1]
~~~~~~ ~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula define el carácter que se utilizará como carácter de relleno para cualquier campo de entrada en la pantalla.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. El carácter de solicitud predeterminado, en caso de que no se codifique ninguna especificación de CARÁCTER, o si
   La cláusula PROMPT puede estar totalmente ausente, es un guión bajo ('_').
3. Los caracteres indicados se transformarán automáticamente en espacios al ingresarlos.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~

6.9.40 PROTEGIDO

Sintaxis del atributo PROTECTED

```
EL TAMAÑO PROTEGIDO ES { identificador }
~~~~~~~~ ~~~~      { entero       }
```

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula extendida PROTECTED limitará el tamaño del campo especificado, independientemente del tamaño de la imagen.1

1. La frase SIZE especifica el tamaño (longitud) del campo. Una vez finalizada la ejecución de ACCEPT o DISPLAY, el cursor se coloca inmediatamente después del campo definido por esta cláusula, a menos que esto coloque el cursor fuera de la ventana de terminal actual. En este caso, el cursor se desplaza hasta el comienzo de la siguiente línea (desplazándose por la ventana si es necesario).
2. Si no se utiliza la frase SIZE, la longitud del campo se establece de manera predeterminada en el tamaño del elemento que se acepta o se muestra. Sin embargo, si se utiliza la frase CONVERT, el tamaño del campo depende del tipo de datos del elemento y del verbo que se utiliza.
   A. Si se está ejecutando el verbo DISPLAY, el tamaño es el mismo que si no se hubiera especificado la frase CONVERT, excepto para los elementos numéricos. En el caso de los elementos numéricos, el tamaño es la cantidad de dígitos del elemento, más uno si no es un entero, más uno si tiene signo. Los casos restantes cubren el tamaño cuando se utiliza una sentencia ACCEPT.
   B. Si el elemento es numérico o se puede editar numéricamente, entonces el tamaño es la cantidad de dígitos del elemento, más uno si no es un entero, más uno si tiene signo.
   C. Si el elemento se edita alfanuméricamente, el tamaño se establece en el número de posiciones 'A' o 'X' especificadas en su cláusula PICTURE.
   D. Para todos los demás tipos de datos, el tamaño del campo se establece en el tamaño del elemento (igual que si CONVERT
   (no se especificaron).
3. Tenga en cuenta que la frase OUTPUT cambia la forma en que se calcula el tamaño de campo predeterminado. Consulte el encabezado anterior para obtener más detalles. Tenga en cuenta también que la frase OUTPUT afecta solo la forma en que se muestran los elementos en la pantalla; el formato interno de los datos aceptados no se ve afectado.
4. Tenga en cuenta que no puede proporcionar la frase CONVERTIR en la sección de pantalla. Por lo tanto, el tamaño de un campo de sección de pantalla siempre es el tamaño de su entrada de pantalla a menos que se especifique la frase TAMAÑO.

[1] ¿O SI? autor incierto

6.9.41 REDEFINE

Sintaxis de la cláusula REDEFINES

REDEFINES identificador-1
~~~~~~~~~

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

La cláusula REDEFINES hace que el elemento de datos en cuya definición se especifica la cláusula REDEFINES (en adelante denominado el objeto de redefinición) ocupe el mismo espacio de almacenamiento físico que el identificador-1 (en adelante denominado el sujeto de la redefinición).

1. Para poder utilizar REDEFINES se deben seguir las siguientes reglas:
   A. El número de nivel de los elementos de datos del sujeto y del objeto debe ser el mismo.
   B. Los números de nivel de los elementos de datos del sujeto y del objeto no pueden ser 66, 78 u 88.
   C. Si N representa el número de nivel del objeto, entonces no se pueden definir otros elementos de datos con número de nivel N entre los elementos de datos del sujeto y del objeto a menos que también sean REDEFINIDOS del sujeto.
   D. Si N representa el número de nivel del objeto, entonces no se podrán definir otros elementos de datos con un número de nivel numéricamente menor que N entre los elementos de datos del sujeto y del objeto.
   E. El tamaño total asignado del elemento de datos del sujeto debe ser el mismo que el tamaño total asignado del elemento de datos del objeto.
   F. No OCURRE (verSección 6.9.35 [OCURRES], página 191) puede ser parte de la definición de los elementos de datos del sujeto o del objeto. Sin embargo, cualquiera de ellos o ambos pueden ser elementos de grupo que contienen elementos de datos con cláusulas OCCURS.
   G. Sin VALOR (verSección 6.9.61 [VALOR], página 239) se puede definir en el elemento de datos del objeto, y ningún elemento de datos subordinado al elemento de datos del objeto puede tener cláusulas VALUE, con la excepción de los nombres de condición de nivel 88.
~~~~~~~~~

6.9.42 CAMBIOS DE NOMBRES

Sintaxis de la cláusula RENAMES

RENOMBRAMIENTO identificador-1 [HASTA|HASTA identificador-2]
~~~~~~~ ~~~~ ~~~~~~~

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

La cláusula RENAMES reagrupa elementos previamente definidos especificando agrupaciones alternativas, posiblemente superpuestas, de elementos de datos elementales.

1. Las palabras reservadas THRU y THROUGH son intercambiables.
2. Debe utilizar el número de nivel 66 para las entradas de descripción de datos que contienen los CAMBIOS DE NOMBRES
   cláusula.
3. Los elementos de datos del identificador 1 y del identificador 2, junto con todos los elementos de datos definidos entre esos dos elementos de datos en la fuente del programa, deben estar todos contenidos dentro de la misma descripción de registro de nivel 01.
4. VerSección 6.8.2 [Elementos de datos de nivel 66], página 149, para obtener información adicional sobre la cláusula RENAMES.
~~~~~~~

6.9.43 REQUERIDO

Sintaxis del atributo REQUERIDO

REQUERIDO
~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula obliga al usuario a ingresar datos en el campo en el que están especificados (o en todos los campos subordinados con capacidad de entrada si se especifica REQUIRED en un elemento del grupo).

1. El CHEQUE VACÍO (verSección 6.9.17 [CHEQUE VACÍO], página 171) y las cláusulas REQUIRED son intercambiables y no pueden usarse juntas en la misma descripción del elemento de datos.
2. Para que tenga efecto, el usuario primero debe mover el cursor al campo que tiene esta cláusula en su definición.
3. La declaración del elemento de datos ACCEPT (verSección 7.8.1.4 [ACEPTAR elemento de datos], página 267) ignorará la tecla Enter y cualquier otra pulsación de tecla que haga que el cursor se mueva a otro elemento de la pantalla a menos que se hayan ingresado datos en el campo. Las teclas de función aún podrán finalizar la operación ACEPTAR.
4. Para que funcione, este atributo debe ser compatible con el paquete "curses" subyacente con el que se creó la implementación de GnuCOBOL. A partir de este momento, el paquete "PDCurses" (usado para compilaciones nativas de Windows o MinGW) no es compatible con REQUIRED.
~~~~~~~~

6.9.44 VIDEO INVERSO

Sintaxis del atributo REVERSE-VIDEO

VIDEO INVERSO
~~~~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

El atributo REVERSE-VIDEO intercambia el COLOR DE PRIMER PLANO especificado o implícito (verSección 6.9.21 [COLOR DEL PRIMER PLANO], página 175) y COLOR DE FONDO (verSección 6.9.6
[COLOR DE FONDO], página 159) atributos para el campo cuya definición contiene esta cláusula (o todos los campos subordinados si se utilizan en un elemento de grupo).

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~~~~~

6.9.45 MISMO QUE

Sintaxis de la cláusula SAME-AS

IGUAL QUE nombre-datos-1
~~~~ ~~

———————————————————————————————————————

La sintaxis es válida en las siguientes secciones: WORKING-STORAGE

SAME AS hace que un elemento de datos herede la misma definición de otro elemento de datos.

1. Data-name-1 es un elemento de datos definido en otra parte del mismo programa.
2. El dato nombre-1 no debe tener subíndice.
3. Una entrada de descripción de datos que especifica la cláusula SAME AS no debe ser seguida inmediatamente por una entrada de descripción de datos subordinada o una entrada de nivel 88.
4. Ni la descripción de data-name-1 ni la descripción de ningún elemento de datos subordinado al tema de la entrada contendrán, directa o indirectamente, una cláusula SAME AS que haga referencia al tema de la entrada o a cualquier elemento del grupo al que esta entrada esté subordinada.
5. La descripción de data-name-1, incluidos sus elementos de datos subordinados, no debe contener una cláusula TYPE que haga referencia al registro al que esta entrada está subordinada.
6. La descripción de data-name-1 no debe contener una cláusula OCCURS. Sin embargo, los elementos subordinados a data-name-1 pueden contener cláusulas OCCURS.
7. Data-name-1 deberá hacer referencia a un elemento elemental o a un elemento de grupo de nivel 1 descrito en la sección de archivo, almacenamiento de trabajo, almacenamiento local o enlace.
8. Si el tema de la entrada es un elemento de nivel 77, data-name-1 deberá hacer referencia a un elemento elemental.
9. Un elemento de grupo al cual el sujeto de la entrada está subordinado no debe contener una cláusula GROUP-USAGE, SIGN o USAGE.
10. El efecto de la cláusula SAME AS es como si la descripción de datos identificada por data-name-1 hubiera sido codificada en lugar de la cláusula SAME AS, excluyendo el número de nivel, el nombre y las cláusulas EXTERNAL, GLOBAL y REDEFINES especificadas para data-name-1; los números de nivel de los elementos subordinados se pueden ajustar como se describe en la regla general 2.
11. Si data-name-1 describe un elemento de grupo:
12. el tema de la entrada es un grupo cuyos elementos subordinados tienen los mismos nombres, descripciones y jerarquía que los elementos subordinados de data-name-1, los números de nivel de los elementos subordinados a ese grupo se ajustan, si es necesario, para preservar la jerarquía de data-name-1, los números de nivel en la jerarquía resultante pueden exceder 49.

```
IDENTIFICACIÓN    DIVISIÓN.
PROGRAMA-ID.      prog.
DATOS             DIVISIÓN.

TRABAJO-ALMACENAMIENTO
```
~~~~

```
SECCIÓN.01
MENSAJE-TEXTO-2 EXTERNO.
    02 NOMBRE DEL ARCHIVO PRINCIPAL     FOTO X(50).
    02 RELLENO REDEFINE EL NOMBRE DEL ARCHIVO PRINCIPAL.
        05 RELLENO PIC 9999.
    02 NOMBRE-ARCHIVO-PRINCIPAL-2.
```

```
            05 RELLENO PIC 9999.
            05 DETALLE-NO FOTO 9999.
      02 RELLENO              MISMO  COMO NOMBRE DEL ARCHIVO PRINCIPAL.
77 NOMBRE DE SALIDA    MISMO  COMO DETALLE-NO GLOBAL.
01 Z-MENSAJE-T2        IGUAL QUE NOMBRE-DE-ARCHIVO-PRINCIPAL-2.
01 Z-MENSAJE-T3.
         49 metros cúbicos                MISMO   COMO MENSAJE-TEXTO-2.
         49 MT3-REN     REDEFINE  MT3 IGUAL QUE MENSAJE-TEXTO-2.

DIVISIÓN DE PROCEDIMIENTOS.
      MOSTRAR NOMBRE DE ARCHIVO PRINCIPAL DEL
      TEXTO DEL MENSAJE 2 MOSTRARDETALLE-NOOF
                                    MENSAJE Z-T2
      MOSTRAR NOMBRE DE ARCHIVO PRINCIPAL DE
      MT3
      MOSTRAR NOMBRE DE
      SALIDA GOBACK.
```

6.9.46 SEGURO

Sintaxis del atributo SECURE

SEGURO
~~~~~~

——

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula hará que todos los datos ingresados en el campo aparezcan en la pantalla como asteriscos.

1. Las cláusulas NO-ECHO y SECURE son intercambiables si la configuración de dialecto -fno-echo-means-secure está activa.
2. Las cláusulas NO-ECHO y SECURE no se pueden utilizar juntas en la misma descripción de elemento de datos.
3. Esta cláusula solo se puede utilizar en un campo que permita la entrada de datos (un campo que contenga USING (verSección 6.9.60 [USO], página 238) o A (verSección 6.9.54 [TO], página 220) cláusula).

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~

6.9.47 LA SEÑAL ES

Sintaxis de la cláusula SIGN-IS

EL SIGNO ESTÁ ANTERIOR|SEGURO [CARÁCTER SEPARADO]
~~~~ ~~~~~~~ ~~~~~~~~ ~~~~~~~~

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

Esta cláusula, permitida únicamente para elementos de datos numéricos de USAGE DISPLAY, especifica cómo se interpretará un símbolo "S" en la cláusula de imagen de un elemento de datos.

1. Las palabras reservadas CHARACTER y IS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. *Sin*la opción CARÁCTER SEPARADO, el signo del valor del elemento de datos se codificará transformando el último (ver FINAL) o el primer (ver INICIAL) dígito de la siguiente manera:

Primer/Último dígito	Valor para lo positivo	Valor para Negativo
0	0	pag
1	1	q
2	2	a
3	3	s
4	4	a
5	5	en
6	6	en
7	7	En
8	8	incógnita
9	9	y

3. Si se utiliza la cláusula SEPARATE CHARACTER, se insertará un carácter '+' o '-' real en el valor del campo como el primer carácter (INICIAL) o el último carácter (FINAL). Tenga en cuenta que tener este carácter integrado en el almacenamiento del elemento de datos no impide que el elemento de datos se utilice como campo de origen en operaciones aritméticas.
4. Cuando se especifica CARÁCTER SEPARADO, el símbolo 'S' en la IMAGEN del elemento de datos se debe contar al determinar el tamaño del elemento de datos.
5. Ni la presencia de un dígito codificado (ver arriba) ni un carácter '+' o '-' real incrustado dentro del almacenamiento del elemento de datos impiden que el elemento de datos se utilice como campo fuente en operaciones aritméticas.
~~~~

6.9.48 TAMAÑO

Sintaxis de la cláusula SIZE

TAMAÑO ES variable-1 | literal-1
~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

El tamaño de la Variable a aceptar de la pantalla. VARIABLE-1 o LITERAL-1 debe ser numérico. 'SIZE <mayor que cero>' Si VARIABLE-1 o LITERAL-1 es menor que la longitud de la variable, entonces solo el número 'SIZE' de caracteres acepta en el campo. La variable se rellena con espacios después de 'SIZE' hasta el final del campo. Si VARIABLE-1 o LITERAL-1 es mayor que la variable, entonces la pantalla se rellena con espacios después de la variable hasta la longitud de 'SIZE'. 'SIZE ZERO' '<Opción SIZE no especificada>' La variable acepta hasta la longitud de su campo.
~~~~

6.9.49 FUENTE

Sintaxis de la cláusula SOURCE

FUENTE ES literal-1 | identificador-1 [REDONDEADO]

Esta sintaxis es válida en las siguientes secciones: INFORME

Esta cláusula adjunta lógicamente un elemento de datos de la sección de informe a otro elemento de datos definido en otra parte de la división de datos.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Cuando se presenta el grupo de informes que contiene esta cláusula, el valor del literal numérico o identificador especificado se moverá automáticamente al elemento de datos del informe antes de la presentación.
3. El identificador especificado puede definirse en cualquier parte de la división de datos, pero si se define enLa sección del informe solo puede ser CONTADOR DE PÁGINAS, CONTADOR DE LÍNEAS o una SUMA (verSección 8.1.92 [SUMA], página 524) encimera.
4. La IMAGEN (verSección 6.9.37 [IMAGEN], página 195) del elemento de datos del informe debe ser tal que sea legalMOVERSE (verSección 7.8.30 [MOVE], página 354) el literal o identificador especificado a un elemento de datos con esa IMAGEN.
5. La opción ROUNDED entra en juego si la cantidad de dígitos a la derecha de un punto decimal real o supuesto es diferente entre el valor literal o identificador especificado (el "valor de origen") y el PICTURE especificado para el campo en cuya definición aparece la cláusula SOURCE (el "campo de destino"). Sin ROUNDED, los dígitos sobrantes en el valor de origen simplemente se truncarán para ajustarse al campo de destino. Con ROUNDED, el valor de origen se redondeará aritméticamente para ajustarse al campo de destino. ConsulteSección 7.6.7 [REDONDEADA], página 257, para obtener información sobre la regla de redondeo MÁS CERCANO A CERO, que es la que se aplicará.

6.9.50 NOMBRES ESPECIALES.

Sintaxis de la cláusula SPECIAL-NAMES

```
ES NOMBRES ESPECIALES { CURSOR }
      ~~~~~~~~~~~~~~ { ESTADO CRT       }
                     { CONTROL DE PANTALLA }
                { ESTADO DEL EVENTO      }
```

Esta sintaxis es válida en las siguientes secciones: DIVISIÓN DE DATOS

La cláusula SPECIAL-NAMES permite identificar elementos de nombres especiales de uso común directamente en la división de datos. Ejemplo de uso:

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa. A continuación se muestra un ejemplo de uso.

```
01   wCURPOS  ESCURSOR DE NOMBRES ESPECIALES.
     03 wCURSOR-FILA PIC 999.
     03 wCURSOR-COL PIC 999.
01   wCRTSTAT ES EL ESTADO DEL CRT CON NOMBRES ESPECIALES PIC 9(5).
```

6.9.51 SUMA DE

Sintaxis de la cláusula SUM-OF

```
SUMA DE { identificador-7 }...    [ { REINICIAR EN FINAL|identificador-8 } ]
~~~     { literal-2       }       { ~~~~~   ~~~~~                 }
                                  { Identificador UPON-9          }
                                    ~~~~
```

Esta sintaxis es válida en las siguientes secciones: INFORME

La cláusula SUM establece un contador de suma cuyo valor se calculará aritméticamente cada vez que se presente el campo.

1. Las palabras reservadas OF y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La cláusula SUM sólo puede aparecer en un grupo de informes CONTROL FOOTING.
3. Si al elemento de datos en el que aparece la cláusula SUM se le ha asignado su propio nombre de identificador, y ese nombre no es FILLER, entonces ese elemento de datos se denomina contador de suma.
4. Todos los elementos de datos del identificador 7 deben ser de naturaleza numérica no editada.
5. Si se define algún elemento de datos de identificador-7 en la sección de informe, debe ser un contador de suma.
6. Cualquier elemento de datos de identificador-7 que sea un contador de suma debe definirse en el mismo grupo de informes que el elemento de datos en el que aparece esta cláusula SUM o debe definirse en un elemento de datos de informe que exista en un nivel inferior en la jerarquía de control de este informe. ConsulteSección 9.5 [Jerarquía de control], página 654, para obtener información adicional.
7. La IMAGEN del elemento de datos del informe en cuya descripción aparece esta cláusula SUM debe ser tal que sea legal MOVERLA (verSección 7.8.30 [MOVE], página 354) el valor del identificador-7 o literal-2 especificado a un elemento de datos con esa IMAGEN.
8. Los siguientes puntos se aplican a la opción UPON:
   A. El identificador del elemento de datos 9 debe ser el nombre de un grupo de detalles especificado en el mismo informe que el grupo de pie de control en el que aparece esta cláusula SUM.
   B. La presencia de una cláusula UPON limita la cláusula SUM a agregar el valor literal numérico o identificador especificado al contador de suma solo cuando se ejecuta una declaración GENERATE identifier-9.
   C. Si no se especifica ninguna cláusula UPON, el valor del identificador-7 o literal-2 se agregará al contador de suma siempre que se ejecute un GENERATE (verSección 7.8.20 [GENERAR], página 333) de cualquier grupo de informes detallados en el informe se ejecuta.
   D. Si solo hay un único grupo de detalles en la definición del informe, la cláusula UPON no tiene sentido.
9. Los siguientes puntos se aplican a la opción RESET:
   A. Si la opción RESET está codificada, FINAL o el identificador 8 (el que esté codificado en RESET) debe ser uno de los cortes de control del informe especificados en la cláusula CONTROLS.
   B. Si no hay una opción RESET codificada, el contador de suma se restablecerá a cero cada vez que se presente el pie de control que contiene la cláusula SUM. Este es el

comportamiento típico que se esperaría.

C. Sin embargo, si desea restablecer el contador SUM solo cuando se presente el pie de control para un corte de control superior en la jerarquía de control, especifique ese corte de control superior en la opción RESET.

6.9.52 SINCRONIZADO

Sintaxis SINCRONIZADA

```
SINCRONIZADO|SINCRONIZADO [ IZQUIERDA|DERECHA ]
~~~~        ~~~~           ~~~~~ ~~~~~~
```

Las cláusulas IZQUIERDA y DERECHA (SINCRONIZADAS) se reconocen sintácticamente pero por lo demás no son funcionales.

———

Esta sintaxis es válida en las siguientes secciones: ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL, ENLACE

Esta cláusula opcional optimiza el almacenamiento de elementos numéricos binarios para almacenarlos de tal manera que la CPU pueda recuperarlos lo más rápido posible.

1. Las palabras reservadas SYNCHRONIZED y SYNCHRONISED son intercambiables y pueden abreviarse como SYNC.

2. Si la cláusula SYNCHRONIZED está codificada en cualquier cosa que no sea un elemento de datos numérico con un USAGE (verSección 6.9.59 [USO], página 228) que especifica el almacenamiento de datos en forma binaria, se ignorará la cláusula SYNCHRONIZED.

3. La sincronización se realiza (por el compilador) de la siguiente manera:

   A. Si el elemento binario ocupa un byte de almacenamiento, no se realiza ninguna sincronización.

   B. Si el elemento binario ocupa dos bytes de almacenamiento, el elemento binario se asigna en el siguiente límite de media palabra.

   C. Si el elemento binario ocupa cuatro bytes de almacenamiento, el elemento binario se asigna en el siguiente límite de palabra.

   D. Si el elemento binario ocupa ocho bytes de almacenamiento, el elemento binario se asigna en el siguiente límite de palabra.

A continuación se muestra la asignación de un grupo de elementos de datos con y sin la opción SYNCHRONIZED. Los bloques grises representan los bytes no utilizados que se asignan en la estructura Group-Item-2 debido a las cláusulas SYNC.

```
01  Group-Item-1.                          01  Group-Item-2.
    05 A            PIC X(1).                  05 A            PIC X(1).
    05 B            USAGE BINARY-SHORT.        05 B    SYNC    USAGE BINARY-SHORT.
    05 C            PIC X(2).                  05 C            PIC X(2).
    05 D            USAGE BINARY-LONG.         05 D    SYNC    USAGE BINARY-LONG.
    05 E            PIC X(3).                  05 E            PIC X(3).
    05 F            USAGE BINARY-DOUBLE.       05 F    SYNC    USAGE BINARY-DOUBLE.
```

6.9.53 SE ACABÓ EL TIEMPO

Sintaxis de la cláusula TIME-OUT

SE ACABÓ EL TIEMPO
~~~~ ~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula forzará un tiempo específico para que se ingresen los datos.
~~~~

6.9.54 A

Sintaxis de la cláusula TO

Identificador TO-5
~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula adjunta lógicamente un elemento de datos de una sección de pantalla a otro elemento de datos definido en otra parte de la división de datos.

1. La cláusula TO se utiliza para definir un campo de entrada de datos sin valor inicial; cuando se ingresa un valor, se guardará en el identificador especificado.
2. El FROM (verSección 6.9.22 [DESDE], página 176), PARA, UTILIZANDO (verSección 6.9.60 [USO], página 238) y VALOR (verSección 6.9.61 [VALOR], página 239) Las cláusulas son mutuamente excluyentes en la definición de cualquier elemento de datos de la sección de pantalla.

6.9.55 TIPO

Sintaxis de la cláusula TYPE

Formato 1: Todas las secciones excepto INFORME

```
[ TIPO A nombre-tipo-1 ]
  ~~~~
```

Formato 2: INFORME SECCIÓN Solamente

```
[ EL TIPO ES {CH|{ENCABEZADO DE CONTROL} [ON|FOR]                    {
FINAL|identificador-1 } [O PÁGINA]      } ]
  ~~~~        ~~   ~~~~~~~ ~~~~~~~                    ~~~~~
[             {RH|{ENCABEZADO DEL INFORME}                           } ]
               ~~   ~~~~~~ ~~~~~~~
[             {PH|{ENCABEZADO DE PÁGINA}                             } ]
               ~~   ~~~~ ~~~~~~~
[             {DE|DETALLE                                            } ]
               ~~ ~~~~~~
[             {CF|{PIE DE CONTROL} [ON|FOR] [{FINAL|identificador-2} [OR PAGE] ]} ]
               ~~   ~~~~~~~ ~~~~~~~                    ~~~~~
[             {                                 [   TODO                ]} ]
                                                      ~~~~~
[             {RF|{PIE DEL INFORME}                                  } ]
               ~~   ~~~~~~ ~~~~~~~
[             {PF|{PIE DE PÁGINA}                                    } ]
               ~~   ~~~~ ~~~~~~~
```

———

La sintaxis es válida en todas las demás secciones excepto: INFORME

Como formato 1

1. La cláusula TYPE indica que la descripción de datos del tema de la entrada está especificada por un tipo de datos definido por el usuario.
2. El tipo de datos definido por el usuario se define utilizando la cláusula TYPEDEF, que se describe en la cláusula TYPEDEF (consulteSección 6.9.56 [TYPEDEF], página 225).
3. Se aplican las siguientes reglas generales: si el tipo-nombre-1 (definido mediante la cláusula TYPEDEF) describe un elemento de grupo, entonces el sujeto de la cláusula TYPE es un elemento de grupo cuyos elementos subordinados tienen los mismos nombres, descripciones y jerarquías que los elementos subordinados del tipo-nombre-1.
4. Dado que el sujeto de la cláusula TYPE puede tener un número de nivel tan alto como 49 y el nombre de tipo 1 puede ser un elemento de grupo con 49 niveles, la cantidad de niveles de esta jerarquía puede superar los 49. De hecho, dado que las descripciones de los nombres de tipo pueden hacer referencia a otros nombres de tipo, no hay límite para la cantidad de niveles en esta jerarquía.
5. Si se especifica una cláusula VALUE en la descripción de datos del sujeto de la cláusula TYPE, cualquier cláusula VALUE especificada en la descripción de type-name-1 se ignora para esta entrada.
6. Las reglas de alcance para los nombres de tipos son similares a las reglas de alcance para los nombres de

datos.

7. No se permite la modificación de referencia para un elemento elemental que sea objeto de una cláusula TYPE.

8. La descripción de type-name-1, incluidos sus elementos de datos subordinados, no puede contener una cláusula TYPE que haga referencia al registro al que está subordinado el sujeto de la cláusula TYPE (que hace referencia a type-name-1).
9. Por ejemplo, A es un elemento de grupo definido mediante la cláusula TYPEDEF. B también es un elemento de grupo definido mediante la cláusula TYPEDEF, pero que también incluye un elemento subordinado de TYPE.
   A. Siendo este el caso, la definición de tipo para A no puede incluir elementos del TIPO B.
10. El sujeto de una cláusula TYPE no puede cambiar de nombre total o parcialmente y no puede redefinirse explícita o implícitamente.
11. Si el sujeto de una cláusula TYPE está subordinado a un elemento de grupo, la descripción de datos del elemento de grupo no puede contener la cláusula USAGE.
12. La cláusula TYPE no puede aparecer en una entrada de descripción de datos con BLANK WHEN ZERO,FORMATO, JUSTIFICADO, IMAGEN, REDEFINE, CAMBIA EL NOMBRE, SEÑAL, SINCRONIZADO,
    o cláusula de USO.
13. La cláusula TYPE se puede especificar en una entrada de descripción de datos con las cláusulas EXTERNAL, GLOBAL, OCCURS, TYPEDEF y VALUE.
14. Las características esenciales de un tipo, que se identifica por su nombre de tipo, son las posiciones relativas y las longitudes de los elementos elementales definidos en la declaración de tipo, y los BLANCOS CUANDO ESTÉN CERO, JUSTIFICADOS, IMAGEN, SIGNO, SINCRONIZADOS y USO.
    cláusulas especificadas para cada uno de estos elementos elementales.
15. La cláusula TYPE no se debe especificar en la misma entrada de descripción de datos con ninguna cláusula excepto BASED, CLASS, CONSTANT RECORD, DEFAULT, DESTINATION, entry-name, EXTERNAL, GLOBAL, INVALID, level-number, OCCURS, PRESENT WHEN, PROP- ERTY, TYPEDEF, VALIDATE-STATUS, VALUE y VARYING.
16. Figura 1. Ejemplo que muestra cómo se pueden utilizar las cláusulas TYPEDEF y TYPE

```
IDENTIFICACIÓN      DIVISIÓN.
PROGRAMA-ID.        prog.
DATOS               DIVISIÓN.
TRABAJO-ALMACENAMIENTO
                    SECCIÓN.
01 NOMBRE-DE-ARCHIVO-PRINCIPAL-TPIC X(50) ES TIPO DEF.
01 DETALLE-NO-T                Foto 9999  ES TYPEDEF.
  01 NOMBRE-DE-ARCHIVO-PRINCIPAL-2T  ES TYPEDEF.
     05 RELLENO    Foto 9999.
     05 DETALLE-NO TIPO A DETALLE-NO-T.

01    MENSAJE-TEXTO-2T ES TYPEDEF.
   02 NOMBRE DEL ARCHIVO PRINCIPAL        TIPO NOMBRE-ARCHIVO-PRINCIPAL-T.
   02 RELLENO REDEFINE EL NOMBRE DEL ARCHIVO PRINCIPAL.
         05 RELLENO PIC 9999.
   02 NOMBRE DEL ARCHIVO PRINCIPAL 2       TIPO NOMBRE-ARCHIVO-PRINCIPAL-2T.
   02 RELLENO                      TIPO NOMBRE-ARCHIVO-PRINCIPAL-T.

01    MENSAJE-TEXTO-2 TIPO EXTERNO MENSAJE-TEXTO-2T.
77 NOMBRE-SALIDA TIPO A DETALLE-NO-T GLOBAL.
01 Z-MESSAGE-T2 TIPO NOMBRE-ARCHIVO-PRINCIPAL-2T.
01 Z-MENSAJE-T3.
```

```
49 metros cúbicos                TIPO MENSAJE-TEXTO-2T.
49 MT3-REN     REDEFINE   MENSAJE TIPO MT3-TEXTO-2T.
```

```
DIVISIÓN DE PROCEDIMIENTOS.
    MOSTRAR NOMBRE DE ARCHIVO PRINCIPAL DEL
    TEXTO DEL MENSAJE 2 MOSTRARDETALLE-NOOF
    MENSAJE Z-T2 MOSTRAR NOMBRE DE ARCHIVO
    PRINCIPAL DE MT3
    MOSTRAR NOMBRE DE
    SALIDA GOBACK.
```

17. Figura 2. Ejemplo que muestra cómo se pueden utilizar las cláusulas TYPEDEF y TYPE

```
DIVISIÓN DE
DATOS.
SECCIÓN DE
ARCHIVO.
ARCHIVO PRT
FD.
01 PRT-REC.
    05 PRT-REGISTRO                  FOTO X(132).
01 INVE-TYP-T ES TIPO DEF            PIC S9(3) VALOR 0.
        88 INVE-TIPO-LIBRO                VALOR 4, 5.
    88  INVE-TIPO-LIBRO-001         VALOR  4.
    88  INVE-TIPO-LIBRO-002         VALOR  5.
    88  ROPA TIPO INVE              VALOR  1, 2, 3.
    88  INVE-TYP-ROPA-SUÉTERES      VALOR  1.
    88  INVE-TYP-ROPA-CALCETINES    VALOR  2.
    88  INVE-TYP-ROPA-PANTALONES    VALOR  3.

ENTRADA DE DATOS FD.
01 DATOS EN GRABACIÓN.
    05 TIPO INVE            TIPO INVE-TYP-T.
    05 RELLENO              FOTO X(80).

SECCIÓN DE TRABAJO-ALMACENAMIENTO.
01 ARTI-PRICE-T TYPEDEF PIC S9(4)V9(2) valor 0.
01 ARTI-COLOR-T TIPO DEF PIC S9(2) VALOR 1.
    88 ARTI-COLOR-AZUL      VALOR1 .
    88 ARTI-COLOR-ROJO      VALOR2 .
    88 ARTI-COLOR-VERDE     VALOR3 .

01   ARTI-TALLA-CA   IMAGEN DE   S9(2) VALOR 10.
                TIPO DEF
01   ARTE-CONTADOR-T TIPO DEF    S9(6) VALOR 0.
                    PIC

01  ARTE-BT TIPO DEF.
    05 ARTI-B-VALOR PIC s9(2).
        88 ARTI-B-AZUL     VALOR1 .
        88 ARTI-B-ROJO     VALOR2 .
        88 ARTI-B-VALOR VERDE 3.

01 PRUEBA-ARTI       TIPO ARTI-BT.
```

```
01 TRABAJO-INVESTIGACIÓN-TIPO    TIPO INVE-TYP-T.

01 ROPA-ARTI ES TIPODEF.
    05 TIPO DE ROPA          TIPO INVE-TYP-T.
```

```
    05 PRECIO              TIPO ARTI-PRECIO-T.
    05 COLOR               TIPO ARTI-COLOR-T.
    05 TALLA DE ROPA       TIPO ARTI-TALLA-T.
    05 RELLENO FOTO X(70).

01 PRENDAS TIPO SUÉTERES-ARTI.
01 CALCETINES      TIPO ROPA-ARTI.
01 PANTALONES      TIPO ROPA-ARTI.

01 LIBRO-ARTI ES TYPEDEF.
    05 TIPO LIBRO        TIPO INVE-TYP-T.
    05 PRECIO            TIPO ARTI-PRECIO-T.
    05 RELLENO           FOTO X(20).
    05 TÍTULO DEL LIBRO  FOTO X(40).
    05 RELLENO           FOTO X(14).

01 LIBRO-001            TIPO   LIBRO-ARTI.
01 LIBRO-002            TIPO   LIBRO-ARTI.

01 SUÉTERES-TIPO CONTADOR ARTI-CONTADOR-T.
01 CUENTA DE CALCETINES  TIPO    ARTE-CONTRA-T.
01 CUENTA PANTALONES     TIPO    ARTE-CONTRA-T.

01 LIBRO-001-CONTADOR TIPO ARTI-CONTADOR-T.
01 LIBRO-002-CONTADOR TIPO ARTI-CONTADOR-T.

01 CUENTA DE ROPA   TIPO    ARTE-CONTRA-T.
01 CUENTA DE LIBROS TIPO    ARTE-CONTRA-T.
```

Esta sintaxis es válida en las siguientes secciones: INFORME

Como formato 2

1. Esta cláusula define el tipo de grupo de informes que se está definiendo para un informe.
2. Esta cláusula es obligatoria en todas las definiciones de elementos de datos de nivel 01 (excepto las constantes de nivel 01) en la sección de informes. Esta cláusula no es válida en ninguna otra definición de elementos de datos de la sección de informes.
3. Puede haber un máximo de un (1) grupo de informes por RD definido con un TIPO de ENCABEZADO DE INFORME, ENCABEZADO DE PÁGINA, PIE DE PÁGINA y PIE DE INFORME.
4. Debe haber un ENCABEZAMIENTO DE CONTROL o un PIE DE CONTROL o ambos especificados para cada entrada especificada en la cláusula CONTROLS ARE del RD.
5. Los distintos grupos de informes que constituyen un informe pueden definirse en cualquier orden.
6. VerSección 9.1 [Léxico RWCS], página 651, para obtener una descripción de los siete tipos diferentes de grupos de informes.

6.9.56 Definición de tipo

Sintaxis de la cláusula TYPEDEF

```
01    nombre-datos-1 ES TYPEDEF
                  ~~ ~~~~~~
```

Esta sintaxis es válida en las siguientes secciones ARCHIVO, ALMACENAMIENTO DE TRABAJO, ALMACENAMIENTO LOCAL y ENLACE

1. La cláusula TYPEDEF identifica una declaración de tipo que crea un tipo de datos definido por el usuario y se utiliza para aplicar este tipo de datos definido por el usuario a la descripción de un elemento de datos.
2. La cláusula TYPEDEF especifica que la entrada de descripción de datos es una declaración de tipo.
3. La cláusula TYPEDEF solo se puede especificar para entradas de nivel 01, que también pueden ser elementos de grupo y para las que se especifica el formato de nombre de datos de la cláusula de nombre de entrada. Si se especifica la cláusula TYPEDEF para una descripción de datos, entonces esa misma descripción de datos debe incluir un nombre de datos, es decir, no se debe especificar con una cláusula FILLER implícita o explícita.
4. Si se especifica un elemento de grupo, todos los elementos subordinados del grupo pasan a formar parte de la declaración de tipo.
5. No se asigna ningún almacenamiento para una declaración de tipo.
6. Estas definiciones de tipo actúan como plantillas que luego pueden usarse para definir nuevos elementos de datos utilizando la cláusula TYPE o que posteriormente pueden referenciarse en una cláusula USAGE.
7. El nuevo elemento de datos adquiere todas las características del tipo de datos definido por el usuario. Si el tipo de datos definido por el usuario es un elemento de grupo, el nuevo elemento de datos tiene elementos subordinados con el mismo nombre, descripción y jerarquía que los que pertenecen al tipo de datos definido por el usuario.
8. Todas las demás cláusulas de descripción de datos, si se especifican, se asumen para cualquier elemento de datos que se defina utilizando el tipo de datos definido por el usuario (dentro de la cláusula TYPE o USAGE).
9. Las siguientes cláusulas no se pueden especificar junto con TYPEDEF: EXTERNAL, GLOBAL, LIKE, OCCURS, REDEFINES.
10. Si se especifica la cláusula TYPEDEF para un elemento de grupo, los elementos subordinados se pueden especificar con cláusulas OCCURS o REDEFINES. TYPEDEF no se puede utilizar con cláusulas OCCURS DEPENDING ON complejas. Esto significa que no se puede especificar una cláusula OCCURS DEPENDING ON dentro de una tabla que sea parte de una cláusula TYPEDEF.
11. La cláusula VALUE no se puede especificar tampoco en las descripciones de datos que especifican el TIPO.Cláusula DEF o en cualquier elemento subordinado excepto los nombres de condición (entradas de nivel 88) dentro de la estructura TYPEDEF.
12. La cláusula TYPE se puede especificar en la misma entrada de descripción de datos que la cláusula TYPEDEF, pero la descripción del objeto de la entrada, incluidos sus elementos subordinados, no debe contener una cláusula TYPE que haga referencia directa o indirectamente a esta definición de tipo.
13. En la SECCIÓN DE ARCHIVO, una entrada de descripción de datos declarada en el nivel número 1 que incluye una cláusula TYPEDEF no es una entrada de descripción de registro.

6.9.57 SUBRAYAR

Sintaxis del atributo UNDERLINE

SUBRAYAR
~~~~~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula UNDERLINE introducirá una línea horizontal en el borde inferior de un campo de pantalla.

1. LA LÍNEA IZQUIERDA (verSección 6.9.28 [LEFTLINE], página 183), SOBRELÍNEA (verSección 6.9.36 [SOBRELINEA], página 194) y las cláusulas UNDERLINE se pueden utilizar en cualquier combinación en la descripción de un solo campo.
2. Esta cláusula es esencialmente no funcional cuando se utiliza dentro del shell de comandos de Windows.(cmd.exe) entornos y programas en ejecución compilados utilizando una implementación de GnuCOBOL creada con "PDCurses" (como las compilaciones de Windows/MinGW).
3. El funcionamiento de esta cláusula en sistemas Cygwin o UNIX/Linux/OSX dependerá de las capacidades de los atributos de video de los controladores de salida del terminal y del software "curses" que se utilice.

VerSección 2.1.12.2 [Paleta de colores y atributos de vídeo], página 21, para obtener más información sobre los colores de la pantalla y los atributos de vídeo.
~~~~~~~~~

6.9.58 SUPERIOR

Sintaxis de la cláusula UPPER

SUPERIOR
~~~~~

———————————————————————————————————————

Esta sintaxis es válida en las siguientes secciones: PANTALLA

La cláusula UPPER fuerza que los datos aceptados estén en mayúsculas.
~~~~~

6.9.59 USO

Sintaxis de la cláusula USAGE

USO ES uso-de-elementos-de-datos
~~~~~

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, TRABAJO-ALMACENAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME

La cláusula USAGE define el formato que se utilizará para almacenar el valor de un elemento de datos.

1. La palabra reservada IS es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. La siguiente tabla resume las diversas especificaciones de USO disponibles en GnuCOBOL.

BINARIO
~~~~~~

Rango de valores: definido por la cantidad de '9' y la presencia o ausencia de una 'S' en la IMAGEN

Formato de almacenamiento: ¿Se permiten valores

binarios enteros negativos compatibles?: Si IMAGEN

contiene 'S' ¿IMAGEN utilizada?: Sí

BINARIO-C-LARGO [FIRMADO]
~~~~~~~~~~~~~

Rango de valores: Dependiendo del hardware del sistema, el rango es de 0 a 2**31 o de 0 a 2**63.

BINARIO-C-LARGO SIN SIGNO
~~~~~~~~~~~~~~ ~~~~~~~~

Rango de valores: Dependiendo del hardware del sistema, el rango es de 0 a 2**32 o de 0 a 2**64.

Restricciones: Este USO solo debe usarse para LLAMADAS directas a C y no debe usarse de otra manera (y no se compilará con ningún -std estricto ya que es una extensión exclusiva de GnuCOBOL).

CHAR BINARIO [FIRMADO]
~~~~~~~~~~~

Rango de valores: -128 – 127

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: Sí
~~~~~~~~~~~

IMAGEN utilizada?: No

CHAR BINARIO SIN FIRMAR
~~~~~~~~~~~ ~~~~~~~~

Rango de valores: 0– 255

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

BINARIO-DOBLE [CON SIGNO]
~~~~~~~~~~~~~

Rango de valores: -9.223.372.036.854.775.808 – 9.223.372.036.854.775.807

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

BINARIO-DOBLE SIN SIGNO
~~~~~~~~~~~~~ ~~~~~~~~

Rango de valores: 0 – 18.446.744.073.709.551.615

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

BINARIO-INT
~~~~~~~~~~

Igual que BINARY-LONG SIGNED

BINARY-LONG [SIGNED]
~~~~~~~~~~~

Rango de valores: -2.147.483.648 – 2.147.483.647

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

BINARIO LARGO SIN SIGNO
~~~~~~~~~~~ ~~~~~~~~

Rango de valores: 0 – 4.294.967.295

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

BINARIO-LARGO-LARGO
~~~~~~~~~~~~~~~~

Igual que BINARIO-DOBLE SIGNO

BINARIO-CORTO [CON SIGNO]
~~~~~~~~~~~~

Rango de valores: -32.768 – 32.767

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

BINARIO-CORTO SIN FIRMA
~~~~~~~~~~~~ ~~~~~~~~

Rango de valores: 0 – 65.535

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

COMPUTACIONALCO
MP

Igual que
BINARIO

COMPUTACIONAL-1
COMP-1

Igual que FLOAT-
SHORT COMPUTATIONAL-2
COMP-2

Igual que FLOAT-
LONG COMPUTATIONAL-3
COMP-3

Igual que PACKED-DECIMAL
~~~~~~~~~~~~

COMPUTACIONAL-4
COMP-4

Igual que

BINARIO

COMPUTACIONAL-5
COMP-5

Rango de valores: Depende de la cantidad de "9" en la IMAGEN y de la configuración de tamaño binario del archivo de configuración utilizado para compilar el programa.

Formato de almacenamiento: ¿Se permiten valores

binarios nativos enteros negativos?: Si la IMAGEN contiene

'S'

IMAGEN utilizada?: Sí

COMPUTACIONAL-6
COMP-6

Rango de valores: definido por la cantidad de '9' y la presencia o ausencia de una 'S' en la IMAGEN

Formato de almacenamiento: Decimal empaquetado sin signo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: Sí

COMPUTACIONAL-X
COMP-X

Rango de valores: Si se utiliza con PIC X, se asigna un byte de almacenamiento por cada "X"; el rango de valores es de 0 al máximo que se puede almacenar en esa cantidad de bytes. Si se utiliza con PIC 9, el rango de valores depende de la cantidad de "9" en PICTURE

Formato de almacenamiento: ¿Se permiten valores binarios negativos

nativos sin signo (X) o con signo (9)?: Si la IMAGEN 9 contiene 'S'

IMAGEN utilizada?: Sí

MOSTRAR
~~~~~~~

Rango de valores: Depende de la IMAGEN: un carácter por X, A, 9,
~~~~~~~

punto, $, Z, 0, *, S (si es CARÁCTER SEPARADO) especificado), +, - o símbolo B en IMAGEN; Agregue 2 bytes más si se utiliza el símbolo de edición DB o CR

Formato de almacenamiento: Personajes

¿Valores negativos permitidos?: Si la IMAGEN contiene 'S'

IMAGEN utilizada?: Sí

FLOTANTE-DECIMAL-16
~~~~~~~~~~~~~~~~

Rango de valores: -9.999999999999999 * 10383 – 9.999999999999999 * 10^{384}

Formato de almacenamiento: Punto flotante Decimal64 IEEE 754 nativo

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

FLOTANTE-DECIMAL-34
~~~~~~~~~~~~~~~~

Rango de valores: -9.99999... * 106143 – 9.99999... * 106144

Formato de almacenamiento: Punto flotante decimal IEEE 754 nativo 128

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

FLOTADOR LARGO
~~~~~~~~~~

Rango de valores:Aproximadamente -1.797693134862316 * 10308 – 1.797693134862316 * 10308

Formato de almacenamiento: Punto flotante nativo IEEE 754 Binary64

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

FLOTADOR-CORTO
~~~~~~~~~~~

Rango de valores:Aproximadamente -3,4028235 * 1038 – 3,4028235 * 1038

Formato de almacenamiento: IEEE 754 nativo Binary32

¿Valores negativos permitidos?: Sí

IMAGEN utilizada?: No

ÍNDICE
~~~~~

Rango de valores: 0 a la máxima dirección posible (32 o 64 bits)

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

NACIONAL
~~~~~~~~

USO NACIONAL, aunque reconocido sintácticamente, no es compatible con GnuCOBOL

DECIMAL EMPAQUETADO
~~~~~~~~~~~~~~

Rango de valores: Definido por la cantidad de '9' y la presencia o ausencia de una 'S' en la IMAGEN

Formato de almacenamiento: ¿Se permiten valores

decimales negativos con signo? Si IMAGEN contiene 'S'

¿IMAGEN utilizada?: Sí

PUNTERO
~~~~~~~

Rango de valores: 0 a la máxima dirección posible (32 o 64 bits)

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

INSTRUCCIONES SOBRE EL PROCEDIMIENTO
~~~~~~~~~~~~~~~~~~

Igual que PROGRAM-
POINTER PROGRAM-POINTER
~~~~~~~~~~~~~~~~

Rango de valores: 0 a la máxima dirección posible (32 o 64 bits)

Formato de almacenamiento: Número entero binario nativo

¿Valores negativos permitidos?: No

IMAGEN utilizada?: No

FIRMADO-INT
~~~~~~~~~~

Igual que BINARY-LONG SIGNED

SIGNED-LONG
~~~~~~~~~~~

Igual que BINARIO-DOBLE SIGNO

SIGNO-CORTO
~~~~~~~~~~~~

Igual que BINARY-SHORT SIGNED

UNSIGNED-INT
~~~~~~~~~~~~

Igual que BINARY-LONG

UNSIGNED UNSIGNED-LONG
~~~~~~~~~~~~~

Igual que BINARY-DOUBLE

UNSIGNED UNSIGNED-SHORT
~~~~~~~~~~~~~~

Igual que BINARY-SHORT UNSIGNED

3. USO ES tipo-nombre-1
   ~~~~~ ~~
4. Esta cláusula USAGE indica que la descripción de los datos del objeto de la entrada se especifica mediante un tipo de datos definido por el usuario. El tipo de datos definido por el usuario se define mediante la cláusula TYPEDEF, que se describe en la cláusula TYPEDEF.

```
Ejemplo 1:

01 estructura-1   TIPO DEF.
     05   Parte 1  imagen
     x(20).05      Parte 2
          imagen x(10).
01   a.
     05   b         USO struct-1.
     05   incógnita USO USHORT.
01 USHORT           imagen 9(04) comp-5 tipo definido.

Lo cual se interpretaría como si hubiera sido codificado

 como: 01       a.
       05   b.
            10   parte 1 foto x(20).
            10   parte 2 foto x(10).
       05   incógnita      foto 9(04) comp-5.
```
   ~~~~~

Ejemplo 2:

```
IDENTIFICACIÓN      DIVISIÓN.
PROGRAMA-ID.        prog.
DATOS               DIVISIÓN.
TRABAJO-ALMACENAMIENTO
                    SECCIÓN.
01 NOMBRE-DE-ARCHIVO-PRINCIPAL-T      PIC X(50) ES TIPO DEF.
01 DETALLE-NO-T              Foto 9999  ES TYPEDEF.
01 NOMBRE-DE-ARCHIVO-PRINCIPAL-2T  ES TYPEDEF.
    05 RELLENO               Foto 9999.
    05 DETALLE-NO                 DETALLE DE USO-NO-T.

01   MENSAJE-TEXTO-2T                    ES TYPEDEF.
  02 NOMBRE DEL ARCHIVO PRINCIPAL      USO NOMBRE-ARCHIVO-PRINCIPAL-T.
  02 RELLENO                 REDEFINE EL NOMBRE DEL ARCHIVO PRINCIPAL.
       05 RELLENO             Foto 9999.
  02 NOMBRE DEL ARCHIVO PRINCIPAL 2      USO   NOMBRE-ARCHIVO-PRINCIPAL-2T.
  02 RELLENO                   USO NOMBRE-ARCHIVO-PRINCIPAL-T.

01   MENSAJE-TEXTO-2   MENSAJE DE USO EXTERNO-TEXTO-2T.

77 NOMBRE DE SALIDA            USO   DETALLE-NO-T GLOBAL.

01 Z-MENSAJE-T2                USO   NOMBRE-ARCHIVO-PRINCIPAL-2T.
01 Z-MENSAJE-T3.
    49 metros cúbicos          USO   MENSAJE-TEXTO-2T.
    49 MT3-REN REDEFINE EL USO DEL MT3 MENSAJE-TEXTO-2T.

DIVISIÓN DE PROCEDIMIENTOS.
      MOSTRAR NOMBRE DE ARCHIVO PRINCIPAL DEL
      TEXTO DEL MENSAJE 2 MOSTRARDETALLE-NOOF
      MENSAJE Z-T2 MOSTRAR NOMBRE DE ARCHIVO
      PRINCIPAL DE MT3
      MOSTRAR NOMBRE DE SALIDA
      GOBACK.
```

Ejemplo 3:

```
77   INT                       PIC S9(09) COMP-5 ES TIPO DEF.
01   CÓDIGO DE RETORNO Z       USO  INT VALOR 0.
01   MENSAJE-TEXTO-2                     ES TYPEDEF.
     02 NOMBRE DEL ARCHIVO PRINCIPAL   FOTO X(50).
     02 RELLENO                FOTO X(79).
01 Z-MESSAGE-T2 USO MENSAJE-TEXTO-2.
77 VIRMSG-TEXTO                FOTOX(129) ES TIPO DEF.
01   MENSAJE-TEXTO-2                     ES TYPEDEF.
     02 NOMBRE DEL ARCHIVO PRINCIPAL   FOTO X(50).
     02 RELLENO                FOTO X(79).
01 Z-MENSAJE-TEXTO             USO  VIRMSG-TEXTO.
01 Z-MESSAGE-T2 REDEFINE EL USO DEL MENSAJE Z-TEXTO MENSAJE-TEXTO-2.
01   W102-TIPO ES TYPEDEF.
```

```
COPIA CAMPOS DEL
SERVIDOR.
```

```
01 TA                    USO  TIPO W102.
01 TN                    USO  TIPO W102.
```

5. Los datos binarios (enteros o de punto flotante) se pueden almacenar en formato Big-Endian o Little-Endian.
   forma.

   La asignación de datos big-endian requiere que los bytes que componen un elemento binario se asignen de manera que el byte menos significativo sea el byte más a la derecha. Por ejemplo, un elemento binario de cuatro bytes que tenga un valor decimal de 20 se asignaría big-endian como 00000014 (mostrado en notación hexadecimal).

   La asignación de datos little-endian requiere que los bytes que componen un elemento binario se asignen de manera que el byte menos significativo sea el byte más a la izquierda. Por ejemplo, un elemento binario de cuatro bytes que tenga un valor decimal de 20 se asignaría little-endian como 14000000 (mostrado en notación hexadecimal).

   Todas las CPU son capaces de comprender el formato big-endian, lo que lo convierte en la forma más compatible de almacenamiento binario en todos los sistemas informáticos.

   Algunas CPU (como los procesadores de arquitectura Intel/AMD i386/x64 que se utilizan en la mayoría de las PC con Windows) prefieren procesar datos binarios almacenados en un formato little-endian. Dado que ese formato es más eficiente en esos sistemas, se lo conoce como formato binario nativo.

   En un sistema que solo admite un formato de almacenamiento binario (generalmente, sería big-endian), los términos formato más eficiente y formato nativo son sinónimos.
6. Los elementos de datos que tienen el atributo UNSIGNED codificado explícitamente, o los elementos DISPLAY, PACKED-DECIMAL, COMP-5, COMP-X que no tienen un símbolo "S" en su cláusula de imagen no pueden conservar los valores negativos que se puedan almacenar en ellos. Almacenar un valor negativo en un campo de este tipo en realidad dará como resultado que se elimine el signo, lo que esencialmente guardará el valor absoluto en el elemento de datos.
7. Los datos decimales empaquetados (es decir, USAGE PACKED-DECIMAL, USAGE COMP-3 o USAGE COMP-6) se almacenan como una serie de bytes de modo que cada byte contiene dos campos de 4 bits, denominados nibbles (ya que comprenden la mitad de un "byte", son simplemente "nibbles"; no se queje, ¡no invento estas cosas!). Cada nibble representa un '9' en la IMAGEN y cada uno contiene un solo dígito decimal codificado como su valor binario (0 = 0000, 1 = 0001, . . . , 9 = 1001).

   El último byte de un elemento de datos PACKED-DECIMAL o COMP-3 siempre tendrá su nibble izquierdo correspondiente al último '9' de la IMAGEN y su nibble derecho reservado como indicador de signo. Este indicador de signo siempre está presente independientemente de si la IMAGEN incluye o no un símbolo 'S'.

   El primer byte del elemento de datos contendrá un nibble izquierdo no utilizado si la IMAGEN tenía un número par de símbolos '9' en ella.

   El indicador de signo tendrá un valor hexadecimal de A a F. Las reglas tradicionales de codificación decimal empaquetada requieren valores hexadecimales de F, A, C o E ("FACE") en el nibble del signo para indicar un valor positivo y B o D para representar un valor negativo (los dígitos hexadecimales 0-9 no están definidos). Prueba con una implementación de MinGW/GnuCOBOL para Windowsmuestra que, de hecho, el dígito hexadecimal D representa un número negativo y cualquier otro dígito hexadecimal denota un número positivo. Por lo tanto, un campo decimal empaquetado PIC S9(3) COMP-3 con un valor de -15 se almacenaría internamente como un 015D hexadecimal en GnuCOBOL.

   Si intenta almacenar un número negativo en un campo decimal empaquetado que no tiene 'S' en su

IMAGEN, en realidad se almacenará el valor absoluto del número negativo.

USAGE COMP-6 no permite valores negativos, por lo tanto, no se asignará ningún nibble de signo. Un elemento de datos USAGE COMP-6 que contenga una cantidad impar de símbolos '9' en su IMAGEN dejará su nibble más a la izquierda sin usar.

8. Las especificaciones de USO FLOAT-DECIMAL-16 y FLOAT-DECIMAL-34 codificarán los datos utilizando el formato IEEE 754 Decimal64 y Decimal128, respectivamente. El primero permite hasta 16 dígitos de precisión exacta, mientras que el segundo ofrece 34. La frase "precisión exacta" se utiliza porque las representaciones binarias tradicionales de números reales decimales en un formato de punto flotante (FLOAT-LONG y FLOAT-SHORT, por ejemplo) solo producen una aproximación del valor real porque muchas fracciones decimales no se pueden representar con precisión en binario. Sin embargo, las representaciones Decimal64 y Decimal128 representan números reales decimales en forma decimal codificada de la misma manera que PACKED-DECIMAL representa un entero decimal en forma decimal dígito por dígito. La manera exacta en que se realiza esta representación es compleja (Wikipedia tiene un excelente artículo sobre el tema; solo busque Decimal64).
9. GnuCOBOL almacena elementos de datos FLOAT-DECIMAL-16 y FLOAT-DECIMAL-34 utilizando el formato Big-Endian o Little-Endian, el que sea nativo del sistema.
10. Las especificaciones de USO FLOAT-LONG y FLOAT-SHORT utilizan el estándar IEEE 754 Binary64 y*Binario32*formatos, respectivamente. Se trata de codificaciones binarias de números decimales reales y, como tales, no pueden representar todos los valores posibles entre los valores mínimo y máximo en el rango para esos usos. Wikipedia tiene un excelente artículo sobre los esquemas de codificación Binary64 y Binary32; solo busque Binary32 o Binary64.

    GnuCOBOL almacena elementos de datos FLOAT-LONG y FLOAT-SHORT utilizando el formato Big-Endian o Little-Endian, el que sea nativo del sistema.
11. Una cláusula USAGE especificada a nivel de elemento de grupo aplicará ese USAGE a todos los elementos de datos subordinados, excepto aquellos que tengan una cláusula USAGE.
12. El único USO que se permite en la sección de informe es VISUALIZACIÓN DE USO.

6.9.60 USANDO

Sintaxis de la cláusula USING

USANDO identificador-1
~~~~~

Esta sintaxis es válida en las siguientes secciones: PANTALLA

Esta cláusula adjunta lógicamente un elemento de datos de una sección de pantalla a otro elemento de datos definido en otra parte de la división de datos.

1. Cuando se muestra el elemento de pantalla cuya definición forma parte de esta cláusula, el valor que se encuentra actualmente en el identificador-1 se moverá automáticamente al elemento de pantalla primero.
2. Cuando se acepta el elemento de pantalla cuya definición forma parte de esta cláusula (o su padre), el contenido actual del elemento de pantalla se guardará nuevamente en el identificador-1 al finalizar la ACEPTACIÓN.
3. El FROM (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220), USING y VALUE (verSección 6.9.61 [VALOR], página 239) Las cláusulas son mutuamente excluyentes en la definición de cualquier elemento de datos de la sección de pantalla.
~~~~~

6.9.61 VALOR

Sintaxis de la cláusula VALOR (nombres de condiciones)

```
{ EL VALOR ES        } {literal-1 [ HASTA|HASTA literal-2 ]}...
  ~~~~~                ~~~~ ~~~~~~~~
{ LOS VALORES SON }
  ~~~~~~
     [CUANDO SE ESTABLECE COMO FALSO ES literal-3 ]
                        ~~~~~
```

Sintaxis de VALOR (Otros elementos de datos)

```
EL VALOR ES [ TODOS ] literal-1
~~~~~       ~~~
```

Sintaxis de VALOR (para tablas)

```
{ EL VALOR ES                 } {{literal-1}..          DESDE ({subíndice-1}..)      [ A
({subíndice-2}..              )]}..
  ~~~~~                       ~~~~                      ~~
{ LOS VALORES SON }
  ~~~~~~
```

Esta sintaxis es válida en lo siguiente:secciones:ARCHIVO, ALMACENAMIENTO EN FUNCIONAMIENTO, ALMACENAMIENTO LOCAL, VINCULACIÓN, INFORME, PANTALLA

La cláusula VALUE se utiliza para definir nombres de condiciones o para asignar valores (en tiempo de compilación) a elementos de datos.

1. Las palabras reservadas ARE e IS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Esta cláusula no se puede especificar en el mismo elemento de datos que un FROM (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220) o UTILIZANDO (verSección 6.9.60 [USO], página 238) cláusula.
3. Los siguientes puntos se aplican al uso de la cláusula VALUE en la definición de un nombre de condición:
   A. Las cláusulas VALUE IS y VALUES ARE son intercambiables.
   B. Las palabras reservadas THRU y THROUGH son intercambiables.
   C. VerSección 6.8.5 [Elementos de datos de nivel 88], página 153, para una discusión sobre cómo se utiliza este formato de VALOR para crear nombres de condiciones.
   D. VerSección 2.1.27.1 [Nombres de las condiciones], página 44, para una discusión sobre cómo se utilizan los nombres de condiciones.
4. Los siguientes puntos se aplican al uso de la cláusula VALUE en la definición de cualquier otro elemento de datos:

A. En este contexto, VALOR especifica un valor de tiempo de compilación inicial que se asignará al almacenamiento ocupado por el elemento de datos en el código objeto del programa generado por el compilador.
B. La cláusula VALUE se ignora en EXTERNAL (verSección 6.9.19 [EXTERNO], página 173) elementos de datos o sobre cualquier elemento de datos definido como subordinado a un elemento de datos EXTERNO.
C. Este formato de la cláusula VALUE no se puede utilizar en ninguna parte de la descripción de un elemento 01 (o cualquiera de sus elementos subordinados) que sirva como descripción de registro FD o SD.
D. Si se utiliza la cláusula ALL opcional, solo se puede utilizar con un valor literal alfanumérico; el valor se repetirá según sea necesario para completar por completo el elemento de datos. A continuación, se muestran algunos ejemplos con y sin ALL (el símbolo b denota un espacio):

```
PIC X(5) VALOR 'A'              *> Abbbb
PIC  X(5)  VALOR  TODOS  'A'  *>  AAAAA
PIC 9(3) VALOR 1                *> 001
PIC 9(3) VALOR TODO '1' *> 111
```

E. Cuando se utiliza en la definición de un elemento de datos de pantalla:
   a. Una constante figurativa no puede suministrarse como literal-1.
   b. Cualquier DE (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220) o UTILIZANDO (verSección 6.9.60 [USO], página 238) se ignorará la cláusula en la definición del mismo elemento de datos.
   c. Si no se especifica ninguna cláusula de imagen, el tamaño del elemento de datos de la pantalla será la longitud del valor literal 1.
   d. Si no hay ninguna cláusula de imagen y se especifica la opción ALL, se ignorará esta opción.
F. Darle a una tabla un valor inicial en tiempo de compilación es uno de los aspectos más complicados de la definición de datos COBOL. Básicamente, existen tres técnicas estándar y una cuarta que puede resultar interesante para quienes estén familiarizados con otras implementaciones COBOL pero sean nuevos en GnuCOBOL. A continuación, se presentan los tres enfoques estándar:
   a. No se preocupe por ello en tiempo de compilación. Utilice INITIALIZE (consulteSección 7.8.24 [INICIALIZAR], página 339) para inicializar todas las ocurrencias de elementos de datos en una tabla (en tiempo de ejecución) a sus valores predeterminados específicos del tipo de datos (numéricos: 0, alfabéticos y alfanuméricos: espacios).
   b. Inicialice tablas pequeñas en tiempo de compilación incluyendo una cláusula VALUE en el elemento del grupo que sirve como padre de la tabla, de la siguiente manera:

```
05   TALLAS DE CAMISA        VALOR "S 14M 15L 16XL17".
     10 TALLA DE CAMISA TBL          OCURRE  4 VECES.
        15 TAMAÑO SST     FOTO X(2).
        15 CUELLO DE ACERO INOXIDABLE   Foto 9(2).
```

   c. Inicialice tablas de casi cualquier tamaño en el momento de la compilación utilizando REDEFINES (verSección 6.9.41 [REDEFINE], página 205) cláusula:

```
05   VALORES DE TALLA DE CAMISA.
     10 FOTOS X(4)           VALOR "S 14".
     10 FOTOS X(4)           VALOR  "M15".
     10 FOTOS X(4)           VALOR "L 16".
     10 FOTOS X(4)           VALOR" X L 1 7 " .
```

```
05    TALLAS DE CAMISA        REDEFINE LOS VALORES DE LAS TALLAS DE LAS CAMISAS.
      10 TALLA DE CAMISA TBL          OCURRE  4 VECES.
          15 TAMAÑO SST     FOTO X(2).
          15 CUELLO DE ACERO INOXIDABLE   Foto 9(2).
```

Es cierto que esta tabla es mucho más detallada que la que se muestra con un grupo VALUE. Sin embargo, lo bueno de esta técnica de inicialización es que puede tener tantos elementos FILLER y VALUE como necesite para una tabla más grande, ¡y esos valores pueden ser tan largos como sea necesario!

G. Muchos compiladores COBOL no permiten el uso de VALUE y OCCURS (verSección 6.9.35 [OCURRE], página 191) en el mismo elemento de datos; además, no permiten una cláusula VALUE en un elemento de datos subordinado a un OCCURS. ¡Sin embargo, GnuCOBOL no tiene ninguna de estas restricciones!

Observe el siguiente ejemplo, que ilustra una cuarta manera en que se pueden inicializar tablas en GnuCOBOL:

```
05  incógnita     OCURRE 6 VECES.
    10 A          PIC X(1) VALOR '?'.
    10 B          PIC X(1) VALOR '%'.
    10 N          FOTO 9(2) VALOR 10.
```

En este ejemplo, los seis elementos "A" se inicializarán en "?", los seis elementos "B" se inicializarán en "%" y los seis elementos "N" se inicializarán en 10. No está claro exactamente cuántas veces será útil este tipo de inicialización, pero está ahí si lo necesita.

5. El FROM (verSección 6.9.22 [DESDE], página 176), A (verSección 6.9.54 [TO], página 220), USANDO (verSección 6.9.60 [USO], página 238) y las cláusulas VALUE son mutuamente excluyentes en la definición de cualquier elemento de datos de la sección de pantalla.
6. Formato-3 (Tabla): La cláusula de valor para tablas especifica el valor inicial de los elementos de datos de la sección de almacenamiento de trabajo y de la sección de almacenamiento local.

Ejemplo 1: Cláusula de valor de tabla

```
03  Nivel 1        ocurre 10 veces
    05  Nivel 2    ocurre 3 veces PIC X.
                       valor "A" "B" "C"DE (2 2) a (3 1)
                             "X" "Y" "Z"DESDE (1 1) hasta (! 3)
                             espacio        DE (3 2).
```

Ejemplo 2: Cláusula de valor de tabla con

```
inicialización 01     Elementos del grupo.
    03                    Valor "Relleno
    implícito". 03        relleno      Foto 1
        valor B"1".
    03  Tabla de variables ocurre 10 veces
          foto 9   valor 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (1) a (10).
- - -
 inicializar elementos del grupo
            con relleno
              Todo a valorar.
```

———————————————————————

Fin del Capítulo 6 — DIVISIÓN DE DATOS

7 DIVISIÓN DE PROCEDIMIENTOS

DIVISIÓN DE PROCEDIMIENTOS Sintaxis

```
DIVISIÓN DE PROCEDIMIENTO [ { USANDO Argumento-Subprograma ...    } ]
~~~~~~~~~ ~~~~~~~~      { ~~~~~~                                  }
                        { ENCADENANDO argumento-del-programa-principal...}
                                  ~~~~~~~~~
                          [ DEVOLVIENDO
identificador-1 ]. [ DECLARATIVOS. ]
                                  ~~~~~~~~~~
~~~~~~~~~~~~
[Rutina del controlador de eventos]. ]

[ FIN DE LOS DECLARATIVOS. ]
~~~ ~~~~~~~~~~~~~~
Lógica general del
programa

[Subprograma anidado.... ]

[ FIN DEL PROGRAMA | FUNCIÓN nombre-1 ]
~~~ ~~~~~~~ ~~~~~~~~
```

La DIVISIÓN DE PROCEDIMIENTOS de cualquier programa GnuCOBOL marca el punto donde se escribe todo el código ejecutable.

7.1 DIVISIÓN DE PROCEDIMIENTOS UTILIZANDO

Sintaxis de argumentos de subprograma de DIVISIÓN DE PROCEDIMIENTOS

```
[ POR { REFERENCIA [ OPCIONAL ]                          } ] identificador-1
      { ~~~~~~~~~~ ~~~~~~~~                              }
      { VALOR [ [ SIN FIRMA ] EL TAMAÑO ES { AUTO   } ] }
        ~~~~~      ~~~~~~~~   ~~~~     { ~~~~       }
                                   { POR DEFECTO     }
                                       { ~~~~~~~~ }
                                       { entero-1 }
```

La cláusula USING define los argumentos que se pasarán a un programa GnuCOBOL que actúa como subprograma.

1. Las palabras reservadas BY e IS son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La cláusula USING sólo debe utilizarse en el encabezado de división de procedimientos de subprogramas (subrutinas o funciones definidas por el usuario).
3. El programa que realiza la llamada pasará cero o más elementos de datos, conocidos como argumentos, a este subprograma; debe haber exactamente tantos elementos de datos de identificador-1 especificados en la cláusula USING como el número máximo de argumentos que alguna vez se le pasarán al subprograma.
4. Si un subprograma no espera ningún argumento, no debe tener una cláusula USING especificada en su encabezado de división de procedimiento.
5. El orden en que se definen los argumentos en la cláusula USING debe corresponder al orden en que el programa que los llama pasará esos argumentos al subprograma.
6. Los identificadores especificados en la cláusula USING deben definirse en la sección de enlace del subprograma. En realidad, no se asigna ningún espacio de almacenamiento para esos identificadores en el subprograma, ya que el almacenamiento real para ellos existirá en el programa que los llama.
7. Un subprograma GnuCOBOL espera que todos los argumentos sean una de dos cosas:
   - La dirección de memoria del elemento de datos real (asignado en el programa de llamada) que se pasa al subprograma.
   - Un valor binario numérico de palabra completa (es decir, USO BINARIO-LARGO (verSección 6.9.59 [EE. UU.- EDAD], página 228)) que es el argumento real que se pasa al subprograma.

   En el caso del primero, la cláusula USING en el encabezado de división de procedimiento debe describir el argumento a través de la cláusula BY REFERENCE; en el segundo caso, se debe codificar una especificación BY VALUE. Esto permite que el código generado por el compilador haga referencia de manera adecuada a los argumentos del subprograma en tiempo de ejecución.
8. BY REFERENCE es el valor predeterminado asumido para el primer argumento USING si no se especifica una cláusula BY para él. Los argumentos subsiguientes asumirán la especificación BY del argumento anterior si no tienen una cláusula BY propia.
9. Los cambios realizados por un subprograma al valor de un argumento especificado en la cláusula USING serán "visibles" para el programa que lo llama solo si se especificó BY REFERENCE explícitamente o se asumió implícitamente para el argumento en el encabezado

de división de procedimiento del subprograma y el argumento fue pasado al subprograma BY REFERENCE por el programa que lo llama. ConsulteSección 11.6.1 [Argumentos del subprograma], página 725, para obtener información adicional sobre la mecánica de cómo se pasan los argumentos a los subprogramas.

10. La cláusula opcional SIZE le permite especificar la cantidad de bytes que ocupará un argumento BY VALUE, donde SIZE DEFAULT especifica 4 bytes (este es el valor predeterminado si no se utiliza ninguna cláusula SIZE), SIZE AUTO especifica el tamaño del argumento en el programa que lo llama y SIZE entero-1 especifica un recuento de bytes específico.
11. La palabra clave opcional UNSIGNED, válida solo si se codifica SIZE AUTO o SIZE entero-1, agregará el atributo unsigned a la especificación del argumento en el código de encabezado de función en lenguaje C generado para el subprograma. Si bien no es de ninguna utilidad cuando el programa que realiza la llamada es un programa GnuCOBOL, esto puede mejorar la compatibilidad con un programa que realiza la llamada en lenguaje C.
12. La palabra clave OPCIONAL, válida solo en argumentos POR REFERENCIA, permite llamar a programas para que codifiquen OMITIDO ese argumento correspondiente cuando llaman a este subprograma. VerSección 7.8.5 [LLAMADA], página 290. para obtener información adicional sobre esta función.

7.2 DIVISIÓN DE PROCEDIMIENTOS ENCADENAMIENTO

DIVISIÓN DE PROCEDIMIENTOS Sintaxis de argumentos del programa principal

[POR REFERENCIA] [OPCIONAL] identificador-1

El término ENCADENAMIENTO proporciona un mecanismo que un programador puede utilizar para recuperar argumentos de la línea de comandos pasados a un programa en el momento de ejecución.

1. El ENCADENAMIENTO POR DIVISIÓN DE PROCEDIMIENTOS solo se puede codificar en un programa principal (es decir, el primer programa que se ejecuta cuando se ejecuta una unidad de compilación GnuCOBOL compilada). No se puede utilizar en ninguna forma de subprograma.
2. La cláusula CHAINING define los argumentos que se pasarán a un programa principal desde el sistema operativo. Los identificadores de argumentos especificados en la cláusula CHAINING se completarán con cadenas de caracteres compuestas por los parámetros especificados al programa en la línea de comandos que lo ejecutó, de la siguiente manera:
   A. Cuando se ejecuta un programa GnuCOBOL desde una línea de comandos, el texto completo de la línea de comandos se dividirá en una serie de símbolos, donde cada símbolo se identifica como una palabra separada de las demás en el texto del comando por al menos un espacio. Por ejemplo, si la línea de comandos fuera /usr/local/myprog THIS IS A TEST, habrá cinco símbolos identificados por el sistema operativo: '/usr/local/myprog', 'THIS', 'IS', 'A' y 'TEST'.
   B. Los tokens delimitados por espacios múltiples se pueden tratar como un solo token encerrándolos entre comillas. Por ejemplo, solo hay tres tokens generados desde la línea de comando C:\Pgms\myprog.exe 'THIS IS A' TEST: 'C:\Pgms\myprog.exe', 'THIS IS A' y 'TEST'. Cuando se utilizan caracteres de comillas para crear tokens de varias palabras, los caracteres de comillas en sí se eliminan del valor del token.
   C. Una vez que se han identificado los tokens, se descartará el primero (el comando); el resto se almacenará en los argumentos de CHAINING cuando el programa comience a ejecutarse, y el segundo token irá al primer argumento, el tercer token irá al segundo argumento y así sucesivamente.
   D. Si hay más tokens que argumentos, los tokens sobrantes se descartarán.
   E. Si hay menos tokens que argumentos, los argumentos sobrantes se inicializarán como si el identificador INITIALIZE fuera 1 (verSección 7.8.24 [INICIALIZAR], página 339) se ejecutaron las declaraciones.
   F. Todos los identificadores especificados en la cláusula CHAINING deben definirse como PIC X, PIC A, elementos de grupo (que se tratan implícitamente como PIC X) o como PIC 9 USAGE DISPLAY. No se recomienda el uso de elementos de datos USAGE BINARY (o similares) como argumentos CHAINING ya que todos los tokens de línea de comandos se conservarán en su formato de carácter original a medida que se mueven a los elementos de datos de argumentos.
   G. Si un identificador de argumento tiene un tamaño de almacenamiento menor que el valor del token que se almacenará en él, los caracteres sobrantes más a la derecha del valor del token se truncarán a medida que se mueva el valor hacia adentro. Cualquier cláusula JUSTIFIED RIGHT en dicho identificador de argumento será ignorada.
   H. Si un argumento es más grande en tamaño de almacenamiento que el valor del token

que se almacenará en él, el valor del token se moverá al identificador del argumento de manera justificada a la izquierda. sin modificar: las posiciones de bytes modificadas en el identificador se llenarán con espacios, a menos que el identificador del argumento

El identificador se define como PIC 9 USAGE DISPLAY, en cuyo caso los bytes no modificados se rellenarán con caracteres '0' del conjunto de caracteres nativos del sistema.

Este comportamiento cuando el argumento se define como PIC 9 puede ser inaceptable, ya que un argumento definido como PIC 9(3) pero que se pasa en un valor de '1' desde la línea de comandos recibirá un valor de '100', no '001'. Considere definir argumentos de línea de comandos "numéricos" como PIC X y luego usar la función intrínseca NUMVAL (consulteSección 8.1.70 [NUMVAL], página 500) función para determinar el valor numérico adecuado.

7.3 DIVISIÓN DE PROCEDIMIENTOS DEVOLUCIÓN

PROCEDIMIENTO DIVISIÓN RETORNO Sintaxis

DEVOLVIENDO identificador-1
~~~~~~~~~

———————————————————————————————————————

La cláusula RETURNING en el encabezado PROCEDURE DIVISION documenta que el subprograma en el que aparece la cláusula devolverá un valor numérico al programa que la llamó. Esto solo está disponible para funciones, ya que no funciona para programas y presenta problemas según la plataforma (sistema operativo) en uso. Debe probar esto para la plataforma específica.

1. La cláusula RETURNING es opcional dentro de una subrutina, ya que no todas las subrutinas devuelven un valor a quien las llama.
2. La cláusula RETURNING es obligatoria dentro de una función definida por el usuario, ya que todas ellas deben devolver un resultado numérico.
3. El elemento de datos identificador-1 debe definirse como un elemento de datos USAGE BINARY-LONG.
4. Los programas principales que desean "devolver" un valor de código de retorno al sistema operativo cuando salen no utilizan RETURNING; lo hacen simplemente MOVEANDO un valor al registro especial RETURN-CODE.
5. Este no es el único mecanismo que puede utilizar un subprograma para devolver un valor a quien lo llama. Otras posibilidades son:
   A. El subprograma puede modificar cualquier argumento que se especifique como POR REFERENCIA en su encabezado DIVISIÓN DE PROCEDIMIENTO. El hecho de que el programa que realiza la llamada pueda realmente "ver" las modificaciones depende de cómo el programa que realiza la llamada le haya pasado el argumento al subprograma. ConsulteSección 7.8.5 [LLAMADA], página 290, para obtener más información.
   B. Un elemento de datos con el GLOBAL (verSección 6.9.24 [GLOBAL], página 178) el atributo especificado en su descripción en el programa que lo llama es automáticamente visible y actualizable por un subprograma anidado con el programa que lo llama. VerSección 11.2 [Independiente vs. Subprogramas contenidos y subprogramas anidados, página 719, para obtener más información sobre la anidación de subprogramas.
   C. Un elemento de datos definido con el EXTERNO (verSección 6.9.19 [EXTERNO], página 173) en un subprograma y el programa que lo llama (mismo nombre en ambos programas) es automáticamente visible y actualizable por ambos programas, incluso si esos programas se compilan por separado uno del otro.
~~~~~~~~~

7.4 DIVISIÓN DE PROCEDIMIENTOS Secciones y Párrafos

La división de procedimientos es la única de las divisiones de COBOL que le permite crear sus propias secciones y párrafos. Estos se denominan colectivamente procedimientos y los nombres que crea para esas secciones y párrafos se denominan nombres de procedimientos.

Los nombres de procedimientos son opcionales en la división de procedimientos y, cuando se utilizan, se nombran enteramente de acuerdo con las necesidades y caprichos del programador.

Los nombres de los procedimientos pueden tener hasta treinta y un (31) caracteres y pueden estar compuestos por letras, números, guiones y guiones bajos. El nombre de un procedimiento no puede comenzar ni terminar con un guión ('-') o un guión bajo ('_'). Esto significa que Main, 0100-Read-Transaction y 17 son nombres de procedimientos perfectamente válidos.

Existen tres circunstancias en las que el uso de determinadas instrucciones u opciones de GnuCOBOL requerirá la especificación de procedimientos. Estas situaciones son:

1. Cuando DECLARATIVOS (verSección 7.5 [DECLARATIVAS], página 250) se especifican.
2. Cuando la instrucción ENTRY (verSección 7.8.14 [ENTRADA], página 319) está siendo utilizado.
3. Cuando se utiliza cualquier declaración de división de procedimiento que haga referencia a procedimientos. Estas declaraciones son:
   - ALTER nombre-del-procedimiento
   - IR A nombre-del-procedimiento
   - FUSIONAR... PROCEDIMIENTO DE SALIDA nombre-procedimiento
   - REALIZAR nombre-del-procedimiento
   - ORDENAR... PROCEDIMIENTO DE ENTRADA nombre-procedimiento y/o ORDENAR... PROCEDIMIENTO DE SALIDA
     nombre-del-procedimiento

7.5 DECLARATIVOS

Sintaxis de los DECLARATIVOS

```
[DECLARATIVAS.                                          ]
[ ~~~~~~~~~~~~~~                                        ]
[ {nombre-sección-1                                     ]
SECCIÓN. [        ~~~~~~~~                              ]
[Declaración USE].
[                                                       ]
[ [oración]...[nombre-párrafo-1.    [oración]...]...}...] [
                                    ]
[ FIN DE LOS DECLARATIVOS.                              ]
[ ~~~~~~~~~~~~~~~~~~~                                   ]
```

Donde la declaración USE puede tener uno de cuatro formatos:

1. Procedimiento de EXCEPCIONES DE ARCHIVO, 2. Procedimiento de DEPURACIÓN,
3. Procedimiento de REPORTE a ejecutar antes de la impresión del Grupo de Informe designado,
4. CONDICIONES DE EXCEPCIÓN procedimientos que se ejecutarán después de la detección de condiciones de excepción.

```
UTILICE { [ GLOBAL ] DESPUÉS DEL PROCEDIMIENTO ESTÁNDAR { EXCEPCIÓN } EN { ENTRADA  } }
~~~ {     ~~~~~~                    { ~~~~~~~~~ }                  { ~~~~~            } }
    {                               { ERROR     }                  { PRODUCCIÓN       } }
    {                               { ~~~~~     }                  { ~~~~~~           } }
    {                                                              { IO               } }
    {                                                              { ~~~              } }
    {                                                              { EXTENDER         } }
    {                                                              { ~~~~~~           } }
 {                                                            { {nombre-archivo-1 }..} }
    {                                                                                   }
    {                                                                                   }
    { PARA DEPURACIÓN EN { nombre-del-procedimiento-1    }                              }
   {       ~~~~~~~~~      { TODOS LOS PROCEDIMIENTOS                                 }   }
    {                       { ~~~ ~~~~~~~~~~~           }                               }
    {                       { REFERENCIAS DE identificador-1 }                          }
    {                                                                                   }
    { [ GLOBAL ] ANTES DE INFORMAR identificador-2                                      }
    {   ~~~~~~    ~~~~~~ ~~~~~~~~~                                                      }
    {                                                                                   }
    { DESPUÉS DE {EC|CONDICIÓN DE EXCEPCIÓN} {nombre-de-excepción-1                   } }
{         ~~ ~~~~~~~~~ ~~~~~~~~~     {nombre-excepción-2 {ARCHIVO nombre-archivo-2}..} } }
    {                                                          ~~~~                     }
```

Las cláusulas AFTER EXCEPTION CONDITION y AFTER EC se reconocen sintácticamente pero por lo demás no son funcionales.

El área DECLARATIVOS de la división de procedimientos permite al programador definir una serie de procedimientos "trampa" (denominados procedimientos declarativos) capaces de interceptar determinados eventos que pueden producirse durante la ejecución del programa. El diagrama de sintaxis anterior muestra el formato de un solo procedimiento de este tipo.

1. Las palabras reservadas AFTER, FOR, ON, PROCEDURE y STANDARD son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. CE y CONDICIÓN DE EXCEPCIÓN son intercambiables.
3. El área declarativa puede contener cualquier número de procedimientos declarativos, pero no se deben codificar dos procedimientos declarativos para atrapar el mismo evento.
4. Los siguientes puntos se aplican a la cláusula USE BEFORE REPORTING:
   A. *identificador-2*Debe ser un grupo de informes.
   B. En tiempo de ejecución, el procedimiento declarativo se ejecutará antes del procesamiento de la presentación del grupo de informes especificado; dentro del procedimiento puede realizar cualquiera de las siguientes acciones:
      - Puede ajustar el valor de cualquier elemento referenciado en SUM (verSección 8.1.92 [SUMA], página 524) o FUENTE (verSección 6.9.49 [FUENTE], página 214) cláusulas en el grupo de informes.
      - Puedes ejecutar la función SUPRESIÓN (verSección 7.8.49 [SUPRIMIR], página 402) para suprimir por completo la presentación del grupo de informes especificado. Tenga en cuenta que suprimirá esta instancia específica de la presentación de ese grupo y no todas.
5. Los siguientes puntos se aplican a la cláusula USE FOR DEBUGGING:
   A. Esta cláusula le permite definir un procedimiento declarativo que se invocará siempre que...
      - ... se hace referencia al identificador-1 en cualquier declaración.
      - . . . se ejecuta el procedimiento-nombre-1.
      - ...se ejecuta cualquier procedimiento (TODOS LOS PROCEDIMIENTOS).
   B. UN USO PARAEl procedimiento declarativo DEBUGGING se ignorará en el momento de la compilación a menos que se especifique WITH DEBUGGING MODE en SOURCE-COMPUTER (verSección 5.1.1 [ORDENADOR FUENTE], página 87) párrafo. Ni el modificador de línea -fdebugging- ni el modificador -debug del compilador activarán esta función.
   C. Cualquier procedimiento declarativo USE FOR DEBUGGING se ignorará en el momento de la ejecución a menos que la variable de entorno de tiempo de ejecución COB_SET_DEBUG (verSección 10.2.3 [Tiempo de ejecución Variables de entorno], página 700) se ha establecido en un valor de 'Y', 'y' o '1'.
   D. El uso típico de un procedimiento declarativo USE FOR DEBUGGING es mostrar el registro especial DEBUG-ITEM, que se creará de forma implícita y automática en su programa si WITH DEBUGGING MODE está activo.

      La estructura de DEBUG-ITEM será la siguiente:

```
01 DEPURAR-ARTÍCULO.
   05 LÍNEA DE DEPURACIÓN      FOTO X(6).
   05 RELLENO                  PIC X(1) ESPACIO DE VALOR.
   05 NOMBRE DE DEPURACIÓN     FOTO X(31).
   05 RELLENO                  PIC X(1) ESPACIO DE VALOR.
   05 DEPURACIÓN-SUB-1          PIC S9(4) SEÑALIZACIÓN LÍDER SEPARADA.
   05 RELLENO                  PIC X(1) ESPACIO DE VALOR.
   05 DEPURACIÓN-SUB-2          PIC S9(4) SEÑALIZACIÓN LÍDER SEPARADA.
   05 RELLENO                  PIC X(1) ESPACIO DE VALOR.
```

```
05 DEPURACIÓN-SUB-3  PIC S9(4) SEÑALIZACIÓN LÍDER SEPARADA.
05 RELLENO           PIC X(1) ESPACIO DE VALOR.
05 DEBUG-CONTENIDO   FOTO X(31).
```

dónde. . .

LÍNEA DE DEPURACIÓN
: . . . es el número de línea del programa de la declaración que activó el procedimiento declarativo.

NOMBRE DE DEPURACIÓN
: . . . es el nombre del procedimiento o el nombre del identificador que activó el procedimiento declarativo.

DEBUG-SUB-1
: . . . es el primer valor de subíndice (si lo hay) para la referencia del identificador que activó el procedimiento declarativo.

DEPURACIÓN-SUB-2
: . . . es el segundo valor de subíndice (si lo hay) para la referencia del identificador que activó el procedimiento declarativo.

DEPURACIÓN-SUB-3
: . . . es el tercer valor de subíndice (si lo hay) para la referencia del identificador que activó el procedimiento declarativo.

DEBUG-CONTENIDO
: ... es una declaración (breve) de la manera en que se ejecutó el procedimiento que activó el procedimiento declarativo o los primeros 31 caracteres del valor del identificador cuya referencia activó el procedimiento declarativo (el valor después de que se ejecutó la declaración).

6. La cláusula USE AFTER STANDARD ERROR PROCEDURE define un procedimiento declarativo que se invoca cada vez que se detecta una falla con el tipo de E/S especificado (o contra el/los archivo(s) especificado(s).
7. El GLOBAL (verSección 6.9.24 [GLOBAL], página 178) La opción, si se utiliza, permite utilizar un procedimiento declarativo en todo el programa que contiene la declaración USE y cualquier subprograma anidado dentro de ese programa.
8. Los procedimientos declarativos no pueden hacer referencia a ningún otro procedimiento definido fuera del alcance de DECLARATIVOS.

7.6 Cláusulas comunes en sentencias ejecutables

7.6.1 AL FINAL + NO AL FINAL

Sintaxis AT END

[AL FINAL declaración imperativa 1]

[NO AL FINAL declaración imperativa-2]

Las cláusulas AT END se pueden especificar en READ (verSección 7.8.34 [LEER], página 365), REGRESAR (verSección 7.8.38 [RETORNO], página 372), BUSCAR (verSección 7.8.41 [BÚSQUEDA], página 376) y BUSCAR TODO (verSección 7.8.42 [BUSCAR TODO], página 377) declaraciones.

1. Los siguientes puntos se refieren al uso de estas cláusulas en READ (verSección 7.8.34 [LEER], página 365) y RETORNO (verSección 7.8.38 [RETORNO], página 372) declaraciones:
   A. La cláusula AT END, si está presente, provocará la declaración imperativa 1 (ver[Imperativo Declaración], página 758) que se ejecutará si la instrucción falla debido a un estado de archivo de 10 (fin de archivo). Ver[Códigos de estado de archivo], página 105, para obtener una lista de posibles códigos de estado de archivo.

      Una cláusula AT END no detectará otros valores de estado de archivo distintos de cero.

      Utilice un DECLARATIVO (verSección 7.5 [DECLARATIVAS], página 250) rutina o unacampo de estado de archivo declarado explícitamente probado después de READ o RETURN para detectar condiciones de error distintas al final del archivo.
   B. Una cláusula NOT AT END hará que se ejecute la declaración imperativa 2 si se realiza la instrucción READ o El intento de RETORNO fue exitoso.
2. Los siguientes puntos se refieren al uso de estas cláusulas en SEARCH (verSección 7.8.41 [BÚSQUEDA], página 376) y BUSCAR TODO (verSección 7.8.42 [BUSCAR TODO], página 377) declaraciones:
   A. Una cláusula AT END detecta y maneja el caso en el que alguna de las formas de búsqueda de tabla no ha podido localizar una entrada que satisfaga las condiciones de búsqueda utilizadas.
   B. La cláusula NOT AT END no está permitida en ninguna forma de búsqueda de tabla.

7.6.2 CORRESPONDIENTE

Tres sentencias GnuCOBOL: ADD (verSección 7.8.2.3 [AGREGAR CORRESPONDIENTE], página 286), MOVER (verSección 7.8.30.2 [MOVIMIENTO CORRESPONDIENTE], página 355) ySUSTRAER
(verSección 7.8.48.3 [RESTAR CORRESPONDIENTE], página 401) apoyan el uso de una Opción CORRESPONDIENTE:

AGREGAR EL ELEMENTO-GRUPO-1 CORRESPONDIENTE AL
ELEMENTO-GRUPO-2 MOVER EL ELEMENTO-GRUPO-1
CORRESPONDIENTE AL ELEMENTO-GRUPO-2
RESTA EL CORRESPONDIENTE grupo-elemento-1 DEL grupo-elemento-2

Esta opción permite que uno o más elementos de datos dentro de un elemento de grupo (elemento de grupo 1, el primero nombrado en la declaración) se emparejen con el elemento de nombre correspondiente (de ahí el nombre) en un segundo elemento de grupo (elemento de grupo 2, el segundo nombrado en la declaración). El contenido del elemento de grupo 1 no se verá afectado por la declaración, mientras que uno o más elementos de datos dentro del elemento de grupo 2 se modificarán.

Para que el elemento de datos 1, definido como subordinado al elemento de grupo elemento de grupo 1, coincida con el elemento de datos 2, que es subordinado al elemento de grupo 2, cada una de las siguientes condiciones debe ser verdadera:

1. Tanto el elemento de datos 1 como el elemento de datos 2 deben tener el mismo nombre, y ese nombre no puede ser explícita o implícitamente FILLER.
2. Tanto el elemento de datos 1 como el elemento de datos 2...
   A. ... debe existir en la misma "profundidad" estructural relativa de definición dentro del elemento-grupo-1
   y grupo-ítem-2, respectivamente
   B. ... y todos los elementos de datos "principales" definidos dentro de cada elemento del grupo deben tener nombres idénticos (pero no de RELLENO).
3. Cuando se usa con un verbo MOVE...
   A. ...se permite que uno de los elementos de datos 1 o 2 (pero no ambos) sea un elemento de grupo
   B. ... y debe ser válido mover el elemento de datos 1 al elemento de datos 2.
4. Cuando se utiliza con los verbos ADD o SUBTRACT, tanto el elemento de datos 1 como el elemento de datos 2 deben ser elementos numéricos, elementales y sin editar.
5. Ni el elemento de datos 1 ni el elemento de datos 2 pueden ser REDEFINES (verSección 6.9.41 [REDEFINE], página 205) o CAMBIA EL NOMBRE (verSección 6.9.42 [RENOMBRAMIENTOS], página 206) de otro elemento de datos.
6. Ni el elemento de datos 1 ni el elemento de datos 2 pueden tener un OCCURS (verSección 6.9.35 [OCURRE], página 191) cláusula, aunque cualquiera de ellas puede contener elementos de datos subordinados que sí tienen una cláusula OCCURS (asumiendo que se aplica la regla 3a)

Observe las definiciones de los elementos de datos 'Q' e 'Y'. . .

```
01  Q.                          01  Y.
    03 X.                           02  A        FOT X(1).
                                                   O
       05  A        FOT 9(1).       02  G1.
            O
       05  G1.                        03  G2.
           10  G2.                       04 B     FOT X(1).
                                                   O
               15 B     FOT X(1).      02  do     FOT X(1).
```

```
                O                                O
05 D                          02 G3.
   O.
   10 RELLE  FOT X(1).           03 G5.
      NO     O
05 G3.                              04 D    FOT X(1).
                                             O
   10 G4.                         03 G6     FOT X(1).
                                             O
      15 D    FOT X(1).          02 Y       FOT 9(1).
              O                              O
05 Y          FOT X(1).          02 F       FOT X(1).
              O                              O
05 F          REDEFINES V1       02 G       FOTO X(4).
```

```
                    FOTO X(1).        02 yo          OCURRE 4 VECES
05 g.                                                FOTO X(1).
    10 G6           OCURRE 4 VECES    66 I           CAMBIA EL
                                                     NOMBRE DE E.
                    FOTO X(1).        02 Y
                                         O.
05 horas            FOTO X(4).           03 K.
05 yo               Foto 9(1).              04 litros.
05 J.                                          05M.
    10 mil.
        15METRO     FOTO X(1).
```

Las siguientes son las coincidencias CORRESPONDIENTES válidas, asumiendo que se está ejecutando la declaración MOVE CORRESPONDING X TO Y (no hay coincidencias correspondientes válidas para ADD CORRESPONDING o SUBTRACT CORRESPONDING porque cada coincidencia potencial viola la regla n.° 4):

A, B, C, G

Los siguientes son los enfrentamientos CORRESPONDIENTES que pasaron la regla n.° 1 (pero fallaron en otra regla) y las razones por las cuales fallaron.

Elemento de datos	**Motivo del fallo**
D	Falla debido a la regla n.° 2b
E	Falla debido a la regla n.° 3b
F	Falla debido a la regla n.° 5
G1	Falla debido a la regla n.° 3a
G2	Falla debido a la regla n.° 3a
G3	Falla debido a la regla n.° 3a
G4	Falla debido a la regla n.° 1
G5	Falla debido a la regla n.° 1
G6	Falla debido a la regla n.° 6
H	Falla debido a la regla n.° 6
I	Falla debido a la regla n.° 5
J	Falla debido a la regla n.° 3a
K	Falla debido a la regla n.° 3a
L	Falla debido a la regla n.° 1
M	Falla debido a la regla n.° 2a

7.6.3 CLAVE INVÁLIDA + CLAVE NO INVÁLIDA

Sintaxis de CLAVE NO VÁLIDA

```
[ CLAVE NO VÁLIDA declaración-imperativa-1 ]
  ~~~~~~~
[ CLAVE NO VÁLIDA declaración-imperativa-2 ]
  ~~~ ~~~~~~~
```

Se pueden especificar cláusulas de CLAVE INVÁLIDAS en DELETE (verSección 7.8.11

[ELIMINAR], página 301),LEER (verSección 7.8.34.2 [LECTURA Aleatoria], página 367), REESCRIBIR (verSección 7.8.39 [REESCRIBIR], página 373), INICIO (verSección 7.8.45 [INICIO], página 393) y ESCRIBIR (verSección 7.8.54 [ESCRIBIR], página 410) declaraciones.

La especificación de una cláusula INVALID KEY permitirá que su programa detecte una condición de falla de E/S (con un código de error de E/S en el FILE-STATUS del archivo (verSección 5.2.1 [SELECCIONAR], página 102) campo) que se ha producido debido a una condición de registro no encontrado y manejarlo con elegancia ejecutando la declaración imperativa 1 (ver[Declaración imperativa], página 758).

Una cláusula opcional NOT INVALID KEY hará que se ejecute la declaración imperativa 2 si la ejecución de la declaración fue exitosa.

7.6.4 CON EXCEPCIÓN + SIN EXCEPCIÓN

Sintaxis ON EXCEPTION

```
[ ON EXCEPTION declaración imperativa 1 ]
      ~~~~~~~~~~
[NO BAJO EXCEPCIÓN declaración imperativa 2 ]
  ~~~      ~~~~~~~~~~
```

Se pueden especificar cláusulas de EXCEPCIÓN en ACCEPT (verSección 7.8.1 [ACEPTAR], página 263), LLAMA (verSección 7.8.5 [LLAMADA], página 290) y DISPLAY (verSección 7.8.12 [PANTALLA], página 302) declaraciones.

La especificación de una cláusula de excepción permitirá que su programa detecte una condición de falla que haya ocurrido y la maneje con elegancia ejecutando la declaración imperativa 1 (ver[Imperativo Declaración], página 758). Si se produce una condición de este tipo en tiempo de ejecución sin que se haya especificado una de estas cláusulas, se generará un mensaje de error (por la biblioteca de tiempo de ejecución de GnuCOBOL) en el dispositivo SYSERR (tubería 2). El programa también puede terminarse, según el tipo y la gravedad del error.

Una cláusula opcional NOT ON EXCEPTION hará que se ejecute la declaración imperativa 2 si la ejecución de la declaración fue exitosa.

7.6.5 EN DESBORDAMIENTO + NO EN DESBORDAMIENTO

Sintaxis ON OVERFLOW

```
[ ON OVERFLOW declaración imperativa 1 ]
      ~~~~~~~~~
[NO EN DESBORDAMIENTO declaración imperativa-2 ]
  ~~~      ~~~~~~~~~
```

Las cláusulas OVERFLOW se pueden especificar en CALL (verSección 7.8.5 [LLAMADA], página 290), CADENA (verSección 7.8.47 [STRING], página 397) y UNSTRING (verSección 7.8.53 [UNSTRING], página 406) declaraciones.

Una cláusula ON OVERFLOW permitirá que su programa detecte una condición de falla que haya ocurrido y la maneje con elegancia ejecutando la declaración imperativa 1 (ver[Declaración imperativa], página 758). Si se produce una condición de este tipo en tiempo de ejecución sin que se haya especificado una de estas cláusulas, se generará un mensaje de error (por la biblioteca de tiempo de ejecución de GnuCOBOL) en el dispositivo SYSERR (tubería 2). El programa también puede terminarse, según el tipo y la gravedad del error.

Una cláusula opcional NOT ON OVERFLOW hará que se ejecute la declaración imperativa 2 si la ejecución de la declaración fue exitosa.

7.6.6 ERROR DE TAMAÑO + ERROR DE TAMAÑO NO

Sintaxis de ERROR DE TAMAÑO

```
[ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
      ~~~~ ~~~~~
[ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
  ~~~      ~~~~ ~~~~~
```

Se pueden incluir cláusulas de ERROR DE TAMAÑO en ADD (verSección 7.8.2 [ADD], página 282), COMPUTAR (verSección 7.8.9 [COMPUTE], página 297), DIVIDIR (verSección 7.8.13 [DIVIDIR], página 313), MULTIPLICA (verSección 7.8.31 [MULTIPLICAR], página 356) y RESTA (verSección 7.8.48 [SUB- TRACT], página 399) declaraciones.

Incluir una cláusula ON SIZE ERROR en una declaración aritmética permitirá que su programa detecte una falla de una declaración aritmética (ya sea generando un resultado demasiado grande para el campo receptor o intentando dividir por cero) y lo maneje con elegancia ejecutando la declaración imperativa 1 (ver[Declaración imperativa], página 758). Las condiciones de desbordamiento del tamaño de campo ocurren silenciosamente, generalmente sin que se generen mensajes de tiempo de ejecución, aunque dichos eventos rara vez se prestan a generar resultados correctos. Los errores de división por cero, cuando no existe una cláusula ON SIZE ERROR, producirán un mensaje de error (por parte de la biblioteca de tiempo de ejecución de GnuCOBOL) al dispositivo SYSERR (tubería 2) y también abortarán el programa.

Una cláusula opcional NOT ON SIZE ERROR hará que se ejecute la declaración imperativa 2 si la ejecución de la declaración aritmética fue exitosa.

7.6.7 REDONDEADO

Sintaxis ROUNDED

```
REDONDEADO [ LA          LEJOS DEL CERO                    }
MODA ES {                ~~~~~~~~~~~~~~~
~~~~~~~    ~~~~        {                                   }
                       { MÁS CERCANO-LEJOS-DE-CERO
                       { }
                       { ~~~~~~~~~~~~~~~~~~~~~~~~~~ }
                       { MÁS CERCANO-PAR                   }
                       { ~~~~~~~~~~~~~
                       {                                   }
                       { MÁS CERCANO A CERO                }
                       { ~~~~~~~~~~~~~~~~~~~~~~            }
                       { PROHIBIDO                         }
                       { ~~~~~~~~~~~                       }
                       { HACIA MAYOR                       }
                       { ~~~~~~~~~~~~~~~                   }
                       { HACIA MENOR                       }
                         ~~~~~~~~~~~~~~                    }
                         TRUNCAMIENTO                      }
                         ~~~~~~~~~~
```

———————————————————————————————————————

GnuCOBOL permite controlar el proceso de redondeo final aplicado a los campos de recepción en todos los verbos aritméticos. Cada una de las instrucciones aritméticas (ADD (verSección 7.8.2 [ADD], página 282), COMPUTAR (verSección 7.8.9 [COMPUTE], página 297), DIVIDIR (verSección 7.8.13 [DIVIDIR], página 313), MULTIPLICA (verSección 7.8.31 [MULTIPLICAR], página 356) y RESTA (ver
Sección 7.8.48 [RESTAR], página 399)) Las declaraciones permiten que se aplique una cláusula ROUNDED opcional a cada elemento de datos recibido.

Las siguientes reglas se aplican al comportamiento de redondeo inducido por esta cláusula.

1. El redondeo solo se aplica cuando el resultado se guarda en un campo receptor con una cláusula ROUNDEDes un valor no entero.
2. La ausencia de una cláusula ROUNDED es lo mismo que especificar ROUNDED MODE IS TRUNCATION.
3. El uso de una cláusula ROUNDED sin una especificación MODE es lo mismo que especificar ROUNDED MODE IS NEAREST-AWAY-FROM-ZERO.

El comportamiento de los ocho modos de redondeo diferentes se define en la siguiente tabla. Tenga en cuenta que un '...' indica que se repite el último dígito. Los ejemplos suponen un campo de recepción de números enteros.

LEJOS DEL CERO

El redondeo se realiza al valor más cercano de mayor magnitud.

-3.510 ⇒-4	+3.510 ⇒ +4
-3.500 ⇒-4	+3.500 ⇒ +4
-3.499. . . ⇒-4	+3.499. . . ⇒+4
-2.500 ⇒-3	+2.500 ⇒ +3
-2.499. . . ⇒-3	+2.499. . . ⇒+3

MÁS CERCANO-A-CERO

El redondeo se realiza al valor más cercano (mayor o menor). Si dos valores están igualmente cerca, se selecciona el valor con el valor absoluto más alto.

-3.510 ⇒-4	+3.510 ⇒ +4
-3.500 ⇒-4	+3.500 ⇒ +4
-3.499. . . ⇒-3	+3.499. . . ⇒+3
-2.500 ⇒-3	+2.500 ⇒ +3
-2.499. . . ⇒-2	+2.499. . . ⇒+2

MÁS CERCANO-PAR

El redondeo se realiza al valor más cercano (mayor o menor). Si dos valores están igualmente cerca, se selecciona el valor cuyo dígito más a la derecha sea par. Este modo a veces se denomina "redondeo bancario".

-3.510 ⇒-4	+3.510 ⇒ +4
-3.500 ⇒-4	+3.500 ⇒ +4
-3.499. . . ⇒-3	+3.499. . . ⇒+3
-2.500 ⇒-2	+2.500 ⇒ +2
-2.499. . . ⇒-2	+2.499. . . ⇒+2

MÁS CERCANO A CERO

El redondeo se realiza al valor más cercano (mayor o menor). Si dos valores están igualmente cerca, se selecciona el valor con el valor absoluto más pequeño.

-3.510 ⇒-4	+3.510 ⇒ +4
-3.500 ⇒-3	+3.500 ⇒ +3
-3.499. . . ⇒-3	+3.499. . . ⇒+3
-2.500 ⇒-2	+2.500 ⇒ +2

-2.499. . . ⇒-2 +2.499. . . ⇒+2

PROHIBIDO

No se realiza ningún redondeo. Si el valor no se puede representar exactamente en el formato deseado, se establece la condición EC-SIZE-TRUNCATION (código de excepción 1005) (y se puede recuperar mediante ACCEPT (consulteSección 7.8.1.7 [ACEPTAR DESDE Runtime-Info], página 279) y los resultados de la operación no están definidos.

-3.510 ⇒Indefinido	+3.510 ⇒ Indefinido
-3.500 ⇒Indefinido	+3.500 ⇒ Indefinido
-3.499. . . ⇒Indefinido	+3.499. . . ⇒Indefinido
-2.500 ⇒Indefinido	+2.500 ⇒ Indefinido
-2.499. . . ⇒Indefinido	+2.499. . . ⇒Indefinido

HACIA-MAYOR QUE

El redondeo se realiza hacia el valor más cercano cuyo valor algebraico sea mayor.

-3.510 ⇒-3	+3.510 ⇒ +4
-3.500 ⇒-3	+3.500 ⇒ +4
-3.499. . . ⇒-3	+3.499. . . ⇒+4
-2.500 ⇒-2	+2.500 ⇒ +3
-2.499. . . ⇒-2	+2.499. . . ⇒+3

HACIA MENOR

El redondeo se realiza hacia el valor más cercano cuyo valor algebraico sea menor.

-3.510 ⇒-4	+3.510 ⇒ +3
-3.500 ⇒-4	+3.500 ⇒ +3
-3.499. . . ⇒-4	+3.499. . . ⇒+3
-2.500 ⇒-3	+2.500 ⇒ +2
-2.499. . . ⇒-3	+2.499. . . ⇒+2

TRUNCAMIENTO

El redondeo se realiza al valor más cercano cuya magnitud sea menor.

-3.510 ⇒-3	+3.510 ⇒ +3
-3.500 ⇒-3	+3.500 ⇒ +3
-3.499. . . ⇒-3	+3.499. . . ⇒+3
-2.500 ⇒-2	+2.500 ⇒ +2
-2.499. . . ⇒-2	+2.499. . . ⇒+2

7.7 Registros especiales

GnuCOBOL, al igual que otros dialectos COBOL, incluye una serie de elementos de datos que están disponibles automáticamente para un programador sin necesidad de definirlos en la división de datos. COBOL se refiere a dichos elementos como registros o registros especiales. Los registros especiales disponibles para un programa GnuCOBOL son los siguientes:

ESTADO DEL COB-CRT
: PIC 9(4) — Este es el elemento de datos predeterminado asignado para su uso por la declaración de elemento de datos ACCEPT (verSección 7.8.1.4 [ACEPTAR elemento de datos], página 267), si no hay ESTADO CRT (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) se especificó la cláusula.

ELEMENTO DE DEPURACIÓN
: Elemento de grupo: un elemento de grupo en el que la información de depuración generada por una sección USE FOR DEBUGGING en el área de declaraciones de la división de procedimientos colocará información que documenta por qué se invocó el procedimiento USE FOR DEBUGGING. Consulte las DECLARATIVAS (consulteSección 7.5 [DECLARATIVAS], página 250) documentación para obtener información sobre la estructura de este registro.

CONTADOR DE LINAJE
: BINARY-LONG SIGNED: existe una ocurrencia de este registro para cada archivo seleccionado que tenga un LINAJE (consulteSección 6.2.1 [Archivo/Ordenar-Descripción], página 123). Si hay varios archivos cuyas descripciones de archivo tienen cláusulas LINAGE, cualquier referencia explícita a este registro requerirá una calificación (utilizando OF file-name). El valor de este registro será el número de línea lógica actual dentro del cuerpo de la página. El valor de este registro no se puede modificar.

CONTADOR DE LINEA
: EN BINARY-LONG SIGNED — Existe una ocurrencia de este registro para cada informe definido en el programa (a través de un RD (verSección 6.6 [SECCIÓN DEL INFORME], página 135)). Si hay varios informes, cualquier referencia explícita a este registro que no se haya hecho en elLa sección de informe requerirá una calificación (OF report-name). El valor de este registro será el número de línea lógica actual en la página actual. El valor de este registro no se puede modificar.

PARÁMETROS DE NÚMERO DE LLAMADA
: BINARY-LONG SIGNED: este registro contiene la cantidad de argumentos pasados a una subrutina, el mismo valor que devolvería la subrutina del sistema incorporada C$NARG (consulteSección 8.2.9 [C$NARG], página 556). Su valor será cero cuando se haga referencia a él en un programa principal. Este registro, cuando se hace referencia a él desde dentro de una función definida por el usuario, devuelve un valor de uno ('1') si la función tiene cualquier número de argumentos y un cero si no tiene argumentos.

CONTADOR DE PAGINAS
: BINARY-LONG SIGNED: existe una ocurrencia de este registro para cada informe que tenga un RD (consulteSección 6.6 [SECCIÓN DEL INFORME], página 135). Si existen varios informes de este tipo, cualquier referencia explícita a este registro que no se haga en la sección del informe se consideraráRequiere calificación (OF nombre-informe). El valor de este registro será el número de página del informe actual. El valor de este registro no se puede modificar.

CÓDIGO DE RETORNO

BINARY-LONG SIGNED: este registro proporciona un elemento de datos numéricos al que una subrutina puede MOVERSE (consulteSección 7.8.30 [MOVE], página 354) un valor (que luego estará disponible para el programa que lo llamó) antes de transferir el control nuevamente al programa que lo llamó, o al cual un programa principal puede MOVER un valor antes

devolver el control al sistema operativo. Muchas subrutinas integradas devolverán un valor utilizando este registro. Estos valores se utilizan, por convención, para indicar el éxito (normalmente con un valor de 0) o el fracaso (normalmente con un valor distinto de cero) del proceso que el programa intentaba realizar. Este registro también puede ser modificado por un subprograma como resultado del uso de RETURNING por parte de ese subprograma (consulteSección 7.3 [DIVISIÓN DE PROCEDIMIENTOS DE REGRESO], página 248) cláusula.

ORDENAR-RETORNO

BINARY-LONG SIGNED: este registro se utiliza para informar el estado de éxito/fracaso de una LIBERACIÓN (consulteSección 7.8.36 [LIBERACIÓN], página 370) o RETORNO (verSección 7.8.38 [REGRESAR], página 372). Se informa un valor de 0 en caso de éxito. Un valor de 16 denota un error. Una instrucción AT END (consulteSección 7.6.1 [AL FINAL + NO AL FINAL], página 253) La condición de una DEVOLUCIÓN no se considera un fracaso.

CUANDO-COMPILADO

PIC X(16) — Este registro contiene la fecha y la hora en que se compiló el programa en el formato 'mm/dd/aaaaa.mm.ss'. Tenga en cuenta que solo se proporciona un año de dos dígitos.

LONGITUD DE Sintaxis

LONGITUD DEL literal-numérico-1 | identificador-1
~~~~~~

———————————————————————————————————————

Los literales alfanuméricos y los identificadores pueden tener opcionalmente como prefijo la cláusula LENGTH OF. El valor de tiempo de compilación generado por esta cláusula será la cantidad de bytes del literal alfanumérico o el tamaño definido (en bytes) del identificador.

1. La palabra reservada OF es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa. A continuación se muestra un ejemplo. Las dos siguientes instrucciones GnuCOBOL muestran el mismo resultado (27):

```
01   Identificador de demostración FOTO X(27).
---
     LONGITUD DE VISUALIZACIÓN DE "Esta es una
     LONGITUD DE Ejemplo" LONGITUD DE VISUALIZACIÓN
     DE Demo-Identifier
```

2. La cláusula LENGTH OF en una referencia literal o de identificador generalmente se puede utilizar en cualquier lugar donde se pueda especificar un literal numérico, con las siguientes excepciones:
   - Como parte de la cláusula FROM de un WRITE (verSección 7.8.54 [ESCRIBIR], página 410) o Declaración de LIBERACIÓN (verSección 7.8.36 [LIBERACIÓN], página 370).
   - Como parte de la cláusula TIMES de una declaración PERFORM (verSección 7.8.33 [EJECUTAR], página 360).
~~~~~~

7.8 Sentencias GnuCOBOL

7.8.1 ACEPTAR

7.8.1.1 ACEPTAR DESDE LA CONSOLA

Sintaxis de ACEPTAR DESDE LA CONSOLA

```
ACEPTAR { identificador-1 }     [ DESDE nombre-mnemónico-1 ]
~~~~~~                            ~~~~
        { OMITIDO     }
          ~~~~~~~

[ FIN-ACEPTAR ]
  ~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para leer un valor de la ventana de la consola o del dispositivo de entrada estándar y almacenarlo en un elemento de datos (identificador-1).

1. Si no se especifica ninguna cláusula FROM, se supone FROM CONSOLE.
2. El nombre mnemónico-1 especificado debe ser uno de los nombres de dispositivo integrados CONSOLE, STDIN, SYSIN o SYSIPT, o un nombre definido por el usuario (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90) nombre mnemónico adjunto a uno de esos cuatro nombres de dispositivos.
3. La entrada se leerá desde la ventana de la consola (CONSOLA) o desde la entrada estándar del sistema (tubería 0 = STDIN, SYSIN o SYSIPT) y se guardará en el identificador-1.
4. Si el identificador-1 es un elemento de datos numéricos, el valor del carácter leído desde la consola o el dispositivo de entrada estándar se analizará de acuerdo con las reglas de entrada a la función intrínseca NUMVAL (verSección 8.1.70 [NUMVAL], página 500), excepto que no se respeta ninguno de los formatos de signos finales.

7.8.1.2 ACEPTAR DESDE LA LÍNEA DE COMANDOS

Sintaxis de ACEPTAR DESDE LA LÍNEA DE COMANDOS

```
ACEPTAR identificador-1
~~~~~~
          DE { LÍNEA DE COMANDOS                          }
          ~~~~ { ~~~~~~~~~~~~~                            }
               { NÚMERO DE ARGUMENTO                      }
               { ~~~~~~~~~~~~~~~~~                        }
               {  ARGUMENTO-VALOR                         }
               { ~~~~~~~~~~~~~~~~                         }
               [ ON EXCEPTION declaración imperativa 1 ]
                    ~~~~~~~~~
               [NO BAJO EXCEPCIÓN declaración imperativa 2 ]
                  ~~~      ~~~~~~~~~
[ FIN-ACEPTAR ]
  ~~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para recuperar información de la línea de comandos del programa.

1. La palabra reservada ON es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Cuando acepta desde la opción LÍNEA DE COMANDOS, recuperará todo el conjunto de argumentos ingresados en la línea de comandos que ejecutó el programa, exactamente como fueron especificados. Analizar los datos devueltos para obtener información significativa será su responsabilidad.
3. Al utilizar COMMAND-LINE o ARGUMENT-VALUE en una plataforma basada en *nix, que incluye Linux, OSX, BSD y bajo Windows que ejecuta msys o MinGW, etc., el proceso de shell expandirá cualquier argumento que tenga un '*' en la lista (como 'a*', 'abc*.*', etc.) y creará una lista de todos los archivos que coincidan con el patrón. Para evitar esto, si no lo desea, coloque todos los argumentos entre comillas, por ejemplo, progundertest "a*" bcd "ef*" "*hg" y el texto entre comillas se pasará textualmente al programa (como en el ejemplo progundertest).
4. Al aceptar ARGUMENT-NUMBER, le estará pidiendo al sistema de tiempo de ejecución de GnuCOBOL que analice los argumentos desde la línea de comandos y devuelva la cantidad de argumentos encontrados. El análisis se realizará de acuerdo con las siguientes reglas:
   A. Los argumentos se separarán utilizando espacios y/o caracteres de tabulación como delimitadores entre argumentos. La cantidad de estos delimitadores que separan dos valores de argumento que no están en blanco es irrelevante.
   B. Las cadenas encerradas entre comillas dobles ('"') se tratarán como un único argumento, independientemente de cuántos espacios o caracteres de tabulación (si los hay) puedan estar insertados dentro de las comillas.
   C. En los sistemas Windows, los caracteres de comillas simples o apóstrofo ('') se tratarán como cualquier otro carácter de datos y no delimitarán cadenas de argumentos.
5. Al aceptar desde ARGUMENT-VALUE, le estará pidiendo al sistema de tiempo de ejecución de GnuCOBOL que analice los argumentos desde la línea de comandos y devuelva el argumento "actual". Usted especifica qué número de argumento es "actual" a través de la opción ARGUMENT-NUMBER en DISPLAY

declaración (verSección 7.8.12.2 [VISUALIZACIÓN EN LA LÍNEA DE COMANDOS], página 303). El análisis de los argumentos se realizará de acuerdo con las reglas establecidas anteriormente.

6. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de recuperar un VALOR-ARGUMENTO. ConsulteSección 7.6.4 [CON EXCEPCIÓN + SIN EXCEPCIÓN], página 256, para obtener información adicional.

7.8.1.3 ACEPTAR DEL MEDIO AMBIENTE

ACEPTAR DEL ENTORNO Sintaxis

```
ACEPTAR identificador-1
~~~~~~
          DE { VALOR AMBIENTAL                    }
          ~~~~ { ~~~~~~~~~~~~~~~~~~               }
               { MEDIO AMBIENTE { literal-1   } }
                 { ~~~~~~~~~~~~ {
       identificador-1 } } [ ON EXCEPTION
       declaración-imperativa-1 ]
            ~~~~~~~~~
       [NO BAJO EXCEPCIÓN declaración imperativa 2 ]
         ~~~
                  ~~~~~~~~~
[ FIN-ACEPTAR ]
  ~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para recuperar valores de variables de entorno.

1. La palabra reservada ON es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Al aceptar desde ENVIRONMENT-VALUE, le pedirá al sistema de tiempo de ejecución de GnuCOBOL que recupere el valor de la variable de entorno cuyo nombre se encuentra actualmente en laRegistro NOMBRE DEL ENTORNO. Se puede colocar un valor en el registro NOMBRE DEL ENTORNO utilizando la opción NOMBRE DEL ENTORNO de la instrucción DISPLAY (consulteSección 7.8.12.3 [DIS- JUEGO CON EL MEDIO AMBIENTE-NOMBRE], página 304).
3. Un enfoque más simple para recuperar el valor de una variable de entorno es utilizar la opción ENVIRONMENT, donde se especifica la variable de entorno cuyo valor se recuperará directamente en la propia declaración ACCEPT.
4. Con la opción ACEPTAR "var" DEL ENTORNO runtime-config-option puedesrecibe el valor establecido en una de las opciones de configuración de tiempo de ejecución enumeradas en el capítulo 10.2.3 Variables de entorno de tiempo de ejecución. A modo de ejemplo: ACCEPT identifier-1 FROM ENVIRON- MENT 'COB INSERT MODE' devuelve si el modo de inserción está activo (= 'true' o 1) o inactivo (= 'false' o 0).
5. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante un intento de recuperar el valor de una variable de entorno inexistente o la recuperación exitosa del valor de una variable de entorno, respectivamente. ConsulteSección 7.6.4 [EN EXCEPCIONES- CIÓN + NO POR EXCEPCIÓN], página 256, para obtener información adicional.

7.8.1.4 ACEPTAR elemento de datos

Sintaxis del elemento de datos ACCEPT

```
ACEPTAR { identificador-1 } [{ DESDE ESTADO-DE-EXCEPCIÓN }] [ DESDE CRT] [ EL MODO ES BLOQUE ]
~~~~~~ { OMITIDO        }      ~~~~ ~~~~~~~~~~~~~~~~      ~~~~ ~~~      ~~~~      ~~~~~
         ~~~~~~~

        [ EN { LÍNEA            NUMERO { entero-1                    } ]
        [ ~~ { ~~~~                    { identificador-2            } ]
        [      {                       { expresión-aritmética-1 }
        ] [    { COLUMNA | NÚMERO DE POSICIÓN { entero-2             } ]
        [      { ~~~     ~~~           { identificador-3            } ]
        [      {                       { expresión-aritmética-2 } ] [
               { { entero-3     }                                   } ]
        [      { { identificador-4 }                                } ]

        [ CON [ AUTO | SALTO AUTOMÁTICO | TERMINACIÓN AUTOMÁTICA |
        TABULACIÓN ]
          ~~~~    ~~~~    ~~~~~~~~~    ~~~~~~~~~~~~~
                  ~~~[ [ NO ] { CAMPANA | PITIDO } ]
                     ~~       ~~~~    ~~~~
                [ PROMPT [ CARÁCTER ES literal-2 | identificador-5 ]
                  ~~~~~~   ~~~~~~~~~
                [ COLOR DE FONDO | COLOR DE FONDO ES entero-4 | identificador-6 ]
                  ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~
                [ COLOR-DE-PRIMERO|COLOR-DE-PRIMERO ES entero-5|identificador-7 ]
                  ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~ [
                RESALTAR | BAJAR LA LUZ ] [ PARPADEO ]
                  ~~~~~~~~~   ~~~~~~~~      ~~~~~
                [ VIDEO INVERSO | INVERSO | INVERTIDO ]
                  ~~~~~~~~~~~~~   ~~~~~~~   ~~~~~~~~
                [ LÍNEA IZQUIERDA ] [ LÍNEA SUPERIOR ] [ SUBRAYADO ]
                  ~~~~~~~~        ~~~~~~~~      ~~~~~~~~~
                [ REQUERIDO | VACÍO-MARCAR ] [ COMPLETO | LONGITUD-MARCAR ]
                  ~~~~~~~~    ~~~~~~~~~~~       ~~~~
                              ~~~~~~~~~~~~[SIN ECO | SIN ECO |
                APAGADO | SEGURO ]
                  ~~~~~~~     ~~~~~~~     ~~~
                            ~~~~~~[ INFERIOR |
                SUPERIOR ]
                  ~~~~~     ~~~~~
                [ DESPLAZARSE [ ARRIBA          ] [ { entero-6} LINEA|LINEAS ] ]
                [ ~~~~~~ [ ~~  ] [ { identificador-8 } ~~~~
                ~~~~~~ ] ] [       [ ABAJO]                                ]
                             ~~~~
                [ { TIEMPO DE ESPERA|TIEMPO DE ESPERA          ]
                    DESPUÉS } { entero-7                       }
                [ { ~~~~~~~ ~~~~~~~~          } { identificador-9 ]
                    }
                [ { ANTES DE TIEMPO           }                   ]
```

~~~~

[[NO] { ACTUALIZACIÓN | PREDETERMINADO }]]

[CONVERSIÓN

~~ ~~~~~~ ~~~~~~~ ~~~~~~~~~~

[CURSOR { identificador-10 }]
~~~~

```
                ~~~~~~
            [ EL TAMAÑO PROTEGIDO ES                { entero-8    } ]
            [              ~~~~      { identificador-11 } ]

            [ CONTROL      {literal-8        } ]
            [ ~~~~~~~~     { identificador-12 } ]

            [ { COLOR | COLOR } ES { entero-9   } ]
            [     ~~~~~~~   ~~~~~      { identificador-13 } ]

        [ EN CASO DE EXCEPCIÓN | ESCAPE declaración imperativa 1 ]
              ~~~~~~~~~~ ~~~~~~
        [ NO EN CASO DE EXCEPCIÓN | ESCAPE declaración imperativa 2 ]
          ~~~      ~~~~~~~~~~ ~~~~~~
[ FIN-ACEPTAR ]
  ~~~~~~~~~~~
```

Las cláusulas FROM CRT, MODE IS BLOCK y CONVERSION se reconocen sintácticamente pero por lo demás no son funcionales.

———

Este formato de la declaración ACCEPT se utiliza para recuperar datos de un elemento de almacenamiento en funcionamiento o de una pantalla de ventana de consola formateada.

1. Las palabras reservadas AFTER, IS, NUMBER y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas COLUMNA, COL y POSICIÓN son intercambiables.
3. Las palabras reservadas TIMEOUT y TIME-OUT son intercambiables.
4. Si el identificador-1 está definido en la SECCIÓN DE PANTALLA (verSección 6.7 [SECCIÓN DE PANTALLA], página 142), se ignorarán todas las cláusulas AT, Attribute-Specification, LOWER, UPPER o SCROLL. En estos casos, se utilizará una DISPLAY implícita (consulteSección 7.8.12.4 [MOSTRAR elemento de datos], página 305) del identificador-1 se producirá antes de que se acepte la entrada. Codificar un identificador-1 DISPLAY explícito antes de un identificador-1 ACCEPT es redundante y generará la penalización de rendimiento de pintar el contenido de la pantalla dos veces.
5. Las distintas cláusulas AT proporcionan un medio para posicionar el cursor en un punto específico de la pantalla antes de leerla. Se puede utilizar una u otra (pero no ambas), de la siguiente manera:
   A. Las cláusulas LINE y COLUMN proporcionan un mecanismo para especificar la posición de la línea y la columna en la que se colocará el cursor antes de permitir que el usuario ingrese datos. En ausencia de una u otra, se asumirá un valor de 1 para la que falta. Sin embargo, la preferencia personal del autor es codificar explícitamente ambas.
   B. El valor literal-3 o identificador-4, si se especifica, debe ser un valor de cuatro o seis dígitos, en el que la primera mitad del número indica la línea en la que debe colocarse el cursor y la segunda mitad indica la columna. Puede codificar solo una de cada cláusula en cualquier ACCEPT.
6. Las opciones WITH (incluidas las distintas especificaciones de atributos individuales) deben codificarse solo una vez.
7. Las siguientes cláusulas de especificación de atributos están permitidas en la declaración ACCEPT; son las mismas que las permitidas para los elementos de datos SCREEN SECTION. Una determinada especificación de atributos se puede utilizar solo una vez en cualquier declaración ACCEPT:

- AUTOMÁTICO (verSección 6.9.3 [AUTO], página 156), SALTO AUTOMÁTICO (verSección 6.9.4 [SALTO AUTOMÁTICO], Página 157), AUTOTERMINAR (esteSección 6.9.5 [TERMINACIÓN AUTOMÁTICA], página 158), PESTAÑA

- COLOR DE FONDO (verSección 6.9.6 [COLOR DE FONDO], página 159)
- BEEP (verSección 6.9.9 [BEEP], página 162), CAMPANA (verSección 6.9.10 [BELL], página 163)
- ANTES DE TIEMPO (verSección 6.9.7 [ANTES DEL TIEMPO], página 160)
- PARPADEO (verSección 6.9.13 [BLINK], página 166)
- COLOR DE PRIMER PLANO (verSección 6.9.21 [COLOR DEL PRIMER PLANO], página 175)
- COMPLETO (verSección 6.9.23 [COMPLETA], página 177), CONTROL DE LONGITUD (verSección 6.9.29 [VERIFICACIÓN DE LONGITUD], página 184)
- DESTACADO (verSección 6.9.26 [HIGHLIGHT], página 180)
- LÍNEA IZQUIERDA (verSección 6.9.28 [LEFTLINE], página 183)
- INFERIOR (verSección 6.9.31 [INFERIOR], página 187)
- LUZ BAJA (verSección 6.9.32 [LOWLIGHT], página 188)
- SOBRELINEA (verSección 6.9.36 [OVERLINE], página 194)
- INSTRUCCIÓN (verSección 6.9.39 [PROMPT], página 203)
- PROTEGIDO (verSección 6.9.40 [PROTEGIDO], página 204)
- REQUERIDO (verSección 6.9.43 [REQUERIDO], página 207), CHEQUE VACÍO (verSección 6.9.17 [CHEQUE VACÍO], página 171)
- VIDEO INVERSO (verSección 6.9.44 [VÍDEO INVERSO], página 208)
- TAMAÑO (verSección 6.9.48 [TAMAÑO], página 213)
- SEGURO (verSección 6.9.46 [SEGURO], página 211), NO-ECO (verSección 6.9.34 [NO-ECHO], página 190)
- TIEMPO FUERA (verSección 6.9.53 [TIEMPO DE ESPERA], página 219)
- ACTUALIZACIÓN (Ver abajo)
- SUPERIOR (verSección 6.9.58 [SUPERIOR], página 227)
- SUBRAYAR (verSección 6.9.57 [SUBRAYADO], página 226)

8. El desplazamiento del CURSOR en el campo al finalizar ACCEPT se devuelve al programa en el identificador 10. El identificador 10 también se puede utilizar para indicar, como desplazamiento, dónde desea que se posicione el cursor cuando se ejecuta ACCEPT.
9. La opción DESPLAZAR hará que todo el contenido de la pantalla se desplace hacia ARRIBA oDOWN por el número especificado de líneas antes de que se muestre cualquier valor en la pantalla. Es sintácticamente admisible especificar una cláusula SCROLL UP así como una cláusula SCROLL DOWN. En tal caso, se respetará la última especificada. Si no se especifica LINES, se asumirá 1 LINE. Estas dos opciones no se pueden utilizar en la sección de pantalla.
10. La opción TIMEOUT hará que ACCEPT no espere más que la cantidad de segundos especificada para recibir una entrada. El tiempo de espera puede especificarse como un entero positivo o un elemento de datos numérico con un valor positivo.
11. La opción UPDATE mostrará los datos existentes antes de permitir que el usuario los actualice. Esto permite que el usuario actualice el campo sin tener que escribirlo todo de nuevo. La cláusula DEFAULT es sinónimo de la cláusula UPDATE. Sin esta cláusula, el campo ACCEPT siempre estará en blanco (se asume que NO UPDATE o NO DEFAULT).

    Esta cláusula permitirá que el campo de datos suministrado se actualice después de haber sido mostrado en la pantalla antes de que se ingresen los datos, sobrescribiéndolo, si es necesario. Cuando no se utiliza esta opción, el campo de entrada se borra antes de la entrada y este es el valor predeterminado, pero se puede cambiar mediante el uso del comando de control del compilador -faccept-with-update que se puede ingresar al iniciar el compilador o incluir en el archivo de configuración, por ejemplo, default.conf se usa cuando se selecciona por defecto -std=default. Para obtener más información, consulteSección

10.1.1 [cobc - El GnuCOBOL

[Compilador], página 673, interruptores de opción) ySección 10.1.7 [Archivos de configuración del compilador], página 691.

12. Este formato de la declaración ACCEPT finalizará por cualquiera de los siguientes eventos:
    A. Cuando se presiona la tecla Enter.
    B. Vencimiento del temporizador TIMEOUT: esto se tratará como si se hubiera presionado la tecla Enter sin ingresar ningún dato.
    C. Cuando se presiona una tecla de función (Fn).
    D. Al presionar las teclas PgUp o PgDn, si la variable de entorno de ejecución COB_SCREEN_EXCEPTIONS (verSección 10.2.3 [Variables del entorno de tiempo de ejecución], página 700) se establece en cualquier valor que no esté en blanco.
    E. Al presionar la tecla Esc si tanto la variable de entorno de ejecución COB_SCREEN_ESC como la variable de entorno de ejecución COB_SCREEN_EXCEPTIONS están configuradas en cualquier valor que no esté en blanco.
    F. Pulsar las teclas de flecha arriba, flecha abajo o Impr Pant (Imprimir pantalla). Sin embargo, estas teclas no se pueden detectar en los sistemas Windows.
13. Lo siguiente se aplica cuando el identificador-1 se define en la SECCIÓN DE PANTALLA:
    A. Los datos alfanuméricos ingresados en el identificador-1 o en cualquier elemento de datos de pantalla subordinado a él deben ser consistentes con la IMAGEN (verSección 6.9.37 [IMAGEN], página 195) de ese elemento. Esto se aplicará en tiempo de ejecución mediante la declaración ACCEPT.
    B. Si el identificador-1 o cualquier elemento de datos de pantalla subordinado a él se define como numérico, los datos ingresados deben ser aceptables como función intrínseca NUMVAL (verSección 8.1.70 [NUMVAL], página 500) entrada (no se permiten puntos decimales, sin embargo). El valor almacenado en el elemento de datos de la pantalla será como si la entrada se hubiera pasado a esa función.
    C. Si el identificador-1 o cualquier elemento de datos de pantalla subordinado a él se define como editado numéricamente, los datos ingresados deben ser aceptables como función intrínseca de NUMVAL-C (verSección 8.1.71 [NUMVAL-C], página 501) entrada (nuevamente, no se permiten puntos decimales). El valor almacenado en el elemento de datos de la pantalla será como si la entrada se hubiera pasado a esa función.
14. Lo siguiente se aplica cuando el identificador-1 no está definido en la SECCIÓN DE PANTALLA:
    A. Los datos alfanuméricos ingresados en el identificador-1 deben ser consistentes con la IMAGEN (verSección 6.9.37 [IMAGEN], página 195) de ese elemento, aunque la declaración ACCEPT no lo impondrá. Puede utilizar las condiciones de clase (consulteSección 2.1.27.2 [Condiciones de la Clase], página 45) después de que se aceptan los datos para imponer el tipo de datos.
    B. Si el identificador-1 se define como numérico, los datos ingresados deben ser aceptables como función intrínseca NUMVAL (verSección 8.1.70 [NUMVAL], página 500) entrada (no se permiten decimales, sin embargo). El valor almacenado en identifier-1 será como si la entrada se hubiera pasado a esa función.
    C. Si el identificador-1 se define como numérico editado, los datos ingresados deben ser aceptables como función intrínseca de NUMVAL-C (verSección 8.1.71 [NUMVAL-C], página 501) entrada (nuevamente, no se permiten puntos decimales). El valor almacenado en el identificador-1 será como si la entrada se hubiera pasado a esa función.
15. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, del intento de entrada/salida de pantalla. ConsulteSección 7.6.4 [SOBRE EX- [ACEPCIÓN + NO POR EXCEPCIÓN], página 256, para obtener información adicional. Se permite la subcláusula ESCAPE en lugar de

EXCEPTION como sinónimo, con el mismo significado.

Después de ejecutar este formato de la declaración ACCEPT, el CRT STATUS del programa (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) El identificador se completará con uno de los siguientes:

Código	Significado
0000	INGRESAR tecla presionada
1001–1064	F1–F64, respectivamente,
fueronprensado 2001	Se presionó PgUp
2002	AvPág. fue presionado
2003	Flecha hacia arriba fue presionado
2004	Flecha hacia abajo fue presionado
2005	Se presionó Esc
2006	Se presionó PrtSc (Imprimir pantalla)
2007	Pestaña
2008	Pestaña Atrás
2009	Llave izquierda
2010	Clave derecha
2011	Se omite la inserción de clave al aceptar
2012	Se omite la tecla Eliminar al aceptar
2013	Se omite la tecla de retroceso al aceptar
2014	Hogar Se omitió la tecla de aceptar
2015	Se omite la tecla Fin al aceptar
2040-	2095 Teclas de excepción para el
ratónManejo 2040	Movimiento del ratón
2041	Presionado a la izquierda
2042	Izquierda liberada
2043	Doble clic izquierdo
2044	Media prensada
2045	MedioL i b e r a d o
2046	Doble clic medio
2047	Derecha presionada
2048	Derecho liberado
2049	Doble clic derecho
2050	CambioM o v e r
2051	Shift izquierda presionado
2052	Shift a la izquierda liberado
2053	Doble clic con cambio a la izquierda
2054	Shift presionado a mitad de camino
2055	Cambio a mitad de camino liberado
2056-	Cambio de medio clic doble
2057	Shift presionado a la derecha
2058	Shift derecho liberado
2059	Doble clic con cambio a la derecha
2060	Ctrl Mover
2061	Ctrl presionado a la izquierda
2062	Ctrl izquierda liberada
2063	Ctrl + doble clic izquierdo
2064	Ctrl presionado a mitad de camino
2065	Ctrl Mid liberado
2066	Ctrl medio doble clic
2067	Ctrl derecho presionado
2068	Ctrl Derecha Liberado
2069	Ctrl + doble clic derecho
2070	Todo se mueve

2071	Alt izquierda presionada
2072	La izquierda alternativa se liberó
2073-	Alt doble clic izquierdo
2074	Alt Mid presionado
2075	Alt Mid liberado
2076	Alt Mid Doble Clic
2077	Alt derecha presionada
2078	La derecha alternativa se libera
2079	Alt derecho doble clic
2080	Rueda arriba
2081	Rueda abajo
2082	Rueda izquierda
2083	Rueda derecha
2084	Rueda de cambio hacia arriba
2085	Rueda de cambio hacia abajo
2086	Rueda de cambio hacia la izquierda
2087	Rueda de cambio a la derecha
2088	Ctrl rueda arriba
2089	Ctrl rueda hacia abajo
2090	Ctrl rueda izquierda
2091	Ctrl rueda derecha
2092	Rueda alternativa hacia arriba
2093	Rueda alternativa hacia abajo
2094	Rueda Alt izquierda
2095	Rueda Alt derecha
	Validación de entrada
8000	NO Campo
8001	Se acabó el tiempo
	Otros errores
9000	Fatal
9001	Campo máximo

16. La tecla real presionada para generar una tecla de función (Fn) dependerá del tipo de dispositivo terminal que esté usando (PC, Macintosh, VT100, etc.) y del tipo de controlador de pantalla mejorado que se configuró con la versión de GnuCOBOL que esté usando.

Por ejemplo, en una compilación de GnuCOBOL para una PC con Windows que usa

MinGW y "PDCurses", F1–F12 son las teclas F reales en el teclado de la PC, F13–F24 se ingresan cambiando la tecla FLas teclas F25–F36 se introducen manteniendo pulsada la tecla Ctrl mientras se pulsa una tecla F, y las teclas F37–F48 se introducen manteniendo pulsada la tecla Alt mientras se pulsa una tecla F. Por otro lado, una implementación de GnuCOBOL creada para Windows con Cygwin y NCurses trata las teclas F1–F12 de la PC como las teclas F1–F12 reales, mientras que las teclas F con mayúsculas introducirán las teclas F11–F20. Con Cygwin/NCurses, las teclas F modificadas con Ctrl y Alt no se reconocen, ni tampoco las teclas F Shift-F11 o Shift-F12.

Los códigos de teclas del mouse se completan solo si se ha habilitado la administración del mouse. Para habilitarratón, primero es necesario configurar COB_MOUSE_FLAGS (ya sea externamente a través del comando de terminal o internamente a través de SET ENVIRONMENT a la "máscara de ratón" aplicable (especificando qué actividades desea que el programa detecte). Aquí hay un ejemplo de configuración de la máscara desde un programa COBOL:

```
COPIA screenio.cpy.

01   Banderas de ratón PIC 9(4).

---

     COMPUTAR indicadores del mouse = MANEJO AUTOMÁTICO DEL MOUSE DE COB
                                    + COB-PERMITIR-IZQUIERDA-ABAJO
                                    + COB-PERMITIR-MEDIO-ABAJO
                                    + COB-PERMITIR-DERECHO-ABAJO

     ESTABLECER EL ENTORNO "COB_MOUSE_FLAGS" EN mouse-flags.
```

17. Una vez hecho esto, cada ACCEPT (extendido) devolverá un valor en COB CRT STATUS que refleja la actividad del mouse, cuando dicha actividad ocurre. Los valores aplicables se muestran en screenio.cpy bajo ?Teclas de excepción para el manejo del mouse?. Si define una variable en SPECIAL NAMES de la siguiente manera:

    ```
    NOMBRES ESPECIALES.
        CURSOR ES nombre-datos. *> donde nombre-datos es PIC 9(4) o 9(6).
    ```
18. También se devolverá la posición del cursor o del ratón. La posición se expresa como fila y columna (rrcc o rrrccc).
19. Las teclas del teclado numérico no son reconocibles en las compilaciones MinGW/PDCurses de Windows de Gnu-COBOL, independientemente de la configuración de bloqueo numérico. Las compilaciones Cygwin/NCurses de Windows reconocen las entradas del teclado numérico correctamente. Aunque no se ha probado durante la preparación de esta documentación, esperaría que las compilaciones nativas de Windows que utilizan PDCurses se comporten como las compilaciones de MinGW y que las compilaciones nativas de Unix que utilizan NCurses se comporten como las compilaciones de Cygwin.
20. La cláusula opcional EXCEPTION-STATUS se puede utilizar para detectar excepciones de un verbo aritmético anterior, como COMPUTE, para recuperar los errores producidos. Estos se recuperan utilizando la función EXCEPTION-STATUS.
21. El uso de CONTROL El valor de literal-8 e identificador-8 en la frase CONTROL se utiliza para especificar una lista de opciones dinámicas. El valor debe ser una cadena de caracteres que consiste en una serie de palabras clave delimitadas por comas; Algunas palabras clave permiten la asignación de un valor siguiendo la palabra clave con un signo igual y el valor. Los espacios en blanco se ignoran en la cadena de caracteres. Las letras minúsculas se tratan como letras mayúsculas dentro de las palabras clave. Las palabras clave especificadas anulan las opciones estáticas correspondientes especificadas como frases para el mismo identificador-1. Las palabras clave se pueden especificar en cualquier orden. Las palabras clave, que especifican opciones que no se aplican a la declaración, se ignoran.

    Las palabras clave que afectan a una declaración ACCEPT son BEEP, BLINK, ECHO, PROMPT, REVERSE, TAB, UNDERLINE, UPDATE, ERASE, ERASE EOL, ERASE EOS, HIGH, LOW, UPPER, NO BEEP, NO BLINK, NO ECHO, NO PROMPT, NO REVERSE, NO TAB, NO UNDERLINE, NO UPDATE, NO ERASE.

    Los significados de estas palabras clave cuando aparecen en el valor del operando de frase CONTROL son los mismos que las frases correspondientes que pueden escribirse como opciones estáticas de la declaración ACCEPT, con la adición de las formas negativas para permitir la supresión de las opciones declaradas estáticamente.

    Las palabras claves UNDERLINE y UPPER no están disponibles como opciones estáticas de la instrucción ACCEPT. Cuando se especifica, UPPER hace que todos los caracteres alfabéticos

en minúscula contenidos en el campo de pantalla se cambien a caracteres alfabéticos en mayúscula antes de la conversión de los datos de entrada y su almacenamiento en el campo receptor.

Cuando se especifica, UNDERLINE hace que el campo en la pantalla se muestre en modo subrayado, siempre que el terminal admita ese modo.

GnuCOBOL proporciona dos palabras clave adicionales en la frase CONTROL que afectan a unaCampo ACEPTAR.

1. FCOLOR = nombre del color

Cuando FCOLOR está presente, el nombre del color especifica el color de primer plano del campo ACCEPT. Este nombre se utiliza como valor predeterminado para las instrucciones ACCEPT posteriores en el programa. El valor predeterminado inicial para el nombre del color es el blanco.

2. BCOLOR = nombre del color

Cuando BCOLOR está presente, color-name especifica el color de fondo del campo DISPLAY. Este valor se utiliza como valor predeterminado para las instrucciones ACCEPT posteriores en el programa. El valor predeterminado inicial para color-name es el negro.

La siguiente tabla contiene una lista de todos los nombres posibles para el nombre del color. La columna de la izquierda contiene el nombre válido del color. La columna de la derecha muestra el color que aparece cuando se especifica una intensidad alta (la intensidad predeterminada).

Nombres de colores COBOL válidos		
Nombres de colores válidos		Valores de color de alta intensidad (valores predeterminados)
Negro	Gris	
Azul	Luz Azul	
Verde	Luz Verde	
Cian	Luz Cian	
Rojo	Luz Rojo	
Magenta	Luz Magenta	
Marrón	Amarillo	
Blanco	Alta intensidad	Blanco

22. Frase de COLOR

La frase COLOR proporciona un método alternativo para configurar los atributos de video. Integer-9 debe ser un literal numérico. Identifier-9 debe ser un elemento de datos numérico. También permite la especificación de colores para los campos y controles de la pantalla. Se pueden configurar con diferentes valores numéricos para expresar varias combinaciones de colores y atributos de video.

Puedes hacer combinaciones sumando los valores adecuados. Se aceptan los siguientes valores de color:

Color	Primer plano	Antecedentes
Negro	1	32
Azul	2	64
Verde	3	96
Cian	4	128
Rojo	5	160
Magenta	6	192
Marrón	7	224
Blanco	8	256

23. Puede especificar otros atributos de vídeo agregando los siguientes valores:

Video inverso	1024
Baja intensidad	2048
Alta intensidad	4096
Subrayar	8192
Parpadeo	16384
Protegido	32768
Fondo de baja intensidad	65536
Fondo de alta intensidad	131072

```
+-------------------------+ ----------- +
```

24. También puede especificar una intensidad alta agregando "8" al valor del color de primer plano.

7.8.1.5 ACEPTAR DESDE FECHA/HORA

ACEPTAR DESDE FECHA/HORA Sintaxis

```
ACEPTAR identificador-1 DESDE { FECHA [ AAAAMMDD ] }
  ~~~~~~                  ~~~~ { ~~~~     ~~~~~~~~   }
                               { DÍA [ AAAADDDD ]     }
                               { ~~~     ~~~~~~~      }
                               { DÍA DE LA SEMANA     }
                               { ~~~~~~~~~~~~~        }
                               { TIEMPO               }
                               { ~~~~                 }
                          { TIEMPO DE MICROSEGUNDO    }
                          { ~~~~~~~~~~~~~~~~~~~~      }
[ FIN-ACEPTAR ]
  ~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para recuperar la fecha y hora actual del sistema o el día actual de la semana y almacenarlos en un elemento de datos.

1. Los datos recuperados del sistema y el formato en que están estructurados variarán de la siguiente manera:

Sintaxis	**Datos recuperados**	**Formato**
FECHA	Fecha actual en formato gregoriano ymmddFECHA AAAAMMDD	Fecha
actual en formato gregoriano	aaaammdd DÍA	Fecha
actual en formato juliano	Sí	
DÍA AAAAAAAA	Actual fecha en forma juliana	aaaaadddd
DÍA DE LA SEMANA	Día actual dentro de una semana donde 1 = Lun- día.., 7 = domingo	d
TIEMPO	Tiempo, incluidas las centésimas de segundo (n)	hmmssnn
TIEMPO DE MICROSEGUNDO	Tiempo, incluyendo microsegundos (u)	hmmssuuuuuu

2. Al utilizar uno de los siguientes: --std=acu o --std=acu-strict, el elemento de datos ACCEPT FROM TIME con un elemento de datos que proporciona más de 12 dígitos se comporta como si se utilizara el elemento de datos ACCEPT FROM MICROSECOND-TIME (seis lugares decimales para fracciones de segundo).
3. Considere utilizar la función estándar FORMATTED-CURRENT-DATE si necesita alta precisión (hasta ocho decimales para fracciones de segundo).

7.8.1.6 ACEPTAR DESDE Screen-Info

ACEPTAR DESDE Sintaxis de información de pantalla

```
ACEPTAR identificador-1
~~~~~~
      DESDE { LÍNEAS|NÚMERO-DE-LÍNEA }
            ~~~~ { ~~~~~ ~~~~~~~~~~~ }
                 { COLUMNAS          }
                 { ~~~~ ~~~~~~~~     }
                 { TECLA ESCAPE      }
                   ~~~~~~ ~~~
[ FIN-ACEPTAR ]
  ~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para recuperar información sobre la ventana de la consola o sobre las interacciones del usuario con ella.

1. Las palabras reservadas LINES y LINE-NUMBER son intercambiables.
2. Las palabras reservadas COLS y COLUMNS son intercambiables.
3. Los siguientes puntos pertenecen al uso de las opciones LÍNEAS y COLUMNAS:
   A. Las opciones LÍNEAS y COLUMNAS recuperarán los componentes respectivos del tamaño de la pantalla de la consola.
   B. Cuando la consola se ejecuta en un entorno de ventanas, este será el tamaño de la ventana en la que se ejecuta el programa, en términos de recuentos de caracteres horizontales (COLUMNAS) o verticales (LÍNEAS), no de píxeles.
   C. Cuando el sistema no ejecuta un entorno de ventanas, se devolverán los atributos de pantalla de la consola física.
   D. Se devolverán valores de 0 si GnuCOBOL no se generó para incluir E/S de pantalla.
   E. Consulte la documentación sobre la subrutina de sistema incorporada CBL_GET_SCR_SIZE (consulteSección 8.2.41 [CBL GET SCR SIZE], página 596) para encontrar otra forma de recuperar esta información.
4. La opción ESCAPE KEY se puede utilizar después de que se haya utilizado un elemento de datos ACCEPT para recuperar datos de una pantalla formateada. El resultado devuelto será el identificador de clave de cuatro dígitos de la tecla especial que se presionó para finalizar el elemento de datos ACCEPT (se devuelve 0000 para la tecla Enter).

   Si especifica que el ESTADO CRT es la cláusula Identificador-1 en NOMBRES ESPECIALES, entonces, para un elemento de datos ACEPTAR, el sistema establece un código en el campo Identificador-1 que indica qué tecla finalizó el elemento de datos ACEPTAR.

   Si la cláusula CRT STATUS IS identifier-1 no se especifica en SPECIAL NAMES, entonces dentro del programa, puede escribir dos ACCEPT consecutivos:

   ```
   ACEPTAR elemento de datos
   ACEPTAR identificador-1 DE LA TECLA ESCAPE
   ```

   De esta manera en el campo identificador-1 tendrás el código de retorno que indica qué tecla completó el dato ACCEPT.

7.8.1.7 ACEPTAR DESDE Runtime-Info

ACEPTAR DESDE Sintaxis de información de tiempo de ejecución

```
ACEPTAR identificador-1
~~~~~~
        DESDE { ESTADO DE EXCEPCIÓN }
           ~~~~ { ~~~~~~~~~~ ~~~~~~ }
                  {NOMBRE DE USUARIO  }
                    ~~~~ ~~~~
[ FIN-ACEPTAR ]
  ~~~~~~~~~~
```

Este formato de la declaración ACCEPT se utiliza para recuperar información en tiempo de ejecución, como el código de excepción de error más reciente y el nombre de usuario del usuario actual.

1. Los siguientes puntos se refieren al uso de la opción ESTADO DE EXCEPCIÓN:
   A. *identificador-1*debe definirse como un elemento PIC X(4).
   B. Ver[Códigos de excepción de error], página 455, para obtener una lista completa de los códigos de excepción y sus significados.
   C. Una alternativa al uso de ACCEPT FROM Runtime-Info es utilizar la función intrínseca EXCEPTION-STATUS (verSección 8.1.26 [ESTADO DE EXCEPCIÓN], página 455).
2. Los siguientes puntos pertenecen al uso de la opción NOMBRE DE USUARIO:
   A. El resultado devuelto es el ID de usuario que se utilizó para iniciar sesión en el sistema, y no el nombre y/o apellido real del usuario en cuestión (a menos que, por supuesto, esa sea la información utilizada como ID de inicio de sesión). No son los números PID o UID sino el nombre asociado con el UID en sistemas basados en *nix.
   B. *identificador-1*Debe definirse lo suficientemente grande para recibir el nombre de usuario más largo del sistema.
   C. Si no se asigna suficiente espacio, el valor devuelto se truncará.
   D. Si se asigna espacio excedente, el valor devuelto se rellenará con espacios (a la derecha).

7.8.1.8 ACEPTAR OMITIDO

ACEPTAR OMITIDO Sintaxis

ACEPTAR OMITIDO
~~~~~~

1. Para consola: Véase 6.17.1.1 (ACEPTAR)DESDE CONSOLA Sintaxis)

2. Para pantalla : Consulte 6.17.1.4 (Sintaxis del elemento de datos

de pantalla ACCEPT) [END-ACCEPT]
~~~~~~~~~~

———————————————————————————————————————

Este formato de la instrucción ACCEPT esperará un evento de teclado que finalice la entrada; teclas de función o Enter/Return, entre otras. ESTADO CRT (COB-CRT-STATUS ESTADO CRT (verSección 5.1.3 [NOMBRES ESPECIALES], página 90) si no se define explícitamente) se establece con el código clave,se encuentra en copy/screenio.cpy. También maneja algunas otras terminaciones de código de tecla que normalmente no se utilizan para completar una aceptación extendida.

1. Los siguientes son ejemplos de códigos clave que se pueden utilizar:
   COB-SCR-INSERTAR
   COB-SCR-
   BORRARCOB-SCR-
   RETROCESO COB-SCR-
   TECLA-INICIO COB-
   SCR-TECLA-FIN
2. Puede utilizar atributos extendidos, útiles para establecer tiempos de espera o posicionamiento.

7.8.1.9 ACEPTAR DESDE EL ESTADO DE EXCEPCIÓN

Sintaxis ACEPTAR DESDE ESTADO DE EXCEPCIÓN

```
ACEPTAR estado de excepción-foto-9-4      DE ESTATUS DE EXCEPCIÓN
~~~~~~                                    ~~~~ ~~~~~~~~~~ ~~~~~~

[ FIN-ACEPTAR ]
~~~~~~~~~~
```

Este formato de la declaración ACCEPT recibirá el estado de cualquier excepción que resulte de un verbo válido anterior.

1. El siguiente es un ejemplo de uso:

```
En WS:
01   estado de excepción  foto 9(4).
- -
En PD:

ACEPTARredondeo inesperadoDESDE    ESTADO DE
EXCEPCIÓN SI el redondeo inesperado NO ES IGUAL A "0000"
ENTONCES
     MOSTRAR "Redondeo inesperado. Código " redondeo-inesperado AL
                 SYSERR
FIN-SI
```

7.8.2 AGREGAR

7.8.2.1 AÑADIR

Sintaxis ADD TO

```
AGREGAR { literal-1    }...
~~~ { identificador-1 }

    PARA { identificador-2
    ~~
        [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] ] }...
          ~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~          }
                               { MÁS CERCANO-LEJOS-DE-CERO }
                                 { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                               { MÁS CERCANO-PAR           }
                               { ~~~~~~~~~~~~~~            }
                               { MÁS CERCANO A CERO        }
                        { ~~~~~~~~~~~~~~~~~~~~~~~          }
                               { PROHIBIDO                 }
                               { ~~~~~~~~~~~               }
                               { HACIA MAYOR               }
                               { ~~~~~~~~~~~~~~~~          }
                               { HACIA MENOR               }
                               { ~~~~~~~~~~~~~~~           }
                               { TRUNCAMIENTO              }
                                 ~~~~~~~~~~
  [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
        ~~~~ ~~~~~
  [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
    ~~~      ~~~~
~~~~~[ FIN-AGREGAR
]
  ~~~~~~~
```

Este formato de la instrucción ADD genera una suma aritmética intermedia de los valores de todos los elementos del identificador 1 y del literal 1. El valor de cada identificador 2 será reemplazado, a su vez, por la suma del valor de ese identificador 2 y la suma intermedia.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Tanto el identificador-1 como el identificador-2 deben ser elementos de datos numéricos sin editar, mientras que el literal-1 debe ser un literal numérico.
3. Un elemento de datos de identificador 1 también puede codificarse como identificador 2. Sin embargo, tenga en cuenta que el valor de dicho elemento de datos se incluirá dos veces en el resultado.
4. El contenido de cada identificador-1 permanecerá sin cambios con esta declaración.
5. El REDONDEADO opcional (verSección 7.6.7 [REDONDEADA], página 257) cláusula disponible para cada uno
   *identificador-2*controlará cómo se guardarán los resultados no enteros.

6. Las cláusulas opcionales ON SIZE ERROR y NOT ON SIZE ERROR se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de realizar un cálculo.

En este caso, la falla se define como un identificador-2 con un número insuficiente de posiciones de dígitos disponibles a la izquierda de cualquier punto decimal implícito. VerSección 7.6.6 [SOBRE ERROR DE TAMAÑO + [NO SE ENCUENTRA EN ERROR DE TAMAÑO], página 257, para obtener información adicional.

7.8.2.2 AGREGAR DONACIONES

Sintaxis de ADD GIVING

```
AGREGAR { literal-1    }...
~~~ { identificador-1 }

  [ Identificador AL-2 ]
    ~~
    DONAR { identificador-3
    ~~~~~~
      [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] ] }...
        ~~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~          }
                              { MÁS CERCANO-LEJOS-DE-CERO }
                                { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { MÁS CERCANO-PAR           }
                              { ~~~~~~~~~~~~~~            }
                              { MÁS CERCANO A CERO        }
                        { ~~~~~~~~~~~~~~~~~~~~~~~         }
                              { PROHIBIDO                 }
                              { ~~~~~~~~~~~               }
                              { HACIA MAYOR               }
                              { ~~~~~~~~~~~~~~~~          }
                              { HACIA MENOR               }
                              { ~~~~~~~~~~~~~~            }
                              { TRUNCAMIENTO              }
                                ~~~~~~~~~~
  [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
        ~~~~ ~~~~~
  [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
    ~~~      ~~~~
~~~~~[ FIN-AGREGAR
]
  ~~~~~~~
```

Este formato de la declaración ADD genera la suma aritmética de los valores de todos los elementos identificador-1, literal-1) e identificador-2 (si hay alguno) y luego guarda esa suma en cada identificador-3.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Tanto el identificador-1 como el identificador-2 deben ser elementos de datos numéricos sin editar, mientras que el literal-1 debe ser un literal numérico; el identificador-3 puede ser un elemento de datos numérico o numérico editado.
3. Si se desea, un elemento de datos de identificador-1 o identificador-2 se puede utilizar como identificador-3.
4. El contenido de cada identificador-1 e identificador-2 permanecerá inalterado por esta declaración, a menos que también se especifiquen como un identificador-3.
5. El valor actual de cada identificador-3 al inicio de la ejecución de la declaración es irrelevante, ya que el contenido de cada identificador-3 simplemente será reemplazado por

la suma calculada.

6. El REDONDEADO opcional (verSección 7.6.7 [REDONDEADA], página 257) cláusula disponible para cada uno
*identificador-3*controlará cómo se guardarán los resultados no enteros.

7. Las cláusulas opcionales ON SIZE ERROR y NOT ON SIZE ERROR se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de realizar un cálculo.En este caso, la falla se define como un identificador-3 con un número insuficiente de posiciones de dígitos disponibles a la izquierda de cualquier punto decimal implícito. VerSección 7.6.6 [SOBRE ERROR DE TAMAÑO + [NO SE ENCUENTRA EN ERROR DE TAMAÑO], página 257, para obtener información adicional.

7.8.2.3 AGREGAR CORRESPONDIENTE

Sintaxis AGREGAR CORRESPONDIENTE

```
AGREGAR IDENTIFICADOR CORRESPONDIENTE-1
~~~
     Identificador TO-2
     ~~
   [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] ]
     ~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~            }
                          { MÁS CERCANO-LEJOS-DE-CERO }
                            { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                          { MÁS CERCANO-PAR             }
                          { ~~~~~~~~~~~~~~              }
                          { MÁS CERCANO A CERO          }
                       { ~~~~~~~~~~~~~~~~~~~~~~~~        }
                          { PROHIBIDO                   }
                          { ~~~~~~~~~~~                 }
                          { HACIA MAYOR                 }
                          { ~~~~~~~~~~~~~~~             }
                          { HACIA MENOR                 }
                          { ~~~~~~~~~~~~~~              }
                          { TRUNCAMIENTO                }
                            ~~~~~~~~~~~
   [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
         ~~~~ ~~~~~
   [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
     ~~~      ~~~~
~~~~~[ FIN-AGREGAR
]
  ~~~~~~~
```

———————————————————————————————————————

Este formato de la declaración ADD genera un código equivalente a ADD TO individual (verSección 7.8.2.1 [AGREGAR A], página 282) declaraciones para las coincidencias correspondientes de elementos de datos encontrados subordinados a los dos identificadores.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Tanto el identificador-1 como el identificador-2 deben ser elementos de grupo.
3. VerSección 7.6.2 [CORRESPONDIENTE], página 254, para obtener información sobre cómo se encontrarán las coincidencias correspondientes entre el identificador-1 y el identificador-2.
4. El REDONDEADO opcional (verSección 7.6.7 [REDONDEADA], página 257) cláusula disponible para cada uno
   *identificador-3*controlará cómo se guardarán los resultados no enteros.
5. Las cláusulas opcionales ON SIZE ERROR y NOT ON SIZE ERROR se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de realizar un cálculo.En este caso, la falla se define como un identificador-3 con un número insuficiente de posiciones de dígitos disponibles a la izquierda de cualquier punto decimal implícito. VerSección 7.6.6 [SOBRE ERROR DE TAMAÑO + [NO SE ENCUENTRA EN ERROR

DE TAMAÑO], página 257, para obtener información adicional.

7.8.3 ASIGNAR

Sintaxis ALLOCATE

FORMATO 1. ASIGNAR un ELEMENTO "BASE".

```
ASIGNAR   identificador-1
~~~~~~~~
          [{ INICIALIZADO } ]     [ DEVOLVIENDO identificador-3 ]
          [{ ~~~~~~~~~~~~~ } ]
                                  ~~~~~~~~~
          [{ INICIALIZADO } ]
          [{ ~~~~~~~~~~~~~ } ]
```

FORMATO 2. ASIGNAR un bloque de memoria.

```
ASIGNAR     expresión aritmética 1        PERSONAJES
~~~~~~~~                                  ~~~~~~~~~~~
          [{ INICIALIZADO } [ A { identificador-2}] ]      DEVOLVIENDO
          identificador-3 [{ ~~~~~~~~~~~~ } [ ~~ {    }] ]    ~~~~~~~~~~
          [{ INICIALIZADO } [     { literal-1     }] ]
          [{ ~~~~~~~~~~~~ }                            ]
```

———

La instrucción ALLOCATE se utiliza para asignar memoria dinámicamente en tiempo de ejecución.

1. Las palabras reservadas INITIALIZED e INITIALISED son intercambiables.
2. Si se especifica el identificador 1, se puede omitir la frase RETURNING; de lo contrario, se deberá especificar la frase RETURNING.
3. Si se utiliza, arithmetic-expression-1 debe ser una expresión aritmética con un valor entero positivo distinto de cero y se debe especificar la frase RETURNING.
4. Si se utiliza, el identificador 1 debe ser un elemento de nivel 01 definido en el almacenamiento de trabajo o el almacenamiento local con BASED (consulteSección 6.9.8 [BASED], página 161) atributo. Puede ser un elemento 01 definido en la sección de enlace sin el atributo BASED, pero no se recomienda utilizar dicho elemento de datos.
5. Si se utiliza, el identificador 3 debe ser un PUNTERO (verSección 6.9.59 [USO], página 228) elemento de datos.
6. La cláusula opcional RETURNING devolverá la dirección del bloque de memoria asignado en el elemento de datos identificador USAGE POINTER-3 especificado. Cuando se utiliza esta opción, el programa conservará el conocimiento del tamaño solicitado originalmente del bloque de memoria asignado en caso de que se produzca una LIBERACIÓN (consulteSección 7.8.19 [GRATIS], página 332) alguna vez se emite una declaración contra el identificador 3.
7. Cuando se utiliza la opción identifier-1 junto con INITIALIZED (o su alternativa internacionalizada INITIALISED), el bloque de memoria asignado se inicializará como si fuera un INITIALIZE identifier-1 WITH FILLER ALL TO VALUE THEN TO DEFAULT (verSección 7.8.24 [INICIALIZAR], página 339) fueron ejecutados.
8. Cuando se utiliza la opción CHARACTERS expresión-aritmética-1, INITIALIZED inicializará

el bloque de memoria asignado a ceros binarios. Si no se utiliza INITIALIZED, el bloque de memoria asignado se inicializará a ceros binarios.

El contenido de la memoria asignada se dejará en manos de las reglas de asignación de memoria vigentes para el sistema operativo en el que se ejecuta el programa.

9. Hay dos formas básicas de utilizar esta afirmación. La más sencilla es:

   ASIGNAR Mi-01-Artículo

   Con esta forma, se asignará un bloque de almacenamiento de tamaño igual al tamaño definido de My-01-Item (que debe haberse definido con el atributo BASED). La dirección de ese bloque de almacenamiento se convertirá en la dirección base de My-01-Item para que este y sus elementos de datos subordinados se puedan utilizar dentro del programa.

   Un segundo enfoque (y equivalente) es:

   ASIGNAR LONGITUD DE CARACTERES My-01-Item DEVOLVIENDO The-Pointer
   ESTABLECER LA DIRECCIÓN DE My-01-Item A The-Pointer

10. Con este formulario, My-01-Item se puede definir con el atributo BASED o en la SECCIÓN DE ENLACE. En lugar de LONGITUD DE My-01-Item, también puede utilizar un tamaño menor al tamaño máximo del campo, siempre que se asegure de que nunca se utilice el campo completo.
11. Hacer referencia a un elemento de datos BASED antes de que se haya asignado su espacio de almacenamiento o después de que se haya liberado (mediante la declaración FREE) dará lugar a "resultados impredecibles". Así es como los manuales de referencia y las especificaciones de estándares hablan de esta situación. Según la experiencia del autor, los resultados son demasiado predecibles: el programa se interrumpe al intentar hacer referencia a un área de memoria no asignada.

7.8.4 ALTERAR

Sintaxis ALTER

ALTERAR procedimiento-nombre-1 PARA CONTINUAR A procedimiento-nombre-2
~~~~~ ~~

———————————————————————————————————————

La declaración ALTER se utilizó en los primeros años del lenguaje COBOL para editar el código objeto de un programa en tiempo de ejecución, cambiando un GO TO (verSección 7.8.22.1 [IR A] simple, página 335) para ramificarse a un lugar en el programa diferente de donde se encuentra la declaración GO TOFue compilado originalmente para.

1. Las palabras reservadas PROCEED y TO (la que aparece después de PROCEED) son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. *nombre-del-procedimiento-1*debe contener sólo una única declaración, y esa declaración debe ser un simple GO TO.
3. El efecto de esta declaración será como si el código de lenguaje de máquina generado para la declaración GO TO en nombre-de-procedimiento-1 se cambiara de modo que la declaración GO TO ahora transfiera el control a nombre-de-procedimiento-2, en lugar de a cualquier nombre de procedimiento especificado en el código fuente del programa.
4. Se ha agregado soporte para el verbo ALTER a GnuCOBOL con el fin de permitir que GnuCOBOL pase las pruebas del Instituto Nacional de Estándares y Tecnología (NIST) para el lenguaje de programación COBOL que requieren soporte para ALTER.
5. Debido al efecto catastrófico que esta declaración tiene sobre la legibilidad del programa y, por lo tanto, sobre la capacidad del programador para depurar problemas con la lógica del programa, se desaconseja FUERTEMENTE el uso de ALTER en programas nuevos.
~~~~~

7.8.5 LLAMAR

Sintaxis CALL

```
LLAM [ {LLAMADA          } ]  [ CON {LLAMADA  ENLACE  ]   {literal-1     }
 AR     ESTÁNDAR                    ESTÁNDAR
                                       }
~~~~ [ {~~~~~~~          } ]  [ ~~~~ {~~~~~~~} ~~~~~~~ ]   {identificado
                                                            r-1}
     [ {ESTÁTICO         } ]  [       {DO     }          ]
     [ {~~~~~~           } ]  [       {~      }          ]
     [ {DO               } ]  [        {PASCAL}          ]
     [ {~                } ]  [       {~~~~~~ }          ]
     [ {EXTERNO          } ]
     [ {~~~~~~           } ]
     [ {PASCAL           } ]
     [ {~~~~~~           } ]
     [ {nombre-mnemónico-1} ]

     [USANDO{[POR{REFERENC {[{          TAM ES AUTO      }] literal-2    }}...]
     IA}]                               AÑO
     [~~~~~{[   {~~~~~~~~~~}] {[{       ~~~~     ~~~~    }] identificador-2}}
                                                                          ]
     [      {[   { CONTENI  }] {[{      TAM ES POR       }]              }}  ]
                   DO                   AÑO     DEFECTO
     [      {[   { ~~~~~~~  }] {[{      ~~~~     ~~~~~~~ }]              }}  ]
     [      {[   { VALOR    }] {[{      TAM ES entero-1}]                }}  ]
                                         AÑO
     [      {[   { ~~~~~    }] {[{      ~~~~             }]              }}  ]
     [      {                  {[{NO    TAM ES AUTO      }]              }}  ]
                                FIRMADO  AÑO
     [      {                  {[{~~~~~~~~ ~~~~     ~~~~ }]              }}  ]
     [      {                  {[{NO    TAM ES entero-2}]                }}  ]
                                FIRMADO  AÑO
     [      {                  {[{~~~~~~~~ ~~~~     ~~~~ }]              }}  ]
     [      {                  {                                OMITIDO  }}  ]
     [      {                  {                                ~~~~~~~  }}  ]

     [      DEVOLVIENDO|DANDO      {      EN
     identificador-3    } ] [ ~~~~~~~~~~ ~~~~~~ {
     DIRECCIÓN DEL identificador-4 } ] [       {
     ~~~~~~~~           } ]
     [                  { NADA              } ]
     [                  { ~~~~~~~           } ]
     [                  { NULO              } ]
     [                  { ~~~~              } ]
     [                  { OMITIDO           } ]
     [                  { ~~~~~~~           } ]


     [ EN CASO DE DESBORDAMIENTO | EXCEPCIÓN declaración imperativa 1 ]
          ~~~~~~~~ ~~~~~~~~~~

     [ NO EN CASO DE DESBORDAMIENTO | EXCEPCIÓN declaración imperativa 2 ]
```

~~~ ~~~~~~~~

~~~~~~~~~~ [FIN DE LA LLAMADA]

~~~~~~~~

———————————————————————————————————————

La instrucción CALL se utiliza para transferir el control a una subrutina. VerCapítulo 11 [Sub-Programación], página 719, para conocer los detalles del uso de subprogramas con programas GnuCOBOL.
~~~~~~~~

1. Las palabras reservadas BY, IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas EXCEPCIÓN y DESBORDAMIENTO son intercambiables.
3. Las palabras reservadas DAR y DEVOLVER son intercambiables.
4. La expectativa es que la subrutina eventualmente devolverá el control al programa que la llamó, momento en el cual el programa que la llamó reanudará la ejecución comenzando con la instrucción inmediatamente posterior a la llamada. Sin embargo, los subprogramas no están obligados a regresar a sus invocadores y son libres de detener la ejecución del programa si así lo desean.
5. La opción mnemonic-name-1 / STATIC / STDCALL, si se utiliza, afecta las convenciones de enlace que se utilizarán para la subrutina que se llama, de la siguiente manera:

STATIC hace que la vinculación con la subrutina se realice de tal manera que se requiera que la subrutina esté vinculada estáticamente con el programa que la llama. Tenga en cuenta que esto permite utilizar la vinculación estática de forma selectiva para cada subrutina.

STDCALL permite utilizar las convenciones de llamada estándar del sistema (en contraposición a las convenciones de llamada de GnuCOBOL) al llamar a una subrutina. La definición de lo que constituye un "estándar del sistema" puede variar de un sistema operativo a otro. Su uso requiere conocimientos especiales sobre los requisitos de enlace de las subrutinas que se pretende llamar. Las subrutinas escritas en GnuCOBOL no necesitan esta opción.

nombre-mnemónico-1

permite utilizar una convención de llamada personalizada. Dichos nombres mnemotécnicos se definen utilizando CALL-CONVENTION (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90). Esa cláusula asocia un valor entero decimal con el mnemotécnico-nombre-1 de modo que los bits individuales activados o desactivados en el equivalente binario del entero afecten la vinculación con la subrutina como se describe en el siguiente cuadro. Las filas del cuadro marcadas con un "No" en la columna Compatible representan posiciones de bits (configuraciones de interruptores) en el valor entero que se aceptan actualmente (para proporcionar compatibilidad con otras implementaciones COBOL) si están codificadas, pero que no se admiten de otro modo.

Tenga en cuenta que el bit 0 es el bit más a la derecha en el valor binario.

Poco	**Apoyado**	**Significado si 0**	**Significado si 1**
0	No	Los argumentos se pasarán en secuencia de derecha a izquierda	Los argumentos se pasarán en secuencia de izquierda a derecha.
1	No	El programa que realiza la llamada eliminará los argumentos procesados de la pila de argumentos.	El programa llamado (subrutina) eliminará los argumentos procesados de la pila de argumentos.

2	Sí	El registro especial CÓDIGO DE RETORNO (verSección 7.7 [Específico- Registros comerciales], página 260) se actualizará además de cualquier dato de DEVOLUCIÓN o DONACIÓN artículo.	El registro especial CÓDIGO DE DEVOLUCIÓN no se actualizará (pero cualquier elemento de datos de DEVOLUCIÓN o ENTREGA sí lo hará).

3	Sí	Si se utiliza el literal CALL, el La subrutina se ubicará y vinculará con el programa que la llama en el momento de la compilación o puede ubicarse y cargarse dinámicamente en el momento de la ejecución, dependiendo de las configuraciones de los interruptores del compilador y las capacidades del sistema operativo.	Si se utiliza el literal CALL, el La subrutina sólo se puede localizar y vincular con el programa que la llama en el momento de la compilación.
4	No	Las convenciones "OPTLINK" de OS/2 no se utilizarán para LLAMAR al subprograma.	Se utilizarán las convenciones "OPTLINK" de OS/2 para LLAMAR al subprograma.
5	No	El "thunk" de Windows de 16 bits no tendrá efecto.	Se utilizará "thunk" de Windows de 16 bits para llamar a la subrutina como una DLL.
6	Sí	No se utilizará la convención STDCALL.	La convención STDCALL, necesaria para utilizar Microsoft Se utilizará la API Win32.

El uso de la opción STATIC en una declaración CALL es equivalente a usar CONVENCIÓN DE LLAMADA 8 (solo se establece el bit 3).

El uso de la opción STDCALL en una declaración CALL es equivalente a usar CALL CONVENTION 64 (solo el bit 6 establecido).

6. El valor de literal-1 o identificador-1 es el punto de entrada del subprograma que desea llamar.
7. Cuando se llama a una subrutina utilizando el identificador-1, se obliga al sistema de ejecución a llamar a un subprograma cargable dinámicamente. El contenido del identificador-1 será el nombre del punto de entrada dentro de ese módulo. Si esta es la primera llamada a cualquier punto de entrada dentro del módulo que se realiza en tiempo de ejecución, el contenido del identificador-1 debe ser el nombre del punto de entrada principal del módulo (que también debe coincidir con el nombre del archivo, menos cualquier extensión exigida por el sistema operativo) del archivo ejecutable que comprende el módulo.
8. Puede forzar al sistema de ejecución de GnuCOBOL a que cargue previamente todos los módulos cargados dinámicamente que el programa podría llamar en el momento en que el programa comience a ejecutarse. Esto se logra mediante el uso de la variable de entorno de ejecución COB_PRE_LOAD (consulteSección 10.2.3 [Variables del entorno de tiempo de ejecución], página 700). Si se usa, esto solo precargará aquellos módulos invocados a través de CALL literal-1, ya que el contenido de tiempo de ejecución del identificador-1 no se puede predecir.
9. Si el subprograma que se llama es un programa GnuCOBOL, y si ese programa tenía la INICIAL (verCapítulo 4 [DIVISIÓN DE IDENTIFICACIÓN], página 81) especificado en su cláusula PROGRAM-ID, todos los datos de división de datos del subprograma se restaurarán a su estado inicial cada vez que se ejecute el subprograma, independientemente de qué punto de entrada dentro del subprograma se esté referenciando.

Este comportamiento de [re]inicialización siempre se aplicará al almacenamiento local de cualquier subprograma (si lo hay), independientemente del uso (o no) de INITIAL.

10. La cláusula USING define una lista de argumentos que pueden pasarse desde el programa que realiza la llamada al subprograma. La forma en que se pasa un argumento determinado a la subrutina depende de la cláusula BY (si la hay) codificada (o implícita) para ese argumento, de la siguiente manera:

    POR REFERENCIA
    : Pasa la dirección del argumento al subprograma. Si el subprograma cambia el contenido de ese argumento, el cambio será "visible" para el programa que lo llama.

POR CONTENIDO
: Pasa la dirección de una copia del argumento al subprograma. Si el subprograma cambia el valor de dicho argumento, el cambio sólo afecta a la copia en el programa que lo llama, no a la versión original.

BY VALUE pasa el valor numérico real del literal o del contenido de los identificadores como argumento. Esta característica existe para proporcionar compatibilidad con C, C++ y otros lenguajes y normalmente no se utilizaría al llamar a subprogramas de GnuCOBOL. Solo los literales numéricos o los elementos de datos numéricos se deben pasar de esta manera.

Si un argumento no tiene una cláusula BY, se asumirá la especificación BY encontrada más recientemente en esa declaración CALL. Si el primer argumento especificado en una CALL no tiene una cláusula BY, se asumirá BY REFERENCE.

11. No se pueden pasar más de 251 argumentos a una subrutina, a menos que el compilador de GnuCOBOL se haya creado con un límite de argumentos específicamente diferente especificado para él. Si tiene acceso al código fuente de GnuCOBOL, puede ajustar este límite modificando el valor de COB_MAX_FIELD_PARAMS en el archivo call.c (que se encuentra en la carpeta libcob) así como la última declaración #if MAX_CALL_FIELD_PARAMS mostrada antes de ejecutar make para crear el compilador y la biblioteca de tiempo de ejecución.
12. La cláusula RETURNING le permite especificar un elemento de datos numéricos en el que la subrutina debe devolver un valor numérico. Si utiliza esta cláusula en la llamada, la subrutina debe incluir una cláusula RETURNING (consulteSección 7.3 [DIVISIÓN DE PROCEDIMIENTO DE DEVOLUCIÓN], página 248) en el encabezado de división de su procedimiento. Por supuesto, una subrutina puede pasar un valor de cualquier tipo en cualquier argumento pasado POR REFERENCIA.
13. Las cláusulas opcionales ON OVERFLOW y NOT ON OVERFLOW (o ON EXCEPTION y NOT ON EXCEPTION, que son intercambiables) se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de LLAMAR a la subrutina. El fracaso, en este contexto, se define como la incapacidad de localizar o cargar el código objeto de la subrutina en el momento de la ejecución. ConsulteSección 7.6.5 [EN DESBORDAMIENTO + NO EN DESBORDAMIENTO], página 256, para obtener información adicional.
14. Call también admite el uso de un punto de entrada almacenado en un PROGRAM-POINTER, lo que evita la búsqueda dinámica en tiempo de ejecución. GnuCOBOL mantiene un caché de búsquedas durante la ejecución de un programa. El uso repetido de una función nombrada no sufre mucha penalización, pero PROGRAM-POINTER será un poco más rápido. Para establecer un PROGRAM-POINTER, use SET program-reference TO ENTRY "name" (o obtenga la dirección de una API y participe en la programación de devolución de llamada).
15. Una extensión de CALL permite una llamada a un Program-Pointer-1 que está preestablecido usando SET program-pointer-1 TO ENTRY x. Además, la cláusula RETURNING puede devolver un puntero de datos o un PROGRAM-POINTER.

7.8.6 CANCELAR

Sintaxis CANCEL

```
CANCELAR { literal-1      }...
~~~~~~ { identificador-1 }
```

La instrucción CANCEL descarga de la memoria el módulo de subprograma cargable dinámicamente que contiene el punto de entrada especificado como literal-1 o identificador-1.

1. Si un módulo cargable dinámicamente descargado por la declaración CANCEL se vuelve a ejecutar posteriormente, todo el almacenamiento de división de datos para ese módulo volverá a estar en su estado inicial.
2. Si la declaración CANCEL realmente descarga físicamente un módulo cargado dinámicamente o simplemente lo marca como descargado lógicamente depende del uso y el valor de la variable de entorno de tiempo de ejecución COB_PHYSICAL_CANCEL (consulteSección 10.2.3 [Variables del entorno de tiempo de ejecución] [ables], página 700).

7.8.7 CERCA

Sintaxis CLOSE

```
CERRAR { nombre-archivo-1 [ { CARRETE|UNIDAD [ PARA ELIMINAR ] } ] }...
~~~~~                      { ~~~~ ~~~~        ~~~~~~~~   }
                           { CON CERRADURA               }
                           {        ~~~~                 }
                           {SIN REBOBINADO               }
                                    ~~ ~~~~~~~
```

Las cláusulas REEL, LOCK y NO REWIND se reconocen sintácticamente pero no son funcionales, excepto la declaración CLOSE...NO REWIND, que generará un estado de archivo de 07 en lugar del 00 habitual (pero no realizará ninguna otra acción).

———

La instrucción CLOSE finaliza el acceso del programa a los archivos especificados.

1. Las palabras reservadas FOR y WITH son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas REEL y UNIT son intercambiables.
3. La instrucción CLOSE sólo se puede ejecutar en archivos que se hayan abierto correctamente.
4. Un CLOSE exitoso escribirá cualquier buffer de registro no escrito restante en el archivo (similar a una declaración UNLOCK (verSección 7.8.52 [DESBLOQUEAR], página 405)) y libere cualquier bloqueo de archivo, independientemente del modo de apertura. Un archivo cerrado dejará de estar disponible para las instrucciones de E/S posteriores hasta que se vuelva a ABRIR.
5. Cuando una LÍNEA DE ORGANIZACIÓN SECUENCIAL (verSección 5.2.1.2 [LÍNEA DE ORGANIZACIÓN SE- QUENTIAL], página 110) o AVANCE DE LÍNEA (ver[LÍNEA AVANZANDO], página 15) el archivo es
   cerrado, se escribirá una secuencia delimitadora final en el archivo para señalar el punto de terminación del registro de datos final en el archivo. Esto solo será necesario si el registro final escrito en el archivo se escribió con DESPUÉS DE AVANZAR (verSección 7.8.54 [ESCRIBIR], página 410) opción.

7.8.8 COMPROMETERSE

Sintaxis COMMIT

COMPROMETERSE
~~~~~~

———————————————————————————————————————

La sentencia COMMIT realiza un DESBLOQUEO en cada archivo abierto actualmente, pero no cierra ninguno de los archivos. Consulte la sentencia UNLOCK (consulteSección 7.8.52 [DESBLOQUEAR], página 405) para obtener detalles adicionales.
~~~~~~

7.8.9 CALCULAR

Sintaxis COMPUTE

```
COMPUTAR { identificador-1
~~~~~~~
      [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] }...
        ~~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~~~          }
                              { MÁS CERCANO-LEJOS-DE-CERO }
                                { ~~~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { MÁS CERCANO-PAR             }
                              { ~~~~~~~~~~~~~~              }
                              { MÁS CERCANO A CERO          }
                        { ~~~~~~~~~~~~~~~~~~~~~~~~~~        }
                              { PROHIBIDO                   }
                              { ~~~~~~~~~~~                 }
                              { HACIA MAYOR                 }
                              { ~~~~~~~~~~~~~~~~~           }
                              { HACIA MENOR                 }
                              { ~~~~~~~~~~~~~~~             }
                              { TRUNCAMIENTO                }
                                ~~~~~~~~~~
        = | IGUAL expresión-aritmética-1          |    expresión booleana 1
          ~~~~~
      [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
            ~~~~ ~~~~~
      [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
        ~~~       ~~~~ ~~~~~
[FIN DEL CÁLCULO]
  ~~~~~~~~~~~
```

———

La instrucción COMPUTE proporciona un medio para realizar fácilmente operaciones aritméticas complejas con una sola instrucción, en lugar de utilizar secuencias engorrosas y posiblemente confusas de instrucciones ADD, SUBTRACT, MULTIPLY y DIVIDE. COMPUTE también permite el uso de exponenciación, una operación aritmética para la que no existe ninguna otra instrucción en COBOL.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La palabra reservada IGUAL es intercambiable con el uso de '='.
3. Cada identificador-1 debe ser un elemento de datos numérico o editado numéricamente.
4. El REDONDEADO opcional (verSección 7.6.7 [REDONDEADA], página 257) cláusula disponible para cada uno
   *identificador-1*controlará cómo se guardarán los resultados no enteros.
5. VerSección 2.1.25 [Expresiones aritméticas], página 40, para obtener más información sobre expresiones aritméticas.
6. Las cláusulas opcionales ON SIZE ERROR y NOT ON SIZE ERROR se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de realizar un cálculo. En este caso, el fracaso se define como tener un identificador-3 con una

cantidad insuficiente de posiciones de dígitos disponibles a la izquierda de cualquier punto decimal implícito o intentar dividir por cero.

VerSección 7.6.6 [EN ERROR DE TAMAÑO + NO EN ERROR DE TAMAÑO], página 257, para obtener información adicional.

7. El verbo COMPUTE también se puede utilizar para evaluar expresiones booleanas. Vea el siguiente ejemplo:

```
>>DIVISIÓN DE
IDENTIFICACIÓN LIBRE DE
FUENTES.PROGRAMA-ID.
TESTBOOLEAN. DIVISIÓN DE
DATOS.
SECCIÓN DE TRABAJO-ALMACENAMIENTO.
01 A imagen 9 comp-x valor B'00000101'.
01 B imagen 9 comp-x valor B'00000011'.
01 c imagen 9          valor VALOR BAJO.

DIVISIÓN DE PROCEDIMIENTOS.
  espacio de exhibición
  mostrar '    a          b      do       a                    b          do
            --'mostrar'---- ----  --------- --------- ---------  ---------
            '
  mostrar '    '  a  '  inicial 'b' 'c'   ' FUNCIÓN BIT-Of(a) ' inicial
        ' FUNCIÓN BIT-Of(b) ' ' FUNCIÓN BIT-OF(c)
  espacio de exhibición     espacio de exhibición
  mostrar '    un operador b    do       a      operador    b          do
            --'mostrar'---- ----  --------- --------- ---------  ---------
            '
  Calcular c = a B-Y b
  mostrar '    '  a  '    Y       ' b '  =  'c'        '   FUNCIÓN BIT-Of(a)      '
        Y                  ' BIT-De(b) ' = 'FUNCIÓN     BIT-DE(c)
              FUNCIÓ
                    N
   calcular  c = un  B-O b
  mostra '    '  a  '    O       ' b '  =  'c'        '       FUNCIÓN BIT-Of(a)   '
     r
        O                  ' BIT-De(b) ' = 'FUNCIÓN     BIT-DE(c)
              FUNCIÓ
                    N
   calcular  c = un  B-XOR b
  mostra '    '          a  '    ' b '  =  'c'        '       FUNCIÓN BIT-Of(a)   '
     r                GRATIS
        GRATIS  'FUNCIÓN    BIT-De(b) ' = 'FUNCIÓN     BIT-DE(c)
   calcular  c = B-NO b
  mostra '    –'    '    NO      ' b '  =  'c'            –   '                '
     r
        NO      'FUNCIÓN    BIT-De(b) ' = 'FUNCIÓN     BIT-DE(c)
  aceptar omitido
  STOP RUN.
```

8. y en particular con los operadores de desplazamiento booleanos, vea el siguiente ejemplo:

```
>> DIVISIÓN DE
IDENTIFICACIÓN LIBRE DE FUENTE.
```

```
PROGRAMA-ID.
TESTBOOLEANSHIFT. DIVISIÓN DE
DATOS.
SECCIÓN DE TRABAJO-ALMACENAMIENTO.
01 A imagen 9 comp-x valor B'00000001'.
01 B imagen 9 comp-x valor B'00000101'.
01 c imagen 9          valor VALOR BAJO.

DIVISIÓN DE PROCEDIMIENTOS.
```

```
espacio de exhibición
mostrar '     a                b       do         a                        b            do
            -- 'mostrar-'   ----    ----     --------- ----------- ---------    ---------
              '
mostrar '     '   a   '   inicial    ' b '      'c'   ' FUNCIÓN BIT-Of(a)
              ' inicial            ' FUNCIÓN BIT-Of(b) '                '
           FUNCIÓN BIT-OF(c)
espacio de exhibición     espacio de exhibición
mostrar '     Un operador b       do         a       operador       b            do
            -- 'mostrar-'   ----    ----     --------- ----------- ---------    ---------
              '
calcular c = a B-SHIFT-L       b
mostrar '     '   a   ' B-MAYÚS-L   ' b '          =  'c'                    '
              FUNCIÓN BIT-Of(a)   ' B-MAYÚS-L       ' FUNCIÓN
         BIT-Of(b) ' = ' FUNCIÓN BIT-OF(c)
calcular c = a B-SHIFT-LC b
mostrar '     '   a   ' B-SHIFT-LC ' b ' =          'c''
              FUNCIÓN BIT-Of(a)            'B-SHIFT-LC ' FUNCIÓN BIT-Of(b) ' = '
         FUNCIÓN BIT-OF(c)
calcular c = a B-SHIFT-R       b
mostrar '     '   a   ' B-MAYÚS-R   ' b '          =  'c'                    '
              FUNCIÓN BIT-Of(a)   ' B-MAYÚS-R       ' FUNCIÓN
         BIT-Of(b) ' = ' FUNCIÓN BIT-OF(c)
calcular c = a B-SHIFT-RC b
mostrar '     '   a   ' B-SHIFT-RC ' b '=          'c''
              FUNCIÓN BIT-Of(a)            'B-SHIFT-RC ' FUNCIÓN BIT-Of(b) ' = '
         FUNCIÓN BIT-OF(c)
aceptar omitido
STOP RUN.
```

7.8.10 CONTINUAR

Sintaxis CONTINUAR

CONTINUAR
~~~~~~~~

```
                    { identificador-1          }
CONTINUAR DESPUÉS                              { literal-1  }      ARTÍCULOS DE SEGUNDA CLASE
~~~~~~~~ ~~~~~      { expresión-aritmética-1 }    ~~~~~~~
```

———

La declaración CONTINUE es una declaración sin operación que puede codificarse en cualquier lugar como un imperativo.declaración (ver[Declaración imperativa], página 758) pueden estar codificados.

1. La instrucción CONTINUE no tiene ningún efecto en la ejecución del programa.
2. Esta declaración (quizás en combinación con uno o dos comentarios apropiados) hace una"Marcador de posición" conveniente, particularmente en ELSE (verSección 7.8.23 [IF], página 338) o CUÁNDO (verSección 7.8.15 [EVALUAR], página 321) cláusulas en las que actualmente no se espera que se incluya ningún códigoser necesario, pero se debe reservar un lugar para que el código maneje las condiciones en cuestión en caso de que alguna vez sea necesario.
3. La extensión opcional de (AFTER) cuando se utiliza con la declaración CONTINUE pausa la ejecución durante un período de tiempo específico.
```

7.8.11 BORRAR

Sintaxis DELETE

```
ELIMINAR REGISTRO nombre-archivo-1
~~~~~~
  [ CLAVE NO VÁLIDA declaración-imperativa-1 ]
    ~~~~~~~
  [ CLAVE NO VÁLIDA declaración-imperativa-2 ]
    ~~~
~~~~~~~[ FIN-
ELIMINAR ]
~~~~~~~~~~
```

La instrucción DELETE elimina lógicamente un registro de un archivo COBOL.

1. Las palabras reservadas KEY y RECORD son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. La ORGANIZACIÓN del archivo-nombre-1 no puede ser LÍNEA DE ORGANIZACIÓN SECUENCIAL (verSección 5.2.1.2 [LÍNEA DE ORGANIZACIÓN SECUENCIAL], página 110).
3. El archivo nombre-de-archivo-1 no puede ser un archivo de trabajo de ordenación/combinación (un archivo descrito mediante una SD (verSección 6.2.1 [Archivo/Ordenar-Descripción], página 123)).
4. Para los archivos en el modo de acceso SECUENCIAL, la última instrucción de entrada-salida ejecutada contra el nombre-de-archivo-1 antes de la ejecución de la instrucción DELETE debe haber sido una instrucción READ en formato secuencial ejecutada exitosamente (verSección 7.8.34.1 [LECTURA secuencial], página 365). Por tanto, dicha lectura identificará el registro que se desea eliminar.
5. Si nombre-archivo-1 es un archivo RELATIVO cuyo ACCESOMODO (verSección 5.2.1.3 [ORGANIZACIÓN RELATIVO], página 112) es cualquieraALEATORIO o DINÁMICO, el registro a eliminar es aquel cuyo número de registro relativo es actualmente el valor del campo especificado como CLAVE RELATIVA del archivo en su declaración SELECT.
6. Si nombre-archivo-1 es un archivo INDEXADO cuyo ACCESOMODO (verSección 5.2.1.4 [ORGANIZACIÓN INDEXADO], página 114) esALEATORIO o DINÁMICO, el registro a eliminar es aquel cuya clave principal es actualmente el valor del campo especificado como CLAVE DE REGISTRO en la declaración SELECT del archivo.
7. Las cláusulas opcionales INVALID KEY y NOT INVALID KEY se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de eliminar un registro. ConsulteSección 7.6.3 [CLAVE INVÁLIDA + CLAVE NO INVÁLIDA], página 255, para obtener información adicional.
8. No se puede especificar ninguna cláusula CLAVE INVÁLIDA o CLAVE NO INVÁLIDA para un archivo cuyo MODO DE ACCESO SEA SECUENCIAL.

7.8.12 MOSTRAR

7.8.12.1 MOSTRAR EN EL dispositivo

Sintaxis del dispositivo DISPLAY UPON

```
PANTALLA { literal-1    }...
~~~~~~~ { identificador-1 }
    [ UPON nombre-mnemónico-1 ]
      ~~~~
          [SIN AVANZAR]
             ~~ ~~~~~~~~~~
    [ ON EXCEPTION declaración imperativa 1 ]
          ~~~~~~~~~~
    [NO BAJO EXCEPCIÓN declaración imperativa 2 ]
      ~~~
                ~~~~~~~~~~
[FIN DE LA
PANTALLA]
  ~~~~~~~~~~~
```

Este formato de la declaración DISPLAY muestra el contenido del identificador especificado y/o los valores literales en el dispositivo de salida del sistema especificado mediante la cláusula UPON.

1. Las palabras reservadas ON y WITH son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Si no se especifica ninguna cláusula UPON, se asumirá UPON CONSOLE. Si se especifica la cláusula UPON, mnemonic-name-1 debe ser uno de los nombres de dispositivo de salida integrados CONSOLE, PRINTER, STDERR, STDOUT, SYSERR, SYSLIST, SYSLST o SYSOUT o un nombre mnemónico asignado a uno de esos dispositivos a través de SPECIAL-NAMES (consulteSección 5.1.3 [NOMBRES ESPECIALES], página 90) párrafo.

   Al visualizar en los dispositivos STDERR o SYSERR o en un nombre-mnemónico-1 adjunto a uno de esos dos dispositivos, la salida se escribirá en la tubería de salida n.° 2, lo que normalmente hará que la salida aparezca en la ventana de salida de la consola. Si lo desea, puede redirigir esa salida a un archivo agregando 2> nombre_archivo al final del comando que ejecuta el programa. Esto se aplica tanto a las versiones de GnuCOBOL para Windows (cualquier tipo) como para Unix.

   Al visualizar en los dispositivos CONSOLE, PRINTER, STDOUT, SYSLIST, SYSLST o SYSOUT o en un mnemotécnico-name-1 adjunto a uno de ellos, la salida se escribirá en el conducto de salida n.° 1, lo que normalmente hará que la salida aparezca en la ventana de salida de la consola. Si lo desea, puede redirigir esa salida a un archivo agregando 1> filename o simplemente > filename al final del comando que ejecuta el programa. Esto se aplica tanto a las versiones de GnuCOBOL para Windows (de cualquier tipo) como para Unix.
3. La cláusula NO ADVANCING, si se utiliza, suprimirá la secuencia de retorno de carro/salto de línea que normalmente se agrega al final de cualquier pantalla de consola.
4. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de mostrar la salida en el dispositivo especificado. ConsulteSección 7.6.4 [CON EXCEPCIÓN + SIN EXCEPCIÓN], página 256, para obtener información adicional.

7.8.12.2 MOSTRAR EN LA LÍNEA DE COMANDOS

Sintaxis de MOSTRAR EN LA LÍNEA DE COMANDOS

```
PANTALLA { literal-1     }...
~~~~~~~ { identificador-1 }
  AL  {  NÚMERO DE ARGUMENTO|LÍNEA DE COMANDOS}
      ~~~~     ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~
    [ ON EXCEPTION declaración imperativa 1 ]
         ~~~~~~~~~
    [NO BAJO EXCEPCIÓN declaración imperativa 2 ]
      ~~~
                ~~~~~~~~~~
[FIN DE LA
PANTALLA]
  ~~~~~~~~~~~
```

Esta forma de la instrucción DISPLAY se puede utilizar para especificar el número de argumento de la línea de comandos que se recuperará mediante una instrucción ACCEPT FROM ARGUMENT-VALUE posterior (consulteSección 7.8.1.2 [ACEPTARDESDE LA LÍNEA DE COMANDOS], página 264) o para especificar un nuevo valor para los propios argumentos de la línea de comandos.

1. La palabra reservada ON es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Al mostrar un valor numérico entero UPON ARGUMENT-NUMBER, especificará qué argumento (por su número relativo) será recuperado por una declaración ACCEPT FROM ARGUMENT-VALUE posterior.
3. La ejecución de DISPLAY UPON COMMAND-LINE influirá en las declaraciones ACCEPT FROM COMMAND-LINE posteriores (que luego devolverán el valor mostrado), pero no influirá en las declaraciones ACCEPT FROM ARGUMENT-VALUE posteriores; estas continuarán devolviendo los parámetros de ejecución del programa original.
4. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de mostrar la salida del elemento especificado. ConsulteSección 7.6.4 [CON EXCEPCIÓN + SIN EXCEPCIÓN], página 256, para obtener información adicional.

7.8.12.3 MOSTRAR EN EL NOMBRE DEL ENTORNO

Sintaxis de MOSTRAR EN NOMBRE DEL ENTORNO

```
PANTALLA { literal-1          }...   AL  {  VALOR AMBIENTAL}
~~~~~~~ { identificador-1 }     ~~~~ { ~~~~~~~~~~~~~~~~~~ }
                                       {  NOMBRE DEL ENTORNO }
                                         ~~~~~~~~~~~~~~~~~
    [ ON EXCEPTION declaración imperativa 1 ]
          ~~~~~~~~~
    [NO BAJO EXCEPCIÓN declaración imperativa 2 ]
      ~~~
                ~~~~~~~~~
[FIN DE LA
PANTALLA]
  ~~~~~~~~~~~
```

Esta forma de la instrucción DISPLAY se puede utilizar para crear o modificar variables de entorno.

1. La palabra reservada ON es opcional y puede omitirse. La presencia o ausencia de esta palabra no tiene efecto sobre el programa.
2. Para crear o cambiar una variable de entorno se requieren dos instrucciones DISPLAY. El siguiente ejemplo establece la variable de entorno MY_ENV_VAR en un valor de 'Valor de demostración':

   MOSTRAR "MY_ENV_VAR" EN EL NOMBRE DEL ENTORNO
   MOSTRAR "Valor Demostrativo" SOBRE EL VALOR AMBIENTAL
3. Las variables de entorno creadas o modificadas desde dentro de los programas GnuCOBOL estarán disponibles para cualquier proceso de sub-shell generado por ese programa (es decir, CALL 'SYSTEM' (verSección 8.2.61 [SISTEMA], página 644)) pero no será conocido por el shell o la ventana de consola que inició el programa GnuCOBOL.
4. Considere utilizar SET ENVIRONMENT (verSección 7.8.43.1 [ESTABLECER ENTORNO], página 379) en lugar de DISPLAY para establecer variables de entorno, ya que es mucho más simple.
5. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de mostrar la salida del elemento especificado. ConsulteSección 7.6.4 [CON EXCEPCIÓN + SIN EXCEPCIÓN], página 256, para obtener información adicional.

7.8.12.4 MOSTRAR elemento de datos

Sintaxis del elemento de datos DISPLAY

```
MOSTRAR     { identificador-1 } [SOBRE CRT|CRT-BAJO ] [ EL MODO ES BLOQUE ]
~~~~~~~  { literal-1        }     ~~~~ ~~~ ~~~~~~~~~~         ~~~~      ~~~~~

          [ EN { LÍNEA              NUMERO { entero-1                    } ]
          [ ~~ { ~~~~                        { identificador-2            } ]
          [      {                           { expresión-aritmética-1 }
          ] [    { COLUMNA | NÚMERO DE POSICIÓN { entero-2                } ]
          [      { ~~~     ~~~               { identificador-3            } ]
          [      {                           { expresión-aritmética-2 } ] [
                 { { entero-3     }                                       } ]
          [      { { identificador-4 }                                    } ]

          [ CON [ CAMPANA | PITIDO ]
             ~~~~    ~~~~     ~~~~
                 [ EN BLANCO { LÍNEA | PANTALLA } ]
                    ~~~~~     ~~~~ ~~~~~~
                 [BORRAR { EOL|EOS                        } ]
                 [ ~~~~~ { ~~~ ~~~                        } ]
                 [          { [HASTA EL FINAL DE ] {LÍNEA | PANTALLA } } ]
                                ~~~          ~~~~     ~~~~~~
                 [ COLOR-DE-FONDO | EL COLOR-DE-FONDO ES |= entero-4 | identificador-6]
                    ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~
                 [ COLOR-DE-PRIMERO| EL COLOR-DE-PRIMERO ES |= entero-5| identificador-7]
                    ~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~ [
                 RESALTAR | DESTACAR ]          [
                 PARPADEAR ]
                    ~~~~~~~~~     ~~~~~~~~          ~~~~~
                 [ VIDEO INVERSO | INVERSO | INVERTIDO ]
                    ~~~~~~~~~~~~~~  ~~~~~~~
                                    ~~~~~~~~[
                 SOBRELÍNEA ] [ SUBRAYADO ]
                    ~~~~~~~~       ~~~~~~~~~
                 [ DESPLAZARSE [ ARRIBA          ] [ { entero-4      } {
                 LÍNEA|LÍNEAS } ] ]
                 [ ~~~~~~ [ ~~  ] [ { identificador-5 }~~~~ ~~~~~   ] ]
                 [            [ ABAJO]                                ]
                                ~~~~
                 [ CONVERSIÓN ]
                    ~~~~~~~~~~
                 [ EL TAMAÑO ES          {
                 entero-5  } ] [ ~~~~       {
                 identificador-6 } ]

                 [ CONTROL
                              {literal-7       } ]
                 [ ~~~~~~~                    {
                 identificador-7 } ]
```

```
[ { COLOR | COLOR } ES { entero-8 } ]
[     ~~~~~~    ~~~~~        { identificador-

8 } ] [ EN CASO DE EXCEPCIÓN      declaración

imperativa 1 ]
```

~~~~~~~~~~

[NO BAJO EXCEPCIÓN declaración imperativa 2]

~~~

~~~~~~~~~~

[FIN DE LA PANTALLA]

~~~~~~~~~~~~

Las cláusulas UPON CRT, UPON CRT-UNDER y CONVERSION se reconocen sintácticamente, pero no son funcionales. Se admiten para proporcionar compatibilidad con el código fuente COBOL escrito para otras implementaciones COBOL.

——

Este formato de la declaración DISPLAY presenta datos en una pantalla formateada.

1. Las palabras reservadas AFTER, LINE, LINES, NUMBER y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Las palabras reservadas COLUMNA y POSICIÓN son intercambiables.
3. Las palabras reservadas LINE y LINES son intercambiables.
4. Si el identificador-1 está definido en la SECCIÓN DE PANTALLA (verSección 6.7 [SECCIÓN DE PANTALLA], página 142), se ignorarán todas las cláusulas AT, Attribute-Specification y WITH. Toda definición de campo, posicionamiento del cursor y control de pantalla se producirán como resultado de la definición de la sección de pantalla del identificador-1.
5. La palabra reservada OMITIDO cuando se utiliza, actuará para posicionar el cursor o cualquier espacio libre de la pantalla sin cambiar ningún contenido de la pantalla.
6. Los siguientes puntos se aplican si el identificador-1 no está definido en la sección de pantalla:
   A. El propósito de la cláusula AT es definir dónde se debe mostrar el identificador 1 en la pantalla. VerSección 7.8.1.4 [ACEPTAR elemento de datos], página 267, para obtener información adicional.
   B. El propósito de la cláusula WITH es definir los atributos visuales que deben aplicarse al identificador-1 cuando se muestra en la pantalla, así como otras características de control de presentación.
   C. Las siguientes cláusulas de especificación de atributos están permitidas en la declaración DISPLAY
      — Estos son los mismos que los permitidos para los elementos de datos de la SECCIÓN DE PANTALLA. Un elemento particular
      *Especificación de atributos*se puede utilizar solo una vez en cualquier PANTALLA:
      - COLOR DE FONDO (verSección 6.9.6 [COLOR DE FONDO], página 159)
      - BEEP (verSección 6.9.9 [BEEP], página 162), CAMPANA (verSección 6.9.10 [BELL], página 163)
      - EN BLANCO (verSección 6.9.11 [EN BLANCO], página 164)
      - PARPADEO (verSección 6.9.13 [BLINK], página 166)
      - BORRAR (verSección 6.9.18 [BORRAR], página 172)
      - COLOR DE PRIMER PLANO (verSección 6.9.21 [COLOR DEL PRIMER PLANO], página 175)
      - DESTACADO (verSección 6.9.26 [HIGHLIGHT], página 180)
      - LUZ BAJA (verSección 6.9.32 [LOWLIGHT], página 188)
      - SOBRELINEA (verSección 6.9.36 [OVERLINE], página 194)
      - VIDEO INVERSO (verSección 6.9.44 [VÍDEO INVERSO], página 208)
      - SUBRAYAR (verSección 6.9.57 [SUBRAYADO], página 226)
   D. CONTROL La frase CONTROL permite la especificación dinámica (en tiempo de ejecución en lugar de en tiempo de compilación) de características. Literal-7 debe ser

un literal no numérico. Identificador-7 debe ser un elemento de datos no numérico. El valor de identificador-7 o literal-7 en la frase CONTROL debe ser una cadena de caracteres que consista en una serie de palabras clave delimitadas

por comas; algunas palabras clave permiten la asignación de un valor si se coloca un signo igual y el valor después de la palabra clave. Los espacios en blanco se ignoran en la cadena de caracteres. Las letras minúsculas se tratan como letras mayúsculas dentro de las palabras clave. Las palabras clave especificadas anulan las opciones estáticas correspondientes especificadas como frases. Las palabras clave se pueden especificar en cualquier orden. Las palabras clave que especifican opciones que no se aplican a la declaración se ignoran.

E. Las palabras clave que afectan a una declaración DISPLAY son BEEP, BLINK, CONVERT, REVERSE, UNDERLINE, ERASE, ERASE EOL, ERASE EOS, HIGH, LOW, NO BEEP, NO BLINK, NO CONVERT, NO REVERSE, NO UNDERLINE NO ERASE.
Los significados de estas palabras clave cuando aparecen en el valor del operando de frase CONTROL son los mismos que las frases correspondientes que pueden escribirse como opciones estáticas de la declaración DISPLAY, con la adición de las formas negativas para permitir la supresión de las opciones declaradas estáticamente.

F. GnuCOBOL proporciona tres palabras clave adicionales en la frase CONTROL que afectan a un campo DISPLAY. 1. FCOLOR = color-name Cuando FCOLOR está presente, color-name especifica el color de primer plano del campo DISPLAY. Este nombre se utiliza entonces como el valor predeterminado para las declaraciones DISPLAY posteriores en el programa. El valor predeterminado inicial para color-name es blanco. 2. BCOLOR = color-name Cuando BCOLOR está presente, color-name especifica el color de fondo del campo DISPLAY. Este valor se utiliza entonces como el valor predeterminado para las declaraciones DISPLAY posteriores en el programa. El valor predeterminado inicial para color-name es blanco.El valor predeterminado para el nombre del color es el negro. La siguiente tabla contiene una lista de todos los nombres posibles para el nombre del color. La columna de la izquierda contiene el nombre del color válido. La columna de la derecha muestra el color que aparece cuando se especifica una intensidad alta (la intensidad predeterminada).

Nombres de colores COBOL válidos	
Nombres de colores válidos	Valores de color de alta intensidad (valores predeterminados)
Negro	Gris
Azul	Luz Azul
Verde	Luz Verde
Cian	Luz Cian
Rojo	Luz Rojo
Magenta	Luz Magenta
Marrón	Amarillo
Blanco	Alta intensidad Blanco

G. GRÁFICOS La palabra clave GRAPHICS hace que los caracteres de la siguiente tabla seantraducido a caracteres de dibujo lineal. Los caracteres que no están incluidos en la siguiente tabla se imprimen sin cambios.

Personajes de dibujo lineal		
Descripción	Carácter de una sola línea	Carácter de dos líneas
esquina inferior derecha	y(+)	Yo(+)
esquina superior derecha	k(+)	K(+)
esquina superior izquierda	yo(+)	L(+)
esquina inferior izquierda	m(+)	M(+)

```
|más                  |        n(+)            |     norte(+)            |
|horizontal  línea    |        q(-)            |        Q(-)             |
|tee                  |        +               |        Yo()             |
izquierdo
|tee                  |        en()            |        EN()             |
derecho
|camiseta de          |        en(-)           |        En(-)            |
abajo
|camiseta             |        En(-)           |        EN(-)            |
superior
|línea vertical       |        incógni         |        INCÓGNI          |
                                 ta()                    TA()
+--------------------+----------------------+-----------------------------------+
```

H. Si los caracteres de dibujo de línea solicitados no están disponibles, el sistema de ejecución utiliza los mejores caracteres disponibles. Si se solicitan caracteres de línea doble y solo hay caracteres de línea simple disponibles, se utilizan. Si no hay caracteres de dibujo de línea disponibles, se utilizan caracteres de signo más, barras verticales y guiones.

Programa de ejemplo que demuestra cómo se dibujan los cuadros:

```
>>IDENTIFICACIÓN DE FUENTE
LIBRE
                    DIVISIÓN.I
D DEL PROGRAMA.     CONTROL1.
DATOS               DIVISIÓN.
TRABAJO-ALMACENAMIENTO
                    SECCIÓN.
01   bandera de éxito PIC X VALOR 'Y'.
     88   éxito      VALUE 'Y', 'y'.
77   LIN-COMENZAR  FOTO   9 9   COMP-5.
77   Lino          FOTO   9 9   COMP-5.
01   Imagen 1 X(75)
          VALOR 'Ingrese "y" si ve caracteres de dibujo lineal. ' & 'El
          primer conjunto (simple/doble)'.
01   Imagen 2 de scr2 X(75)
          VALOR 'usa HIGHLIGHT, el segundo usa '&
          'LUZ BAJA, PARPADEO y MAGENTA.'.
01   control de gráficos PIC X(50) VALOR 'ALTO,
GRÁFICOS'. PROCEDIMIENTO DIVISIÓN.
     MOVER 2 A LIN
     MOSTRAR scr1 EN LÍNEA LIN COL 2
     AGREGAR   1 A LIN
     MOSTRAR scr2 EN LA LÍNEA LIN COL
     2MOVER 5 A LIN-START
     LLEVAR A CABOD S P   C O L
     MOVER 12 A LIN-START
     MUEVA "LOW BLINK FCOLOR=MAGENTA GRAPHICS" A graphcontrol
     EJECUTAR DSPCOL
     ACEPTAR bandera de éxito A LAS 1801
     ACTUALIZACIÓN REQUERIDA SI éxito Y COB-CRT-
```

```
            STATUS = 0
                REGRESAR
            REGRESANDO 0 DE LO
            CONTRARIO
                REGRESO REGRESANDO 1.
        Código DSC01.
      *>    Gráficos de una sola
            líneaMOVER LIN-START A
            LIN
```

```
        MOSTRAR                         LÍNE Lino COL 05, CONTROL     control
        "lqqqqwqqqqk" AÑADIR              A        UM                   gráfico.
                1 A LIN                            NA
        MOSTRAR "x                      LÍNE Lino COL 05, CONTROL     control
                incógnita                 A        UM                   gráfico.
                x"AGREGAR       1                  NA
        A LIN
        MOSTRAR                         LÍNE Lino COL 05, CONTROL     control
        "tqqqqnqqqqu" AÑADIR              A        UM                   gráfico.
                1 A LIN                            NA
        MOSTRAR "x                      LÍNE Lino COL 05, CONTROL     control
                incógnita                 A        UM                   gráfico.
                x"AGREGAR       1                  NA
        A LIN
        MOSTRAR "mqqqqvqqqqj"           LÍNE Lino COL 05, CONTROL     control
*>      Doble linea    gráficos           A        UM                   gráfico.
        MOVER LIN-START A LIN                      NA
        DISPLAY
        "LQQQQWQQQQK"                   LÍNE Lino COL 20, CONTROL     control
        AGREGAR   1  A LIN                A        UM                   gráfico.
        MOSTRAR "X      incógnita                  NA
                X"AGREGAR       1
        A LIN
        MOSTRAR
        "TQQQQNQQQQU" AÑADIR
                1  A LIN
        MOSTRAR "X      incógnita
                X"AGREGAR       1
        A LIN
        PANTALLA "MQQQQVQQQQJ"
```

COLOR La frase COLOR proporciona un método alternativo para configurar los atributos de video. Integer-8 debe ser un literal numérico. Identifier-8 debe ser un elemento de datos numérico. También permite la especificación de colores para los campos y controles de pantalla. Se pueden configurar con diferentes valores numéricos para expresar varias combinaciones de colores y atributos de video. Puede realizar combinaciones sumando los valores apropiados. Se aceptan los siguientes valores de color:

Color	Primer plano	Antecedentes
Negro	1	32
Azul	2	64
Verde	3	96
Cian	4	128
Rojo	5	160
Magenta	6	192
Marrón	7	224
Blanco	8	256

I. Puede especificar otros atributos de vídeo agregando los siguientes valores:

Video inverso	1024
Baja intensidad	2048
Alta intensidad	4096
Subrayar	8192
Parpadeo	16384
Protegido	32768

```
|Fondo de baja intensidad|        65536 |
|Fondo de alta intensidad| 131072 |
+-------------------------+------------+
```

J. También puede especificar una intensidad alta agregando "8" al valor del color de primer plano.

Programa de ejemplo que demuestra cómo se utiliza COLOR.

```
        >>FUENTE
DIVISIÓN DE IDENTIFICACIÓN
LIBRE. PROGRAMA-ID.
COLOR1.
*> Ejemplo de uso de la cláusula
"COLOR" DIVISIÓN DE DATOS.
SECCIÓN DE TRABAJO-ALMACENAMIENTO.
01 wLIN                FOTO     99 valor cero.
01 wCOLOR              PIC 999999 valor cero.
01  wAdelante negro    FOT 999  valor     1.
                        O
01  wEspalda negra     FOT 999  valor    32.
                        O
01  wAzul-Adelante     FOT 999  valor     2.
                        O
01  wEspalda azul      FOT 999  valor    64.
                        O
01  wVerde-Adelante    FOT 999  valor     3.
                        O
01  wEspalda verde     FOT 999  valor    96.
                        O
01  wCyan-Adelante     FOT 999  valor     4.
                        O
01  wCyan-Espalda      FOT 999  valor   128.
                        O
01  wRojo-Adelante     FOT 999  valor     5.
                        O
01  wEspalda roja      FOT 999  valor   160.
                        O
01  wMagenta-          FOT 999  valor     6.
    Adelante            O
01  wMagenta-Atrás     FOT 999  valor   192.
                        O
01  wMarrón-Adelante   FOT 999  valor     7.
                        O
01  wEspalda marrón    FOT 999  valor   224.
                        O
01  wDelantero         FOT 999  valor     8.
    blanco              O
01  wEspalda blanca    FOT 999  valor   256.
                        O
01  Vídeo inverso          FO   999999  valor     1024.
                           TO
01  wBaja                  FO   999999  valor     2048.
    intensidad             TO
01  wAlta                  FO   999999  valor     4096.
```

```
    intensidad                    TO
01  wSubrayar                     FO   999999  valor     8192.
                                  TO
01  Parpadeo                      FO   999999  valor    16384.
                                  TO
01  wProtegido                    FO   999999  valor    32768.
                                  TO
01  wFondo de baja intensidad     FO   999999  valor    65536.
                                  TO
01  wFondo de alta intensidad     FO   999999  valor   131072.
                                  TO

 DIVISIÓN DE
  PROCEDIMIENTOS.
  AGREGAR 1 a WLIN
  Calcular wCOLOR = wCian-Adelante + wBlanco-Atras + wSubrayado
  FUNCIÓN DE VISUALIZACIÓN CONCATENAR ("XXX COLOR=" wCOLOR "
                          XXXXXXXXX ")EN LINEA wLIN COL 001 COLOR wCOLOR.
  AGREGAR 1 a WLIN
  Calcular wCOLOR = wMarrón-Adelante + wRojo-Espalda +
  wAltaIntensidadFUNCIÓN DE VISUALIZACIÓN CONCATENAR ("XXX COLOR="
  wCOLOR " XXXXXXXXX ")
                          EN LINEA wLIN COL 001 COLOR wCOLOR.
```

```
AGREGAR 1 a WLIN
calcular wCOLOR = wMarrón-Adelante + wRojo-Espalda + wAltaIntensidad +
wVideoInverso FUNCIÓN DE VISUALIZACIÓN CONCATENAR ("XXX COLOR=" wCOLOR "
XXXXXXXXX ")
                    EN LINEA wLIN COL 001 COLOR wCOLOR.
AGREGAR 1 a WLIN
Calcular wCOLOR = wBlanco-Adelante + wVerde-Espalda + wParpadeo
FUNCIÓN DE VISUALIZACIÓN CONCATENAR ("XXX COLOR="    wCOLOR "
                    XXXXXXXXX ") EN LA LÍNEA wLIN COL 001 COLOR
                    wCOLOR.
DETÉNGASE, CORRA.
```

K. VerSección 7.8.1.4 [ACEPTAR elemento de datos], página 267, para obtener información adicional sobre las otras opciones de la cláusula WITH.

7. Las cláusulas opcionales ON EXCEPTION y NOT ON EXCEPTION se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, del intento de entrada/salida de pantalla. ConsulteSección 7.6.4 [EN EXCEPCIÓN + NO EN EXCEPCIÓN], página 256, para obtener información adicional.

Cuando se utiliza DISPLAY con Línea y columna donde se utilizan múltiples variables o literales antes de LINE, solo se mostrará la primera.

Si esto es necesario, se utiliza CONCATENAR para construir más de un elemento junto antes de la visualización, por ejemplo, FUNCIÓN DE VISUALIZACIÓN CONCATENAR (VARS-1 VARS-2) EN 0201.

Cuando se utiliza DISPLAY sin controles de línea o columna, solo puede aparecer una variable o literal en una línea, por lo que también se debe emplear el ejemplo anterior.

7.8.12.5 Elemento de datos DISPLAY (Microsoft v1-v2)

Sintaxis del elemento de datos DISPLAY

```
MOSTRAR     [posición-especificación] {identificador-2 | literal-1} ...
~~~~~~~
  [ CON [ Atributo-Especificación ]...
    ~~~~
        [BORRAR                    { PANTALLA|LÍNEA }       ]

        [ DESPLAZARSE { ARRIBA          } [ { entero-3       } LINEA|LINEAS ] ]
        [~~~~~~ {~~          } [ { identificador-3 }            ] ]
        [            { ABAJO}                                     ]
                        ~~~~
        [ TAMAÑO { entero-4
                                  } ]
        [ ~~~~ { identificador-
        4 } ]

[FIN DE LA PANTALLA]
  ~~~~~~~~~~~~

  dónde  La especificación de posición es
  { (posición-especificación-num, posición-especificación-num) }
  { (, número-especificación-posición)        }
  { (posición-especificación-num,)            }

  dónde  núm-especificación-posición      es
  { identificador-1 } [{ + } entero-1 ]
  { entero-2       } [{ - }                ]
```

Este formato de la declaración DISPLAY presenta datos en una pantalla formateada utilizando el formato de Microsoft de los compiladores v1 y v2 (MsDos).

7.8.13 DIVIDIR
7.8.13.1 DIVIDIR EN

Sintaxis DIVIDIR EN

```
DIVIDIR { literal-1        } EN { literal-2          } DANDO { identificador-3
~~~~~~ { identificador-1 } ~~~~ { identificador-2 } ~~~~~~

                [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] ] }...
                 ~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~          }
                                      { MÁS CERCANO-LEJOS-DE-CERO }
                                        { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                                      { MÁS CERCANO-PAR           }
                                       { ~~~~~~~~~~~~~            }
                                      { MÁS CERCANO A CERO        }
                                 { ~~~~~~~~~~~~~~~~~~~~~~~        }
                                      { PROHIBIDO                 }
                                       { ~~~~~~~~~~~              }
                                      { HACIA MAYOR               }
                                       { ~~~~~~~~~~~~~~~          }
                                      { HACIA MENOR               }
                                       { ~~~~~~~~~~~~~~           }
                                      { TRUNCAMIENTO              }
                                        ~~~~~~~~~~
    [ Identificador RESTO-4 ]
      ~~~~~~~~~~
    [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
          ~~~~ ~~~~~
    [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
      ~~~      ~~~~
~~~~~[ FIN DE LA
DIVISIÓN ]
   ~~~~~~~~~~
```

Para mayor claridad, se proporcionan los siguientes ejemplos para utilizar con los distintos tipos de la declaración DIVIDE cuando se utiliza BY, INTO y GIVING.

Operación DIVIDIR	Un	B	do	D
DIVIDIR A EN B	La	Si/La		
DIVIDIR A EN B DANDO C	Un	B	LICENCIADO EN LETRAS	
DIVIDIR A ENTRE B DANDO C	Un	B	A/B	
DIVIDIR A EN B DANDO A C EL RESTO D	Un	B	Entero(B/A)	Resto entero

Este formato de la declaración DIVIDE dividirá un valor numérico (especificado como un elemento de datos literal o numérico) en otro valor numérico (también especificado como un elemento de datos literal o numérico) y luego reemplazará el contenido de uno o más elementos de datos receptores con los resultados de esa división.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Tanto el identificador-1 como el identificador-2 deben ser elementos de datos numéricos sin editar y el literal-1 debe ser un literal numérico.
3. Se realizará una operación de división para cada identificador 2, por turno. Cada uno de los resultados de esas divisiones se guardará en el elemento de datos del identificador 2 correspondiente.
4. Si algún identificador-2 es un elemento de datos numérico entero, el resultado calculado cuando ese identificador-2 se divide por literal-1 o identificador-1 también será un entero; cualquier resto de esa división será descartado.
5. El REDONDEADO opcional (verSección 7.6.7 [REDONDEADA], página 257) cláusula disponible para cada uno
   *identificador-2*controlará cómo se guardarán los resultados no enteros.
6. Las cláusulas opcionales ON SIZE ERROR y NOT ON SIZE ERROR se pueden utilizar para detectar y reaccionar ante el fracaso o el éxito, respectivamente, de un intento de realizar un cálculo. En este caso, el fracaso se define como un truncamiento numérico causado por un identificador-2 con unnúmero insuficiente de posiciones de dígitos disponibles a la izquierda de cualquier punto decimal implícito, o un intento de dividir por cero. VerSección 7.6.6 [EN ERROR DE TAMAÑO + NO EN ERROR DE TAMAÑO], página 257, para obtener información adicional.

7.8.13.2 DIVIDIR EN DAR

Sintaxis de DIVIDIR EN DAR

```
DIVIDIR { literal-1        } EN { literal-2          } DANDO { identificador-3
~~~~~~ { identificador-1 } ~~~~ { identificador-2 } ~~~~~~

                [REDONDEADO [ EL MODO ES { LEJOS DE CERO              } ] ] }...
                 ~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~          }
                                      { MÁS CERCANO-LEJOS-DE-CERO }
                                        { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                                      { MÁS CERCANO-PAR           }
                                       { ~~~~~~~~~~~~~            }
                                      { MÁS CERCANO A CERO        }
                            { ~~~~~~~~~~~~~~~~~~~~~~~~            }
                                      { PROHIBIDO                 }
                                       { ~~~~~~~~~~~             }
                                      { HACIA MAYOR               }
                                       { ~~~~~~~~~~~~~~~~        }
                                      { HACIA MENOR               }
                                       { ~~~~~~~~~~~~~~~         }
                                      { TRUNCAMIENTO              }
                                        ~~~~~~~~~~

    [ Identificador RESTO-4 ]
      ~~~~~~~~~~

    [ EN CASO DE ERROR DE TAMAÑO declaración-imperativa-1 ]
          ~~~~ ~~~~~

    [ ERROR DE TAMAÑO NO FUNCIONA declaración imperativa 2 ]
      ~~~      ~~~~

~~~~~[ FIN DE LA
DIVISIÓN ]
  ~~~~~~~~~~
```

———

Este formato de la declaración DIVIDE dividirá un valor numérico (especificado como un elemento de datos literal o numérico) en otro valor numérico (también especificado como un elemento de datos literal o numérico) y luego reemplazará el contenido de uno o más elementos de datos receptores con los resultados de esa división.

1. Las palabras reservadas IS y ON son opcionales y pueden omitirse. La presencia o ausencia de estas palabras no tiene efecto sobre el programa.
2. Tanto el identificador-1 como el identificador-2 deben ser elementos de datos numéricos sin editar, mientras que el identificador-3 y el identificador-4 deben ser elementos de datos numéricos (editados o sin editar).
3. Tanto el literal-1 como el literal-2 deben ser literales numéricos.
4. Si la cláusula REMAINDER está codificada, es posible que solo se especifique un identificador-3.
5. Se calcula el resultado obtenido cuando el valor de literal-2 o identificador-2 se divide por el valor de literal-1 o identificador-1; este resultado luego se traslada a cada identificador-3, a su vez, aplicando las reglas definidas por ROUNDED (verSección 7.6.7 [REDONDEADA], página 257) cláusula (si la hay) para ese identificador-3 al movimiento.
6. Si se especifica una cláusula REMAINDER, el valor del único identificador-3 (como se indicó

anteriormente,Si se especifica REMAINDER, solo puede haber un único identificador (codificado en 3) en la declaración)

after it was assigned a value according to the previous rule will be multiplied by the value of literal-1 or identifier-1; that result is then subtracted from the value of literal-2 or identifier-2 and that result is the value which is moved to identifier-4.

7. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point, or an attempt to divide by zero. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.13.3 DIVIDE BY GIVING

DIVIDE BY GIVING Syntax

```
DIVIDE { literal-1         } BY { literal-2        } GIVING { identifier-3
~~~~~~ { identifier-1 } ~~ { identifier-2 } ~~~~~~

                [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                } ] ] }...
                  ~~~~~~~    ~~~~     { ~~~~~~~~~~~~~~~~         }
                                      { NEAREST-AWAY-FROM-ZERO }
                                        { ~~~~~~~~~~~~~~~~~~~~~~ }
                                      { NEAREST-EVEN              }
                                       { ~~~~~~~~~~~~~            }
                                      { NEAREST-TOWARD-ZERO       }
                            { ~~~~~~~~~~~~~~~~~~~~~~              }
                                      { PROHIBITED                }
                                       { ~~~~~~~~~~~             }
                                      { TOWARD-GREATER            }
                                       { ~~~~~~~~~~~~~~~         }
                                      { TOWARD-LESSER             }
                                       { ~~~~~~~~~~~~~~          }
                                      { TRUNCATION                }
                                        ~~~~~~~~~~
    [ REMAINDER identifier-4 ]
      ~~~~~~~~~~
    [ ON SIZE ERROR imperative-statement-1 ]
          ~~~~ ~~~~~
    [ NOT ON SIZE ERROR imperative-statement-2 ]
      ~~~      ~~~~ ~~~~~
 [ END-DIVIDE ]
   ~~~~~~~~~~
```

This format of the DIVIDE statement will divide one numeric value (specified as a literal or numeric data item) by another numeric value (also specified as a literal or numeric data item) and will then replace the contents of one or more receiving data items with the results of that division.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be numeric unedited data items while both identifier- 3 and identifier-4 must be numeric (edited or unedited) data items.
3. Both literal-1 and literal-2 must be numeric literals.
4. If the REMAINDER clause is coded, there may be only one identifier-3 specified.
5. The result obtained when the value of literal-1 or identifier-1 is divided by the value of literal-2 or identifier-2 is computed; this result is then moved into each identifier-3, in turn, applying the rules defined by the ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause (if any) for that identifier-3 to the move.
6. If a REMAINDER clause is specified, the value of the one and only identifier-3 (as stated earlier, if REMAINDER is specified there may only be a single identifier-3 coded on the statement)

after it was assigned a value according to the previous rule will be multiplied by the value of literal-2 or identifier-2; that result is then subtracted from the value of literal-1 or identifier-1 and that result is the value which is moved to identifier-4.

7. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point, or an attempt to divide by zero. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.14 ENTRY

ENTRY Syntax

```
ENTRY  [ {STDCALL          } ]  [ WITH {STDCALL} LINKAGE ]   literal-1
~~~~   [ {~~~~~~~          } ]  [ ~~~~ {~~~~~~~} ~~~~~~~ ]
       [ {STATIC           } ]  [      {C       }         ]
       [ {~~~~~~           } ]  [      {~       }         ]
       [ {C                } ]  [      {PASCAL }          ]
       [ {~                } ]  [      {~~~~~~ }          ]
       [ {EXTERN           } ]
       [ {~~~~~~           } ]
       [ {PASCAL           } ]
       [ {~~~~~~           } ]
       [ {mnemonic-name-1} ]

       [USING  {[BY{REFERENCE}] {[{          SIZE  IS  AUTO      }] literal-2    }}...]
       [~~~~~  {[   {~~~~~~~~~}] {[{          ~~~~      ~~~~      }] identifier-2}}    ]
       [       {[   { CONTENT  }] {[{          SIZE  IS  DEFAULT   }]              }}    ]
       [       {[   { ~~~~~~~  }] {[{          ~~~~      ~~~~~~~   }]              }}    ]
       [       {[   { VALUE    }] {[{          SIZE  IS  integer-1}]               }}    ]
       [       {[   { ~~~~~    }] {[{          ~~~~                }]              }}    ]
       [       {                  {[{UNSIGNED  SIZE  IS  AUTO      }]              }}    ]
       [       {                  {[{~~~~~~~~  ~~~~      ~~~~      }]              }}    ]
       [       {                  {[{UNSIGNED  SIZE  IS  integer-2}]               }}    ]
       [       {                  {[{~~~~~~~~  ~~~~      ~~~~      }]              }}    ]
       [       {                  {                                    OMITTED    }}    ]
       [       {                  {                                    ~~~~~~~    }}    ]
```

Format 2 (Special purpose and for GO TO) ENTRY

```
FOR GO TO literal-3
~~~~~ ~~~ ~~ ~~
```

———

The ENTRY statement is used to define an alternate entry-point into a subroutine, along with the arguments that subroutine will be expecting.

1. The reserved word BY is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. You may not use an ENTRY statement in a nested subprogram, nor may you use it in any form of user-defined function.
3. The USING clause defines the arguments the subroutine entry-point supports. This list of arguments must match up against the USING clause of any CALL statement that will be invoking the subroutine using this entry-point.
4. Each ENTRY-Argument specified on the ENTRY statement must be defined in the linkage section of the subroutine in which the ENTRY statement exists.
5. The literal-1 value will specify the entry-point name of the subroutine. It must be specified

exactly on CALL statements (with regard to the use of upper- and lower-case letters) as it is specified on the ENTRY statement.

6. The meaning of REFERENCE, CONTENT and VALUE are the same as the equivalent specifications on the CALL statement (see Section 7.8.5 [CALL], page 290). Whatever specification will be used for an argument on the CALL to this entry-point should match the specification used in the corresponding ENTRY-Argument. The same rules regarding the presence or absence of a BY clause on a CALL statement apply to the presence or absence of a BY clause on the corresponding argument of the ENTRY statement.
7. The GO TO with the ENTRY FOR is an GnuCOBOL special purpose extension for use with various GnuCobol tools.

7.8.15 EVALUATE

EVALUATE Syntax

```
   EVALUATE Selection-Subject-1 [ ALSO Selection-Subject-2 ]...
   ~~~~~~~~                     ~~~~
{ { WHEN Selection-Object-1 [ ALSO Selection-Object-2 ] }...
     ~~~~                     ~~~~
        [ imperative-
   statement-1 ] }... [ WHEN
   OTHER
     ~~~~ ~~~~~
        imperative-statement-other  ]

[ END-EVALUATE ]
   ~~~~~~~~~~~~
```

EVALUATE Selection Subject Syntax

```
{ TRUE          }
{ ~~~~          }
{ FALSE         }
{ ~~~~~         }
{ expression-   }
{ 1 identifier- }
{ 1             }
  literal-1
```

EVALUATE Selection Object Syntax

```
{ ANY                                                   }
{ ~~~                                                   }
{ TRUE                                                  }
{ ~~~~                                                  }
{ FALSE                                                 }
{ ~~~~~                                                 }
{ partial-expression-1                                  }
{                                                       }
{ { expression-2 } [ THRU|THROUGH { expression-3 } ] }
{ { literal-2    }                 { literal-3    }    }
```

The EVALUATE statement provides a means of defining processing that should take place under any number of mutually-exclusive conditions.

1. The reserved words THRU and THROUGH are interchangeable.
2. There must be at least one WHEN clause (in addition to any WHEN OTHER clause) specified on any EVALUATE statement.

3. There must be at least one Selection-Subject specified on the EVALUATE statement. Any number of additional Selection-Subject clauses may be specified, using the ALSO reserved word to separate each from the prior.
4. Each WHEN clause (other than the WHEN OTHER clause, if any) must have the same number of Selection-Object clauses as there are Selection-Subject clauses.
5. When using THRU, the values on both sides of the THRU must be the same class (both numeric, both alphanumeric, etc.).
6. A partial-expression is one of the following:
   A. A Class Condition without a leading identifier-1 (see Section 2.1.27.2 [Class Conditions], page 45).
   B. A Sign Condition without a leading identifier-1 (see Section 2.1.27.3 [Sign Conditions], page 47).
   C. A Relation Condition with nothing to the left of the relational operator (see Section 2.1.27.5 [Relation Conditions], page 49).
7. At execution time, each Selection-Subject on the EVALUATE statement will have its value matched against that of the corresponding Selection-Object on a WHEN clause, in turn, until:
   A. A WHEN clause has each of its Selection-Object(s) successfully matched by the corresponding Selection-Subject; this will be referred to as the 'Selected WHEN clause'.
   B. The complete list of WHEN clauses (except for the WHEN OTHER clause, if any) has been exhausted. In this case, there is no 'Selected WHEN Clause'.
8. If a 'Selected WHEN Clause' was identified:
   A. The imperative-statement-1 (see [Imperative Statement], page 758) immediately following the 'Selected WHEN Clause' will be executed. If the 'Selected WHEN Clause' is lacking an imperative-statement-1, the first imperative-statement-1 found after any following WHEN clause will be executed.
   B. Once the imperative-statement-1 has been executed, or no imperative-statement-1 was found anywhere after the 'Selected WHEN Clause', control will proceed to the state- ment following the END-EVALUATE or, if there is no END-EVALUATE, the first statement that follows the next period. If, however, the imperative-statement-1 included a GO TO statement, and that GO TO was executed, then control will transfer to the procedure named on the GO TO instead.
9. If no 'Selected WHEN Clause' was identified:
   A. The WHEN OTHER clause's imperative-statement-other will be executed, if such a clause was coded.
   B. Control will then proceed to the statement following the END-EVALUATE or the first statement that follows the next period if there is no END-EVALUATE. If,however, the imperative-statement-other included a GO TO statement, and that GO TO was executed, then control will transfer to the procedure named on the GO TO instead.
10. In order for a Selection-Subject to match the corresponding Selection-Object on a WHEN clause, at least one of the following must be true:
    A. The Selection-Object is ANY
    B. The implied Relation Condition Selection-Subject = Selection Object is TRUE — See Section 2.1.27.5 [Relation Conditions], page 49, for the rules on how the comparison will be made.
    C. The value of the Selection-Subject falls within the range of values specified by the THRU clause of the Selection-Object

D. If the Selection-Object is a partial-expression, then the conditional expression that would be represented by coding Selection-Subject Selection-Object evaluates to TRUE

11. Here is a sample program that illustrates the EVALUATE statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOEVALUATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Test-Digit                    PIC 9(1).
    88 Digit-Is-Odd VALUE 1, 3, 5, 7, 9.
    88 Digit-Is-Prime VALUE 1, 3, 5, 7.
PROCEDURE DIVISION.
P1. PERFORM UNTIL EXIT
    DISPLAY "Enter a digit (0 Quits): " WITH
        NO ADVANCING
    ACCEPT Test-Digit
    IF  Test-Digit  =  0
        EXIT PERFORM
    END-IF
    EVALUATE Digit-Is-Odd ALSO Digit-Is-Prime
    WHEN TRUE ALSO FALSE
        DISPLAY Test-Digit " is ODD" WITH
            NO ADVANCING
    WHEN TRUE ALSO TRUE
        DISPLAY Test-Digit " is PRIME" WITH NO
            ADVANCING
    WHEN FALSE ALSO ANY
        DISPLAY Test-Digit " is EVEN" WITH
            NO ADVANCING
    END-EVALUATE
    EVALUATE Test-Digit
    WHEN < 5
        DISPLAY " and it's small too"
    WHEN < 8
        DISPLAY " and it's medium too"
    WHEN OTHER
        DISPLAY " and it's large too" END-
    EVALUATE
END-PERFORM
DISPLAY "Bye!"
STOP RUN
.
```

Console output when run (user input follows the colons on the prompts for input):

```
Enter a digit (0 Quits): 1
1  is PRIME and it's small
too Enter a digit (0
Quits): 2
2  is EVEN and it's small
too Enter a digit (0
Quits): 3
3  is PRIME and it's small
```

```
too Enter a digit (0
Quits): 4
```

```
4 is EVEN and it's small
too Enter a digit (0
Quits): 5
5 is PRIME and it's
medium too Enter a digit
(0 Quits): 6
6 is EVEN and it's
medium too Enter a
digit (0 Quits): 7
7 is PRIME and it's
medium too Enter a digit
(0 Quits): 8
8 is EVEN and it's large
too Enter a digit (0
Quits): 9
9 is ODD and it's large
too Enter a digit (0
Quits): 0 Bye!
```

7.8.16 EXAMINE

EXAMINE Syntax

```
EXAMINE identifier-1
~~~~~~~

    {              {  ALL            }                                    }
    {              {  ~~~            }                                    }
    { TALLYING{    LEADING       } literal-1 [REPLACING BY literal-2] }
    { ~~~~~~~~~ {  ~~~~~~~       }              ~~~~~~~~~ ~~              }
    {              {  UNTIL FIRST    }                                    }
    {              {  ~~~~~ ~~~~~    }                                    }
    {                                                                     }
    {              {  ALL            }                                    }
    {              {  ~~~            }                                    }
    { REPLACING {  LEADING       } literal-3 BY literal-4                }
    { ~~~~~~~~~ {  ~~~~~~~       }              ~~                       }
    {              { [UNTIL] FIRST }                                      }
    {              {  ~~~~~  ~~~~~ }                                      }
```

The EXAMINE statement is the pre runner for INSPECT which should be used over the pre 1970 Cobol standard EXAMINE, and it is used to count the number of times a specified character appears in a data item and/or to replace a character with another character.

1. This statement is only available subject to specific dialects being set when running the GnuCOBOL compiler.
2. In all cases, the description of identifier must be such that its usage is display (explicitly or implicitly).
3. When identifier represents a non numeric data item, examination starts at the leftmost character and proceeds to the right. Each character in the data item is examined in turn. For purposes of the EXAMINE statement, external floating point items are treated as non numeric data items.
4. When identifier represents a numeric data item, this data item must consist of numeric char- acters, and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn.
5. If the letter 'S' is used in the PICTURE of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.
6. Each literal must consist of a single character belonging to a class consistent with that of the identifier; in addition, each literal may be any figurative constant except ALL. If identifier is numeric, each literal must be an unsigned integer or the figurative constant ZERO (ZEROES, ZEROS).
7. When Format 1 is used, an integral count is created which replaces the value of a special register called TALLY, whose implicit description is that of an unsigned integer of five digits.
8. When the ALL option is used, this count represents the number of occurrences of literal-1.

9. When the LEADING option is used, this count represents the number of occurrences of literal-1 prior to encountering a character other than literal-1.
10. When the UNTIL FIRST option is used, this count represents all characters encountered before the first occurrence of literal-1.
11. Whether Format 2 is used, or the REPLACING option of Format 1, the replacement rules are the same. They are as follows:
12. When the ALL option is used, literal-2 is substituted for each occurrence of literal-1.
13. When the LEADING option is used, the substitution of literal-2 for each occurrence of literal-1 terminates as soon as a character other than literal-1 or the right hand boundary of the data item is encountered.
14. When the UNTIL FIRST option is used, the substitution of literal-2 terminates as soon as literal-1 or the right hand boundary of the data item is encountered.

7.8.17 EXHIBIT

EXHIBIT Syntax

```
EXHIBIT [CHANGED][NAMED] [position-spec] [ERASE] {identifier-1 | literal-1} ...
~~~~~~~  ~~~~~~~  ~~~~~                          ~~~~~
          [UPON mnemonic-name-1]
           ~~~~

where position-spec is

   {(position-spec-num,  position-spec-num)}
   {(, position-spec-num)                  }
   {(position-spec-num, )                  }

where position-spec-num is

   {identifier-2} [{+} integer-2]
   {integer-1     }  [{-}        ]
```

The EXHIBIT statement causes an (optionally conditional) display of the literals, and/or identifiers (optionally preceded by the identifier name) specified in the statement for the purposes of debugging.

1. EXHIBIT is only present to ease migrations, this is an archaic language element in Gnu- COBOL (but without the archaic message warning because there is no explicit dialect configuration for that other than the reserved word). Depending on the -std used it will either compile or not. As the standard says about archaic language elements:

   it should not be used in new compilation groups because better programming practices exist

   The use of DISPLAY is more portable and allows for the same feature-set (with the excep- tion of CHANGED which GnuCOBOL may not support for a long time and with the need of a literal for NAMED). This statement can be removed from any later version.
2. The reserved words NAMED, CHANGED are optional and may be omitted. The presence or absence of these words has no effect upon the program.
3. Each identifier specified in the EXHIBIT statement can be any class of data. TALLY and RETURN-CODE are the only special registers that can be used as identifiers.
4. Literals and identifiers displayed by the EXHIBIT statement are separated by a space on the displayed line.
5. Each literal can be any figurative constant other than ALL.
6. If the literal is numeric, it must be an unsigned integer.
7. Each execution of an EXHIBIT NAMED statement displays each identifier or literal specified, with each identifier (including any qualifiers and subscripts) followed by a "=" (equal sign) and its current value. They all appear on a single line on the order in which they appear in the statement.
8. Each execution of an EXHIBIT CHANGED NAMED statement displays each identifier or literal specified, with each identifier (including any qualifiers and subscripts) followed by a "=" (equal sign) and its current value. They all appear on a single line on the order in which they

appear in the statement. However, the display for each identifier (name and

value) is conditional on the value of that identifier having changed since the last execution of the current EXHIBIT statement. If one or more of the identifier values have not changed, neither the name nor the value is printed for those identifiers. If none of the identifier values has changed, and no literals are specified, no display takes place (display of a blank line is suppressed).

9. Each execution of an EXHIBIT CHANGED statement displays the current value of each identifier or literal specified. They all appear on a single line on the order in which they appear in the statement. However, the value display for each identifier is conditional on the value of that identifier having changed since the last execution of the current EXHIBIT statement. If one or more of the identifier values have not changed, the value for those identifiers are not printed and spaces are inserted instead. If none of the identifier values has changed, and no literals are specified, a blank line is displayed (display of a blank line is not suppressed).
10. Each execution of an EXHIBIT statement with neither the CHANGED nor the NAMED option displays each identifier or literal specified. They all appear on a single line in the order in which they appear in the statement.
11. An EXHIBIT statement is the same as an EXHIBIT NAMED statement.
12. ERASE is a display attribute which got into GnuCOBOL when MS-COBOL support was increased, it is not yet implemented.
13. The CHANGED clause is not yet implemented but recognised by GnuCOBOL.
14. The UPON mnemonic-name-1 clause is not yet implemented but recognised by GnuCOBOL.

7.8.18 EXIT

EXIT Syntax

```
EXIT [ { PROGRAM          }    [ { RETURNING } ] { identifier-1 } ]
~~~~                           [ { GIVING   } ] { literal-1    } ]
       { FUNCTION         } ]
       { ~~~~~~~~         } ]
       { PERFORM [ CYCLE ] } ]
       { ~~~~~~~ ~~~~~    } ]
       { SECTION          } ]
       { ~~~~~~~          } ]
       { PARAGRAPH        } ]
         ~~~~~~~~~
```

The EXIT statement is a multi-purpose statement; it may provide a common end point for a series of procedures, exit an in-line PERFORM, paragraph or section or it may mark the logical end of a subprogram, returning control back to the calling program.

1. The EXIT PROGRAM statement is not legal anywhere within a user-defined function.
2. The EXIT FUNCTION statement cannot be used anywhere within a subroutine.
3. Neither EXIT PROGRAM nor EXIT FUNCTION may be used within a USE GLOBAL routine in DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250).
4. The following points describe the EXIT statement with none of the optional clauses:
   A. When this form of an EXIT statement is used, it must be the only statement in the procedure (paragraph or section) in which it occurs. This is not enforced in GnuCOBOL.
   B. This usage of the EXIT statement simply provides a common "GO TO" end point for a series of procedures, as may be seen in the following example:

```
01  Switches.
    05 Input-File-Switch PIC X(1).
       88 EOF-On-Input-File VALUE ''Y'' FALSE ''N''.
---
    SET EOF-On-Input-File TO FALSE.
    PERFORM 100-Process-A-Transaction THRU 199-Exit UNTIL
        EOF-On-Input-File.
---
100-Process-A-Transaction.
    READ Input-File AT END
        SET EOF-On-Input-File TO TRUE
        GO TO 199-Exit
    END-READ.
    IF Input-Rec of Input-File = SPACES
        GO TO 199-Exit      *> IGNORE BLANK RECORDS!
    END-IF.
    <<process the record just read>>
199-Exit.
    EXIT.
```

   C. In this case, the EXIT statement takes no other run-time action.

5. The following points apply to the EXIT PARAGRAPH and EXIT SECTION statements:
   A. If an EXIT PARAGRAPH statement or EXIT SECTION statement resides in a paragraph within the scope of a procedural PERFORM (see Section 7.8.33.1 [Procedural PERFORM], page 360), control will be returned back to the PERFORM for evaluation of any TIMES, VARYING and/or UNTIL clauses.
   B. If an EXIT PARAGRAPH statement or EXIT SECTION statement resides outside the scope of a procedural PERFORM, control simply transfers to the first executable statement in the next paragraph (EXIT PARAGRAPH) or section (EXIT SECTION).
   C. The following shows how the previous example could have been coded without a GO TO by utilizing an EXIT PARAGRAPH statement.

```
01  Switches.
    05 Input-File-Switch PIC X(1).
       88 EOF-On-Input-File VALUE ''Y'' FALSE ''N''.
---
    SET EOF-On-Input-File TO FALSE.
    PERFORM 100-Process-A-Transaction
        UNTIL EOF-On-Input-File.
---
100-Process-A-Transaction.
    READ Input-File AT END
        SET EOF-On-Input-File TO TRUE EXIT
        PARAGRAPH
    END-READ.
    IF Input-Rec of Input-File = SPACES
        EXIT PARAGRAPH *> IGNORE BLANK RECORDS!
    END-IF.
    <<process the record just read>>
```

6. The following points apply to the EXIT PERFORM and EXIT PERFORM CYCLE statements:
   A. The EXIT PERFORM and EXIT PERFORM CYCLE statements are intended to be used in con- junction with an in-line PERFORM statement (see Section 7.8.33.2 [Inline PERFORM], page 362).
   B. An EXIT PERFORM CYCLE statement will terminate the current iteration of the in-line PERFORM, giving control to any TIMES, VARYING and/or UNTIL clauses for them to determine if another cycle needs to be performed.
   C. An EXIT PERFORM statement will terminate the in-line PERFORM outright, transferring control to the first statement following the END-PERFORM (if there is one) or to the next sentence following the PERFORM if there is no END-PERFORM.
   D. This last example shows the final modification to the previous examples by using an in-line PERFORM along with EXIT PERFORM and EXIT PERFORM CYCLE statements:

```
PERFORM  F O R E V E R
    READ Input-File AT END EXIT
        PERFORM
    END-READ
    IF Input-Rec of Input-File = SPACES
        EXIT PERFORM CYCLE *> IGNORE BLANK RECORDS!
    END-IF
    <<process the record just read>>
END PERFORM
```

7. The following points apply to the EXIT PROGRAM and EXIT FUNCTION statements:
   A. The EXIT PROGRAM and EXIT FUNCTION statements terminate the execution of a subroutine (i.e. a program that has been CALLed by another) or user-defined function, respectively, returning control back to the calling program.
   B. An EXIT PROGRAM statement returns control back to the statement following the CALL (see Section 7.8.5 [CALL], page 290) of the subprogram. An EXIT FUNCTION statement returns control back to the processing of the statement in the calling program that invoked the user-defined function.
   C. For EXIT PROGRAM statement usage of RETURNING statement or GIVING statement will provide value defined by indentifer-1 or literal-1 back to the calling routine.
   D. If executed by a main program, neither the EXIT PROGRAM nor EXIT FUNCTION statements will take any action.
   E. The COBOL2002 standard has made a common extension to the COBOL language — the GOBACK statement (see Section 7.8.21 [GOBACK], page 334) — a standard language element; the GOBACK statement should be strongly considered as the preferred alternative to both EXIT PROGRAM and EXIT FUNCTION for new subprograms.

7.8.19 FREE

FREE Syntax

```
FREE { [ ADDRESS OF ] identifier-1 }...
~~~~       ~~~~~~~
```

———————————————————————————————————————

The FREE statement releases memory previously allocated to the program by the ALLOCATE statement (see Section 7.8.3 [ALLOCATE], page 287).

1. The ADDRESS OF clause is optional and may be omitted. The presence or absence of this clause has no effect upon the program.
2. *identifier-1* must have a USAGE (see Section 6.9.59 [USAGE], page 228) of POINTER, or it must be an 01-level data item with the BASED (see Section 6.9.8 [BASED], page 161) attribute.
3. If identifier-1 is a USAGE POINTER data item and it contains a valid address, the FREE statement will release the memory block the pointer references. In addition, any BASED data items that the pointer was used to provide an address for will become un-based and therefore unusable. If identifier-1 did not contain a valid address, no action will be taken.
4. If identifier-1 is a BASED data item and that data item is currently based (meaning it currently has memory allocated to it), its memory is released and identifier-1 will become un-based and therefore unusable. If identifier-1 was not based, no action will be taken.

7.8.20 GENERATE

GENERATE Syntax

```
GENERATE { report-name-1 }
~~~~~~~~ { identifier-1    }
```

The GENERATE statement presents data to a report.

1. The following points apply when identifier-1 is specified:
   A. *identifier-1* must be the name of a DETAIL (see Section 9.1 [RWCS Lexicon], page 651) report group.
   B. If necessary, identifier-1 may be qualified with a report name.
   C. The file in whose FD a REPORT clause exists for the report in which identifier-1 is a detail group must be opened for OUTPUT or EXTEND at the time the GENERATE is executed. See Section 7.8.32 [OPEN], page 358, for information on file open modes.
   D. The report in which identifier-1 is a DETAIL group must have been successfully initiated via the INITIATE statement (see Section 7.8.25 [INITIATE], page 343) and not yet terminated via the TERMINATE statement (see Section 7.8.50 [TERMINATE], page 403) at the time the GENERATE is executed.
   E. If at least one GENERATE statement of this form is executed against a report, the report is said to be a detail report. If no GENERATE statements of this form are executed against a report, the report is said to be a summary report.
2. The following points apply when report-name-1 is specified:
   A. *report-name-1* must be the name of a report having an RD defined for it in the report section.
   B. There must be at least one CONTROL (see Section 9.1 [RWCS Lexicon], page 651) group defined for report-name-1.
   C. There cannot be more than one DETAIL group defined for report-name-1.
   D. The file in whose FD a REPORT report-name-1 clause exists must be open for OUTPUT or EXTEND at the time the GENERATE is executed.
   E. *report-name-1* must have been successfully initiated (via INITIATE report-name-1) and not yet terminated (via TERMINATE) at the time the GENERATE is executed. See Section 7.8.32 [OPEN], page 358, for information on file open modes.
   F. The DETAIL group which is defined for report-name-1 will be processed but will not actually be presented to any report page. This will allow summary processing to take place. If all GENERATE statements are of this form, the report is said to be a summary report. If at least one GENERATE identifier-1 is executed, the report is considered to be a detail report.
3. When the first GENERATE statement for a report is executed, the contents of all control fields are saved so they may be referenced during the processing of subsequent GENERATE statements.
4. When, during the processing of a subsequent GENERATE, it is determined that a control field has changed value (ie. a control break has occurred), the appropriate control footing and control heading processing will take place and a snapshot of the current values of all control fields will again be saved.

7.8.21 GOBACK

GOBACK Syntax

```
GOBACK [ { RETURNING|GIVING { literal-1          }    ]
~~~~~~   { ~~~~~~~~~~ ~~~~~~ { identifier-1 }
```

The GOBACK statement is used to logically terminate an executing program.

1. If executed within a subprogram (i.e. a subroutine or user-defined function), GOBACK behaves like an EXIT PROGRAM or EXIT FUNCTION statement, respectively.
2. If executed within a main program, GOBACK will act as a STOP RUN statement.
3. The optional RETURNING clause provides the opportunity to return a numeric value to the operating system (technically, an exit status The manner in which the exit status value is interrogated by the operating system varies. Windows can use %ERRORLEVEL% to query the exit status while Unix shells such as sh, bash and ksh can query the exit status as $?. Other Unix shells may have different ways to access the exit status.

7.8.22 GO TO

7.8.22.1 Simple GO TO

Simple GO TO Syntax

```
GO TO procedure-name-1
~~
GO TO ENTRY literal-3
~~    ~~~~~
```

This form of the GO TO statement unconditionally transfers control in a program to the first executable statement within the specified procedure-name-1.

1. The reserved word TO is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. If this format of the GO TO statement appears in a consecutive sequence of imperative statements (see [Imperative Statement], page 758) within a sentence, it must be the final state- ment in the sentence.
3. If a GO TO is executed within the scope of. . .
   A. ...an in-line PERFORM (see Section 7.8.33 [PERFORM], page 360), the PERFORM is terminated as control of execution transfers to procedure-name-1.
   B. ...a procedural PERFORM (see Section 7.8.33 [PERFORM], page 360), and procedure-*name-1* lies outside the scope of that PERFORM, the PERFORM is terminated as control of execution transfers to procedure-name-1.
   C. ...a MERGE statement (see Section 7.8.29 [MERGE], page 351) OUTPUT PROCEDURE or within the scope of either an INPUT PROCEDURE or OUTPUT PROCEDURE of a SORT statement (see Section 7.8.44.1 [File-Based SORT], page 388), and procedure-name-1 lies outside the scope of that procedure, the SORT or MERGE operation is terminated as control of execution transfers to procedure-name-1. Any sorted or merged data accu- mulated to that point is lost.
4. A GO TO ENTRY is an GnuCOBOL special purpose extension for use with various GnuCobol tools and is not part of any current ISO standard. See also ENTRY.
5. The GO TO ENTRY format has to be used together with ENTRY FOR GO TO.

7.8.22.2 GO TO DEPENDING ON

GO TO DEPENDING ON Syntax

```
GO TO {procedure-name-1} ...
~~
                                    DEPENDING ON identifier-1
                                    ~~~~~~~~~~
GO TO ENTRY {literal-3} ...
~~      ~~~~~
                                    DEPENDING ON identifier-1
                                    ~~~~~~~~~~
```

This form of the GO TO statement will transfer control to any one of a number of specified procedure names depending on the numeric value of the identifier specified on the statement.

1. The reserved word TO is optional and may be omitted. The presence or absence of this word has no effect upon the program but does help with read-ability.
2. The PICTURE (see Section 6.9.37 [PICTURE], page 195) and/or USAGE (see Section 6.9.59 [USAGE], page 228) of the specified identifier-1 must be such as to define it as a numeric, unedited, preferably unsigned integer data item.
3. If the value of identifier-1 has the value 1, control will be transferred to the 1st specified procedure name. If the value is 2, control will transfer to the 2nd procedure name, and so on.
4. The GO TO ENTRY ... DEPENDING ON format has to be used together with ENTRY FOR GO TO. If control of execution is transferred to a procedure named on the statement, and the GO TO is executed within the scope of. . .
   A. ...an in-line PERFORM (see Section 7.8.33 [PERFORM], page 360), the PERFORM is terminated as control of execution transfers to the procedure named on the statement.
   B. ...a procedural PERFORM (see Section 7.8.33 [PERFORM], page 360), and procedure-*name-1* lies outside the scope of that PERFORM, the PERFORM is terminated as control of execution transfers to the procedure named on the statement.
   C. ...a MERGE statement (see Section 7.8.29 [MERGE], page 351) OUTPUT PROCEDURE or within the scope of either an INPUT PROCEDURE or OUTPUT PROCEDURE of a SORT statement (see Section 7.8.44.1 [File-Based SORT], page 388), and procedure-name-1 lies outside the scope of that procedure, the SORT or MERGE operation is terminated as control of execution transfers to the procedure named on the statement. Any sorted or merged data accumulated to that point is lost.
5. If the value of identifier-1 is less than 1 or exceeds the total number of procedure names specified on the statement, control will simply fall through into the next statement following the GO TO.
6. The following example shows how GO TO ... DEPENDING ON may be used in a real appli- cation situation, and compares it against an alternative — EVALUATE (see Section 7.8.15 [EVALUATE], page 321).

GO TO DEPENDING ON

```
    GO TO
      ACCT-TYPE-1
      ACCT-TYPE-2
      ACCT-TYPE-3
    DEPENDING ON Acct-Type.
    <<< Invalid Acct Type >>>
    GO TO All-Done.
Acct-Type-1.
    <<< Handle Acct Type 1 >>>
    GO TO All-Done.
Acct-Type-2.
    <<< Handle Acct Type 2 >>>
    GO TO All-Done.
Acct-Type-3.
    <<< Handle Acct Type 3 >>>
All-Done.
```

EVALUATE

```
EVALUATE Acct-Type
WHEN 1
    <<< Handle Acct Type 1 >>>
WHEN 2
    <<< Handle Acct Type 2 >>>
WHEN 3
    <<< Handle Acct Type 3 >>>
WHEN OTHER
    <<< Invalid Acct Type >>>
END-EVALUATE.
```

7. Current programming philosophy would prefer the use of the EVALUATE statement to that of this form of the GO TO statement.

7.8.23 IF

IF Syntax

```
IF conditional-expression
~~
THEN { imperative-statement-1 }
     { NEXT SENTENCE          }
       ~~~~ ~~~~~~~~
[ ELSE { imperative-statement-2 } ]
  ~~~~ { NEXT SENTENCE              }
       ~~~~ ~~~~~~~~
[ END-IF ]
  ~~~~~~
```

The IF statement is used to conditionally execute an imperative statement (see [Imperative Statement], page 758) or to select one of two different imperative statements to execute based upon the TRUE/FALSE value of a conditional expression.

1. The reserved word THEN is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. You cannot use both NEXT SENTENCE and the END-IF scope terminator in the same IF statement.
3. If conditional-expression evaluates to TRUE, imperative-statement-1 will be executed regard- less of whether or not an ELSE clause is present. Once imperative-statement-1 has been executed, control falls into the first statement following the END-IF or to the first statement of the next sentence if there is no END-IF clause.
4. If the optional ELSE clause is present and conditional-expression evaluates to false, then (and only then) imperative-statement-2 will be executed. Once imperative-statement-2 has been executed, control falls into the first statement following the END-IF or to the first statement of the next sentence if there is no END-IF clause.
5. The clause NEXT SENTENCE may be substituted for either imperative-statement, but not both. If control reaches a NEXT SENTENCE clause due to the truth or falsehood of conditional-expression, control will be transferred to the first statement of the next sentence found in the program (the first statement after the next period).

   NEXT SENTENCE was needed for COBOL programs that were coded according to pre-1985 standards that wish to nest one IF statement inside another. See Section 2.1.29 [Use of VERB/END-VERB Constructs], page 54, for an explanation of why NEXT SENTENCE was necessary.

   Programs coded for 1985 (and beyond) standards don't need it, instead using the ex- plicit scope-terminator END-IF to inform the compiler where imperative-statement-2 (or imperative-statement-1 if there is no ELSE clause coded) ends. New GnuCOBOL programs should be coded to use the END-IF scope terminator for IF statements. See Section 2.1.29 [Use of VERB/END-VERB Constructs], page 54, for additional information.

7.8.24 INITIALIZE

INITIALIZE Syntax

```
INITIALIZE|INITIALISE  identifier-1...
~~~~~~~~~~ ~~~~~~~~~~ [
    WITH FILLER ]
           ~~~~~~
  [ { category-name-1 } TO VALUE ]
      { ALL           }     ~~~~~
        ~~~
    [ THEN REPLACING { category-name-2 DATA BY
           ~~~~~~~~~                   ~~
          [ LENGTH OF ] { literal-1  } }...     ]
           ~~~~~~~      {  identifier-1  }

    [ THEN TO DEFAULT ]
             ~~~~~~~
```

The INITIALIZE statement initializes each identifier-1 with certain specific values, depending upon the options specified.

1. The reserved words DATA, OF, THEN, TO and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The reserved words INITIALIZE and INITIALISE are interchangeable.
3. The WITH FILLER, REPLACING and DEFAULT clauses are meaningful only if identifier-1 is a group item. They are accepted if it's an elementary item, but will serve no purpose. The VALUE clause is meaningful in both cases.
4. A category-name-1 and/or category-name-2 may be any of the following:

   ALPHABETIC
   : The PICTURE (see Section 6.9.37 [PICTURE], page 195) of the data item only contains A symbols.

   ALPHANUMERIC
   : The PICTURE of the data item contains only X or a combination of A and 9 symbols.

   ALPHANUMERIC-EDITED
   : The PICTURE of the data item contains only X or a combination of A and 9 symbols plus at least one B, 0 (zero) or / symbol.

   NUMERIC
   : The data item is one that is described with a picture less USAGE (see Section 6.9.59 [USAGE], page 228) or has a PICTURE composed of nothing but P, 9, S and V symbols.

   NUMERIC-EDITED
   : The PICTURE of the data item contains nothing but the symbol 9 and at least one of the editing symbols $, +, -, CR, DB, ., ,, * or Z.

   NATIONAL
   : The data item is one containing nothing but the N symbol.

NATIONAL-EDITED
The data item contains nothing but N, B, / and 0 symbols.

5. From the sequence of identifier-1 data items specified on the INITIALIZE statement, a list of initialized fields referred to as the field list in the remainder of this section, will include:
   A. Every identifier-1 that is an elementary item, including any that may have the REDEFINES (see Section 6.9.41 [REDEFINES], page 205) clause in their descriptions.
   B. Every non-FILLER elementary item subordinate to identifier-1, provided that elementary item neither contains a REDEFINES clause in its definition nor belongs to a group item subordinate to identifier-1 which contains a REDEFINES clause in its definition.
   C. If the optional WITH FILLER clause is included on the INITIALIZE statement, then every FILLER elementary item subordinate to each identifier-1 will be included as well, provided that elementary item neither contains a REDEFINES clause in its definition nor belongs to a group item subordinate to identifier-1 which contains a REDEFINES clause in its definition..
6. Once a field list has been determined, each item in that field list will be initialized as if an individual MOVE (see Section 7.8.30 [MOVE], page 354) statement to that effect had been coded. The rules for initialization are as follows:
7. If no VALUE, REPLACING or DEFAULT clauses are coded, each member of the field list will be initialized as if the figurative constant ZERO (if the field list item is numeric or numeric-edited) or SPACES (otherwise) were being moved to it.
8. If a VALUE clause is specified on the INITIALIZE statement, each qualifying member of the field list having a compile-time VALUE (see Section 6.9.61 [VALUE], page 239) specified in its definition will be initialized to that value. Field list members with VALUE clauses will qualify for this treatment as follows:
   A. If the ALL keyword was specified on the VALUE clause, all members of the field list with VALUE clauses will qualify.
   B. If category-name-1 is specified instead of ALL, only those members of the field list with VALUE clauses that also meet the criteria set down for the specified category-name (see the list above) will qualify.
   C. If you need to apply VALUE initialization to multiple category-name-1 values, you will need to use multiple INITIALIZE statements.
9. If a REPLACING clause is specified on the INITIALIZE statement, each qualifying member of the field list that was not already initialized by a VALUE clause, if any, will be initialized to the specified literal-1 or identifier-1 value.

   Only those as-yet uninitialized list members meeting the criteria set forth for the specified *category-name-2* will qualify for this initialization.

   If you need to apply REPLACING initialization to multiple category-name-2 values, you may repeat the syntax after the reserved word REPLACING, as necessary.
10. If a DEFAULT clause is specified, any remaining uninitialized members of the field list will be initialized according to the default for their class (numeric and numeric-edited are initialized to ZERO, all others are initialized to SPACES).
11. The following example may help your understanding of how the INITIALIZE statement works. The sample code makes use of the COBDUMP program to dump the storage that is (or is not) being initialized. See Section "COBDUMP" in GnuCOBOL Sample Programs, for a source and cross-reference listing of the COBDUMP program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   DemoInitialize.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Item-1.
    05 I1-A VALUE ALL '*'.
       10 FILLER                    PIC X(1).
       10 I1-A-1                    PIC 9(1) VALUE 9.
    05 I1-B                         USAGE BINARY-CHAR.
    05 I1-C                         PIC A(1) VALUE 'C'.
    05 I1-D                         PIC X/X VALUE 'ZZ'.
    05 I1-E                         OCCURS 2 TIMES PIC   9.
PROCEDURE DIVISION.
000-Main.
    DISPLAY "MOVE HIGH-VALUES TO Item-1"
        PERFORM 100-Init-Item-1
        CALL "COBDUMP" USING Item-1
        DISPLAY " "

    DISPLAY "INITIALIZE Item-1"
        INITIALIZE Item-1
        CALL "COBDUMP" USING Item-1
        PERFORM 100-Init-Item-1 DISPLAY "
        "

    DISPLAY "INITIALIZE Item-1 WITH FILLER"
        MOVE HIGH-VALUES TO Item-1
        INITIALIZE Item-1 WITH FILLER
        CALL "COBDUMP" USING Item-1
        PERFORM 100-Init-Item-1 DISPLAY "
        "

    DISPLAY "INITIALIZE Item-1 ALL TO VALUE"
        MOVE HIGH-VALUES TO Item-1
        INITIALIZE Item-1 ALPHANUMERIC TO VALUE
        CALL "COBDUMP" USING Item-1
        PERFORM 100-Init-Item-1
        DISPLAY " "

    DISPLAY "INITIALIZE Item-1 REPLACING NUMERIC BY 1"
        MOVE HIGH-VALUES TO Item-1
        INITIALIZE Item-1 REPLACING NUMERIC BY 1
        CALL "COBDUMP" USING Item-1
        PERFORM 100-Init-Item-1
        DISPLAY " "

    STOP RUN
    .

100-Init-Item-1.
    MOVE HIGH-VALUES TO Item-1
    .
```

When executed, this program produces the following output:

```
MOVE HIGH-VALUES TO Item-1
<-Addr-> Byte <---------------- Hexadecimal ----------------> <---- Char---------------- >
======== ==== ================================================ ================
00404058       1 FF FF FF FF FF FF FF FF FF                                  .........

INITIALIZE Item-1
<-Addr-> Byte <---------------- Hexadecimal ----------------> <---- Char---------------- >
======== ==== ================================================ ================
00404058       1 FF 30 00 20 20 2F 20 30 30                                  .0. / 00

INITIALIZE Item-1 WITH FILLER
<-Addr-> Byte <---------------- Hexadecimal ----------------> <---- Char---------------- >
======== ==== ================================================ ================
00404058       1 20 30 00 20 20 2F 20 30 30                                   0. / 00

INITIALIZE Item-1 ALL TO VALUE
<-Addr-> Byte <---------------- Hexadecimal ----------------> <---- Char---------------- >
======== ==== ================================================ ================
00404058       1 2A 2A FF 43 5A 5A 20 FF FF                                  **.CZZ ..

INITIALIZE Item-1 REPLACING NUMERIC BY 1
<-Addr-> Byte <---------------- Hexadecimal ----------------> <---- Char---------------- >
======== ==== ================================================ ================
00404058       1 FF 31 01 FF FF FF FF 31 31                                  .1.....11
```

7.8.25 INITIATE

INITIATE Syntax

INITIATE report-name-1

———————————————————————————————————————

The INITIATE statement starts Report-Writer Control System (RWCS) processing for a report.

1. Each report-name-1 must be the name of a report having an RD (see Section 6.6 [REPORT SECTION], page 135) defined for it.
2. The file in whose FD (see Section 6.2.1 [File/Sort-Description], page 123) a REPORT report-name-1 clause exists must be open for OUTPUT or EXTEND at the time the INITIATE statement is executed. See Section 7.8.32 [OPEN], page 358, for more information on file open modes.
3. The INITIATE statement will initialize all of the following for each report named on the statement:
   - All sum counters, if any, will be set to 0
   - The report's LINE-COUNTER special register (see Section 7.7 [Special Registers], page 260) will be set to 0
   - The report's PAGE-COUNTER special register will be set to 1
4. No report content will actually presented to the report file as a result of a success- ful INITIATE statement — that will not occur until the first GENERATE statement (see Section 7.8.20 [GENERATE], page 333) is executed.

7.8.26 INSPECT

INSPECT Syntax

```
INSPECT { literal-1                  }
~~~~~~~ { identifier-1               }
        { function-reference-1 }

  [ TALLYING { identifier-2 FOR { ALL|LEADING|TRAILING { literal-2     } }
   ~~~~~~~~                     ~~~ { ~~~ ~~~~~~~ ~~~~~~~~ { identifier-3 } }
                                      { CHARACTERS                           }
                                        ~~~~~~~~~~
              [ | { AFTER|BEFORE } INITIAL { literal-3    } | ] }...      ]
                |    ~~~~~ ~~~~~~            {  identifier-4  }  |

  [ REPLACING { { { ALL|FIRST|LEADING|TRAILING { literal-4         } }
   ~~~~~~~~~~  { { ~~~ ~~~~~ ~~~~~~~ ~~~~~~~~ { identifier-5 } }
                 { CHARACTERS                                    }
                 { ~~~~~~~~~~~                                   }

                      BY { [ ALL ] literal-5 }
                   ~~ {     ~~~               }
                       { identifier-6         }

              [ | { AFTER|BEFORE } INITIAL { literal-6    } | ] }...      ]
                |    ~~~~~ ~~~~~~            {  identifier-7  }  |

  [ CONVERTING { { literal-7   } TO { literal-8       }
   ~~~~~~~~~~  { identifier-8 } ~~ { identifier-9 }

              [ | { AFTER|BEFORE } INITIAL { literal-9     } | ] ]
                |    ~~~~~ ~~~~~~            {  identifier-10  }  |
```

The INSPECT statement is used to perform various counting and/or data-alteration operations against strings.

1. The reserved word INITIAL is optional and may be omitted. The presence or absence of this words has no effect upon the program.
2. If a CONVERTING clause is specified, neither the TALLYING nor REPLACING clauses may be used.
3. If either the TALLYING or REPLACING clauses are specified, the CONVERTING clause cannot be used.
4. If both the TALLYING and REPLACING clauses are specified, they must be specified in the order shown.
5. All literals and identifiers must be explicitly or implicitly defined as alphanumeric or alphabetic.
6. If function-reference-1 is specified, it must be an invocation of an intrinsic function that returns a string result. Additionally, only the TALLYING clause may be specified.
7. If literal-1 is specified, only the TALLYING clause may be specified.

8. Whichever is specified — literal-1, identifier-1 or function-reference-1 — that item will be referred to in the discussions that follows as the 'inspect subject '.
9. The three optional clauses control the operation of this statement as follows:
   A. The CONVERTING clause replaces one or more individual characters found in the inspect subject with a different character in much the same manner as is possible with the TRANSFORM statement (see Section 7.8.51 [TRANSFORM], page 404).
   B. The REPLACING clause replaces one or more sub strings located in the inspect subject with a different, but equally-sized replacement sub string. If you need to replace a sub string with another of a different length, consider using either the SUBSTITUTE intrin- sic function (see Section 8.1.90 [SUBSTITUTE], page 522) or the SUBSTITUTE-CASE intrinsic function (see Section 8.1.91 [SUBSTITUTE-CASE], page 523).
   C. The TALLYING clause counts the number of occurrences of one or more strings of characters in the inspect subject.
10. The optional INITIAL clauses may be used to limit the range of characters in the inspect subject that the CONVERTING, REPLACING or TALLYING instruction in which they occur will apply. We call this the 'target range' of the inspect subject. The target range is defined as follows:
    A. If there is no INITIAL clause specified, the target range is the entire inspect subject.
    B. Either a BEFORE phrase, an AFTER phrase or both may be specified. They may be specified in any order.
    C. The starting point of the target range will be the first character following the sub string identified by the AFTER specification. The ending point will be the last character immediately preceding the sub string identified by the BEFORE specification.
    D. If no AFTER is specified, the first character position of the target range will be character position #1 of the inspect subject.
    E. If no BEFORE is specified, the last character position of the target range will be the last character position of the inspect subject.
11. The following points apply to the use of the TALLYING clause:
    A. While there will typically be only be a single set of counting instructions on an INSPECT:

       ```
       INSPECT  Character-String TALLYING
            C-ABC FOR ALL "ABC"
       ```

       There could be multiple counting instructions specified:

       ```
       INSPECT  Character-String TALLYING
            C-ABC FOR ALL "ABC"
                          C-BCDE FOR ALL "BCDE"
       ```

       When there are multiple instructions, the one specified first will take priority over the one specified second, (and so forth) as the INSPECT proceeds forward through the inspect subject, character-by-character.

       With the above example, if the inspect subject were --ABCDEF BCDEF--, the final result of the counting would be that C-ABC would be incremented by 1 while C-BCDE would be incremented only once; although the human eye clearly sees two 'BCDE' sequences, the INSPECT ... TALLYING would only "see" the second — the first would have been processed by the first (higher-priority) counting instruction.
    B. Each set of counting instructions contains the following information:
       a. A target range, specified by the presence of an AFTER INITIAL and/or BEFORE INITIAL clause; the rules for specifying target ranges were covered previously.

b. A Target Sub string — this is a sequence of characters to be located somewhere in the inspect subject and counted. Target sub strings may be defined as a literal value (figurative constants are allowed) or by the contents of an identifier. If the target sub string is specified as a figurative constant, it will be assumed to have a length of one ('1') character. The keywords before the literal or identifier control how many target sub strings could be identified from that replacement instruction, as follows:

   ALL — identifies every possible target sub string that occurs within the target range. There are three occurrences of ALL 'XX' found in aXXabbXXccXXdd.

   LEADING — identifies only an occurrence of the target sub string found either at the first character position of the target range or immediately following a previously-found occurrence. There are no occurrences of LEADING 'XX' found in aXXabbXXccXXdd, but there is one occurrence of LEADING 'a' (the first character).

   TRAILING — identifies only an occurrence of the target sub string found either at the very end of the target range or toward the end, followed by nothing but other occurrences. There are no occurrences of LEADING 'XX' found in aXXabbXXccXXdd, but there are two occurrences of TRAILING 'd'.

   The CHARACTERS option will match any one single character, regardless of what that character is.

C. *identifier-2* will be incremented by 1 each time the target sub string is found within the target range of the inspect subject. The INSPECT statement will not zero-out identifier-2 at the start of execution of the INSPECT — it is the programmer's responsibility to ensure that all identifier-2 data items are properly initialized to the desired starting values prior to execution of the INSPECT.

12. The following points apply to the use of the REPLACING clause:

A. While there will typically be only be a single set of replacement instructions on an INSPECT:

```
INSPECT Character-String REPLACING
    ALL "ABC" BY "DEF"
```

There could be multiple replacement instructions:

```
INSPECT Character-String REPLACING
    ALL "ABC" BY "DEF"
                ALL "BCDE" BY "WXYZ"
```

When there are multiple replacement instructions, the one specified first will take priority over the one specified second, (and so forth) as the INSPECT proceeds forward through the inspect subject, character-by-character.

With the above example, if the inspect subject were --ABCDEF BCDEF--, the final result of the replacement would be --DEFDEF WXYZF--.

B. Each set of replacement instructions contains the following information:

a. A target range, specified by the presence of an AFTER INITIAL and/or BEFORE INITIAL clause; the rules for specifying target ranges were covered previously.

b. A Target Sub string — this is a sequence of characters to be located somewhere in the inspect subject and subsequently replaced with a new value. Target sub strings, which are specified before the BY keyword, may be defined as a literal value (figurative constants are allowed) or by the contents of an identifier. If the target sub string is specified as a figurative constant, it will be assumed to have a length of one ('1') character. The keywords before the literal or identifier control

how many target sub strings could be identified from that replacement instruction, as follows:

ALL — identifies every possible target sub string that occurs within the target range. There are three occurrences of ALL 'XX' found in aXXabbXXccXXdd.

FIRST — the first occurrence of the target sub string found within the target range. The FIRST 'XX' found in aXXabbXXccXXdd would be the one found between the 'a' and 'b' characters.

LEADING — an occurrence of the target sub string found either at the first character position of the target range or immediately following a previously-found occurrence. There are no occurrences of LEADING 'XX' found in aXXabbXXccXXdd, but there is one occurrence of LEADING 'a' (the first character).

TRAILING — an occurrence of the target sub string found either at the very end of the target range or toward the end, followed by nothing but other occurrences. There are no occurrences of LEADING 'XX' found in aXXabbXXccXXdd, but there are two occurrences of TRAILING 'd'.

The CHARACTERS option will match any one single character. When you use this option, the replacement sub string (see the next item) must be exactly one char- acter in length.

c. A Replacement Sub string — this is the sequence of characters that should replace the target sub string. Replacement sub strings are specified after the BY keyword. They too may be specified as a literal, either with or without an ALL prefix (again, figurative constants are allowed) or the value of an identifier. If a figurative constant is coded, the ALL keyword will be assumed, even if it wasn't specified. Literals without ALL will either be truncated or padded with spaces on the right to match the length of the target sub string. Literals with ALL or figurative constants will be repeated as necessary to match the length of the target sub string. Identifiers specified as replacement sub strings must be defined with a length equal to that of the target sub string.

13. When both REPLACING and TALLYING are specified:
    A. The INSPECT statement will make a single pass through the sequence of characters comprising the inspect subject. As the pointer to the current inspect target character reaches a point where it falls within the explicit or implicit target ranges specified on the operational instructions of the two clauses, the actions specified by those instructions will become eligible to be taken. As the character pointer reaches a point where it falls past the end of target ranges, the instructions belonging to those target ranges will become disabled.
    B. At any point in time, there may well be multiple REPLACING and/or TALLYING operational instructions active. Only one of the TALLYING and one of the REPLACING instructions (if any) can be executed for any one character pointer position. In each case, it will be the first of the instructions in each category that produces a match with its target string specification.
    C. When both a TALLYING and a REPLACING instruction have been selected for execution, the TALLYING instruction will be executed first. This guarantees that TALLYING will compute occurrences based upon the initial value of the inspect subject before any replacements occur.
14. The following points apply to the use of the CONVERTING clause:
    A. A CONVERTING clause performs a series of single-character substitutions against a data item in much the same manner as is possible with the TRANSFORM statement (see Section 7.8.51 [TRANSFORM], page 404).

B. Unlike the TALLYING and REPLACING clauses, both of which may have multiple opera- tions specified, there may be only one CONVERTING operation per INSPECT.
C. If the length of literal-7 or identifier-8 (the "from" string) exceeds the length of literal-8 or identifier-9 (the "to" string), then the "to" string will be assumed to be padded to the right with enough spaces to make it the same length as the "from" string.
D. If the length of the "from" string is less than the length of the "to" string, then the "to" string will be truncated to the length of the "from" string.
E. Each character, in turn, within the "from" string will be searched for in the target range of the inspect subject. Each located occurrence will be replaced by the corresponding character of the "to" string.

7.8.27 JSON GENERATE

JSON GENERATE Syntax

```
JSON GENERATE identier-1 FROM identier-2
~~~~ ~~~~~~~~~                ~~~~
    [COUNT IN identier-3]
     ~~~~~
    NAME OF {identier-4 IS literal-1} .          .    .]
    ~~~~
    SUPPRESS {identier-5} .      .    .]
    ~~~~~~~~
    [ON EXCEPTION imperative-statement-1
     ~~ ~~~~~~~~~~
     NOT ON EXCEPTION imperative-statement-2]
     ~~~       ~~~~~~~~~~
    END-JSON
    ~~~~~~~~
```

The JSON GENERATE statement converts a JSON record based on the structure and content of the input data item.

7.8.28 JSON PARSE

JSON PARSE Syntax

```
JSON PARSE identier-1 INTO identier-2
~~~~ ~~~~~            ~~~~
    [WITH DETAIL]
          ~~~~~~
    [NAME OF {identier-3 IS literal-1} .          .    .]
     ~~~~
    [SUPPRESS {identier-4} .      .    .]
     ~~~~~~~~
    [ON EXCEPTION imperative-statement-1
          ~~~~~~~~~~
     NOT ON EXCEPTION imperative-statement-2 ]
     ~~~       ~~~~~~~~~~
END-JSON
~~~~~~~~
```

———————————————————————————————————————

The JSON PARSE statement starts an event-driven JSON parser, the processing procedure is performed as each component of the JSON document is identied in order.

7.8.29 MERGE

MERGE Syntax

```
MERGE sort-file-1
~~~~~
    { ON { ASCENDING                          } KEY identifier-1... }...
      { ~~~~~~~~~~        }
           { DESCENDING }
             ~~~~~~~~~~
    [ WITH DUPLICATES IN ORDER ]
             ~~~~~~~~~~
    [ COLLATING SEQUENCE IS alphabet-name-1 ]
      ~~~~~~~~~~
      USING file-name-1 file-name-2...
      ~~~~~
    { OUTPUT PROCEDURE IS procedure-name-1          }
    { ~~~~~~ ~~~~~~~~~~                             }
    {            [  THRU|THROUGH procedure-name-2 ] }
    {               ~~~~ ~~~~~~~                    }
    { GIVING file-name-3...                         }
    { ~~~~~~                                        }
```

The DUPLICATES clause is syntactically recognized but is otherwise non-functional.

The MERGE statement merges the contents of two or more files that have each been pre-sorted on a set of specified identical keys.

1. The reserved words IN, IS, KEY, ON, ORDER, SEQUENCE and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The reserved words THRU and THROUGH are interchangeable.
3. GnuCOBOL always behaves as if the WITH DUPLICATES IN ORDER clause is specified, even if it isn't.

   While any COBOL implementation's sort or merge facilities guarantee that records with duplicate key values will be in proper sequence with regard to other records with different key values, they generally make no promises as to the resulting relative sequence of records having duplicate key values with one another.

   Some COBOL implementations provide this optional clause to force their sort and merge facilities to retain duplicate key-value records in their original input sequence, relative to one another.
4. The sort-file-1 named on the MERGE statement must be defined using a sort description (SD (see Section 6.2.1 [File/Sort-Description], page 123)). This file is referred to in the remainder of this discussion as the merge work file.
5. Each file-name-1, file-name-2 and file-name-3 (if specified) must reference ORGANIZATION LINE SEQUENTIAL (see Section 5.2.1.2 [ORGANIZATION LINE SEQUENTIAL], page 110) or ORGANIZATION SEQUENTIAL (see Section 5.2.1.1 [ORGANIZATION SEQUENTIAL], page 109) files. These files must be defined using a file description (FD (see Section 6.2.1 [File/Sort-Description], page 123)).
6. The identifier-1 . . . field(s) must be defined as field(s) within a record of sort-file-1.

7. The record descriptions of file-name-1, file-name-2, file-name-3 (if any) and sort-file-1 are assumed to be identical in layout and size. While the actual data names used for fields in these files' records may differ, the structure of records, PICTURE (see Section 6.9.37 [PICTURE], page 195) of fields, USAGE (see Section 6.9.59 [USAGE], page 228) of fields, size of fields and location of fields within the records should match field-by-field across all files, at least as far as the KEY fields are concerned.
8. A common programming technique when using the MERGE statement is to define the records of all files involved as simple elementary items of the form 01 record-name PIC X(n). where n is the record size. The only file where records are actually described in detail would then be sort-file-1.
9. The following rules apply to the files named on the USING clause:
   A. None of them may be open at the time the MERGE is executed.
   B. Each of those files is assumed to be already sorted according to the specifications set forth on the MERGE statement's KEY clause.
   C. No two of those files may be referenced on a SAME RECORD AREA (see Section 5.2.2 [SAME RECORD AREA], page 116), SAME SORT AREA or SAME SORT-MERGE AREA statement.
10. The merging process is as follows:
   A. As the MERGE statement begins execution, the first record in each of the USING files is read automatically.
   B. As the MERGE statement executes, the current record from each of the USING files is examined and compared to each other according to the rules set forth by the KEY clause and the alphabet (see Section 5.1.3.1 [Alphabet-Name-Clause], page 94) specified on the COLLATING SEQUENCE clause. The record that should be next in sequence will be written to the merge work file and the USING file from which that record came will be read so that its next record is available. As end-of-file conditions are reached on USING files, those files will be excluded from further processing — processing continues with the remaining files until all the contents of all of them have been exhausted.
   C. After the merge work file has been populated, the merged data will be written to each file-name-3 if the GIVING clause was specified, or will be processed by utilizing an OUTPUT PROCEDURE.
   D. When GIVING is specified, none of the file-name-3 files can be open at the time the MERGE statement is executed.
   E. When an output procedure is used, the procedure(s) specified on the OUTPUT PROCEDURE clause will be invoked as if by a procedural PERFORM (see Section 7.8.33.1 [Procedural PERFORM], page 360) statement with no VARYING, TIMES or UNTIL options specified. Merged records may be read from the merge work file — one at a time — within the output procedure using the RETURN (see Section 7.8.38 [RETURN], page 372) statement.

      A GO TO statement (see Section 7.8.22 [GO TO], page 335) that transfers control out of the output procedure will terminate the MERGE statement but allows the program to continue executing from the point where the GO TO statement transferred control to. Once an output procedure has been "aborted" using a GO TO it cannot be resumed, and the contents of the merge work file are lost. You may, however, re-execute the MERGE statement itself. Using a GO TO statement to prematurely terminate a merge, or re-starting a previously-cancelled merge is not considered good programming style and should be avoided.

      An output procedure should be terminated in the same way a procedural PERFORM statement would be. Usually, this action will be taken once the RETURN statement

indicates that all records in the merge work file have been processed, but termination

could occur at any time — via an EXIT statement (see Section 7.8.18 [EXIT], page 329) — if required.

Neither a file-based SORT statement (see Section 7.8.44.1 [File-Based SORT], page 388) nor another MERGE statement may be executed within the scope of the procedures comprising the output procedure unless those statements utilize a different sort or merge work file.

F. Once the output procedure terminates, or the last file-name-3 file has been populated with merged data, the output phase — and the MERGE statement itself — is complete.

7.8.30 MOVE

7.8.30.1 Simple MOVE

Simple MOVE Syntax

```
MOVE { literal-1          } TO identifier-2...
~~~~ { identifier-1 } ~~
```

The Simple MOVE statement moves a specific value to one or more receiving data items.

1. The MOVE statement will replace the contents of one or more receiving data items (identifier- 2) with a new value — the one specified by literal-1 or identifier-1.
2. Only numeric data can be moved to a numeric or numeric-edited identifier-2. A MOVE involving numeric data will perform any necessary format conversions that might be necessary due to differing USAGE (see Section 6.9.59 [USAGE], page 228) specifications.
3. The contents of the identifier-1 data item will not be changed, unless that same data item appears as an identifier-2. Note that such situations will cause a warning message to be issued by the compiler, if warning messages are enabled.

7.8.30.2 MOVE CORRESPONDING

MOVE CORRESPONDING Syntax

```
MOVE CORRESPONDING identifier-1 TO identifier-2...
~~~~ ~~~~                          ~~
```

———

The MOVE CORRESPONDING statement similarly-named items from one group item to another.

1. The reserved word CORRESPONDING may be abbreviated as CORR.
2. Both identifier-1 and identifier-2 must be group items.
3. See Section 7.6.2 [CORRESPONDING], page 254, for a discussion of how corresponding matches between two group items are established.
4. When corresponding matches are established, the effect of a MOVE CORRESPONDING on those matches will be as if a series of individual MOVEs were done — one for each match.

7.8.31 MULTIPLY

7.8.31.1 MULTIPLY BY

MULTIPLY BY Syntax

```
MULTIPLY { literal-1         } BY { identifier-2
~~~~~~~~ { identifier-1 } ~~

      [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                } ] ] }...
         ~~~~~~~    ~~~~      { ~~~~~~~~~~~~~~~~          }
                              { NEAREST-AWAY-FROM-ZERO }
                                { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { NEAREST-EVEN              }
                              { ~~~~~~~~~~~~~             }
                              { NEAREST-TOWARD-ZERO       }
                        { ~~~~~~~~~~~~~~~~~~~~~~~         }
                              { PROHIBITED                }
                              { ~~~~~~~~~~~               }
                              { TOWARD-GREATER            }
                              { ~~~~~~~~~~~~~~~           }
                              { TOWARD-LESSER             }
                              { ~~~~~~~~~~~~~~            }
                              { TRUNCATION                }
                                ~~~~~~~~~~

    [ ON SIZE ERROR imperative-statement-1 ]
          ~~~~ ~~~~~
    [ NOT ON SIZE ERROR imperative-statement-2 ]
       ~~~      ~~~~ ~~~~~
[ END-MULTIPLY ]
   ~~~~~~~~~~~~
```

The MULTIPLY BY statement computes the product of one or more data items (identifier-2) and either a numeric literal or another data item.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be numeric un-edited data items; literal-1 must be a numeric literal.
3. The product of identifier-1 or literal-1 and each identifier-2, in turn, will be computed and moved to each of the identifier-2 data items, replacing the prior contents.
4. The value of identifier-1 is not altered, unless that same data item appears as an identifier-2.
5. The optional ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause available to each *identifier-2* will control how non-integer results will be saved.
6. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.31.2 MULTIPLY GIVING

MULTIPLY GIVING Syntax

```
MULTIPLY { literal-1        } BY { literal-2        } GIVING { identifier-3
~~~~~~~~ { identifier-1 } ~~ { identifier-2 } ~~~~~~

      [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                } ] ] }...
         ~~~~~~~    ~~~~     { ~~~~~~~~~~~~~~~~          }
                             { NEAREST-AWAY-FROM-ZERO }
                               { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                             { NEAREST-EVEN              }
                             { ~~~~~~~~~~~~~~            }
                             { NEAREST-TOWARD-ZERO       }
                       { ~~~~~~~~~~~~~~~~~~~~~~~~        }
                             { PROHIBITED                }
                             { ~~~~~~~~~~~               }
                             { TOWARD-GREATER            }
                             { ~~~~~~~~~~~~~~~~          }
                             { TOWARD-LESSER             }
                             { ~~~~~~~~~~~~~~~           }
                             { TRUNCATION                }
                               ~~~~~~~~~~
    [ ON SIZE ERROR imperative-statement-1 ]
          ~~~~ ~~~~~
    [ NOT ON SIZE ERROR imperative-statement-2 ]
      ~~~      ~~~~ ~~~~~
[ END-MULTIPLY ]
   ~~~~~~~~~~~~
```

The MULTIPLY GIVING statement computes the product of two literals and/or data items and saves that result in one or more other data items.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be numeric un-edited data items; literal-1 and literal-2 must be numeric literals.
3. The product of identifier-1 or literal-1 and identifier-2 or literal-2 will be computed and moved to each of the identifier-3 data items, replacing their old contents.
4. Neither the value of identifier-1 nor identifier-2 will be altered, unless either appears as an *identifier-3*.
5. The optional ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause available to each *identifier-2* will control how non-integer results will be saved.
6. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.32 OPEN

OPEN Syntax

```
OPEN {[ EXCLUSIVE ] { INPUT       } [sharing-mode] file-name-1 [open-options]
}...
~~~~                { ~~~~~  }
                    { OUTPUT }
                    { ~~~~~~ }
                    { I-O    }
                    { ~~~    }
                    { EXTEND }
                      ~~~~~~
```

where "sharing-mode" is:

```
[                   { ALL OTHER } ]
[                   { ~~~       } ]
[ SHARING WITH { NO
OTHER               } ]
[ ~~~~~~~           { ~~        } ]
[                   { READ ONLY } ]
[                   { ~~~~ ~~~~
```

}] where "open-options" is:

```
[ [ WITH ] { LOCK   } ]
[ [ FOR  ] { ~~~~   } ]
[                     ]
[              {ALL     } ]
[              {~~~     } ]
[              {READERS } ] [ WITH NO REWIND ]
[              {~~~~~~~ } ] [          ~~ ~~~~~~ ]
[ ALLOWING {UPDATERS } ] [ REVERSED            ]
[ ~~~~~~~~~ {~~~~~~~~                          }
] [ ~~~~~~~~~          ] [
               {WRITERS  } ]
[              {~~~~~~~~ } ]
[              {NO OTHERS} ]
[              {~~       } ]
```

The NO REWIND, and REVERSED clauses are syntactically recognized but are otherwise non-functional.

———————————————————————————————————————

The OPEN statement makes one or more files described in your program available for use.

1. The reserved words OTHER and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The SHARING and WITH LOCK clauses may not both be specified in the same OPEN statement.
3. Any file defined in a GnuCOBOL program must be successfully opened before it or any of its record descriptions may be referenced on:

   A CLOSE statement (see Section 7.8.7 [CLOSE], page 295)

A DELETE statement (see Section 7.8.11 [DELETE], page 301)

A READ statement (see Section 7.8.34 [READ], page 365)

A REWRITE statement (see Section 7.8.39 [REWRITE], page 373) A

START statement (see Section 7.8.45 [START], page 393)

An UNLOCK statement (see Section 7.8.52 [UNLOCK], page 405) A

WRITE statement (see Section 7.8.54 [WRITE], page 410)

4. Any attempt to open a file that is already open will fail with a file status of 41 (see [File Status Codes], page 105).
5. Any open failure (including status 41) may be trapped using DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250) or an error procedure established using the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine or even just checking the status field defined. It is up to the programmer to check for bad statuses and respond accordingly such as issue a CLOSE before dealing with the problem.
6. The INPUT, OUTPUT, I-O and EXTEND open modes inform GnuCOBOL of the manner in which you wish to use the file, as follows:

   INPUT You may only read the existing contents of the file — only the CLOSE, READ, START and UNLOCK statements will be allowed. This enforcement takes place at execution time, not compilation time.

   OUTPUT You may only write new content (which will completely replace any previous file contents) to the file — only the CLOSE, UNLOCK and WRITE statements will be allowed. This enforcement takes place at execution time, not compilation time.

   I-O You may perform any operation you wish against the file — all file I/O statements will be allowed.

   EXTEND You may only write new content (which will be appended after the previously existing file contents) to the file — only the CLOSE, UNLOCK and WRITE statements will be allowed. This enforcement takes place at execution time, not compilation time. You cannot extend an empty file; this will not generate a runtime error, but no output will appear in the file.
7. The SHARING clause informs the GnuCOBOL file runtime modules how you are willing to co-exist with any other GnuCOBOL programs that may attempt to open the same file after your program does. See Section 2.1.30.1 [File Sharing], page 56, for an explanation of the SHARING clause.
8. The WITH LOCK option will be functional only if your GnuCOBOL build can support it. GnuCOBOL built for MinGW or native Windows will not, because the Unix fcntl primitive doesn't exist in those environments. GnuCOBOL built for Cygwin or Unix will.

7.8.33 PERFORM

7.8.33.1 Procedural PERFORM

Procedural PERFORM Syntax

```
PERFORM procedure-name-1 [ THRU|THROUGH procedure-name-2 ]
~~~~~~~                  ~~~~ ~~~~~~~~
    [ { [ WITH TEST { BEFORE } ] { VARYING-Clause              } } ]
      {           ~~~~ { ~~~~~~ }     { UNTIL conditional-expression-1 } }
      {                { AFTER }       ~~~~~                            }
      {                  ~~~~~                                          }
      { UNTIL EXIT|FOREVER                                              }
      { ~~~~~ ~~~~ ~~~~~~~~                                             }
      { { literal-1       } TIMES                                       }
      { { identifier-1 } ~~~~~                                          }
```

This format of the PERFORM statement is used to transfer control to one or more procedures, which will return control back when complete. Execution of the procedure(s) can be done a single time, multiple times, repeatedly until a condition becomes TRUE or forever (with some way of breaking out of the control of the PERFORM or of halting program execution within the procedure(s)).

1. The reserved word WITH is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. The reserved words THRU and THROUGH are interchangeable.
3. The reserved word and phrase FOREVER and UNTIL EXIT are interchangeable.
4. Both procedure-name-1 and procedure-name-2 must be procedure division sections or para- graphs defined in the same program as the PERFORM statement. If procedure-name-2 is specified, it must follow procedure-name-1 in the program's source code.
5. The perform scope is defined as being the statements within procedure-name-1, the state- ments within procedure-name-2 and all statements in all procedures defined between them.
6. *literal-1* must be a numeric literal or a reference to a function that returns a numeric value. The value must be an integer greater than zero.
7. *identifier-1* must be an elementary un-edited numeric data item with an integer value greater than zero.
8. Without the UNTIL, UNTIL EXIT, TIMES, VARYING-Clause (see Section 7.8.33.3 [VARY- ING], page 363) or FOREVER clauses, the code within the perform scope will be executed once, after which control will return to the statement following the PERFORM.
9. The FOREVER option will repeatedly execute the code within the perform scope with no conditions defined for termination of the repetition — it will be up to the programmer to in- clude an EXIT SECTION statement (see Section 7.8.18 [EXIT], page 329) or EXIT PARAGRAPH statement within the procedure(s) being performed that will break out of the loop.
10. The TIMES option will repeat the execution of the code within the perform scope a fixed number of times. When the PERFORM statement begins execution, an internal repeat counter (not accessible to the programmer) will be set to the value of literal-1 or the value within identifier-1.

If the counter has a value greater than zero, the statement(s) within the PERFORM scope will be executed, after which the counter will be decremented by 1 with each repetition. Once that counter reaches zero, repetition will cease and control will fall into the next statement following the PERFORM.

If the identifier-1 option was used, altering the value of that data item within the perform scope will not affect the repetition count.

11. The UNTIL conditional-expression-1 option will repeat the code within the perform scope until the specified conditional expression evaluates to a TRUE value.
12. The optional WITH TEST clause will control whether UNTIL testing occurs BEFORE the statements within the perform scope are executed on each iteration (creating the possibility — if conditional-expression-1 is initially TRUE — that the statements within the perform scope will never be executed) or AFTER (guaranteeing the statements within the perform scope will be executed at least once).

    The default, if this clause is absent, is WITH TEST BEFORE. This

    clause may not be coded when the TIMES clause is used.
13. The optional VARYING-Clause is a mechanism that creates an advanced loop-management mechanism complete with one or more numeric data items being automatically incremented (or decremented) on each loop iteration as well as the termination control of an UNTIL clause. See Section 7.8.33.3 [VARYING], page 363, for the details.

7.8.33.2 Inline PERFORM

Inline PERFORM Syntax

```
PERFORM
~~~~~~~
  [ { [ WITH TEST { BEFORE } ] { VARYING-Clause              } } ]
    {           ~~~~ { ~~~~~~ }      { UNTIL conditional-expression-1 } }
    {                { AFTER }        ~~~~~                              }
    {                  ~~~~~                                            }
    { UNTIL EXIT|FOREVER                                                }
    { ~~~~~ ~~~~ ~~~~~~~~                                               }
    { { literal-1       } TIMES                                         }
    { { identifier-1 } ~~~~~                                            }

    imperative-statement-1

END-PERFORM
~~~~~~~~~~~
```

This format of the PERFORM statement is identical in operation to the procedural PERFORM, except for the fact that the statement(s) comprising the perform scope (imperative-statement-1) (see [Imperative Statement], page 758) are now specified in-line with the PERFORM code rather than in procedures located elsewhere within the program.

7.8.33.3 VARYING

VARYING Syntax

```
VARYING identifier-2 FROM { literal-2            } [ BY { literal-3       } ]
~~~~~~~                   ~~~~ { identifier-3 }       ~~ { identifier-4 } [
          UNTIL conditional-expression-1 ]
            ~~~~~
[ AFTER identifier-5 FROM { literal-4   } [ BY { literal-5       } ]
  ~~~~~                   ~~~~ { identifier-6 }       ~~ { identifier-7 } [
          UNTIL conditional-expression-2 ] ]...
            ~~~~~
```

The VARYING clause, available on both formats of the PERFORM statement, is a looping mechanism that allows for the specification of one or more numeric data items that will be initialized to a programmer-specified value and automatically incremented by another programmer-specified value after each loop iteration.

1. All identifiers used in a VARYING-Clause must be elementary, un-edited numeric data items. All literals must be numeric literals.
2. The following points describe the sequence of events that take place as a result of the VARYING portion of the clause:
   A. When the PERFORM begins execution, the FROM value will be moved to identifier.
   B. If the PERFORM specifies or implies WITH TEST BEFORE, conditional-expression-1 will be evaluated and processing of the PERFORM will halt if the expression evaluates to TRUE. If WITH TEST BEFORE was not specified or implied, or if the conditional expression evaluated to FALSE, processing proceeds with step C.
   C. The statements within the perform scope will be executed. If a GO TO executed within the perform scope transfers control to a point outside the perform scope, processing of the PERFORM will halt.
   D. When the statements within the perform scope terminate the loop iteration, by one of:
      - allowing the flow of execution to attempt to fall past the last statement in the perform scope
      - executing an EXIT PERFORM CYCLE statement (see Section 7.8.18 [EXIT], page 329)
      - executing an EXIT PARAGRAPH statement or EXIT SECTION statement when there is only one paragraph (or section) in the perform scope (this option only applies to a procedural PERFORM)

      If WITH TEST AFTER was specified, control will return back to the PERFORM, where conditional-expression-1 will be evaluated, and processing of the PERFORM will halt if the expression evaluates to TRUE. If WITH TEST AFTER was not specified, or if the conditional expression evaluated to FALSE, processing continues with the next step.
   E. The BY value, if any, will be added to identifier-2. If no BY is specified, it will be treated as if BY 1 had been specified.
   F. Return to step C.
3. Most VARYING-Clauses have no AFTER specified. Those that do, however, are establishing a loop-within-a-loop situation where the process described above in steps ('A') through ('F') will take place from the AFTER, and those six processing steps actually replace step C of the VARYING. This "nesting" process can continue indefinitely, with each additional AFTER.

An example should really help you see this at work. Observe the following code which defines a two-dimensional (3 row by 4 column) table and a pair of numeric data items to be used to subscript references to each element of the table:

```
01  PERFORM-DEMO.
    05 PD-ROW          OCCURS 3 TIMES.
       10 PD-COL       OCCURS 4 TIMES
          15 PD        PIC X(1).
01  PD-Col-No          PIC 9 COMP.
01  PD-Row-No          PIC 9 COMP.
```

Let's say the 3x4 "grid" defined by the above structure has these values:

```
A B C D E
F G H I J
K L
```

This code will display ABCDEFGHIJKL on the console output window:

```
PERFORM WITH TEST AFTER
        VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No = 3 AFTER
           PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No = 4
     DISPLAY PD (PD-Row-No, PD-Col-No) WITH NO ADVANCING
END-PERFORM
```

While this code will display AEIBFJCGKDHL on the console output window:

```
PERFORM WITH TEST AFTER
        VARYING PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No = 4 AFTER
           PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No = 3
     DISPLAY PD (PD-Row-No, PD-Col-No) WITH NO ADVANCING
END-PERFORM
```

While we're looking at sample code, this code displays ABCEFG:

```
PERFORM
        VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No = 3
           AFTER PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No = 4
     DISPLAY PD (PD-Row-No, PD-Col-No) WITH NO ADVANCING
END-PERFORM
```

By removing the WITH TEST clause, the statement is now assuming WITH TEST BEFORE. Since testing now happens before the DISPLAY statement gets executed, when PD-Row-No is 3 and PD-Col-No is 4 the DISPLAY statement won't be executed.

Most COBOL programmers, when using WITH TEST BEFORE explicitly or implicitly have developed the habit of using '>' rather than '=' on UNTIL clauses. This would make the sample code:

```
PERFORM
        VARYING PD-Row-No FROM 1 BY 1 UNTIL PD-Row-No > 3 AFTER
           PD-Col-No FROM 1 BY 1 UNTIL PD-Col-No > 4
     DISPLAY PD (PD-Row-No, PD-Col-No) WITH NO ADVANCING
END-PERFORM
```

With this change, ABCDEFGHIJKL is once again displayed.

7.8.34 READ

7.8.34.1 Sequential READ

Sequential READ Syntax

```
READ file-name-1 [ { NEXT|PREVIOUS } ] RECORD [ INTO identifier-1 ]
~~~~                 { ~~~~ ~~~~~~~~~ }            ~~~~
 [ { IGNORING LOCK } ]
    { ~~~~~~~~~ ~~~~      }
    { WITH [ NO ] LOCK }
    {           ~~    ~~~~ }
    { WITH KEPT LOCK      }
    {        ~~~~ ~~~~    }
    { WITH IGNORE LOCK }
    {        ~~~~~~ ~~~~ }
    { WITH WAIT        }
             ~~~~
 [ AT END imperative-statement-1 ]
       ~~~
 [ NOT AT END imperative-statement-2 ]
    ~~~     ~~~
[ END-READ ]
  ~~~~~~~~
```

This form of the READ statement retrieves the next (or previous) record from a file.

1. The reserved words AT, RECORD and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The file-name-1 file must have been defined via an FD (see Section 6.2.1 [File/Sort-Description], page 123), not an SD.
3. The file-name-1 file must currently be open for INPUT (see [File OPEN Modes], page 359) or I-O.
4. If file-name-1 is an ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) file with an ACCESS MODE RANDOM, this statement cannot be used.
5. If file-name-1 was specified as ACCESS MODE SEQUENTIAL, this is the only format of the READ statement that is available.
6. If file-name-1 is an ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) file with ACCESS MODE DYNAMIC, this statement as well as a random READ (see Section 7.8.34.2 [Random READ], page 367) may be used.
7. The keywords NEXT and PREVIOUS specify what "direction of travel" the reading process will take through the file. If neither is specified, NEXT is assumed.
8. The PREVIOUS option is available only for ORGANIZATION INDEXED files.
9. When reading any sequential (any organization) or relative file, the "next" direction refers to the physical sequence of records in the file. When reading an indexed file, the "next"

and "previous" directions refer to the sequence of primary or alternate record key values in the file's records, regardless of where the records physically occur within the file.

10. The minimal statement READ file-name-1 is perfectly legal according to both READ formats. For that reason, when ACCESS MODE DYNAMIC has been specified and you want to tell the GnuCOBOL compiler that this minimal statement should be treated as a sequential READ, you must add either NEXT or PREVIOUS to the statement (otherwise it will be treated as a random READ).
11. A successful sequential READ will retrieve the next available record from file-name-1, in either a "next" or "previous" direction from the most-recently-read record, depending upon the use of the NEXT or PREVIOUS option. The newly-retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file's FD. If the optional INTO clause is present, a copy of the just-retrieved record will be automatically moved to identifier-1.
12. When an ORGANIZATION RELATIVE file has been successfully read, the file's RELATIVE KEY (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) field will be automatically populated with the relative record number (ordinal occurrence number) of the record in the file.
13. The optional LOCK options may be used to manually control access to the retrieved record by other programs while this program is running. See Section 2.1.30.2 [Record Locking], page 58, to review the various record locking behaviours.
14. The optional AT END clause, if coded, is used to detect and react to the failure of an attempt to retrieve another record from the file due to an end-of-file (i.e. no more records) condition.
15. The optional NOT AT END clause, if coded, will check for a file status value of 00. See [File Status Codes], page 105, for additional information.

7.8.34.2 Random READ

Random READ Syntax

```
READ file-name-1 RECORD [ INTO identifier-1 ]
~~~~                    ~~~~
  [ { IGNORING LOCK   } ]
    { ~~~~~~~~~ ~~~~      }
      { WITH [ NO ] LOCK }
    {           ~~    ~~~~ }
{ WITH KEPT LOCK         }
    {         ~~~~ ~~~~     }
  { WITH IGNORE LOCK }
    {         ~~~~~~ ~~~~ }
    { WITH WAIT        }
              ~~~~
  [ KEY IS identifier-2 ]
    ~~~
  [ INVALID KEY imperative-statement-1 ]
    ~~~~~~~
  [ NOT INVALID KEY imperative-statement-2 ]
    ~~~ ~~~~~~~
[ END-READ ]
  ~~~~~~~~
```

This form of the READ statement retrieves an arbitrary record from an ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) file.

1. The reserved words IS, KEY (on the INVALID and NOT INVALID clauses), RECORD and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The file-name-1 file must have been defined via an FD (see Section 6.2.1 [File/Sort-Description], page 123), not an SD.
3. The file-name-1 file must currently be open for INPUT (see [File OPEN Modes], page 359) or I-O.
4. If the ACCESS MODE of file-name-1 is SEQUENTIAL, or the ORGANIZATION of the file is any form of sequential, this format of the READ statement cannot be used.
5. If the ACCESS MODE of file-name-1 is RANDOM, this is the only format of the READ statement that is available.
6. If file-name-1 is an ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) file with ACCESS MODE DYNAMIC, this statement as well as a sequential READ (see Section 7.8.34.1 [Sequential READ], page 365) may be used.
7. The minimal statement READ file-name-1 is perfectly legal according to both READ formats. For that reason, when ACCESS MODE DYNAMIC has been specified and you want to tell the GnuCOBOL compiler that this minimal statement should be treated as a random READ, you must omit the NEXT or PREVIOUS available to the sequential format of the READ statement to ensure the statement will be treated as a random READ.

8. The optional KEY clause tells the compiler how a record is to be located in the file. If the KEY clause is absent, and the file is

   ORGANIZATION RELATIVE
   : the contents of the field declared as the file's RELATIVE KEY will be used to identify a record

   ORGANIZATION INDEXED
   : the contents of the field declared as the file's RECORD KEY will be used to identify a record.

   If the KEY clause is specified, and the file is

   ORGANIZATION RELATIVE
   : the contents of identifier-2 will be used as the relative record number of the record to be accessed. identifier-2 need not be the RELATIVE KEY (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) field of the file (although it could be if you wish).

   ORGANIZATION INDEXED
   : *identifier-2 must* be the RECORD KEY (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) or one of the file's ALTERNATE RECORD KEY fields (if any). The current contents of that field will identify the record to be accessed. If an alternate record key is used, and that key allows duplicate values, the record accessed will be the first one having that key value.
9. Once read from the file, the newly-retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file's FD. If the optional INTO clause is present, a copy of the just-retrieved record will be automatically moved to identifier-1.
10. When an ORGANIZATION RELATIVE file has been successfully read, the file's RELATIVE KEY (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) field will be automatically populated with the relative record number (ordinal occurrence number) of the record in the file.
11. The optional LOCK options may be used to manually control access to the retrieved record by other programs while this program is running. See Section 2.1.30.2 [Record Locking], page 58, to review the various record locking behaviours.
12. The optional INVALID KEY and NOT INVALID KEY clauses may be used to detect and react to the failure or success, respectively, by detecting non-zero (typically 23 = key not found = record not found) and 00 file status codes, respectively. See [File Status Codes], page 105, for additional information.

7.8.35 READY TRACE

READY TRACE Syntax

READY TRACE
~~~~~ ~~~~~

———————————————————————————————————————

The READY TRACE statement turns procedure or procedure-and-statement tracing on.

1. In order for this statement to be functional, tracing code must have been generated into the compiled program using either the -ftrace switch (procedures only) or -ftraceall switch (procedures and statements).
2. Tracing may be turned off at any point by executing the RESET TRACE statement (see Section 7.8.37 [RESET TRACE], page 371).
3. The COB_SET_TRACE run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) provides another way to control tracing. If this environment variable is set to a value of 'Y' prior to the start of program execution, tracing starts at the point the program begins execution, as if READY TRACE were the first executed statement.
~~~~~

7.8.36 RELEASE

RELEASE Syntax

```
RELEASE record-name-1 [ FROM { literal-1              } ]
~~~~~~~                      ~~~~ { identifier-1 }
```

The RELEASE statement adds a new record to a sort work file.

1. This statement is valid only within the INPUT PROCEDURE of a file-based SORT statement (see Section 7.8.44.1 [File-Based SORT], page 388).
2. The specified record-name-1 must be a record defined to the sort description (SD (see Section 6.2.1 [File/Sort-Description], page 123)) of the sort work file being processed by the current sort.
3. The optional FROM clause will cause literal-1 or identifier-1 to be automatically moved into record-name-1 prior to writing record-name-1's contents to the file-name-1. If this clause is not specified, it is the programmer's responsibility to populate record-name-1 with the desired data prior to executing the RELEASE.

7.8.37 RESET TRACE

RESET TRACE Syntax

RESET TRACE
~~~~~ ~~~~~

———————————————————————————————————————

The RESET TRACE statement turns procedure or procedure-and-statement tracing off.

1. By default, procedure and procedure-and-statement tracing is off as programs begin execu- tion. The READY TRACE statement (see Section 7.8.35 [READY TRACE], page 369) can be used to turn tracing on.
2. In order for this statement to be functional, tracing code must have been generated into the compiled program using either the -ftrace switch (procedures only) or -ftraceall switch (procedures and statements).
3. The COB_SET_TRACE run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) provides another way to control tracing. If this environment variable is set to a value of 'Y' prior to the start of program execution, tracing started at the point the program begins execution, as if READY TRACE were the first executed statement. The RESET TRACE statement, if executed, will then turn off tracing.
~~~~~

7.8.38 RETURN

RETURN Syntax

```
RETURN sort-file-name-1 RECORD
~~~~~~
 [ INTO identifier-1 ]
   ~~~~
   AT  END  imperative-statement-1
       ~~~
 [ NOT AT END imperative-statement-2 ]
   ~~~      ~~~
[ END-
RETURN ]
~~~~~~~~~~
```

The RETURN statement reads a record from a sort or merge work file.

1. The reserved words AT and RECORD are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The RETURN statement is valid only within the OUTPUT PROCEDURE of a file-based SORT (see Section 7.8.44.1 [File-Based SORT], page 388) or a MERGE statement (see Section 7.8.29 [MERGE], page 351) statement.
3. The sort-file-name-1 file must be a sort- or merge work file defined with a SD (see Section 6.2.1 [File/Sort-Description], page 123), not an FD.
4. A successful RETURN will retrieve the next available record from sort-file-name-1. The newly- retrieved record data will be saved into the 01-level record structure(s) that immediately follow the file's SD. If the optional INTO clause is present, a copy of the just-retrieved record will be automatically moved to identifier-1.
5. The mandatory AT END clause is used to detect and react to the failure of an attempt to retrieve another record from the file due to an end-of-file (i.e. no more records) condition.
6. The optional NOT AT END clause, if coded, will check checking for a file status value of 00. See [File Status Codes], page 105, for additional information.

7.8.39 REWRITE

REWRITE Syntax

```
REWRITE { record-name-1      [ FROM { identifier-1 } ] }
~~~~~~~ {                    [ ~~~~ { literal-1     } ] }
        {                                               }
        { FILE file-name-1   FROM { identifier-1 }      }
        { ~~~~               ~~~~ { literal-1        }  }

    [ WITH [ NO ] LOCK ]
            ~~     ~~~~
    [ INVALID KEY imperative-statement-1 ]
      ~~~~~~~
    [ NOT INVALID KEY imperative-statement-2 ]
      ~~~ ~~~~~~~
[ END-REWRITE
]
```

The REWRITE statement replaces a logical record on a disk file.

1. The reserved words KEY and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The record-name-1 specified on the statement must be defined as an 01-level record subordinate to the File Description (FD (see Section 6.2.1 [File/Sort-Description], page 123)) of a file that is currently open for I-O (see [File OPEN Modes], page 359).
3. The optional FROM clause will cause literal-1 or identifier-1 to be automatically moved into record-name-1 prior to writing record-name-1's contents to the file-name-1. If this clause is not specified, it is the programmer's responsibility to populate record-name-1 with the desired data prior to executing the REWRITE.
4. This statement may not be used with ORGANIZATION LINE SEQUENTIAL (see Section 5.2.1.2 [ORGANIZATION LINE SEQUENTIAL], page 110) files.
5. Rewriting a record does not cause the contents of the file to be physically updated until the next block of the file is read, a COMMIT (see Section 7.8.8 [COMMIT], page 296) or UNLOCK statement (see Section 7.8.52 [UNLOCK], page 405) is issued or that file is closed.
6. If the file has ORGANIZATION SEQUENTIAL (see Section 5.2.1.1 [ORGANIZATION SEQUENTIAL], page 109):
   A. The record to be rewritten will be the one retrieved by the most-recently executed READ (see Section 7.8.34 [READ], page 365) of the file.
   B. If the FD of the file contains the RECORD CONTAINS or RECORD IS VARYING clause, and that clause allows the record size to vary, the size of record-name-1 cannot be altered.
7. If the file has ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114):
   A. If the file has ACCESS MODE SEQUENTIAL, the record to be rewritten will be the one retrieved by the most-recently executed READ of the file. If the file has ACCESS MODE RANDOM or ACCESS MODE DYNAMIC, no READ is required before a record may be rewritten
   — the RELATIVE KEY or RECORD KEY definition for the file, respectively, will specify the

record to be updated.

   B. If the FD of the file contains the RECORD CONTAINS or RECORD IS VARYING clause, and that clause allows the record size to vary, the size can be altered.

8. The optional LOCK options may be used to manually control access to the re-written record by other programs while this program is running. See Section 2.1.30.2 [Record Locking], page 58, to review the various record locking behaviours.
9. The optional INVALID KEY and NOT INVALID KEY clauses may be used to detect and react to the failure or success, respectively, by detecting non-zero (typically 23 = key not found = record not found) and 00 file status codes, respectively. See [File Status Codes], page 105, for additional information.

7.8.40 ROLLBACK

ROLLBACK Syntax

ROLLBACK
~~~~~~~~

———————————————————————————————————————

The ROLLBACK statement has the same effect as if an UNLOCK statement (see Section 7.8.52 [UNLOCK], page 405) were executed against every open file in the program.

1. All locks currently being held for all open files will be released.
2. See Section 2.1.30.2 [Record Locking], page 58, to review the various record locking behaviours.
~~~~~~~~

7.8.41 SEARCH

SEARCH Syntax

```
SEARCH  table-name-1
~~~~~~
    [ VARYING index-name-1 ]
      ~~~~~~~
    [ AT END imperative-statement-1 ]
         ~~~
    { WHEN conditional-expression-1 imperative-statement-2 }...
      ~~~~
[ END-SEARCH ]
  ~~~~~~~~~~
```

———

The SEARCH statement is used to sequentially search a table, stopping either once a specific value is located within the table or when the table has been completely searched.

1. The reserved word AT is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. The searching process will be controlled through a Search Index — a data item with a USAGE (see Section 6.9.59 [USAGE], page 228) of INDEX. The search index is either the index-name-1 identifier specified on the VARYING clause or — if no VARYING is specified — the USAGE INDEX data item implicitly created by an INDEXED BY (see Section 6.9.35 [OCCURS], page 191) clause in the table's definition.
3. At the time the SEARCH statement is executed, the current value of the search index data item will define the starting position in the table where the searching process will begin. Typically, one initializes that index to a value of 1 before starting the SEARCH via SET search-index TO 1.
4. Each of the conditional-expression-ns on the WHEN clause(s) should involve a data element within the table, subscripted using the search index.
5. The searching process is as follows:
   A. Each conditional-expression-n will be evaluated, in turn, until either one evaluates to a value of TRUE or all have evaluated to FALSE.
   B. The imperative-statement-n (see [Imperative Statement], page 758) specified on the WHEN clause whose conditional-expression-n evaluated to TRUE will be executed; after that, the search will be considered complete and control will fall into the first executable statement following the SEARCH.
   C. If all conditional-expression-ns evaluated to FALSE:
      - The search index will be incremented by 1
      - If the search index now has a value greater than the number of entries in the table, the search is considered to have failed and the imperative-statement-1 on the optional AT END clause, if any, will be executed. After that, control will fall into the first executable statement following the SEARCH.
      - If the search index now has a value less than or equal to the number of entries in the table, search processing returns back to step A.

7.8.42 SEARCH ALL

SEARCH ALL Syntax

```
SEARCH ALL table-name-1
~~~~~~ ~~~
    [ AT END imperative-statement-1 ]
          ~~~
      WHEN  conditional-expression-1  imperative-statement-2
      ~~~~
[ END-SEARCH ]
~~~~~~~~~~
```

The SEARCH ALL statement performs a binary, or half-interval, search against a sorted table. This is generally significantly faster than performing a sequential SEARCH of a table, especially if the table contains a large number of entries.

1. The reserved word AT is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. To be eligible for searching via SEARCH ALL:
   A. The OCCURS clause of table-name-1 must contain the following elements:
      - An INDEXED BY entry to define an implicit search index data item with a USAGE (see Section 6.9.59 [USAGE], page 228) of INDEX.
      - An ASCENDING KEY or DESCENDING KEY clause to specify the field within the table by which all entries in the table are sorted.
   B. Just because the table has one or more KEY clauses doesn't mean the data is actually in that sequence in the table — the actual sequence of the data must agree with the KEY clause(s)! A table-based SORT (see Section 7.8.44.2 [Table SORT], page 392) can prove very useful in this regard.
   C. No two records in the table may have the same KEY field values. If the table has multiple KEY definitions, then no two records in the table may have the same combination of KEY field values.
3. If rule A is violated, the compiler will reject the SEARCH ALL. If rules B and/or C are violated, there will be no message issued by the compiler, but the run-time results of a SEARCH ALL against the table will probably be incorrect.
4. The conditional-expression-1 should involve the KEY field(s), using the search index (the table's INDEXED BY index name) as a subscript.
5. The function of the single, mandatory, WHEN clause is to compare the key field(s) of the table, as indexed by the search index data item, against whatever literal and/or identifier values you are comparing the key field(s) to in the conditional-expression-1 in order to locate the desired entry in the table. The search index will be automatically varied in a manner designed to require the minimum number of tests.
6. The internal processing of the SEARCH ALL statement begins by setting internal "first" and "last" pointers to the 1st and last entry locations of the table. Processing then proceeds as follows:
   A. The entry half-way between "first" and "last" is identified. We'll call this the "current" entry, and will set its table entry location into index-name-1.

B. The conditional-expression-1 is evaluated. This comparison of the key(s) against the target literal/identifier values will have one of three possible outcomes:
   - If the key(s) and value(s) match, imperative-statement-2 (see [Imperative Statement], page 758) is executed, after which control falls through into the next statement following the SEARCH ALL.
   - If the key(s) are LESS THAN the value(s), then the table entry being searched for can only occur in the "current" to "last" range of the table, so a new "first" pointer value is set (it will be set to the "current" pointer).
   - If the key(s) are GREATER THAN the value(s), then the table entry being searched for can only occur in the "first" to "current" range of the table, so a new "last" pointer value is set (it will be set to the "current" pointer).

C. If the new "first" and "last" pointers are different than the old "first" and "last" pointers, there's more left to be searched, so return to step A and continue.

D. If the new "first" and "last" pointers are the same as the old "first" and "last" pointers, the table has been exhausted and the entry being searched for cannot be found; imperative-statement-1 is executed, after which control falls through into the next statement following the SEARCH ALL. If there is no AT END clause coded, control simply falls into the next statement following the SEARCH ALL.

7. The net effect of the above algorithm is that only a fraction of the number of elements in the table need ever be tested in order to decide whether or not a particular entry exists. This is because the half the remaining entries in the table are discarded each time an entry is checked.
8. Computer scientists will compare the two techniques implemented by the SEARCH and SEARCH ALL statements as follows:
9. When searching a table with N entries, a sequential search will need an average of N/2 tests and a worst case of N tests in order to find an entry and N tests to identify that an entry doesn't exist.
10. When searching a table with N entries, a binary search will need a worst-case of log2(N) tests in order to find an entry and log2(N) tests to identify that an entry doesn't exist (N = the number of entries in the table), where log2 is the base-2 logarithm function.

Here's a more practical view of the difference. Let's say that a table has 1,000 entries in it. With a sequential search, on average, you'll have to check 500 of them to find an entry and you'll have to look at all 1,000 of them to find that an entry doesn't exist.

With a binary search, express the number of entries as a binary number (1,000 = 1111101000), count the number of digits in the result (which is, essentially, what a logarithm is, when rounded up to the next integer — the number of digits a decimal number would have if expressed in the logarithm's number base). In this case, we end up with 10 — that is the worst-case number of tests required to find an entry or to identify that it doesn't exist. That's quite an improvement!

7.8.43 SET

7.8.43.1 SET ENVIRONMENT

SET ENVIRONMENT Syntax

```
SET ENVIRONMENT { literal-1          } TO { literal-2          }
~~~ ~~~~~~~~~~~ { identifier-1 } ~~ { identifier-2 }
```

The SET ENVIRONMENT statement provides a straight-forward means of setting environment values from within a program.

1. The value of literal-1 or identifier-1 specifies the name of the environment variable to set.
2. The value of literal-2 or identifier-2 specifies the value to be assigned to the environment variable.
3. SET ENVIRONMENT is especially useful for managing the configuration variables listed in chapter 10.2.3 Run Time Environment Variables and in particular the variables in chapter
   10.2.3.5 Screen I/O.
4. As an example, use SET ENVIRONMENT 'COB_HIDE_CURSOR' TO 'TRUE' to make the cursor invisible when doing a DISPLAY or an ACCEPT.
5. Environment variables created or changed from within GnuCOBOL programs will be available to any sub-shell processes spawned by that program (i.e. CALL "SYSTEM") but will not be known to the shell or console window that started the GnuCOBOL program.

This is a much simpler and more readable means of setting environment variables than by using the DISPLAY UPON ENVIRONMENT-NAME statement (see Section 7.8.12.3 [DISPLAY UPON ENVIRONMENT-NAME], page 304). For example, these two code sequences produce identical results:

```
DISPLAY "VARNAME" UPON ENVIRONMENT-NAME
DISPLAY "VALUE" UPON ENVIRONMENT-VALUE

SET ENVIRONMENT "VARNAME" TO "VALUE"
```

7.8.43.2 SET Program-Pointer

SET Program-Pointer Syntax

```
SET program-pointer-1 TO ENTRY { literal-1         }
~~~                      ~~ ~~~~~ { identifier-1 }
```

The SET Program-Pointer statement allows you to retrieve the address of a procedure division code module — specifically the PROGRAM-ID, FUNCTION-ID or an entry-point established via the ENTRY statement (see Section 7.8.14 [ENTRY], page 319).

1. The USAGE (see Section 6.9.59 [USAGE], page 228) of program-pointer-1 must be PROGRAM-POINTER.
2. The literal-1 or identifier-1 value specified must name a primary entry-point name (PROGRAM-ID of a subroutine or FUNCTION-ID of a user-defined function) or an alternate entry-point defined via an ENTRY statement within a subprogram.
3. Once the address of a procedure division code area has been acquired in this way, the address could be passed to a subroutine (usually written in C) for whatever use it needs it for. For examples of PROGRAM-POINTERs at work, see the discussions of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL_ERROR_PROC], page 575) and CBL_EXIT_PROC built-in system subroutine (see Section 8.2.28 [CBL_EXIT_PROC], page 577).

7.8.43.3 SET ADDRESS

SET ADDRESS Syntax

```
SET [ ADDRESS OF ] { pointer-name-1 }...
~~~    ~~~~~~~ ~~      {  identifier-1     }

      TO [ ADDRESS OF ]      { pointer-name-2 }
      ~~    ~~~~~~~ ~~       {  identifier-2     }
```

The SET ADDRESS statement can be used to work with the addresses of data items rather than their contents.

1. When the ADDRESS OF clause is used before the TO you will be using this statement to alter the address of a linkage section or BASED (see Section 6.9.8 [BASED], page 161) data item. Without that clause you will be assigning an address to one or more data items whose USAGE (see Section 6.9.59 [USAGE], page 228) is POINTER.
2. When the ADDRESS OF clause is used after the TO, this statement will be identifying the address of identifier-2 as the address to be assigned to identifier-1 or stored in pointer-name-1.
3. If the ADDRESS OF clause is absent after the TO, the contents of pointer-name-2 will serve as the address to be assigned.

7.8.43.4 SET Index

SET Index Syntax

```
SET index-name-1 TO { literal-1          }
~~~                 ~~ { identifier-2 }
```

This statement assigns a value to a USAGE INDEX data item.

1. Either the USAGE (see Section 6.9.59 [USAGE], page 228) of index-name-1 should be INDEX, or index-name-1 must be identified in a table INDEXED BY clause.

7.8.43.5 SET UP/DOWN

SET UP/DOWN Syntax

```
SET identifier-1 { UP         } BY [ LENGTH OF ] { literal-1     }
~~~              { ~~     } ~~     ~~~~~~ ~~     {  identifier-2  }
                 { DOWN   }
                    ~~~~
```

Use this statement to increment or decrement the value of an index or pointer by a specified amount.

1. The USAGE (see Section 6.9.59 [USAGE], page 228) of identifier-1 must be INDEX, POINTER or PROGRAM-POINTER.
2. The typical usage when identifier-1 is a USAGE INDEX data item is to increment its value UP or DOWN by 1, since an index is usually being used to sequentially walk through the elements of a table.

7.8.43.6 SET Condition Name

SET Condition Name Syntax

```
SET  condition-name-1...     TO { TRUE    }
~~~                          ~~ { ~~~~    }
                                { FALSE }
                                  ~~~~~
```

The SET Condition Name statement provides one method of specifying the TRUE / FALSE value of a level-88 condition name.

1. By setting the specified condition-name-1(s) to a TRUE or FALSE value, you will actually be assigning a value to the parent data item(s) to which the condition name data item(s) is(are) subordinate to.
2. When specifying TRUE, the value assigned to each parent data item will be the first value specified on the condition name's VALUE clause.
3. When specifying FALSE, the value assigned to each parent data item will be the value specified for the FALSE clause of the condition name's definition; if any condition-name-1 occurrence lacks a FALSE clause, the SET statement will be rejected by the compiler.

7.8.43.7 SET Switch

SET Switch Syntax

```
SET  mnemonic-name-1...      TO  { ON    }
~~~                            ~~ { ~~     }
                                  { OFF }
                                     ~~~
```

This form of the SET statement is used to turn switches on or off.

1. Switches are defined using the SPECIAL-NAMES (see Section 5.1.3 [SPECIAL-NAMES], page 90) paragraph.
2. Switches may be tested via the IF statement (see Section 7.8.23 [IF], page 338) and a Switch-Status Condition. See Section 2.1.27.4 [Switch-Status Conditions], page 48, for more information.

7.8.43.8 SET ATTRIBUTE

SET ATTRIBUTE Syntax

```
SET identifier-1 ATTRIBUTE { { BELL                  } { ON  }...
~~~              ~~~~~~~~~~ { ~~~~                  } { ~~  }
                            { BLINK                 } { OFF }
                            { ~~~~~                 }   ~~~
                            { HIGHLIGHT             }
                            { ~~~~~~~~~~            }
                            { LEFTLINE              }
                            { ~~~~~~~~~             }
                            { LOWLIGHT              }
                            { ~~~~~~~~~             }
                            { OVERLINE              }
                            { ~~~~~~~~~             }
                            { REVERSE-VIDEO         }
                            { ~~~~~~~~~~~~~         }
                            { UNDERLINE             }
                              ~~~~~~~~~
```

The SET ATTRIBUTE statement may be used to modify one or more attributes of a screen section data item at run-time.

1. When making an attribute change to identifier-1, the change will not become visible on the screen until the screen section data item containing identifier-1 is next accepted (if identifier-1 is an input field) or is next displayed (if identifier-1 is not an input field).
2. The attributes shown in the syntax diagram are the only ones that may be altered by this statement. See Section 6.9 [Data Description Clauses], page 154, for information on their usage.

7.8.43.9 SET LAST EXCEPTION

SET ATTRIBUTE Syntax

```
SET LAST EXCEPTION TO { OFF }
~~~ ~~~~ ~~~~~~~~~~ ~~       ~~~
```

The SET LAST EXCEPTION statement will set the last program exception status to indicate no exception.

1. The predefined object reference EXCEPTION-OBJECT is set to null, and the last exception status is set to indicate no exception.
2. This action resets the global exception object completely (FUNCTION EXCEPTION-{FILE, LOCATION, STATEMENT, STATUS }), and will not show anything afterwards), no matter what the last exception was (such as a divide by zero). Use with care.

7.8.44 SORT

7.8.44.1 File-Based SORT

File-Based SORT Syntax

```
SORT sort-file-1
~~~~
    { ON { ASCENDING                        } KEY identifier-1... }...
       { ~~~~~~~~~~        }
            { DESCENDING }
              ~~~~~~~~~~
    [ WITH DUPLICATES IN ORDER ]
              ~~~~~~~~~~
    [ COLLATING SEQUENCE IS alphabet-name-1 ]
       ~~~~~~~~~

    { INPUT PROCEDURE IS procedure-name-1            }
    { ~~~~~~ ~~~~~~~~~~                              }
    {            [  THRU|THROUGH procedure-name-2 ]  }
    {               ~~~~ ~~~~~~~                     }
    { USING file-name-1 ...                          }
       ~~~~~

    { OUTPUT PROCEDURE IS procedure-name-3           }
    { ~~~~~~ ~~~~~~~~~~                              }
    {            [  THRU|THROUGH procedure-name-4 ]  }
    {               ~~~~ ~~~~~~~                     }
    { GIVING file-name-2  ...                        }
       ~~~~~~
```

The DUPLICATES clause is syntactically recognized but is otherwise non-functional.

This format of the SORT statement is designed to sort large volumes of data according to one or more key fields.

1. The reserved words IN, IS, KEY, ON, ORDER, SEQUENCE and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The reserved words THRU and THROUGH are interchangeable.
3. GnuCOBOL always behaves as if the WITH DUPLICATES IN ORDER clause is specified, even if it isn't.

   While any COBOL implementation's sort or merge facilities guarantee that records with duplicate key values will be in proper sequence with regard to other records with different key values, they generally make no promises as to the resulting relative sequence of records having duplicate key values with one another.

   Some COBOL implementations provide this optional clause to force their sort and merge facilities to retain duplicate key-value records in their original input sequence, relative to one another.
4. The sort-file-1 named on the SORT statement must be defined using a sort description (SD

(see Section 6.2.1 [File/Sort-Description], page 123)). This file is referred to in the remainder of this discussion as the sort work file.

5. If specified, file-name-1 and file-name-2 must reference ORGANIZATION LINE SEQUENTIAL (see Section 5.2.1.2 [ORGANIZATION LINE SEQUENTIAL], page 110) or ORGANIZATION SEQUENTIAL (see Section 5.2.1.1 [ORGANIZATION SEQUENTIAL], page 109) files. These files must be defined using a file description (FD (see Section 6.2.1 [File/Sort-Description], page 123)). The same file(s) may be used for file-name-1 and file-name-2.
6. The identifier-1 . . . field(s) must be defined as field(s) within a record of sort-file-1.
7. A sort work file is never opened or closed.
8. The sorting process works in three stages — the Input Stage, the Sort Stage and the Output Stage.
9. The following points pertain to the Input Stage:
   A. The data to be sorted is loaded into the sort work file either by copying the entire contents of the file(s) named on the USING clause (done automatically by the sort) or by utilizing an input procedure.
   B. When USING is specified, none of the file-name-1 files may be open at the time the SORT statement is executed.
   C. When an input procedure is used, the procedure(s) specified on the INPUT PROCEDURE clause will be invoked as if by a procedural PERFORM statement (see Section 7.8.33.1 [Procedural PERFORM], page 360) with no VARYING, TIMES or UNTIL options specified. Records will be loaded into the sort work file — one at a time — within the input procedure using the RELEASE statement (see Section 7.8.36 [RELEASE], page 370). This, by the way, is how you could sort the contents of relative or indexed files.

      A GO TO statement (see Section 7.8.22 [GO TO], page 335) that transfers control out of the input procedure will terminate the SORT statement but allows the program to continue executing from the point where the GO TO statement transferred control to. Once an input procedure has been "aborted" using a GO TO it cannot be resumed, and the contents of the sort work file are lost. You may, however, re-execute the SORT statement itself.1

      An input procedure should be terminated in the same way a procedural PERFORM state- ment would be.

      Neither a another file-based SORT statement nor a MERGE statement may be executed within the input procedure unless those statements utilize a different sort or merge work file.
   D. Once the input procedure terminates, the input phase is complete.
   E. As data is loaded into the sort work file, it is actually being buffered in dynamically-allocated memory. Only if the amount of data to be sorted exceeds the amount of available sort memory (128 MB) will actual disk files be allocated and utilized. There is a COB_SORT_MEMORY run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) that you may use to allocate more or less memory to the sorting process.
10. The following points pertain to the Sort Stage:
   A. The sort will take place by arranging the data records in the sequence defined by the KEY specification(s) on the SORT statement according to the COLLATING SEQUENCE specified on the SORT (if any) or — if none was defined — the PROGRAM COLLATING SEQUENCE

[1] Using a GO TO statement to prematurely terminate a sort, or re-starting a previously-cancelled sort is not considered good programming style and should be avoided.

(see Section 5.1.2 [OBJECT-COMPUTER], page 88). Keys may be any supported data type and USAGE (see Section 6.9.59 [USAGE], page 228) except for level-78 or level-88 data items.

B. For example, let's assume we're sorting a series of financial transactions. The SORT statement might look like this:

```
SORT Sort-File
    ASCENDING   KEY Transaction-Date
    ASCENDING   KEY Account-Number
    DESCENDING KEY Transaction-Amount
```

The effect of this statement will be to sort all transactions into ascending order of the date the transaction took place (oldest first, newest last). Unless the business running this program is going out of business, there are most-likely many transactions for any given date. Therefore, within each grouping of transactions all with the same date, transactions will be sub-sorted into ascending sequence of the account number the transactions apply to. Since it's quite possible there might be multiple transactions for an account on any given date, a third level sub-sort will arrange all transactions for the same account on the same date into descending sequence of the actual amount of the transaction (largest first, smallest last). If two or more transactions of $100.00 were recorded for account #12345 on the 31st of August 2009, those transactions will be retained in the order in which they were loaded into the sort work file.

C. Should disk work files be necessary due to the amount of data being sorted, they will be automatically allocated to disk in a folder defined by the TMPDIR run-time environment variable, TMP run-time environment variable or TEMP run-time environment variable run-time environment variables (see Section 10.2.3 [Run Time Environment Variables], page 700) (checked for existence in that sequence). These disk files will be automatically purged upon SORT termination or program execution termination (normal or otherwise).

11. The following points pertain to the Output Stage:

A. Once the sort stage is complete, a copy of the sorted data will be written to each file-name-2 if the GIVING clause was specified. None of the file-name-2 files can be open at the time the sort is executed.

B. When an output procedure is used, the procedure(s) specified on the OUTPUT PROCEDURE clause will be invoked as if by a procedural PERFORM statement (see Section 7.8.33.1 [Procedural PERFORM], page 360) with no VARYING, TIMES or UNTIL options specified. Records will be retrieved from the sort work file — one at a time — within the output procedure using the RETURN statement (see Section 7.8.38 [RETURN], page 372).

A GO TO statement (see Section 7.8.22 [GO TO], page 335) that transfers control out of the output procedure will terminate the SORT statement but allows the program to continue executing from the point where the GO TO statement transferred control to. Once an output procedure has been "aborted" using a GO TO it cannot be resumed, and the contents of the sort work file are lost. You may, however, re-execute the SORT statement itself. USING A GO TO statement[2]

An output procedure should be terminated in the same way a procedural PERFORM statement would be.

Neither a another file-based SORT statement nor a MERGE statement may be executed within the output procedure unless those statements utilize a different sort or merge work file.

[2] To prematurely terminate a sort, or re-starting a previously-cancelled sort is not considered good programming style and should be avoided.

C. Once the output procedure terminates, the sort is complete.

7.8.44.2 Table SORT

Table SORT Syntax

```
SORT table-name-1
~~~~
    { ON { ASCENDING                          } KEY identifier-1... }...
      { ~~~~~~~~~~~       }
           { DESCENDING }
             ~~~~~~~~~~
    [ WITH DUPLICATES IN ORDER ]
             ~~~~~~~~~~
    [ COLLATING SEQUENCE IS alphabet-name-1 ]
      ~~~~~~~~~~
```

The DUPLICATES clause is syntactically recognized but is otherwise non-functional.

———

This format of the SORT statement sorts relatively small quantities of data — namely data contained in a data division table — according to one or more key fields.

1. The reserved words IN, IS, KEY, ON, ORDER, SEQUENCE and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. GnuCOBOL always behaves as if the WITH DUPLICATES IN ORDER clause is specified, even if it isn't.

   While any COBOL implementation's sort or merge facilities guarantee that records with duplicate key values will be in proper sequence with regard to other records with different key values, they generally make no promises as to the resulting relative sequence of records having duplicate key values with one another.

   Some COBOL implementations provide this optional clause to force their sort and merge facilities to retain duplicate key-value records in their original input sequence, relative to one another.
3. The table-name-1 data item must be a table defined in any data division section except the report or screen sections.
4. The data within table-name-1 will be sorted in-place (i.e. no sort file is required).
5. The sort will take place by rearranging the data in table-name-1 into the sequence defined by the KEY specification(s) on the SORT statement, according to the COLLATING SEQUENCE specified on the SORT (if any) or — if none was defined — the PROGRAM COLLATING SEQUENCE (see Section 5.1.2 [OBJECT-COMPUTER], page 88). Keys may be any supported data type and USAGE (see Section 6.9.59 [USAGE], page 228) except for level-78 or level-88 data items.
6. If you are sorting table-name-1 for the purpose of preparing the table for use with a SEARCH ALL statement (see Section 7.8.42 [SEARCH ALL], page 377), care must be taken that the KEY specifications on the SORT agree with those in the table's definition.
7. Although the specification of one or more KEY clauses is optional, currently, a table sort with no KEY specification(s) made on the SORT statement is unsupported by GnuCOBOL and will be rejected by the compiler.

7.8.45 START

START Syntax

```
START file-name-1
~~~~~
  [ { FIRST                                                    } ]
    { ~~~~~                                                    }
    { LAST                                                     }
    { ~~~~                                                     }
    { KEY { IS EQUAL TO | IS = | EQUALS        } identifier-1 }
      ~~~ {      ~~~~~              ~~~~~~      }
          { IS GREATER THAN | IS >              }
          {      ~~~~~~~                        }
          { IS GREATER THAN OR EQUAL TO | IS >= }
          {      ~~~~~~~       ~~ ~~~~~         }
          { IS NOT LESS THAN                    }
          {      ~~~ ~~~~                       }
          { IS LESS THAN | IS <                 }
          {      ~~~~                           }
          { IS LESS THAN OR EQUAL TO | IS <=    }
          {      ~~~~        ~~ ~~~~~           }
          { IS NOT GREATER THAN                 }
                 ~~~ ~~~~~~~

      [ WITH {SIZE}      arithmetic-expression ]
             ~~~~
      [      {LENGTH} arithmetic-expression ]
             ~~~~~~

  [ INVALID KEY imperative-statement-1 ]
    ~~~~~~~
  [ NOT INVALID KEY imperative-statement-2 ]
    ~~~ ~~~~~~~
[ END-START ]
  ~~~~~~~~~
```

———————————————————————————————————————

The START statement defines the logical starting point within a relative or indexed file for subsequent sequential read operations. It positions an internal logical record pointer to a particular record in the file, but does not actually transfer any of that record's data into the record buffer.

1. The reserved words IS, THAN and TO are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. To use this statement, file-name-1 must be an ORGANIZATION RELATIVE (see Section 5.2.1.3 [ORGANIZATION RELATIVE], page 112) or ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) file that must have been defined with an ACCESS MODE DYNAMIC or ACCESS MODE SEQUENTIAL in its SELECT statement (see Section 5.2.1 [SELECT], page 102).
3. At the time this statement is executed, file-name-1 must be open in either INPUT or I-O (see [File OPEN Modes], page 359) mode.

4. If file-name-1 is a relative file, identifier-1 must be the defined RELATIVE KEY of the file.
5. If file-name-1 is an indexed file, identifier-1 must be the defined RECORD KEY of the file or any of the ALTERNATE RECORD KEY fields for the file.
6. If no FIRST, LAST or KEY clause is specified, KEY IS EQUAL TO xxx will be assumed, where xxx is the defined RELATIVE KEY of (if file-name-1 is a relative file) or the defined RECORD KEY (if file-name-1 is an indexed file).
7. After successful execution of a START statement, the internal logical record pointer into the file-name-1 data will be positioned to the record which satisfied the actual or implied FIRST, LAST or KEY clause specification, as follows:

   FIRST the logical record pointer will point to the first record in the file.

   LAST the logical record pointer will point to the last record in the file.

   KEY (specified or implied), and the relation used is. Warning: Later versions of the compiler may well not use implied, so always specify it and it makes the code easier to read any way.

   SIZE WITH SIZE or LENGTH arithmetic-expression specifies the number of characters in the key to be used in the positioning process.

   LENGTH WITH LENGTH or SIZE arithmetic-expression specifies the number of characters in the key to be used in the positioning process.

   SIZE and LENGTH are interchangeable and mean exactly the same.

   EQUAL TO, GREATER THAN or GREATER THAN OR EQUAL TO (or equivalent)
   the logical record pointer will be specified to the first record satisfy- ing the relation condition; to identify this record. The file's contents are searched in a first-to-last (in sequence of the key implied by the KEY clause), provided the relation is

   LESS THAN, LESS THAN OR EQUAL TO or NOT GREATER THAN (or equivalent)
   the logical record pointer will be specified to the last record satisfy- ing the relation condition; to identify this record. The file's contents are searched in a last-to-first (in sequence of the key implied by the KEY clause)

   The next sequential READ statement will read the record that is pointed to by the logical record pointer.
8. The optional INVALID KEY and NOT INVALID KEY clauses may be used to detect and react to the failure or success, respectively, by detecting non-zero (typically 23 = key not found = record not found) and 00 file status codes, respectively. See [File Status Codes], page 105, for additional information.

7.8.46 STOP

STOP Syntax

```
STOP {   RUN [ { RETURNING|GIVING  { literal-1     }           } ] }
~~~~ {  ~~~    { ~~~~~~~~~ ~~~~~~  { identifier-1 }           }   }
     {         {                                               }   }
     {         { WITH { ERROR   } STATUS [ { literal-2     } ] }   }
     {         {      { ~~~~~   }          { identifier-2 }   }   }
     {         {      { NORMAL }                              }   }
     {                 ~~~~~~                                     }
     { literal-3                                                  }
```

The STOP statement suspends program execution. Some options will allow program execution to resume while others return control to the operating system.

1. The reserved words STATUS and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The reserved words RETURNING and GIVING are interchangeable.
3. The RUN clause halts the program without displaying any special message to that effect.
4. The literal-3 clause displays the specified text on the SYSOUT/STDOUT device, waits for the user to press the Enter key and then — once the key has been pressed — allows the program to continue execution.
5. The optional RETURNING clause provides the opportunity to return a numeric value to the operating system (an exit status). The manner in which the exit status may be interrogated by the operating system varies, but Windows can use %ERRORLEVEL% to query the exit status while Unix shells such as sh, bash and ksh can query the exit status as $?. Other Unix shells may have different ways to access return code values.
6. The STATUS clause provides another means of returning an exit status. Using the STATUS clause is functionally equivalent to using the RETURNING clause.
7. Using the STATUS clause without a literal-2 or identifier-2 will return an exit status of 0 if the NORMAL keyword is used or a 1 if ERROR was specified.
8. Your program will always return an exit status, even if no RETURNING or STATUS clause is specified. In the absence of the use of these clauses, the value in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) at the time the STOP statement is executed will be used as the exit status.
9. Any programmer-defined exit procedure (established via the CBL_EXIT_PROC built-in system subroutine (see Section 8.2.28 [CBL_EXIT_PROC], page 577)) will be executed by STOP RUN, but not by STOP literal-3.
10. Valid return code values can be in the range -2147483648 to +2147483647.
11. The three code snippets below are all equivalent. They show different ways in which a GnuCOBOL program may be coded to pass an exit status value of 16 back to the operating system and then halt.
    1.
    ```
    STOP RUN RETURNING 16
    ```
    2.
    ```
    MOVE 16 TO RETURN-CODE
    STOP RUN
    ```

3.

```
STOP RUN WITH ERROR STATUS 16
```

7.8.47 STRING

STRING Syntax

```
STRING
~~~~~~
   { { literal-1       } [ DELIMITED BY { SIZE        } ] }...
     {  identifier-1  }   ~~~~~~~~~~   { ~~~~          }
                                       { literal-2     }
                                       { identifier-2 }
     INTO identifier-3
     ~~~~
   [ WITH POINTER identifier-4 ]
           ~~~~~~~
   [ ON OVERFLOW imperative-statement-1 ]
         ~~~~~~~~
   [ NOT ON OVERFLOW imperative-statement-2 ]
     ~~~     ~~~~~~~~
[ END-STRING ]
  ~~~~~~~~~~
```

The STRING statement is used to concatenate all or a part of one or more strings together, forming a new string.

1. The reserved words BY, ON and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. All literals and identifiers (except for identifier-4) must be explicitly or implicitly defined with a USAGE (see Section 6.9.59 [USAGE], page 228) of DISPLAY. Any of the identifiers may be group items.
3. The POINTER data item — identifier-4 — must be a non-edited elementary integer numeric data item with a value greater than zero.
4. Each literal-1 / identifier-1 will be referred to as a source item. The receiving data item is *identifier-3*.
5. The STRING statement's processing is based upon a current character pointer. The initial value of the current character pointer will be the value of identifier-4 at the time the STRING statement began execution. If no POINTER clause is coded, a value of 1 (meaning "the 1st character position") will be assumed for the current character pointer's initial value.
6. For each source item, the contents of the sending item will be copied — character-by- character — into identifier-3 at the character position specified by the current character pointer. After each character is copied, the current character pointer will be incremented by 1 so that it points to the position within identifier-3 where the next character should be copied.
7. The DELIMITED BY clause specifies how much of each source item will be copied into identifier-3. DELIMITED BY SIZE (the default if no DELIMITED BY clause is specified) causes the entire contents of the source item to be copied into identifier-3.
8. Using DELIMITED BY literal-2 or DELIMITED BY identifier-2 causes only the contents of the source item up to but not including the character sequence specified by the literal or identifier to be copied.

9. STRING processing will cease when one of the following occurs:
   A. The initial value of the current character pointer is less than 1 or greater than the number of characters in identifier-3, or. . .
   B. The value of the current character pointer exceeds the size of identifier-3 at the point the STRING statement wants to copy a character into identifier-3, or. . .
   C. All sending items have been fully processed
10. If event A occurs, identifier-3 will remain unchanged.
11. The occurrence of either event A or B triggers what is referred to as an overflow condition.
12. The identifier-3) is neither automatically initialized (to spaces or any other value) at the start of a STRING statement nor will it be space-filled should the total number of sending item characters copied into it be less than its size. You may explicitly initialize identifier- 3 yourself via the INITIALIZE (see Section 7.8.24 [INITIALIZE], page 339) or MOVE (see Section 7.8.30 [MOVE], page 354) statements before executing the STRING if you wish.
13. The optional ON OVERFLOW and NOT ON OVERFLOW clauses may be used to detect and react to the occurrence or not, respectively, of an overflow condition. See Section 7.6.5 [ON OVERFLOW + NOT ON OVERFLOW], page 256, for additional information.

7.8.48 SUBTRACT

7.8.48.1 SUBTRACT FROM

SUBTRACT FROM Syntax

```
SUBTRACT { literal-1          }...    FROM { identifier-2
~~~~~~~~ { identifier-1 }              ~~~~

        [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                } ] ] }...
          ~~~~~~~   ~~~~      { ~~~~~~~~~~~~~~~~~~        }
                              { NEAREST-AWAY-FROM-ZERO }
                                { ~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { NEAREST-EVEN              }
                              { ~~~~~~~~~~~~~~            }
                              { NEAREST-TOWARD-ZERO       }
                        { ~~~~~~~~~~~~~~~~~~~~~~~~        }
                              { PROHIBITED                }
                              { ~~~~~~~~~~~               }
                              { TOWARD-GREATER            }
                              { ~~~~~~~~~~~~~~~~~         }
                              { TOWARD-LESSER             }
                              { ~~~~~~~~~~~~~~~           }
                              { TRUNCATION                }
                                ~~~~~~~~~~~

    [ ON SIZE ERROR imperative-statement-1 ]
          ~~~~ ~~~~~

    [ NOT ON SIZE ERROR imperative-statement-2 ]
      ~~~      ~~~~ ~~~~~

[ END-SUBTRACT ]
  ~~~~~~~~~~~~
```

This format of the SUBTRACT statement generates the arithmetic sum of all arguments that appear before the FROM (identifier-1 or literal-1) and subtracts that sum from each identifier-2.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be numeric unedited data items.
3. *literal-1* must be a numeric literal.
4. The optional ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause available to each *identifier-2* will control how non-integer results will be saved.
5. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.48.2 SUBTRACT GIVING

SUBTRACT GIVING Syntax

```
SUBTRACT { literal-1          }...   FROM identifier-2
~~~~~~~~ { identifier-1 }            ~~~~

    GIVING { identifier-3
    ~~~~~~
        [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                 } ] ] }...
          ~~~~~~~   ~~~~      { ~~~~~~~~~~~~~~~~             }
                              { NEAREST-AWAY-FROM-ZERO }
                                { ~~~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { NEAREST-EVEN                 }
                              { ~~~~~~~~~~~~~~               }
                              { NEAREST-TOWARD-ZERO          }
                        { ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ }
                              { PROHIBITED                   }
                              { ~~~~~~~~~~~                  }
                              { TOWARD-GREATER               }
                              { ~~~~~~~~~~~~~~~~             }
                              { TOWARD-LESSER                }
                              { ~~~~~~~~~~~~~~~              }
                              { TRUNCATION                   }
                                ~~~~~~~~~~
    [ ON SIZE ERROR imperative-statement-1 ]
          ~~~~ ~~~~~
    [ NOT ON SIZE ERROR imperative-statement-2 ]
      ~~~      ~~~~ ~~~~~
[ END-SUBTRACT ]
  ~~~~~~~~~~~~
```

The SUBTRACT GIVING statement generates the arithmetic sum of all arguments that appear before the FROM (identifier-1 or literal-1), subtracts that sum from the contents of identifier-2 and then replaces the contents of the identifiers listed after the GIVING (identifier-3) with that result.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be numeric unedited data items.
3. *literal-1* must be a numeric literal.
4. *identifier-3* must be a numeric (edited or unedited) data item.
5. The optional ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause available to each *identifier-2* will control how non-integer results will be saved.
6. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.48.3 SUBTRACT CORRESPONDING

SUBTRACT CORRESPONDING Syntax

```
SUBTRACT CORRESPONDING identifier-1 FROM identifier-2
~~~~~~~~                            ~~~~
  [ ROUNDED [ MODE IS { AWAY-FROM-ZERO                } ] ]
     ~~~~~~~    ~~~~     { ~~~~~~~~~~~~~~~~          }
                         { NEAREST-AWAY-FROM-ZERO }
                           { ~~~~~~~~~~~~~~~~~~~~~~~ }
                         { NEAREST-EVEN              }
                         { ~~~~~~~~~~~~~             }
                         { NEAREST-TOWARD-ZERO       }
                   { ~~~~~~~~~~~~~~~~~~~~~~~         }
                         { PROHIBITED                }
                         { ~~~~~~~~~~~               }
                         { TOWARD-GREATER            }
                         { ~~~~~~~~~~~~~~~           }
                         { TOWARD-LESSER             }
                         { ~~~~~~~~~~~~~~            }
                         { TRUNCATION                }
                           ~~~~~~~~~~
  [ ON SIZE ERROR imperative-statement-1 ]
        ~~~~ ~~~~~
  [ NOT ON SIZE ERROR imperative-statement-2 ]
     ~~~      ~~~~ ~~~~~
[ END-SUBTRACT ]
  ~~~~~~~~~~~~
```

———

The SUBTRACT CORRESPONDING statement generates code equivalent to individual SUBTRACT FROM statements for corresponding matches of data items found subordinate to the two identifiers.

1. The reserved words IS and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. Both identifier-1 and identifier-2 must be group items.
3. See Section 7.6.2 [CORRESPONDING], page 254, for information on how corresponding matches will be found between identifier-1 and identifier-2.
4. The optional ROUNDED (see Section 7.6.7 [ROUNDED], page 257) clause available to each *identifier-2* will control how non-integer results will be saved.
5. The optional ON SIZE ERROR and NOT ON SIZE ERROR clauses may be used to detect and react to the failure or success, respectively, of an attempt to perform a calculation. In this case, failure is defined as being an identifier-2 with an insufficient number of digit positions available to the left of any implied decimal point. See Section 7.6.6 [ON SIZE ERROR + NOT ON SIZE ERROR], page 257, for additional information.

7.8.49 SUPPRESS

SUPPRESS Syntax

SUPPRESS PRINTING
~~~~~~~~

———————————————————————————————————————

The SUPPRESS statement causes the presentation of a report group to be suppressed.

1. The reserved word PRINTING is optional and may be omitted. The presence or absence of this word has no effect upon the program.
2. This statement may only appear within a USE BEFORE REPORTING procedure (in DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250)).
3. SUPPRESS only prevents the presentation of the report group within whose USE BEFORE REPORTING procedure the statement occurs.
4. This statement must be executed each time presentation of the report group is to be suppressed.
5. When a report group's presentation is suppressed, none of the following operations for the report will take place:
   A. Actual presentation of the report group in question.
   B. Processing of any LINE (see Section 6.9.30 [LINE], page 185) clauses within the report group in question.
   C. Processing of the NEXT GROUP (see Section 6.9.33 [NEXT GROUP], page 189) clause (if any) within the report group in question.
   D. Any modification to the LINE-COUNTER special register (see Section 7.7 [Special Registers], page 260).
   E. Any modification to the PAGE-COUNTER special register.
~~~~~~~~

7.8.50 TERMINATE

TERMINATE Syntax

TERMINATE report-name-1...
~~~~~~~~~

———————————————————————————————————————

The TERMINATE statement causes the processing of the specified report(s) to be completed.

1. Each report-name-1 must be the name of a report having an RD (see Section 6.6 [REPORT SECTION], page 135) defined for it.
2. The specified report name(s) must be currently initiated (via INITIATE (see Section 7.8.25 [INITIATE], page 343)) and cannot yet have been terminated.
3. The TERMINATE statement will present each CONTROL FOOTING (if any), in reverse sequence of the control hierarchy, starting with the most minor up to FINAL (if any). During the presentation of these groups and the processing of any USE BEFORE REPORTING procedures for those groups, the prior set of control data item values will be available, as though a control break had been detected at the most major control data name.
4. During the presentation of the CONTROL FOOTING groups, any necessary PAGE FOOTING and PAGE HEADING groups will be presented as well.
5. Finally,the REPORT FOOTING group, if any, will be presented.
6. If an INITIATE is followed by a TERMINATE with no intervening GENERATE (see Section 7.8.20 [GENERATE], page 333) statements (all pertaining to the same report, of course), no report groups will be presented to the output file.
~~~~~~~~~

7.8.51 TRANSFORM

TRANSFORM Syntax

```
TRANSFORM identifier-1 CHARACTERS FROM { literal-1            } TO { literal-2        }
~~~~~~~~~                              ~~~~ { identifier-2 } ~~ { identifier-3 }
```

The TRANSFORM statement scans a data item performing a series of mono-alphabetic substitu- tions, defined by the arguments before and after the TO clause.

1. Both literal-1 and/or literal-2 must be alphanumeric literals.
2. All of identifier-1, identifier-2 and identifier-3 must either be group items or alphanumeric data items. Numeric data items with a USAGE (see Section 6.9.59 [USAGE], page 228) of DISPLAY are accepted, but will generate warning messages from the compiler.
3. The TRANSFORM statement will replace characters within identifier-1 that are found in the string specified before the TO keyword with the corresponding characters from the string specified after the TO keyword.
4. Usage of word CHARACTERS has no effect on operations other than for appearance.
5. This statement exists within GnuCOBOL to provide compatibility with COBOL programs written to pre-1985 standards. The TRANSFORM statement was made obsolete in the 1985 standard of COBOL, having been replaced by the CONVERTING clause of the INSPECT statement (see Section 7.8.26 [INSPECT], page 344). New programs should be coded to use INSPECT CONVERTING rather than TRANSFORM.

7.8.52 UNLOCK

UNLOCK Syntax

```
UNLOCK filename-1 RECORD|RECORDS
```

This statement synchronizes any as-yet unwritten file I/O buffers to the specified file (if any) and releases any record locks held for records belonging to file-name-1.

1. The reserved words RECORD and RECORDS are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. If file-name-1 is a Sort/Merge work file, no action will be taken.
3. Not all GnuCOBOL implementations support locking. Whether they do or not depends upon the operating system they were built for and the build options that were used when GnuCOBOL was generated. When a program using one of those GnuCOBOL implementations issues an UNLOCK, it will ignored. There will be no compiler message issued. Buffer syncing, if needed, will still occur.
4. For GnuCOBOL UNLOCK is implied in GnuCOBOL on file close so there's no use to do it afterwards. A CLOSE will always trigger syncing the file to disk, too.
5. Doing UNLOCK before a close, will explicit unlock any records with a lock when running on INDEXED files, for other files it will release any locks on the file if it wasn't opened for exclusive locking and will trigger syncing to disk (not done for any INDEXED file).
6. When using Linux and for that matter most *nix platforms, the system maintains it's own cache and buffers for file processing so there can and most likely will be a short delay before all data is written out to disk.
7. See Section 2.1.30.2 [Record Locking], page 58, for additional information on record locking.

7.8.53 UNSTRING

UNSTRING Syntax

```
UNSTRING identifier-1
~~~~~~~~
      DELIMITED BY {    [ ALL ] literal-1    } [ OR { [ ALL ] literal-2  } ]...
      ~~~~~~~~~~   {    ~~~                  }    ~~ {    ~~~              }
                   { identifier-2             }       { identifier-3       }

      INTO  { identifier-4
      ~~~~  [ DELIMITER IN                   ] [ COUNT IN identifier-6    ] }...
            identifier-5
              ~~~~~~~~~~                          ~~~~~
    [ WITH  POINTER identifier-7 ]
            ~~~~~~~
    [ TALLYING IN identifier-8 ]
      ~~~~~~~~
    [ ON OVERFLOW imperative-statement-1 ]
         ~~~~~~~~
    [ NOT ON OVERFLOW imperative-statement-2 ]
      ~~~     ~~~~~~~~
[ END-UNSTRING ]
  ~~~~~~~~~~~~
```

The UNSTRING statement parses a string, extracting any number of sub strings from it.

1. The reserved words BY, IN and ON are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. *identifier-1* through identifier-5 must be explicitly or implicitly defined with a USAGE (see Section 6.9.59 [USAGE], page 228) of DISPLAY. Any of those identifiers may be group items.
3. Both literal-1 and literal-2 must be alphanumeric literals.
4. Each of identifier-6, identifier-7 and identifier-8 must be elementary non-edited integer numeric items.
5. At the time the UNSTRING statement begins execution, identifier-7 must have a value greater than 0.
6. *identifier-1* will be referred to as the source string and each identifier-4 will be referred to as a destination field in the following discussions.
7. The UNSTRING statement's processing is based upon a current character pointer, the initial value of which will be the value of identifier-7 at the time the UNSTRING statement began execution. If no POINTER clause is coded, a value of 1 (meaning "the 1st character position") will be assumed for the current character pointer's initial value.
8. The source string will be parsed into sub strings starting from the current character pointer position. Sub strings are identified by using the various delimiter strings specified on the DELIMITED BY clause as inter-sub string separators.
9. Using the ALL option allows a delimiter sequence to be an arbitrarily long sequence of occurrences of the delimiter literal whereas its absence treats each occurrence as a separate delimiter. When multiple delimiters are specified, they will be looked for in the source string in the sequence in which they are coded.

10. Two consecutive delimiter sequences will identify a null sub string.
11. Identified sub strings will be moved into each destination field in the sequence they are identified; values moved into a destination field will be truncated if the sub string length exceeds the destination field length, or padded with spaces if the destination field length exceeds the sub string length. Both truncation and padding will be controlled by the presence or absence of a JUSTIFIED (see Section 6.9.27 [JUSTIFIED], page 181) clause on the destination field.
12. Each destination field may have an optional DELIMITER clause. If a DELIMITER clause is specified, identifier-5 will have the delimiter character string used to identify the sub string for the destination field moved into it. If a destination field was not altered (because an insufficient number of sub strings were identified), identifier-5 for that destination field will also be unchanged.
13. Each destination field may have an optional COUNT clause. If a COUNT clause is specified, identifier-6 will have the size of the sub string (in characters) for the destination field moved into it. If a destination field was not altered (because an insufficient number of sub strings were identified), identifier-6 for that destination field will also be unchanged. The COUNT/identifier-6 will be greater than the size of the destination field when the sub string length exceeds the destination field length. This condition is not detected by the ON OVERFLOW clause.
14. If a TALLYING clause is coded, identifier-8 will be incremented by 1 each time a destination field is populated.
15. None of identifier-4, identifier-5, identifier-6, identifier-7 or identifier-8 are initialized by the UNSTRING statement. You need to do that yourself via a MOVE (see Section 7.8.30 [MOVE], page 354) or INITIALIZE statement (see Section 7.8.24 [INITIALIZE], page 339).
16. UNSTRING processing will cease when one of the following occurs:
    A. The initial value of the current character pointer is less than 1 or greater than the number of character positions in identifier-1, or. . .
    B. All destination fields have been fully processed
17. If event A occurs, none of the destination field contents (or the contents of their DELIMITER or COUNT identifiers) will be changed.
18. An overflow condition exists if either event A occurs, or if event B occurs with at least one character position in identifier-1 not having been processed.
19. The optional ON OVERFLOW and NOT ON OVERFLOW clauses may be used to detect and react to the occurrence or not, respectively, of an overflow condition. See Section 7.6.5 [ON OVERFLOW + NOT ON OVERFLOW], page 256, for additional information.

The following sample program illustrates the UNSTRING statement statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
DEMOUNSTRING. DATA
DIVISION.
WORKING-STORAGE SECTION.
01  Full-Name                  PIC X(40).
01  Parsed-Info.
    05 Last-Name               PIC X(15).
    05 First-Name              PIC X(15).
    05 MI                      PIC X(1).
    05 Delim-LN                PIC X(1).
    05 Delim-FN                PIC X(1).
    05 Delim-MI                PIC X(1).
```

```
        05 Count-LN                     BINARY-CHAR.
        05 Count-FN                     BINARY-CHAR.
        05 Count-MI                     BINARY-CHAR.
        05 Tallying-Ctr                 BINARY-CHAR.
    PROCEDURE DIVISION.
    P1. PERFORM UNTIL EXIT
          DISPLAY "Enter Full Name (null quits):" WITH NO
               ADVANCING
          ACCEPT Full-Name
          IF Full-Name = SPACES
             EXIT PERFORM
          END-IF
          INITIALIZE Parsed-Info
          UNSTRING Full-Name
             DELIMITED BY ", "
                           OR ","
                           OR ALL SPACES
             INTO Last-Name
                           DELIMITER IN Delim-LN
                           COUNT IN Count-LN
                     First-Name
                           DELIMITER IN Delim-FN
                           COUNT IN Count-FN
                     MI
                           DELIMITER IN Delim-MI
                           COUNT IN Count-MI
             TALLYING Tallying-Ctr
        DISPLAY "First-Name=" First-
        Name
                  " Delim='"        Delim-FN
                  "'  Count="       Count-FN
        DISPLAY "MI              =" MI "                    "
                  " Delim='"        Delim-MI
                  "'  Count="       Count-MI
        DISPLAY "Last-Name =" Last-Name "
                  Delim='"          Delim-LN
                  "' Count="        Count-LN
        DISPLAY "Tally=          " Tallying-Ctr
      END-PERFORM
      DISPLAY "Bye!"
      STOP RUN   .
```

The following is sample output from the program:

```
Enter Full Name (null quits):Cutler, Gary L First-
Name=Gary                    Delim=' '
Count=+004
MI          =L               Delim=' ' Count=+001
Last-Name =Cutler            Delim=',' Count=+006
Tally=        +003
Enter Full Name (null quits):Snoddgrass,Throckmorton,P
First-Name=Throckmorton      Delim=',' Count=+012
MI          =P               Delim=' ' Count=+001
```

```
Last-Name =Snoddgrass          Delim=','  Count=+010
```

```
Tally=        +003
Enter Full Name (null quits):Munster  Herman First-
Name=Herman                  Delim=' ' Count=+006
MI          =                Delim=' ' Count=+000
Last-Name =Munster           Delim=' ' Count=+007
Tally=        +002
Enter Full Name (null quits):
Bye!
```

7.8.54 WRITE

WRITE Syntax

```
WRITE { record-name-1      [ FROM { identifier-1  } ] }
 ~~~~~ {                   [ ~~~~ { literal-1     } ] }
       {                                              }
       { FILE file-name-    FROM { identifier-1  }    }
       1
       { ~~~~               ~~~~ { literal-1     }    }

    [ { BEFORE } ADVANCING { { literal-2  } LINE|LINES } ]
     [ { ~~~~~~ }            { { identifier-2              } ]
    [ { AFTER  }            { PAGE                        } ]
     [    ~~~~~              { ~~~~                        } ]
                             { mnemonic-name-1             } ]

    [ WITH [ NO ] LOCK ]
              ~~     ~~~~

    [ AT END-OF-PAGE|EOP imperative-statement-1 ]
          ~~~~~~~~~~~ ~~~
    [ NOT AT END-OF-PAGE|EOP imperative-statement-2 ]
      ~~~      ~~~~~~~~~~~ ~~~
    [ INVALID KEY imperative-statement-3 ]
      ~~~~~~~
    [ NOT INVALID KEY imperative-statement-4 ]
      ~~~ ~~~~~~~
[ END-WRITE ]
  ~~~~~~~~~
```

The WRITE statement writes a new record to an open file.

1. The reserved words ADVANCING, AT, KEY, LINE, LINES and WITH are optional and may be omitted. The presence or absence of these words has no effect upon the program.
2. The reserved words END-OF-PAGE and EOP are interchangeable.
3. The record-name-1 specified on the statement must be defined as an 01-level record subordinate to the File Description (FD (see Section 6.2.1 [File/Sort-Description], page 123)) of a file that is currently open for OUTPUT (see [File OPEN Modes], page 359), EXTEND or I-O.
4. The optional FROM clause will cause literal-1 or identifier-1 to be automatically moved into record-name-1 prior to writing record-name-1's contents to the appropriate file. If this clause is not specified, it is the programmer's responsibility to populate record-name-1 with the desired data prior to executing the WRITE.
5. The optional LOCK options may be used to manually control access to the just-written record by other programs while this program is running. See Section 2.1.30.2 [Record Locking], page 58, to review the various record locking behaviour.
6. The optional INVALID KEY and NOT INVALID KEY clauses may be used when writing to relative or indexed files to detect and react to the failure (non-zero file status code) or success (00 file status code), respectively, of the statement. See [File Status Codes], page 105, for additional information.

7. When WRITE is used against an ORGANIZATION LINE SEQUENTIAL (see Section 5.2.1.2 [ORGANIZATION LINE SEQUENTIAL], page 110) file, with or without the LINE ADVANCING (see [LINE ADVANCING], page 15) option, an end-of-record delimiter character sequence will be written to the file to signify where one record ends and the next record begins. This delimiter sequence will be either of the following:
   - A line-terminator sequence consisting of an ASCII carriage-return/line-feed character sequence (X'0D0A') if you are running a MinGW or native Windows build of Gnu- COBOL
   - A line-terminator sequence consisting of an ASCII line-feed character (X'0A') if you are running a Cygwin, Linux, Unix or OSX build of GnuCOBOL
8. The following points pertain to the use (or not) of the ADVANCING clause:
   A. Using this clause with any organization other than ORGANIZATION LINE SEQUENTIAL will either be rejected outright by the compiler (relative or indexed files) or may introduce unwanted characters into the file (ORGANIZATION SEQUENTIAL (see Section 5.2.1.1 [ORGANIZATION SEQUENTIAL], page 109)).
   B. If no ADVANCING clause is specified on a WRITE to a line-advancing file, AFTER ADVANCING 1 LINE will be assumed; on other than line-advancing files, BEFORE ADVANCING 1 LINE will be assumed.
   C. When BEFORE ADVANCING is used (or implied), the record is written to the file before the ADVANCING action writes line-terminator characters to the file.
   D. If AFTER ADVANCING is used (or implied), the ADVANCING action writes line-terminator characters to the file and then the record data is written to the file.
   E. The ADVANCING n LINES clause will introduce the specified number of line-terminator character sequences into the file either before the written record (AFTER ADVANCING) or after the written record (BEFORE ADVANCING).
   F. If the LINAGE (see Section 6.2.1 [File/Sort-Description], page 123) clause is absent from the file's FD:
      a. The ADVANCING PAGE clause will introduce an ASCII formfeed character into the file either before the written record (AFTER PAGE) or after the written record (BEFORE PAGE).
      b. Management of areas on the printed page such as top-of page headers, bottom- of-page footers, dealing with "full page" situations and the like are the complete responsibility of the programmer.
   G. If the LINAGE clause is present in the file's FD:
      a. The ADVANCING PAGE clause will introduce the appropriate number of line-terminator character sequences into the file either before the written record (AFTER ADVANCING) or after the written record (BEFORE ADVANCING) so as to force the printer to automatically advance to a new sheet of paper when the file prints. No formfeed characters will be generated when LINAGE is specified — instead, it is assumed that the printer to which the report will be printed will be loaded with special forms that conform to the specifications defined by the LINAGE clause.
      b. Management of areas on the printed page such as top-of page headers, bottom-of-page footers, dealing with "full page" situations and the like are now the joint responsibility of the programmer and the GnuCOBOL run-time library, which provides tools such as the LINAGE-COUNTER special register (see Section 7.7 [Special Registers], page 260) and the END-OF-PAGE clause to deal with page formatting issues.

c. The AT END-OF-PAGE clause will be triggered, thus executing imperative-statement-1 (see [Imperative Statement], page 758), if the WRITE statement introduces a data line or line-feed character into the file at a line position within the Page Footer area defined by the LINAGE clause. The NOT AT END-OF-PAGE clause will be triggered (thus executing imperative-statement-2) if no end-of-page condition occurred during the WRITE.

7.8.55 XML GENERATE

XML GENERATE Syntax

```
XML GENERATE identifier-1 FROM identifier-2
~~~ ~~~~~~~~~                 ~~~~
      [COUNT IN identifier-3]
        ~~~~~
     [ WITH ENCODING codepage ]
              ~~~~~~~~
     [ WITH XML-DECLARATION ]
              ~~~~~~~~~~~~~~~
     [ WITH ATTRIBUTES
     ]
              ~~~~~~~~~~
     [NAMESPACE IS {identifier-4 }[ NAMESPACE-PREFIX IS {identifier-5 }]
      ~~~~~~~~~~    {literal-4      }  ~~~~~~~~~~~~~~~~~  {literal-5      }]]

     [NAME OF     {identifier-6 IS literal-6 } ...      ]
      ~~~~
     [TYPE OF     {identifier-7 IS {ATTRIBUTE|ELEMENT|CONTENT}} ...]
      ~~~~                                 ~~~~~~~~~ ~~~~~~~ ~~~~~~~

     [SUPPRESS {identifier-8 [when-phrase] }   ...    ]
      ~~~~~~~~~ {generic-suppression-phrase }

     [ ON EXCEPTION imperative-statement-1 ]
            ~~~~~~~~~~
     [ NOT ON EXCEPTION imperative-statement-2 ]
        ~~~      ~~~~~~~~~~
[ END-XML ]
  ~~~~~~~
```

when-phraseFormat

```
WHEN   {      ZERO      }   [ [ OR ] {      ZERO      }  ] ...
~~~~   {     ZEROES     }            {     ZEROES     }
       {      ZEROS     }            {      ZEROS     }
       {      SPACE     }            {      SPACE     }
       {     SPACES     }            {     SPACES     }
       {  LOW-VALUE     }            {  LOW-VALUE     }
       {  LOW-VALUES    }            {  LOW-VALUES    }
       {  HIGH-VALUE    }            {  HIGH-VALUE    }
       {  HIGH-VALUES   }            {  HIGH-VALUES   }
```

Generic-suppression-phraseFormat

```
[[EVERY   {NUMERIC     [ATTRIBUTE|ELEMENT|CONTENT]  } ]     when-phrase    ]
          {NONNUMERIC [ATTRIBUTE|ELEMENT|CONTENT] }
          {ATTRIBUTE                                }
          {CONTENT                                  }
```

{ELEMENT }

———————————————————————————————————————

The XML GENERATE statement creates an XML document based on the structure and content of the input data item. It takes as input a source data item to generate and store the corresponding XML document in a target data item. Both the source and receive data items must be declared in program's Data Division. The XML generated document follows syntax rules described at http://www.w3.org/TR/xml.

GENERATE.

1. Identifier-1 is an alphanumeric data item, it will contain the generated XML document output. It must be large enough to contain the generated XML document. Typically, it should be from five to ten times the size of identifier-2, depending on the length of the data-name or data-names within identifier-2. If identifier-1 is not large enough, an error condition exists at the end of the XML GENERATE statement (see XML-CODE).
2. If identifier-1 is longer than the generated XML document, only the initial part of identifier-1 changes. The rest of identifier-1 will contain the data that was present before this execu- tion of the XML GENERATE statement. To avoid referring to that data, either initialize identifier-1 to spaces before the XML GENERATE statement or specify the COUNT IN phrase.
3. The element tag names in the XML documents generated from identifier-2 are derived from the name of the data item specified by identifier-2 and from any eligible data-names that are subordinate to identifier-2. The following rules apply:
   a. The exact mixed-case spelling of data-names from the data description entry is retained. The spellings from any references to that data item (for example, in an OCCURS DEPENDING ON clause) are not used.
   b. Data-names beginning with a digit are prefixed by an underscore. For example, the data-name 4C becomes XML tag name 4C.
   c. Names of data items that contain characters that are illegal in XML version 1.0 are prefixed by hex., and the content itself is expressed in hexadecimal.
   d. The XML declaration (header) is not generated. No extra white space, new line, etc. is inserted to make the generated XML more readable.

FROM.

4. The group or elementary data item identifier-2 is to be converted to the XML document. identifier-2 must not be described with the RENAMES clause. The following data items specified by identifier-2 are ignored by the XML GENERATE statement:
   a. Any unnamed or FILLER data item with its subordinates.
   b. Any elementary data item that assumes a pointer, e.g. USAGE POINTER.
   c. Any data item (with its subordinates) subordinate to identifier-2 that is described with the REDEFINES clause.
   d. Any group item (with its subordinates) subordinate to identifier-2 that contains a non-unique name within the same level.
   e. Any group item that does not contain at least one elementary data item of category alphabetic, alphanumeric, numeric, or index.
   f. The content of each elementary data item within identifier-2 is converted to character format.

   g. Alphabetic, alphanumeric, alphanumeric-edited, external floating-point, and numeric-edited items are not converted.
5. Fixed-point numeric data items are converted as if they were moved to a numeric-edited item that has:
   a. An explicit decimal point, if the numeric item has at least one decimal position.
   b. The same number of decimal positions as the numeric item.
   c. A leading '-' picture symbol if the data item is signed and has an S in its PICTURE clause.
6. For COMPUTATIONAL-5 (COMP-5) binary data items, the number of integer positions depends on the number of '9' symbols in the picture character string. If the data item has one to four '9' picture symbols, the number of integer positions is five minus the number of decimal places. If the data item has five to nine '9' picture symbols, the number of integer positions is ten minus the number of decimal places. If the data item has 10 to 18 '9' picture symbols, the number of integer positions is 20 minus the number of decimal places.
7. All other fixed-point numeric data items will have as many integer positions as the numeric item, but with at least one integer position.
8. Internal floating-point data items are converted as if they were moved to a data item as follows:
   a. For COMP-1: an external floating-point data item with PICTURE -9.9(8)E+99.
   b. For COMP-2: an external floating-point data item with PICTURE -9.9(17)E+99 (illegal because of the number of digit positions).
9. Index data items are converted as if they were declared USAGE COMP-5 PICTURE S9(9). After conversion, leading and trailing spaces and leading zeroes are removed, as described under Data trimming.
10. After conversion, if a data item contains characters that are illegal in XML, the value in the data item before conversion or trimming is represented in hexadecimal, and an element tag name with the prefix "hex." is substituted for the regular tag name. For example, if data item Customer-Name is found at run time to contain LOW-VALUES, the XML element tag name 'hex.Customer-Name' is used instead of the normal 'Customer-Name', and the content is represented as a string of pairs of zero digits.
11. Any remaining instances of the five special characters & (ampersand), (apostrophe), > (greater-than sign), < (less-than sign), and (quotation mark) are converted into the equivalent XML references: &, ', >, <, and ", respectively.
12. Trailing spaces and leading zeroes are eliminated. Values converted from signed numeric val- ues have their leading space removed if the value is positive. Values converted from numeric items have leading zeroes eliminated (after any initial minus sign). This is up to but not including the digit immediately before the actual or implied decimal point. Trailing zeroes after a decimal point are retained. For example: 012.340 becomes -12.340; 0000.45 becomes
    0.45 ; 0013 becomes 13 ; 0000 becomes 0. Character values from alphabetic, alphanumeric data items have either trailing or leading spaces removed, depending on whether the corresponding data items have left or right justification, respectively left being the default. Trailing spaces are removed from values whose corresponding data items do not specify the JUSTIFIED clause. Leading spaces are removed from values whose data items do specify the JUSTIFIED clause. If a character value consists solely of spaces, all spaces are removed but one.

    COUNT IN.
13. Identifier-3 contains the count of generated XML characters. It must be an integer data item and normally contains a length of the generated XML document in bytes.

14. Use the COUNT IN phrase to determine the total number of character positions, in bytes, that were generated. identifier-3 will then contain this information after XML GENERATE executes. You can use identifier-3 as a reference modification length field to refer to the part of identifier-2 that contains the generated XML document.
15. Identifier-1, identifier-2, identifier-3 must not overlap each other.

    XML-CODE.
16. After execution of the XML GENERATE statement, the special register XML-CODE contains either zero, which indicates successful completion, or a non-zero exception code.

Exception code (decimal)	Description
0	The receiver contains the successfully generated XML document. The COUNT [IN] data item contains the count of character positions in the generated XML document.
400	The receiver was too small to contain the generated XML document. The COUNT IN data item, if specified, contains the count of character positions that were actually generated.
401	A DBCS data-name contained a character that, when converted to Unicode, was not valid in an XML element or attribute name.
402	The first character of a DBCS data-name, when converted to Unicode, was not valid as the first character of an XML element or attribute name.
403	The value of an OCCURS DEPENDING ON variable exceeded 16,777,215.
410	The CCSID (coded character set identifier) page specified by the CODEPAGE compiler option is not supported for conversion to Unicode.
411	The CCSID (coded character set identifier) specified by the CODEPAGE compiler option is not a supported single-byte CCSID.
414	The CCSID (coded character set identifier) specified for the XML document was invalid or was not supported.
415	The receiver was national, but the encoding specified for the document was not UTF-16.
416	The XML name space identifier contained invalid XML characters.
417	Element character content or an attribute value contained characters that are illegal in XML content. XML generation has continued, with the element tag name or the attribute name prefixed with 'hex.' and the original data value represented in the document in hexadecimal.
418	Substitution characters were generated by encoding conversion.
419	The XML name space prefix was invalid.
420	The receiver was alphanumeric and the input included national or DBCS data or names, but the encoding specified for the document was not 1208.
600 - 699	Internal error. Report the error to your service representative.

17. The XML-CODE special register has the implicit definition

    01 XML-CODE PICTURE S9(9)USAGE BINARY VALUE 0.
18. If an error (XML-CODE NOT = 0) occurs during generation of the XML document, control is passed to the conditional statement imperative-statement-1. If ON EXCEPTION is not specified control is passed to the end of the XML GENERATE statement.

19. If no error (XML-CODE = 0) occurs during generation of the XML document, control is passed to the conditional statement imperative-statement-2. If NOT ON EXCEPTION is not specified, control is passed to the end of the XML GENERATE statement.
20. Example of the above :

```
01    wXML-DOCUMENT pic x(512).
01    wCHAR-COUNT      pic  9(9)  binary.
01    wSOURCE.
      05    wA        pic x(10) value "aaa".
      05    wB.
            10    wC pic x(10) value "ccc".
            10    wD  pic x(10) value "ddd".
      05    wE        pic x(10) value "eee".
      -  -  -

XML  GENERATE  wXML-DOCUMENT
       FROM          wSOURCE
       COUNT IN wCHAR-COUNT
  ON EXCEPTION
         DISPLAY 'XML generation error ' XML-CODE
         STOP RUN
  NOT ON EXCEPTION
         DISPLAY 'XML document generated.' wXML-DOCUMENT(1:wCHAR-COUNT)
  END-XML
```

The code generates the following XML document:

```
<wSOURCE><wA>aaa</wA><wB><wC>ccc</wC><wD>ddd</wD></wB><wE>eee</wE></wSOURCE>
```

ENCODING, XML-DECLARATION

21. You can code the ENCODING phrase of the XML GENERATE statement to specify the coded character set identifier CCSID of the generated XML document. If you do not use the ENCODING phrase, the document encoding is determined by the category of the receiving data item and by the CODEPAGE compiler option. The XML-DECLARATION phrase causes the generated XML document to have an XML declaration that includes version information and an encoding declaration.

```
01    MESSAGE.
      05    MSG  PIC X(80) VALUE 'Hello COBOL!'

      XML GENERATE OUTPUT FROM MESSAGE
            WITH ENCODING 1208
            WITH XML-DECLARATION
```

The code above generates the following XML document:

```
<?XML VERSION=1.0" ENCODING=UTF-8"?><MESSAGE><MSG>Hello COBOL!</MSG></MESSAGE>
```

If you do not code the XML-DECLARATION phrase, an XML declaration is not generated.

```
CCSID        Description
------      ------------

1208        UTF-8
1047        Latin 1 / Open Systems
1140, 37    USA, Canada, . . . Euro Country Extended Code Page (ECECP),
                                   Country Extended Code Page (CECP)
1141, 273   Austria,  Germany ECECP, CECP
1142, 277   Denmark,  Norway ECECP, CECP
1143, 278   Finland,  Sweden ECECP, CECP
1144, 280   Italy ECECP, CECP
1145, 284   Spain, Latin America (Spanish) ECECP, CECP
1146, 285   UK ECECP, CECP
1147, 297   France ECECP, CECP
1148, 500   International  ECECP,  CECP
1149, 871   Iceland  ECECP,  CECP
```

ATTRIBUTES.

22. You can use the ATTRIBUTES phrase of the XML GENERATE statement to have each elementary data item included in the generated XML document to be expressed as an attribute of the XML element that corresponds to its immediately superordinate data item, rather than as a child element.

```
01   wXML-INPUT.
     05   wNAME       PIC X(10) VALUE "PETER".
     05   wSALARY.
          10 wBASIC   PIC X(05) VALUE "20000".
          10 wHRA     PIC X(03) VALUE "3000".
     05   wDEPT       PIC X(03) VALUE "TRADE".

  XML    GENERATE XML-OUT FROM wXML-INPUT
                                WITH ATTRIBUTE.
```

23. The code would then generate the following XML document, in which wNAME and wDEPT are expressed as attributes of element XML-INPUT and wBASIC and wHRA become at- tributes of element wSALARY

```
<wXML-INPUT wNAME=PETER wDEPT=TRADE>
<wSALARY wBASIC=20000" wHRA=3000"></wSALARY>
</wXML-INPUT>
```

NAMESPACE and NAMESPACE-PREFIX

24. Use the NAMESPACE phrase to identify a namespace for the generated XML document. If the NAMESPACE phrase is not specified, or if identifier-4 has length zero or contains all spaces, the element names of XML documents produced by the XML GENERATE statement are not in any namespace.
25. Use the NAMESPACE-PREFIX phrase to qualify the start and end tag of each element in the generated XML document with a prefix.

26. If the NAMESPACE-PREFIX phrase is not specified, or if identifier-5 is of length zero or contains all spaces, the namespace specified by the NAMESPACE phrase specifies the default namespace for the document. In this case, the namespace declared on the root element applies by default to each element name in the document, including that of the root element. (Default namespace declarations do not apply directly to attribute names).
27. If the NAMESPACE-PREFIX phrase is specified, and identifier-5 is not of length zero and does not contain all spaces, then the start and end tag of each element in the generated document is qualified with the specified prefix. The prefix should therefore prefer- ably be short. When the XML GENERATE statement is executed, the prefix must be a valid XML name, but without the colon (:), as defined in Namespaces in XML 1.0. at http://www.w3.org/TR/xml-names/#ns-decl. The prefix can have trailing spaces, which are removed before use.

    Identifier-4, literal-4; identifier-5, literal-5
28. Identifier-4, literal-4: The namespace identifier, which must be a valid Uniform Re- source Identifier (URI) as defined in Uniform Resource Identifier (URI): Generic Syntax at http://www.rfc-editor.org/rfc/rfc3986.txt.
29. Identifier-5,literal-5: The namespace prefix, which serves as an alias for the namespace identifier.
30. Identifier-4 and identifier-5 must reference data items of category alphanumeric.
31. Identifier-4 and identifier-5 must not overlap identifier-1 or identifier-3.
32. Literal-4 and literal-5 must be of category alphanumeric and must not be figurative constants.
33. For full details about namespaces, see Namespaces in XML 1.0 at http://www.w3.org/TR/xml-names/#ns-decl.

```
01 MESSAGE.
    05 MSG PIC X(80) VALUE 'Hello COBOL!'.
01 NAME-SPACE PIC X(10) VALUE "HTTP://GNUCOBOL.COM".

XML GENERATE OUTPUT FROM MESSAGE
      WITH ENCODING 1208
      WITH XML-DECLARATION
```

The code above generates the following XML document:

```
<MESSAGE XMLNS=HTTP://GNUCOBOL><MSG>Hello COBOL!</MSG></MESSAGE>
```

If you do not specify a namespace, the element names in the generated XML document are not in any namespace.

```
01   wRegards.
     05   msg   pic x(80)   value 'Hello,COBOL!'.
01   NS         pic x(20)   value 'http://sample'.
01   NP         pic x(5)    value 'pfx'.
---

XML Generate Doc from Greeting
      namespace          is NS
```

```
namespace-prefix is NP

<pfx:wRegards xmlns:pfx="http://sample"><pfx:msg>Hello,COBOL!</pfx:msg></pfx:wRegar
```

NAME.

34. Allows you to supply element and attribute names.
35. Identifier-6 must reference identifier-2 or one of its subordinate data items. It cannot be a function identifier and cannot be reference modified or subscripted. It must not specify any data item which is ignored by the XML GENERATE statement. For more information about identifier-2, see the description of identifier-2. If identifier-6 is specified more than once in the NAME phrase, the last specification is used.
36. Literal-6 must be an alphanumeric literal containing the attribute or element name to be generated in the XML document corresponding to identifier-6. It must be a valid XML local name.

    TYPE.

37. Allows you to control attribute and element generation.
38. Identifier-7 must reference an elementary data item that is subordinate to identifier-2. It cannot be a function identifier and cannot be reference modified or subscripted. It must not specify any data item which is ignored by the XML GENERATE statement. For more information about identifier-2, see the description of identifier-2. If identifier-7 is specified more than once in the TYPE phrase, the last specification is used.
    1. If the XML GENERATE statement also includes a WITH ATTRIBUTES phrase, the TYPE phrase has precedence for identifier-7.
    2. When ATTRIBUTE is specified, identifier-7 must be eligible to be an XML attribute. identifier-7 is expressed in the generated XML as an attribute of the XML element immediately superordinate to identifier-7 rather than as a child element.
    3. When ELEMENT is specified, identifier-7 is expressed in the generated XML as an element. The XML element name is derived from identifier-7 and the element character content is derived from the converted content of identifier-7 as described in Operation of XML GENERATE.
    4. When CONTENT is specified, identifier-7 is expressed in the generated XML as element character content of the XML element that corresponds to the data item immediately superordinate to identifier-7. The value of the element character content is derived from the converted content of identifier-7 as described in Operation of XML GENERATE. When CONTENT is specified for multiple identifiers all corresponding to the same superordinate identifier, the multiple contributions to the element character content are concatenated.
39. The exact mixed-case spelling of data-names from the data description entry is retained.
40. You likely want to change the COBOL variable names to all lower case or use the NAME phrase for all data entries (at least if you expect it to be all in lower case as in your "target sample").

```
IDENTIFICATION DIVISION.
PROGRAM-ID. XMLPROG.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TOTAL-CHAR PIC 9(05) VALUE 0.
01 x PIC X(200).
```

```
01 y.
    03 z PIC X(15) VALUE "hello,COBOL!".
    03 az PIC X(15) VALUE "goodbye, COBOL!".
    03 ab.
        05 abc PIC x(03) value spaces.

PROCEDURE DIVISION.
XML GENERATE x FROM y
      COUNT IN TOTAL-CHAR
      WITH XML-DECLARATION
      NAME OF abc IS "ABCDEF",
                  z IS "zeta" TYPE
      OF          z IS ATTRIBUTE
      SUPPRESS WHEN SPACES
END-XML

DISPLAY "0. [" TOTAL-CHAR "] COUNT"
DISPLAY "1. [" FUNCTION TRIM(x) "]"

XML GENERATE x FROM abc,
      NAME OF abc IS "sfdasd"
DISPLAY "0. [" TOTAL-CHAR "] COUNT"
DISPLAY "2. [" FUNCTION TRIM(x) "]"

XML GENERATE x FROM ab,
      WITH ATTRIBUTES.
DISPLAY "0. [" TOTAL-CHAR "] COUNT"
DISPLAY "3. [" FUNCTION TRIM(x) "]"
DISPLAY "4. [" x(1:TOTAL-CHAR) "]".
```

SUPPRESS

41. Optionally, you can code the SUPPRESS phrase to specify whether individual data items are generated based on whether or not they meet certain criteria. If the SUPPRESS phrase is specified, identifier-1 must be large enough to contain the generated XML document before any suppression.
42. With the generic-suppression-phrase, elementary items subordinate to identifier-2 that are not otherwise ignored by XML GENERATE operations are identified generically for poten- tial suppression.
43. Either items of class numeric, if the NUMERIC keyword is specified, or items that are not of class numeric, if the NONNUMERIC keyword is specified, or both, might be suppressed.
44. If the ATTRIBUTE keyword is specified, only items that would be expressed in the gener- ated XML document as an XML attribute are identified for potential suppression.
45. If the ELEMENT keyword is specified, only items that would be expressed in the generated XML document as an XML element are identified for potential suppression.
46. If the CONTENT keyword is specified, only items that would be expressed in the generated XML document as element character content of the XML element corresponding to the data item superordinate to the CONTENT data item are identified for potential suppression.
47. If multiple generic-suppression-phrase are specified, the effect is cumulative.
48. Identifier-8 explicitly identifies items for potential suppression.

49. If the WHEN phrase is specified, identifier-8 must reference an elementary data item that is subordinate to identifier-2 and that is not otherwise ignored by the XML GENERATE operations. identifier-8 cannot be a function identifier and cannot be reference modified or subscripted.
50. If the WHEN phrase is omitted, identifier-8 can reference not only an elementary data item but also a group data item. That group data item and all data items that are subordinate to the group item are suppressed. If identifier-8 is specified more than once in the SUPPRESS phrase, the last specification is used. The explicit suppression specification for identifier-8 overrides the suppression specification that is implied by any generic-suppression-phrase, if identifier-8 is also one of the identifiers generically identified.
51. If identifier-8 is specified, the following rules apply to it:
    A. If ZERO, ZEROES, or ZEROS is specified in the WHEN phrase, identifier-8 must not be of USAGE DISPLAY
    B. If SPACE or SPACES is specified in the WHEN phrase, identifier-8 must be of USAGE DISPLAY, DISPLAY-1, or NATIONAL. If identifier-8 is a zoned or national decimal item, it must be an integer.
    C. If LOW-VALUE, LOW-VALUES, HIGH-VALUE, or HIGH-VALUES is specified in the WHEN phrase, identifier-8 must be of USAGE DISPLAY. If identifier-8 is a zoned decimal item, it must be an integer.
52. If the generic-suppression-phrase is specified, data items are selected for potential suppres- sion according to the following rules:
    A. If ZERO, ZEROES, or ZEROS is specified in the WHEN phrase, all data items except those that are defined with USAGE DISPLAY-1 are selected.
    B. If SPACE or SPACES is specified in the WHEN phrase, data items of USAGE DISPLAY, DISPLAY-1, or NATIONAL are selected. For zoned or national decimal items, only integers are selected.
    C. If LOW-VALUE, LOW-VALUES, HIGH-VALUE, or HIGH-VALUES is specified in the WHEN phrase, data items of USAGE DISPLAY are selected. For zoned decimal items, only integers are selected.
53. When the SUPPRESS phrase is specified, a group item subordinate to identifier-2 is suppressed in the generated XML document if all the eligible items subordinate to the group item are suppressed or if, after suppressing any subordinate items, the XML corresponding to the group item would be an empty element with no attributes. The root element is always generated, even if all the items subordinate to identifier-2 are suppressed.

```
01 wG.
    02 HiddenInfo.
        03 SSN pic x(11) value '123-45-6789'.
        03 HomeAddress pic x(50) value '123 Main St, Anytown, USA'.
    02 Atable value spaces.
        03 wA pic AAA occurs 5.
    02 Btable value spaces.
        03 wB pic XXX occurs 5.
    02 Ctable value zeros.
        03 wC pic 999 occurs 5.

Move 'abc' to wA(1)
Move 123     to wC(3)
XML GENERATE wDoc from wG
```

```
        suppress HiddenInfo
                 every nonnumeric element when space
                 every numeric     element when zero
    END-XML
```

Generated document:

```
<wG>
    <Atable><A>abc</A></Atable>
    <Ctable><C>123</C></Ctable>
</wG>
```

ON EXCEPTION and NOT ON EXCEPTION.

54. When an error occurs during XML document generation, an exception condition exists. An example of this is when identifier-1 is not large enough to contain the generated XML document. In this case, XML generation stops and the content of the receiver, identifier- 1, is undefined. If the COUNT IN phrase was specified, identifier-3 contains the number of character positions that were generated. This can range from zero to the length of identifier-1.
55. If the ON EXCEPTION phrase is specified, control is transferred to imperative-statement-1. If it is not specified, NOT ON EXCEPTION phrases are ignored, and control is transferred to the end of the XML GENERATE statement.
56. At termination of an XML GENERATE statement, special register XML-CODE contains either 0, indicating successful completion of XML generation, or a non-zero error code, indicating that an exception occurred during XML generation.
57. If no exception conditions arise during generation of the XML document, control is passed to imperative-statement-2, if specified, or to the end of the XML GENERATE statement. If an ON EXCEPTION phrase is specified, it is ignored. Special register XML-CODE contains a zero after the XML GENERATE statement has finished executing.

END-XML

58. Is an explicit scope terminator that delimits the scope of XML GENERATE statement. With END-XML, conditional XML GENERATE statements can be nested in other condi- tional statements. Conditional XML GENERATE statements specify the ON EXCEPTION or NOT ON EXCEPTION phrase.
59. The scope of a conditional XML GENERATE or XML PARSE statement is terminated by:
    a. An END-XML phrase at the same level of nesting
    b. A separator period

NESTED XML GENERATE STATEMENTS.

60. When a given XML GENERATE statement appears as imperative-statement-1 or imperative-statement-2, or as part of imperative-statement-1 or imperative-statement-2 of another XML GENERATE statement, that given XML GENERATE statement is a nested XML GENERATE statement.
61. Nested XML GENERATE statements are considered to be matched XML GENERATE and END-XML combinations proceeding from left to right. For this reason, when END-XML phrases are encountered, they are matched with the nearest preceding XML GENERATE statements that have not already been terminated.

7.8.56 XML PARSE

XML PARSE Syntax

```
XML PARSE identier-1
~~~ ~~~~~
      WITH  ENCODING  identier-2
            ~~~~~~~~
                        literal-1

      RETURNING NATIONAL
      ~~~~~~~~~~ ~~~~~~~~~
   VALIDATING WITH identier-3
   ~~~~~~~~~~~
   PROCESSING PROCEDURE IS procedure-name-1      {  THROUGH  }
   ~~~~~~~~~~~ ~~~~~~~~~~                        {  THRU     } procedure-name-2
   ON EXCEPTION imperative-statement-1
       ~~~~~~~~~~
   NOT ON EXCEPTION imperative-statement-2
   ~~~       ~~~~~~~~~~
   END-XML
   ~~~~~~~
```

The XML PARSE statement starts an event-driven XML parser, the processing procedure is performed as each component of the XML document is identied in order.

End of Chapter 7 — PROCEDURE DIVISION

8 Functions

8.1 Intrinsic Functions

GnuCOBOL supports a wide variety of "intrinsic functions" that may be used anywhere in the PROCEDURE DIVISION where a literal is allowed. For example:

```
MOVE FUNCTION LENGTH(Employee-Last-Name) TO Employee-LN-Len
```

Note how the word FUNCTION is part of the syntax when you use an intrinsic function. You can use intrinsic functions without having to include the reserved word FUNCTION via settings in the REPOSITORY (see Section 5.1.4 [REPOSITORY], page 99) paragraph. You may accomplish the same thing by specifying the -fintrinsics switch to the GnuCOBOL compiler when you compile your programs.

User-written functions (see Section 11.1 [Subprogram Types], page 719) never require the FUNCTION keyword when they are executed, because each user-written function a program uses must be included in that program's REPOSITORY paragraph, which therefore makes the FUNCTION keyword optional.

The following intrinsic functions, known to other "dialects" of COBOL, are defined to GnuCOBOL as reserved words but are not otherwise implemented currently. Any attempts to use these functions will result in a compile-time error message. However they are described at the end of this chapter.

BOOLEAN-OF-INTEGER
CHAR-NATIONAL
DISPLAY-OF
EXCEPTION-FILE-N
EXCEPTION-LOCATION-N
INTEGER-OF-BOOLEAN
NATIONAL-OF
STANDARD-COMPARE

Date and Time Formats
~~~~~~~~~~~~~~~~~~~~~

For functions FORMATTED-CURRENT-DATE, FORMATTED-DATE, FORMATTED-TIME, and FORMATTED-DATETIME, the format literal argument indicates the format of the date or time value that is the result of the function. The result of the function will have the same type as its format literal, which can be alphanumeric, national or UTF-8.

For functions INTEGER-OF-FORMATTED-DATE, SECONDS-FROM-FORMATTED-TIME, and TEST-FORMATTED-DATETIME, the format literal indicates the format of the date or time value specified as the second argument of the function.

The permissible format strings are listed as follows. For a full description of each subfield in the format literals, including a range of permissible values in data associated with the formats, see the Value meanings and limits section.

Integer date form:
~~~~~~~~~~~~~~~~~

A value in integer date form is a positive integer that represents the number of days since 31 December, 1600 in the Gregorian calendar.

It must be greater than zero and less than or equal to the value of FUNCTION INTEGER-OF-DATE (99991231), which is 3,067,671.

Standard date form:
~~~~~~~~~~~~~~~~~~

A value in standard date form is an integer of the form YYYYMMDD, calculated using (YYYY * 10,000) + (MM * 100) + DD, where:

YYYY represents the year in the Gregorian calendar. It must be an integer in the range [1601, 9999].

MM represents a month and must be an integer in the range [01, 12].

DD represents a day and must be an integer in the range [01, 31], valid for the specified month and year combination.

Julian date form:
~~~~~~~~~~~~~~~~

A value in Julian date form is an integer of the form YYYYDDD, calculated using (YYYY * 1000) + DDD, where:

YYYY represents the year in the Gregorian calendar. It must be an integer in the range [1601, 9999].

DDD represents the day of the year. It must be a positive integer in the range [1, 366], valid for the year specified.

UTC offset value:
~~~~~~~~~~~~~~~~

A UTC offset value is an integer representation of offset from UTC (Coordinated Universal Time) expressed in minutes. The value must be greater than or equal to -1439 and less than or equal to 1439.

Note: The offset value 1439 represents 23 hours 59 minutes, which is one minute less than a day.
Standard numeric time form

A value in standard numeric time form is a numeric value representing seconds past midnight. The value must be greater than or equal to zero and less than 86,400

Date and time formats:
~~~~~~~~~~~~~~~~~~~~~

For functions FORMATTED-CURRENT-DATE, FORMATTED-DATE, FORMATTED-TIME, and FORMATTED-DATETIME, the format literal argument indicates the format of the date or time value that is the result of the function. The result of the function will have the same type as its format literal, which can be alphanumeric, national or UTF-8.

For functions INTEGER-OF-FORMATTED-DATE, SECONDS-FROM-FORMATTED-TIME, and TEST-FORMATTED-DATETIME, the format literal indicates the format of the date or time value specified as the second argument of the function.

The permissible format strings are listed as follows. For a full description of each subfield in the format literals, including a range of permissible values in data associated with the formats, see Value meanings and limits.

Date formats	Format literals
Basic calendar date	YYYYMMDD
Extended calendar date	YYYY-MM-DD
Basic ordinal date	YYYYDDD
Extended ordinal date	YYYY-DDD
Basic week date	YYYYWwwD
Extended week date	YYYY-Www-D

Integer-seconds time formats:

Integer-seconds time formats	Format literals
Basic local time	hhmmss
Extended local time	hh:mm:ss
Basic Coordinated Universal Time (UTC)	hhmmssZ
Extended UTC time	hh:mm:ssZ
Basic offset time	hhmmss+hhmm
Extended offset time	hh:mm:ss+hh:mm

Fractional-seconds time formats:

Fractional-seconds time formats	Format literals
Basic local time	hhmmss.ssss
Extended local time	hh:mm:ss.ssss
Basic Coordinated Universal Time (UTC)	hhmmss.ssssZ
Extended UTC time	hh:mm:ss.ssssZ
Basic offset time	hhmmss.ssss+hhmm
Extended offset time	hh:mm:ss.ssss+hh:mm

Note: The period is used as the decimal separator, and four "s" characters after the period are used for illustrative purposes. The number of "s" characters that might be specified after the decimal separator in these formats might range from 1 to 9.

Value meanings and limits:

The permissible date and time formats have the following meanings and limits:

Format	Meaning and limits
YYYY	Year, 1601-9999
MM	Month, 01-12
DD	Day of month, 01-{28|29|30|31} dependent on month sub-field

DDD Day of year for ordinal date formats, 001-365|366 ww
Week of year, 01-53
D Day of week, 1-7 W
–
hh Hours, 00-23
mm Minutes, 00-59
ss Seconds, 00-59
.s Fractional seconds, always prefixed with '.' then 1-9 's'
+|-hh:mm UTC offset hours (extended times only), the offset can be adjusted upward (by a '+' prefix) or downward (by a - prefix). A prefix of 0 (zero) indicates that an offset of UTC is not available on the system.
Z UTC time indicator

Value meanings and limits:
~~~~~~~~~~~~~~~~~~~~~~~~~~

The permissible date and time formats have the following meanings and limits:

Format Meaning and limits:
~~~~~~~~~~~~~~~~~~~~~~~~~~

YYYY Year, 1601-9999
MM Month, 01-12
DD Day of month, 01-{28|29|30|31} dependent on month sub-field
DDD Day of year for ordinal date formats, 001-365|366
ww Week of year, 01-53
D Day of week, 1-7 W
–
hh Hours, 00-23
mm Minutes, 00-59
ss Seconds, 00-59
.s Fractional seconds, always prefixed with '.' then 1-9 's'
+|-hh:mm UTC offset hours (extended times only), the offset can be adjusted upward (by a '+' prefix) or downward (by a - prefix).

A prefix of 0 (zero) indicates that an offset of UTC is not available on the system.
Z UTC time indicator

The supported intrinsic functions are listed in the following sections, along with their syntax and usage notes.

8.1.1 ABS

ABS Function Syntax

ABS(number)
~~~

———————————————————————————————————————

This function determines and returns the absolute value of number (a numeric literal or data item) supplied as an argument.

Note that ABSOLUTE-VALUE has an alias for this function.
~~~

8.1.2 ACOS

ACOS Function Syntax

ACOS(cosine)
~~~~

———————————————————————————————————————

The ACOS function determines and returns the trigonometric arc-cosine, or inverse cosine, of *cosine* value (a numeric literal or data item) supplied as an argument.

The result will be an angle, expressed in radians. You may convert this to an angle measured in degrees, as follows:

```
COMPUTE degrees = ( radians * 180 ) / FUNCTION PI
```
~~~~

8.1.3 ANNUITY

ANNUITY Function Syntax

ANNUITY(interest-rate, number-of-periods)
~~~~~~~

This function returns a numeric value approximating the ratio of an annuity paid at interest-rate (numeric data item or literal) for each of number-of-periods (numeric data items or literals).

interest-rate is the rate of interest paid at each payment. If you only have an annual interest rate and you wish to compute monthly annuity payments, divide the annual interest rate by 12 and use that value for interest-rate.

Multiply the result of this function times the desired principal amount to determine the amount of each period's payment.

A note for the financially challenged: an annuity is basically a reverse loan; an accountant would take the result of this function multiplied by -1 times the principal amount to compute a loan payment you are making.

1. Here is an example of a program using this function. Given a total amount of 100,000 USD and an annual interest of 5% the program calculates the monthly payment for the duration of one year, two years ... up to 10 years.

```
            >>SOURCE FREE
IDENTIFICATION DIVISION.
PROGRAM-ID. PANNUITY.
*> Given a total amount of 100,000 and an annual interest of 5%
*> the program calculates monthly payment for a duration of 1 year, 2 years
*> ...up to 10 years.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Total-Loan          Pic  9(9)V99  value  100000.
01  Interest-Rate       Pic  999V99   value  0.05.
01  Interest-RateP      Pic  999V99   value  zero.
01  Months              Pic  999      value  zero.
01  Years               Pic  999      value  zero.
01  Monthly-Payment     Pic  9(9)V99.
01  Total-Payments      Pic  9(9)V99.

PROCEDURE DIVISION.
DISPLAY SPACE
COMPUTE Interest-RateP = Interest-Rate * 100
DISPLAY 'Total Loan: ' Total-Loan ' USD - Interest Rate: ' Interest-RateP '%' DISPLAY SPACE
DISPLAY ' Y  M Monthly Amount  Total Payments' DISPLAY '-'
PERFORM 10 TIMES
   ADD 12 to Months
   COMPUTE Monthly-Payment = Total-Loan * FUNCTION ANNUITY ((Interest-Rate / 12), Months)
```
~~~~~~~

```
    COMPUTE Total-Payments = Monthly-payment * Months
    COMPUTE Years = Months / 12
    DISPLAY Years ' ' Months '          ' Monthly-Payment ' USD   '
  Total-Payments END-PERFORM
  ACCEPT omitted
  GOBACK.
```

2. Other additional documentation:
3. When the value of Interest-Rate is zero, the value returned by the function is the approximation of: (1 / Number-Periods) When the value of Interest-Rate is not zero, the value of the function is the approximation of: (Interest-Rate / (1 - (1 + Interest-Rate) ** (- (Number-Periods))))

8.1.4 ASIN

ASIN Function Syntax

ASIN(sine)
~~~~

———————————————————————————————————————

The ASIN function determines and returns the trigonometric arc-sine, or inverse sine, of sine value (a numeric literal or data item) supplied as an argument.

The result will be an angle, expressed in radians. You may convert this to an angle measured in degrees, as follows:

```
COMPUTE degrees = ( radians * 180 ) / FUNCTION PI
```
~~~~

8.1.5 ATAN

ATAN Function Syntax

ATAN(tangent)
~~~~

———————————————————————————————————————

Use this function to determine and return the trigonometric arc-tangent, or inverse tangent, of *tangent* value (a numeric literal or data item) supplied as an argument.

The result will be an angle, expressed in radians. You may convert this to an angle measured in degrees, as follows:

```
COMPUTE degrees = ( radians * 180 ) / FUNCTION PI
```
~~~~

8.1.6 BIT-OF

BIT-OF Function Syntax

BIT-OF (argument-1)
~~~~~~

BIT-OF function returns an alphanumeric character string of '1' and '0' characters, which represents the binary value of each byte in the argument used on input.

1. The function type is alphanumeric.
2. *argument-1* must be a data item, literal, or an intrinsic function result of any data class.

Returned values:

1. An alphanumeric character string consisting of the binary representation of each byte in *argument-1*.
2. The length of the character string returned, in bytes, is eight times the length of argument-1, in bytes.

```
        >>SOURCE FREE
*> Example of use of function BIT-OF
identification division.
program-id. pgmbitof.
environment division.
configuration section.
   repository. function all intrinsic.
data division.
working-storage section.
01 AAA PIC XXX VALUE "1 2".
01 BBB PIC XXX VALUE "A B".

procedure division.
  display BIT-OF(1)        at 0110
  display BIT-OF(2)        at 0210
  display BIT-OF(3)        at 0310
  display  BIT-OF(0123)  at  0410
  display  BIT-OF(AAA)  at  0510
  display  BIT-OF(BBB)  at  0610
  accept omitted
  stop  run.

Produces :

00110001
00110010
00110011
001100000011000100110010001100 11
0011000100100000000110010
0100000100100000001000010
```
~~~~~~

8.1.7 BIT-TO-CHAR

BIT-TO-CHAR Function Syntax

BIT-TO-CHAR {argument-1)

BIT-TO-CHAR function returns a character string that represents a bit pattern supplied on input.

1. The function type is alphanumeric.
2. *argument-1* must be an alphanumeric literal, alphanumeric data item, or alphanumeric group item.
3. *argument-1* must consist only of the characters "0" and "1".
4. The length of argument-1 must be a multiple of 8 bytes.

Returned values:

1. A character string consisting of bytes representing the sequence of "0" and "1" characters in argument-1.
2. The length of the result string is equal to the length of the input string divided by 8.

```
        >>SOURCE FREE
*> Example of use of function BIT-TO-CHAR
identification  division.
program-id. pgmbittochar.
environment division.
configuration section.
   repository. function all intrinsic.
data division.
working-storage section.
01 AAA PIC X(8) VALUE "0110000".

procedure division.
   display  BIT-TO-CHAR("00110000")  at  0610
   display  BIT-TO-CHAR("00110001")  at  0710
   display  BIT-TO-CHAR("00110010")  at  0810
   display BIT-TO-CHAR("00110011") at 0910
   display BIT-TO-CHAR(AAA)            at 1010
   accept omitted
   stop run.
```

Produces:

```
0
1
2
3
a
```

8.1.8 BYTE-LENGTH

BYTE-LENGTH Function Syntax

BYTE-LENGTH(string)

BYTE-LENGTH returns the length — in bytes — of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal). This intrinsic function is identical to the LENGTH-AN (see Section 8.1.44 [LENGTH-AN], page 474) function. Note that the value returned by this function is not necessarily the number of characters comprising string, but rather the number of actual bytes required to store it.

For example, if string is encoded using a double-byte character set such as Unicode UTF-16 (where each character is represented by 16 bits of storage, not the 8-bits inherent to character sets like ASCII or EBCDIC), then calling this function with a string argument whose PICTURE (see Section 6.9.37 [PICTURE], page 195) is N(4) would return a value of 8 rather than the value 4.

Contrast this with the LENGTH (see Section 8.1.43 [LENGTH], page 473) function.

8.1.9 CHAR

CHAR Function Syntax

CHAR(integer)

This function returns the character in the ordinal position specified by integer (a numeric integer literal or data item with a value of 1 or greater) from the COLLATING SEQUENCE (see Section 5.1.2 [OBJECT-COMPUTER], page 88) being used by the program.

For example, if the program is using the (default) ASCII character set, CHAR(34) returns the 34th character in the ASCII character set — an exclamation-point ('!'). If you are using this function to convert a numeric value to its corresponding ASCII character, you must use an argument value one greater than the numeric value.

If an argument whose value is less than 1 or greater than 256 is specified, the character in the program collating sequence corresponding to a value of all zero bits is returned.

The following code is an alternative approach when you just wish to convert a number to its ASCII equivalent:

```
01  Char-Value.
    05  Numeric-Value          USAGE  BINARY-CHAR.
...
    MOVE numeric-character-value TO Numeric-Value
```

The Char-Value item now has the corresponding ASCII character value.

8.1.10 COMBINED-DATETIME

COMBINED-DATETIME Function Syntax

COMBINED-DATETIME(days, seconds)

This function returns a 12-digit numeric result, the first seven digits of which are the integer value of days argument (a numeric data item or literal) and the last five of which are the integer value of seconds argument (also a numeric data item or literal).

If days is less than 1 or greater than 3,067,671, or if seconds is less than 1 or greater than 86,400, a value of 0 is returned and a runtime error will result.

days Must be in integer date form. For details, see Integer date form. A value in integer date form is a positive integer that represents a number of days succeeding 31 December 1600, in the Gregorian calendar. It is based on a starting date of Monday, 1 January 1601 and integer date 1 represents Monday, 1 January 1601.

seconds Must be in standard numeric time form. For details, see Standard numeric time form. A value in standard numeric time form is a numeric value representing seconds past midnight.

The returned value is determined by arithmetic expression Days-1 + (Seconds-2/100000). The date occupies the integer part of the returned value and the time is represented in the fractional part of the returned value.

Example Given the integer date form value "143951", which represents the date 15 February 1995, and the standard numeric time form value "18867.812479168304", which represents the time "05:14:27.812479168304", the returned value would be exactly "143951.1886781247".

8.1.11 CONCAT

CONCAT Function Syntax

CONCAT | CONCATENATE (argument-1 [, argument-2]...)

This function concatenates the argument-1, argument-2, . . . (group items, USAGE DISPLAY elementary items and/or alphanumeric literals) together into a single string result.

If a numeric literal or PIC 9 identifier is specified as an argument, decimal points, if any, will be removed and negative signs in PIC S9 fields or numeric literals will be inserted as defined by the SIGN IS (see Section 6.9.47 [SIGN IS], page 212) clause (or absence thereof) of the field. Numeric literals are processed as if SIGN IS TRAILING SEPARATE were in effect.

8.1.12 CONCATENATE

CONCATENATE Function Syntax

```
CONCAT | CONCATENATE (argument-1 [, argument-2 ]...)
```

This function concatenates the string-1, string-2, . . . (group items, USAGE DISPLAY elementary items and/or alphanumeric literals) together into a single string result.

If a numeric literal or PIC 9 identifier is specified as an argument, decimal points, if any, will be removed and negative signs in PIC S9 fields or numeric literals will be inserted as defined by the SIGN IS (see Section 6.9.47 [SIGN IS], page 212) clause (or absence thereof) of the field. Numeric literals are processed as if SIGN IS TRAILING SEPARATE were in effect.

CONCATENATE is a GnuCOBOL extention BUT also see the ISO standard CONCAT function.

8.1.13 CONTENT-LENGTH

CONTENT-LENGTH Function Syntax

CONTENT-LENGTH argument-1

Scans for a NUL byte delimiter of the data starting at address in given pointer, and returns the length. The NUL byte is not included in the count. An EC-DATA-PTR-NUL exception is set to exist if the pointer is NUL, and a zero length is returned.

Function CONTENT-LENGTH is a GnuCOBOL extention.

Example:

```
01  ptr USAGE POINTER.
01  str    PIC X(4)    VALUE z"abc".

    SET        ptr TO ADDESS OF str.
    DISPLAY    FUNCTION CONTENT-LENGTH (str).
```

Will display 3.

8.1.14 CONTENT-OF

CONTENT-OF Function Syntax

CONTENT-OF pointer-1 { length }

Takes a pointer and optional length. Returns a character field of the data addressed by the pointer, either up to a NUL byte or to the given length.

The NUL byte is not included in the data when no optional length is given. With an optional count, the character field can hold any content including NUL bytes,

An EC-DATA-PTR-NUL exception is set to exist if the pointer is NUL, and a zero length space is returned.

An EC-SIZE-TRANCATION is set if the resulting field would exceed character field limits and the data is truncated.

Reference modification is allowed on resulting field.

Function CONTENT-OF is a GnuCOBOL extention.

8.1.15 COS

COS Function Syntax

COS(angle)
~~~

———————————————————————————————————————

The COS function determines and returns the trigonometric cosine of angle (a numeric literal or data item) supplied as an argument.

angle is assumed to be a value expressed in radians. If you need to determine the cosine of an angle measured in degrees, you first need to convert that angle to radians as follows:

```
COMPUTE radians = ( degrees * FUNCTION PI) / 180
```
~~~

8.1.16 CURRENCY-SYMBOL

CURRENCY-SYMBOL Function Syntax

CURRENCY-SYMBOL
~~~~~~~~~~~~~~~

———————————————————————————————————————

The CURRENCY-SYMBOL function returns the currency symbol character currently in effect for the locale under which your program is running. On UNIX systems, your locale is established via the LANG run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) environment variable. On Windows, the Control Panel's "Regional and Language Options" define the locale.

Changing the currency symbol via the SPECIAL-NAMES (see Section 5.1.3 [SPECIAL- NAMES], page 90) paragraph's CURRENCY SYMBOL setting will not affect the value returned by this function.
~~~~~~~~~~~~~~~

8.1.17 CURRENT-DATE

CURRENT-DATE Function Syntax

CURRENT-DATE

Returns the current date and time as the following 21-character structure:

```
01   CURRENT-DATE-AND-TIME.
     05  CDT-Year                  PIC  9(4).
     05  CDT-Month                 PIC  9(2).  *>  01-12
     05  CDT-Day                   PIC  9(2).  *>  01-31
     05  CDT-Hour                  PIC  9(2).  *>  00-23
     05  CDT-Minutes               PIC  9(2).  *>  00-59
     05  CDT-Seconds               PIC  9(2).  *>  00-59
     05  CDT-Hundredths-Of-Secs    PIC  9(2).  *>  00-99
     05  CDT-GMT-Diff-Hours        PIC  S9(2)
                                   SIGN LEADING SEPARATE.
      05 CDT-GMT-Diff-Minutes       PIC 9(2). *> 00 or 30
```

Since this function has no arguments, no parenthesis should be specified.

8.1.18 DATE-OF-INTEGER

DATE-OF-INTEGER Function Syntax

DATE-OF-INTEGER(integer)

This function returns a numeric calendar date in yyyymmdd (i.e. Gregorian) format. The date is determined by adding the number of days specified as integer (a numeric integer data item or literal) to the date December 31, 1600. For example, DATE-OF-INTEGER(1) returns 16010101 while DATE-OF-INTEGER(150000) returns 20110908.

A value less than 1 or greater than 3067671 (9999/12/31) will return a result of 0.

8.1.19 DATE-TO-YYYYMMDD

DATE-TO-YYYYMMDD Function Syntax

DATE-TO-YYYYMMDD(yymmdd [, yy-cutoff [, yy-execution-time]])

You can use this function to convert the six-digit Gregorian date specified as yymmdd (a numeric integer data item or literal) to an eight-digit format (yyyymmdd).

The optional yy-cutoff (a numeric integer data item or literal) argument is the year cutoff used to delineate centuries; if the year component of the date meets or exceeds this cutoff value, the result will be 19yymmdd; if the year component of the date is less than the cutoff value, the result will be 20yymmdd. The default cutoff value if no second argument is given will be 50.

The optional yy-execution-time argument (a numeric integer data item or literal) The default execution time value if no third argument is given will be now equivalent to specifying (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4))).

8.1.20 DAY-OF-INTEGER

DAY-OF-INTEGER Function Syntax

DAY-OF-INTEGER(integer)

This function returns a calendar date in yyyyddd (i.e. Julian) format. The date is deter- mined by adding the number of days specified as integer (a numeric integer data item or lit- eral) to December 31, 1600. For example, DAY-OF-INTEGER(1) returns 1601001 while DAY-OF-INTEGER(250000) returns 2011251.

A value less than 1 or greater than 3067671 (9999/12/31) will return a result of 0.

8.1.21 DAY-TO-YYYYDDD

DAY-TO-YYYYDDD Function Syntax

DAY-TO-YYYYDDD(yyddd [, yy-cutoff [, yy-execution-time]])

You can use this function to convert the five-digit Julian date specified as yyddd (a numeric integer data item or literal) to a seven-digit numeric Julian format (yyyyddd).

The optional yy-cutoff argument (a numeric integer data item or literal) is the year cutoff used to delineate centuries; if the year component of the date meets or exceeds this cutoff value, the result will be 19yyddd; if the year component of the date is less than the cutoff, the result will be 20yyddd. The default cutoff value if no second argument is given will be 50.

The optional yy-execution-time argument (a numeric integer data item or literal) The default execution time value if no third argument is given will be now equivalent to specifying (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4))).

8.1.22 E

E Function Syntax

```
E
~
```

This function returns the mathematical constant E (the base of natural log- arithms). The maximum precision with which this value may be returned is 2.718281828459045235360287471352662 5.

Since this function has no arguments, no parenthesis should be specified.

8.1.23 EXCEPTION-FILE

EXCEPTION-FILE Function Syntax

EXCEPTION-FILE
~~~~~~~~~~~~~~

This function returns I/O exception information from the most-recently executed input or output statement. The information is returned as a 34-character string, where the first two characters are the two-digit file status value (see [File Status Codes], page 105) and the remaining 32 are the file-name-1 specification from the file's SELECT (see Section 5.2.1 [SELECT], page 102) statement.

The name returned after the file status information will be returned only if the returned file status value is not 00.

Since this function has no arguments, no parenthesis should be specified.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.
~~~~~~~~~~~~~~

8.1.24 EXCEPTION-LOCATION

EXCEPTION-LOCATION Function Syntax

EXCEPTION-LOCATION

This function returns exception information from the most-recently failing statement. The information is returned to a 1023 character string in one of the following formats, depending on the nature of the failure:

- primary-entry-point-name; paragraph OF section; statement-number
- primary-entry-point-name; section; statement-number
- primary-entry-point-name; paragraph; statement-number
- primary-entry-point-name; statement-number

Since this function has no arguments, no parenthesis should be specified.

The program must be compiled with the -debug switch, -ftraceall switch or -g switch for this function to return any meaningful information.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.

8.1.25 EXCEPTION-STATEMENT

EXCEPTION-STATEMENT Function Syntax

EXCEPTION-STATEMENT
~~~~~~~~~~~~~~~~~~~

This function returns the most-recent COBOL statement that generated an exception condition.

Since this function has no arguments, no parenthesis should be specified.

The program must be compiled with the -debug switch, -ftraceall switch or -g switch for this function to return any meaningful information.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.
~~~~~~~~~~~~~~~~~~~

8.1.26 EXCEPTION-STATUS

EXCEPTION-STATUS Function Syntax

EXCEPTION-STATUS
~~~~~~~~~~~~~~~~

———————————————————————————————————————

This function returns the error type (a text string — see column 2 of the upcoming table for the possible values) from the most-recent COBOL statement that generated an exception condition.

Since this function has no arguments, no parenthesis should be specified.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.

The following are the error type strings, and their corresponding exception codes and descriptions.

Code	Error Type	Description
0101	EC-ARGUMENT-FUNCTION	Function argument error
0202	EC-BOUND-ODO	OCCURS ... DEPENDING ON data item out of bounds
0204	EC-BOUND-PTR	Data-pointer contains an address that is out of bounds
0205	EC-BOUND-REF-MOD	Reference modifier out of bounds
0207	EC-BOUND-SUBSCRIPT	Subscript out of bounds
0303	EC-DATA-INCOMPATIBLE	Incompatible data exception
0500	EC-I-O	input-output exception
0501	EC-I-O-AT-END	I-O status 1x
0502	EC-I-O-EOP	An end of page condition occurred
0504	EC-I-O-FILE-SHARING	I-O status 6x
0505	EC-I-O-IMP	I-O status 9x
0506	EC-I-O-INVALID-KEY	I-O status 2x
0508	EC-I-O-LOGIC-ERROR	I-O status 4x
0509	EC-I-O-PERMANENT-ERROR	I-O status 3x
050A	EC-I-O-RECORD-OPERATION	I-O status 5x
~~~~~~~~~~~~~~~~

0601	EC-IMP-ACCEPT	Implementation-defined accept condition
0602	EC-IMP-DISPLAY	Implementation-defined display condition
0A00	EC-OVERFLOW	Overflow condition
0A02	EC-OVERFLOW-STRING	STRING overflow condition
0A03	EC-OVERFLOW-UNSTRING	UNSTRING overflow condition
0B05	EC-PROGRAM-NOT-FOUND	Called program not found
0D03	EC-RANGE-INSPECT-SIZE	Size of replace item in inspect differs
1000	EC-SIZE	Size error exception
1004	EC-SIZE-OVERFLOW	Arithmetic overflow in calculation
1005	EC-SIZE-TRUNCATION	Significant digits truncated in store
1007	EC-SIZE-ZERO-DIVIDE	Division by zero
1202	EC-STORAGE-NOT-ALLOC	The data-pointer specified in a FREE statement does not identify currently allocated storage
1203	EC-STORAGE-NOT-AVAIL	The amount of storage requested by an ALLOCATE statement is not available

8.1.27 EXP

EXP Function Syntax

EXP(number)
~~~

———————————————————————————————————————

Computes and returns the value of the mathematical constant e raised to the power specified by number (a numeric literal or data item).
~~~

8.1.28 EXP10

EXP10 Function Syntax

EXP10(number)
~~~~~

———————————————————————————————————————

Computes and returns the value of 10 raised to the power specified by number (a numeric literal or data item).
~~~~~

8.1.29 FACTORIAL

FACTORIAL Function Syntax

FACTORIAL(number)
~~~~~~~~~

This function computes and returns the factorial value of number (a numeric literal or data item).
~~~~~~~~~

8.1.30 FORMATTED-CURRENT-DATE

FORMATTED-CURRENT-DATE Function Syntax

FORMATTED-CURRENT-DATE (argument-1)

FORMATTED-CURRENT-DATE returns the current date and time provided by the system at run-time, formatted according to date-and-time-format according to the argument type.

FUNCTION FORMATTED-CURRENT-DATE gives you exactly what you asked it to, including up to nanoseconds (8 decimal positions in the seconds) [but the system may only provide miliseconds, especially on older win32].

The function argument must be a national or alphanumeric literal and the content, a combined date and time format.

The returned value is formatted to the same form as argument-1.

8.1.31 FORMATTED-DATE

FORMATTED-DATE Function Syntax

FORMATTED-DATE (argument-1, argument-2)

FORMATTED-DATE uses a format to convert a date in integer date form to a date in the requested format. The returned value will be in date format.

argument-1 shall be a national or alphanumeric literal.

argument-2 shall be a value in integer date form.

Example Given the date format "YYYYMMDD" and the value "143951", which represents the date 15 February 1995, the returned value would be "19950215".

8.1.32 FORMATTED-DATETIME

FORMATTED-DATETIME Function Syntax

FORMATTED-DATETIME (argument-1, argument-2, argument-3, argument-4)

FORMATTED-DATETIME uses a combined time and date form to convert and combine a date in integer form and a numeric time expressed as seconds past midnight in UTC. See Date and Time Formats for details.

argument-1 shall be a national or alphanumeric literal.

argument-2 shall be a value in integer date form.

argument-3 shall be a value in standard numeric time form.

argument-4 is an integer specifying the offset from UTC expressed in minutes. If specified but have a value equal or less than 1439.

Note: The offset value 1439 represents 23 hours 59 minutes which is one minutes less than a day.

argument-4 must not be specified if the time portion in argument-1 is neither a UTC nor an offset format.

The returned value is a representation of the date contained in argument-2 combined with the time contained in argument-3 according to the format in argument-1.

If the format in argument-1 indicates that the returned value is to be expressed in UTC, the time portion of the returned value reflects the adjustment of the value in argument-3 by the offset in argument-4.

If the format in argument-1 indicates that the time is to be returned as an offset from UTC, the value in argument-3 is reflected directly in the time portion of the returned value and the offset in argument-4 is reflected directly in the offset portion of the returned value.

Example If the first argument has the format "YYMMDDThhmmss.ss+hhmm", the second argument the value "143951", the third argument the value "18867.812479168304", and the fourth argument the value "+300", the returned value would be "19950215T05142781+0500".

8.1.33 FORMATTED-TIME

FORMATTED-TIME Function Syntax

FORMATTED-TIME (argument-1, argument-2, argument-3)

FORMATTED-TIME converts a value representing seconds past midnight formatted time of day with optional offset.

argument-1 shall be a national or alphanumeric literal.

argument-2 shall be a value in integer time form.

argument-3 is an integer specifying the offset from UTC expressed in minutes. If specified but have a value equal or less than 1439.

Note: The offset value 1439 represents 23 hours 59 minutes which is one minutes less than a day.

argument-3 must not be specified if the time portion in argument-1 is neither a UTC nor an offset format.

Returned value :

Is a representation of the standard numeric time contained in argument-2 according to the format in argument-1.

If the format in argument-1 indicates that the returned value is to be expressed in UTC, the time portion of the returned value reflects the adjustment of the value in argument-2 by the offset in argument-3.

If the format in argument-1 indicates that the time is to be returned as an offset from UTC, the value in argument-2 is reflected directly in the time portion of the returned value and the offset in argument-3 is reflected directly in the offset portion of the returned value.

Example If the first argument has the format "hhmmss.ss+hhmm", the second argument the value "18867.812479168304" which represents the local time, and the third argument the value "-300", which represents the five hours that Eastern Standard Time (EST) differs from UTC, the returned value would be "05142781-0500".

8.1.34 FRACTION-PART

FRACTION-PART Function Syntax

FRACTION-PART(number)
~~~~~~~~~~~~~

This function returns that portion of number (a numeric data item or a numeric literal) that occurs to the right of the decimal point. FRACTION-PART(3.1415), for example, returns a value of 0.1415. This function is equivalent to the expression:

number -- FUNCTION INTEGER-PART(number)

Example:
display "base - " FUNCTION FRACTION-PART(FLOATER).
Gives
base - 000.456789

1. When moved to a variable, it MUST have a preceding 'V' in the PICTURE, i.e., PIC v(4).
~~~~~~~~~~~~~

8.1.35 HEX-OF

HEX-OF Function Syntax

HEX-OF {argument-1)
~~~~~~

———————————————————————————————————————

HEX-OF function returns an alphanumeric character string consisting of a hexadecimal representation of the argument used on input.

1. The type of the function is alphanumeric.
2. *argument-1* must be a data item, literal, or an intrinsic function result of any data class.

Returned values:

1. An alphanumeric character string consisting of a hexadecimal representation of argument-1.
2. The length of the character string returned, in bytes, is twice the length of argument-1, in bytes.
~~~~~~

8.1.36 HEX-TO-CHAR

HEX-TO-CHAR Function Syntax

HEX-TO-CHAR {argument-1)

HEX-TO-CHAR function returns a character string that represents the hexadecimal digit characters supplied on input.

1. The type of the function is alphanumeric.
2. *argument-1* must be an alphanumeric literal, alphanumeric data item, or alphanumeric group item.
3. *argument-1* must consist only of the characters '0' through '9', 'A' through 'F', and 'a' through 'f'.
4. The length of argument-1 must be a multiple of 2 bytes.

Returned values:

1. A character string of bytes representing the hexadecimal digit characters of argument-1.
2. The length of the result string is equal to the length of the input string divided by 2.

8.1.37 HIGHEST-ALGEBRAIC

HIGHEST-ALGEBRAIC Function Syntax

HIGHEST-ALGEBRAIC(numeric-identifier)

This function returns the highest (i.e. largest or farthest away from 0 in a positive direction if numeric-identifier is signed) value that could possibly be stored in numeric-identifier. Or in other words the HIGHEST-ALGEBRAIC function provide the ability to manipulate numeric data items in a manner similar to the means that HIGH-VALUES permit with alphanumeric data items, but without the risks of the data incompatibilities associated with those figurative constants.

8.1.38 INTEGER

INTEGER Function Syntax

INTEGER(number)
~~~~~~~

———————————————————————————————————————

The INTEGER function returns the greatest integer value that is less than or equal to number (a numeric literal or data item).
~~~~~~~

8.1.39 INTEGER-OF-DATE

INTEGER-OF-DATE Function Syntax

INTEGER-OF-DATE(date)

This function converts date (a numeric integer data item or literal) — presumed to be a Grego- rian calendar form standard date (YYYYMMDD) — to internal date form (the number of days that have transpired since 1600/12/31).

Once in that form, mathematical operations may be performed against the internal date before it is transformed back into a date using the DATE-OF-INTEGER (see Section 8.1.18 [DATE-OF-INTEGER], page 447) or DAY-OF-INTEGER (see Section 8.1.20 [DAY-OF-INTEGER], page 449) function.

8.1.40 INTEGER-OF-DAY

INTEGER-OF-DAY Function Syntax

INTEGER-OF-DAY(date)

This function converts date (a numeric integer data item or literal) — presumed to be a Julian calendar form standard date (YYYYDDD) — to internal date form (the number of days that have transpired since 1600/12/31).

Once in that form, mathematical operations may be performed against the internal date before it is transformed back into a date using the DATE-OF-INTEGER (see Section 8.1.18 [DATE-OF-INTEGER], page 447) or DAY-OF-INTEGER (see Section 8.1.20 [DAY-OF-INTEGER], page 449) function.

8.1.41 INTEGER-OF-FORMATTED-DATE

INTEGER-OF-FORMATTED-DATE Function Syntax

INTEGER-OF-FORMATTED-DATE (argument-1, argument-2)

INTEGER-OF-FORMATTED-DATE converts a date that is in specified format to integer date form.

argument-1 shall be a national or alphanumeric literal. The content must be either a date format or a combined date and time format.

argument-2 shall be a data item of the same type as argument-1.

If argument-1 is a date format the content of argument-2 shall be a valid date in that format.

If argument-1 is a combined date and time format, the content of argument-2 shall be a valid combined date and time in same format.

8.1.42 INTEGER-PART

INTEGER-PART Function Syntax

INTEGER-PART(number)

Returns the integer portion of number (a numeric literal or data item).

8.1.43 LENGTH

LENGTH Function Syntax

LENGTH(string)
~~~~~~

———————————————————————————————————————

Returns the length — in characters — of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal).

The value returned by this function is not the number of bytes of storage occupied by string, but rather the number of actual characters making up the string. For example, if string is encoded using a double-byte character set such as Unicode UTF-16 (where each character is represented by 16 bits of storage, not the 8-bits inherent to character sets like ASCII or EBCDIC), then calling this function with a string argument whose PICTURE is X(4) would return a value of 4 rather than the value 8 (the actual number of bytes of storage occupied by that item).

Contrast this function with the BYTE-LENGTH (see Section 8.1.8 [BYTE-LENGTH], page 437) and LENGTH-AN (see Section 8.1.44 [LENGTH-AN], page 474) functions.
~~~~~~

8.1.44 LENGTH-AN

LENGTH-AN Function Syntax

LENGTH-AN(string)

This function returns the length — in bytes of storage — of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal).

This intrinsic function is identical to the BYTE-LENGTH (see Section 8.1.8 [BYTE-LENGTH], page 437) function.

Note that the value returned by this function is not the number of characters making up the string, but rather the number of actual bytes of storage required to store string. For example, if string is encoded using a double-byte character set such as Unicode UTF-16 (where each character is represented by 16 bits of storage, not the 8-bits inherent to character sets like ASCII or EBCDIC), then calling this function with a string argument whose PICTURE is N(4) would return a value of 8 rather than the value 4.

Contrast this with the LENGTH (see Section 8.1.43 [LENGTH], page 473) function.

8.1.45 LOCALE-COMPARE

LOCALE-COMPARE Function Syntax

LOCALE-COMPARE(argument-1, argument-2 [, locale])

The LOCALE-COMPARE function returns a character indicating the result of comparing argument-1 and argument-2 using a culturally-preferred ordering defined by a locale.

Either or both of the 1st two arguments may be an alphanumeric literal, a group item or an elementary item appropriate to storing alphabetic or alphanumeric data. If the lengths of the two arguments are unequal, the shorter will be assumed to be padded to the right with spaces.

The two arguments will be compared, character by character, against each other until their relationship to each other can be determined. The comparison is made according to the cultural rules in effect for locale name or for the current locale if no locale argument is specified. Once that relationship is determined, a one-character alphanumeric value will be returned as follows:

- '<' — If argument-1 is determined to be less than argument-2
- '=' — If the two arguments are equal to each other
- '>' — If argument-1 is determined to be greater than argument-2

See [LOCALE Names], page 91, for a list of typically-available locale names.

8.1.46 LOCALE-DATE

LOCALE-DATE Function Syntax

LOCALE-DATE(date [, locale])

Converts the eight-digit Gregorian date (a numeric integer data item or literal) from yyyymmdd format to the format appropriate to the current locale. On a Windows system, this will be the "short date" format as set using Control Panel.

You may include an optional second argument to specify the locale name (group item or PIC X identifier) you'd like to use for date formatting. If used, this second argument must be an identifier. Locale names are specified using UNIX-standard names.

8.1.47 LOCALE-TIME

LOCALE-TIME Function Syntax

LOCALE-TIME(time [, locale])

Converts the four- (hhmm) or six-digit (hhmmss) time (a numeric integer data item or literal) to a format appropriate to the current locale. On a Windows system, this will be the "time" format as set using Control Panel.

You may include an optional locale name (a group item or PIC X identifier) you'd like to use for time formatting. If used, this second argument must be an identifier. Locale names are specified using UNIX-standard names.

8.1.48 LOCALE-TIME-FROM-SECONDS

LOCALE-TIME-FROM-SECONDS Function Syntax

LOCALE-TIME-FROM-SECONDS(seconds [, locale])

Converts the number of seconds since midnight (a numeric integer data item or literal) to a format appropriate to the current locale. On a Windows system, this will be the "time" format as set using Control Panel.

You may include an optional locale name (a group item or PIC X identifier) you'd like to use for time formatting. If used, this second argument must be an identifier. Locale names are specified using UNIX-standard names.

See [LOCALE Names], page 91, for a list of typically-available locale names.

8.1.49 LOG

LOG Function Syntax

LOG(number)
~~~

———————————————————————————————————————

Computes and returns the natural logarithm (base e) of number (a numeric literal or data item).
~~~

8.1.50 LOG10

LOG10 Function Syntax

LOG10(number)
~~~~~

———————————————————————————————————————

Computes and returns the base 10 logarithm of number (a numeric literal or data item).
~~~~~

8.1.51 LOWER-CASE

LOWER-CASE Function Syntax

LOWER-CASE(string)

This function returns the value of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal), converted entirely to lower case.

What constitutes a "letter" (or upper/lower case too, for that manner) may be influenced through the use of a CHARACTER CLASSIFICATION (see Section 5.1.2 [OBJECT-COMPUTER], page 88).

8.1.52 LOWEST-ALGEBRAIC

LOWEST-ALGEBRAIC Function Syntax

LOWEST-ALGEBRAIC(numeric-identifier)

This function returns the lowest (i.e. smallest or farthest away from 0 in a negative direction if numeric-identifier is signed) value that could possibly be stored in numeric-identifier. Or in other words this function LOWEST-ALGEBRAIC provides the ability to manipulate numeric data items in a manner similar to the means that LOW-VALUES permit with alphanumeric data items, but without the risks of the data incompatibilities associated with those figurative constants.

8.1.53 MAX

MAX Function Syntax

```
MAX(number-1  [,  number-2  ]...)
~~~
```

This function returns the maximum value from the specified list of numbers (each number-n may be a numeric data item or a numeric literal).

8.1.54 MEAN

MEAN Function Syntax

```
MEAN(number-1  [,  number-2  ]...)
```

This function returns the statistical mean value of the specified list of numbers (each number-n may be a numeric data item or a numeric literal).

8.1.55 MEDIAN

MEDIAN Function Syntax

```
MEDIAN(number-1  [,  number-2  ]...)
~~~~~~
```

This function returns the statistical median value of the specified list of numbers (each number-n may be a numeric data item or a numeric literal).

8.1.56 MIDRANGE

MIDRANGE Function Syntax

MIDRANGE(number-1 [, number-2]...)

The MIDRANGE (middle range) function returns a numeric value that is the arithmetic mean (average) of the values of the minimum and maximum numbers from the supplied list. Each number-n may be a numeric data items or a numeric literal.

8.1.57 MIN

MIN Function Syntax

MIN(number-1 [, number-2]...)
~~~

———————————————————————————————————————

This function returns the minimum value from the specified list of numbers (each number-n may be a numeric data item or a numeric literal).
~~~

8.1.58 MOD

MOD Function Syntax

MOD(value, modulus)
~~~

———————————————————————————————————————

This function returns the value of value modulo modulus (essentially the remainder from the division of value by modulus). Both arguments may be numeric data items or numeric literals. Either (or both) may have a non-integer value.
~~~

8.1.59 MODULE-CALLER-ID

MODULE-CALLER-ID Function Syntax

MODULE-CALLER-ID

This function returns the null string if it is executed within a main program. When executed with a subprogram, it returns the entry-point name of the program that called the subprogram.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.

8.1.60 MODULE-DATE

MODULE-DATE Function Syntax

MODULE-DATE
~~~~~~~~~~~

This function Returns the date the GnuCOBOL program that is executing the function was compiled, in the form yyyymmdd.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.
~~~~~~~~~~~

8.1.61 MODULE-FORMATTED-DATE

MODULE-FORMATTED-DATE Function Syntax

MODULE-FORMATTED-DATE
~~~~~~~~~~~~~~~~~~~~~

This function returns the fully-formatted date and time when the program executing the function was compiled. The exact format of this returned string value may vary depending on the operating system and GnuCOBOL build type.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.
~~~~~~~~~~~~~~~~~~~~~

8.1.62 MODULE-ID

MODULE-ID Function Syntax

MODULE-ID
~~~~~~~~~

This function returns the primary entry-point name (i.e. the PROGRAM-ID or FUNCTION-ID of the program. See Chapter 4 [IDENTIFICATION DIVISION], page 81, for information on those clauses.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.
~~~~~~~~~

8.1.63 MODULE-PATH

MODULE-PATH Function Syntax

MODULE-PATH
~~~~~~~~~~~

This function returns the full path to the executable version of this GnuCOBOL program. The filename component of this value will be exactly as typed on the command line, down to the use of upper- and lower-case letters and presence (or absence) of any extension.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.
~~~~~~~~~~~

8.1.64 MODULE-SOURCE

MODULE-SOURCE Function Syntax

MODULE-SOURCE
~~~~~~~~~~~~~

———————————————————————————————————————

The filename of the source code of the program (as specified on the cobc command when the program was compiled) is returned by this function.

The discussion of the MODULE-TIME (see Section 8.1.65 [MODULE-TIME], page 495) function includes a sample program that uses this function.

Since this function has no arguments, no parenthesis should be specified.
~~~~~~~~~~~~~

8.1.65 MODULE-TIME

MODULE-TIME Function Syntax

MODULE-TIME
~~~~~~~~~~~

This function returns the time the GnuCOBOL program was compiled, in the form hhmmss. Since this function has no arguments, no parenthesis should be specified. The following sample program uses all the MODULE- Functions:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOMODULE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION ALL INTRINSIC.
PROCEDURE DIVISION.
000-Main.
    DISPLAY "MODULE-CALLER-ID          = [" MODULE-CALLER-ID ']'
    DISPLAY "MODULE-DATE               = [" MODULE-DATE ']'
    DISPLAY "MODULE-FORMATTED-DATE = [" MODULE-FORMATTED-DATE ']'
    DISPLAY "MODULE-ID                 = [" MODULE-ID ']'
    DISPLAY "MODULE-PATH               = [" MODULE-PATH ']'
    DISPLAY "MODULE-SOURCE             = [" MODULE-SOURCE ']'
    DISPLAY "MODULE-TIME               = [" MODULE-TIME ']'
    STOP RUN
    .
```

The program produces this output when executed:

```
MODULE-CALLER-ID = []
MODULE-DATE = [20180522]
MODULE-FORMATTED-DATE = [May 22 2018 12:43:14]
MODULE-ID = [DEMOMODULE]
MODULE-PATH = [/home/vince/cobolsrc/ACAS/demomodule]
MODULE-SOURCE = [demomodule.cbl]
MODULE-TIME = [124314]
```
~~~~~~~~~~~

8.1.66 MONETARY-DECIMAL-POINT

MONETARY-DECIMAL-POINT Function Syntax

MONETARY-DECIMAL-POINT

MONETARY-DECIMAL-POINT returns the character used to separate the integer portion from the fractional part of a monetary currency value according to the rules currently in effect for the locale under which your program is running.

On UNIX (including OSX, Windows/Cygwin and Windows/MinGW) systems, your locale is established via the LANG run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Using the DECIMAL-POINT IS COMMA (see Section 5.1.3 [SPECIAL-NAMES], page 90) clause in your program will not affect the value returned by this function.

Since this function has no arguments, no parenthesis should be specified.

8.1.67 MONETARY-THOUSANDS-SEPARATOR

MONETARY-THOUSANDS-SEPARATOR Function Syntax

MONETARY-THOUSANDS-SEPARATOR

This function returns the character used to separate the thousands digit groupings of monetary currency values according to the rules currently in effect for the locale under which your program is running.

On UNIX (including OSX, Windows/Cygwin and Windows/MinGW) systems, your locale is established via the LANG run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Using the DECIMAL-POINT IS COMMA (see Section 5.1.3 [SPECIAL-NAMES], page 90) clause in your program will not affect the value returned by this function.

Since this function has no arguments, no parenthesis should be specified.

8.1.68 NUMERIC-DECIMAL-POINT

NUMERIC-DECIMAL-POINT Function Syntax

NUMERIC-DECIMAL-POINT

This function returns the character used to separate the integer portion of a non-integer numeric item from the fractional part according to the rules currently in effect for the locale under which your program is running.

On UNIX (including OSX, Windows/Cygwin and Windows/MinGW) systems, your locale is established via the LANG run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Using the DECIMAL-POINT IS COMMA (see Section 5.1.3 [SPECIAL-NAMES], page 90) clause in your program will not affect the value returned by this function.

Since this function has no arguments, no parenthesis should be specified.

8.1.69 NUMERIC-THOUSANDS-SEPARATOR

NUMERIC-THOUSANDS-SEPARATOR Function Syntax

NUMERIC-THOUSANDS-SEPARATOR

This function returns the character used to separate the thousands digit groupings of numeric values according to the rules currently in effect for the locale under which your program is running.

On UNIX (including OSX, Windows/Cygwin and Windows/MinGW) systems, your locale is established via the LANG run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) environment variable. On Windows, the Control Panel's Regional and Language Options define the locale.

Using the DECIMAL-POINT IS COMMA (see Section 5.1.3 [SPECIAL-NAMES], page 90) clause in your program will not affect the value returned by this function.

Since this function has no arguments, no parenthesis should be specified.

8.1.70 NUMVAL

NUMVAL Function Syntax

NUMVAL(string)
~~~~~~

———————————————————————————————————————

The NUMVAL function converts a string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) to its corresponding numeric value.

The string must have any of the following formats, where '#' represents a sequence of one or more decimal digits:

```
#    -#    +#    #-    #+    #CR    #DB
#.#    -#.#    +#.#    #.#-    #.#+    #.#CR    #.#DB
```

There must be at least one digit character in the string.

Leading and/or trailing spaces are allowed, as are spaces before the first digit.

The character period in argument-1 string, represents the decimal separator. The character comma in argument-1 represents the grouping separator. When the DECIMAL-POINT IS COMMA clause is specified, the character comma shall be used in argument-1 to represent the decimal separator and the character period shall be used to represent the grouping separator.

Note: Locale-based functionality equivalent to NUMVAL can be obtained by using the NUMVAL-C function with the LOCALE keyword. A currency sign is optional in NUMVAL-C. The locale category LC_MONETARY will be used because there is no sign convention specified in locale category LC_NUMERIC.

Returned values:

The returned value is the numeric value represented by string.

If it contains a CR, DB, or the minus sign ('-'), the returned value is negative.
~~~~~~

8.1.71 NUMVAL-C

NUMVAL-C Function Syntax

NUMVAL-C (string [, symbol]
~~~~~~~~

[, LOCALE locale-name-1] [, ANYCASE])

This function converts a string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) representing a currency value to its corresponding numeric value.

The currency string if any, and any grouping separators preceding the decimal separator are ignored. Optionally, the currency string, sign convention, grouping separator and the decimal separator permitted in the character string may be specified by locale category LC-MONETARY, or the currency string may be specified by symbol.

The optional symbol character represents the currency symbol (a non-space single-character group item, USAGE DISPLAY elementary item or alphanumeric literal) that may be used as the currency character in string. Any spaces including leading or trailing are ignored. If no symbol is specified, the value that would be returned by the CURRENCY-SYMBOL intrinsic function (see Section 8.1.16 [CURRENCY-SYMBOL], page 445) will be used.

If this references the LOCALE :

Changing the currency symbol via the SPECIAL-NAMES paragraph's CURRENCY SYMBOL setting will not affect the value returned by this function.

While NUMVAL-C will always use the currency symbol that is specified via the SPECIAL-NAMES paragraph's CURRENCY SYMBOL (or the system default which is currently always '$').

string may have any of the following formats, where '#' represents a sequence of one or more decimal digits and '$' represents the symbol character:

```
#    -#    +#    #-    #+    #CR    #DB
#.#    -#.#    +#.#    #.#-    #.#+    #.#CR    #.#DB
$#    -$#    +$#    $#-    $#+    $#CR    $#DB
$#.#    -$#.#    +$#.#    $#.#-    $#.#+    $#.#CR    $#.#DB
```

There must be at least one digit character in the string.

Leading and/or trailing spaces are allowed, as are spaces before and/or after the currency symbol, sign, CR and DB characters.

If the ANYCASE keyword is used the matching rules for detecting a currency string in argument-*1* are case-insensitive. If the ANYCASE keyword is not specified, the matching rules are case-sensitive.

If neither symbol nor the LOCALE keyword is specified, there shall be only one currency string used, either the default currency sign or a currency string specified in the SPECIAL-NAMES paragraph.

The returned value is the numeric value represented by string.

When the LOCALE keyword is specified, the returned value is negative if string contains a negative sign.

When the LOCALE keyword is not specified, the returning value is negative if string contains CR, DB, or a minus sign.
~~~~~~~~

8.1.72 NUMVAL-C-2

NUMVAL-C Function Syntax

NUMVAL-C (argument-1 [, argument-2]
~~~~~~~~

[, LOCALE locale-name-1] [, ANYCASE])

This function returns the numeric value represented by the character string specified by *argument-1* and defined as alphanumeric.

argument-2, the currency string if any, and any grouping separators preceding the decimal separator are ignored. Optionally, the currency string, sign convention, grouping separator and the decimal separator permitted in the character string may be specified by locale category LC-MONETARY, or the currency string may be specified by argument-2.

The optional alphanumeric argument-2 character represents the currency symbol (a non-space and at least one single-character item, that may be used as the currency character in argument-1. Any spaces including leading or trailing are ignored. If no argument-2 is specified, the value that would be returned by the CURRENCY-SYMBOL intrinsic function (see Section 8.1.16 [CURRENCY-SYMBOL], page 445) will be used. argument-2 must not contain any of the digits
- through 9, characters '*', '+', '-', ',' or '.'; or the two consecutive letters CR or DB, whether upper or lower case or a combination of both.

argument-2 specifies a currency string that may appear in argument-1.

If the ANYCASE keyword is specified, the matching rules for detecting a currency string in *argument-1* are case-insensitive. If not specified, the matching rules are case-sensitive.

If neither argument-2 nor the LOCALE keyword is specified, there shall be only one currency string used, either the default currency sign or a currency string specified in the SPECIAL-NAMES paragraph.

While NUMVAL-C will always use the currency symbol that is specified via the SPECIAL-NAMES paragraph's CURRENCY SYMBOL (or the system default which is currently always '$') argument-1 shall have any of the following formats, where '#' represents a sequence of one or more decimal digits and '$' represents the symbol character:

```
#    -#    +#    #-    #+    #CR    #DB
#.#    -#.#    +#.#    #.#-    #.#+    #.#CR    #.#DB
$#    -$#    +$#    $#-    $#+    $#CR    $#DB
$#.#    -$#.#    +$#.#    $#.#-    $#.#+    $#.#CR    $#.#DB
```

There must be at least one digit character in the string.

Leading and/or trailing spaces are allowed, as are spaces before and/or after the currency symbol, sign, CR and DB characters.

The returned value is the numeric value represented by argument-1.

When the LOCALE keyword is specified, the returned value is negative if string contains a negative sign and when not specified, the returning value is negative if string contains CR, DB, or a minus sign.
~~~~~~~~

8.1.73 NUMVAL-F

NUMVAL-F Function Syntax

NUMVAL-F(char)
~~~~~~~~

———————————————————————————————————————

This function converts a string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) representing a floating-point value to its corresponding numeric value.

```
#   -#   +#   #E#   -#E#   +#E#
#E+#   -#E+#   +#E+#   #E-#   -#E-#   +#E-#
#.#   -#.#   +#.#   #.#E#   -#.#E#   +#.#E#
#.#E+#   -#.#E+#   +#.#E+#   #.#E-#   -#.#E-#   +#.#E-#
```

There must be at least one digit character both before and after the E in the string. Leading and/or trailing spaces are allowed, as are spaces before and/or after any sign characters.
~~~~~~~~

8.1.74 ORD

ORD Function Syntax

ORD(char)
~~~

———————————————————————————————————————

This function returns the ordinal position in the program character set (usually ASCII) corresponding to the 1st character of char argument (a group item, USAGE DISPLAY elementary item or alphanumeric literal).

For example, assuming the program is using the standard ASCII collating sequence, ORD('!') returns 34 because '!' is the 34th ASCII character. If you are using this function to convert an ASCII character to its numeric value, you must subtract one from the result.

The following code is an alternative approach when you just wish to convert an ASCII character to its numeric equivalent:

```
01  Char-Value.
    05  Numeric-Value          USAGE  BINARY-CHAR.
- - -
    MOVE "character" TO Char-Value
```

Numeric-Value now has the numeric value of character.
~~~

8.1.75 ORD-MAX

ORD-MAX Function Syntax

```
ORD-MAX(char-1 [, char-2 ]...)
```

This function returns the ordinal position in the argument list corresponding to the char-n whose 1st character has the highest position in the program collating sequence (usually ASCII).

For example, assuming the program is using the standard ASCII collating sequence, ORD-MAX('Z', 'z', '!') returns 2 because the 2nd character in the argument list (the ASCII character 'z') occurs after 'Z' and '!' in the program collating sequence. Each char-n argument may be a group item, USAGE DISPLAY elementary item or alphanumeric literal.

8.1.76 ORD-MIN

ORD-MIN Function Syntax

```
ORD-MIN(char-1 [, char-2 ]...)
```

This function returns the ordinal position in the argument list corresponding to the char-n whose 1st character has the lowest position in the program collating sequence (usually ASCII).

For example, assuming the program is using the standard ASCII collating sequence, ORD-MIN('Z', 'z', '!') returns 3 because the 3rd character in the argument list (the ASCII character '!') occurs before 'Z' and 'z' in the program collating sequence. Each char-n argument may be a group item, USAGE DISPLAY elementary item or alphanumeric literal.

8.1.77 PI

PI Function Syntax

```
PI
~~
```

This function returns the mathematical constant PI. The maximum precision with which this value may be returned is 3.14159265358979323846264338327950 29.

Since this function has no arguments, no parenthesis should be specified.

8.1.78 PRESENT-VALUE

PRESENT-VALUE Function Syntax

PRESENT-VALUE(rate, value-1 [, value-2])

The PRESENT-VALUE function returns a value that approximates the present value of a series of future period-end amounts specified by the various value-n arguments at a discount rate specified by the rate argument.

All arguments are numeric data items and/or numeric literals.

The following equation summarizes how present value is calculated, where N is the number of value arguments:

$$presentvalue = \sum_{i=1}^{**N} \frac{value_i}{(1 + rate)**i}$$

1. Example of function in use :

```
            >>SOURCE FREE
IDENTIFICATION DIVISION.
PROGRAM-ID. PPresValue.
*> The sample: you pay for a machine 1500 USD
*> You rent the machine at 350 USD per year per 5 years (= 1750).
*> The program calculates (NET)PRESENT VALUE of 1750
*> when the discount rate is 1%, 2% up to 10%. DATA
DIVISION.
WORKING-STORAGE SECTION.
01   PaymentsNum       constant  as      5.
01   PaymentsAmount    constant as 350.
01   Expenditure       PIC 999999V99 value 1500.
01   TOTAL-CASH-FLOW   PIC 999999V99 value zero.
01   DiscountRate      PIC S99V99 VALUE 0.00.
77   DiscountRateP     PIC S99V99 VALUE 0.00.
01   filler.
     05 PaymentAmount PIC S9999V99 OCCURS PaymentsNum TIMES VALUE PaymentsAmount.
01   PresValue         PIC    9(6)V99    value zero.
01   NET-PresValue     PIC S9(6)V99      value zero.

PROCEDURE DIVISION.
COMPUTE TOTAL-CASH-FLOW =
     FUNCTION SUM (PaymentAmount(1) PaymentAmount(2) PaymentAmount(3)
                   PaymentAmount(4) PaymentAmount(5))
DISPLAY SPACE
DISPLAY    'Expenditure: ' Expenditure ' Total Cashflow: ' TOTAL-CASH-FLOW
DISPLAY SPACE

PERFORM 10 TIMES
  COMPUTE DiscountRate   = DiscountRate + 0.01
  COMPUTE DiscountRateP = DiscountRate * 100
```

```
    COMPUTE PresValue ROUNDED =
        FUNCTION PRESENT-VALUE (DiscountRate PaymentAmount(1) PaymentAmount(2)
                                PaymentAmount(3) PaymentAmount(4) PaymentAmount(5))
    COMPUTE NET-PresValue = - Expenditure + PresValue
    DISPLAY 'DiscountRate: ' DiscountRateP '%    PresValue: ' PresValue
            ' NET-PresValue: ' NET-PresValue
END-PERFORM
ACCEPT omitted
GOBACK.
```

2. This is a case where passing parameters to the intrinsic function PRESENT-VALUE would need the ability to indicate "ALL" and then write the statement as:
3. FUNCTION PRESENT-VALUE (DiscountRate PaymentAmount(ALL))
4. GnuCOBOL does not have this feature (ALL parameter) i.,e has NOT YET been implemented.

8.1.79 RANDOM

RANDOM Function Syntax

RANDOM[(seed)]
~~~~~~

This function returns a pseudo-random non-integer value in the range 0>= to <1 (for example, 0.123456789).

The purpose of the optional seed argument, is to initialize the chain of pseudo-random num- bers that will be returned by the function. Not only will calls to this function using the same seed value return the same pseudo-random number, but so will all subsequent executions of the function without a seed. This is actually a good thing when you are testing your program because you can rely on always receiving the same sequence of "random" numbers if you always start using the same seed.

The seed may be any form of literal or data item or arithmetic expression. If seed is numeric, its numeric value will serve as the seed value. If seed is alphanumeric, a value for it will be determined as if it were used as an argument to NUMVAL (see Section 8.1.70 [NUMVAL], page 500).

Take, for example, the following sample program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMORANDOM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    Pseudo-Random-Number          USAGE COMP-1.
PROCEDURE DIVISION.
000-Main.
      MOVE FUNCTION RANDOM(1) TO Pseudo-Random-Number
      DISPLAY Pseudo-Random-Number
      PERFORM 4 TIMES
            MOVE FUNCTION RANDOMTO  Pseudo-Random-Number
            DISPLAY Pseudo-Random-Number
      END-PERFORM
      STOP RUN
      .
```

Every time this program is executed, it will produce the same output, because the same sequence of pseudo-random numbers will be generated:

```
0.5441364
0.047651578
0.77186662
0.056523036
0.63045478
```

Once your program has been thoroughly tested, you'll want different sequences to be gener- ated each time the program runs. One possible way to accomplish this is to use a seed that is likely to be different every time the program is executed, as is likely to be the case if the first MOVE statement in the previous example were replaced by this:
~~~~~~

```
MOVE RANDOM(FUNCTION CURRENT-DATE(1:16))
   TO Pseudo-Random-Number
```

The first 16 characters returned by the CURRENT-DATE (see Section 8.1.17 [CURRENT- DATE], page 446) function will be a number in the format YYYYMMDDhhmmssnn, where YYYYMMDD is the current calendar date and hhmmssnn is the current time of day to the one one-hundredth of a second. Since two different executions of the program will never get identical CURRENT-DATE values (unless they are executed in extremely close time frames to one another), using those first sixteen characters as the RANDOM seed will guarantee that receiving a duplicate sequence of pseudo-random numbers in two different executions of the program will be highly unlikely.

If you do not use a seed then the RANDOM function will automatically determine an internal different seed at each execution.

Often you need to generate a random number between two numbers. Following example shows how to do it.

```
        >>source free
 IDENTIFICATION DIVISION.
 PROGRAM-ID. RANDOM-RANGE.
*> Generate random number from 25 and 70 DATA
 DIVISION.
 WORKING-STORAGE SECTION.
 01    RANDOM-NUMBER PIC 9999 value zero.
 PROCEDURE DIVISION.
 perform 10 times
       compute   RANDOM-NUMBER = 25 + (70 - 25) * (FUNCTION RANDOM)
       display   RANDOM-NUMBER
 end-perform
 display         space
*> or, to include the upper value 60 as possible choice perform
 10 times
       compute   RANDOM-NUMBER = 25 + (70 - 25 + 1) * (FUNCTION RANDOM)
       display   RANDOM-NUMBER
 end-perform
 STOP RUN.
```

8.1.80 RANGE

RANGE Function Syntax

```
RANGE(number-1  [,  number-2  ]...)
~~~~~
```

The RANGE function returns a value that is equal to the value of the maximum number-n in the argument list minus the value of the minimum number-n argument.

All number-n arguments are numeric data items and/or numeric literals.

8.1.81 REM

REM Function Syntax

REM(number,divisor)
~~~

———————————————————————————————————————

This function returns a numeric value that is the remainder of number divided by divisor. Both arguments must be numeric data items or numeric literals.
~~~

8.1.82 REVERSE

REVERSE Function Syntax

REVERSE(string)
~~~~~~~

This function returns the byte-by-byte reversed value of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal).
~~~~~~~

8.1.83 SECONDS-FROM-FORMATTED-TIME

SECONDS-FROM-FORMATTED-TIME Function Syntax

SECONDS-FROM-FORMATTED-TIME(format,time)

This function decodes the string time — whose value represents a formatted time — and returns the total number of seconds that string represents.

The time string must contain hours, minutes and seconds. The time argument may be specified as a group item, USAGE DISPLAY elementary item or an alphanumeric literal.

The format argument is a string (a group item, USAGE DISPLAY elementary item or an alphanumeric literal) documenting the format of time using hh, *mm* and ss to denote where the respective time information can be found. Any other characters found in format represent character positions that will be ignored. For example, a format of hhmmss indicates that time will be treated as a six-digit string value where the first two characters are the number of hours, the next two represent minutes and the last two represent seconds. A format of hh:mm:ss, however, describes time as an eight-character string where characters 3 and 6 will be ignored.

8.1.84 SECONDS-PAST-MIDNIGHT

SECONDS-PAST-MIDNIGHT Function Syntax

SECONDS-PAST-MIDNIGHT

This function returns the current time of day expressed as the total number of elapsed seconds since midnight.

Since this function has no arguments, no parenthesis should be specified.

8.1.85 SIGN

SIGN Function Syntax

SIGN(number)
~~~~

———————————————————————————————————————

The SIGN function returns a -1 if the value of number (a numeric literal or numeric data item) is negative, a zero if the value of number is exactly zero and a 1 if the value of number if greater than 0.
~~~~

8.1.86 SIN

SIN Function Syntax

SIN(angle)
~~~

———————————————————————————————————————

This function determines and returns the trigonometric sine of angle (a numeric literal or numeric data item).

The angle is assumed to be a value expressed in radians. If you need to determine the sine of an angle measured in degrees, you first need to convert that angle to radians as follows:

```
COMPUTE radians = ( degrees * FUNCTION PI) / 180
```
~~~

8.1.87 SQRT

SQRT Function Syntax

SQRT (number)
~~~~

———————————————————————————————————————

The SQRT function returns a numeric value that approximates the square root of number (a numeric data item or numeric literal with a non-negative value).

The following two statements produce identical results:

```
01  Result              PIC 9(4).9(10).
...
    MOVE FUNCTION SQRT(15) TO Result
    COMPUTE Result = 15 ** 0.5
```
~~~~

8.1.88 STANDARD-DEVIATION

STANDARD-DEVIATION Function Syntax

STANDARD-DEVIATION(number-1 [, number-2]...)

This function returns the statistical standard deviation of the list of number-n arguments (numeric data items or numeric literals).

8.1.89 STORED-CHAR-LENGTH

STORED-CHAR-LENGTH Function Syntax

STORED-CHAR-LENGTH(string)

Returns the length — in bytes — of the specified string (a group item, USAGE DISPLAY elementary item or alphanumeric literal), minus the total number of trailing spaces, if any.

8.1.90 SUBSTITUTE

SUBSTITUTE Function Syntax

SUBSTITUTE(string, from-1, to-1 [, from-n, to-n]...)

This function parses string, replacing all occurrences of from-n strings with the corresponding *to-n* strings.

The from-n strings must match sequences in string exactly with regard to value and case. A from-n string does not have to be the same length as its corresponding to-n string.

All arguments are group items, USAGE DISPLAY elementary items or alphanumeric literals.

A null to-n string will be treated as a single space.

When using Variables in place of string attention to NOT wanting Leading or trailing spaces usage of function TRIM needs to be utilised as failure to do so will result in variables treated with any unwanted spaces leading and/or trailing, i.e.,

```
move      function  SUBSTITUTE        (WS-Dest-File-Path,
                        function TRIM (WS-Inbound-Path),
                        function TRIM (WS-Desc-Path))
                                  to WS-Dest-File-Path
```

8.1.91 SUBSTITUTE-CASE

SUBSTITUTE-CASE Function Syntax

SUBSTITUTE-CASE(string, from-1, to-1 [, from-n, to-n]...)

The SUBSTITUTE-CASE function operates the same as the SUBSTITUTE (see Section 8.1.90 [SUBSTITUTE], page 522) function, except that from-n string matching is performed without regard to case.

All arguments are group items, USAGE DISPLAY elementary items or alphanumeric literals.

When using Variables in place of string attention to NOT wanting Leading or trailing spaces usage of function TRIM needs to be utilised as failure to do so will result in variables treated with any unwanted spaces leading and/or trailing, i.e.,

```
move      function SUBSTITUTE-CASE (WS-Dest-File-Path,
                    function TRIM (WS-Inbound-Path),
                    function TRIM (WS-Desc-Path))
                              to WS-Dest-File-Path
```

8.1.92 SUM

SUM Function Syntax

```
SUM(number-1  [,  number-2  ]...)
~~~
```

———

The SUM function returns a value that is the sum of number-n arguments (these may be numeric data items or numeric literals).

8.1.93 TAN

TAN Function Syntax

TAN(angle)
~~~

———————————————————————————————————————

This function determines and returns the trigonometric tangent of angle (a numeric literal or numeric data item).

The angle is assumed to be a value expressed in radians. If you need to determine the tangent of an angle measured in degrees, you first need to convert that angle to radians as follows:

```
COMPUTE radians = ( degrees * FUNCTION PI) / 180
```
~~~

8.1.94 TEST-DATE-YYYYMMDD

TEST-DATE-YYYYMMDD Function Syntax

TEST-DATE-YYYYMMDD (date)

This function determines if the supplied date argument (a numeric integer data item or literal) is a valid date.

A valid date is one of the form yyyymmdd in the range 1601/01/01 to 9999/12/31, with no more than the expected maximum number of days in the month, accounting for leap year.

If the date is valid, a 0 value is returned. If it isn't, a value of 1, 2 or 3 is returned signalling the problem lies with the year, month or day, respectively.

8.1.95 TEST-DAY-YYYYDDD

TEST-DAY-YYYYDDD Function Syntax

TEST-DAY-YYYYDDD (date)

This function determines if the supplied date (a numeric integer data item or literal) is a valid date.

A valid date is one of the form yyyyddd in the range 1601001 to 9999365. Leap year is accounted for in determining the maximum number of days in a year.

If the date is valid, a 0 value is returned. If it isn't, a value of 1 or 2 is returned signalling the problem lies with the year or day, respectively.

8.1.96 TEST-FORMATTED-DATETIME

TEST-FORMATTED-DATETIME Function Syntax

TEST-FORMATTED-DATETIME (argument-1, argument-2)

TEST-FORMATTED-DATETIME tests whether a date literal representing a date, a time or a combined date and time is valid according to the specified format.

argument-1 must a literal of type alphanumeric, UTF-8 or national, that contains a date, time or combined data time format. See Date and Time formats for details.

argument-2 must be a data item of the same type as argument-1.

Returned value:

If no format or range problems occur during evaluation of argument-2 according to the format in argument-1, the returned value is zero. Otherwise the returned value is the ordinal character position at which the first error in argument-2 was detected.

Example

Using the following arguments, it will generates a return value of 5, as the fifth character of argument-2 ("4") contains an incorrect value for the first digit of the month representation.

```
FUNCTION TEST-FORMATTED-DATETIME("YYYYMMDD", "20124523")
```

8.1.97 TEST-NUMVAL

TEST-NUMVAL Function Syntax

TEST-NUMVAL (string)

The TEST-NUMVAL function evaluates string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) for being appropriate for use as the string argument to a NUMVAL (see Section 8.1.70 [NUMVAL], page 500) function, returning to a integer a zero value if it is appropriate otherwise if one or more characters are in error, the position of the first character in error or the length of the field plus one for other cases such as all spaces.

Note that these errors include but are not limited to: argument (string) is zero length, contains only spaces or contains valid characters but is incomplete, such as the string '+.'.

8.1.98 TEST-NUMVAL-C

TEST-NUMVAL-C Function Syntax

TEST-NUMVAL-C (string[,symbol])

This function evaluates string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) for being appropriate for use as the string argument to a NUMVAL-C (see Section 8.1.71 [NUMVAL-C], page 501) function, returning to a integer a zero value if it is appropriate otherwise if one or more characters are in error, the position of the first character in error or the length of the field plus one for other cases such as all spaces.

Note that these errors include but are not limited to: argument (string) is zero length, contains only spaces or contains valid characters but is incomplete, such as the string '+.'.

The optional symbol argument serves the same function — and has the same default and possible values — as the corresponding argument of the NUMVAL-C function.

8.1.99 TEST-NUMVAL-F

TEST-NUMVAL-F Function Syntax

TEST-NUMVAL-F (string)

This function evaluates string (a group item, USAGE DISPLAY elementary item or alphanumeric literal) for being appropriate for use as the string argument to a NUMVAL-F (see Section 8.1.73 [NUMVAL-F], page 503) function, returning to a integer a zero value if it is appropriate otherwise if one or more characters are in error, the position of the first character in error or the length of the field plus one for other cases such as all spaces.

Note that these errors include but are not limited to: argument (string) is zero length, contains only spaces or contains valid characters but is incomplete, such as the string '+.'.

8.1.100 TRIM

TRIM Function Syntax

```
TRIM(string [, LEADING|TRAILING ])
~~~~                ~~~~~~~ ~~~~~~~~
```

This function removes LEADING or TRAILING spaces from string (a group item, USAGE DISPLAY elementary item or alphanumeric literal).

The second argument is specified as a keyword, not a quoted string or identifier. If no second argument is specified, both leading and trailing spaces will be removed. The case (upper, lower or mixed) of this argument is irrelevant.

8.1.101 UPPER-CASE

UPPER-CASE Function Syntax

UPPER-CASE(string)

This function returns the value of string (a group item, USAGE DISPLAY elementary item or alphanumeric literal), converted entirely to upper case.

What constitutes a "letter" (or upper/lower case too, for that manner) may be influenced through the use of a CHARACTER CLASSIFICATION (see Section 5.1.2 [OBJECT-COMPUTER], page 88).

8.1.102 VARIANCE

VARIANCE Function Syntax

VARIANCE(number-1 [, number-2]...)

This function returns the statistical variance of the specified list of number-n arguments (these may be numeric data items or numeric literals).

8.1.103 WHEN-COMPILED

WHEN-COMPILED Function Syntax

WHEN-COMPILED
~~~~~~~~~~~~~

———

The WHEN-COMPILED intrinsic function, not to be confused with the WHEN-COMPILED (see Section 7.7 [Special Registers], page 260) special register, returns the date and time the program was compiled, in ASCII.

Since this function has no arguments, no parenthesis should be specified.

Unlike the WHEN-COMPILED special register, which has an ASCII value of the compilation date/time in the format mm/dd/yyhh.mm.ss, the WHEN-COMPILED intrinsic function returns the compilation date/time as an ASCII string in the format yyyymmddhhmmssnnooooo, where yyyymmdd is the date, hhmmss is the time, nn is the hundredths of a second component of the compilation time, if available (or 00 if it isn't) and ooooo is the time zone offset from GMT.

If the -fintrinsics=WHEN-COMPILED switch or -fintrinsics=ALL switch is specified to the compiler or the REPOSITORY (see Section 5.1.4 [REPOSITORY], page 99) paragraph spec- ifies either FUNCTION WHEN-COMPILED INTRINSIC or FUNCTION ALL INTRINSIC, then references to WHEN-COMPILED (without a leading FUNCTION keyword will always reference this intrinsic function and there will be no way to access the WHEN-COMPILED special register.
~~~~~~~~~~~~~

8.1.104 YEAR-TO-YYYY

YEAR-TO-YYYY Function Syntax

YEAR-TO-YYYY(yy [, yy-cutoff [, yy-execution-time]])

YEAR-TO-YYYY converts yy — a two-digit year — to a four-digit format (yyyy).

The optional yy-cutoff argument is the year cutoff used to delineate centuries; if yy meets or exceeds this cutoff value, the result will be 19yy; if yy is less than the cutoff, the result will be 20yy. The default cutoff value if no second argument is given will be 50.

The optional yy-execution-time argument (a numeric integer data item or literal) The default execution time value if no third argument is given will be now equivalent to specifying (FUNCTION NUMVAL (FUNCTION CURRENT-DATE (1:4))).

All arguments must be numeric data items or numeric literals.

8.1.105 BOOLEAN-OF-INTEGER

BOOLEAN-OF-INTEGER Function Syntax

BOOLEAN-OF-INTEGER(argument-1 argument-2)

This option is not yet implemented.

The included file NEWS will indicate when it is.

BOOLEAN-OF-INTEGER returns a boolean item of usage bit representing the binary value of *argument-1*. argument-2 specifies the length of the boolean data item that is returned.

argument-1 must be a positive integer.

argument-2 must be a positive non-zero integer

Returned value is a boolean item of usage bit that has the same bit configuration as the binary representation of the value of argument-1, where the rightmost boolean position is the low-order binary digit. The boolean value is zero-filled or truncated on the left, if necessary, in order to return a boolean item whose length is specified by argument-2 in therms of boolean positions.

8.1.106 CHAR-NATIONAL

CHAR-NATIONAL Function Syntax

CHAR-NATIONAL(argument-1)

This option is not yet implemented.

The included file NEWS will indicate when it is.

CHAR-NATIONAL returns a one character value that is a character in the national program collating sequence having the ordinal position equal to the value of the argument.

argument-1 must be a integer and greater than zero and less than or equal to the number of positions in the national program collating sequence.

8.1.107 DISPLAY-OF

DISPLAY-OF Function Syntax

DISPLAY-OF(argument-1 [argument-2])

This option is not yet implemented.

The included file NEWS will indicate when it is.

DISPLAY-OF returns a character string containing the alphabetic coded character set representation of the national characters in the argument.

argument-1 must be of class national.

argument-2 must be a of class alphabetic or alphanumeric and is one character position in length. It specifies an alphanumeric substitution character for use in conversion of national characters for which there is no corresponding alphanumeric character.

A character string is returned with each national character of argument-1 converted to its corresponding alphanumeric character representation, if any.

If argument-2 is specified, the alphanumeric substitution character is returned for each na- tional character in argument-1 that has no corresponding alphanumeric character representation.

If argument-2 is un-specified, and argument-1 contains a national character for which there is no corresponding alphanumeric character representation, an substitution character is used as the corresponding alphanumeric character and the EC-DATA-CONVERSION exception condition is set.

The length of the returned value is the number of character positions of usage display required to hold the converted argument and depends on the number of characters contained in argument- 1.

8.1.108 EXCEPTION-FILE-N

EXCEPTION-FILE-N Function Syntax

EXCEPTION-FILE-N
~~~~~~~~~~~~~~~~

This option is not yet implemented.

The included file NEWS will indicate when it is.

EXCEPTION-FILE-N returns a national character string that is the I/O status value and file-name of the file connector, if any, associated with the last exception status.

The value returned has a length that is based on its contents and the concents are as follows:

If the last exception status is not an EC-I-O exception condition, the returned value is two national zeros.

The returned value is two national spaces when the last exception status indicates an EC-I-O exception condition that originates from one of the following statements:

- a RAISE statement.
- an EXIT or a GOBACK statement with a RAISING phrase that specifies an EC-I-O exception-name.

Otherwise the returned value is a character string that is as long as is needed to contain the I-O status value and the filename. The first two characters are the I-O status value in national characters. The succeeding characters contain the file-name exactly as specified in the SELECT clause converted at runtime to the runtime national character set.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.
~~~~~~~~~~~~~~~~

8.1.109 EXCEPTION-LOCATION-N

EXCEPTION-LOCATION-N Function Syntax

EXCEPTION-LOCATION-N

This option is not yet implemented.

The included file NEWS will indicate when it is.

EXCEPTION-LOCATION-N returns an national character string containing exception information from the most-recently failing statement. The information is returned to a 1023 character string in one of the following formats, depending on the nature of the failure:

- primary-entry-point-name; paragraph OF section; statement-number
- primary-entry-point-name; section; statement-number
- primary-entry-point-name; paragraph; statement-number
- primary-entry-point-name; statement-number

Since this function has no arguments, no parenthesis should be specified.

The program must be compiled with the -debug switch, -ftraceall switch or -g switch for this function to return any meaningful information.

The documentation of the CBL_ERROR_PROC built-in system subroutine (see Section 8.2.27 [CBL ERROR PROC], page 575) built-in subroutine illustrates the use of this function.

8.1.110 INTEGER-OF-BOOLEAN

INTEGER-OF-BOOLEAN Function Syntax

INTEGER-OF-BOOLEAN(argument-1)

This option is not yet implemented.

The included file NEWS will indicate when it is.

INTEGER-OF-BOOLEAN returns the numeric value of the boolean string in argument-1 which is class boolean.

Returned value as argument-1 is assigned to a temporary boolean data item of usage bit described with the same number of boolean positions.

The unsigned binary value represented by the same bit configuration as the bit configuration of that temporary boolean data item is determined.

Note: Binary representation is a mathematical concept. It is not required that this representation be the same as a COBOL representation.

8.1.111 NATIONAL-OF

NATIONAL-OF Function Syntax

NATIONAL-OF(argument-1 [argument-2])

This option is not yet implemented.

The included file NEWS will indicate when it is.

———————————————————————————————————————

NATIONAL-OF returns a character string containing the national character representation of the characters in the argument which must be of class boolean.

A character string is returned with each alphanumeric character in argument-1 converted to its corresponding national coded character set representation.

If argument-2 is specified, each character in argument-1 that has no corresponding national representation is converted to the substitution character specified by argument-2.

If argument-2 is unspecified and argument-1 contains an alphanumeric character for which there is no corresponding national character representation, a substitution character is used as the corresponding national character and the EC-DATA-CONVERSION exception condition is set to exist.

The length of the returned value is the number of character positions of usage national required to hold the converted argument and depends on the number of characters contained in argument-1.

8.1.112 STANDARD-COMPARE

STANDARD-COMPARE Function Syntax

STANDARD-COMPARE(argument-1 argument-2 [ordering-name-1] [argument-4])

This option is not yet implemented.

The included file NEWS will indicate when it is.

STANDARD-COMPARE returns a character indicating the result of comparing argument-1 as a alphanumeric and argument-2 using a cultural ordering table.

1. *argument-1* shall be of class alphabetic, alphanumeric, or national.
2. *argument-2* shall be of class alphabetic, alphanumeric, or national.
3. *argument-1* and argument-2 may be of different classes.
4. Neither argument-1 nor argument-2 shall be a zero-length literal.
5. *ordering-name-1*, if specified, shall be associated with a cultural ordering table in the ORDER TABLE clause of the SPECIAL-NAMES paragraph. ordering-name-1 identifies the ordering table to be used for the comparison. If ordering-name-1 is not specified, the default ordering table 'ISO14651 2010 TABLE1' described in Appendix A of ISO/IEC 14651:2011 shall be used.
6. *argument-4*, if specified, shall be a positive nonzero integer.

Returned values:

1. If argument-4 is unspecified, the highest level defined in the ordering table is used for the comparison.
2. If the cultural ordering table is not available on the processor, or the specified ordering level is not available, or the level number specified by argument-4 is not defined in the ordering table, the EC-ORDER-NOT-SUPPORTED exception condition is set.
3. If the arguments are of different classes, and one is national, the other argument is converted to class national for purposes of comparison.
4. For purposes of comparison, trailing spaces are truncated from the operands except that an operand consisting of all spaces is truncated to a single space.
5. *argument-1* and argument-2 are compared in accordance with the ordering table and or- dering level being used.

   Note: This comparison is culturally sensitive and the default ordering table is acceptable for most cultures. It is not necessarily a character-by-character comparison and not necessarily a case-sensitive comparison. In order to use this function, users should understand the types of comparisons specified by ISO/IEC 14651:2D11 and the ordering tables in use for their installation.

6. The returned value is:

   '=' the arguments compare equal,

   '-=.:' *argument-1* is less than argument-2,

   ':>' *argument-1* is greater than argument-2.

7. The length of the returned value is 1.

8.2 Built-In System Subroutines

There are a number of built-in system subroutines included with GnuCOBOL.

Generally, these routines are intended to match those available in Micro Focus COBOL, ACUCOBOL and directly for GnuCOBOL.

It is recommended to change the CBL_OC routines to CBL_GC for forward compatibility as at some point they will be removed as they are a hangover from Open Cobol.

Prefix explanation:

C$ –> ACU

CBL_ –> MF

CBL_GC_ (For backwards compatibility some routines are also available as CBL_OC_ as well): but these wonderful extensions are only available with GnuCOBOL.

These routines, all executed via their upper-case names via the CALL statement (see Section 7.8.5 [CALL], page 290), are capable of performing the following functions:

- Changing the current directory
- Copying files
- Creating a directory
- Creating, Opening, Closing, Reading and Writing byte-stream files
- Deleting directories (folders)
- Deleting files
- Determining how many arguments were passed to a subroutine
- Getting file information (size and last-modification date/time)
- Getting the length (in bytes) of an argument passed to a subroutine
- Justifying a field left-, right- or center-aligned
- Moving files (a destructive "copy")
- Putting the program "to sleep", specifying the sleep time in seconds
- Putting the program "to sleep", specifying the sleep time in nanoseconds; Caveat : although you'll express the time in nanoseconds, Windows systems will only be able to sleep at a millisecond granularity
- Retrieving information about the currently-executing program
- Submitting a command to the shell environment appropriate for the version of GnuCOBOL you are using for execution

Early versions of Micro Focus COBOL allowed programmers to access various runtime library routines by using a single two-digit hexadecimal number as the entry-point name. These were known as call-by-number routines. Over time, Micro Focus COBOL evolved, replacing most of the call-by-number routines with ones accessible using a more conventional call-by-name technique.

Most of the call-by-number routines have evolved into even more powerful call-by-name routines, many of which are supported by GnuCOBOL.

Some of the original call-by-number routines never evolved call-by-name equivalents; GnuCOBOL supports some of these routines. See all currently used in Appendix's C and D. (Items marked as **).

The following sections describe the various built-in subroutines. All subroutine arguments are mandatory except where explicitly noted to the contrary. Any subroutine returning a value to the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) could utilize

the RETURNING clause on the CALL statement to return the result back to the full-word binary data item of your choice.

8.2.1 C$CALLEDBY

C$CALLEDBY Built-In Subroutine Syntax

```
CALL "C$CALLEDBY" USING prog-name-area
```

This routine returns the name of the program that called the currently-executing program. The program name will be returned, left-justified and space filled, in prog-name-area argument, which should be a PIC X elementary item or a group item. If prog-name-area is too small to receive the entire program name, the program name value will be truncated (on the right) to fit.

The RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to one of the following values:

-1 An error occurred. The prog-name-area contents will be unchanged.

0 The program calling C$CALLEDBY was not called by any other program (in other words, it is a main program). The prog-name-area contents will be set entirely to spaces.

1 The program calling C$CALLEDBY was indeed called by another program, and that pro- gram's name has been saved in prog-name-area.

8.2.2 C$CHDIR

C$CHDIR Built-In Subroutine Syntax

```
CALL "C$CHDIR" USING directory-path, result
```

This routine makes directory-path (an alphanumeric literal or identifier) the current directory.

The return code of the operation is returned both in the result argument (any non-edited numeric identifier) as well as in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260). The return code of the operation will be either 0=Success or 128=failure.

The directory change remains in effect until the program terminates (in which the orig- inal current directory at the time the program was started will be automatically restored) or until another C$CHDIR or a CBL_CHANGE_DIR built-in system subroutine (see Section 8.2.18 [CBL CHANGE DIR], page 566) is executed.

8.2.3 C$COPY

C$COPY Built-In Subroutine Syntax

```
CALL "C$COPY" USING src-file-path, dest-file-path, 0
```

Use this subroutine to copy file src-file-path to dest-file-path as if it were done via the cp (Unix/OSX) or COPY (Windows) command.

Both file path arguments may be alphanumeric literals or identifiers.

The third argument is required, but is unused.

If the attempt to copy the file fails (for example, it or the destination directory doesn't exist), the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 128; on successful completion it will be set to 0.

8.2.4 C$DELETE

C$DELETE Built-In Subroutine Syntax

```
CALL "C$DELETE" USING file-path, 0
```

This routine deletes the file specified by the file-path argument (an alphanumeric literal or identifier) just as if that were done using the rm (Unix/OSX) or ERASE (Windows) command.

The second argument is required, but is unused.

If the attempt to delete the file fails (for example, it doesn't exist), the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 128; on successful completion it will be set to 0.

8.2.5 C$FILEINFO

C$FILEINFO Built-In Subroutine Syntax

```
CALL "C$FILEINFO" USING file-path, file-info
```

With this routine you may retrieve the size of the file specified as the file-path argument (an alphanumeric literal or identifier) and the date/time that file was last modified. File size information may not be available in the particular GnuCOBOL build / Operating System combination you are using and may therefore always be returned as zero. The information is returned to the file-info argument, which is defined as the following 16-byte area:

```
01 File-Info.
    05 File-Size-In-Bytes   PIC 9(18) COMP.
    05 Mod-YYYYMMDD         PIC 9(8)  COMP. *> Modification Date
    05 Mod-HHMMSS00         PIC 9(8)  COMP. *> Modification Time
```

The last two decimal digits in the modification time will always be 00.

If the subroutine is successful, a value of 0 will be returned in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260). Failure to retrieve the needed statistics on the file will cause a RETURN-CODE special register value of 35 to be passed back. Supplying less than two arguments will generate a 128 RETURN-CODE special register value.

8.2.6 C$GETPID

C$GETPID Built-In Subroutine Syntax

CALL "C$GETPID"
~~~~

———————————————————————————————————————

Use this subroutine to return the PID (process ID) of the executing GnuCOBOL program. The PID value is returned into the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

There are no arguments to this routine.
~~~~

8.2.7 C$JUSTIFY

C$JUSTIFY Built-In Subroutine Syntax

```
CALL "C$JUSTIFY" USING data-item, "justification-type"
```

Use C$JUSTIFY to left, right or center-justify an alphabetic, alphanumeric or numeric edited data-item. The optional justification-type argument indicates the type of the justification to be performed. Its value is interpreted as follows:

'C' the value will be centered

'R' the value will be right-justified, space-filled to the left

'L' the value will be left-justified, space-filled to the right

If it begins with anything else, or is absent, it will be treated as if it is present and begins with a capital 'R'

8.2.8 C$MAKEDIR

C$MAKEDIR Built-In Subroutine Syntax

```
CALL "C$MAKEDIR" USING dir-path
```

With this routine you may create a new directory — the name of which is supplied as the *dir-path* argument (an alphanumeric literal or identifier).

Only the lowest-level directory (last) in the specified path can be created — all others must already exist. This subroutine will not behave as a mkdir -p (Unix) or mkdir /p (Windows).

The RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

8.2.9 C$NARG

C$NARG Built-In Subroutine Syntax

```
CALL "C$NARG" USING arg-count-result
```

This subroutine returns the number of arguments passed to the program that calls it back to in the numeric field arg-count-result. When called from within a user-defined function, a value of one (1) is returned if any arguments were passed to the function or a zero (0) otherwise.

When called from a main program, the returned value will always be 0.

8.2.10 C$PARAMSIZE

C$PARAMSIZE Built-In Subroutine Syntax

```
CALL "C$PARAMSIZE" USING argument-number
```

This subroutine returns the size (in bytes) of the subroutine argument supplied using the *argument-number* parameter (a numeric literal or data item).

The size is returned in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

If the specified argument does not exist, or an invalid argument number is specified, a value of 0 is returned.

8.2.11 C$PRINTABLE

C$PRINTABLE Built-In Subroutine Syntax

```
CALL "C$PRINTABLE" USING data-item [ , char ]
```

The C$PRINTABLE subroutine converts the contents of the data-item specified as the first argument to printable characters. Those characters that are deemed printable (as defined by the character set used by data-item) will remain unchanged, while those that are NOT printable will be converted to the character specified as the second argument.

If no char argument is provided, a period ('.') will be used.

Note: CBL_GC_PRINTABLE replaces this although it is currently still supported for legacy reasons.

8.2.12 C$SLEEP

C$SLEEP Built-In Subroutine Syntax

```
CALL "C$SLEEP" USING seconds-to-sleep
~~~~               ~~~~~
```

C$SLEEP puts the program to sleep for the specified number of seconds and/or fractions of a second. The seconds-to-sleep argument may be a numeric literal or data item.

Sleep times less than 1 will be interpreted as 0, subject to the speed of the CPU and the O/S (Operating System) used, as well as the timing of the generated C code, which will immediately returns control to the calling program without any sleep delay.

When using a variable argument defined as 9(n)v9(m) where n is maximum seconds in 7 days, i.e., (60 x 60 x 24 x 7) = 604,800 (seconds) and m is at a point too fast for the CPU and O/S. In practice m should be 2 for a hundredth of a second but actual testing against the target CPU would be needed.

The maximum time can be adjusted by the define MAX_SLEEP TIME during compilation of the compiler [and no I do not know where it is in the codebase] e.g.:

```
/* maximum sleep time in seconds, currently 7 days */
#define MAX_SLEEP_TIME 3600*24*7
```

Extract from a working program :-

This routine completes a process on newly arrived files at 30 minutes past each hour When complete sleeps for 60 minutes - time past nn:30:..

```
*>   In WS ---
*> Computed last finished, based on Sleep secs
*>      changes depend on UPDATE run & finished time.
*>
 01   WS-Cycle-Process-Data.
      03   WS-Sleep-Minutes      pic  9(5)          value 60.      *>Default 60 = 1 hour
      03   WS-Sleep-Seconds      pic  9(5)          value  zeros.
      03   WS-Cycle-Start        pic  99            value  30.    *> time to run past the      hour
      03   WS-Cycle-Secs         pic  s9(6)         value  zeros.
      03   WS-Tmp-Secs1          pic  s9(6)         value  zero.
      03   WS-Rerun-Cycles-No    pic  s999          value  -1.
      03   WS-Tmp-Sleep          pic  9(6)          value  zero.
*>
 01   Current-Date-And-Time.
      03   CDT-DateTime.                                *> 16
           05   CDT-Year         pic  9(4).
           05   CDT-Month        pic  9(2).  *>  01-12
           05   CDT-Day          pic  9(2).  *>  01-31
           05   CDT-Hour         pic  9(2).  *>  00-23
           05   CDT-Minutes      pic  9(2).  *>  00-59
           05   CDT-Seconds      pic  9(2).  *>  00-59
```

```
        05   CDT-Huns-Of-Secs pic 9(2). *> 00-99
    03  filler redefines CDT-DateTime.
        05   CDT-DDT            pic 9(8).
        05   filler             pic x(8).
    03  filler.
        05 CDT-GMT-Diff-HH  pic S9(2)     sign leading separate.
        05 CDT-GMT-Diff-MMM pic 9(2). *> 00 or 30

In PD ---

*>
*> Now if set to sleep do so,
*>   for normal processing multi (24) times per day
*>
*> First use minutes & at end convert to secs
*>
                move      FUNCTION Current-Date to Current-Date-And-Time
                compute   WS-Tmp-Secs1 = WS-Cycle-Secs - ((CDT-Minutes * 60) + CDT-Seconds) if
                          WS-Tmp-Secs1 is negative
                       or WS-Tmp-Secs1 = zero
                          add          WS-Sleep-Seconds to WS-Tmp-Secs1
                end-if
                move      WS-Tmp-Secs1 to WS-Tmp-Sleep
*>
                move      spaces to Log-Msg
                string    "Time Now "
                          CDT-Hour
                          ":"
                          CDT-Minutes
                          ":"
                          CDT-Seconds
                          space
                          "Sleep time "
                          WS-Tmp-Sleep
                                          into  Log-Msg
                  display     FUNCTION TRIM (Log-Msg) at 1601 with erase eol
  *> Code skipped
*>
*> Allow ESCape to terminate after NEXT update cycle but time will be out 1 sec
*>
                accept    Accept-Reply at line WS-Lines col 42 TIME-OUT 1 if
                          Cob-Crt-Status = Cob-Scr-Esc
                          go to AA040-EOJ
                end-if
                call      "C$SLEEP" using WS-Tmp-Sleep
                go to     AA032-Test-Update
    end-if
```

8.2.13 C$TOLOWER

C$TOLOWER Built-In Subroutine Syntax

```
CALL "C$TOLOWER" USING data-item, BY VALUE convert-length
```

This routine will converts the convert-length (a numeric literal or data item) leading characters of data-item (an alphanumeric identifier) to lower-case.

The convert-length argument must be specified BY VALUE (see Section 7.8.5 [CALL], page 290). Any characters in data-item after the convert-length point will remain unchanged.

If convert-length is negative or zero, no conversion will be performed.

8.2.14 C$TOUPPER

C$TOUPPER Built-In Subroutine Syntax

```
CALL "C$TOUPPER" USING data-item, BY VALUE convert-length
```

This routine will converts the convert-length (a numeric literal or data item) leading characters of data-item (an alphanumeric identifier) to upper-case.

The convert-length argument must be specified BY VALUE (see Section 7.8.5 [CALL], page 290). Any characters in data-item after the convert-length point will remain unchanged.

If convert-length is negative or zero, no conversion will be performed.

8.2.15 CBL_ALARM_SOUND

CBL_ALARM_SOUND Built-In Subroutine Syntax

CALL "CBL_ALARM_SOUND" USING ???
~~~~ ~~~~~

??? More information needed from compiler developers. ???

———————————————————————————————————————

This routine will create a noise (a beep) using the internal speaker if present. [Place Holder].
~~~~

8.2.16 CBL_AND

CBL_AND Built-In Subroutine Syntax

```
CALL "CBL_AND" USING item-1, item-2, BY VALUE byte-length
~~~~               ~~~~~                      ~~~~~
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	0
0	1	0
1	0	0
1	1	1

This subroutine performs a bit-by-bit logical AND operation between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the AND process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.17 CBL_BELL_SOUND

CBL_BELL_SOUND Built-In Subroutine Syntax

```
CALL "CBL_BELL_SOUND" USING     ???
~~~~                        ~~~~~
```

??? More information needed from compiler developers. ???

———

This routine will create a noise (a beep) using the internal speaker if present. [Place Holder].
Works the same as function CBL_ALARM_SOUND, so there.

8.2.18 CBL_CHANGE_DIR

CBL_CHANGE_DIR Built-In Subroutine Syntax

```
CALL "CBL_CHANGE_DIR" USING directory-path
```

This routine makes directory-path (an alphanumeric literal or identifier) the current directory.

The return code of the operation, which will be either 0=Success or 128=failure, is returned in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

The directory change remains in effect until the program terminates (in which the original current directory at the time the program was started will be automatically restored) or until another CBL_CHANGE_DIR or a C$CHDIR built-in system subroutine (see Section 8.2.2 [C$CHDIR], page 549) is executed.

8.2.19 CBL_CHECK_FILE_EXIST

CBL_CHECK_FILE_EXIST Built-In Subroutine Syntax

```
CALL "CBL_CHECK_FILE_EXIST" USING file-path, file-info
```

With this routine you may retrieve the size of the file specified as the file-path argument (an alphanumeric literal or identifier) and the date/time that file was last modified. File size information may not be available in the particular GnuCOBOL build / Operating System combination you are using and may therefore always be returned as zero.

The information is returned to the file-info argument, which is defined as the following 16-byte area:

```
01 file-info.
    05 File-Size-In-Bytes   PIC 9(18)  COMP.
    05 Mod-DD               PIC 9(2)   COMP.  *> Modification Date
    05 Mod-MO               PIC 9(2)   COMP.
    05 Mod-YYYY             PIC 9(4)   COMP.
    05 Mod-HH               PIC 9(2)   COMP.  *> Modification Time
    05 Mod-MM               PIC 9(2)   COMP.
    05 Mod-SS               PIC 9(2)   COMP.
    05 FILLER               PIC 9(2)   COMP.  *> Always 00
```

If the subroutine is successful, a value of 0 will be returned in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260). Failure to retrieve the needed statistics on the file will cause a RETURN-CODE special register value of 35 to be passed back. Supplying less than two arguments will generate a 128 RETURN-CODE special register value.

8.2.20 CBL_CLOSE_FILE

CBL_CLOSE_FILE Built-In Subroutine Syntax

```
CALL "CBL_CLOSE_FILE" USING file-handle
```

The CBL_CLOSE_FILE subroutine closes a byte stream file previously opened by either the CBL_OPEN_FILE built-in system subroutine (see Section 8.2.49 [CBL_OPEN_FILE], page 604) or CBL_CREATE_FILE built-in system subroutine (see Section 8.2.23 [CBL_CREATE_FILE], page 571) subroutines.

If the file defined by the file-handle argument (a PIC X(4) USAGE COMP-X data item) was opened for output, an implicit CBL_FLUSH_FILE built-in system subroutine (see Section 8.2.29 [CBL_FLUSH_FILE], page 579) will be performed before the file is closed.

If the subroutine is successful, a value of 0 will be returned in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260). Failure will cause a RETURN-CODE special register value of -1 to be passed back.

An example of the use of OPEN, CLOSE, READ etc., can be found in Contribs program printcbl.cbl which is basically also in cobxref.cbl as program 3.

8.2.21 CBL_COPY_FILE

CBL_COPY_FILE Built-In Subroutine Syntax

CALL "CBL_COPY_FILE" USING src-file-path, dest-file-path

Use this subroutine to copy file src-file-path to dest-file-path as if it were done via the cp (Unix/OSX) or COPY (Windows) command.

Both arguments may be alphanumeric literals or identifiers.

If the attempt to copy the file fails (for example, it or the destination directory doesn't exist), the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 128; on successful completion it will be set to 0.

8.2.22 CBL_CREATE_DIR

CBL_CREATE_DIR Built-In Subroutine Syntax

```
CALL "CBL_CREATE_DIR" USING dir-path
~~~~                  ~~~~~
```

With this routine you may create a new directory — the name of which is supplied as the *dir-path* argument (an alphanumeric literal or identifier).

Only the lowest-level directory (last) in the specified path can be created — all others must already exist. This subroutine will not behave as a mkdir -p (Unix) or mkdir /p (Windows).

The RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

8.2.23 CBL_CREATE_FILE

CBL_CREATE_FILE Built-In Subroutine Syntax

```
CALL "CBL_CREATE_FILE" USING file-path, 2, 0, 0, file-handle
~~~~                   ~~~~~
```

The CBL_CREATE_FILE subroutine creates the new file specified using the file-path argument and opens it for output as a byte-stream file usable by CBL_WRITE_FILE built-in system subroutine (see Section 8.2.58 [CBL_WRITE_FILE], page 613).

Arguments 2, 3 and 4 should be coded as the constant values shown. CBL_CREATE_FILE is actually a special-case of the CBL_OPEN_FILE built-in system subroutine (see Section 8.2.49 [CBL_OPEN_FILE], page 604) routine — see that routine for a description of the meanings of arguments 2, 3 and 4.

A file-handle (PIC X(4) USAGE COMP-X) will be returned, for use on any subsequent CBL_WRITE_FILE built-in system subroutine (see Section 8.2.58 [CBL_WRITE_FILE], page 613) or CBL_CLOSE_FILE built-in system subroutine (see Section 8.2.20 [CBL_CLOSE_FILE], page 568) calls.

The success or failure of the subroutine will be reported back in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260), with a value of -1 indicating an invalid argument and a value of 0 indicating success.

8.2.24 CBL_DELETE_DIR

CBL_DELETE_DIR Built-In Subroutine Syntax

```
CALL "CBL_DELETE_DIR" USING dir-path
```

This subroutine deletes an empty directory.

The only argument — dir-path (an alphanumeric literal or identifier) — is the name of the directory to be deleted.

Only the lowest-level directory (last) in the specified path will be deleted, and that directory must be empty to be deleted.

The RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to the return code of the operation; the value will be either 0=Success or 128=failure.

8.2.25 CBL_DELETE_FILE

CBL_DELETE_FILE Built-In Subroutine Syntax

```
CALL "CBL_DELETE_FILE" USING file-path
~~~~                   ~~~~~
```

This routine deletes the file specified by the file-path argument (an alphanumeric literal or identifier) just as if that were done using the rm (Unix/OSX) or ERASE (Windows) command.

If the attempt to delete the file fails (for example, it doesn't exist), the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 128; on successful completion it will be set to 0.

8.2.26 CBL_EQ

CBL EQ Built-In Subroutine Syntax

```
CALL "CBL_EQ" USING item-1, item-2, BY VALUE byte-length
~~~~              ~~~~~                      ~~~~~
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	1
0	1	0
1	0	0
1	1	1

This subroutine performs a bit-by-bit comparison between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the EQ process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.27 CBL_ERROR_PROC

CBL_ERROR_PROC Built-In Subroutine Syntax

CALL "CBL_ERROR_PROC" USING function, program-pointer

This routine registers a general error-handling routine.

The function argument must be a numeric literal or a 32-bit binary data item (USAGE BINARY-LONG, for example) with a value of 0 or 1. A value of 0 means that you will be registering ("installing") an error procedure while a value of 1 indicates you're de-registering ("uninstalling") a previously-installed error procedure.

The program-pointer must be a data item with a USAGE (see Section 6.9.59 [USAGE], page 228) of PROGRAM-POINTER containing the address of your error procedure. This item should be given a value using the SET Program-Pointer statement (see Section 7.8.43.2 [SET Program-Pointer], page 380). If the error procedure is written in GnuCOBOL, it must be a subroutine, not a user-defined function.

A success (0) or failure (non-0) result will be passed back in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

A custom error procedure will trigger when a runtime error condition is encountered. An error procedure may be registered by a main program or a subprogram, but regardless of from where it was registered, it applies to the overall program compilation group and will trigger when a runtime error occurs anywhere in the executable program. If the error procedure was defined by a subprogram, that program must be loaded at the time the error procedure is executed.

An error procedure may be used to take whatever actions might be warranted to display additional information or to gracefully close down work in progress, but it cannot prevent the termination of program execution; should the error procedure not issue its own STOP RUN, control will return back to the standard error routine when the error procedure exits.

The code within the handler will be executed and — once the handler issues a return, if it was written in C, or an EXIT PROGRAM statement (see Section 7.8.18 [EXIT], page 329) or GOBACK statement, if it was written in GnuCOBOL, the system-standard error handling routine will be executed.

Only one user-defined error procedure may be in effect at any time.

The following is a sample GnuCOBOL program that registers an error procedure. The output of that program is shown as well. As as you can see, the error handler's messages appear followed by the standard GnuCOBOL message.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoERRPROC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    Err-Proc-Address                    USAGE PROGRAM-POINTER.
PROCEDURE DIVISION.
S1.
    DISPLAY 'Program is starting'
    SET Err-Proc-Address TO ENTRY 'ErrProc'
    CALL 'CBL_ERROR_PROC' USING 0, Err-Proc-Address
```

```
12.         CALL 'Tilt' *> THIS DOESN'T EXIST!!!!
13.         DISPLAY 'Program is stopping'
14.         STOP RUN
15.         .
16.     END PROGRAM DemoERRPROC.
17.
18.     IDENTIFICATION DIVISION.
19.     PROGRAM-ID. ErrProc.
20.     PROCEDURE DIVISION.
21.     000-Main.
22.         DISPLAY 'Error: ' FUNCTION EXCEPTION-LOCATION
23.         DISPLAY '           ' FUNCTION EXCEPTION-STATEMENT
24.         DISPLAY '           ' FUNCTION EXCEPTION-FILE
25.         DISPLAY '           ' FUNCTION EXCEPTION-STATUS
26.         DISPLAY '*** Returning to Standard Error Routine ***'
27.         EXIT PROGRAM
28.         .
29.     END PROGRAM ErrProc.
```

When executed, this sample program generates the following console output.

```
E:\Programs\Demos>demoerrproc
Program is starting
Error: DemoERRPROC; S1; 12 CALL
        00
        EC-PROGRAM-NOT-FOUND
*** Returning to Standard Error Routine ***
DEMOERRPROC.cbl: 27: libcob: Cannot find module 'Tilt'

E:\Programs\Demos>
```

8.2.28 CBL_EXIT_PROC

CBL_EXIT_PROC Built-In Subroutine Syntax

CALL "CBL_EXIT_PROC" USING function, program-pointer

This routine registers a general exit-handling routine.

The function argument must be a numeric literal or a 32-bit binary data item (USAGE BINARY-LONG, for example) with a value of 0 or 1. A value of 0 means that you will be registering ("installing") an exit procedure while a value of 1 indicates you're deregistering ("uninstalling") a previously-installed exit procedure.

The program-pointer must be a data item with a USAGE (see Section 6.9.59 [USAGE], page 228) of PROGRAM-POINTER containing the address of your exit procedure.

A success (0) or failure (non-0) result will be passed back in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

An exit procedure, once registered, will trigger whenever a STOP RUN statement (see Section 7.8.46 [STOP], page 395) or a GOBACK statement (see Section 7.8.21 [GOBACK], page 334) is executed anywhere in the program. The exit procedure may execute whatever code is desired to undertake an orderly shut down of the program. Once the exit procedure terminates by executing an EXIT PROGRAM statement (see Section 7.8.18 [EXIT], page 329) or a GOBACK statement, the system-standard program termination routine will be executed.

Only one user-defined exit procedure may be in effect at any time.

The following is a sample GnuCOBOL program that registers an exit procedure. The output of that program is shown as well.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. demoexitproc.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    Exit-Proc-Address                    USAGE  PROGRAM-POINTER.
PROCEDURE DIVISION.
000-Register-Exit-Proc.
      SET Exit-Proc-Address TO ENTRY "ExitProc"
      CALL "CBL_EXIT_PROC" USING 0, Exit-Proc-Address IF
      RETURN-CODE NOT = 0
            DISPLAY 'Error: Could not register Exit Procedure'
      END-IF
      .
099-Now-Test-Exit-Proc.
      DISPLAY
            'Executing a STOP RUN...'
      END-DISPLAY
      GOBACK.
END  PROGRAM  demoexitproc.

IDENTIFICATION DIVISION.
PROGRAM-ID. ExitProc.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01   Display-Date                PIC XXXX/XX/XX.
01   Display-Time                PIC XX/XX/XX.
01   Now                         PIC X(8).
01   Today                       PIC X(8).
PROCEDURE DIVISION.
000-Main.
     DISPLAY '*** STOP RUN has been executed ***'
     ACCEPT Today FROM DATE YYYYMMDD
     ACCEPT Now     FROM TIME
     MOVE Today TO Display-Date
     MOVE Now   TO  Display-Time
     INSPECT Display-Time REPLACING ALL '/' BY ':'
     DISPLAY '***      ' Display-Date '                    '
     Display-Time '                    ***' GOBACK.
END PROGRAM ExitProc.
```

8.2.29 CBL_FLUSH_FILE

CBL_FLUSH_FILE Built-In Subroutine Syntax

```
CALL "CBL_FLUSH_FILE" USING file-handle
~~~~                  ~~~~~
```

In Micro Focus COBOL, calling this subroutine flushes any as-yet unwritten buffers for the (output) file whose file-handle is specified as the argument to disk.

This routine is non-functional in GnuCOBOL. It exists only to provide compatibility for applications that may have been developed for Micro Focus COBOL.

8.2.30 CBL_GC_FORK

CBL_GC_FORK Built-In Subroute Syntax

```
CALL "CBL_GC_FORK" USING Child-PID
~~~~                 ~~~~~
```

CBL_GC_FORK allows you to fork the current COBOL process to a new one.

The current content of the process's storage (including LOCAL-STORAGE) will be identical, any file handles get invalid in the new process, positions and file and record locks are only available to the original process.

This system routine is not available on Windows (exception: GCC on Cygwin).

Parameters: none

Returns: pid (the child process gets '0' returned, the calling process gets the pid of the created child).

Negative values are returned for system dependant error codes and -1 if the function is not available on the current system.

CBL_GC_FORK allows you to fork the current COBOL process to a new one. The current content of the process' storage (including LOCAL-STORAGE) will be identical, any file handles get invalid in the new process, positions and file / record locks are only available to the original process. This system routine is not available on Windows (exception: gcc on Cygwin). Parameters: none Returns: pid (the child process gets 0 returned, the calling process gets the pid of the created children). Negative values are returned for system dependant error codes and -1 if the function is not available on the current system.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. prog.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CHILD-PID      PIC S9(9) BINARY.
01  WAIT-STS       PIC S9(9) BINARY.
PROCEDURE DIVISION.
    CALL      "CBL_GC_FORK"  RETURNING CHILD-PID END-CALL
    EVALUATE TRUE
              WHEN CHILD-PID = ZERO
                   PERFORM CHILD-CODE
              WHEN CHILD-PID > ZERO
                   PERFORM PARENT-CODE
              WHEN CHILD-PID = -1
                   DISPLAY 'CBL_GC_FORK is not available on the current' '
                   system!'
                   PERFORM CHILD-CODE
                   MOVE 0 TO CHILD-PID
                   PERFORM PARENT-CODE
              WHEN  O T H E R
                   MULTIPLY -1 BY CHILD-PID END-MULTIPLY
                   DISPLAY 'CBL_GC_FORK returned system error: ' CHILD-PID
    END-EVALUATE
    STOP      RUN.
```

```
CHILD-CODE.
    CALL        "C$SLEEP" USING 1 END-CALL
    DISPLAY     "Hello, I am the child"
    MOVE        2 TO RETURN-CODE.
PARENT-CODE.
    DISPLAY     "Hello, I am the parent"
    CALL        "CBL_GC_WAITPID"  USING CHILD-PID RETURNING WAIT-STS
    MOVE        0  TO RETURN-CODE
    EVALUATE TRUE
                WHEN WAIT-STS >= 0
                    DISPLAY 'Child ended with status: ' WAIT-STS WHEN
                WAIT-STS = -1
                    DISPLAY 'CBL_GC_WAITPID is not available on the ' 'current
                    system!'
                WHEN WAIT-STS < -1
                    MULTIPLY -1 BY WAIT-STS END-MULTIPLY
                    DISPLAY 'CBL_GC_WAITPID returned system error: ' WAIT-STS
    END-EVALUATE.
```

8.2.31 CBL_GC_GETOPT

CBL_GC_GETOPT Built-In Subroutine Syntax

```
CALL "CBL_GC_GETOPT" USING BY REFERENCE SHORTOPTIONS LONGOPTIONS LONGIND
                           BY VALUE LONG-ONLY
                           BY REFERENCE RETURN-CHAR OPT-VAL
```

CBL_GC_GETOPT adapts the well-known option parser, getopt, to GnuCOBOL.

The usage of this system routine is described by the following example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG.
DATA DIVISION.
WORKING-STORAGE SECTION.
78   SHORTOPTIONS VALUE "jkl".
01   LONGOPTIONS.
     05 OPTIONRECORD OCCURS 2 TIMES.
        10  OPTIONNAME   PIC X(25).
        10  HAS-VALUE    PIC 9.
        10  VALPOINT     POINTER  VALUE NULL.
        10  RETURN-VALUE PIC X(4).
01   LONGIND             PIC 99.
01   LONG-ONLY           PIC 9 VALUE 1.
01   RETURN-CHAR         PIC X(4).
01   OPT-VAL             PIC X(10).
01   COUNTER             PIC 9 VALUE 0.
```

We first need to define the necessary fields for getopt's shortoptions, longoptions, longoption index (longind), long-only-option (long-only) and also the fields for return values return-char and opt-val (arbitrary size with trimming, see return codes).

The shortoptions are written down as an alphanumeric field (i.e., a string with arbitrary size) as follows:

```
"ab:c::d"
```

This means we want getopt to look for short options named 'a', 'b', 'c' or 'd', require an option value for 'b', and accept an optional one for 'c'.

The longoptions are defined as a table of records with oname, has-value, valpoint and val.[1]

oname defines the name of a longoption. has-value defines if an option value is demanded (has-val = 1), optional (has-val = 2) or not required (has-val = 0).

valpoint is a pointer used to specify an address to save getopt's return value to. The pointer is optional. If it is NULL, getopt returns a value as usual. If you use the pointer it has to point to a PIC X(4) field. The field val is a PIC X(4) character which is returned if the longoption was recognized.

The longoption structure is immutable! You can only vary the number of records.

[1] Say what? the discussion and code seem to have diverged.

Now we have the tools to run CBL_GC_GETOPT within the procedure division.

```
PROCEDURE DIVISION.
        MOVE      "version" to OPTIONNAME (1).
        MOVE      0  TO HAS-VALUE (1).
        MOVE      'V' TO RETURN-VALUE (1).
        MOVE      "verbose" TO OPTIONNAME (2).
        MOVE      0 TO HAS-VALUE (2).
        MOVE      'V' TO RETURN-VALUE (2).
        PERFORM  WITH TEST AFTER UNTIL RETURN-CODE = -1
                 CALL 'CBL_GC_GETOPT' USING
                      BY REFERENCE SHORTOPTIONS LONGOPTIONS LONGIND
                      BY VALUE LONG-ONLY
                      BY REFERENCE RETURN-CHAR OPT-VAL
                 END-CALL
                 DISPLAY RETURN-CHAR END-DISPLAY
                 DISPLAY OPT-VAL END-DISPLAY
        END-PERFORM
        STOP RUN.
```

The example shows how we initialize all parameters and call the routine until CBL_GC_GETOPT runs out of options and returns -1.

return-char might contain the following regular character if an option was recognized:

?	undefined or ambiguous option
1	non-option (only if first byte of so is '-')
0	valpoint != NULL and we are writing the return value to the specified address
-1	no more options (or reach the first non-option if first byte of shortoptions is '+')

The return-codes of CBL_GC_GETOPT are:

1	a non-option (only if first byte of so is '-')
0	valpoint != NULL and we are writing the return value to the specified address
-1	no more options (or reach the first non-option if first byte of shortoptions is '+')
2	truncated option value in opt-val (because opt-val was too small)
3	a regular answer from getopt

8.2.32 CBL_GC_HOSTED

CBL_GC_HOSTED Built-In Subroutine Syntax

```
CALL "CBL_GC_HOSTED" USING ARG-1       ARG-2
~~~~                       ~~~~~
```

Note replaces CBL_OC_HOSTED which is kept as a legacy item.

———

CBL_GC_HOSTED provides access to the following C hosted variables:

argc binary-long by value

argv pointer to char **

stdin, stdout, stderr
pointer

errno giving address of errno in pointer to binary-long, use based for more

Direct access and conditional access to the following variables:

tzname pointer to pointer to array of two char pointers

timezone C long, will be seconds west of UTC

daylight C int, will be 1 during daylight savings

The system will need HAVE TIMEZONE defined for these to return anything meaningful. Attempts made when they are not available will return 1 from CBL GC HOSTED.

It returns 0 when match, 1 on failure, case matters as does length, "arg" won't match.

The usage of this system routine is described by the following example.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. HOSTED.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  Argc        BINARY-LONG.
 01  Argv        POINTER.
 01  Stdin       POINTER.
 01  Stdout      POINTER.
 01  Stderr      POINTER.
 01  Errno       POINTER.
 01  Err         BINARY-LONG BASED.
 01  Domain      FLOAT-LONG VALUE 3.0.
 01  Tzname      POINTER.
 01  Tznames     POINTER BASED.
     05  Tzs     POINTER OCCURS 2.
 01  Timezone    BINARY-LONG.
 01  Daylight    BINARY-SHORT.
*>
 PROCEDURE DIVISION.
     call     "CBL_GC_HOSTED" using stdin "stdin"
     display  "stdin : " stdin
```

```
        call        "feof" using by value stdin
        display     "feof stdin : " return-code
        call        "CBL_GC_HOSTED" using stdout "stdout"
        display     "stdout : " stdout
        call        "fprintf" using by value stdout by content "Hello" &
        x"0a" call     "CBL_GC_HOSTED" using stderr "stderr"
        display     "stderr : " stderr
        call        "fprintf" using by value stderr by content "on err" & x"0a"
        call        "CBL_GC_HOSTED" using argc "argc"
        display     "argc : " argc
        call        "CBL_GC_HOSTED" using argv "argv"
        display     "argv : " argv
        call        "args"  using  by  value  argc  argv call
                    "CBL_GC_HOSTED" using errno "errno"
        display     "&errno : " errno
        set         address of err to errno
        display     "errno : " err
        call        "acos" using by value domain
        display     "errno after acos(3.0): " err ", EDOM is 33" call
                    "CBL_GC_HOSTED" using argc "arg"
        display     "'arg' lookup : " return-code
        call        "CBL_GC_HOSTED" using null "argc"
        display     "null with argc : " return-code
        display     "argc is still : " argc
*> the following only returns zero if the system has HAVE_TIMEZONE set call
                    "CBL_GC_HOSTED" using daylight "daylight "
        display     "'timezone' lookup : " return-code
        if          return-code not = 0
                    display "system doesn't has timezone"
        else
                    display "timezone is : " timezone
                    call "CBL_GC_HOSTED" using daylight "daylight
                    " display "'daylight' lookup : " return-code
                    display "daylight is : " daylight
                    set environment "TZ" to "PST8PDT"
                    call static "tzset" returning omitted on exception
                              continue end-call
                    call "CBL_GC_HOSTED" using tzname
                    "tzname" display "'tzname' lookup : "
                    return-code
*> tzs(1) will point to z"PST" and tzs(2) to z"PDT"
                    if     return-code equal 0 and tzname not equal null then set
                           address of tznames to tzname
                           if     tzs(1) not equal null then
                                  display "tzs #1 : " tzs(1)
                           end-if
                           if     tzs(2) not equal null then
                                  display "tzs #2 : " tzs(2)
                           end-if
                    end-if
        end-if
        goback.
```

end program hosted.

Note that the legacy name of this routine that starts with CBL_OC is deprecated, as is NANOSLEEP but will still work. It is recommended that all library routines names starting with CBL_OC are replaced with CBL_GC to minimise issues.

8.2.33 CBL_GC_NANOSLEEP

CBL_GC_NANOSLEEP Built-In Subroutine Syntax

```
CALL "CBL_GC_NANOSLEEP" USING nanoseconds-to-sleep
~~~~                    ~~~~~
```

Note replaces CBL_OC_NANOSLEEP which is kept as a legacy item.

This subroutine puts the program to sleep for the specified number of nanoseconds.

The effective granularity of nanoseconds-to-sleep values will depend upon the granularity of the system clock your computer is using and the timing granularity of the operating system that computer is running.

For example, you will not expect to see any difference between values of 1, 100, 500 or 1000, but you should see a difference between values such as 250000000 and 500000000.

The nanoseconds-to-sleep argument is a numeric literal or data item.

There are one billion nanoseconds in a second, so if you wanted to put the program to sleep for 1/4 second you'd use a nanoseconds-to-sleep value of 250000000.

Note that the legacy name of this routine starts with "CBL_OC" is deprecated, as is HOSTED, but will still work. It is recommended that all library routines names starting with "CBL_OC" are replaced with "CBL_GC" to minimise issues.

8.2.34 CBL_GC_PRINTABLE

CBL_GC_PRINTABLE Built-In Subroutine Syntax

```
CALL "CBL_GC_PRINTABLE" USING data-item [ , char ]
```

Note replaces C$PRINTABLE which is kept as a legacy item.

The CBL_GC_PRINTABLE subroutine converts the contents of the data-item specified as the first argument to printable characters.

Those characters that are deemed printable (as defined by the character set used by data-item) will remain unchanged, while those that are not printable will be converted to the character specified as the second argument.

If no char argument is provided, a period ('.') will be used.

8.2.35 CBL_GC_SCR_DUMP

CBL GC SCR DUMP Built-In Subroutine Syntax

CALL "CBL_GC_SCR_DUMP" USING file-name, return-code
~~~~ ~~~~~

———

Use this subroutine to writes the current contents of the screen to the file named by file-name. This function prepares the file that will be used with the CBL GC SCR RESTORE function.

The following is a sample GnuCOBOL program that shows how to use the DUMP and RESTORE routines.

```
>>SOURCE FORMAT IS FREE
REPLACE ==:BCOL:== BY ==with BACKGROUND-COLOR==
        ==:FCOL:== BY ==FOREGROUND-COLOR==.
IDENTIFICATION DIVISION.
program-id. SCRDUMPRESTORE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 Black constant as 00.
01 Green constant as 02.
01 wBco          pic 9(02) value Green.
01 wFco          pic 9(02) value Black.
01 wLin          pic 99 value 05.
01 wCol          pic 99 value 10.
01 wScreenName PIC X(256) value space. 01
wScrOk           BINARY-LONG  value  zero.
01 wScrOk9       pic 99.
01 .
    03 Bmess pic x(30) occurs 20 value space.

PROCEDURE DIVISION.
    *> D I S P L A Y 1st S C R E E N
    move Green to wBco move Black to wFco
    move '111111111111111111111111111111' to
    Bmess(1) move '>>>>>>>>>> MESSAGE 1 <<<<<<<<<' to
    Bmess(2) move '111111111111111111111111111111'
    to Bmess(3) move
    '111111111111111111111111111111' to Bmess(4)
    move '111111111111111111111111111111' to
    Bmess(5) move '111111111111111111111111111111'
    to Bmess(6) move '------------------------------' to Bmess(7)
    perform DisplayMessage thru DisplayMessageEx
    display 'PRESS ENTER TO DUMP 1st SCREEN ' at line 01 col 01 :BCOL: 07 :FCOL: black accept omitted
    *> save 1st screen
    move 'DUMPSCREEN.TMP' & x'00' to wScreenName
    call 'CBL_GC_SCR_DUMP' using by reference wScreenName returning wScrOk
    end-call display '1st SCREEN DUMPED - RETURN CODE IS: ' at line 01 col 01
    :BCOL: 07 :FCOL: b
    move wScrOk to wScrOk9
    display wScrOk9 at line 01 col 41 :BCOL: 07 :FCOL: black
```
~~~~

```
    accept omitted
    display 'PRESS ENTER TO DISPLAY 2nd SCREEN          ' at line 01 col 01
    :BCOL: 07 : accept omitted
    *> D I S P L A Y 2nd S C R E E N DISPLAY ' '
    AT 0101 WITH ERASE EOS
    move 12 to wLin move 20 to wCol move
    04 to wBco move Black to wFco
    move '+----------------------------+' to Bmess(1) move
    '|22222222222222222222222222222|' to Bmess(2)
    move '|22222222222222222222222222222|' to Bmess(3)
    move '| MESSAGE 2 |' to Bmess(4)
    move '|22222222222222222222222222222|' to Bmess(5)
    move '|22222222222222222222222222222|' to Bmess(6)
    move '+----------------------------+' to Bmess(7)
    perform DisplayMessage thru DisplayMessageEx
    display 'PRESS ENTER TO RESTORE 1st SCREEN          ' at line 01 col 01
    :BCOL: 07 : accept omitted
    *> restore 1st screen
    call 'CBL_GC_SCR_RESTORE' using by reference wScreenName returning wScrOk end-
    call CALL 'CBL_DELETE_FILE' USING wScreenName
    display '1st SCREEN RESTORED - RETURN CODE IS: ' at line 01 col 01 :BCOL: 07 :FCOL:
    move wScrOk to wScrOk9
    display wScrOk9 at line 01 col 41 :BCOL: 07 :FCOL: black accept
    omitted
    STOP RUN.

DisplayMessage.
    display Bmess(1) at line wLin + 1 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(2) at line wLin + 2 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(3) at line wLin + 3 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(4) at line wLin + 4 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(5) at line wLin + 5 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(6) at line wLin + 6 col wCol :BCOL: wBco :FCOL: wFco
    display Bmess(7) at line wLin + 7 col wCol :BCOL: wBco :FCOL: wFco
    continue.
DisplayMessageEx. exit.
```

8.2.36 CBL_GC_SCR_RESTORE

CBL_GC_SCR_RESTORE Built-In Subroutine Syntax

```
CALL "CBL_GC_SCR_RESTORE" USING file-name, return-code
```

Use this subroutine to restore the screen to the way it looked in the dump file which must have been created using the CBL_GC_SCR_DUMP function.

8.2.37 CBL_GC_SET_SCR_SIZE

CBL_GC_SET_SCR_SIZE Built-In Subroutine Syntax

CALL "CBL_GC_SET_SCR_SIZE" USING no-of-lines, no-of-cols

Use this subroutine to set the current console screen size.

When the system is running in a windowed environment, this will be the sizing of the console window in which the program is executing. When the system is not running a windowing environment, the physical console screen attributes will be set. In environments such as a Windows console window, where the logical size of the window may far exceed that of the physical console window, the size set will be that for the physical console window.

The size data must be in binary form or any other numeric forms to be accepted.

The following are possibly typical no-of-lines and no-of-columns definitions:

```
01  NO-OF-LINES        USAGE  BINARY-CHAR UNSIGNED.
01  NO-OF-COLUMNS      USAGE  BINARY-CHAR UNSIGNED.
```

This system call will only work if the terminal program used is compatible for such operations and not all are.

8.2.38 CBL_GC_WAITPID

CBL_GC_WAITPID Built-In Subroutine Syntax

```
CALL "CBL_GC_WAITPID" USING ARG-1
~~~~                  ~~~~~
                      RETURNING RET-STATUS
                      ~~~~~~~~~
```

CBL_GC_WAITPID allows you to wait until another system process ended.

Additionally you can check the process's return code.

Parameters: none

Returns: function-status / child-status

Negative values are returned for system dependant error codes and -1 if the function is not available on the current system.

```
CALL      "CBL_GC_WAITPID"  USING CHILD-PID RETURNING WAIT-STS
MOVE      0  TO RETURN-CODE
DISPLAY   'CBL_GC_WAITPID ended with status: ' WAIT-STS
```

8.2.39 CBL_GET_CSR_POS

CBL_GET_CSR_POS Built-In Subroutine Syntax

CALL "CBL_GET_CSR_POS" USING cursor-locn-buffer
~~~~ ~~~~~

This subroutine will retrieve the current cursor location on the screen, returning a 2-byte value into the supplied cursor-locn-buffer. The first byte of cursor-locn-buffer will receive the current line (row) location while the second receives the current column location.

The returned location data will be in binary form, and will be based upon starting values of 0, meaning that if the cursor is located at line 15, column 12 at the time this routine is called, a value of (14,11) will be returned.

The following is a typical cursor-locn-buffer definition:

```
01  CURSOR-LOCN-BUFFER.
    05 CURSOR-LINE          USAGE  BINARY-CHAR UNSIGNED.
    05 CURSOR-COLUMN        USAGE  BINARY-CHAR UNSIGNED.
```

Values of 1 (Line) and 1 (column) will be returned if GnuCOBOL was not generated to include screen I/O.
~~~~

8.2.40 CBL_GET_CURRENT_DIR

CBL_GET_CURRENT_DIR Built-In Subroutine Syntax

```
CALL "CBL_GET_CURRENT_DIR" USING BY VALUE 0,
                                 BY VALUE length,
                                 BY REFERENCE buffer
```

This retrieves the fully-qualified pathname of the current directory, saving up to length characters of that name into buffer.

The first argument is unused, but must be specified. It must be specified BY VALUE (see Section 7.8.5 [CALL], page 290).

The length argument must be specified BY VALUE. The buffer argument must be specified BY REFERENCE.

The value specified for the length argument (a numeric literal or data item) should not exceed the actual length of buffer argument.

If the value specified for the length argument is LESS THAN the actual length of buffer argument, the current directory path will be left-justified and space filled within the first length bytes of buffer — any bytes in buffer after that point will be unchanged.

If the routine is successful, a value of 0 will be returned to the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260). If the routine failed because of a problem with an argument (such as a negative or 0 length), a value of 128 will result. Finally, if the 1st argument value is anything but zero, the routine will fail with a 129 value.

8.2.41 CBL_GET_SCR_SIZE

CBL_GET_SCR_SIZE Built-In Subroutine Syntax

```
CALL "CBL_GET_SCR_SIZE" USING no-of-lines, no-of-cols
~~~~                    ~~~~~
```

Use this subroutine to retrieve the current console screen size.

When the system is running in a windowed environment, this will be the sizing of the console window in which the program is executing. When the system is not running a windowing environment, the physical console screen attributes will be returned. In environments such as a Windows console window, where the logical size of the window may far exceed that of the physical console window, the size returned will be that of the physical console window. Two one-byte values will be returned — the first will be the current number of lines (rows) while the second will be the number of columns.

The returned size data will be in binary form.

The following are typical no-of-lines and no-of-columns definitions:

```
01  NO-OF-LINES        USAGE  BINARY-CHAR UNSIGNED.
01  NO-OF-COLUMNS      USAGE  BINARY-CHAR UNSIGNED.
```

GnuCOBOL run-time screen management must have been initialized prior to CALLing this routine in order to receive meaningful values. This means that a DISPLAY data-item statement (see Section 7.8.12.4 [DISPLAY data-item], page 305) or a ACCEPT data-item statement (see Section 7.8.1.4 [ACCEPT data-item], page 267) must have been executed prior to executing the CALL statement.

Zero values will be returned if the screen has not been initialized and values of 24 (lines) and 80 (columns) will be returned if GnuCOBOL was not generated to include screen I/O.

Maximum values for BINARY-CHAR is 255 and any excess to this will be wrong so if lines or columns exceed 255, results will not be valid.

8.2.42 CBL_IMP

CBL_IMP Built-In Subroutine Syntax

```
CALL "CBL_IMP" USING item-1, item-2, BY VALUE byte-length
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	1
0	1	1
1	0	0
1	1	1

This subroutine performs a bit-by-bit logical implies process between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the IMP process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.43 CBL_NIMP

CBL_NIMP Built-In Subroutine Syntax

CALL "CBL_NIMP" USING item-1, item-2, BY VALUE byte-length

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	0
0	1	0
1	0	1
1	1	0

This subroutine performs the negation of a bit-by-bit logical implies process between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the NIMP process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.44 CBL_NOR

CBL_NOR Built-In Subroutine Syntax

```
CALL "CBL_NOR" USING item-1, item-2, BY VALUE byte-length
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	1
0	1	0
1	0	0
1	1	0

This subroutine performs the negation of a bit-by-bit logical or' process between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the NOR process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.45 CBL_NOT

CBL_NOT Built-In Subroutine Syntax

```
CALL "CBL_NOT" USING item-1, BY VALUE byte-length
```

This subroutine "flips" the left-most 8*byte-length bits of item-1, changing 0 bits to 1, and 1 bits to 0. The changes are made directly in item-1.

The item-1 argument must be a data item. The length of item-1 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be passed using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-1 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.46 CBL_OC_GETOPT

CBL_GC_GETOPT Built-In Subroutine Syntax

```
CALL "CBL_GC_GETOPT" USING BY REFERENCE SHORTOPTIONS LONGOPTIONS LONGIND
                           BY VALUE LONG-ONLY
                           BY REFERENCE RETURN-CHAR OPT-VAL
```

Use function CBL_GC_GETOPT instead as redundant and will be removed.

8.2.47 CBL_OC_HOSTED

CBL_GC_HOSTED Built-In Subroutine Syntax

```
CALL "CBL_GC_HOSTED" USING ARG-1        ARG-2
~~~~                       ~~~~~
```

Note replaces CBL_OC_HOSTED which is kept as a legacy item.

Use function CBL_GC_HOSTED instead as redundant and will be removed.

8.2.48 CBL_OC_NANOSLEEP

CBL_GC_NANOSLEEP Built-In Subroutine Syntax

```
CALL "CBL_GC_NANOSLEEP" USING nanoseconds-to-sleep
```

Note replaces CBL_OC_NANOSLEEP which is kept as a legacy item.

Use function CBL_GC_NANOSLEEP instead as redundant and will be removed.

8.2.49 CBL_OPEN_FILE

CBL_OPEN_FILE Built-In Subroutine Syntax

```
CALL "CBL_OPEN_FILE" USING file-path, access-mode, 0, 0, handle
```

This routine opens an existing file for use as a byte-stream file usable by CBL_WRITE_FILE or CBL_READ_FILE.

The file-path argument is an alphanumeric literal or data-item.

The access-mode argument is a numeric literal or data item with a PIC X USAGE COMP-X (or USAGE BINARY-CHAR) definition; it specifies how you wish to use the file, as follows:

1 input (read-only)

2 output (write-only)

3 input and/or output

The third and fourth arguments would specify a locking mode and device specification, respectively, but they're not implemented in GnuCOBOL (currently, at least) — just specify each as 0.

The final argument (handle) is a PIC X(4) USAGE COMP-X item that will receive the handle to the file. That handle is used on all other byte-stream functions to reference this specific file.

A RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) value of -1 indicates an invalid argument, while a value of 0 indicates success. A value of 35 means the file does not exist.

An example of the use of OPEN, CLOSE, READ etc., can be found in Contribs program printcbl.cbl which is basically also in cobxref.cbl as program 3.

8.2.50 CBL_OR

CBL_OR Built-In Subroutine Syntax

```
CALL "CBL_OR" USING item-1, item-2, BY VALUE byte-length
~~~~              ~~~~~                      ~~~~~
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	0
0	1	1
1	0	1
1	1	1

This subroutine performs a bit-by-bit logical or process between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the OR process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.51 CBL_READ_FILE

CBL_READ_FILE Built-In Subroutine Syntax

```
CALL "CBL_READ_FILE" USING handle, offset, nbytes, flag, buffer
```

This routine reads nbytes of data starting at byte number offset from the byte-stream file defined by handle into buffer.

The handle argument (PIC X(4) USAGE COMP-X) must have been populated by a prior call to CBL_OPEN_FILE built-in system subroutine (see Section 8.2.49 [CBL_OPEN_FILE], page 604).

The offset argument (PIC X(8) USAGE COMP-X) defines the location in the file of the first byte to be read. The first byte of a file is byte offset 0 and MUST be preset to zero for first use.

The nbytes argument (PIC X(4) USAGE COMP-X) specifies how many bytes (maximum) will be read. If the flag argument is specified as 128, the size of the file (in bytes) will be returned into the file offset argument (argument 2) upon completion. Not all operating system/GnuCOBOL environments may be able to retrieve file sizes in such cases, a value of zero will be returned. The only other valid value for flags is 0. This argument may be specified either as a numeric literal or as a PIC X USAGE COMP-X data item.

Upon completion, the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 0 if the read was successful or to 10 if an "end-of-file" condition occurred. If a value of -1 is returned, a problem was identified with the subroutine arguments.

8.2.52 CBL_READ_KBD_CHAR

CBL_READ_KBD_CHAR Build-In Subroutine Syntax

```
CALL "CBL_READ_KBD_CHAR" USING char RETURNING status-code.
```

Waits until a character is typed from the terminal and then read it with no echo.

Parameters: char PIC X. Receives the character that was typed, in ASCII.

status-code PIC XX COMP-5.

If RETURNING is not used the RETURN-CODE special register receives the status-code where zero is success and non-zero it is not.

[Above information taken from MF WB manual].

8.2.53 CBL_RENAME_FILE

CBL_RENAME_FILE Built-In Subroutine Syntax

```
CALL "CBL_RENAME_FILE" USING old-file-path, new-file-path
```

You may use this subroutine to rename a file.

The file specified by old-file-path will be "renamed" to the name specified as new-file-path. Each argument may be an alphanumeric literal or data item.

Despite what the name of this routine might make you believe, this routine is more than just a simple "rename" — it will actually move the file supplied as the 1st argument to the file specified as the 2nd argument. Think of it as a two-step sequence, first copying the old-file-path file to the new-file-path file and then a second step where the old-file-path is deleted.

If the attempt to move the file fails (for example, it doesn't exist), the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 128; on successful completion it will be set to 0.

8.2.54 CBL_RUNTIME_ERROR

CBL RUNTIME ERROR Built-In Subroutine Syntax

```
CALL "CBL_RUNTIME_ERROR" USING      ???
~~~~                                ~~~~~
```

??? More information needed from compiler developers. ???

Text to be added...

8.2.55 CBL_SET_CSR_POS

CBL_SET_CSR_POS Build-In Subroutine Syntax

```
CALL "CBL_SET_CSR_POS" USING cursor-locn-buffer
~~~~                   ~~~~~
```

Set current cursor position on terminal.

This subroutine will set the cursor location on the screen, using a 2-byte value into the supplied cursor-locn-buffer. The first byte of cursor-locn-buffer is for the line (row) location while the second sets the column location.

The two byte data block must be in binary form, and will be based upon starting values of 0, meaning that if the routine is called with a value of (14,11) cursor will be located at line 15, column 12.

The following is a typical cursor-locn-buffer definition:

```
01  CURSOR-LOCN-BUFFER.
    05 CURSOR-LINE          USAGE  BINARY-CHAR UNSIGNED.
    05 CURSOR-COLUMN        USAGE  BINARY-CHAR UNSIGNED.
```

8.2.56 CBL_TOLOWER

CBL_TOLOWER Built-In Subroutine Syntax

```
CALL "CBL_TOLOWER" USING data-item, BY VALUE convert-length
```

This routine will convert the first convert-length (a numeric literal or data item) characters of *data-item* (an alpha-numeric identifier) to lower-case.

The convert-length argument must be specified BY VALUE (see Section 7.8.5 [CALL], page 290). It specifies how many (leading) characters in data-item will be converted — any characters after that will remain unchanged.

If convert-length is negative or zero, no conversion will be performed.

8.2.57 CBL_TOUPPER

CBL_TOUPPER Built-In Subroutine Syntax

```
CALL "CBL_TOUPPER" USING data-item, BY VALUE convert-length
```

This routine will convert the first convert-length (a numeric literal or data item) characters of *data-item* (an alpha-numeric identifier) to upper-case.

The convert-length argument must be specified BY VALUE (see Section 7.8.5 [CALL], page 290). It specifies how many (leading) characters in data-item will be converted — any characters after that will remain unchanged.

If convert-length is negative or zero, no conversion will be performed.

8.2.58 CBL_WRITE_FILE

CBL_WRITE_FILE Built-In Subroutine Syntax

```
CALL "CBL_WRITE_FILE" USING handle, offset, nbytes, 0, buffer
```

This routine writes nbytes of data from buffer to the byte-stream file defined by handle starting at byte number offset within the file.

The handle argument (PIC X(4) USAGE COMP-X) must have been populated by a prior call to CBL OPEN FILE. The offset argument (PIC X(4) USAGE COMP-X) defines the location in the file of the first byte to be written to. The first byte of a file is byte offset 0.

The nbytes argument (PIC X(4) USAGE COMP-X) specifies how many bytes (maximum) will be written.

Currently, the only allowable value for the flags argument is 0. This argument may be specified either as a numeric literal or as a PIC X(1) USAGE COMP-X data item.

Upon completion, the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) will be set to 0 if the write was successful or to 30 if an I/O error condition occurred. If a value of -1 is returned, a problem was identified with the subroutine arguments.

8.2.59 CBL_XOR

CBL_XOR Built-In Subroutine Syntax

```
CALL "CBL_XOR" USING item-1, item-2, BY VALUE byte-length
```

Old Arg 1 Bit	Old Arg 2 Bit	New Arg 2 Bit
0	0	0
0	1	1
1	0	1
1	1	0

This subroutine performs a bit-by-bit logical exclusive or process between the left-most 8*byte-length corresponding bits of item-1 and item-2, storing the resulting bit string into item-2. The truth table shown to the left documents the XOR process.

The item-1 argument may be an alphanumeric literal or a data item and item-2 must be a data item. The length of both item-1 and item-2 must be at least 8*byte-length.

The byte-length argument may be a numeric literal or data item, and must be specified using BY VALUE (see Section 7.8.5 [CALL], page 290).

Any bits in item-2 after the 8*byte-length point will be unaffected.

A result of zero will be passed back in theRETURN-CODE special register (see Section 7.7 [Special Registers], page 260).

8.2.60 EXTFH

EXTFH Built-In Subroutine Syntax

```
CALL "EXTFH" USING      opcode fcd
~~~~                ~~~~~
```

———————————————————————————————————————

The Callable File Handler (EXTFH), is primarily an API for file I-O and a kind of an industry standard.

It can be used to access to GnuCOBOL files by some other programming languages (like C, Visual Basic, Python ...) on several platforms.

It also allows to use GnuCOBOL LIBCOB's fileio routines from other compilers (AcuCobol, MicroFocus, IBM Cobol ...) and vice versa (which of course needs a license). This can also help during migration phases.

The EXTFH API also provides features such as: Record locking, Data compression, New Index creation, Relative Byte Address, ...

You can also write your own file handler and run it in place of EXTFH, when it conforms to the call interface defined in this chapter.

The Callable File Handler handles all file organizations: sequential, line sequential, relative and indexed. The file operations performed by EXTFH fit into the following categories: Open files, Read and Write records, Move through records, Delete records, File and records locking, Unlocking records, Close files.

GnuCOBOL EXTFH file handling include support a callable EXTFH interface also provided by several compilers including Micro Focus The GnuCOBOL external file handler can be directly invoked from COBOL too, using CALL "EXTFH"

To have the compiled program call yourfh() for file I/O use: cobc -fcallfh=yourfh

In turn yourfh() may call EXTFH() to use I/O functions from GnuCOBOL.

Support for the EXTFH includes support for FH--FCD and FH--KEYDEF.

Some sample use of EXTFH.

a. Write a GnuCOBOL program that accepts as parameters: file type (sequential, relative indexed etc ...), number and length of keys, both the primary and any alternative keys.

This GnuCOBOL program, always the same, will be able to create various types of files (sequential, relative, indexed) with a primary key that can be 5 bytes or 20 or 30 bytes long, with a record length of 100 bytes or 1000 bytes etc, etc, .. simply by accepting parameters with one or many ACCEPT statements ... with as an example an open output and a close.

b. Allow a Microfocus COBOL (or AcuCOBOL or RM Cobol or Realia COBOL ...) program to read an indexed file that was created with a GnuCOBOL program that used BDB or VB-ISAM.

Conversely, read from a GnuCOBOL program an indexed file created by a Microfocus COBOL program with its proprietary system for example IDX4. This would require that EXTFH be available in the non GnuCOBOL environment.

c. Generate file io statistics. Each I-O operation can record the number of times a read, write, delete, open - any file I-O statement.

You could record the start of the call to EXTFH, the return from EXTFH - even the before and after image of the record processed.

You can implement recording stats from the file activity in your EXTFH services. The intent to provide operation reporting performance / consumption.

Each I-O operation can record the number of times a read, write, delete, open - any file I-O statement. You could record the start of the call to EXTFH, the return from EXTFH - even the before and after image of the record processed.

If you use EXTFH you can gather the statistics and only need to recompile the program - no change in them needed at all.

You can create a new EXTFH module named "myiostats.cbl" that uses the "xfhfcd3.cpy" for its LINKAGE and to get the 78 values, which only counts the statistics and possibly write them out when a CLOSE was requested; filter in that program as you like (for example for the filename, filetype, ...) before adding to the statistics; end with CALL "EXTFH" USING fcd-record to let libcob do the actual file io; compile with with cobc -c myiostats.cbl .

Compile your programs with an additional -fcallfh=myiostats myiostats.o to route all io through them. Consequently it does not affect the GnuCOBOL's EXTFH.

To use EXTFH in any COBOL environment that needs to provide an option like GnuCOBOL has with -fcallfh or MF with its CALLFH directive (GnuCOBOL supports that when included in source files as well).

To be able to call another COBOL's EXTFH entry point from a GnuCOBOL module to allow the io be routed to that, this environment also needs this entry point and a way to call it directly or via a "to GnuCOBOL foreign" COBOL module that calls into that. So this part means that you can only use it from a GnuCOBOL module if there's some way to access it.

d. You could read from a Python or Visual Basic program etc etc ... an indexed file written with the GnuCOBOL program using BDB or VB-ISAM as along as you use the applicable structure to call into EXTFH (the C structure is part of libcob / common.h).

The COBOL syntax defining a file (the SELECT and the FD statement) cause the compiler to create the data areas needed to define the file and to hold the file records.

EXTFH uses 4 data areas during file operations: File Control Description (FCD), Record Area, Filename Area, Key Definition Block.

To use the EXTFH you must follow these steps:

a. Allocate data areas for the File Control Description (FCD), Record Area, Filename Area and Key Definition Block(for indexed files only).

b. Initialize all the data areas to binary zeros to ensure that EXTFH does not receive invalid values.

c. Fill in the fields in the Record Area, Filename Area, and Key Definition Block that are needed for the selected Operation Code.

d. Set the pointers in the File Control Description (FCD) to point at Record Area, Filename Area and Key Definition Block (for indexed files only)

e. Select an Operation Code to decide which file operation to be performed

f. Fill in the fields in the FCD that are needed for the selected Operation Code.

g. Call the File Handler.

h. Determine the success of the file I/O operation by checking the returned File Status

i. Process any data in the the Record Area and the FCD output fields

Use the following syntax to call the File Handler from COBOL: CALL "EXTFH" USING opcode fcd where opcode is a PIC X(2) COMP-X field specifying the operation code.

FILE CONTROL DESCRIPTION (FCD)

The FCD area contains information about the file in use, set the appropriate fields in the FCD before calling EXTFH.

After performing the specified operation, EXTFH completes the appropriate fields in the FCD before passing control back to the calling program.

All unused or reserved areas of the FCD must be set to binary zeros.

The FCD used on the call to open a file must be the one used on all subsequent calls to that file. The supplied GnuCOBOL copyfile XFHFCD3.CPY contains a COBOL definition of the FCD.

```
01 FCD-AREA.
    COPY 'xfhfcd3.cpy'.
```

The structure of the FCD is defined below, bit 7 is always the leftmost bit.

When using COBOL, this and the following pointers are USAGE POINTER items, set using a statement of the form: SET POINTER TO ADDRESS OF data item

where:

pointer is one of these fields,

data item is the relevant data area.

Offset and size are in bytes.

OFFSET	SIZE	DESCRIPTION
0	2	File Status Code. See also offsets 6 and 33. After every operation except a COMMIT or ROLLBACK operation, this field is updated with a standard file status value telling you the status of the operation.
2	3	Reserved. Must be set to binary zeros.
5	1	File organization: 0 Line sequential, 1 Sequential, 2 Indexed, 3 Relative
6	1	User Status and Access Mode indicators. User Status is indicated by bit 7. The bit is set if you have defined a file status. Defining a file status can affect how some operations (such as RETRYLOCK) are performed. When calling EXTFH directly, this bit should be set. however, file status is obtained directly from offset 0. Access mode is indicated by bits 6-0: 0 - Sequential access mode, 4 - Random access mode,8 - Dynamic access mode.
7	1	Open mode: you must set this field to 128 before opening a file. This field is written to by the Callable File Handler so that, on return, it indicates the open mode of the file: 0 INPUT, 1 OUTPUT, 2 I/O, 3 EXTEND, 128 File is closed.
8	3	Reserved. Must be set to binary zeros.
11	2	Length of filename.
13	8	Relative byte address. This field is used instead of the 4 byte field at offset 72 if bit 4 of byte 93 is set.
21	3	Reserved. Must be set to binary zeros.
24	1	Lock mode flags for shareable files: Bit 7 - Lock on multiple records

Bit 6 - WRITELOCK directive enabled
Bit 5 - RETRYOPEN
Bit 4 - SKIPLOCK directive enabled
Bit 3 - RETRYLOCK directive enabled
Bit 2 - Lock mode MANUAL
Bit 1 - Lock mode AUTOMATIC
Bit 0 - Lock mode EXCLUSIVE

25 1 Other flags:
Bit 7 - OPTIONAL file (OPEN INPUT)
Bit 6 - Reserved
Bit 5 - Not OPTIONAL (OPEN I/O and EXTEND)
Bit 4 - Filename is EXTERNAL
Bit 3 - Reserved
Bit 2 - NODETECTLOCK directive enabled
Bit 1 - Multiple Reel file (Record Sequential)
Bit 0 - Line Advancing file (Record Sequential).

26 2 Reserved. Must be set to binary zeros.

28 4 File handle

32 1 Reserved. Must be set to binary zeros.

33 1 File status type. See also offsets 0 and 6:
Bit 7 Set for ANSI'85 COBOL file status by default, otherwise ANSI'74 COBOL file status
Bits 6-3 Reserved
Bit 2 Enables tab insertion for line sequential files Bit 1 Enables null insertion for line sequential files Bit 0 Reserved.
Bits 1 and 2 are set/unset by EXTFH at open time, depending on the settings of the N and T RTS switches. These bits can be set/unset after an OPEN operation to override the N and T RTS switches.

34 1 File format:
0 - Operating system default. C-ISAM on UNIX, same as specifying a value of 3 on DOS, Windows and OS/2.
1 - C-ISAM format
2 - LEVEL II COBOL format
3 - Micro Focus COBOL format
4 - IDXFORMAT"4" format All platforms except UNIX:
5 - Btrieve format with ANSI emulation
6 - Btrieve format without ANSI emulation All platforms:
8 - IDXFORMAT"8" format
11 - Sequential files only. Mainframe print file.
14 - Heap file

35 3 Reserved. Must be set to binary zeros.

38 2 Maximum record length (in bytes).

40 3 Reserved. Must be set to binary zeros.
32-bit:

42 1 File descriptor. Returned for line sequential fixed length

sequential and relative files and C-ISAM files on UNIX.

43 4 Relative record number.

47	1	Recording mode: Bits 7-1 Reserved,Bit 0: 0 - Fixed, 1 - Variable
48	2	Current record length (in bytes).
50	2	Minimum record length (in bytes).
52	2	Key-of-Reference (indexed files). This field identifies the Key-of-Reference, used for random READ operations. To specify the prime key, set this field to zero. For example, to specify the first alternate key defined, use a value 1; the second alternate key defined use value 2. Or: Line count (Line Sequential files). This field specifies the number of lines to skip when writing a file. For example: WRITE AFTER ADVANCING line-count LINES
54	2	Effective key length. When using a START operation on indexed files, you can specify only the leading part of a key instead of the whole key. You should set this field to the number of bytes to be used in the comparison.It must be > than zero and no bigger than the key being used.
56	4	Pointer to the record area.
60	4	Pointer to the filename area.
64	4	Pointer to key definition block.
68	4	Reserved. Must be set to binary zeros.
72	4	Relative byte address. Unless bit 4 of byte 93 is set, in which case the relative byte address is held at offset 13
76	2	Reserved. Must be set to binary zeros.
78	1	Data compression routine indicator: 0 = No compression 1-127 = Micro Focus data compression routine number (1=CBLDC001) 128-255 = User-defined data compression routine number (128=USRDC128)
79	4	Fileshare V2 session-id.
83	2	Fileshare V2 file-id.
85	6	Reserved. Must be set to binary zeros.
91	1	Bit 7- Interlanguage locking (LOCKTYPE 1), Bits 6-0- Reserved.
92	1	Fileshare V2 flags: Bit 7 - Transaction logging; Bits 6-0 - Reserved.
93	1	Configuration flags: Bit 7 - WRITETHRU Bit 6 - Use Relative Byte Address Bit 5 - Update current record pointer Bit 4 - Use offset 13 for relative byte address, instead of offset 72 Bits 3-2 - Reserved Bit 1 - Check COBFSTATCONV Bit 0 – Set if IGNORELOCK required.
94	1	Reserved. Must be set to binary zeros.
95	1	Bit 7- Use EBCDIC collating sequence Bit 6- Set if file is to have WRITE AFTER ADVANCING Bit 5- Set if file is to have WRITE BEFORE ADVANCING

		Bit 4- ADV byte
		Bit 3- Ignore minimum length checking on variable-length files
		Bits 2-0- Reserved.
96	1	Index cache size.
97	1	Index cache area.
98	2	Reserved.

RECORD AREA

The record area is an area into which records are read, and from which records are (re)written.

The size of the record area must be four bytes larger than the largest record in the file. The record length fields in the FCD always contain true record lengths, not the length of the record area. (See offsets 38, 48 and 50 in the FCD.)

The following GnuCOBOL sample code shows how the FCD is set up to point to the record area:

```
01 wRECORD-AREA          PIC X(90).
   ---
   SET FCD-RECORD-ADDRESS TO ADDRESS OF wRECORD-AREA
   ---
```

FILENAME AREA

The filename area contains the name of the file in use. It can contain drive and/or path information as well as the actual name of the file. If the actual name is shorter than the length of the buffer specified in the FCD, it must be terminated by a space.

This data area must be filled before the first operation on the file.

The following GnuCOBOL sample code shows how the FCD is set up to point to the filename area:

```
01 wFILENAME-AREA        PIC X(60) VALUE "file001.dat".
   ---
   MOVE LENGTH OF wFILENAME-AREA      TO FCD-NAME-LENGTH
   SET FCD-FILENAME-ADDRESS TO ADDRESS OF wFILENAME-AREA
   ---
```

KEY DEFINITION BLOCK

The Key Definition BLOCK is used to hold index key information during operations on indexed files. It consists of three data areas: Global Information area, Key Definition area, Component Definition area.

Global Information Area.

It tells the size of the Key Definition Area and how many keys are in the file. All unused or reserved areas must be initialized to binary zeros. The structure is shown below.

OFFSET	SIZE	DESCRIPTION
0	2	Length of the Key Definition Block
2	4	Reserved. Must be set to binary zeros
6	2	Number of keys
8	6	Reserved. Must be set to binary zeros.

Key Definition Area Parameter Block

It describes the keys used in the indexed file and consists of one Key Definition for each key in the file. You must define all keys before their components. The order of defining the keys is important. The ordinal position of the key is used to identify it.

For example, if you have an indexed file with a prime key and two alternate keys, the Key Definition area would contain three key definitions. The prime key is key 0, the first alternate is key 1, and the second alternate is key 2. All unused or reserved areas must be initialized to binary zeros.

The structure is shown below.

```
OFFSET SIZE DESCRIPTION
------- ----- ------------------------------------------------------------------
0       2     Component count. For ordinary keys, the component count is 1. For split
              keys, the component count is the number of components making up the
              split key.
2       2     Offset to first Component Definition area for this key.
              This offset is relative to the start of the Global Information area,
              starting at 0.
4       1     Key flags:
              Bit 7 - Reserved. Must be set to binary zeros
              Bit 6 - Duplicates allowed
              Bit 5 - Reserved. Must be set to binary zeros
              Bit 4 - Set to indicate that this is the prime key. If this is
              not set for any key, the file handler assumes that the first
              key is the prime key.
              Bit 3 - Reserved. Must be set to binary zeros
              Bit 2 - Reserved. Must be set to binary zeros
              Bit 1 - Sparse key. See offset 6
              Bit 0 - Reserved. Must be set to binary zeros.
5       1     Compression flags:
              Bits 7-3 - Reserved. Must be set to binary
              zeros Bit 2 - Compression of trailing
              spaces
              Bit 1 - Compression of leading
              characters Bit 0 - Compression of
              duplicates
6       2     Sparse character. If bit 1 of the key flags (offset 4) is set,
              the key is suppressed if it is entirely made up of this
              character.
7       8     Reserved. Must be set to binary zeros.
```

Component Definition Area

It follows the Key Definition Area and contains one component definition for each key component. Each key consists of one component, unless defined as a split key, when each component of the key requires its own Component Definition.

The Component Definitions define the location in the record and length of the key component. All unused or reserved areas must be initialized to binary zeros.

The structure of a Component Definition is shown below.

```
OFFSET SIZE DESCRIPTION
------- ----- ------------------------------------------------------------------
```

0	2	Reserved. Must be set to binary zeros.

2 4 Offset of component in the record (starting at 0).
6 4 Length of the component (in bytes).

OPERATION CODES

There are two types of operation codes:

Standard: The code contain x"FA" in the leftmost byte; the least significant byte indicates the specific operation

Special: The codes contain x"00" in the leftmost significant byte; the least significant byte indicates the specific operation

Opcode A 2-character operation code specifying an exact operation PIC X(2) COMP-X .

Operation The operation performed by the chosen opcode.

File Type L Line Sequential, S Record Sequential(including vS), R Relative(including vR), I Indexed, vS Variable format Sequential only, vR Variable format Relative only

Input fields The offset values of fields in the FCD to be set before calling.

Output fields The offset values of fields set by EXTFH during the call.

STANDARD OPERATION CODES

They are identified by x"FA" in the leftmost significant byte of the operation code.

The least significant byte indicates the specific operation, as shown in the following sections.

OPEN

Initiates the processing of files.

OP CODE	OPERATION		FILE TYPE
00	OPEN	INPUT	L S R I
01	OPEN	OUTPUT	L S R I
02	OPEN	I-O	L S R I
03	OPEN	EXTEND	L S R I
04	OPEN	INPUT WITH NO REWIND	L S
05	OPEN	OUTPUT WITH NO REWIND	L S
08	OPEN	INPUT REVERSED	L S

Input Fields:

Offset 5 File organization
Offset 6 Access mode
Offset 7 Open mode. Must be set to 128 before opening a file.
Offset 11 Length of filename
Offset 24 Lock mode flags
Offset 25 Other flags
Offset 34 File format
Offset 38 Maximum record length
Offset 47 Recording mode
Offset 50 Minimum record length
Offset 60 Pointer to the filename area
Offset 64 Pointer to the key definition block (I)
Offset 78 Data compression
Offset 91 Interlanguage locking
Filename area
Key Definition Area (only for Indexed Files)

Output Fields:
Offset 0 User file status
Offset 7 Open mode
FCD Use for all subsequent accesses to this file

Opening a File More Than Once.

With EXTFH you can assign several FCDs to the same physical file and have them all open at the same time. The operating system counts it as just one open file. The physical file is not closed until every logical file assigned to it has been closed.

CLOSE

Terminates the processing of files.

OP CODE	OPERATION	FILE TYPE
80	CLOSE	L S R I
81	CLOSE WITH LOCK	L S R I
82	CLOSE WITH NO REWIND	L S
84	CLOSE REEL/UNIT	L S
85	CLOSE REEL/UNIT FOR REMOVAL	L S
86	CLOSE REEL/UNIT WITH NO REWIND	L S

Input Fields: None
Output Fields: Offset 0 - User file status

READ

Makes available a specified record (random access) or the next or previous logical record (sequential access).

OP CODE	OPERATION	FILE TYPE
8D	READ (sequential) WITH NO LOCK	L S R I
D8	READ (sequential) WITH LOCK	L S R I
D9	READ (sequential) WITH KEPT LOCK	L S R I
F5	READ (sequential)	L S R I
8C	READ PREVIOUS WITH NO LOCK	R I
DE	READ PREVIOUS WITH LOCK	R I
DF	READ PREVIOUS WITH KEPT LOCK	R I
F9	READ PREVIOUS	R I
8E	READ (random) WITH NO LOCK	R I
DA	READ (random) WITH LOCK	R I
DB	READ (random) WITH KEPT LOCK	R I
F6	READ (random)	R I
8F	READ (direct) WITH NO LOCK	L S R I
D6	READ (direct) WITH LOCK	L S R I
D7	READ (direct) WITH KEPT LOCK	L S R I
C9	READ (direct)	L S R I

F1 READ (position) L S R I

Input Fields:
Offset 43 Relative record number (R) if READ random or direct Offset 52 Key identifier (I)
Offset 56 Pointer to the record area

Output Fields:
Offset 0 User file status Offset 48 Current record length Offset 72 Relative byte address Record Area

WRITE/REWRITE

Releases a logical record for an output or input-output file. For sequential files, it can also be used for vertical positioning of lines in a logical page.

The REWRITE operation logically replaces a record existing in a disk file. The WRITE operation releases record locks and writes records.

OP CODE	OPERATION	FILE TYPE
E1	WRITE BEFORE	L S
E2	WRITE AFTER	L S
E3	WRITE BEFORE TAB	L S
E4	WRITE AFTER TAB	L S
E5	WRITE BEFORE PAGE	L S
E6	WRITE AFTER PAGE	L S
EC	WRITE BEFORE mnemonic name	S
ED	WRITE AFTER mnemonic name	S
F3	WRITE	L S R I
F4	REWRITE	L S R I

Input Fields:
Offset 43 Relative record number (R)
Offset 48 Current record length Offset 52 Line count (WRITE only) Offset 56 Pointer to the record area

Output Fields:
Offset 0 User file status Offset 72 Relative byte address

START

Provides a basis for logical positioning in a relative or indexed file, for subsequent retrieval of records.

OP CODE	OPERATION	FILE TYPE

E8	START	equal to full length prime key R I
E9	START	equal to (any key/record number) R I
EA	START	greater than (>) R I
EB	START	not less than (>=) R I
FE	START	less than (<) R I
FF	START	less than or equal to (<=) R I

Input Fields:
Offset 43 Relative record number (R)
Offset 52 Key identifier (I)
Offset 54 Effective key length (I)
Offset 56 Pointer to the record area

Output Fields:
Offset 0 User file status

STEP

Steps to the next physical record, this operation enables very fast access to a record.

OP CODE	OPERATION	FILE TYPE
90	STEP NEXT WITH NO LOCK	L S R I
D4	STEP NEXT WITH LOCK	L S R I
D5	STEP NEXT WITH KEPT LOCK	L S R I
CA	STEP NEXT	L S R I
92	STEP FIRST WITH NO LOCK	L S R I
D0	STEP FIRST WITH LOCK	L S R I
D1	STEP FIRST WITH KEPT LOCK	L S R I
CC	STEP FIRST	L S R I

Input Fields: Offset 56 Pointer to the record area
Output Fields: Offset 0 User file status

STEP

operations are a method of sequentially retrieving records in a relative or indexed file without having to read via a key or relative record number. Generally, this is a faster method of accessing data in the file.

A STEP NEXT operation returns the record that is physically stored immediately after the one that was last retrieved in the file (either by a STEP or READ operation).

If you are using relative byte addressing and you have specified that the current record pointer should be updated to the record you have just accessed, STEP operations are relative to this new position.

The relative byte address is returned on every STEP operation, so if you need to update the current record pointer to the record just retrieved, you can use the READ (direct) operation using the address returned. The current record pointer is not affected by STEP operations.

DELETE

Logically removes a record from a disk file.

OP CODE	OPERATION	FILE TYPE
F7	DELETE	vS R I

Input Fields
Offset 43 Non-sequential access to relative record number (R)
Offset 56 Non-sequential access to record pointer (I)

Output Fields:
Offset 0 User file status

DELETE FILE

Physically removes the specified file from the physical devices on which it resides.

OP CODE	OPERATION	FILE TYPE
F8	DELETE FILE	L S R I

Input Fields:
Offset 5 File organization
Offset 11 Filename length
Offset 34 File format
Offset 60 Pointer to the filename area

Filename Area
Output Fields:
Offset 0 User file status

COMMIT

Releases all record locks in all files held by this run unit. This operation code always requires use of the FCD.

OP CODE	OPERATION	FILE TYPE
DC	COMMIT (UNLOCK all files)	L S R I

Input Fields: Always requires FCD used by the file
Output Fields: None

ROLLBACK

Releases all record locks in all files held by this run unit.

OP CODE	OPERATION	FILE TYPE

```
----- ----------------------------------- --------
DD    ROLLBACK (UNLOCKs all files)          L S R I
```

Input Fields: None
Output Fields: None

UNLOCK

Releases all record locks held by the run unit on a named file.

```
OP                                          FILE
CODE OPERATION                              TYPE
----- ----------------------------------- --------
0E    UNLOCK                                L S R I
```

Input Fields: None

Output Fields: Offset 0 User file status

SPECIAL OPERATION CODES

They must be identified by the hexadecimal code x"00" in the most significant byte of the operation code.

The least significant byte indicates the operation, as shown in the following sections. GET

FILE INFO

Returns information on keys for the specified file in the format of the Key Definition Block for index files and general file information for all file types supported.

```
OP                                          FILE
CODE OPERATION                              TYPE
----- ----------------------------------- --------
06    Return file information               vS vR I
```

Input Fields:
Offset 5 File organization (can be x"FF")
Offset 11 Length of filename
Offset 60 Pointer to the filename area
Offset 64 Pointer to the key definition area (I)

Output Fields:
Offset 0 User file status
Offset 5 File organization

Key definition Area (I) Offset
34 File format
Offset 38 Maximum record length
Offset 47 Recording mode
Offset 50 Minimum record length
Offset 72 File size
Offset 78 Data compression routine

It is your responsibility to ensure that you have enough space allocated for the key definition block to hold all the information returned.

Failure to allocate enough space causes corruption of some data areas. For variable-length sequential files and variable-length relative files, no key definition information exists.

If the organization byte of the FCD is set to x"FF", this tries to determine the file type and return with the relevant information set. This mode of operation is not recommended if fixed-length sequential files are involved as it might be impossible to determine the difference between the first record of a fixed-length sequential file and the header of a variable-length sequential file. When the file type is determined as incompatible, the error code 9/161 is returned.

CREATE NEW INDEX

Creates a new .idx file, containing only header information for the file to allow new indexes to be added.

OP CODE	OPERATION	FILE TYPE
07	Open new index	I

Input fields: same as open
Output fields: same as open

GET NEXT RECORD

Gets the next physical record from the index data file.

OP CODE	OPERATION	FILE TYPE
08	Get next record	I

Input fields: offset 56 pointer to record area Output

fields:
offset 0 user file status offset
48 current record length offset
72 relative byte offset

ADD KEY VALUE

Adds key value to the index for the key specified

OP CODE	OPERATION	FILE TYPE
09	Add key value	I

Input fields:
offset 52 key identifier
offset 56 pointer to record area

Output fields: offset 0 user file status

FLUSH FILE

Ensure all data for a specific file is flushed to disk.

OP CODE	OPERATION	FILE TYPE
0C	Flush File	L S R I

Input Fields: None
Output Fields: None

UNLOCK RECORD

Unlocks a specific record in a file.

OP CODE	OPERATION	FILE TYPE
0F	Unlock record	I

Input Fields: Offset 56 Pointer to the record area

Output Fields: None

SAMPLE PROGRAM

EXTFH is very powerful, it requires many parameters to work and provides many return data.An example is very useful to better clarify all these aspects.

The following example also shows the COBOL structure of the FCD area and the Key Definition Area

The following is a single source EXTFH02.COB to be compiled with a single command but which includes two programs EXTFH02A and EXTFH02B.

The first program creates an indexed file with 10 records with the normal I-O statements and calls the second program.

The second program instead uses EXTFH for OPEN, READ REWRITE START READ PREVIOUS and CLOSE operations.

```
        >>SOURCE FREE
    IDENTIFICATION DIVISION.
    PROGRAM-ID. EXTFH02A.
    *>*********************************************************
    *> CREATE AN INDEXED FILE WITH 10 RECORDS FOR NEXT PROGRAM
    *>*********************************************************
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
    SELECT OUTFILE
        ASSIGN TO "FILE01EXTFH"
        ORGANIZATION IS INDEXED
```

```
        RECORD KEY     IS PRIME-KEY
        ACCESS MODE  IS
        SEQUENTIAL FILE STATUS    IS
        FS-OUT.
DATA DIVISION.
FILE SECTION.
FD OUTFILE.
01    OUT-REC.
        05    PRIME-KEY PIC X(25) VALUE ALL 'K'.
        05    IN-DATA      PIC X(10) VALUE ALL 'x'.
WORKING-STORAGE SECTION.
01    FS-OUT                PIC   X X    VALUE  s p a c e .
01    OUT-RECNUM      PIC 999 VALUE 0.
PROCEDURE DIVISION.
        OPEN OUTPUT OUTFILE.
        INITIALIZE OUT-REC ALL TO VALUE.
        perform 10 times
           MOVE  "RECORD-"     TO   PRIME-KEY (1:7)
           ADD 1     TO OUT-RECNUM MOVE OUT-RECNUM TO PRIME-KEY (8:3)
           DISPLAY 'WRITE: ' OUT-REC      WRITE  OUT-REC
        end-perform
        DISPLAY OUT-RECNUM ' records written'. CLOSE
        OUTFILE
        display 'Enter to continue...' accept omitted CALL
        'EXTFH02B'
        STOP RUN.
END  PROGRAM  E X T F H 0 2 A .

*>*************************************************************
*> USE CALL 'EXTFH' TO OPEN READ REWRITE START READPREV CLOSE
*>*************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. EXTFH02B.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 OperationCode.
        05    wOperationType     pic x(01).
              78    Standard-Type                    value x'FA'.
              78    Special-Type                     value x'00'.
        05    wOperationCode                pic x(01).
              *> Standard Codes (Type is x'FA')
              78    Open-Input                       value  x'00'.
              78    Open-Output                      value  x'01'.
              78    Open-I-O                         value  x'02'.
              78    Open-Extend                      value  x'03'.
              78    Open-Input-No-Rewind             value  x'04'.
              78    Open-Output-No-Rewind            value  x'05'.
              78    Open-Input-Reversed              value  x'08'.
              78    Close-File                       value  x'80'.
              78    Close-Lock                       value  x'81'.
              78    Close-No-Rewind                  value  x'82'.
              78    Close-Reel-Unit                  value  x'84'.
```

```
        78  Close-Reel-Unit-For-Removal value x'85'.
        78  Close-Reel-Unit-No-Rewind    value x'86'.
        78  Read-Seq-No-Lock             value x'8D'.
        78  Read-Seq-Lock                value x'D8'.
        78  Read-Seq-Kept-Lock           value x'D9'.
        78  Read-Seq                     value x'F5'.
        78  Read-Previous-No-Lock        value x'8C'.
        78  Read-Previous-Lock           value x'DE'.
        78  Read-Previous-Kept-Lock      value x'DF'.
        78  Read-Previous                value x'F9'.
        78  Read-Random-No-Lock          value  x'8E'.
        78  Read-Random-Lock             value  x'DA'.
        78  Read-Random-Kept-Lock        value x'DB'.
        78  Read-Random                  value  x'F6'.
        78  Read-Direct-No-Lock          value x'8F'.
        78  Read-Direct-Lock             value x'D6'.
        78  Read-Direct-Kept-Lock        value x'D7'.
        78  Read-Direct                  value x'C9'.
        78  Read-Position                value x'F1'.
        78  Write-Before                 value x'E1'.
        78  Write-After                  value x'E2'.
        78  Write-Before-Tab             value x'E3'.
        78  Write-After-Tab              value x'E4'.
        78  Write-Before-Page            value x'E5'.
        78  Write-After-Page             value x'E6'.
        78  Write-Record                 value x'F3'.
        78  Rewrite-Record               value x'F4'.
        78  Start-Equal-Prime-Key        value x'E8'.
        78  Start-Equal-Key              value x'E9'.
        78  Start-Greater                value x'EA'.
        78  Start-Not-Less               value x'EB'.
        78  Start-Less                   value x'FE'.
        78  Start-Less-Or-Equal          value x'FF'.
        78  Step-Next-No-Lock            value x'90'.
        78  Step-Next-Lock               value x'D4'.
        78  Step-Next-Kept-Lock          value x'D5'.
        78  Step-Next                    value x'CA'.
        78  Step-First-No-Lock           value x'92'.
        78  Step-First-Lock              value x'D0'.
        78  Step-First-Kept-Lock         value x'D1'.
        78  Step-First                   value x'CC'.
        78  Delete-Record                value x'F7'.
        78  Delete-File                  value x'F8'.
        78  Unlock-Current               value x'0E'.
        78  Commit-All                   value x'DC'.
        78  Rollback-All                 value x'DD'.
*> Special codes    (Type is x'00')
        78  Get-File-Info                value x'06'. *> not avail. for BDB files
        78  Open-New-Index               value x'07'.
        78  Get-Next-Record              value x'08'.
        78  Add-Key-Value                value x'09'.
```

```
            78   Unlock-Record                    value x'0F'.

      01    FCD-AREA. *> or use: COPY 'XFHFCD3.CPY'.
            10 FCD-FILE-STATUS.
               15 FCD-STATUS-KEY-1              pic  x.
               15 FCD-STATUS-KEY-2              pic  x.
               15 FCD-BINARY redefines FCD-STATUS-KEY-2 pic x comp-x.
            10 FCD-LENGTH                        pic xx comp-x.
            10 FCD-VERSION                       pic x comp-x.
               78 fcd--version-number           value 1.
            10 FCD-ORGANIZATION                  pic x comp-x.
               78 fcd--line-sequential-org value 0.
               78 fcd--sequential-org           value 1.
               78 fcd--indexed-org              value 2.
               78 fcd--relative-org             value 3.
               78  fcd--determine-org            value 255. *> see wOperationCode 0006:
            10 FCD-ACCESS-MODE                   pic x comp-x.
               78  fcd--sequential-access       value  0.
               78  fcd--dup-prime-access        value  1.
               78  fcd--random-access           value  4.
               78  fcd--dynamic-access          value  8.
               78  fcd--status-defined          value  h"80".
            10  FCD-OPEN-MODE                    pic x comp-x. *> open mode
               78  fcd--open-input              value  0.
               78  fcd--open-output             value  1.
               78  fcd--open-i-o                value  2.
               78  fcd--open-extend             value  3.
               78  fcd--open-max                value  3.
               78  fcd--open-closed             value  128.
            10 FCD-RECORDING-MODE                pic x comp-x. *> recording mode
               78 fcd--recmode-fixed            value 0.
               78 fcd--recmode-variable         value 1.
            10 FCD-FILE-FORMAT                   pic x comp-x.
               78 fcd--format-liiv1             value 0.
               78 fcd--format-cisam             value 1.
               78 fcd--format-liiv2             value 2.
               78 fcd--format-cobol2            value 3.
               78 fcd--format-idx4              value 4.
               78 fcd--format-btrieve-ansi value 5.
               78  fcd--format-btrieve-non-ansi     value 6.
      *>       78  fcd--format-rlio             value  7.
               78  fcd--format-big              value  8.
               78  fcd--format-leafrec          value  9.
               78  fcd--format-cst              value  10.
               78  fcd--format-mvs-print        value  11.
      *>       78                               value  13.
               78  fcd--format-heap             value  14.
               78  fcd--format-esds             value  15.
               78  fcd--format-qsamv            value  255.
               78  fcd--max-file-format         value  16. *> 1    > max permissible      format:
            10 FCD-DEVICE-FLAG                   Pic x comp-x.
```

```
   78  fcd--dev-normal              value  0.
   78  fcd--dev-device              value  1.
   78  fcd--dev-stdin               value  2.
   78  fcd--dev-stdout              value  3.
   78  fcd--dev-stderr              value  4.
   78  fcd--dev-badname             value  5.
   78  fcd--dev-input-pipe          value  6.
   78  fcd--dev-output-pipe         value  7.
   78  fcd--dev-i-o-pipe            value  8.
   78  fcd--dev-library             value  9.
   78  fcd--dev-disk-file           value  10.
   78  fcd--dev-null                value  11.
   78  fcd--dev-disk-redir          value  12.
   78  fcd--dev-no-map              value  13.
10 FCD-LOCK-ACTION                   pic x comp-x. *> Used only in c-isam type calls
   78 fcd--getlock                  value 1.
   78 fcd--nolock                   value 2.
   78 fcd--ignorelock               value 3.
10 FCD-DATA-COMPRESS                 pic x comp-x.
10 FCD-BLOCKING                      pic x comp-x.
10 FCD-additional-status redefines FCD-blocking pic x comp-x.
10 FCD-IDXCACHE-SIZE                 pic x comp-x.
10 FCD-PERCENT                       pic x comp-x.
10 FCD-REC-COUNT-SET redefines FCD-PERCENT     pic x comp-x.
10 FCD-BLOCK-SIZE                    pic x comp-x.
10 FCD-FLAGS-1                       pic x comp-x.
   78 fcd--mainframe-compat         value  h"80".
   78  fcd--ansi-line-adv           value h"40".
   78  fcd--return-key-only         value h"20".
   78 fcd--bypass-esds              value h"10".
   78 fcd--no-xfhname-mapping       value  h"08".
   78 fcd--dont-call-xfhtrace       value h"04".
   78 fcd--call-xfhtrace            value h"02".
   78 fcd--fcd-decl                 value h"01". *> declaratives exist:
10 FCD-FLAGS-2                       pic x comp-x.
   78   fcd--convert-dbspace        value h"01".
10 fcd-mvs-flags                     pic x comp-x. *> MVS flag bits
   78    fcd--file-is-syspunch      value  h"10".
   78    fcd--file-is-indd          value  h"08".
   78    fcd--file-is-outdd         value  h"04".
   78    fcd--amode-31bit           value  h"02".
   78    fcd--amode-24bit           value  h"01".
   78    fcd--amode-bits  value  fcd--amode-31bit  +  fcd--amode-24bit.
10 FCD-STATUS-TYPE                     pic x comp-x.
   78 fcd--ans85-status              value h"80".
   78 fcd--no-space-fill             value h"40".
   78 fcd--no-strip-spaces           value h"20".
   78 fcd--no-expand-tabs            value h"10".
   78  fcd--rec-term-bit             value h"08".
   78 fcd--insert-tabs               value h"04".
   78  fcd--insert-nulls             value h"02".
```

```
          78 fcd--cr-delimiter              value h"01".
          78 fcd--modify-writes      value fcd--insert-tabs + fcd--insert-nulls.
       10 FCD-OTHER-FLAGS                    pic x comp-x.
          78 fcd--optional-file             value h"80".
          78 fcd--nodetectlock-input        value h"40".
          78 fcd--not-optional              value h"20".
          78 fcd--external-name             value h"10".
          78 fcd--get-info                  value h"08".
          78 fcd--nodetectlock              value h"04".
          78 fcd--multiple-reel             value h"02".
          78 fcd--line-advancing            value h"01".
          78 fcd--special-sequential
               value fcd--optional-file + fcd--multiple-reel + fcd--line-advancing.
       10 FCD-TRANS-LOG                       pic x comp-x.
          78 fcd--open-input-shared         value h"80".
          78 fcd--allow-input-locks         value h"40".
          78 fcd--no-read-sema              value h"20".
          78 fcd--expand-positioning-bit value h"10".
          78 fcd--no-seq-check              value h"08".
          78  fcd--dat-term-bit             value h"04".
          78 fcd--slow-read                 value h"02".
          78 fcd--suppress-adv              value h"01".
       10 FCD-LOCKTYPES                       pic x comp-x.
          78 fcd--interlang-locking         value h"80".
          78 fcd--allow-readers             value h"40".
          78 fcd--separate-lock-file        value h"20".
          78 fcd--single-open               value h"10".
          78  fcd--nfs-file-lock            value h"08".
          78  fcd--nfs-file-lock-hp         value h"04".
          78  fcd--nfs-file-locks  value  fcd--nfs-file-lock  +  fcd--nfs-file-lock-hp.
       10 FCD-FS-FLAGS                        pic x comp-x.
          78 fcd--transaction-processing-bit value h"80".
          78 fcd--recovery-run-b            value h"04".
          78 fcd--fs-server-bit             value h"02".
       10 FCD-CONFIG-FLAGS                    pic x comp-x.
          78  fcd--writethru-bit            value  h"80".
          78  fcd--relative-bit             value  h"40".
          78  fcd--set-crp-bit              value  h"20".
          78  fcd--bigfile-bit              value  h"10".
*>        78  fcd--return-percent            value  h"08".
*>        78  fcd--dont-call-xfhconv         value  h"04".
          78  fcd--call-cobfstatconv        value  h"02".
          78  fcd--ignorelock-bit           value  h"01".
       10 FCD-MISC-FLAGS                      pic x comp-x.
          78 fcd--mainframe-hostfd          value h"80".
          78 fcd--set-idxdatbuf             value h"40".
          78 fcd--load-onto-heap            value h"20".
          78 fcd--usage-unknown             value  h"10".
          78 fcd--recmode-s                 value h"08".
          78 fcd--recmode-u                 value h"04".
          78 fcd--external-fcd              value h"02".
```

```
   78  fcd--closed-with-lock          value h"01".
10 FCD-CONFIG-FLAGS2                   pic x comp-x.
   78  fcd--file-is-ebcdic            value h"80".
   78 fcd--file-has-write-after value h"40".
   78 fcd--file-has-write-before value h"20".
   78 fcd--file-has-adv-specified value h"10".
   78 fcd--no-min-len-check          value h"08".
   78 fcd--no-key-check              value h"04".
   78  fcd--convert-to-ascii         value h"02".
   78 fcd--rm-behaviour              value h"01".
   78 fcd--file-has-before-or-after
        value fcd--file-has-write-before + fcd--file-has-write-after.
10 FCD-LOCK-MODE                       pic x comp-x.
   78  fcd--multilock-bit            value h"80".
   78  fcd--writelock-bit            value h"40".
   78 fcd--retry-open-bit            value h"20".
   78 fcd--skip-lock-bit             value h"10".
   78 fcd--retry-lock-bit            value h"08".
   78  fcd--manual-lock-bit          value h"04".
   78 fcd--auto-lock-bit             value h"02".
   78 fcd--exclusive-bit             value h"01".
   78 fcd--sharing-bits value fcd--manual-lock-bit + fcd--auto-lock-bit.
10 FCD-SHR2                            pic x comp-x.
   78  fcd--file-max-bit             value h"08".
   78 fcd--file-pointer-bit          value h"04".
   78  fcd--retry-time-bit           value h"02".
   78 fcd--start-unlock              value h"01".
10  FCD-IDXCACHE-BUFFS        pic  x  comp-x.
10  FCD-INTERNAL-FLAGS-1      pic  x  comp-x.
10  FCD-INTERNAL-FLAGS-2      pic  x  comp-x.
10                            pic  x  comp-x occurs     15.
10  FCD-NLS-ID                pic  xx  comp-x.      *>     NLS id (else 0)
10  FCD-FS-FILE-ID            pic  xx  comp-x.
10  fcd-retry-open-count      pic  xx  comp-x.
10  FCD-NAME-LENGTH           pic  xx  comp-x.
10  fcd-idxname-length        pic  xx  comp-x.
10  fcd-retry-count           pic  xx  comp-x.
*> Indexed key identifier for random READ
10 FCD-KEY-ID                 pic xx comp-x.
10 FCD-LINE-COUNT             pic xx comp-x. *> Line count (seq files)
10 FCD-USE-FILES              pic  x     comp-x.
10 FCD-GIVE-FILES             pic  x     comp-x.
10 FCD-KEY-LENGTH             pic xx comp-x. *> Effective key length
                                                 *>    (START KEY LENGTH IS n)
10                            pic x comp-x occurs 20.
10  FCD-CURRENT-REC-LEN       pic x(4) comp-x. *> Current record length
10 FCD-MIN-REC-LENGTH         pic x(4) comp-x. *> Minimum record length
10 FCD-MAX-REC-LENGTH         pic x(4) comp-x. *> Max record length
10 FCD-SESSION-ID             pic  x(4)  comp-x.
10                            pic x comp-x occurs 24.
10 FCD-RELADDR-OFFSET         pic  x(8)  comp-x.
```

```
        10 FCD-RELADDR              redefines FCD-RELADDR-OFFSET pic x(8) comp-x.
        10 FCD-RELADDR-BIG          redefines FCD-RELADDR-OFFSET pic x(8) comp-x.
        10 FCD-MAX-REL-KEY          pic  x(8)  comp-x.
        10 FCD-RELATIVE-KEY         pic  x(8)  comp-x.
        10 FCD-PTR-FILLER1          pic x(8) comp-x.
        10 FCD-HANDLE               redefines FCD-PTR-FILLER1 usage pointer.
        10  FCD-HANDLE-NUM         redefines  FCD-PTR-FILLER1  pic  x(4)  comp-x.
        10 FCD-PTR-FILLER2          pic  x(8)  comp-x.      *> Pointer to record area
        10 FCD-RECORD-ADDRESS       redefines FCD-PTR-FILLER2 usage pointer.
        10 FCD-PTR-FILLER3          pic x(8) comp-x.        *> Pointer to file name
        10 FCD-FILENAME-ADDRESS     redefines FCD-PTR-FILLER3 usage pointer.
        *> Pointer to index name (applies only if separate index file exists)
        10 FCD-PTR-FILLER4          pic x(8) comp-x.
        10 FCD-IDXNAME-ADDRESS      redefines FCD-PTR-FILLER4 usage pointer.
        10 FCD-INDEX-NAME           redefines FCD-PTR-FILLER4 usage pointer.
        10 FCD-PTR-FILLER5          pic x(8) comp-x.        *> Pointer to key def block
        10 FCD-KEY-DEF-ADDRESS      redefines FCD-PTR-FILLER5 usage pointer.
        10 FCD-PTR-FILLER6          pic x(8) comp-x.        *> Pointer to collating seq
        10 FCD-COL-SEQ-ADDRESS      redefines FCD-PTR-FILLER6 usage pointer.
        10 FCD-PTR-FILLER7          pic x(8) comp-x.        *> Pointer to using list
        10 FCD-FILDEF-ADDRESS       redefines FCD-PTR-FILLER7 usage pointer.
        10 FCD-PTR-FILLER8          pic x(8) comp-x.
        10 FCD-DFSORT-ADDRESS       redefines FCD-PTR-FILLER8 usage pointer.

    01  FILE-NAME                          PIC  X(64).
    01  RECORD-AREA.
        05   PRIME-KEY                      PIC  X(25).
        05   DATA-AREA                      PIC  X(10).
    01  RECORD-AREA2.
        05   PRIME-KEY2                     PIC  X(25).
        05   DATA-AREA2                     PIC  X(10).

    01 KEY-DEF-BLOCK.
       47 GLOBAL-INFORMATION-AREA.
          49 LENGTH-OF-KEY-DEF-AREA         pic  9(04)  comp-x.
          49 key-version                    pic 9(02) comp-x value 2.
          49 filler                         pic 9(06) comp-x. *> reserved
          49 NUMBER-OF-KEYS                 pic  9(04)  comp-x.
          49 filler                         pic 9(13) comp-x. *> reserved
*> key-def-area is repeated for the n.of keys defined by NUMBER-OF-KEYS
       47 KEY-DEF-AREA.
          49 component-count                pic  9(4)  comp-x.
*> The offset for the component-specification for this key
          49 component-defs                 pic 9(4) comp-x.
          49 key-flags                      pic 9(2) comp-x.
             78 KEY2KEYFLAG-DUPS-IN-ORDER       value  h"40".
             78 KEY2KEYFLAG-PRIME                value  h"10".
             78 KEY2KEYFLAG-SPARSE-KEY           value  h"02".
          49 key-compression pic 9(2) comp-x.
             78  KEY2COMPRESS-TRAILING-NULLS      value  h"08".
             78  KEY2COMPRESS-TRAILING-SPACES  value  h"04".
```

```
            78  KEY2COMPRESS-IDENTICAL-CHARS  value  h"02".
            78 KEY2COMPRESS-FOLLOWING-DUP      value    h " 0 1 " .
            78 KEY2COMPRESS-NO-COMPRESSION     value    h " 0 0 " .
            78 KEY2COMPRESS-DEFAULT value KEY2COMPRESS-NO-COMPRESSION.
        49 sparse-characters                pic x(2).
        49 filler                           pic x(8). *> reserved
*> component-specifications for all keys follows after the key definition area
*> for all the keys.
    47 COMPONENT-DEF-AREA.
        49 component-flags                  pic 9(2) comp-x.
        49 component-type                   pic 9(2) comp-x.
            78 KEY2PARTTYP-NUMERIC              value   h"80".
            78 KEY2PARTTYP-SIGNED               value   h"40".
            78 KEY2PARTTYP-COMP                 value   h"20".
            78 KEY2PARTTYP-COMP-3               value   h"21".
            78 KEY2PARTTYP-COMP-X               value   h"22".
            78 KEY2PARTTYP-COMP-5               value   h"23".
            78 KEY2PARTTYP-FLOAT                value   h"24".
            78 KEY2PARTTYP-COMP-6               value   h"25".
            78 KEY2PARTTYP-DISPLAY              value   h"00".
            78 KEY2PARTTYP-SIGN-TRAIL-INCL value     h"00".
            78 KEY2PARTTYP-SIGN-TRAIL-SEP       value   h"01".
            78  KEY2PARTTYP-SIGN-LEAD-INCL      value   h"02".
            78 KEY2PARTTYP-SIGN-LEAD-SEP        value   h"03".
            78 KEY2PARTTYP-SIGN-LEAD-FLOAT value     h"04".
        49   component-offset                pic 9(9)   comp-x.
        49   component-length                pic 9(9)   comp-x.

01    OP-N                        PIC   X COMP-X.
01    OP-X REDEFINES OP-N         PIC X.

PROCEDURE DIVISION.
    PERFORM 1000-OPEN-FILE.
    INITIALIZE  RECORD-AREA.
    MOVE 'RECORD-003KKKKKKKKKKKKKKK' TO PRIME-KEY.
    PERFORM 2000-READ-RECORD.
    DISPLAY 'AFTER RANDOM READ ------> ' RECORD-AREA '<' DISPLAY SPACE PERFORM
    3000-REWRITE-RECORD.
    PERFORM 2000-READ-RECORD.
    DISPLAY 'READ AFTER REWRITE -----> ' RECORD-AREA '<' DISPLAY SPACE
    PERFORM 5000-CLOSE-FILE.

    PERFORM 1000-OPEN-FILE.
    PERFORM 6000-START.
    PERFORM 7000-READ-PREVIOUS.
    DISPLAY 'AFTER 1ST READ PREVIOUS -> ' RECORD-AREA2 '<' DISPLAY SPACE PERFORM
    7000-READ-PREVIOUS.
    DISPLAY 'AFTER 2ND READ PREVIOUS -> ' RECORD-AREA2 '<' DISPLAY SPACE
    PERFORM 5000-CLOSE-FILE.
    STOP RUN.
```

```
1000-Open-File.
    MOVE 'FILE01EXTFH'          TO FILE-NAME
    MOVE STANDARD-TYPE          TO wOperationType
    MOVE OPEN-I-O               TO wOperationCode
    MOVE LOW-VALUES             TO FCD-AREA
    MOVE LENGTH OF FCD-AREA     TO FCD-LENGTH
    MOVE LENGTH OF FILE-NAME    TO FCD-NAME-LENGTH
    MOVE fcd--version-number    TO FCD-VERSION
    SET  FCD-FILENAME-ADDRESS   TO ADDRESS OF FILE-NAME
    SET  FCD-RECORD-ADDRESS     TO ADDRESS OF RECORD-AREA
    SET  FCD-KEY-DEF-ADDRESS    TO ADDRESS OF KEY-DEF-AREA
    MOVE fcd--exclusive-bit     TO FCD-LOCK-MODE
    MOVE fcd--format-cobol2     TO FCD-FILE-FORMAT
    MOVE fcd--indexed-org       TO FCD-ORGANIZATION
    MOVE fcd--dynamic-access    TO FCD-ACCESS-MODE
    MOVE fcd--recmode-fixed     TO FCD-RECORDING-MODE
    MOVE LENGTH OF RECORD-AREA  TO FCD-MIN-REC-LENGTH
    MOVE LENGTH OF RECORD-AREA  TO FCD-MAX-REC-LENGTH
    MOVE LENGTH OF RECORD-AREA  TO FCD-CURRENT-REC-LEN
    move fcd--open-closed       TO FCD-OPEN-MODE
    MOVE 16                     TO FCD-FS-FLAGS
    MOVE 1                      TO NUMBER-OF-KEYS
    MOVE LENGTH OF PRIME-KEY    TO COMPONENT-LENGTH
    display "01--- OPEN I-O"
    PERFORM 9000-CALL-EXTFH.
2000-Read-Record.
    INITIALIZE  RECORD-AREA.
    MOVE 'RECORD-003KKKKKKKKKKKKKKK' TO PRIME-KEY.
    MOVE  READ-RANDOM-LOCK      TO wOperationCode.
    SET FCD-RECORD-ADDRESS      TO ADDRESS OF RECORD-AREA.
    display "02--- READ"
    PERFORM 9000-CALL-EXTFH.
3000-Rewrite-Record.
    MOVE REWRITE-RECORDTO       wOperationCode.
    MOVE '1234567890' TO DATA-AREA.
    display "03--- REWRITE"
    PERFORM 9000-CALL-EXTFH.
5000-Close-File.
    MOVE CLOSE-FILE             TO wOperationCode.
    display "04--- CLOSE"
    PERFORM 9000-CALL-EXTFH.
6000-Start.
    SET FCD-RECORD-ADDRESSTO    ADDRESS OF RECORD-AREA.
    MOVE 25TO                   FCD-KEY-LENGTH.
    MOVE  HIGH-VALUES           TO  PRIME-KEY.
    MOVE START-LESS             TO wOperationCode.
    display "05--- START"
    PERFORM 9000-CALL-EXTFH.
7000-Read-Previous.
    MOVE READ-PREVIOUS  TO wOperationCode.
    INITIALIZE RECORD-AREA2.
```

```
        SET FCD-RECORD-ADDRESS       TO  ADDRESS OF RECORD-AREA2.
        display "06--- READ PREVIOUS"
        PERFORM 9000-CALL-EXTFH.
    9000-CALL-EXTFH.
        move '99' to FCD-FILE-STATUS
        display 'BEFORE OpCode: ' FUNCTION HEX-OF(OperationCode)
        perform 9900-Display
        CALL 'EXTFH' USING OperationCode FCD-AREA
        display 'AFTER OpCode : ' FUNCTION HEX-OF(OperationCode)
        perform 9900-Display
        accept omitted
        IF FCD-FILE-STATUS NOT = '00'
            MOVE wOperationCode TO OP-X
            IF FCD-STATUS-KEY-1 = '9'
                DISPLAY 'FILE ERROR, STATUS: 9/' FCD-BINARY '
                            wOperationCode: ' OP-N
            ELSE
                DISPLAY 'FILE ERROR, STATUS:      ' FCD-FILE-STATUS '
                                                  wOperationCode: ' OP-N
            END-IF
            STOP RUN
        END-IF.
    9900-Display.
        display 'FILE-STATUS   : ' FCD-FILE-STATUS
        if wOperationCode NOT =
              READ-PREVIOUS display
              'RECORD-AREA : '    RECORD-AREA
        else   display 'RECORD-AREA2 : ' RECORD-AREA2
        end-if
        display 'KEY-DEF-AREA :' display FUNCTION HEX-OF(KEY-DEF-AREA)
        continue.
END PROGRAM EXTFH02B.
```

RELATIVE BYTE ADDRESSING

When you read or write a record, you obtain its relative byte address. Using this address, you can re-read, re-write or delete the record without using keys and indexes.

This is a fast method of accessing records, but has limitations :

- A normal relative byte address operation does not affect the current record pointer. There- fore, a READ (sequential) after a relative byte addressing operation returns the record after the last one accessed using normal access methods. It does not return the record after the one you accessed using relative byte addressing.

- It is possible, however, to update the file position indicator so it points to the record you accessed using relative byte addressing. A subsequent READ returns you to the record after the one you accessed using the relative byte address operation.

- To update the file position indicator, you should set Bit 5 of the configuration flags (offset 93) in the FCD before calling the Callable File Handler to perform the relative byte address operation.

- Relative byte operations update the index when a record is re-written or deleted.

- If the record has been deleted since the relative byte address for it was obtained, a read or write using its relative byte address usually fails.

- However, you should be aware that it is possible for the record to have been replaced by a different record.

Record locking is supported by relative byte addressing.

The relative byte address of a record is returned in offset 72 of the FCD (or offset 13, if bit 4 of offset 93 is set) on all operations that involve specific records.

To use this address, simply save the contents of this field after a READ operation. Note that if the operation is unsuccessful, the value in the relative byte address is undefined.

Once you have obtained the relative byte address of a record, you can perform a number of operations. These are outlined in the following sections.

You can read a specific record from a file in two ways using the relative address:

Put the relative address in the relative byte address field of the FCD and set bit 6 of the configuration flags in the FCD (offset 93).

If you want to update the file position indicator to this record, set bit 5 of the configuration flags (offset 93).

Perform a READ (random) WITH NO LOCK, READ (random) WITH LOCK, READ (random) WITH KEPT LOCK or READ (random) operation on the file.

The READ (direct) operations work the same as described above, but do not require certain bits in the FCD to be set.

The READ (direct) operations always return the record at the address given in the relative byte address field of the FCD and update the file position indicator.

Both of the above methods provide a way of switching the current key-of-reference to a different key. For example, if READ (sequential) operations are performed via the primary key, you can start reading via the first alternate key from the current record using the following steps:

a. Read the next record in the file (the address of this record is in the FCD relative byte address field).

b. Put the new key-of-reference in the key-of-reference field in the FCD (offset 52).

c. Perform a READ (direct) operation to return the record at this address. The key-of-reference is changed to the new one.

d. Read the next record in the file. This is the next record in the new key-of-reference index.

You can re-write a record to a specific address by specifying that address in the relative byte address field of the FCD and setting bit 6 of the configuration flags in the FCD (offset 93). If you want to update the file position indicator, set bit 5 of the configuration flags in the FCD (offset 93). You can delete a record at a specific address by specifying that address in the relative byte address field of the FCD and setting bit 6 of the configuration flags in the FCD (offset 93).

CREATING A NEW INDEX

That enables you to recreate an index file using only the information in the data file. To recreate an index file, your program must:

a. Open the file using the open-new-index operation code.

b. Read records from the file using the get-next-rec operation code.

c. Create an index from each separate record, using the add-index operation code.

GnuCOBOL Example:

```
78 open-new-index        value  x"0007".
78 get-next-rec          value x"0008".
```

```
78 add-key-value              value x"0009".
78 close-file                 value x"fa80".
        ...
   move open-new-index to opcode
   perform EXTFH-op
   move get-next-rec       to opcode
   perform EXTFH-op
   perform until fcd-status(1:1) not = "0"
       perform varying fcd-key-id from 0 by 1 until fcd-key-id = key-count or
                 fcd-status(1:1) not = "0"
             move add-key-value to opcode
             perform EXTFH-op
       end-perform
       move get-next-rec to opcode
       perform EXTFH-op
   end-perform
   move close-file to opcode
   perform EXTFH-op
        ...

EXTFH-op.
   call "EXTFH" using opcode,
   fcd if fcd-status of fcd (1:1)
                                  "1"
        move 1 to return-code
   end-if.
```

ACCESSING A COMPILER GENERATED FCD

When normal I/O operations (OPEN, READ ...) in a GnuCOBOL program, an FCD is created for each file.

To access this FCD, set up a FCD definition in the Linkage Section of your program. An example FCD COBOL definition is supplied in a file called xfhfcd3.cpy.

This definition (that takes up no physical memory) can then be mapped onto the FCD you want to read or alter using the following SET statement in your program:

SET ADDRESS OF fcd TO ADDRESS OF fh–fcd OF namefile

where the parameters are:

fcd The name of the FCD definition in the Linkage Section of your program;

fh–fcd OF namefile (note the double hyphen). The pointer special register automatically allocated to the file with FD-name: namefile.

Following this SET operation, the data items defined in the Linkage Section group item fcd become the fields of the FCD of the file referenced in the SET statement.

Similarly, you can access the Key Definition Block by:

SET ADDRESS OF kda TO ADDRESS OF fh–keydef OF namefile

where the parameters are:

kda the name of the key definition Block in the Linkage Section of your program;

fh–keydef of indexedfile the pointer special register automatically allocated to the file with FD-name indexfile to point to its Key Definition Block.

TURN ON KEY COMPRESSION IN A FILE

Following sample program turns on key compression in a file by setting bits in the file's Key Definition Block.

```
    select
        MASTERFILE
        assign to ...
        organization is indexed
        record key is M-RecKey
        alternate key is M-AltKey1 with duplicates
        alternate key is M-AltKey2
        alternate key is M-AltKey3 with duplicates. select
     INDEXEDFILE
        assign to ...
        organization is indexed
        record key is I-RecKey
        alternate key is I-AltKey1 with duplicates.
    ---
    ---
LINKAGE SECTION.
01 KeyDefinitionBlock.
     03 filler                pic x(6).
     03 KeyCount              pic 9(4) comp-x.
     03 filler                pic x(6).
     03 KeyDefinition         occurs 1 to 4 times depending on KeyCount.
        05 filler             pic x(5).
        05 KeyCompression pic 9(2) comp-x.
        05 filler             pic x(10).
   ---

PROCEDURE DIVISION.
*>
*> set appropriate key compressions:
*>            +--  trailing  space  compression
*>            |+-- leading character compression
*>            ||+-- duplicate key compression
*>            |||
*> 7  =  00000111  - all compressions
*> 6  =  00000110  - leading character & trailing
                    space
*> 2  =  00000010  - leading character compression
*> 1  =  00000001  - duplicate key compression

  set address of KeyDefinitionBlock to address of fh--keydef of MASTERFILE move
  4 to KeyCount
  move  6  to  KeyCompression(1)
  move  7  to  KeyCompression(2)
  move  6  to  KeyCompression(3)
  move  7  to  KeyCompression(4)
  open I-O MASTERFILE
  -  -  -
  set address of KeyDefinitionBlock to address of fh--keydef of INDEXEDFILE move 2 to
  KeyCount
```

```
move 4 to KeyCompression(1)
```

```
move 1 to KeyCompression(2)
open input INDEXEDFILE
```

See the file NEWS for other details as implementation may varies between compiler versions.

8.2.61 SYSTEM

SYSTEM Built-In Subroutine Syntax

```
CALL "SYSTEM" USING command
~~~~                ~~~~~
```

This subroutine submits command (an alphanumeric literal or data item) to a command shell for execution as if it were typed into a console/terminal window.

A shell will be opened subordinate to the GnuCOBOL program issuing the call to SYSTEM.

Output from the command (if any) will appear in the command window in which the GnuCOBOL program was executed.

On a Unix system, the shell environment will be established using the default shell program. This is also true when using a GnuCOBOL build created with and for OSX or the Cygwin Unix emulator.

With native Windows Windows/MinGW builds, the shell environment will be the Windows console window command processor (usually cmd.exe) appropriate for the version of Windows you're using.

To trap output from the executed command and process it within the GnuCOBOL program, use a redirection ('>') to send the command output to a temporary file which you read from within the program once control returns.

The exit status of the executed command will be available in the RETURN-CODE special-register.

8.2.62 X"91"

X"91" Built-In Subroutine Syntax

CALL X"91" USING return-code, function-code, binary-variable-arg

———————————————————————————————————————

The original Micro Focus version of this routine is capable of providing a wide variety of functions. GnuCOBOL supports just three of them but more on the way subject to version of compiler (see file NEWS):

- Turning runtime switches (SWITCH-1, . . . , SWITCH-8) on.
- Turning runtime switches (SWITCH-1, . . . , SWITCH-8) off.
- Retrieving the number of arguments passed to a subroutine.

The return-code argument must be a one-byte binary numeric data item (USAGE BINARY-CHAR is recommended). It will receive a value of 0 if the operation was successful, 1 otherwise.

The function-code argument must be either a numeric literal or a one-byte binary numeric data item (USAGE BINARY-CHAR is recommended).

The third argument — variable-arg — is defined differently depending upon the function-code value, as follows:

11 Sets and/or clears all eight of the COBOL switches (SWITCH-1 through SWITCH-8). See Section 5.1.3 [SPECIAL-NAMES], page 90, for an explanation of those switches. Also referred to as programmable 0 - 7.

The variable-arg argument should be an OCCURS 8 TIMES table of USAGE BINARY-CHAR.

Each occurrence that is set to a value of zero prior to the CALL X"91" will cause the corresponding switch to be cleared. Each occurrence set to 1 prior to the CALL X"91" will cause the corresponding switch to be set.

Values other than 0 or 1 will be ignored.

12 Reads all eight of the COBOL switches (SWITCH-1 through SWITCH-8) and debug switches as prorammable 0 - 7.

The variable-arg argument should be an OCCURS 8 TIMES table of USAGE BINARY-CHAR.

Each of the 1st eight occurrences of the array will be set to either 0 or 1 — 1 if the corresponding switch is set, 0 otherwise.

13 + 14 Allow access to runtime switches 1 - 26 as A-Z. In case of A.N.T: set related runtime setting. Option 13 read them and option 14 sets them.

15 Prepare for program lookup. Checks to see if a program exists. You pass the program name and its length in parameter. When routine exits, result is zero if found and non-zero if not. Parameter is a group item consisting of two data items: a PIC X COMP-X specifyinh length in bytes of data item containing the file name and PIC X data item of varaiable length containing the file name.

16 Retrieves the number of arguments passed to the program executing the CALL X"91", saving that number into the variable-arg argument. That should be a binary numeric data item (USAGE BINARY-CHAR is recommended).

35 Prepare for DOS EXEC call.

46-49 Prepare for file specific settings LS_NULLS/LS_TABS.

69 Prepare for directory search.

8.2.63 X"E4"

X"E4" Built-In Subroutine Syntax

CALL X"E4"
~~~~

Use X"E4" to clear the screen. There are no arguments and no returned value.

8.2.64 X"E5"

X"E5" Built-In Subroutine Syntax

CALL X"E5"
~~~~

The X"E5" routine will sound the PC "bell". There are no arguments and no returned value.

8.2.65 X"F4"

X"F4" Built-In Subroutine Syntax

```
CALL X"F4" USING byte, table
~~~~       ~~~~~
```

———

This routine packs the low-order (rightmost) bit from each of the eight 1-byte items in table into the corresponding bit positions of the single-byte data item byte.

The byte data item need be only a single byte in size. If it is longer, the excess will be unaffected by this subroutine.

The table data item must be at least 8 bytes long. If it is longer, the excess will be ignored by this subroutine.

Typically, table is defined similarly to the following:

```
01 Table-Arg.
   05 Each-Byte OCCURS 8 TIMES USAGE BINARY-CHAR.
```

8.2.66 X"F5"

X"F5" Built-In Subroutine Syntax

```
CALL X"F5" USING byte, table
~~~~       ~~~~~
```

———————————————————————————————————————

This routine unpacks each bit of the single-byte data item byte into the low-order (rightmost) bit of each of the corresponding eight 1-byte items in table. The other seven bit positions of each of the first eight entries in table will be set to zero.

The byte data item need be only a single byte in size. If it is longer, the excess will be unaffected by this subroutine.

The table data item must be at least 8 bytes long. If it is longer, the excess will be ignored by this subroutine.

Typically, table is defined similarly to the following:

```
01 Table-Arg.
   05 Each-Byte OCCURS 8 TIMES USAGE BINARY-CHAR.
```

———————————————————

End of Chapter 8 — Functions

9 Report Writer Usage

9.1 RWCS Lexicon

There are a number of terms that describe various aspects of the operation of the Report Writer Control System (RWCS). Understanding the meanings of these terms is vital to developing an understanding of the subject.

Control Break
: An event that is triggered when a control field on an RWCS-generated report changes value. It is these events that trigger the generation of control heading and control footing groups.

Control Field
: A field of data being presented within a detail group; as the various detail groups that comprise the report are presented, they are presumed to appear in sorted sequence of the control fields contained within them. As an example, a department-by-department sales report for a chain of stores would probably be sorted by store number and – within like store numbers – be further sorted by department number. The store number will undoubtedly serve as a control field for the report, allowing control heading groups to be presented before each sequence of detail groups for the same store and control footing groups to be presented after each such sequence.

Control Footing
: A report group that appears immediately after one or more detail groups of an RWCS-generated report. Such are produced automatically as a result of a control break. This type of group typically serves as a summary of the detail group(s) that precede it, as might be the case on a sales report for a chain of stores, where the detail groups documenting sales for each department (one department per detail group) from the same store might be followed by a control footing that provides a summation of the department-by-department sales for that store.

Control Heading
: A report group that appears immediately before one or more detail groups of an RWCS-generated report. Such are produced automatically as a result of a control break. This type of group typically serves as an introduction to the detail group(s) that follow, as might be the case on a sales report for a chain of stores, where the detail groups documenting sales for each department (one department per detail group) from the same store might be preceded by a control heading that states the full name and location of the store.

Detail Group
: A report group that contains the detailed data being presented for the report.

Page Footing
: A report group that appears at the bottom of every page of an RWCS-generated report. Information typically found within such a report group might be:

- The date the report was generated
- The current page number of the report

Page Heading
: A report group that appears at the top of every page of an RWCS-generated report. Information typically found within such a report group might be:

- A title for the report

- The date the report was generated
- The current page number of the report
- Column headings describing the fields within the detail group(s)

Report Footing
: A report group that occurs only once in an RWCS-generated report — as the very last presented report group of the report. These typically serve as a visual indication that the report is finished.

Report Group
: One or more consecutive lines on a report that serve a common informational purpose or function. For example, lines of text that are displayed at the top or bottom of every printed page of a report.

Report Heading
: A report group that occurs only once in an RWCS-generated report — as the very first presented report group of the report. These typically serve as an introduction to the report.

9.2 The Anatomy of a Report

Every report has the same basic structure, as shown here, even though not all reports will have all of the groups shown. In fact, it is a very unusual report indeed that actually has every one of these groups:

- REPORT HEADING
- PAGE HEADING [1]
- CONTROL HEADING(S) [2]
- DETAIL GROUP(S) [2]
- CONTROL FOOTING(S) [2]
- FINAL CONTROL FOOTING
- PAGE FOOTING [1]
- REPORT FOOTING

[1] Presented throughout the report, as needed
[2] Repeated, as needed

These groups will be presented (printed) across however many formatted pages are necessary to hold them. No single report group will be allowed to cross page boundaries.

The management of paging, enforcement of the groups cannot span pages rule and almost every aspect of report generation are handled entirely by the Report Writer Control System.

9.3 The Anatomy of a Report Page

Each page of a report is divided into as many as five (5) areas, as shown in the following diagram.

```
 _____________________________
|                             |
| Top-of-page Unusable Area   |—# Lines: LINES AT TOP (LINAGE)
|_____________________________|
|                             |—Line #: HEADING (RD)
| Heading Area                |
|_____________________________|—Line #: FIRST DETAIL (RD) - 1
```

```
|                                  |—Line #: FIRST DETAIL (RD)
|                                  |
| Body Area                        |—Line #: LAST CONTROL HEADING (RD)
|                                  |—Line #: LAST DETAIL (RD)
|__________________________________|—Line #: FOOTING (RD)
|                                  |—Line #: FOOTING (RD) + 1
| Footing Area                     |
|__________________________________|
|                                  |
| Bottom-of-page Unusable Area     |—# Lines: LINES AT BOTTOM (LINAGE)
|__________________________________|
```

When describing a report via the RD (see Section 6.6 [REPORT SECTION], page 135) clause, the total number of usable lines are specified as the PAGE LIMIT value; this value is the sum of the number of lines contained in the Heading, Body and Footing Areas.

The unusable areas of a page (if any) will appear above and below that usable area. You don't specify the unusable area in the RD, but rather using a LINAGE (see Section 6.2.1 [File/Sort-Description], page 123) clause in the FD of the file the report is "attached" to.

The various report groups will be presentable in the various areas of a page, as follows:

REPORT HEADING
: Heading Area — An exception to this is the situation where the report heading report group contains the NEXT GROUP NEXT PAGE (see Section 6.9.33 [NEXT GROUP], page 189) option; in those cases, the report heading will be presented on a page by itself (anywhere on that page) at the beginning of the report.

PAGE HEADING
: Heading Area

CONTROL HEADING
: Body Area, but no line of a control heading is allowed past the line number specified by LAST CONTROL HEADING

DETAIL
: Body Area, but no line of a detail report group is allowed past the line number specified by LAST DETAIL

CONTROL FOOTING
: Body Area, but no line of a control footing report group is allowed past the line number specified by FOOTING

PAGE FOOTING
: Footing Area

REPORT FOOTING
: Footing Area — An exception to this is the situation where the report footing report group contains the NEXT PAGE option in its LINE (see Section 6.9.30 [LINE], page 185) clause; in those cases, the report footing will be presented on a page by itself at the end of the report.

9.4 How RWCS Builds Report Pages

A report created via a WRITE statement (see Section 7.8.54 [WRITE], page 410) will contain carriage-control information. Most notably, ASCII form-feed characters (X'0C') will be written to the report file to support the statement's ADVANCING PAGE option. Whether the data for a

report line created via ADVANCING PAGE occurs before or after the form-feed character depends upon whether the programmer coded WRITE record-name BEFORE ADVANCING PAGE or WRITE record-name AFTER ADVANCING PAGE, respectively.

The GnuCOBOL implementation of RWCS does not issue any carriage-control information to the report files it produces — instead, it relies upon the information coded in the RD for the report (specifically the PAGE LIMITS and related options) and its internally-generated and managed LINE-COUNTER special register (see Section 7.7 [Special Registers], page 260) for the report to know when to issue any blank lines to the file to fill-out the end of a printed page.

Because this is the way the GnuCOBOL RWCS works, in order to design an RWCS-generated report you'll need to know answers to the following questions:

1. What printer(s) will the report be printed on?
2. What paper orientation will you use, — Landscape (long edge of the paper at the top and bottom of page), or Portrait (long edge of the paper at the left and right of page)?
3. What tool will be used to print the report (direct printing to the device, notepad.exe, MS-Word, . . .)?
4. What font and font size will be used for the report when it is printed? RWCS-generated reports will assume that a fixed-width font such as "Courier", "Lucida Console", "Consolas" and the like will be used to print, as variable-pitch fonts would make the proper alignment of columns of data on reports virtually impossible.
5. When unprintable area exists at all four margins of the paper? These are generally caused by the printer itself or by its software driver.
6. What is the maximum number of lines per page that may be printed on a single sheet of paper?
7. What is the maximum number of characters that may be printed on one line?

Once you know the answer to questions 1-4, you may easily determine the answers to the remaining questions as follows:

1. Prepare a text file containing 100 or so records, each consisting of a numeric scale (123456789012345678901234. . .).
2. Print the file in a manner consistent with your answers to questions 1-4.
3. Add any necessary additional digits to each record in your test file (if lines weren't full) or remove characters from the end of each record if lines wrapped. If you made changes, reprint the file.
4. Now that you know exactly how long each record may be, add additional records and reprint. Continue until printing overflows to a second page.
5. The first page you print is now a perfect template to use when designing reports — it shows, given the answers to questions 1-4, every available printable character position on a page! The number of lines printed on that page becomes your PAGE LIMIT value for the RD.

The remaining PAGE LIMIT values can be established as required by your report(s).

Using identifier rather than integer specifications in the RD will give your program the ability — at run time — to accommodate multiple printers, fonts, font sizes and paper orientation. Just follow the above steps for each combination you wish your program to support.

9.5 Control Hierarchy

Every report that employs control breaks has a natural hierarchy of those control breaks based upon the manner in which the data the report is being generated from is sorted. This con-

cept is best understood using an example which assumes a COBOL program to process sales data collected from every computerized cash register across a chain of stores having multiple departments is being developed.

The application that collects data from the various cash registers at each store will generate data records that look like this to a COBOL program:

```
01 Sales-For-Register.
    05 Sales-Date            PIC 9(8).
    05 Time-Collected        PIC 9(6).
    05 Register-Number       PIC 9(7).
    05 Store-Number          PIC 9(3).
    05 Department-Number     PIC 9(3).
    05 Total-Sales           PIC 9(6)V99.
```

Your task is to develop a report that shows the sales total from each cash register and summarizes those sales by department within each store, by store and also generates a total sales figure for the day across all stores.

To accomplish this, you will use a SORT statement (see Section 7.8.44 [SORT], page 388) to sort the file of cash register sales data into:

1. Ascending sequence of store number
2. Within each store, data will be sorted into ascending sequence of department number
3. If there are multiple cash registers in a particular department of a specific store, the data needs to be further sorted so that the cash registers are ordered in sequence of their register number.

So, assuming a sort file has been defined and its record layout (essentially a mirror of the raw data file) is defined as follows:

```
01 Sorted-Sales-For-Register.
    05 Sorted-Sales-Date            PIC 9(8).
    05 Sorted-Time-Collected        PIC 9(6).
    05 Sorted-Register-Number       PIC 9(7).
    05 Sorted-Store-Number          PIC 9(3).
    05 Sorted-Department-Number     PIC 9(3).
    05 Sorted-Total-Sales           PIC 9(6)V99.
```

Then the SORT statement to accomplish the desired sequencing would be:

```
SORT SORT-FILE
    ASCENDING KEY Sorted-Store-Number
                  Sorted-Department-Number
                  Sorted-Register-Number
    USING Input-file
    OUTPUT PROCEDURE 100-Generate-Report
```

As a result of the sort, our program might expect to see data somewhat like this (date, time and sales totals are shown as ". . ."):

```
    +-------------------- Register Number
    |        +------------- Store Number
    |        |   +---------- Department Number
    |        |   |
...0535240001001...
```

```
...0589130001001...
...0625174001001...
...0122234001002...
...0732345001002...
...0003423001003...
...2038774001004...
...0112646002001...
...9963348002002...
...3245677002003...
...4456778002003...
...0002345002004...
```

Because of the sort, the most-frequently changing value of the three sort keys will be that of Sorted-Register-Number. This essentially defines the "detail" level of the report.

The next most-frequently changing value is that of Sorted-Department-Number, and the least-frequently changing value is that of Sorted-Store-Number. remember that the program should be generating totals each time one of these two values change, plus a grand total of sales at the end of the report. These three points are the Control Break points of the report.

When the report is defined, it's RD would contain a CONTROLS ARE clause that lists the control breaks in least- to most-frequent sequence of changing. This would be coded as:

```
CONTROLS ARE FINAL, Sorted-Store-Number, Sorted-Department-Number
```

A FINAL control break only occurs once, at the very end of the report. The CONTROL FOOTING for this break will be the one that produces the grand total of sales for all stores.

The next break listed on the CONTROLS clause will be the one that occurs next most-frequently (Sorted-Store-Number). This control break will be the one that produces the summation for each entire store, and will have its own CONTROL FOOTING.

The next (and last, in this case) break listed on the CONTROLS clause will be the one that occurs even more frequently (Sorted-Department-Number). The CONTROL FOOTING for this control field will be the one that summarizes sales for each department within a store.

This sequence of control breaks from least- to most-frequent (in other words, in the order they occur on the CONTROLS ARE clause) is the 'control hierarchy' of the report; control breaks that occur more frequently than others are said to be at a lower level in the control hierarchy.

Defining a control hierarchy (via CONTROLS ARE) that does not match the actual sequence in which data will be processed is a great way to guarantee a "broken" report. I'll show you an example in a later section.

9.6 An Example

This section contains an example of the RWCS at work. The complete program, presented here, is a stripped-down version of a program I have used to generate a report for a class I teach on PC hardware. This report will provide benchmark statistics on a variety of popular AMD and Intel CPUs. The data for the report was obtained from the website www.cpubenchmark.net in December of 2013. By the time you are reading this, that data will most likely have become rather out of date, but it illustrates RWCS well enough.

9.6.1 Data

Here is the data that the program will be reading. Each record reflects the aggregated benchmark scoring for one particular CPU, as scores for benchmarks against that CPU have been reported to the cpubenchmark.net website by their PassMark benchmark software. The data consists of

four fields. Fields are separated from one another by a single comma. The descriptions of the fields are as follows:

Benchmark Score
A five-digit number showing the aggregated benchmark scores for the CPU; the higher this number, the better the CPU performed in benchmark testing.

Vendor The name of the vendor who makes the CPU. In this data, that will either be "AMD" (American Micro Devices) or "INTEL".

Family The 7-character family of CPU products the CPU falls into. This will have values such as "A4", "A10", "Core i5", "Core i7", etc.

Model The specific model of CPU within the family.

The first record of data shown below shows that the aggregated score of all benchmarks reported for the AMD A10-4600M CPU is 3145, as compared to the second record which shows that the aggregated score reported of all benchmarks reported for the Intel Core-i7-4960X CPU is 14291.

The following is the complete set of input data used for this example. This is by no means the complete set of data available at cpubenchmark.net — it is just a representative sample used for this example. For my class, I give my students a report showing the results for almost a thousand CPUs.

For the sake of brevity, this document lists the data in three columns.

```
03145,AMD,A10,4600M
14291,Intel,Core i7,4960X
02505,AMD,A10,4655M
03449,AMD,A10,4657M
04251,AMD,A10,5700
02758,AMD,A10,5745M
03332,AMD,A10,5750M
03253,AMD,A10,5757M
04798,AMD,A10,5800B
04677,AMD,A10,5800K
04767,AMD,A10,6700
05062,AMD,A10,6800K
00677,AMD,A4,1200
00559,AMD,A4,1250
01583,AMD,A4,3300
01237,AMD,A4,3300M
01227,AMD,A4,3305M
01263,AMD,A4,3310MX
01193,AMD,A4,3320M
01343,AMD,A4,3330MX
01625,AMD,A4,3400
01768,AMD,A4,3420
01685,AMD,A4,4300M
01169,AMD,A4,4355M
01919,AMD,A4,5000
01973,AMD,A4,5150M
02078,AMD,A4,5300
01632,AMD,A4,5300B
02305,AMD,A4,6300
01634,AMD,A6,1450
01964,AMD,A6,3400M
02101,AMD,A6,3410MX
02078,AMD,A6,3420M
02277,AMD,A6,3430MX
01995,AMD,A6,3500
02798,AMD,A6,3600
02892,AMD,A6,3620
05421,AMD,FX,6100
05813,AMD,FX,6120
06194,AMD,FX,6200
06388,AMD,FX,6300
07017,AMD,FX,6350
06163,AMD,FX,8100
06605,AMD,FX,8120
06845,AMD,FX,8140
07719,AMD,FX,8150
08131,AMD,FX,8320
09067,AMD,FX,8350
09807,AMD,FX,9370
10479,AMD,FX,9590
03076,Intel,Core i3,3110M
03301,Intel,Core i3,3120M
03655,Intel,Core i3,3130M
03820,Intel,Core i3,3210
02266,Intel,Core i3,3217U
04219,Intel,Core i3,3220
03724,Intel,Core i3,3220T
04407,Intel,Core i3,3225
02575,Intel,Core i3,3227U
01885,Intel,Core i3,3229Y
04259,Intel,Core i3,3240
03793,Intel,Core i3,3240T
04414,Intel,Core i3,3245
04757,Intel,Core i3,3250
03443,Intel,Core i3,4000M
02459,Intel,Core i3,4010U
02003,Intel,Core i3,4010Y
04904,Intel,Core i3,4130
04041,Intel,Core i3,4130T
05115,Intel,Core i3,4330
05117,Intel,Core i3,4340
03807,Intel,Core i5,3210M
03995,Intel,Core i5,3230M
03126,Intel,Core i5,3317U
03917,Intel,Core i5,4300U
01743,Intel,Core i5,4300Y
04804,Intel,Core i5,4330M
03604,Intel,Core i5,4350U
06282,Intel,Core i5,4430
05954,Intel,Core i5,4430S
06517,Intel,Core i5,4440
07061,Intel,Core i5,4570
06474,Intel,Core i5,4570R
06803,Intel,Core i5,4570S
02503,Intel,Core i5,4570T
07492,Intel,Core i5,4670
07565,Intel,Core i5,4670K
06351,Intel,Core i5,4670T
03701,Intel,Core i7,3517U
03449,Intel,Core i7,3517UE
04588,Intel,Core i7,3520M
03912,Intel,Core i7,3537U
04861,Intel,Core i7,3540M
04009,Intel,Core i7,3555LE
06144,Intel,Core i7,3610QE
07532,Intel,Core i7,3610QM
06988,Intel,Core i7,3612QE
06907,Intel,Core i7,3612QM
05495,Intel,Core i7,3615QE
07310,Intel,Core i7,3615QM
07759,Intel,Core i7,3630QM
07055,Intel,Core i7,3632QM
06516,Intel,Core i7,3635QM
04032,Intel,Core i7,3667U
04271,Intel,Core i7,3687U
03479,Intel,Core i7,3689Y
08347,Intel,Core i7,3720QM
08512,Intel,Core i7,3740QM
09420,Intel,Core i7,3770
09578,Intel,Core i7,3770K
09074,Intel,Core i7,3770S
```

```
03232,AMD,A6,3650
03327,AMD,A6,3670
01630,AMD,A6,4400M
01296,AMD,A6,4455M
02440,AMD,A6,5200
01958,AMD,A6,5350M
01878,AMD,A6,5357M
01906,AMD,A6,5400B
02174,AMD,A6,5400K
02384,AMD,A6,6400K
02050,AMD,A8,3500M
02426,AMD,A8,3510MX
02245,AMD,A8,3520M
02276,AMD,A8,3530MX
02866,AMD,A8,3550MX
03215,AMD,A8,3800
03217,AMD,A8,3820
03552,AMD,A8,3850
03682,AMD,A8,3870K
02709,AMD,A8,4500M
02193,AMD,A8,4555M
04052,AMD,A8,5500
03464,AMD,A8,5500B
02434,AMD,A8,5545M
03052,AMD,A8,5550M
02935,AMD,A8,5557M
04348,AMD,A8,5600K
04390,AMD,A8,6500
04719,AMD,A8,6600K
04055,AMD,FX,4100
04153,AMD,FX,4130
04094,AMD,FX,4150
04774,AMD,FX,4170
04711,AMD,FX,4300
05247,AMD,FX,4350
04101,Intel,Core i5,3320M
05902,Intel,Core i5,3330
05690,Intel,Core i5,3330S
05781,Intel,Core i5,3335S
03280,Intel,Core i5,3337U
02252,Intel,Core i5,3339Y
06282,Intel,Core i5,3340
04327,Intel,Core i5,3340M
05372,Intel,Core i5,3340S
06199,Intel,Core i5,3350P
04314,Intel,Core i5,3360M
04555,Intel,Core i5,3380M
03589,Intel,Core i5,3427U
03479,Intel,Core i5,3437U
03057,Intel,Core i5,3439Y
06442,Intel,Core i5,3450
06071,Intel,Core i5,3450S
06576,Intel,Core i5,3470
06077,Intel,Core i5,3470S
04591,Intel,Core i5,3470T
05991,Intel,Core i5,3475S
06828,Intel,Core i5,3550
06631,Intel,Core i5,3550S
06993,Intel,Core i5,3570
07118,Intel,Core i5,3570K
06709,Intel,Core i5,3570S
05414,Intel,Core i5,3570T
04333,Intel,Core i5,4200M
03355,Intel,Core i5,4200U
02358,Intel,Core i5,4200Y
02382,Intel,Core i5,4210Y
03482,Intel,Core i5,4250U
04381,Intel,Core i5,4258U
04663,Intel,Core i5,4288U
04786,Intel,Core i5,4300M
08280,Intel,Core i7,3770T
08995,Intel,Core i7,3820
08548,Intel,Core i7,3820QM
09025,Intel,Core i7,3840QM
09196,Intel,Core i7,3920XM
12107,Intel,Core i7,3930K
09052,Intel,Core i7,3940XM
12718,Intel,Core i7,3960X
12823,Intel,Core i7,3970X
03992,Intel,Core i7,4500U
04507,Intel,Core i7,4558U
04892,Intel,Core i7,4600M
04484,Intel,Core i7,4600U
03680,Intel,Core i7,4610Y
04345,Intel,Core i7,4650U
07352,Intel,Core i7,4700EQ
08161,Intel,Core i7,4700HQ
07946,Intel,Core i7,4700MQ
08002,Intel,Core i7,4702HQ
07647,Intel,Core i7,4702MQ
08066,Intel,Core i7,4750HQ
07367,Intel,Core i7,4765T
09969,Intel,Core i7,4770
10190,Intel,Core i7,4770K
09803,Intel,Core i7,4770S
08803,Intel,Core i7,4770T
10078,Intel,Core i7,4771
08567,Intel,Core i7,4800MQ
09969,Intel,Core i7,4820K
09331,Intel,Core i7,4850HQ
09323,Intel,Core i7,4900MQ
13620,Intel,Core i7,4930K
09754,Intel,Core i7,4930MX
10262,Intel,Core i7,4960HQ
```

9.6.2 Program

Here is the program that will be producing the report. Pay attention to how the data is sorted and how the control hierarchy (CONTROLS ARE) relates to the SORT.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMORWCS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY. FUNCTION ALL INTRINSIC.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CPU-FILE          ASSIGN TO "CPUDATA.txt"
                             LINE SEQUENTIAL.
    SELECT  REPORT-FILE      ASSIGN TO "CPUREPORT.txt"
                             LINE SEQUENTIAL.
    SELECT SORT-FILE         ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.
FD  CPU-FILE.
01  CPU-REC                 PIC X(26).

FD  REPORT-FILE
```

```
        REPORT IS CPU-Report.

SD      SORT-FILE.
01      SORT-REC.
        05 F-SR-Score-NUM              PIC 9(5).
        05 F-SR-Vendor-TXT             PIC X(5).
        05 F-SR-Family-TXT             PIC X(7).
        05 F-SR-Model-TXT              PIC X(6).
WORKING-STORAGE SECTION.
01      WS-Date                        PIC 9(8).

01      WS-Family-Counters.
        05 WS-FC-AVE                   PIC 9(5)V99.
        05  WS-FC-Qty                  BINARY-LONG.
        05  WS-FC-Total-NUM            BINARY-LONG.

01      WS-Flags.
        05 WS-F-EOF                    PIC  X(1).

01      WS-One-Const                   PIC  9 VALUE 1.

01      WS-Overall-Counters.
        05 WS-OC-AVE                   PIC  9(5)V99.
        05 WS-OC-Qty                   BINARY-LONG.
        05 WS-OC-Total-NUM             BINARY-LONG.

01      WS-Starz                       PIC X(44) VALUE ALL '*'.

01      WS-Vendor-Counters.
        05 WS-VC-AVE                   PIC 9(5)V99.
        05 WS-VC-Qty                   BINARY-LONG.
        05 WS-VC-Total-NUM             BINARY-LONG.

REPORT SECTION.
RD      CPU-Report CONTROLS
        ARE FINAL
                        F-SR-Vendor-TXT
                        F-SR-Family-TXT

        PAGE LIMIT IS          36 LINES
            HEADING            1
            FIRST DETAIL 5
            LAST DETAIL        36.

01      TYPE IS PAGE HEADING.
        05 LINE NUMBER PLUS 1.
            10 COL 1      SOURCE WS-Date                PIC  9999/99/99.
            10 COL 14 VALUE 'CPU Benchmark Scores'.
            10 COL 37 VALUE 'Page:'.
            10 COL 43 SOURCE PAGE-COUNTER                PIC  Z9.
        05 LINE NUMBER PLUS 1.
```

```
    10 COL 1     SOURCE  WS-Starz                    PIC X(44).
05 LINE NUMBER PLUS 1.
    10 COL 1     VALUE '**'.
    10 COL 6     VALUE 'All CPU Data From cpubenchmark.net'.
    10 COL 43 VALUE '**'.
05 LINE NUMBER PLUS 1.
    10 COL 1     SOURCE  WS-Starz                    PIC X(44).

01  TYPE CONTROL HEADING F-SR-Family-TXT.
    05 LINE NUMBER PLUS 1.
    10 COL 1     SOURCE F-SR-Vendor-TXT              PIC X(6).
    10 COL 8     SOURCE F-SR-Family-TXT              PIC X(7).
    05 LINE NUMBER PLUS 1.
    10 COL 1     VALUE 'Family'.
    10 COL 9     VALUE 'Model'.
    10 COL 16 VALUE 'Benchmark Score (High to Low)'. 05
    LINE NUMBER PLUS 1.
    10  COL  1    VALUE  '======'.
    10  COL  9    VALUE  '======'.
    10  COL  16  VALUE  '============================'.

01  Detail-Line TYPE IS DETAIL.
    05 LINE NUMBER PLUS 1.
    10 COL 1     SOURCE F-SR-Family-TXT PIC X(7) GROUP INDICATE.
    10 COL 9     PIC X(6)            SOURCE F-SR-Model-TXT.
    10 COL 16 PIC ZZZZ9             SOURCE F-SR-Score-NUM.

01  End-Family TYPE IS CONTROL FOOTING F-SR-Family-TXT.
    05 LINE NUMBER PLUS 1.
    10 COL 9                        VALUE  'Ave...'.
    10 COL 16 PIC ZZZZ9.99          SOURCE  WS-FC-AVE.
    10 COL 25                       VALUE  '('.
    10 COL 26 PIC ZZ9               SUM     WS-One-Const.
    10 COL 30                       VALUE  'Family  CPUs)'.

01  End-Vendor TYPE IS CONTROL FOOTING F-SR-Vendor-TXT.
    05 LINE NUMBER PLUS 1.
    10 COL 9                        VALUE  'Ave...'.
    10 COL 16 PIC ZZZZ9.99          SOURCE  WS-VC-AVE.
    10 COL 25                       VALUE  '('.
    10 COL 26 PIC ZZ9               SUM     WS-One-Const.
    10 COL 30                       VALUE  'Vendor CPUs)'.

01  End-Overall TYPE IS CONTROL FOOTING FINAL.
    05 LINE NUMBER PLUS 1.
    10 COL 9                        VALUE  'Ave...'.
    10 COL 16 PIC ZZZZ9.99          SOURCE  WS-OC-AVE.
    10 COL 25                       VALUE  '('.
    10 COL 26 PIC ZZ9               SUM     WS-One-Const.
    10 COL 30                       VALUE  'CPUs)'.
```

```
PROCEDURE DIVISION.
DECLARATIVES.
000-End-Family SECTION.
      USE BEFORE REPORTING End-Family.
1.    IF WS-FC-Qty > 0
            COMPUTE WS-FC-AVE = WS-FC-Total-NUM / WS-FC-Qty
      ELSE
            MOVE 0 TO WS-FC-AVE
      END-IF
      MOVE 0 TO WS-FC-Qty
                  WS-FC-Total-NUM
      -
000-End-Vendor SECTION.
      USE BEFORE REPORTING End-Vendor.
1.    IF WS-VC-Qty > 0
            COMPUTE WS-VC-AVE = WS-VC-Total-NUM / WS-VC-Qty ELSE
            MOVE 0 TO WS-VC-AVE
      END-IF
      MOVE 0 TO WS-VC-Qty
                  WS-VC-Total-NUM
      -
000-End-Overall  SECTION.
      USE BEFORE REPORTING End-Overall.
1.    IF WS-OC-Qty > 0
            COMPUTE WS-OC-AVE = WS-OC-Total-NUM / WS-OC-Qty
      ELSE
            MOVE 0 TO WS-OC-AVE
      END-IF
      MOVE 0 TO WS-OC-Qty
                  WS-OC-Total-NUM
      -
END DECLARATIVES.

010-Main SECTION.
1.    ACCEPT  WS-Date  FROM  DATE YYYYMMDD
      SORT SORT-FILE
            ASCENDING KEY        F-SR-Vendor-TXT
                                 F-SR-Family-TXT
            DESCENDING KEY       F-SR-Score-NUM
            ASCENDING KEY        F-SR-Model-TXT
            INPUT PROCEDURE 100-Pre-Process-Data
            OUTPUT PROCEDURE 200-Generate-Report
      STOP RUN
      -
100-Pre-Process-Data  SECTION.
1.    OPEN  INPUT CPU-FILE
      PERFORM FOREVER
            READ CPU-FILE
            AT END
                  EXIT  PERFORM
```

```
            END-READ
            MOVE SPACES TO SORT-REC
            UNSTRING CPU-REC DELIMITED BY ',’
                INTO F-SR-Score-NUM,
                        F-SR-Vendor-TXT,
                        F-SR-Family-TXT,
                        F-SR-Model-TXT
            RELEASE   SORT-REC
        END-PERFORM
        CLOSE CPU-FILE
        -
    200-Generate-Report SECTION.
    1.  INITIALIZE WS-Family-Counters
                        WS-Flags
        OPEN OUTPUT REPORT-FILE
        INITIATE CPU-Report
        RETURN SORT-FILE
        AT END
            MOVE ’Y’ TO WS-F-EOF
        END-RETURN
          PERFORM UNTIL WS-F-EOF = ’Y’
             GENERATE Detail-Line
            ADD 1TO WS-FC-          Qty      WS-OC-
                                        Qty      WS-
                                        VC-Qty
            ADD F-SR-Score-NUM TO WS-FC-Total-NUM
                                        WS-OC-Total-NUM
                                        WS-VC-Total-NUM
            RETURN SORT-FILE
            AT END
                MOVE ’Y’ TO WS-F-EOF
            END-RETURN
        END-PERFORM
        TERMINATE CPU-Report
        CLOSE REPORT-FILE
        -
```

9.6.3 Generated Report Pages

Finally, here's the report the program generates!

```
2013/12/24       CPU  Benchmark Scores        Page:     1
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
AMD       A10
Family    Model    Benchmark Score (High to Low)
======    ======   ============================
A10       6800K      5062
          5800B      4798
          6700       4767
          5800K      4677
```

```
            5700      4251
            4657M     3449
            5750M     3332
            5757M     3253
            4600M     3145
            5745M     2758
            4655M     2505
            Ave...    3817.90 ( 11 Family CPUs)
AMD       A4
Family    Model     Benchmark Score (High to Low)
======    ======    ============================
A4          6300      2305
            5300      2078
            5150M     1973
            5000      1919
            3420      1768
            4300M     1685
            5300B     1632
            3400      1625
            3300      1583
           3330MX     1343
           3310MX     1263
            3300M     1237
            3305M     1227
            3320M     1193
```

```
2013/12/24        CPU  Benchmark Scores        Page:     2
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
A4          4355M     1169
            1200       677
            1250       559
            Ave...    1484.47 ( 17 Family CPUs)
AMD       A6
Family    Model     Benchmark Score (High to Low)
======    ======    ============================
A6          3670      3327
            3650      3232
            3620      2892
            3600      2798
            5200      2440
            6400K     2384
           3430MX     2277
            5400K     2174
           3410MX     2101
            3420M     2078
            3500      1995
            3400M     1964
            5350M     1958
```

```
            5400B     1906
            5357M     1878
            1450      1634
            4400M     1630
            4455M     1296
            Ave...    2220.22 ( 18 Family CPUs)
AMD       A8
Family    Model     Benchmark Score (High to Low)
======    ======    ============================
A8        6600K     4719
          6500      4390
          5600K     4348
```

```
2013/12/24       CPU  Benchmark Scores        Page:     3
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
A8        5500      4052
          3870K     3682
          3850      3552
          5500B     3464
          3820      3217
          3800      3215
          5550M     3052
          5557M     2935
          3550MX    2866
          4500M     2709
          5545M     2434
          3510MX    2426
          3530MX    2276
          3520M     2245
          4555M     2193
          3500M     2050
          Ave...    3148.68 ( 19 Family CPUs)
AMD       FX
Family    Model     Benchmark Score (High to Low)
======    ======    ============================
FX        9590      10479
          9370       9807
          8350       9067
          8320       8131
          8150       7719
          6350       7017
          8140       6845
          8120       6605
          6300       6388
          6200       6194
          8100       6163
          6120       5813
```

```
2013/12/24      CPU  Benchmark Scores        Page:     4
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
FX          6100       5421
            4350       5247
            4170       4774
            4300       4711
            4130       4153
            4150       4094
            4100       4055
            Ave...     6457.00 ( 19 Family CPUs)
            Ave...     3448.86 ( 84 Vendor CPUs)
Intel     Core i3
Family     Model   Benchmark Score (High to Low)
======    ======  ============================
Core i3 4340       5117
            4330       5115
            4130       4904
            3250       4757
            3245       4414
            3225       4407
            3240       4259
            3220       4219
            4130T      4041
            3210       3820
            3240T      3793
            3220T      3724
           3130M      3655
           4000M      3443
           3120M      3301
           3110M      3076
            3227U      2575
            4010U      2459
            3217U      2266
            4010Y      2003
```

```
2013/12/24      CPU  Benchmark Scores        Page:     5
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
Core i3 3229Y      1885
            Ave...     3677.76 ( 21 Family CPUs)
Intel     Core i5
Family     Model   Benchmark Score (High to Low)
======    ====== ============================
Core i5 4670K      7565
            4670       7492
            3570K      7118
            4570       7061
```

```
          3570      6993
          3550      6828
          4570S     6803
          3570S     6709
          3550S     6631
          3470      6576
          4440      6517
          4570R     6474
          3450      6442
          4670T     6351
          3340      6282
          4430      6282
          3350P     6199
          3470S     6077
          3450S     6071
          3475S     5991
          4430S     5954
          3330      5902
          3335S     5781
          3330S     5690
          3570T     5414
          3340S     5372
          4330M     4804
```

```
2013/12/24      CPU  Benchmark Scores       Page:    6
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
Core  i5  4300M     4786
          4288U     4663
          3470T     4591
          3380M     4555
          4258U     4381
          4200M     4333
          3340M     4327
          3360M     4314
          3320M     4101
          3230M     3995
          4300U     3917
          3210M     3807
          4350U     3604
          3427U     3589
          4250U     3482
          3437U     3479
          4200U     3355
          3337U     3280
          3317U     3126
          3439Y     3057
          4570T     2503
          4210Y     2382
```

```
            4200Y     2358
            3339Y     2252
            4300Y     1743
            Ave...    5026.13 ( 52 Family CPUs)
Intel     Core i7
Family     Model   Benchmark Score (High to Low)
======    ====== ============================
Core i7 4960X       14291
            4930K    13620
            3970X    12823
```

```
2013/12/24       CPU  Benchmark Scores        Page:     7
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
Core  i7  3960X    12718
            3930K    12107
            4960HQ  10262
            4770K    10190
            4771      10078
            4770       9969
            4820K      9969
            4770S      9803
            4930MX   9754
            3770K      9578
            3770       9420
            4850HQ   9331
            4900MQ   9323
            3920XM   9196
            3770S      9074
            3940XM   9052
            3840QM   9025
            3820       8995
            4770T      8803
            4800MQ   8567
            3820QM   8548
            3740QM   8512
            3720QM   8347
            3770T      8280
            4700HQ   8161
            4750HQ   8066
            4702HQ   8002
            4700MQ   7946
            3630QM   7759
            4702MQ   7647
            3610QM   7532
            4765T      7367
```

```
2013/12/24       CPU  Benchmark Scores        Page:     8
```

```
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
Core i7 4700EQ     7352
           3615QM  7310
           3632QM  7055
           3612QE  6988
           3612QM  6907
           3635QM  6516
           3610QE  6144
           3615QE  5495
           4600M   4892
           3540M   4861
           3520M   4588
           4558U   4507
           4600U   4484
           4650U   4345
           3687U   4271
           3667U   4032
           3555LE  4009
           4500U   3992
           3537U   3912
           3517U   3701
           4610Y   3680
           3689Y   3479
           3517UE  3449
           Ave...  7725.58 ( 58 Family CPUs)
           Ave...  6005.16 (131 Vendor CPUs)
           Ave...  5006.42 (215 CPUs)
```

9.7 Control Hierarchy (Revisited)

The sample program just discussed presents a great opportunity to show what can happen if you don't define the control hierarchy of a report properly.

I changed the CONTROLS ARE clause on the sample program from this:

```
CONTROLS ARE FINAL
                F-SR-Vendor-TXT
                F-SR-Family-TXT
```

To this:

```
CONTROLS ARE FINAL
                F-SR-Family-TXT
                F-SR-Vendor-TXT
```

And then ran the report again. Here are the first two pages of that new report. See what happened to the control breaks?

```
2013/12/24      CPU  Benchmark Scores        Page:     1
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
AMD       A10
Family    Model    Benchmark Score (High to Low)
======    ======   ============================
A10       6800K     5062
          5800B     4798
          6700      4767
          5800K     4677
          5700      4251
          4657M     3449
          5750M     3332
          5757M     3253
          4600M     3145
          5745M     2758
          4655M     2505
           Ave...   3817.90 ( 11 Vendor CPUs)
           Ave...   3817.90 ( 11 Family CPUs)
AMD       A4
Family    Model    Benchmark Score (High to Low)
======    ======   ============================
A4        6300      2305
          5300      2078
          5150M     1973
          5000      1919
          3420      1768
          4300M     1685
          5300B     1632
          3400      1625
          3300      1583
         3330MX     1343
         3310MX     1263
          3300M     1237
          3305M     1227
```

```
2013/12/24      CPU  Benchmark Scores        Page:     2
*****************************************
**     All CPU Data From cpubenchmark.net    **
*****************************************
A4        3320M     1193
          4355M     1169
          1200       677
          1250       559
           Ave...   1484.47 ( 17 Vendor CPUs)
```

```
          Ave...   1484.47 ( 17 Family CPUs)
AMD      A6
Family    Model   Benchmark Score (High to Low)
======    ======  ============================
A6        3670     3327
          3650     3232
          3620     2892
          3600     2798
          5200     2440
          6400K    2384
         3430MX    2277
          5400K    2174
         3410MX    2101
          3420M    2078
          3500     1995
          3400M    1964
          5350M    1958
          5400B    1906
          5357M    1878
          1450     1634
          4400M    1630
          4455M    1296
          Ave...   2220.22 ( 18 Vendor CPUs)
          Ave...   2220.22 ( 18 Family CPUs)
AMD      A8
Family    Model   Benchmark Score (High to Low)
======    ======  ============================
A8        6600K    4719
```

9.8 Turning PHYSICAL Page Formatting Into LOGICAL Formatting

You can trick RWCS into using the PAGE LIMIT values as logical specifications rather than physical ones quite easily — simply include an ASCII form-feed (X'0C') character into your page heading design! Here's how the sample program shown earlier could be easily modified:

Simply Change This. . .

```
01   TYPE IS PAGE HEADING.
      05 LINE NUMBER 1.
         10  COL  1    SOURCE WS-Date PIC 9999/99/99.
         10  COL  14   VALUE 'CPU Benchmark Scores'.
         10  COL  37   VALUE 'Page:'.
         10  COL  43   SOURCE PAGE-COUNTER PIC Z9.
     05  LINE NUMBER PLUS   1.
         10 COL 1       SOURCE  WS-Starz PIC   X(44).
     05  LINE NUMBER PLUS   1.
         10  COL  1    VALUE  '**'.
         10  COL  6    VALUE  'All CPU          From ' &
                               Data
                               'cpubenchmark.net'.
         10  COL  43   VALUE  '**'.
```

```
05 LINE NUMBER PLUS 1.
    10 COL 1     SOURCE WS-Starz PIC X(44).
```

To This. . .

```
01   TYPE IS PAGE HEADING.
   05 LINE NUMBER 1.                    *> NEW
     10 COL 1  VALUE X'0C'.             *> NEW
   05 LINE NUMBER PLUS 1.               *> CHANGED
       10 COL 1     SOURCE WS-Date PIC 9999/99/99.
       10  COL  14  VALUE 'CPU Benchmark Scores'.
       10  COL  37  VALUE 'Page:'.
       10  COL  43  SOURCE PAGE-COUNTER PIC Z9.
   05  LINE NUMBER PLUS   1.
       10 COL 1     SOURCE  WS-Starz PIC   X(44).
   05  LINE NUMBER PLUS   1.
       10  COL  1    VALUE  '**'.
       10  COL  6    VALUE  'All CPU        From ' &
                         Data
                         'cpubenchmark.net'.
       10  COL  43  VALUE  '**'.
   05 LINE NUMBER PLUS 1.
       10 COL 1     SOURCE WS-Starz PIC X(44).
```

RWCS will still be counting lines to decide when to close off one page and start a new one, but when a new page is started its page heading will physically form-feed the printer when the report is printed. As long as any printer you plan on using supports at least as many physical print lines as what is defined as the PAGE LIMIT value in whatever paper orientation and font you plan on (or are limited to) printing in, you have now divorced your program from the physical realities of the printer!

Of course, whatever software you are using to deliver the printed document to the printer with, must allow the ASCII form-feed character to pass through to the printer.

———————————————————

End of Chapter 9 — Report Writer Usage

10 Interfacing With The OS

10.1 Compiling Programs

Program source files should have by convention, extensions of .cob or .cbl.

Program file names should match exactly the specification of PROGRAM-ID (including case).

Spaces cannot be included in primary entry-point names and therefore must not be included in program file names.

The GnuCOBOL compiler will translate your COBOL program into C source code, compile that C source code into executable binary form using the C compiler specified when GnuCOBOL was built and link that executable binary into:

Directly executable form
: This is an executable file directly-executable from the command-line. On Windows computers, this would be an .exe file. On Unix systems, this will be a file with no specific extension, but with execute permissions. This file will include the main program as well as any static-linked subprograms.

Static-linkable form
: This is a single subprogram compiled into object-code form, ready to be linked in with a main program to form a directly-executable program. On windows computers, these generally are .o (object-code) files.

Dynamically-loadable executable form
: These are dynamically-loadable object code files ready to be invoked from other programs at execution time. On Windows systems, these would be .dll files, while on Unix systems they are typically .so files (OSX uses .dylib).

10.1.1 cobc - The GnuCOBOL Compiler

The GnuCOBOL compiler is named cobc (cobc.exe on a Windows system).

The following describes the syntax and option switches of the cobc command. This informa- tion may be displayed by entering the command cobc --help or cobc -h.

GnuCOBOL compiler for most COBOL dialects with lots of extensions Sorted

within groups

```
Usage: cobc [options]... file...

Options:
  -h, --help              display this help and exit

  -du(mpversion)          display compiler version and exit
  -i, --info              display compiler information (build/environment)
                          and exit
  -q, --brief             reduced displays, commands invoked not shown
  -v, --verbose           verbose mode, display additional information;
                          multiple -v options increase the verbosity,
                          the maximum is 3 as follows:
                          (1) display compiler version and the commands
                          invoked by the compiler,
```

```
                          (2) pass verbose option to assembler/compiler
                          (3) pass verbose option to linker
  -V, --version           display compiler version information and exit
  -###                    like -v but commands not executed

  -A <options>            add <options> to the C compile phase
  -b                      combine all input files into a single
                          dynamically loadable module
  -c                      compile and assemble, but do not link
  --conf=<file>           user-defined dialect configuration; see -std
  -C                      translation only; convert COBOL to C
  -d, --debug             enable all run-time error checking,
                          equal to -fstack-check -fec=EC-ALL
  -D <define>             define <define> for COBOL compilation
  -ext <extension>        add file extension for resolving COPY
  -E                      preprocess only; do not compile or link
  -F, --free              use free source format (alias for -fformat=free)
  --fixed                 use fixed source format (default; alias for
                          -fformat=fixed)
  -fec=<exception-name>   enable code generation for <exception-name>,
                          see --list-exceptions for the possible values,
                          sets -fsource-location
  -fno-ec=<exception-name>
                          disable code generation for <exception-name>
  -g                      enable C compiler debug and stack check
  -I <directory>          add <directory> to copy/include search path
  -j [<args>], --job[=<args>]
                          run program after build, passing <args>
  -K <entry>              generate CALL to <entry> as static
  -l <lib>                link the library <lib>
  -L <directory>          add <directory> to library search path
  --list-exceptions       display exception names
  --list-intrinsics       display intrinsic functions
  --list-mnemonics        display mnemonic names
  --list-reserved         display reserved words and internal registers
  --list-system           display system routines
  -m                      build a dynamically loadable module (default)
  -o <file>               place the output into <file>
  -O, -O2, -O3, -Os       enable optimization
  -O0                     disable optimization
  -P[=<dir or file>]      generate preprocessed program listing (.lst)
  -Q <options>            add <options> to the C link phase
  -std=<dialect>          warnings/features for a specific dialect
                          <dialect> can be one of:
                          default, cobol2014, cobol2002, cobol85, xopen,
                          ibm-strict, ibm, mvs-strict, mvs,
                          mf-strict, mf, bs2000-strict, bs2000,
                          acu-strict, acu, rm-strict, rm, gcos-strict,
                          gcos;
                          see configuration files in directory config
  -S                      compile only; output assembly file
```

```
  --save-temps[=<dir>]      save intermediate files
                            * default: current directory
  -t <file>                 generate and place a program listing into <file>
  --tlines=<lines>          specify lines per page in listing, default = 55
  -T <file>                 generate and place a wide program listing into <file>
  -x                        build an executable program
  -X,  --Xref               generate internal cross reference


Warning options:
  -w                        disable all warnings
  -Wall                     enable most warnings (all except as noted below)
  -Wadditional              additional warnings only raised with -Wall
  -Warchaic                 warn if archaic features are used
  -Warithmetic-osvs         warn if arithmetic expression precision has changed
  -Wcall-params             warn about non 01/77 items for CALL parameters
                            *  NOT set with -Wall
  -Wcolumn-overflow         warn about text after program-text area, FIXED format
                            *  NOT set with -Wall
  -Wconstant-expression     warn about expressions that always resolve to true/false
  -Wconstant-numlit-expression
                            warn about numeric expressions that always resolve to
                            true/false
  -Wdangling-text           warn about source text after program-area
                            *  NOT set with -Wall
  -Werror                   treat all warnings as errors
  -Werror=<warning>         treat specified <warning> as error
  -Wextra                   like -Wall but enable some extra warning flags
  -Wimplicit-define         warn whenever data items are implicitly defined
                            *  NOT set with -Wall
  -Winitial-value           warn if initial VALUE clause is ignored
  -Wlarger-01-redefines     warn about larger redefines allowed by COBOL standards
  -Wlinkage                 warn about dangling LINKAGE items
                            *  NOT set with -Wall
  -Wobsolete                warn if obsolete features are used
  -Woverlap                 warn about overlapping MOVE of items
  -Wno-corresponding        do not warn about CORRESPONDING with no matching items
                             *  ALWAYS active
  -Wno-dialect              do not warn about dialect specific issues
                             *  ALWAYS active
  -Wgoto-different-section
                            warn about GO TO a praragraph defined in a different section
  -Wno-error                don't treat warnings as errors
  -Wno-error=<warning>      don't treat specified <warning> as error
  -Wno-goto-section         do not warn about GO TO section-name
                             *  ALWAYS active
  -Wno-ignored-error        do not warn about errors in code parts which are
                            unreachable and so normally ignored
                             *  ALWAYS active
  -Wno-missing-newline      do not warn about missing newlines
                             *  ALWAYS active
```

```
  -Wno-others                 do not warn about different issues
                                * ALWAYS active
  -Wno-pending                do not warn if pending features are used
                                * ALWAYS active
  -Wno-repository-checks
                              do not warn/check for program/function/external signature mismatch
                              * ALWAYS active
  -Wno-unfinished             do not warn if unfinished features are used
                              * ALWAYS active
  -Wno-unsupported            do not warn if runtime does not support a feature used
  -Wno-<warning>              disable warning enabled by default, -Wall or -Wextra
  -Wparentheses               warn if parentheses are omitted around AND within OR
  -Wpossible-overlap          warn  about MOVE of items that may overlap depending on
                              variables
                              * NOT set with -Wall
  -Wpossible-truncate         warn about possible field truncation
                              * NOT set with -Wall
  -Wprototypes                warn about missing FUNCTION prototypes/definitions
  -Wredefinition              warn about non-referenced ambiguous data items
  -Wstrict-typing             warn strictly about type mismatch
  -Wsuspicious-perform-thru
                              warn if PERFORM THRU references procedures not in ascending
                              order or multiple sections
                              * ALWAYS active
    -Wterminator                warn about lack of scope terminator END-XXX
                                * NOT set with -Wall
  -Wtruncate                  warn about field truncation from constant assignments
  -Wunreachable               warn about likely unreachable statements
                              * NOT set with -Wall


Compiler options:
  -fsign=[ASCII|EBCDIC]       define display sign representation
                              * default: machine native
  -ffold-copy=[UPPER|LOWER]
                              fold COPY subject to value
                              * default: no transformation
  -ffold-call=[UPPER|LOWER]
                              fold PROGRAM-ID, CALL, CANCEL subject to value
                              * default: no transformation
  -fmax-errors=<number>       maximum number of errors to report before
                              compilation is aborted
                              * default:  128
  -fintrinsics=[ALL|intrinsic
                              function name(,name,...)]
                              intrinsics to be used without FUNCTION keyword
  -fdump=<scope>              dump data fields on abort, <scope> may be
                              a combination of: ALL, WS, LS, RD, FD, SC, LO
  -fcallfh=<name>             specifies <name> to be used for I/O
                              as external provided EXTFH interface module
```

```
-febcdic-table=[DEFAULT|RESTRICTED-GC|IBM|GCOS]
                        define EBCDIC translation table:
                        *  default: translation to extended ASCII as per MF
                        *  restricted-gc: translation from restricted ASCII only
                        *  ibm: translation to restricted ASCII as per IBM
                        *  gcos: translation to extended ASCII as per GCOS7
-fdefault-colseq=[ASCII|EBCDIC|NATIVE]
                        define default collating sequence
                        *  default:  NATIVE
-fstack-extended        store origin of entrypoints and PERFORM
                        *  turned on by -debug/-dump
-fno-remove-unreachable
                        disable remove of unreachable code
                        *  turned off by -g
-ftrace                 generate trace code
                        *  scope:  executed  SECTION/PARAGRAPH
-ftraceall              generate trace code
                        *  scope:  executed  SECTION/PARAGRAPH/STATEMENTS
-fsyntax-only           syntax error checking only; don't emit any output
-fdebugging-line        enable debugging lines
                        *  'D' in indicator column or floating >>D
-fsource-location       generate source location code
                        *  turned on by -debug/-ftraceall/-fec/-dump
-fimplicit-init         automatic initialization of the COBOL runtime system
-fno-recursive-check    disable check of recursive program call;
                        effectively compiling as RECURSIVE program
-fstack-check           PERFORM stack checking
                        *  turned on by -debug/-g
-fsection-exit-check    check that code execution does not leave the scope of
                        SECTIONs
-fimplicit-goback-check
                        check that code execution does not end implicit at end of PROCEDURE
                        DIVISION
-fwrite-after           use AFTER 1 for WRITE of LINE SEQUENTIAL
                        *  default: BEFORE 1
-fmfcomment             '*' in column 1 treated as comment with listing suppression
                        *  FIXED/COBOL85/VARIABLE format only
-facucomment            '$' in indicator area treated as '*',
                        '|' treated as floating comment
-fno-trunc              allow numeric field overflow
                        *  non-ANSI behaviour
-fsingle-quote          use a single quote (apostrophe) for QUOTE
                        *  default: double quote
-foptional-file         treat all files as OPTIONAL
                        *  unless NOT OPTIONAL specified
-fstatic-call           output static function calls for the CALL statement
-fno-gen-c-decl-static-call
                        disable generation of C function declarations
                        for subroutines with static CALL
-fgen-c-line-directives
                        generate source location directives in C code;
```

```
                              *  turned on by -g
  -fgen-c-labels              generate extra labels in C sources;
                              *  turned on by -g
    -fno-theaders               suppress all headers and output of compilation
                                 options from listing while keeping page breaks
  -fno-tsource                suppress source from listing
  -fno-tmessages              suppress warning and error summary from listing
  -ftsymbols                  specify symbols in listing
  -fno-diagnostics-show-option
                              suppress output of option that directly
                              controls the diagnostic


Compiler dialect configuration options:
  -freserved-words=<value>
                              use of complete/fixed reserved words
  -ftab-width=1..12           number of spaces that are assumed for tabs
  -ftext-column=72..255       right margin column number for fixed-form reference format
  -fpic-length=<number>       maximum number of characters allowed in the PICTURE
                              character-string
  -fword-length=1..63         maximum word-length for COBOL (= programmer defined) words
  -fliteral-length=<number>
                              maximum literal size in general
  -fnumeric-literal-length=1..38
                              maximum numeric literal size
  -fdefaultbyte=<value>       default initialization for fields without VALUE, may be
                              one of
                              *  character in quotes
                              *  decimal 0..255 representing a character
                              *  "init"  to  initialize  to  PICTURE/USAGE
                              *  "none" to do no explicit initialization
                              *  default: "init"
  -fformat=<value>            default reference-format, may be one of: FIXED, FREE,
                              COBOL85, VARIABLE, XOPEN, XCARD, CRT, TERMINAL, COBOLX
  -fbinary-size=<value>       binary byte size - defines the allocated bytes according to
                              PIC, may be one of: 2-4-8, 1-2-4-8, 1--8
  -fbinary-byteorder=<value>
                              binary byte order, may be one of: native, big-endian
  -fassign-clause=<value>
                              how to interpret 'ASSIGN word': as 'ASSIGN EXTERNAL word' or 'ASSIGN
                              DYNAMIC word', may be one of: dynamic, external,
                              ibm (= external), mf (= dynamic)
  -fscreen-section-rules=<value>
                              which compiler's rules to apply to SCREEN SECTION item
                              clauses, may be one of: acu, gc, mf, rm, std, xopen
  -fdpc-in-data=<value>       whether DECIMAL-POINT IS COMMA has effect in XML/JSON
                              GENERATE, may be one of: none, xml, json, all
  -ffilename-mapping          resolve file names at run time using environment variables
  -fpretty-display            alternate formatting of numeric fields
  -fbinary-truncate           numeric truncation according to ANSI
  -fcomplex-odo               allow complex OCCURS DEPENDING ON
```

-fodoslide adjust items following OCCURS DEPENDING (implies complex-odo)
-findirect-redefines allow REDEFINES to other than last equal level number
-frelax-syntax-checks allow certain syntax variations (e.g. REDEFINES position)
-fref-mod-zero-length allow zero length reference-modification (only changed with EC-BOUND-REF-MOD active)
-frelax-level-hierarchy
allow non-matching level numbers
-fselect-working require ASSIGN USING items to be in WORKING-STORAGE
-flocal-implies-recursive
LOCAL-STORAGE SECTION implies RECURSIVE attribute
-fsticky-linkage LINKAGE SECTION items remain allocated between invocations
-fmove-ibm MOVE operates as on IBM (left to right, byte by byte)
-fperform-osvs exit point of any currently executing perform is recognized if reached
-farithmetic-osvs limit precision in intermediate results to precision of final result (less accurate)
-fconstant-folding evaluate constant expressions at compile time
-fhostsign allow hexadecimal value 'F' for NUMERIC test of signed PACKED DECIMAL field
-fprogram-name-redefinition
program names don't lead to a reserved identifier
-faccept-update set WITH UPDATE clause as default for ACCEPT dest-item, instead of WITH NO UPDATE
-faccept-auto set WITH AUTO clause as default for ACCEPT dest-item, instead of WITH TAB
-fconsole-is-crt assume CONSOLE IS CRT if not set otherwise
-fno-echo-means-secure NO-ECHO
hides input with asterisks like SECURE
-fline-col-zero-default
assume a field DISPLAY starts at LINE 0 COL 0 (i.e. at the cursor), not LINE 1 COL 1
-fdisplay-special-fig-consts
special behaviour of DISPLAY SPACE/ALL X'01'/ALL X'02'/ALL X'07'
-fbinary-comp-1 COMP-1 is a 16-bit signed integer
-fnumeric-pointer POINTER is a 64-bit unsigned integer
-fmove-non-numeric-lit-to-numeric-is-zero
imply zero in move of non-numeric literal to numeric items
-fimplicit-assign-dynamic-var
implicitly define a variable if an ASSIGN DYNAMIC does not match any data item
-fdevice-mnemonics specifying device by mnemonic
-fxml-parse-xmlss XML PARSE XMLSS
-fareacheck check contents of Area A (when reference format supports Area A enforcement),
enabled checks include:
* division, section, paragraph names, level indicators (FD, SD, RD, and CD),
and toplevel numbers (01 and 77) must start in Area A;
* statements must not start in Area A; and
* separator periods must not be within Area A

```
-fcomment-paragraphs=<support>
                    comment paragraphs in IDENTIFICATION DIVISION (AUTHOR,
                    DATE-WRITTEN, ...)
-fcontrol-division=<support>
                    CONTROL DIVISION
-fpartial-replace-when-literal-src=<support>
                    apply partial replacing with literal source operand even
                    when it replaces with spaces only;
                    *  "skip" prevents such replacements
-fmemory-size-clause=<support>
                    MEMORY-SIZE clause
-fmultiple-file-tape-clause=<support>
                    MULTIPLE-FILE-TAPE clause
-flabel-records-clause=<support>
                    LABEL-RECORDS clause
-fvalue-of-clause=<support>
                    VALUE-OF clause
-fdata-records-clause=<support>
                    DATA-RECORDS clause
 -ftop-level-occurs-clause=<support>
                      OCCURS clause on top-level
 -fsame-as-clause=<support>
                    SAME AS clause
 -ftype-to-clause=<support>
                    TYPE TO clause
-fusage-type=<support>          USAGE  type-name
-fsynchronized-clause=<support>
                    SYNCHRONIZED clause
-fsync-left-right=<support>
                    LEFT/RIGHT phrases in SYNCHRONIZED clause
-fspecial-names-clause=<support>
                    SPECIAL-NAMES clause
  -fgoto-statement-without-name=<support>
                      GOTO statement without name
  -fstop-literal-statement=<support>
                    STOP-literal  statement
  -fstop-identifier-statement=<support>
                    STOP-identifier  statement
  -fstop-error-statement=<support>
                    STOP ERROR statement
 -fdebugging-mode=<support>
                    DEBUGGING MODE and debugging indicator
-fuse-for-debugging=<support>
                    USE FOR DEBUGGING
-fpadding-character-clause=<support>
                    PADDING CHARACTER clause
 -fnext-sentence-phrase=<support>
                    NEXT SENTENCE phrase
-flisting-statements=<support>
                    listing-directive statements EJECT, SKIP1, SKIP2, SKIP3
-ftitle-statement=<support>
```

```
                                listing-directive statement TITLE
 -fentry-statement=<support>
                                ENTRY statement
-fmove-noninteger-to-alphanumeric=<support>
                                 move noninteger to alphanumeric
  -fmove-figurative-constant-to-numeric=<support>
                                 move figurative constants to numeric
  -fmove-figurative-space-to-numeric=<support>
                                 move figurative constant SPACE to numeric
  -fmove-figurative-quote-to-numeric=<support>
                                  move figurative constant QUOTE to numeric
 -fodo-without-to=<support>
                                OCCURS DEPENDING ON without to
 -fsection-segments=<support>
                                section  segments
-falter-statement=<support>
                                ALTER statement
-fcall-overflow=<support>
                                OVERFLOW clause for CALL
  -fnumeric-boolean=<support>
                                boolean literals (B'1010')
-fhexadecimal-boolean=<support>
                                hexadecimal-boolean literals (BX'A')
  -fnational-literals=<support>
                                national literals (N'UTF-16 string')
  -fhexadecimal-national-literals=<support>
                                hexadecimal-national literals (NX'265E')
  -fnational-character-literals=<support>
                                 non-standard national literals (NC'UTF-16 string')
  -fhp-octal-literals=<support>
                                  HP COBOL octal literals (%377)
 -facu-literals=<support>
                                ACUCOBOL-GT literals (#B #O #H #X)
 -fword-continuation=<support>
                                 continuation of COBOL words
  -fnot-exception-before-exception=<support>
                                NOT ON EXCEPTION before ON EXCEPTION
-faccept-display-extensions=<support>
                                extensions to ACCEPT and DISPLAY
-frenames-uncommon-levels=<support>
                                RENAMES of 01-, 66- and 77-level items
-flarger-redefines=<support>
                                allow larger REDEFINES items
  -fsymbolic-constant=<support>
                                constants defined in SPECIAL-NAMES
-fconstant-78=<support>
                                constant with level 78 item (note: has left to right
                                precedence in expressions)
-fconstant-01=<support>
                                constant with level 01 CONSTANT AS/FROM item
-fperform-varying-without-by=<support>
```

```
                    PERFORM VARYING without BY phrase (implies BY 1)
-freference-out-of-declaratives=<support>
                    references to sections not in DECLARATIVES from within
                    DECLARATIVES
-fprogram-prototypes=<support>
                    CALL/CANCEL with program-prototype-name
-fcall-convention-mnemonic=<support>
                    specifying call-convention by mnemonic
-fcall-convention-linkage=<support>
                    specifying call-convention by WITH ... LINKAGE
-fnumeric-value-for-edited-item=<support>
                    numeric literals in VALUE clause of numeric-edited items
-fincorrect-conf-sec-order=<support>
                    incorrect order of CONFIGURATION SECTION paragraphs
-fdefine-constant-directive=<support>
                    allow >> DEFINE CONSTANT var AS literal
-ffree-redefines-position=<support>
                    REDEFINES clause not following entry-name in definition
-frecords-mismatch-record-clause=<support>
                    record sizes does not match RECORD clause
-frecord-delimiter=<support>
                    RECORD DELIMITER clause
-fsequential-delimiters=<support>
                    BINARY-SEQUENTIAL and LINE-SEQUENTIAL phrases in
                    RECORD DELIMITER
-frecord-delim-with-fixed-recs=<support>
                    RECORD DELIMITER clause on file with fixed-length records
-fmissing-statement=<support>
                    missing statement (e.g. empty IF / PERFORM)
-fmissing-period=<support>
                    missing period in PROCEDURE DIVISION (when reference format
                    supports Area A enforcement)
-fzero-length-literals=<support>
                    zero-length literals, e.g. '' and ""
-fxml-generate-extra-phrases=<support>
                    XML GENERATE's phrases other than COUNT IN
-fcontinue-after=<support>
                    AFTER phrase in CONTINUE statement
-fgoto-entry=<support>          ENTRY FOR GOTO and GOTO ENTRY statements
-fassign-variable=<support>
                    ASSIGN [TO] variable in SELECT
-fassign-using-variable=<support>
                    ASSIGN USING/VARYING variable in SELECT
-fassign-ext-dyn=<support>
                    ASSIGN EXTERNAL/DYNAMIC in SELECT
-fassign-disk-from=<support>
                    ASSIGN DISK FROM variable in SELECT
-fvsam-status=<support>
                    VSAM status in FILE STATUS
-fself-call-recursive=<support>
                    CALL to own PROGRAM-ID implies RECURSIVE attribute
```

```
  -frecord-contains-depending-clause=<support>
                                DEPENDING clause in RECORD CONTAINS
  -fpicture-l=<support>         PICTURE string with 'L' character
        where <support> is one of the following:
        'ok', 'warning', 'archaic', 'obsolete', 'skip', 'ignore', 'error',
        'unconformable'

        -fnot-reserved=<word>     word to be taken out of the reserved words list
 -freserved=<word>              word to be added to reserved words list
 -freserved=<word>:<alias>
                                word to be added to reserved words list as alias
 -fnot-register=<word>          special register to disable
 -fregister=<word> or <word>:<definition>
                                special register to enable
```

Each file specified on the cobc command constitutes a Compilation Unit. A compilation unit may be a single GnuCOBOL program — with or without nested subprograms(see Section 11.2 [Independent vs Contained vs Nested Subprograms], page 719) — or multiple GnuCOBOL programs, separated by END PROGRAM or END FUNCTION marker lines, as appropriate. See Section 11.2 [Independent vs Contained vs Nested Subprograms], page 719, for some examples of these marker lines.

A compilation unit may also be a C-language source program, recognized as such by having a file extension of .c or an assembly-language program, recognized by its file extension of .s. In such a case, COBOL compilation of that file will be bypassed by the cobc command; instead, the file will be passed directly to the C compiler or assembler (executed automatically by cobc).

A compilation unit may also be an object-code module (output from the C compiler), recognized as such by having a file extension of .o. In these situations, all compilation will be bypassed, and the object code will be "bound" into the generated executable by the linker (an ld command executed internally by the cobc command).

Pre-compiled object-code subprograms may be automatically located by the GnuCOBOL compiler and the loader by using the LD_LIBRARY_PATH compilation-time environment variable (see Section 10.1.4 [Compilation Time Environment Variables], page 687). If they are locatable through that environment variable, they need not be named on the cobc command.

The collection of compilation units supplied on a single cobc execution constitute a Compilation Group. All executable code produced from a single compilation group will be collected together into a single executable file, whose filename will be the same as that of the first compilation unit specified on the cobc command.

The simplest mode of compilation is to generate a single executable file from one or more GnuCOBOL source files:

```
cobc -x mainprog.cbl sub1.cbl sub2.cbl
```

The main program must be the first program found in the first compilation unit (mainprog.cbl). The remainder of that compilation unit as well as the rest of the files in the compilation group (sub1.cbl and sub2.cbl) must be independent and/or contained subprograms (see Section 11.2 [Independent vs Contained vs Nested Subprograms], page 719).

This command assumes that all source files are in the directory from which the cobc command was executed. You are, of course, free to include full pathnames with any filename, if necessary.

With the -x switch on the compiler command, a single directly-executable executable file (UNIX, Windows/Cygwin, OSX) or "exe" file (Windows, Windows/MinGW) will be gener-

ated. This executable file has the compiled code for all COBOL programs contained within the compilation group specified on the cobc command included in the file.

Any subroutines or user-defined functions that weren't included in any of the source files comprising the compilation group will be treated as dynamically loadable subprograms (see Section 11.4 [Dynamic vs Static Subprograms], page 721).

Optionally, the -o switch may be used in addition to -x to specify the name of the generated executable file. If -o switch is not specified, the filename of the 1st compilation unit will be used as the name of the executable file. The appropriate extension for the generated file (.exe, on a Windows computer, for example) will be added to the filename that is explicitly specified or implicitly assumed for the output file.

Compilations may be performed to generate dynamically-loadable modules (or dynamically-loadable libraries, as they are frequently called). These compilations are performed by using the -m switch instead of -x switch:

```
cobc -m mainprog.cbl sub1.cbl sub2.cbl
```

When the -m switch is used, an operating-system-specific dynamically-loadable module is generated for each individual compilation unit, using the filename of each compilation unit as its module filename and either an extension of .so (UNIX, Windows/Cygwin), .dylib (OSX) or .dll (Windows, Windows/MinGW).

You may compile GnuCOBOL subprograms into assembler source code which can then be assembled and linked with a main program when that main program is compiled. To create such an assembler source file, compile the subprogram(s) as follows:

```
cobc -S sprog1.cbl
```

The above generates an assembler source file named sprog1.s. If you have multiple subprograms to compile this way, just string their file names out on the command. Each will be translated to its own assembler source file.

Later, when you wish to compile a calling program and combine any needed assembly language subroutines in (as static subroutines — see Section 11.4 [Dynamic vs Static Subprograms], page 721), use a command such as this:

```
cobc -x mainprog.cbl sprog1.s
```

10.1.2 cobc option -Xref an example

The following shows the output from using -Xref.

```
Using the internal -Xref as extra options -X -t prog.list :

GnuCOBOL 3.1.2.0          prog.cbl                Thu Nov 25 15:31:52 2021 Page 0001

LINE     PG/LN   A...B.....................................................

000001   000001  IDENTIFICATION     DIVISION.
000002   000002  PROGRAM-ID.         prog.
000003   000003  ENVIRONMENT DIVISION.
000004   000004  CONFIGURATION SECTION.
000005   000005  DATA                DIVISION.
000006   000006  WORKING-STORAGE     SECTION.
000007   000007  COPY 'values.cpy'.
000001C  000001  78  I    VALUE 20.
000002C  000002  78  J    VALUE 5000.
000003C  000003  78  M    VALUE 5.
```

```
000008  000008 01   SETUP-REC.
000009  000009      05  FL1         PIC X(04).
000010  000010      05  FL2         PIC ZZZZZ.
000011  000011      05  FL3         PIC 9(04).
000012  000012      05  FL4         PIC  9(08)  COMP.
000013  000013      05  FL5         PIC 9(04) COMP-4.
000014  000014      05  FL6         PIC Z,ZZZ.99.
000015  000015      05  FL7         PIC S9(05) SIGN LEADING SEPARATE.
000016  000016      05  FL8         PIC X(04).
000017  000017      05  FL9 REDEFINES FL8 PIC 9(04).
000018  000018      05  FLA.
000019  000019          10  FLB OCCURS I TIMES.
000020  000020              15  FLC PIC X(02).
000021  000021          10  FLD     PIC X(20).
000022  000022      05  FLD1        PIC X(100).
000023  000025      05  FLD3        PIC X(3).
000024  000026      05  FLD4        PIC X(4).
000025  000023      05  FLD2  OCCURS M TO J TIMES DEPENDING ON FL5.
000026  000024          10  FILLER PIC X(01).
000027  000027  PROCEDURE        DIVISION.
000028  000028      STOP RUN.
```

```
GnuCOBOL 3.1.2.0        prog.cbl        Thu Nov 25 15:31:52 2021        Page 0002

NAME                            DEFINED             REFERENCES

SETUP-REC                       8       referenced by child
FL1                             9       not referenced
FL2                             10      not referenced
FL3                             11      not referenced
FL4                             12      not referenced
FL5                             13        25                            x1
FL6                             14      not referenced
FL7                             15      not referenced
FL8                             16      not referenced
FL9                             17      not referenced
FLA                             18      not referenced
FLB                             19      not referenced
FLC                             20      not referenced
FLD                             21      not referenced
FLD1                            22      not referenced
FLD3                            23      not referenced
FLD4                            24      not referenced
FLD2                            25      not referenced
```

```
GnuCOBOL 3.1.2.0        prog.cbl        Thu Nov 25 15:31:52 2021        Page 0003

LABEL                           DEFINED             REFERENCES

E prog                          27
```

0 warnings in compilation group
0 errors in compilation group

10.1.3 Cross Reference listing using cobxref

This program is found in the contrib area or by itself in Sourceforge under its own name (cobxref).

The following shows the output from using cobxref with the same program example.

Note that cobxref also reports the COPY depth which can be up to 9 and includes the program being compiled as depth 1.

Using cobxref with command cobxref prog.cbl :

```
ACS Cobol Xref v2.02.03                Dictionary  File  for  PROG        25/11/2021   15:22:00:04
                                         Page        1

     1   000001 IDENTIFICATION        DIVISION.
     2   000002 PROGRAM-ID.           prog.
     3   000003 ENVIRONMENT DIVISION.
     4   000004 CONFIGURATION SECTION.
     5   000005 DATA                  DIVISION.
     6   000006 WORKING-STORAGE    SECTION.
     7   000007*COPY 'values.cpy'.
     8   000001 78     I      VALUE  2 0 .
     9   000002 78     J      VALUE  5 0 0 0 .
    10   000003 78     M      VALUE  5 .
    11   000008 01     SETUP-REC.
    12   000009          05   FL1          PIC X(04).
    13   000010          05   FL2          PIC ZZZZZ.
    14   000011          05   FL3          PIC 9(04).
    15   000012          05   FL4          PIC  9(08)  COMP.
    16   000013          05   FL5          PIC 9(04) COMP-4.
    17   000014          05   FL6          PIC Z,ZZZ.99.
    18   000015          05   FL7          PIC S9(05) SIGN LEADING SEPARATE.
    19   000016          05   FL8          PIC X(04).
    20   000017          05   FL9   REDEFINES FL8 PIC 9(04).
    21   000018          05   FLA.
    22   000019               10    FLB OCCURS I TIMES.
    23   000020                     15    FLC PIC X(02).
    24   000021               10    FLD     PIC X(20).
    25   000022          05   FLD1         PIC X(100).
    26   000025          05   FLD3         PIC X(3).
    27   000026          05   FLD4         PIC X(4).
    28   000023          05   FLD2   OCCURS M TO J TIMES DEPENDING ON FL5.
    29   000024               10    FILLER PIC X(01).
    30   000027 PROCEDURE             DIVISION.
    31   000028          STOP  R U N .
    32              *>>>Info: Total Copy Depth Used = 02

ACS Cobol Xref v2.02.03                Dictionary  File  for  PROG        25/11/2021   15:22:00:04
```

```
                                        Page      2
Symbols of Module: PROG (PROG)
--------------------------------

Data Section (WORKING-STORAGE)   Defn       Locations
--------------------------------+-------------------------------------------------------------

FL1                              000012W
FL2                              000013W
FL3                              000014W
FL4                              000015W
FL5                              000016W   000028
FL6                              000017W
FL7                              000018W
FL8                              000019W   000020
FL9                              000020W
FLA                              000021W
FLB                              000022W
FLC                              000023W
FLD                              000024W
FLD1                             000025W
FLD2                             000028W
FLD3                             000026W
FLD4                             000027W
I                                000008W   000022
J                                000009W   000028
M                                000010W   000028
SETUP-REC                        000011W

ACS Cobol Xref v2.02.03            Dictionary  File  for  PROG       25/11/2021   15:22:00:04

                                        Page      3
Symbols of Module: PROG (PROG)
--------------------------------

Procedure                        Defn       Locations
--------------------------------+-------------------------------------------------------------

None
```

10.1.4 Compilation Time Environment Variables

The following are the various environment variables that can play a role in the compilation of GnuCOBOL programs.

COB_CC * Set to the name of the C compiler you wish GnuCOBOL to use.

Use this feature at your own risk – you should always use the c compiler your gnucobol build was generated for.

COB_CFLAGS *
Set to any switches that you'd like to pass on to the C compiler from the cobc compiler (in addition to any that cobc will specify).

COB_CONFIG_DIR *
: Set to the path to the folder where GnuCOBOL config files are kept.

COB_COPY_DIR *
: If copybooks your program needs are not stored in the same directory as your program, set this environment variable to the folder in which the copybooks may be found (IBM mainframe programmers will recognize this as SYSLIB).

COB_LDADD
: Set to any additional linker switches (ld) that can specify where standard libraries that must be linked with the program can be found. The default is "" (empty).

COB_LDFLAGS
: Set to any linker (ld) switches that you'd like to pass on to the C compiler from the cobc compiler (in addition to any that cobc will specify).

COB_LIBS *
: Set to any linker switches (ld) that specify where standard libraries that must be linked with the program can be found.

COBCPY
: This environment variable provides an additional means of specifying where copybooks may be found by the compiler (see also COB COPY DIR, above).

LD_LIBRARY_PATH
: If you are planning on using static-linked subroutine libraries, set this variable to the path of the directory containing your libraries.

TMPDIR
TMP
: Set to a directory/folder appropriate to create temporary files in. The intermediate working files created by the compiler will be created here (and deleted once they're no longer needed).

  The variable TMPDIR is checked for a valid path first; if that isn't set, then TMP is checked.

  On a Windows system, the TMP environment variable is normally set for you when you logon. If you wish to use a different temporary folder, you may set TMPDIR yourself and have no fear of disrupting other Windows software that relies on TMP.

---

* The starred environment variables above have default values, established when the version of GnuCOBOL you are using was built. To see these default values, as well as other build-specific information, execute the command:

```
cobc -i
```

10.1.5 Predefined Compilation Variables

GnuCOBOL defines compilation variables when certain conditions are true.

If the condition associated with a variable is false, the variable is not defined during compilations.

DEBUG
: The -d debug flag is specified.

EXECUTABLE
: The module being compiled contains the main program.

GCCOMP The size of a COMP item is determined according to the GnuCOBOL scheme, where for a PICTURE of length:

1-2	item has 1 byte
3-4	item has 2 bytes
5-9	item has 4 bytes
10-18	item has 8 bytes.

HOSTSIGNS
A signed packed-decimal item's value may be considered NUMERIC if the sign has value X"F".

IBMCOMP The size of a COMP item is determined according to the IBM scheme, where for a PICTURE of length:

1-4	item has 2 bytes
5-9	item has 4 bytes
10-18	item has 8 bytes.

MODULE The module being compiled does not contain the main program.

NOHOSTSIGNS
A signed packed-decimal item's value may not be considered

NUMERIC if the sign has value X"F".

NOIBMCOMP
The size of a COMP item is not determined according to the IBM scheme.

NOSTICKYLINKAGE
Sticky-linkage (linkage-section items remaining allocated between invocations) is not enabled.

NOTRUNC Numeric data items are truncated according to their internal representation.

P64 Pointers are greater than 32 bits long

STICKY-LINKAGE
Sticky-linkage (linkage-section items remaining allocated between invocations) is enabled.

TRUNC Numeric data items are truncated according to their PICTURE clauses.

While still supported, this may well be removed in the future and should not be used. See GCCOMP and GNUCOBOL instead:

OCCOMP The size of a COMP item is determined according to the GnuCOBOL scheme, where for a PICTURE of length:

1 - 2	*item* = 1 byte
3 - 4	*item* = 2 bytes
5 - 9	*item* = 4 bytes
10 - 18	*item* = 8 bytes

10.1.6 Locating Copybooks

The GnuCOBOL compiler will attempt to locate copybooks by searching for them in the fol- lowing folders. The search will occur in the sequence shown below, and will terminate once a copybook is found.

1. The folder named as the library-name-1 on the COPY statement (see Section 3.2 [COPY], page 63).
2. The folder in which the program being compiled resides.
3. The folder named on the -I switch.
4. Each of the folders named on the COB_CPY_DIR compilation-time environment variable (see Section 10.1.4 [Compilation Time Environment Variables], page 687) in order of presentation.
5. Each of the folders named on the COBCPY compilation-time environment variable (see Section 10.1.4 [Compilation Time Environment Variables], page 687) in order of presentation.

   A single folder may be named or multiple folders may be specified, separated by a system-appropriate delimiter character. When multiple folders are specified, they will be searched in the order they are named on the environment variable.

   If the GnuCOBOL compiler you are using was built to utilize a native Windows environment, use a semicolon (';') as the delimiter character.

   If, however, the GnuCOBOL compiler was built for a Unix, OSX or Linux environment, or was built for a Windows environment utilizing either the Cygwin or MinGW Unix emulators, use a colon character (':') as the delimiter.
6. For v1.1 only, the single folder specified on the COB_COPY_DIR environment variable. (If not defined, for order see results of 'cobc –info').

As each of the above folders is searched for a copybook — COPY XXXXXXXX., for example — the GnuCOBOL compiler will attempt to locate the copybook file by any of the following names, in the sequence shown:

XXXXXXXX.CPY
XXXXXXXX.CBL
XXXXXXXX.COB
XXXXXXXX.cpy
XXXXXXXX.cbl
XXXXXXXX.cob
XXXXXXXX

The COPY statement is case-sensitive on UNIX systems; COPY copybookname and COPY COPYBOOKNAME will both fail to locate the CopyBookName copybook on a UNIX system.

Windows implementations of GnuCOBOL may, or may not, be similarly case sensitive with regard to copybook names, depending upon the Windows version and GnuCOBOL build options — it is safest to simply treat the COPY command as case-sensitive in all environments.

It is possible, however, to automatically cause all COPY statements to "fold" the names of all copybooks to upper-case by specifying the -ffold-copy switch with the upper option (i.e. --fold-copy=upper) to the GnuCOBOL compiler. Similarly, names could be folded to lower-case by using the lower option (i.e. --fold-copy=lower. If copybook libraries are main- tained entirely using upper- or lower-case file names and extensions, either of these options will allow copybooks to be found regardless of how the programmer entered their names on COPY statements.

Case-folding may also be turned on and off within the program source code using the CDF >>SET statement (see Section 3.6 [>>SET], page 70).

10.1.7 Compiler Configuration Files

GnuCOBOL uses compiler configuration files to define various options that will control the compilation process. These configuration files are specified using the -conf switch compilation switch and are found in the folder defined by the COB_CONFIG_DIR compilation-time environment variable (see Section 10.1.4 [Compilation Time Environment Variables], page 687).

If this is not defined under *nix it will default to /usr/local/share/gnucobol/config.

The following is a verbatim listing of the default configuration file (the one used if you don't specify the -conf switch), just to show you the types of settings that may appear and taken from v3.1.2 :

```
# GnuCOBOL compiler configuration #
# Copyright (C) 2001-2012, 2014-2025 Free Software Foundation, Inc.
# Written by Keisuke Nishida, Roger While, Simon Sobisch, Edward Hart, # Ron
Norman
#
# This file is part of GnuCOBOL. #
# The GnuCOBOL compiler is free software: you can redistribute it
# and/or modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation, either version 3 of the #
License, or (at your option) any later version.
#
# GnuCOBOL is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of #
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.          See the
# GNU General Public License for more details. #
# You should have received a copy of the GNU General Public License #
along with GnuCOBOL.      If not, see <https://www.gnu.org/licenses/>.


# Value: any string
name: "GnuCOBOL"

#  Value:  enum
standard-define                          0
#           CB_STD_GC  = 0,
#           CB_STD_MF,
#           CB_STD_IBM,
#           CB_STD_MVS,
#           CB_STD_BS2000,
#           CB_STD_ACU,
#           CB_STD_85,
#           CB_STD_2002,
#           CB_STD_2014

# Value: int
```

```
tab-width:                      8
text-column:                    72
# Maximum word-length for COBOL words / Programmer defined words # Be
aware that GC checks the word length against COB_MAX_WORDLEN # first
(currently 63)
word-length:                    63

# Maximum literal size in general literal-
length:                         8191

# Maximum numeric literal size (absolute maximum: 38)
numeric-literal-length:         38

# Maximum number of characters allowed in the character-string (max. 255)
pic-length:                     255

# Default assign type
# Value: 'dynamic', 'external'
assign-clause:                  dynamic

# If yes, file names are resolved at run time using #
environment variables.
# For example, given ASSIGN TO "DATAFILE", the file name will be #
    1. the value of environment variable 'DD_DATAFILE' or
#   2. the value of environment variable 'dd_DATAFILE' or #
    3. the value of environment variable 'DATAFILE' or
#   4. the literal "DATAFILE"
# If no, the value of the assign clause is the file name. #
filename-mapping:               yes

# Alternate formatting of numeric fields
pretty-display:                 yes

# Allow complex OCCURS DEPENDING ON
complex-odo:                    no

# Allow REDEFINES to other than last equal level number
indirect-redefines:             no

# Binary byte size - defines the allocated bytes according to PIC #
Value:              signed   unsigned   bytes
#                   ------   --------   -----
# '2-4-8'            1 -  4    same        2
#                    5 -  9    same        4
#                   10 - 18    same        8
#
# '1-2-4-8'          1 -  2    same        1
#                    3 -  4    same        2
#                    5 -  9    same        4
#                   10 - 18    same        8
```

```
#
# '1--8'          1 -  2     1 -  2       1
#                 3 -  4     3 -  4       2
#                 5 -  6     5 -  7       3
#                 7 -  9     8 -  9       4
#                10 - 11    10 - 12       5
#                12 - 14    13 - 14       6
#                15 - 16    15 - 16       7
#                17 - 18    17 - 18       8
#
binary-size:                    1-2-4-8

# Numeric truncation according to ANSI
binary-truncate:                yes

# Binary byte order
# Value: 'native', 'big-endian'
binary-byteorder:               big-endian

# Allow larger REDEFINES items
larger-redefines-ok:            no

# Allow certain syntax variations (eg. REDEFINES position)
relax-syntax-checks:            no

# Allow zero length reference-modification
# (only checked with active EC-BOUND-REF-MOD)
ref-mod-zero-length:            yes

# Perform type OSVS - If yes, the exit point of any currently #
executing perform is recognized if reached.
perform-osvs:                   no

# Compute intermediate decimal results like IBM OSVS
arithmetic-osvs:                no

# MOVE like IBM (mvc); left to right, byte by byte move-ibm: no

# SELECT RELATIVE KEY and ASSIGN fields must be in WORKING-STORAGE
select-working:         no

# LOCAL-STORAGE SECTION implies RECURSIVE attribute
local-implies-recursive:                no

# If yes, LINKAGE SECTION items remain allocated
# between invocations.
sticky-linkage:                 no

# If yes, allow non-matching level numbers
relax-level-hierarchy:          no
```

```
# If yes, evaluate constant expressions at compile time
constant-folding:                yes

# Allow Hex 'F' for NUMERIC test of signed PACKED DECIMAL
field hostsign:                  no

# If yes, set WITH UPDATE clause as default for ACCEPT dest-item, # except if WITH
NO UPDATE clause is used
accept-update:                   no

# If yes, set WITH AUTO clause as default for ACCEPT dest-item, # except if
WITH TAB clause is used
accept-auto:                     no

# If yes, DISPLAYs and ACCEPTs are, by default, done on the CRT (i.e., using #
curses).
console-is-crt:                  no

# If yes, allow redefinition of the current program's name. This prevents its # use
in a prototype-format CALL/CANCEL statement.
program-name-redefinition:       yes

# If yes, NO ECHO/NO-ECHO/OFF is the same as SECURE (hiding input with # asterisks,
not spaces).
no-echo-means-secure:            no

# If yes, the first item in a field screen ACCEPT/DISPLAY (e.g. DISPLAY x UPON # CRT) is
located after the previous ACCEPT/DISPLAY (as though LINE 0 COL 0 had # been
specified).
line-col-zero-default:           yes

# If yes, DISPLAY SPACES acts as ERASE EOS, DISPLAY X"01" acts as ERASE EOL, # DISPLAY
X"02" acts as BLANK SCREEEN and DISPLAY X"07" acts as BELL. Note
# DISPLAY LOW-VALUE is excluded from this; it will always just position the #
cursor.
display-special-fig-consts:      no

# If yes, COMP-1 is a signed 16-bit integer and any PICTURE clause is ignored. binary-
comp-1:                          no

# If yes, POINTER is handled as BINARY-DOUBLE UNSIGNED instead of its own class numeric-
pointer:                         no

# auto-adjust to zero like MicroFocus does
move-non-numeric-lit-to-numeric-is-zero:  no

# If yes, implicitly define a variable for an ASSIGN DYNAMIC which does not # match an
existing data item.
implicit-assign-dynamic-var:     yes
```

```
# What rules to apply to SCREEN SECTION items clauses
screen-section-rules:                    gc

# Whether DECIMAL-POINT IS COMMA has effect in XML/JSON GENERATE dpc-in-
data:                                    xml

# Dialect features
#  Value: 'ok', 'warning', 'archaic', 'obsolete', 'skip', 'ignore', 'error', #
         'unconformable'

alter-statement:                         obsolete
comment-paragraphs:                      obsolete
call-overflow:                           archaic
data-records-clause:                     obsolete
debugging-mode:                          ok
use-for-debugging:                       ok
listing-statements:                      skip      # may be a user-defined word
title-statement:                         skip      # may be a user-defined word
entry-statement:                         ok
goto-statement-without-name:             obsolete
label-records-clause:                    obsolete
memory-size-clause:                      obsolete
move-noninteger-to-alphanumeric:         error
move-figurative-constant-to-numeric:     archaic
move-figurative-space-to-numeric:        error
move-figurative-quote-to-numeric:        obsolete
multiple-file-tape-clause:               obsolete
next-sentence-phrase:                    archaic
odo-without-to:                          warning
padding-character-clause:                obsolete
section-segments:                        ignore
stop-literal-statement:                  obsolete
stop-identifier-statement:               obsolete
same-as-clause:                          ok
type-to-clause:                          ok
usage-type:                                      ok
synchronized-clause:                     ok
special-names-clause:                    ok
top-level-occurs-clause:                 ok
value-of-clause:                         obsolete
numeric-boolean:                         ok
hexadecimal-boolean:                     ok
national-literals:                       ok
hexadecimal-national-literals:           ok
national-character-literals:             warning
# TO-DO: Add separate config option for H"..." to be unsupported,numeric,non-numeric(acu)
acu-literals:                            unconformable
hp-octal-literals:                       unconformable
word-continuation:                       warning
not-exception-before-exception:          ok
accept-display-extensions:               ok
```

```
renames-uncommon-levels:                ok
symbolic-constant:                      ok
constant-78:                            ok
constant-01:                            ok
perform-varying-without-by:             ok
reference-out-of-declaratives:          warning
program-prototypes:                     ok
call-convention-mnemonic:               ok
call-convention-linkage:                ok
numeric-value-for-edited-item:          ok
incorrect-conf-sec-order:               ok
define-constant-directive:              archaic
free-redefines-position:                warning
records-mismatch-record-clause      warning
record-delimiter:                       ok
sequential-delimiters:                  ok
record-delim-with-fixed-recs:           ok
missing-statement:                      warning
zero-length-literals:                   ok
xml-generate-extra-phrases:             ok
continue-after:                         ok
goto-entry:                             warning
assign-variable:                        ok
assign-using-variable:                  ok
assign-ext-dyn:                         ok
assign-disk-from:                       ok
vsam-status:                            ignore

# use complete word list; synonyms and exceptions are specified below
reserved-words:         default

# not-reserved:
# Value: Word to be taken out of the reserved words list not-
reserved:       TERMINAL
# reserved:
#     Entries of the form word-1=word-2 define word-1 as an alias for default #
reserved word word-2. No spaces are allowed around the equal sign. reserved:
    AUTO-SKIP=AUTO
reserved:       AUTOTERMINATE=AUTO
reserved:       BACKGROUND-COLOUR=BACKGROUND-COLOR
reserved:       BEEP=BELL
reserved:       BINARY-INT=BINARY-LONG  reserved:
                BINARY-LONG-LONG=BINARY-DOUBLE
reserved:       CELLS=CELL
reserved:       COLOURS=COLORS
reserved:       EMPTY-CHECK=REQUIRED
reserved:       EQUALS=EQUAL
reserved:       FOREGROUND-COLOUR=FOREGROUND-COLOR
reserved:       HIGH-VALUES=HIGH-VALUE
reserved:       INITIALISE=INITIALIZE
reserved:       INITIALISED=INITIALIZED
```

reserved:	LENGTH-CHECK=FULL reserved:
	LOW-VALUES=LOW-VALUE
reserved:	ORGANISATION=ORGANIZATION
reserved:	PIXELS=PIXEL
reserved:	SYNCHRONISED=SYNCHRONIZED
reserved:	TIMEOUT=TIME-OUT
reserved:	VALUES=VALUE
reserved:	ZEROES=ZERO
reserved:	ZEROS=ZERO

10.2 Running Programs

Once GnuCOBOL programs have been compiled into either directly-executable programs (created via the -x switch) or dynamically-loadable libraries (created via the -m switch), those programs may be executed from any shell environment. The exact manner in which the two are executed will differ, as described in the upcoming sections.

10.2.1 Direct Execution

GnuCOBOL programs compiled with the -x switch will be generated as directly-executable programs. For example, a native Windows or Windows/MinGW build of GnuCOBOL will generate an .exe file when the -x switch switch is specified to the compiler.

On Unix, OSX, or Windows/Cygwin builds, the -x switch switch will generate an executable binary file, usually with no particular extension unless one is explicitly requested of the compiler via the -o switch.

On a UNIX system this means the programs may be executed from a command shell such as bash, csh, ksh and so forth. When a GnuCOBOL program runs on a Windows system, it runs within a console window (i.e. cmd.exe). OSX versions of GnuCOBOL programs run within a terminal.app window.

Interactions between the program and the user will take place using the standard input, standard output and standard error streams. Any screen section I/O performed by the program will take place within the terminal window.

Direct program execution syntax is [path]program [arguments].

For example:

```
/usr/local/printaccount  ACCT=6625378
```

or

```
C:\\Users\\Me\\Documents\\Programs\\printaccount.exe  ACCT=6625378
```

10.2.2 Executing Dynamically-Loadable Libraries

As discussed previously, dynamically-loadable libraries are created via the compiler's -m switch. Once so created, the program(s) in these libraries are executed from the command line (via the GnuCOBOL cobcrun utility), or as dynamically-loadable subprograms.

10.2.2.1 cobcrun - Command-line Execution

It is possible to generate executable modules for all GnuCOBOL programs, not just subprograms, by choosing to use the -m switch option to specify the loader output format, even for main programs.

Some may prefer to compile their GnuCOBOL main programs into these dynamically-loadable modules in the interests of using the same general compilation command for all programs without having to think "Is it a main program or a subprogram?".

Main programs compiled in this manner should be executed as follows:

```
[path]cobcrun program [arguments]
```

Do not specify the .so or .dll extension on the program name. The program value must exactly match the primary entry-point name of the main program (including upper- and lower- case letters), unless you are planning on using Call Folding (see Section 10.2.2.2 [Dynamically Loaded Subprograms], page 699).

The general usage and syntax of cobcrun is as follows as issued by running cobcrun -h (or –help) :

GnuCOBOL module loader

```
Usage: cobcrun [options] PROGRAM [parameter ...]
  or:    cobcrun options

Options:
  -h,  -help                          display this help and exit
  -V,  -version                       display cobcrun and runtime version and exit
  -i,  -info                          display runtime information (build/environment)
  -v,  -verbose                       display extended output with --info
  -c  <file>,  -config=<file>         set runtime configuration from <file>
  -r, -runtime-config                 display current runtime configuration
                                      (value and origin for all settings)
  -M <module>, -module=<module>       set entry point module name and/or load path
                                      where -M module prepends any directory to the
                                      dynamic link loader library search path
                                      and any basename to the module preload list (COB_LIBRARY_PATH
                                      and/or COB_PRELOAD)
```

Here are two examples of using cobcrun. First, on a Unix, OSX or Windows/Cygwin system:

```
cd /usr/local
cobcrun  printaccount  acct=6625378
```

Or, on a Native Windows or Windows/MinGW system:

```
cd C:\Users\Me\Documents\Programs
cobcrun printaccount.exe acct=6625378
```

Note how the cobcrun command does not allow a path to be specified with the program name — the directory in which the programs dynamically loadable module exists must either be the current directory or must be defined in the current PATH.

10.2.2.2 Dynamically Loaded Subprograms

Dynamically-loaded subprograms are executed (from a COBOL syntax point of view) just like any other subprograms. What makes them unique, however, is that they are loaded into memory only when they are actually used the first time during the execution of a program.

When a dynamically-loadable module needs to be loaded (because it is not already in memory from a previous subprogram execution), the dynamically-loadable library will be sought by libcob in each directory named in the library specified by the COB_LIBRARY_PATH run-time environment variable will be searched and if not found then. Finally, if it still cannot be located, execution will be terminated with an error message (libcob: Cannot find module 'xxxxxxxx'). For speed performance purposes all such should therefore be stored within the directories pointed to by the environment variable COB_LIBRARY_PATH run-time environment variable.

The process of locating dynamically-loadable modules is case sensitive on UNIX systems; CALL "dynsub" and CALL "DYNSUB" will both fail to locate the DynSub.so library on a UNIX system.

Windows implementations of GnuCOBOL may, or may not, be similarly case sensitive with regard to library names, depending upon the Windows version and GnuCOBOL build options — it is safest to simply treat library names as case sensitive in all environments.

It is possible, however, to automatically cause all library names to 'fold' to upper-case by specifying the -ffold-call switch with the "upper" option (i.e. --fold-call=upper) to

the GnuCOBOL compiler. Similarly, library names could be folded to lower-case by using the "lower" option (i.e. --fold-call=lower. If libraries are maintained entirely using upper- or lower-case file names, either of these options will allow libraries to be found regardless of how the programmer entered their names on CALL statements.

See Chapter 11 [Sub-Programming], page 719, for a complete discussion of sub-programming.

10.2.3 Run Time Environment Variables

The following is a list of the various environment variables that can play a role in the execution of GnuCOBOL programs.

COB_DISPLAY_WARNINGS
: If set to a value of 'Y', any run-time warnings (such as noting the implicit closing of open files when a GOBACK statement (see Section 7.8.21 [GOBACK], page 334) or STOP statement (see Section 7.8.46 [STOP], page 395) with the RUN option is executed) will be displayed. Any other value for this environment variable (including not setting the variable at all) will suppress such messages.

COB_LIBRARY_PATH
: At runtime, GnuCOBOL will attempt to locate and load any application dynami- cally loadable libraries used this variable that points to the directories specified, if it wasn't found there, using the PATH environment variable.

COB_LOAD_CASE
: If set to either UPPER or LOWER, this environment variable will internally convert referenced entry point names to either upper- or lower-case before initiating searches for dynamically loadable modules. The UPPER and LOWER values of the environment variable are actually case insensitive.

COB_PHYSICAL_CANCEL
: If set to 'Y', 'y' or '1', a CANCEL statement (see Section 7.8.6 [CANCEL], page 294) will physically unload a subprogram dynamically loadable module. If set to anything else, a CANCEL statement (see Section 7.8.6 [CANCEL], page 294) logically unloads a module so that subsequent use will re-initialize the module as if it had actually been reloaded, but the overhead of actually reloading the module will be avoided.

COB_PRE_LOAD
: If set to any non-null value, this variable will cause all dynamically loadable libraries to be loaded when the program begins execution (rather than searching for and loading the module upon first use).

COB_SET_DEBUG
: If a USE FOR DEBUGGING (see Section 7.5 [DECLARATIVES], page 250) section ex- ists, the code within it will be disabled unless this environment variable is set to a value of 'Y', 'y' or '1'.

COB_SET_TRACE
: If the -ftrace switch (trace procedures) or -ftraceall switch (trace procedures and statements) was used when the program was compiled, setting this environment variable to a value of 'Y' will activate the trace at the point the program begins execution. Setting this environment variable to any other value (or never setting it to ANY value) will disable tracing. Tracing, if configured by one of the two switches described above, can also be controlled via the the READY TRACE statement (see Section 7.8.35 [READY TRACE], page 369) and RESET TRACE statement (see Section 7.8.37 [RESET TRACE], page 371). If COB SET TRACE is set to Y, then

tracing will always occur regardless of the presence of READY TRACE or RESET TRACE so in effect they will have no action on program execution.

COB_SCREEN_ESC
: If set to any non-blank value, this variable allows a ACCEPT data-item statement (see Section 7.8.1.4 [ACCEPT data-item], page 267) to detect the Esc key.

COB_SCREEN_EXCEPTIONS
: Setting this variable to any non-blank value will allow the ACCEPT data-item statement (see Section 7.8.1.4 [ACCEPT data-item], page 267) to detect the pressing of the Esc, PgUp and PgDn keys.

COB_SORT_MEMORY
: The value of this variable (an integer) will be used to define how much memory will be allocated for use in sorting. If the value is 1048576 or greater, that value will be used "as is" as the amount of memory (in bytes) to allocate. If the value is less than 1048576, the value will specify how many MB of memory will be allocated. The default sort memory amount is 128 MB.

COB_SWITCH_n
: (n=0 to 15); These environment variables correspond to SWITCH-0 through SWITCH-15, defined in the SPECIAL-NAMES (see Section 5.1.3 [SPECIAL-NAMES], page 90) paragraph. Setting them to ON will activate them; any other value turns them off.

COB_SYNC If set to a value of upper- or lower-case 'p', this variable will force a file commit every time a file is written to (ensuring that data is immediately written to the file rather than retained in memory until a future commit occurs). This will slow-down update access to files, but will provide for better integrity in the event of a program failure.

COB_TRACE_FILE
: If set to any non-null value, this environment variable specifies the file to which all -ftrace switch and -ftraceall switch output will be written. If this is NOT set to a value, all -ftrace switch and -ftraceall switch output will be written to STDERR, where it may be piped via a "2> filename" on the command that executes the program.

DB_HOME If your GnuCOBOL build uses the Berkeley Database (BDB) package, use this environment variable to specify the folder in which the lock management files to be associated with all non-SORT files opened by the program will be stored. ORGANIZATION INDEXED (see Section 5.2.1.4 [ORGANIZATION INDEXED], page 114) files will also have their data file allocated in the folder pointed to by this environment variable, if it exists.. Having this variable defined will activate record locking features on the READ statement (see Section 7.8.34 [READ], page 365), REWRITE statement (see Section 7.8.39 [REWRITE], page 373) and WRITE statement (see Section 7.8.54 [WRITE], page 410). Even with DB HOME, locking will not work with ORGANIZATION SEQUENTIAL (see Section 5.2.1.1 [ORGANIZATION SEQUEN- TIAL], page 109), ORGANIZATION LINE SEQUENTIAL (see Section 5.2.1.2 [ORGA- NIZATION LINE SEQUENTIAL], page 110) or ORGANIZATION RELATIVE
files with GnuCOBOL builds created for Windows/MinGW. ORGANIZATION INDEXED locks will work with Windows/MinGW + BDB and all locks will work for all file organizations with UNIX GnuCOBOL builds.

OS Use this environment variable to retrieve the value of the Operating System in which the program is running.

The statement ACCEPT identifier-1 FROM ENVIRONMENT "OS" returns the value "Windows_NT" if the program is running in a WINDOWS 10 environment.

PATH The GnuCOBOL "bin" directory should be defined in the PATH.

TMPDIR
TMP
TEMP One of these environment variables must be set to a directory/folder appropriate to create temporary files in. They will be checked in the order shown. This will be used by the SORT statement (see Section 7.8.44 [SORT], page 388) and MERGE statement (see Section 7.8.29 [MERGE], page 351) to create temporary work files. You may also use this folder for any temporary files your application may require.

Also used during execution of programs is runtime.cfg also found in /usr/local/share/gnucobol/config for *nix and this file can also be changed to match your environment if needed. When viewing, note the Default settings.

10.2.3.1 General instructions

The initial runtime.cfg file is found in the $COB_CONFIG_DIR/config (COB_CONFIG_DIR defaults to installdir/gnucobol).
The environment variable COB_RUNTIME_CONFIG may define a different runtime configuration file to read.

If settings are included in the runtime environment file multiple times then the last setting value is used, no warning occurs.

Settings via environment variables always take precedence over settings that are given in runtime configuration files. And the environment is checked after completing processing of the runtime configuration file(s)

All values set to string variables or environment variables are checked for ${envvar} and replacement is done at the time of the setting.

Any environment variable may be set with the directive setenv .
Example: setenv
COB_LIBRARY_PATH ${LD_LIBRARY_PATH}

Any environment variable may be unset with the directive unsetenv (one var per line).
Example: unsetenv COB_LIBRARY_PATH

Runtime configuration files can include other files with the directive include.
Example: include my-runtime-configuration-file

To include another configuration file only if it is present use the directive includeif.
You can also use ${envvar} inside this.
Example: includeif ${HOME}/mygc.cfg

```
If you want to reset a parameter to its default
   value use: reset parametername
```

Most runtime variables have boolean values, some are switches, some have string values, integer values and some are size values.
The boolean values will be evaluated as following:

```
        to true:          1, Y, ON, YES, TRUE     (no matter of case)
        to false:         0, N, OFF
```

A 'size' value is an integer optionally followed by K, M, or G for kilo, mega or giga.

For convenience a parameter in the runtime.cfg file may be defined by using either the environment variable name or the parameter name.
In most cases the environment variable name is the parameter name (in upper case) with the prefix COB_ .

Note: If you want to *slightly* speed up a program's startup time, remove all of the comments from the actual real configuration file that is processed

10.2.3.2 General Environment

```
Environment name:    COB_DISABLE_WARNINGS
   Parameter name:   disable_warnings
          Purpose:   turn off runtime warning messages
             Type:   boolean
          Default:   false
          Example:   DISABLE_WARNINGS    TRUE

Environment name:    COB_ENV_MANGLE
   Parameter name:   env_mangle
          Purpose:   names checked in the environment would get non alphanumeric
                     change to '_'
             Type:   boolean
          Default:   false
          Example:   ENV_MANGLE  TRUE

Environment name:    COB_SET_DEBUG
   Parameter  name:  debugging_mode
          Purpose:   to enable USE ON DEBUGGING procedures that were active
                     during compile-time because of WITH DEBUGGING MODE,
                     otherwise the code generated will be skipped
             Type:   boolean
          Default:   false
          Example:   COB_SET_DEBUG   1

Environment name:    COB_SET_TRACE
   Parameter name:   set_trace
          Purpose:   to enable COBOL trace feature
             Type:   boolean
          Default:   false
```

```
          Example:  SET_TRACE    TRUE

 Environment name:  COB_TRACE_FILE
   Parameter name:  trace_file
          Purpose:  to define where COBOL trace output should go
             Type:  string
          Default:  stderr
          Example:  TRACE_FILE    ${HOME}/mytrace.log

 Environment name:  COB_TRACE_FORMAT
   Parameter name:  trace_format
          Purpose:  to define format of COBOL trace output
             Type:  string
          Default:  "%P %S Line: %L"
                    %P is replaced by Program-Id/Function-Id minimal length 29 with
                       prefix
                    %I is replaced by Program-Id/Function-Id variable length,
                       without prefix
                    %L is replaced by Line number, right justified, length 6
                    %S is replaced by statement type and name
                    %F is replaced by source file name
          Example:  TRACE_FORMAT "Line: %L %S"
             Note:  format of GC2.2 and older:
                    "PROGRAM-ID: %I        Line:  %L              %S"

For v4.0+
 Environment name:  COB_TRACE_IO
   Parameter name:  trace_io
          Purpose:  define if I/O details should be added to trace
             Type:  boolean
          Default:  false
          Example:  TRACE_IO true

 Environment name:  COB_DUMP_FILE
   Parameter name:  dump_file
          Purpose:  to define where COBOL dump output should go
             Note:  The -fdump=all compile option prepares for dump
             Type:  string            : $$ is replaced by process
             id
          Default:  stderr
          Example:  DUMP_FILE  ${HOME}/mytrace.log

 Environment name:  COB_DUMP_WIDTH
   Parameter name:  dump_width
          Purpose:  To define COBOL dump line length
             Type:  integer
          Default:  100
          Example:  dump_width  1 2 0

For v4.0+
 Environment name:  COB_STATS_RECORD
   Parameter name:  stats_record
```

Purpose: Define if I/O statistics should be written
Type: boolean
Default: false
Example: STATS_RECORD true

For v4.0+
Environment name: COB_STATS_FILE
Parameter name: stats_file
Purpose: To define where COBOL I/O statistics should be written The file is appended to
Type: string
Default: stderr
Example: STATS_FILE ${HOME}/mystats.txt

Environment name: COB_CURRENT_DATE
Parameter name: current_date
Purpose: Specify an alternate Date/Time to be returned to ACCEPT clauses this is used for testing purposes or to tweak a missing offset partial setting is allowed
Type: numeric string in format YYYYDDMMHH24MISS or date string
Default: the operating system date is used
Example: COB_CURRENT_DATE "2016/03/16 16:40:52"
current_date YYYYMMDDHHMMSS+01:00

10.2.3.3 Call Environment

Environment name: COB_LIBRARY_PATH
Parameter name: library_path
Purpose: Paths for dynamically-loadable modules
Type: string
Note: The default paths .:/installpath/extras are always added to the given paths
Example: LIBRARY_PATH /opt/myapp/test:/opt/myapp/production

Environment name: COB_PRE_LOAD
Parameter name: pre_load
Purpose: modules that are loaded during startup, can be used to CALL COBOL programs or C functions that are part of a module library
Type: string
Note: The modules listed should NOT include extensions, the runtime will use the right ones on the various platforms, COB_LIBRARY_PATH is used to locate the modules
Example: PRE_LOAD COBOL_function_library:external_c_library

Environment name: COB_LOAD_CASE
Parameter name: load_case
Purpose: Resolve ALL called program names to UPPER or LOWER case Type: Only use UPPER or LOWER
Default: if not set program names in CALL are case sensitive

Example: LOAD_CASE UPPER

Environment name: COB_PHYSICAL_CANCEL
Parameter name: physical_cancel
Purpose: Physically unload a dynamically-loadable module on CANCEL, this frees some RAM and allows the change of modules during run-time but needs more time to resolve CALLs (both to active and not-active programs)
Alias: default_cancel_mode, LOGICAL_CANCELS (0 = yes)
Type: boolean (evaluated for true only)
Default: false
Example: PHYSICAL_CANCEL TRUE

10.2.3.4 File I/O

Environment name: COB_VARSEQ_FORMAT
Parameter name: varseq_format
Purpose: Declare format used for variable length sequential files
- different types and lengths precede each record
- 'length' is the data length, does not include the prefix

Type:
0 means 2 byte record length (big-endian) + 2 NULs
1 means 4 byte record length (big-endian)
2 means 4 byte record length (local machine int)
3 means 2 byte record length (big-endian)

Default: 0
Example: VARSEQ_FORMAT 1

For v4.0+
Environment name: COB_VARREL_FORMAT
Parameter name: varrel_format
Purpose: Declare format to be used for variable length relative files Type:
gc means 'size_t' record length (local machine) precedes maxiumum length data record
mf means file is in Micro Focus format
b32 means Big-Endian 32-bit 'int' record length precedes data b64 means Big-Endian 64-bit 'int' record length precedes data l32 means Little-Endian 32-bit 'int' record length precedes data l64 means Little-Endian 64-bit 'int' record length precedes data
Default: gc
NOTE: 'gc' results in files which cannot be used if copied between machines of different hardware archeticture
Example: VARREL_FORMAT mf

For v4.0+
Environment name: COB_FIXREL_FORMAT
Parameter name: fixrel_format
Purpose: Declare format to be used for fixed length relative files (different types and lengths preceding each record) Type: b32 means 4 byte record length (big-endian)

```
                    l32 means 4 byte record length (little-endian)
                    b64 means 8 byte record length (big-endian) l64
                    means 8 byte record length (little-endian)
                    mf   means  Micro Focus default
                    gc   means  GnuCOBOL default (local 'size_t')
        Default:    gc   fixed  size with no record length prefix
        Example:    FIXREL_FORMAT  B 3 2

For v4.0+
 Environment name:  COB_VARFIX_FORMAT
   Parameter name:  varfix_format
          Purpose:  Declare format to be used for fixed length relative files Type:
                    gc   means 'size_t' record length (local machine) precedes
                         fixed length data record
                   mf  means file is in Micro Focus format
                  b32  means Big-Endian 32-bit 'int' record length precedes data b64
                       means Big-Endian 64-bit 'int' record length precedes data l32
                       means Little-Endian 32-bit 'int' record length precedes data l64
                       means Little-Endian 64-bit 'int' record length precedes data
        Default: gc
           NOTE: 'gc' results in files which cannot be used if copied between
                       machines of different hardware archeticture
        Example: VARFIX_FORMAT mf

 Environment name:  COB_FILE_PATH
   Parameter name:  file_path
          Purpose:  Define default location where data files are stored
             Type:  file path directory
          Default:  .   (current directory)
          Example:  FILE_PATH ${HOME}/mydata

 Environment name:  COB_LS_FIXED
   Parameter name:  ls_fixed
          Purpose:  Defines if LINE SEQUENTIAL files should be fixed length
                    (or variable, by removing trailing spaces)
            Alias:  STRIP_TRAILING_SPACES  (0 = yes)
             Type:  boolean
          Default:  false Example:
                    LS_FIXED TRUE

 Environment name:  COB_LS_NULLS
   Parameter name:  ls_nulls
          Purpose:  Defines for LINE SEQUENTIAL files what to do with data which is
                    not DISPLAY type.                This could happen if a LINE
                    SEQUENTIAL record has BINARY/COMP data fields in it.
                    For v4.0+ (may change before its release candidate)
                    This option is only for GnuCOBOL format files
             Type:  boolean
          Default:  false
             Note:  The TRUE setting will insert a null character x"00" before
                    those values to escape them, and redo on read-in.
```

```
          Example:  LS_NULL = TRUE

For v4.0+
 Environment name:  COB_LS_SPLIT
   Parameter name:  ls_split
          Purpose:  Defines for LINE SEQUENTIAL files what to do when a
                    record is longer than the program handles. If
                    'ls_split=true' then the data is returned as multiple
                    records
             Type:  boolean
          Default:  false
                    The record is truncated and the file skips to the next LF
          Example:  LS_SPLIT = TRUE

For v4.0+
 Environment name:  COB_LS_VALIDATE
   Parameter name:  ls_validate
          Purpose:  Defines for LINE SEQUENTIAL files that the data should be
                    validated. If any record has non-DISPLAY characters then an
                    error status of 34 is returned
                    For v4.0+ (may change before its release candidate)
                    This option is only for GnuCOBOL format files
             Type:  boolean
          Default:  true
             Note:  The TRUE setting does data validation
                    The FALSE setting lets non-DISPLAY characters be written If
                    LS_NULLS is set, then LS_VALIDATE is not checked
          Example:  LS_VALIDATE = FALSE

For v4.0+ (may be replaced before its release candidate)
 Environment name: COB_MF_FILES
   Parameter name: mf_files
          Purpose:  Declares that all files in the program should be in
                    Micro Focus compatible format.
             Type:  boolean (evaluated for true only)
          Default:  false
          Example:  mf_files  True

For v4.0+ (may be replaced before its release candidate)
 Environment name:  COB_MF_LS_NULLS
   Parameter name:  mf_ls_nulls
          Purpose:  Defines for Micro Focus compatible LINE SEQUENTIAL files
                    what to do with data which is not DISPLAY type.
                    This could happen if a LINE SEQUENTIAL record has
                    BINARY/COMP data fields in it.
             Type:  boolean
          Default:  true
             Note:  The  TRUE setting will handle files that contain COMP data in a similar
                    manner to the method used by Micro Focus COBOL
          Example:  MF_LS_NULLS  = TRUE

For v4.0+ (may be replaced before its release candidate)
```

Environment name: COB_MF_LS_INSTAB
Parameter name: mf_ls_instab
Purpose: Defines for LINE SEQUENTIAL files that multiple spaces should be replaced by a TAB character, assuming a 'tab set' value of 8. Each TAB means to skip to the next column that is a multiple of 8
Similar to Micro Focus INSERTTAB=ON option
Type: boolean
Default: false
Example: MF_LS_INSTAB = TRUE

For v4.0+ (may be replaced before its release candidate)
Environment name: COB_MF_LS_SPLIT
Parameter name: mf_ls_split
Purpose: Defines for Micro Focus compatible LINE SEQUENTIAL files what to do when a record is longer than the program handles.
If 'mf_ls_split=true' then
the data is returned as multiple records
Type: boolean
Default: true
Example: MF_LS_SPLIT = FALSE

For v4.0+ (may be replaced before its release candidate)
Environment name: COB_MF_LS_VALIDATE
Parameter name: mf_ls_validate
Purpose: Defines for Micro Focus compatible LINE SEQUENTIAL files that the data should be validated.
If any record has non-DISPLAY
characters then an error status of 34
is returned
Type: boolean
Default: false
Note: The TRUE setting does data validation
The FALSE setting lets non-DISPLAY characters be written If MF_LS_NULLS is set, then MF_LS_VALIDATE is not checked
Example: MF_LS_VALIDATE = FALSE

For v4.0+
Environment name: COB_SHARE_MODE
Parameter name: share_mode
Purpose: Defines what file sharing option should be used
Type: -- choice of values ---
none - nothing overrides application code
read - files opened as SHARE READ ONLY all
- files opened as SHARE ALL OTHERS no
- files opened as SHARE NO OTHERS
Default: none
Example: share_mode = ALL

For v4.0+
Environment name: COB_RETRY_MODE
Parameter name: retry_mode

```
          Purpose:  Defines what I/O retry sharing option should be used
             Type:  --- choice of values ---
                    none     - nothing overrides application
                    code never     - I/O is never retried
                    forever - I/O will be retried until success
          Default:  none
          Example:  retry_mode = never

For v4.0+
 Environment name:  COB_RETRY_TIMES
   Parameter name:  retry_times
          Purpose:  Defines how many times I/O should be retried
             Type:  integer
          Default:  0
          Example:  retry_times = 10

For v4.0+
 Environment name:  COB_RETRY_SECONDS
   Parameter name:  retry_seconds
          Purpose:  Defines how many seconds I/O should be retried
             Type:  integer
          Default:  0
          Example:  retry_seconds = 6

For v4.0+
 Environment name:  COB_KEYCHECK
   Parameter name:  keycheck
          Purpose:  Must INDEXED file keys match COBOL SELECT exactly
             Type:  boolean
          Default:  true
          Example:  keycheck = off

 Environment name:  COB_SYNC
   Parameter name:  sync
          Purpose:  Should the file be synced to disk after each write/update
             Type:  boolean
          Default:  false
          Example:  SYNC: TRUE

 Environment name:  COB_SORT_MEMORY
   Parameter name:  sort_memory
          Purpose:  Defines how much RAM to assign for sorting data if this size
                    is exceeded the                    SORT   will be
                    done on disk instead of memory
             Type:  size     but must be more than 1M
          Default:  128M
          Example:  SORT_MEMORY 64M

 Environment name:  COB_SORT_CHUNK
   Parameter name:  sort_chunk
          Purpose:  Defines how much RAM to assign for sorting data in chunks
```

Type: size but must be within 128K and 16M
Default: 256K
Example: SORT_CHUNK 1M

10.2.3.5 Screen I/O

Environment name: COB_BELL
Parameter name: bell
Purpose: Defines how a request for the screen to beep is handled Type: FLASH, SPEAKER, FALSE, BEEP
Default: BEEP
Example: BELL SPEAKER

Environment name: COB_HIDE_CURSOR
Parameter name: hide_cursor
Purpose: hide the cursor; 0=visible, 1=hidden
Type: boolean
Default: false
Example: hide_cursor Y

Environment name: COB_REDIRECT_DISPLAY
Parameter name: redirect_display
Purpose: Defines if DISPLAY output should be sent to 'stderr'
Type: boolean
Default: false
Example: redirect_display Yes

Environment name: COB_SCREEN_ESC
Parameter name: screen_esc
Purpose: Enable handling of ESC key during ACCEPT
Type: boolean
Default: false
Note: is only evaluated if COB_SCREEN_EXCEPTIONS is active Example: screen_esc Yes

Environment name: COB_SCREEN_EXCEPTIONS
Parameter name: screen_exceptions
Purpose: Enable exceptions for function keys during ACCEPT
Type: boolean
Default: false
Example: screen_exceptions Yes

Environment name: COB_TIMEOUT_SCALE
Parameter name: timeout_scale
Purpose: Specify translation in milliseconds for ACCEPT clauses BEFORE TIME value / AFTER TIMEOUT
Type: integer
0 means 1000 (Micro Focus COBOL compatible), 1 means 100 (ACUCOBOL compatible), 2 means 10, 3 means 1

```
          Default:  0
          Example:  timeout_scale  3

Environment name:   COB_INSERT_MODE
   Parameter name:  insert_mode
          Purpose:  Specify default insert mode for ACCEPT; 0=off, 1=on
             Type:  boolean
          Default:  false
             Note:  also sets the cursor type (if available)
          Example:  insert_mode Y

Environment name:   COB_MOUSE_FLAGS
   Parameter name:  mouse_flags
          Purpose:  Specify which mouse events will be sent as function key to
                    the application during ACCEPT and how they will be
                    handled
             Type:  int (by bits)
          Default:  1
             Note:  0 disables the mouse cursor, any other value enables it, any
                    value containing 1 will enable internal handling (click to
                    position, double-click to enter).
                    See copy/screenio.cpy for list of events and their values.
            Alias:  MOUSE_FLAGS
          Example:  11 (enable internal handling => 1, left press => 2,
                        double-click => 8; 1+2+8=11)

Environment name:   COB_MOUSE_INTERVAL
   Parameter name:  mouse_interval
          Purpose:  Specifies the maximum time (in thousands of a second) that
                    can elapse between press and release events for them to be
                    recognized as a click.
             Type:  int (0 - 166)
          Default:  100
             Note:  0 disables the click resolution (instead press + release are
                    recognized), also disables positioning by mouse click

Environment name:   COB_DISPLAY_PRINT_PIPE
   Parameter name:  display_print_pipe
          Purpose:  Defines command line used for sending output of
                    DISPLAY UPON PRINTER to (via pipe)
                    This is very similar to Micro Focus COBPRINTER
             Note:  Each executed DISPLAY UPON PRINTER statement causes a
                    new invocation of command-line (= new process start).
                    Each invocation receives the data referenced in the
                    DISPLAY statement and is followed by an
                    end-of-file condition.
                    COB_DISPLAY_PRINT_FILE, if set, takes precedence
                    over COB_DISPLAY_PRINT_PIPE.
            Alias:  COBPRINTER
             Type:  string
          Default:  not set
```

Example: print 'cat >>/tmp/myprt.log'

Environment name: COB_DISPLAY_PRINT_FILE
Parameter name: display_print_file
Purpose: Defines file to be appended to by DISPLAY UPON PRINTER
Note: Each DISPLAY UPON PRINTER opens, appends and closes the file.
Type: string : $$ is replaced by process id
Default: not set
Example: display_printer '/tmp/myprt.log'

Environment name: COB_DISPLAY_PUNCH_FILE
Parameter name: display_punch_file
Purpose: Defines file to be created on first DISPLAY UPON SYSPUNCH/SYSPCH
Note: The file will be only be closed on runtime exit.
Type: string : $$ is replaced by process id
Default: not set
Example: display_punch './punch_$$.out'

Environment name: COB_LEGACY
Parameter name: legacy
Purpose: keep behaviour of former runtime versions, currently only for setting screen attributes for non input fields
Type: boolean
Default: not set
Example: legacy true

Environment name: COB_EXIT_WAIT
Parameter name: exit_wait
Purpose: to wait on main program exit if an extended screenio DISPLAY was issued without an ACCEPT following
Type: boolean
Default: true
Example: COB_EXIT_WAIT o f f

Environment name: COB_EXIT_MSG
Parameter name: exit_msg
Purpose: string to display if COB_EXIT_WAIT is processed, set to '@w{}' if no actual display but an ACCEPT should be done
Type: string
Default: 'end of program, please press a key to exit' (localized) Example: COB_EXIT_MSG '@w{}'

10.2.3.6 Report I/O

Environment name: COB_COL_JUST_LRC
Parameter name: col_just_lrc
Purpose: If true, then COLUMN defined as LEFT, RIGHT or CENTER will have the data justified within the field limits

If false, then the data is just copied into the column as is Type: boolean
Default: TRUE
Example: col_just_lrc True

10.2.3.7 File I/O Environment Variables and/or dictionary file

GnuCOBOL 4.+ only!

Before a file is opened a check is done for environment variables that may define various attributes of the file
First a check is made for attributes for files of the same ORGANIZATION IX_OPTIONS for INDEXED, SQ_OPTIONS for SEQUENTIAL, RL_OPTIONS for RELATIVE LS_OPTIONS for LINE SEQUENTIAL, LA_OPTIONS for LINE ADVANCING SEQUENTIAL
If none of these are present, it then checks for IO_OPTIONS

Then an additional check is done for IO_asgnmame where 'asgname' was the ASSIGN EXTERNAL name used in the program

The environment variable (or dictionary file) may contain any of the following keywords, separated by spaces and/or commas

You can specify just the keyword and it is assumed to mean set to true, or no-keyword (or no_keyword or nokeyword) which means set to false, or keyword=true or keyword=false. The valid keywords are:

Keyword	Meaning
type=xx	Set file organization where 'xx' is one of IX = INDEXED, SQ = SEQUENTIAL, RL = RELATIVE LS = LINE SEQUENTIAL, LA = LINE ADVANCING
mf	Set file to Micro Focus compatible format
gc	Set to original GnuCOBOL default format
recsz	The size for fixed size record file
maxsz	Maximum record size for variable length records minsz Minimum record size for variable length records
ls_nulls	Do NUL insertion before characters less than a SPACE, Default: false
ls_validate	Validate data for LINE Sequential Files, Default: true crlf Lines end with CR LF (Windows format for text files) lf Lines end with LF (Unix format for text files)
sync	Sync all writes to disk
B32	Use 32-bit Big-Endian format 'int' as record length L32 Use 32-bit Little-Endian format 'int' as record length
B64	Use 64-bit Big-Endian format 'int' or 'size_t' as record length L64 Use 64-bit Little-Endian format 'int' or 'size_t' as record length trace Enable I/O trace when program execution tracing is enabled
stats	Write I/O statistic information on file close
retry_times	Default number of times to retry I/O
retry_seconds	Number of seconds between I/O retry attempts

```
retry_forever      Retry I/O forever
retry_never        Never retry I/O operations
ignore_lock        Ignore record locks
advancing_lock     Advance to the next record if lock condition
share_all          Share file with ALL others
share_read         Share file for READ only
share_no           Share file with NO others

 ---- For INDEXED files -----
format=ixhandler INDEXED file format: CISAM,DISAM,VBISAM,BDB,LMDB
format=auto        INDEXED file format is determined by inspecting the file
nkeys=n            number of indexes
key1=(loc:len)     loc (zero relative) of key, len of key
key2=(loc:len,loc:len ...)
                   define composite index
dupn=Y             index allows dups

  ---- For INDEXED BDB files -----
big_endian         Set internal 'int' byte order to BIG ENDIAN
little_endian      Set internal 'int' byte order to LITTLE ENDIAN
```

10.2.4 Program Arguments

Regardless of the manner in which a main program is executed (i.e. directly or via cobcrun), any arguments specified to the program may be retrieved via any of the following:

- ACCEPT FROM COMMAND-LINE (see Section 7.8.1.2 [ACCEPT FROM COMMAND-LINE], page 264)
- PROCEDURE DIVISION CHAINING (see Section 7.2 [PROCEDURE DIVISION CHAINING], page 246)

10.3 Binary Truncation

By default, the GnuCOBOL compiler will truncate binary data items to the precision indicated by their PICTURE (see Section 6.9.37 [PICTURE], page 195) clause, if they have one. This applies to COMP, BINARY and COMP-4 items Only.
The fact is, however, that binary truncation has a significant effect on the performance of GnuCOBOL programs. When binary truncation is in effect, arithmetic operations performed against all types of numeric data items (even USAGE DISPLAY) are slowed down.

Before continuing, it's worth making the point that we're not talking about astronomical performance degradations here. Today's computers are fast, and a user sitting at the keyboard, running a GnuCOBOL program is unlikely to notice. But, if you have a GnuCOBOL program that has to process large amounts of data, performing some significant "number crunching" against that data as it goes, the impact of truncation could become noticeable.

The following program compares the performance of performing arithmetic operations (in a totally non-scientific, non-rigorous way) against data items with a USAGE (see Section 6.9.59 [USAGE], page 228) of DISPLAY, COMP and BINARY-LONG. It was actually my intent when I first wrote the program to merely demonstrate the relative performance differences between different types of numeric data storage, and it certainly met that objective.

```
      IDENTIFICATION DIVISION.
```

```
PROGRAM-ID.
DEMOMATH.
DATA DIVISION.
WORKING-STORAGE SECTION.
01   Begin-Time.
     05  BT-HH                          PIC  9(2).
     05  BT-MM                          PIC  9(2).
     05  BT-SS                          PIC  9(2).
     05  BT-HU                          PIC  9(2).
01   Binary-Item            BINARY-LONG  SIGNED  VALUE  0.
01   Comp-Item               COMP     PIC  S9(9)   VALUE  0.
01   Display-Item            DISPLAY PIC  S9(9)   VALUE  0.
01   End-Time.
     05 ET-HH                          PIC  9(2).
     05 ET-MM                          PIC  9(2).
     05 ET-SS                          PIC  9(2).
     05 ET-HU                          PIC  9(2).
78   Repeat-Count                      VALUE 10000000.
01   Time-Diff                         PIC ZZ9.99.
PROCEDURE DIVISION.
010-Test-Usage-DISPLAY.
     ACCEPT    Begin-Time   FROM TIME
     PERFORM Repeat-Count TIMES
          ADD 7 TO Display-Item
     END-PERFORM
     PERFORM   100-Determine-Time-Diff
     DISPLAY 'USAGE DISPLAY: ' Time-Diff ' SECONDS'
     -
020-Test-Usage-COMP.
     ACCEPT Begin-Time FROM TIME
     PERFORM Repeat-Count TIMES
          ADD 7 TO Comp-Item
     END-PERFORM
     PERFORM   100-Determine-Time-Diff
     DISPLAY 'USAGE COMP:      ' Time-Diff ' SECONDS'
     -
040-Test-Usage-BINARY.
     ACCEPT Begin-Time FROM TIME
     PERFORM Repeat-Count TIMES
          ADD 7 TO Binary-Item
     END-PERFORM
     PERFORM   100-Determine-Time-Diff
     DISPLAY 'USAGE BINARY:    ' Time-Diff ' SECONDS'
     -
099-Done.
     STOP RUN
     -
100-Determine-Time-Diff.
     ACCEPT End-Time FROM TIME
     COMPUTE Time-Diff =
       ( (ET-HH * 360000 + ET-MM * 6000 + ET-SS * 100 + ET-HU)
       - (BT-HH * 360000 + BT-MM * 6000 + BT-SS * 100 + BT-HU) )
```

```
/ 100
-
```

Each data item has 7 added to it ten million times.

The time (to one-one-hundredth of a second) will be retrieved before and after each test and the difference between the two is displayed. This is why the computations were done so many times — it was to make sure the timing was measurable with only a 1/100 second "stopwatch".

I also ran the tests multiple times, just to make sure I had consistent results (I did). Like I mentioned earlier, this is not a rigorous, scientific benchmark of numeric performance; it's just a quick-and-dirty comparison.

Here are the results:

```
Test 1:
USAGE DISPLAY: 1.72 SECONDS
USAGE COMP:0.62 SECONDS USAGE
                    BINARY:
0.02 SECONDS

Test 2:
USAGE DISPLAY: 1.69 SECONDS
USAGE COMP:0.61 SECONDS USAGE
                    BINARY:
0.02 SECONDS

Test 3:
USAGE DISPLAY: 1.69 SECONDS
USAGE COMP:0.65 SECONDS USAGE
                    BINARY:
0.02 SECONDS
```

The results I saw here were consistent with those that would have been obtained from most of the COBOL implementations I have ever worked with — USAGE COMP has a significant performance advantage over USAGE DISPLAY and USAGE BINARY-LONG (and presumably the other BINARY-xxx usages as well) perform identically, within the measurement tolerances of the test.

Imagine my surprise, however, when I discovered that the use of -fnotrunc switch also made a difference:

```
Test 4:
USAGE DISPLAY: 1.72 SECONDS
USAGE COMP:0.07 SECONDS USAGE
                    BINARY:
0.02 SECONDS

Test 5:
USAGE DISPLAY: 1.72 SECONDS
USAGE COMP:0.07 SECONDS USAGE
                    BINARY:
0.02 SECONDS

Test 6:
USAGE DISPLAY: 1.73 SECONDS
```

```
USAGE COMP:0.06 SECONDS USAGE
                        BINARY:
0.02 SECONDS
```

As you can see, there was a huge drop in USAGE COMP timings by turning off truncation. As

a result, I see absolutely no reason whatsoever why the -fnotrunc switch option shouldn't be used on all GnuCOBOL compilations.

If you want to squeeze every last bit of performance out of your GnuCOBOL programs, don't forget to investigate the -O switch, -O2 switch and the -Os switch, all of which influence the optimization of compiled code. Actually run programs using various optimization switches (or not) and compare execution times against those of unoptimized compiled versions of your programs. Don't just compare the generated C code because sometimes the differences can't be seen at the C source-code level.

```
Test 7:
cobc -x demomath.cbl -O2;demomath
USAGE DISPLAY:      1.68 SECONDS
USAGE COMP:         0.60 SECONDS
USAGE BINARY:       0.00 SECONDS

Test 8:
cobc -x demomath.cbl -fnotrunc -O2;demomath
USAGE DISPLAY:      1.67 SECONDS
USAGE COMP:0.01      SECONDS
USAGE BINARY:       0.00
S E C O N D S
```

All tests above carried out under Linux with a AMD FX8350 under very low loading prior to the test. I would have also tried on a i7-7700 but that is under Windows 10 and I do not have a GC version on it - Vince.

————————————————————

End of Chapter 10 — Interfacing With The OS

11 Sub-Programming

11.1 Subprogram Types

Simply stated, a Subprogram is a program that is invoked by another program; the subprogram performs whatever its designed operations are and — when complete — typically returns control back to the program that invoked it. There are two different types of subprograms supported by GnuCOBOL, subroutines and user-defined functions. The distinction between these two subprogram types lies in the manner in which they are executed.

When program A invokes subprogram B as a Subroutine, it does so using a special statement dedicated to that function (the CALL statement (see Section 7.8.5 [CALL], page 290), just as if B were one of the built-in system subroutines.

When program A invokes program B as a User-Defined Function, it does so in a manner identical to how B would have been invoked had it been one of the many built-in intrinsic functions.

In either instance, program A is referred to as the Calling Program while program B is known as the Called Program. GnuCOBOL programs may be a calling program, a called program or both.

A program written in the C programming language may serve as either the calling or called program too. A called program may act as a calling program to another called program. When a calling program does not serve as a called program to any program, that calling program is known as a Main Program.

Both subroutines and user-defined functions may return a value. The value they return must be an integer in the range -2147483648 to +2147483647. This value will be available in the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260) and also as the value of the data item specified on the RETURNING (see Section 7.8.5 [CALL], page 290) clause of a subroutine's CALL.

11.2 Independent vs Contained vs Nested Subprograms

Subprograms (either subroutines or user-defined functions) can be implemented in three different ways.

Independent Subprograms

These are subprograms that are coded as the only COBOL program in their Compilation Unit (see [Compilation Unit], page 683).

Contained Subprograms

These are subprograms which occur in the same Compilation Unit as a main program and/or other subprograms. Each contained subprogram is separated from the next via an END PROGRAM marker line. As an example. . .

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUB1.
- - -
END PROGRAM SUB1.
IDENTIFICATION DIVISION.
PROGRAM-ID. SUB2.
- - -
END PROGRAM SUB2.
```

Program source code may be concatenated as shown here, provided an END PROGRAM marker naming the PROGRAM-ID of the just-completed program is used to separate one program from another.

There's no reason that user-defined functions cannot be included too — they'll just have FUNCTION-IDs and will be ended by END FUNCTION markers.

The last program in any GnuCOBOL source file need not have an END marker. When multiple programs occur in a source file, it is assumed that the programs are related to one another in that they will be CALLed or executed as functions from the others.

Nested Subprograms

It is also possible to create source files where GnuCOBOL programs are nested inside each other. Take for example these four GnuCOBOL programs:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
---
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG2.
---
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG3.
---
END PROGRAM
PROG3.
END PROGRAM
PROG2.
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG4.
---
END PROGRAM
PROG4.
END PROGRAM
PROG1.
```

Here we see that PROG2 is nested inside of PROG1 because there is no END PROGRAM marker separating them. This means that data items or files defined within PROG1 can be used within PROG2 simply by attaching the GLOBAL (see Section 6.9.24 [GLOBAL], page 178) attribute to them back in PROG1 when they are defined.

Similarly, since there is no END PROGRAM marker separating PROG3 from PROG2, it is possible for PROG3 to access GLOBAL files and data items defined within PROG2. Since PROG2 is nested within PROG1, any GLOBAL resources defined within PROG1 will be available to PROG3 as well.

The two END PROGRAM markers for PROG3 and PROG2 (note their sequence) mean that PROG4 is nested within PROG1 only. It will not have access to any GLOBAL resources defined within either PROG2 or PROG3.

The END PROGRAM PROG1. marker, since it is the last line in the source file, is entirely optional.

11.3 Alternate Entry Points

Any subroutine may have multiple entry-points defined within it. This means the subroutine could be called either via a CALL 'program-id' or a CALL 'entry-point' statement. There may be any number of alternate entry-points defined within a subroutine.

Alternate entry-points provide multiple ways in which the same subroutine may be called; presumably, each entry-point will provide some different functionality to the calling program. For

example, if you wished to write a subroutine that manipulates "student" records in a database, you might have the primary entry-point name retrieve a student record from the database, while the alternate entry points Add-Student, Update-Student and Delete-Student could provide the alternate functions implied by their entry-point names.

The alternative to using multiple entry points in your subroutine, by the way, would be to include an additional argument to the primary (and only) entry point of the subroutine; this new argument might be named STUDENT-FUNCTION and might have values of 'FETCH', 'ADD', 'UPDATE' or 'DELETE'.

The primary entry-point for any subroutine is always the first executable statement following any DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250) in the procedure division. The name of that entry-point (the name that will be called) is the subroutine's PROGRAM-ID (see Chapter 4 [IDENTIFICATION DIVISION], page 81).

An alternate entry point is added to a subroutine using the ENTRY statement (see Section 7.8.14 [ENTRY], page 319).

When an alternate entry-point is called, execution within the subroutine will begin at the first executable statement following the ENTRY statement.

11.4 Dynamic vs Static Subprograms

Any subprogram may be either statically or dynamically loaded into memory.

A Static Subprogram is one which was in the same Compilation Unit (see [Compilation Unit], page 683) as the other program(s) which call it, therefore meaning that its executable object code is part of the same executable file as its calling program. The static subprogram was therefore loaded into memory as part of and at the same time as the calling program.

A Dynamic Subprogram is one whose executable object code exists as an executable file separate from that containing the calling program; these two programs were therefore each com- piled in their own separate Compilation Group (see [Compilation Group], page 683). Dynamic subprograms are located and loaded into memory the first time they are executed. Dynamic subprograms may be unloaded from memory via the CANCEL statement (see Section 7.8.6 [CAN- CEL], page 294), if desired.

GnuCOBOL subprograms may be created as either static or dynamic subprograms, as desired by the programmer.

To demonstrate, assume that a GnuCOBOL Main Program (whose code resides in the file M.cbl) will be calling three subprograms, named A, B and C (these are the PROGRAM-IDs of the three subprograms, and their source code may be found in the files A.cbl, B.cbl and C.cbl, respectively.

Here is how these four programs would be compiled if the three subprograms are to be static:

```
cobc -x M.cbl A.cbl B.cbl C.cbl
```

This command informs the compiler (cobc) that four programs are to be compiled (the first named on the command must always be the main program), and a single executable file is to be created (due to the -x switch).

Here is how the main program and the three subprograms could be compiled if the three subprograms are to be dynamic:

```
cobc -x M.cbl
cobc -m A.cbl B.cbl C.cbl
```

These commands will create an executable file for the main program (-x switch) and three separate dynamically-loadable libraries (see -m switch), one for each of the three subprograms.

Had we wished, we could have created a single dynamically-loadable library containing all three subprograms by adding the -b switch to their compilation:

```
cobc -m -b A.cbl B.cbl C.cbl
```

Dynamically-loadable libraries are also known by the term dynamically-loadable modules. The two terms are synonymous.

Here are the rules about GnuCOBOL dynamically-loadable modules:

1. There may be multiple GnuCOBOL subprograms contained within a single dynamically-loadable library if the -b switch is used in addition to -m. If not, each subprogram will be compiled to a separate dynamically-loadable library.
2. Dynamically-loadable modules will be named xxxxxxxx.dll on a Windows system, xxxxxxxx.so on a Unix system or xxxxxxxx.dylib on an OSX system, where xxxxxxxx exactly matches, including the usage of upper- and lower-case letters, the primary entry-point name (PROGRAM-ID or FUNCTION-ID) or an alternate entry point name defined via the ENTRY statement (see Section 7.8.14 [ENTRY], page 319) of any one of the GnuCOBOL programs included in that module.
3. The first time any of the GnuCOBOL subprograms in a dynamically-loadable module are invoked, the entry-point referenced must be the one for which the .dll, .so or .dylib file is named.
4. When a dynamically-loadable module needs to be loaded (because it is not already in memory from a previous subprogram execution), the dynamically-loadable library will be sought in the same directory from which the main program was loaded. If it cannot be found there, each directory named in the PATH run-time environment variable (see Section 10.2.3 [Run Time Environment Variables], page 700) will be searched. If it was not located in any of those directories, the library specified by the COB_LIBRARY_PATH run-time environment variable will be searched. Finally, if it still cannot be located, execution will be terminated with an error message (libcob: Cannot find module 'xxxxxxxx').
5. Once the dynamically-loadable module has been successfully loaded, any of the entry-points contained within it are now available for reference.
6. Dynamically-loadable modules may be removed from memory via the CANCEL statement (see Section 7.8.6 [CANCEL], page 294).
7. Once a dynamically-loadable module is actually loaded into memory, even if it is subsequently unloaded (via the CANCEL statement), its list of entry-points remain available to the GnuCOBOL run-time library and subsequent re-executions of any of those entry points will be able to bypass the search (rule #4) as well as the first-execution rule (rule #3).

Consult the documentation on the COB_PRE_LOAD run-time environment variable, COB_PHYSICAL_CANCEL run-time environment variable and COB_LOAD_CASE run-time environment variable run-time environment variables (see Section 10.2.3 [Run Time Environment Variables], page 700) for additional options when using dynamically-loadable modules.

11.5 Subprogram Execution Flow

When a subprogram is invoked, the flow of execution will differ slightly depending on whether the subprogram is a subroutine or a user-defined function.

11.5.1 Subroutine Execution Flow

When a subroutine is CALLed:

1. The calling program issues a statement of the form CALL 'entry-point' USING ... to transfer control to the subroutine.

2. The executable for the called program will be located and loaded into memory:
   A. If it is a static subroutine, it will already be part of the executable program issuing the CALL (see Section 7.8.5 [CALL], page 290).
   B. If it is a dynamic subroutine, the GnuCOBOL run-time system will check to see if a dynamically-loadable module containing the subprogram's entry point was al- ready located. If it was, no further "location" activity is needed. If not, the dynamically-loadable module will be located (see [Locating Dynamically-Loadable Modules], page 722).
   C. Once the module has been located (if location was needed), it will be loaded into memory (if not already loaded).
3. Execution of the calling program is suspended and control will transfer to the called program, as follows:
   A. If the PROGRAM-ID (see Chapter 4 [IDENTIFICATION DIVISION], page 81) clause of the subprogram included the INITIAL clause, the program will be reinitialized back to its compile-time state. This will happen regardless of the INITIAL clause the first time the subprogram is executed.
   B. Local-storage, if any, will be allocated and initialized.
   C. Execution will begin at the first executable statement following the subprograms entry-point. The entry point will be either the first executable statement following any DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250) that might be present (if the subprogram was invoked using its primary entry-point name) or the first executable statement following the ENTRY statement (see Section 7.8.14 [ENTRY], page 319) nam- ing the entry-point specified on the CALL if the subprogram was invoked using an alternate entry point.
4. The flow of execution will then progress through the coding of the subprogram as it would with any other program.
5. If the subprogram issues a STOP statement (see Section 7.8.46 [STOP], page 395) with the RUN option, program execution ceases and control returns to the operating system or whatever execution shell invoked the main program.
6. If the subprogram wishes to return control back to the calling program, it will do so using either the GOBACK statement (see Section 7.8.21 [GOBACK], page 334) or the EXIT PROGRAM statement (see Section 7.8.18 [EXIT], page 329). At this time:
   A. If the subprograms procedure division header or ENTRY statement included a RETURNING, the value of the data item found on that clause is moved to the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260); this behaviour can be altered utilizing the CALL-CONVENTION (see Section 5.1.3 [SPECIAL-NAMES], page 90) feature to leave RETURN-CODE unchanged.
   B. Local-storage, if any, is de-allocated.
   C. If the calling program included a RETURNING clause on the CALL statement that invoked the subprogram, the value of the RETURNING data item in the subroutine is moved to that data item. If there was no RETURNING specified in the subroutine, the value of the RETURN-CODE special register is moved to that data item.
   D. Execution will resume back in the calling program with the first executable statement following the CALL that invoked the subprogram.

11.5.2 User-Defined Function Execution Flow

When a user-defined function is executed:

1. The object code for the called program (the user-defined function) will be located, as follows:
   A. If it is a static user-defined function, it will already be part of the executable file containing the calling program.
   B. If it is a dynamic user-defined function, the GnuCOBOL run-time system will check to see if a dynamically-loadable module containing the function's entry point was already located. If it was, no further "location" activity is needed. If not, the dynamically-loadable module will be located (see [Locating Dynamically-Loadable Modules], page 722).
   C. Once the module has been located (if location was needed), it will be loaded into memory (if not already loaded).
2. Execution of the calling program is suspended and control will transfer to the called program, as follows:
   A. Local-storage, if any, will be allocated and initialized.
   B. Execution will begin with the first executable statement in the procedure division following any DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250) that might be present.
3. The flow of execution will then progress through the coding of the function as it would with any other program.
4. If the function issues a STOP statement (see Section 7.8.46 [STOP], page 395) with the RUN option, program execution ceases and control returns to the operating system or whatever execution shell invoked the main program.
5. If the function wishes to return control back to the calling program, it will do so using either the GOBACK statement (see Section 7.8.21 [GOBACK], page 334) or the EXIT FUNCTION statement (see Section 7.8.18 [EXIT], page 329). At this time:
   A. The value of the data item found on the user-defined functions PROCEDURE DIVISION RETURNING (see Section 7.3 [PROCEDURE DIVISION RETURNING], page 248) clause is moved to the RETURN-CODE special register (see Section 7.7 [Special Registers], page 260).
   B. Local-storage, if any, is de-allocated.
   C. Execution will resume back in the calling program at the point where the returned value of the function is needed. At that point, the value in the RETURN-CODE special register will be used for the function's value.

11.6 Sharing Data Between Calling and Called Programs

11.6.1 Subprogram Arguments

11.6.1.1 Calling Program Considerations

Data items defined in a calling program may be passed to either type of called program (subroutine or user-defined function) as arguments.

Arguments must be described in both the calling and called programs, and while they don't need to have the same names in both programs, they should be described in an identical manner with regard to the following characteristics:

- PICTURE (see Section 6.9.37 [PICTURE], page 195) (including both type and length)
- SIGN (see Section 8.1.85 [SIGN], page 517)
- SYNCHRONIZED (see Section 6.9.52 [SYNCHRONIZED], page 217)
- USAGE (see Section 6.9.59 [USAGE], page 228)

A subroutine may be passed a maximum of 251 arguments; if you build the GnuCOBOL software yourself from the distributed source, you CAN change this value by altering the defined value of COB_MAX_FIELD_PARAMS in the call.h header file but also see 7.8.5.11 for more information. There is no built-in GnuCOBOL limit to how many arguments a user-defined function may be passed.

Whether or not changes made to an argument within a subroutine will be "visible" to the calling program depends on how the argument was passed. There are three ways in which arguments may be passed from a calling program to a subroutine, as defined by the use of optional BY clauses in the CALL (see Section 7.8.5 [CALL], page 290) statement's list of arguments.

As an example, the following statement passes three arguments to a subroutine — each argument is passed differently.

```
CALL "subroutine" USING BY REFERENCE arg-1
                        BY CONTENT arg-2
                        BY VALUE arg-3
END-CALL
```

The three ways arguments are passed are as follows.

BY REFERENCE
: When a subroutine argument is passed BY REFERENCE, the subroutine is passed the address of the actual data item being passed as an argument. The item may be anything defined within the data division of the program. If the subroutine modifies the contents of this argument, the calling program will "see" the results of that change when the subroutine returns control. This is the default manner in which GnuCOBOL passes arguments to a subroutine, should no BY clauses be included on the CALL.

BY CONTENT
: When a subroutine is passed an argument BY CONTENT, the subroutine is passed the address of a copy of the actual data being passed as an argument. The item may be anything defined within the data division of the program. The copy is made each time the CALL statement is executed, immediately before the CALL actually takes place. If the subroutine modifies the contents of this argument, it will be the copy that is modified, not the original data item; the calling program will therefore not "see" the results of that change when the subroutine returns control.

BY VALUE

Passing a subroutine argument BY VALUE passes the actual value of the data being passed as an argument. The item may be any elementary binary numeric item defined within the data division of the program. If the subroutine modifies the contents of this argument, the calling program will not "see" the results of that change when the subroutine returns control.

The first two ways in which arguments may be passed (BY REFERENCE and BY CONTENT) are intended for use when a GnuCOBOL program is being called, while the first and third (BY REFERENCE and BY VALUE) are intended for use when a C program is being called. You can use BY VALUE arguments when calling GnuCOBOL subroutines, but remember that those arguments are limited to being a numeric binary data item.

Arguments to user-defined functions are always passed BY REFERENCE.

11.6.1.2 Called Program Considerations

When coding a GnuCOBOL subprogram (a subroutine or user-defined function), all arguments to the subprogram must be defined in the subprogram's linkage section.

These arguments must be explicitly included on the PROCEDURE DIVISION USING (see Section 7.1 [PROCEDURE DIVISION USING], page 244) clause that lists the arguments in the sequence in which they will be passed to the subprogram.

These arguments described in the PROCEDURE DIVISION USING clause may each be defined as either BY REFERENCE, if the calling program is passing them either BY REFERENCE or BY CONTENT, or as BY VALUE if they are being passed BY VALUE.

By default, all arguments are assumed to be BY REFERENCE unless explicitly stated otherwise on the procedure division header.

Arguments to a user-defined function are always to be specified as BY REFERENCE (either explicitly or by not using any BY).

If the subprogram returns a value, the data item in which the value is returned must also be defined in the subprogram's linkage section, with a USAGE (see Section 6.9.59 [USAGE], page 228) of BINARY-LONG SIGNED, or its equivalent.

11.6.2 GLOBAL Data Items

Another way in which a data item may be shared between a calling program (A) and a called program (B) is by defining the data item in the calling program and attaching the GLOBAL (see Section 6.9.24 [GLOBAL], page 178) clause to it so that it may be used within the called program. In order for this to work, program B (the one called by program A) must be a nested subprogram within program A.

Here's a small example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoGLOBAL.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    Arg GLOBAL                    PIC X(10).
PROCEDURE DIVISION.
000-Main.
      MOVE ALL "X" TO Arg CALL
      "DemoSub" END-CALL
```

```
        DISPLAY "DemoGLOBAL: " Arg END-DISPLAY
        GOBACK
        -
    IDENTIFICATION DIVISION.
    PROGRAM-ID. DemoSub.
    PROCEDURE DIVISION.
    000-Main.
        MOVE ALL "*" TO Arg.
        GOBACK
        -
    END PROGRAM DemoSub.
    END PROGRAM
    DemoGLOBAL.
```

When the program runs, it produces the output:

```
DemoGLOBAL: **********
```

11.6.3 EXTERNAL Data Items

The final way in which a data item may be shared between a calling program (A) and a called program (B) is by defining the data item (with the same name) in both programs and attaching the EXTERNAL (see Section 6.9.19 [EXTERNAL], page 173) clause to it (again, in both pro- grams). This approach works regardless of whether the called program is nested within the calling program or not. It also works even if the two programs are compiled separately.

Here's a demonstration:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoEXTERNAL.
ENVIRONMENT DIVISION. DATA
DIVISION.
WORKING-STORAGE SECTION.
01    Arg EXTERNAL                    PIC X(10).
PROCEDURE DIVISION.
000-Main.
    MOVE ALL "X" TO Arg CALL
    "DemoSub" END-CALL
    DISPLAY "DemoEXTERNAL: " Arg END-DISPLAY
    GOBACK
    -
END PROGRAM DemoEXTERNAL.
IDENTIFICATION DIVISION.
PROGRAM-ID. DemoSub.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    Arg EXTERNAL                    PIC X(10).
PROCEDURE DIVISION.
000-Main.
    MOVE ALL "*" TO Arg.
    GOBACK
    -
END PROGRAM DemoSub.
```

When the program runs, it produces the output:

```
DemoEXTERNAL: **********
```

11.7 Recursive Subprograms

A subroutine may CALL itself, either directly or indirectly from another subroutine or user-defined function that it CALLs. Any subroutine that indulges in this sort of behaviour (called recursion) is called a Recursive Subprogram.

Any GnuCOBOL subroutine can be recursively invoked only if it is defined to the GnuCOBOL compiler as being a recursive subroutine. This is accomplished by adding the RECURSIVE at- tribute to its PROGRAM-ID (see Chapter 4 [IDENTIFICATION DIVISION], page 81).

All User-defined functions are automatically capable of being executed recursively.

Here is an example of a main program (DEMOFACT) that calls both a subprogram (SUB) and a user-defined function (FUNC) to compute the factorial value of a number.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEMOFACT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    FUNCTION RECURSIVEFUNC.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Result      USAGE  BINARY-LONG.
01  Arg         USAGE  BINARY-LONG.
PROCEDURE DIVISION.
000-Main.
    MOVE 6 TO Arg
    CALL "RECURSIVESUB"
        USING BY CONTENT Arg
        RETURNING Result
    DISPLAY Arg "! = "
            Result
    DISPLAY Arg "! = "
            RECURSIVEFUNC(Arg)
    GOBACK
    .
END PROGRAM DEMOFACT.
IDENTIFICATION DIVISION.
PROGRAM-ID. SUB RECURSIVE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Result      USAGE  BINARY-LONG.
01  Next-Arg    USAGE  BINARY-LONG.
01  Next-Result USAGE BINARY-LONG.
LINKAGE SECTION.
01  Arg         USAGE  BINARY-LONG.
PROCEDURE DIVISION USING Arg
                RETURNING Result.
000-Main.
    DISPLAY "Entering SUB"
            " Arg=" Arg
    IF Arg = 1
      MOVE 1 TO Result DISPLAY
      "Leaving SUB"
```

```
                    " Returning " Result
        ELSE
           SUBTRACT 1 FROM Arg
                 GIVING Next-Arg
           CALL "SUB"
                  USING BY CONTENT Next-Arg
                  RETURNING Next-Result
           COMPUTE Result =
                    Arg * Next-Result
           DISPLAY "Leaving SUB"
                    " Returning "
                    Result "=" Arg "*"
                    Next-Result
        END-IF
        GOBAC
        K
        -
    END PROGRAM SUB.
    IDENTIFICATION DIVISION.
    FUNCTION-ID. FUNC.
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    REPOSITORY.
        FUNCTION RECURSIVEFUNC.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    LINKAGE SECTION.
    01   Arg        USAGE  BINARY-LONG.
    01   Result     USAGE BINARY-LONG
                    SIGNED.
    PROCEDURE DIVISION USING Arg
                        RETURNING Result.
    000-Main.
        DISPLAY  "Entering  FUNC"
                 " Arg=" Arg
        IF Arg = 1
           MOVE 1 TO Result
        ELSE
           COMPUTE Result = Arg *
                    FUNC(Arg - 1)
        END-IF
        DISPLAY "Leaving FUNC"
                 " Returning " Result
        GOBACK
        -
    END FUNCTION FUNC.
```

When DEMOFACT is executed, the output shown below is generated.

```
E:\Programs\Demos>demofact
Entering RECURSIVESUB Arg=+0000000006
Entering RECURSIVESUB Arg=+0000000005
Entering RECURSIVESUB Arg=+0000000004
```

```
Entering RECURSIVESUB Arg=+0000000003
Entering RECURSIVESUB Arg=+0000000002
Entering RECURSIVESUB Arg=+0000000001
Leaving RECURSIVESUB Returning +0000000001
Leaving    RECURSIVESUB    Returning    +0000000002=+0000000002*+0000000001
Leaving    RECURSIVESUB    Returning    +0000000006=+0000000003*+0000000002
Leaving    RECURSIVESUB    Returning    +0000000024=+0000000004*+0000000006
Leaving    RECURSIVESUB    Returning    +0000000120=+0000000005*+0000000024
Leaving RECURSIVESUB Returning +0000000720=+0000000006*+0000000120
+0000000006! = +0000000720
Entering   RECURSIVEFUNC   Arg=+0000000006
Entering   RECURSIVEFUNC   Arg=+0000000005
Entering   RECURSIVEFUNC   Arg=+0000000004
Entering   RECURSIVEFUNC   Arg=+0000000003
Entering   RECURSIVEFUNC   Arg=+0000000002
Entering RECURSIVEFUNC Arg=+0000000001
Leaving   RECURSIVEFUNC   Returning   +0000000001
Leaving   RECURSIVEFUNC   Returning   +0000000002
Leaving   RECURSIVEFUNC   Returning   +0000000006
Leaving   RECURSIVEFUNC   Returning   +0000000024
Leaving   RECURSIVEFUNC   Returning   +0000000120
Leaving RECURSIVEFUNC Returning +0000000720
+0000000006! = +0000000720
```

11.8 Combining GnuCOBOL and C Programs

The upcoming sections deal the issues pertaining to calling C language programs from GnuCOBOL programs, and vice versa. Two additional sections provide samples illustrating specifics as to how those issues are overcome in actual program code.

11.8.1 GnuCOBOL Run-Time Library Requirements

Like most other implementations of the COBOL language, GnuCOBOL utilizes a run-time li- brary. When the first program executed in a given execution sequence is a GnuCOBOL program, any run-time library initialization will be performed by the compiled COBOL code in a manner that is transparent to the C-language programmer. If, however, a C program is the first to execute, the burden of performing GnuCOBOL run-time library initialization falls upon the C program. See Section 11.8.5 [C Main Programs Calling GnuCOBOL Subprograms], page 733, for an example of how to do this.

11.8.2 String Allocation Differences Between GnuCOBOL and C

Both languages store strings as a fixed-length continuous sequence of characters.

COBOL stores these character sequences up to a specific quantity limit imposed by the PICTURE (see Section 6.9.37 [PICTURE], page 195) clause of the data item. For example: 01 LastName PIC X(15)..

There is never an issue of exactly what the length of a string contained in a USAGE DISPLAY (see Section 6.9.59 [USAGE], page 228) data item is — there are always exactly how ever many characters as were allowed for by the PICTURE clause. In the example above, LastName will always contain exactly fifteen characters; of course, there may be anywhere from 0 to 15 trailing SPACES as part of the current LastName value.

C actually has no "string" data type; it stores strings as an array of char data type items where each element of the array is a single character. Being an array, there is an upper limit to how many characters may be stored in a given "string". For example:

```
char lastName[15]; /* 15 chars: lastName[0] through lastName[14] */
```

C provides a robust set of string-manipulation functions to copy strings from one char array to another, search strings for certain characters, compare one char array to another, concatenate char arrays and so forth. To make these functions possible, it was necessary to be able to define the logical end of a string. C accomplishes this via the expectation that all strings (char arrays) will be terminated by a NULL character (x'00'). Of course, no one forces a programmer to do this, but if [s]he ever expects to use any of the C standard functions to manipulate that string they had better be null-terminating their strings!

So, GnuCOBOL programmers expecting to pass strings to or receive strings from C programs had best be prepared to deal with the null-termination issue, as follows:

1. Pass a quoted literal string from GnuCOBOL to C as a zero-delimited string literal (Z'string').

2. Pass alphanumeric (PIC X) or alphabetic (PIC A) data items to C subroutines by appending an ASCII NUL character (X'00') to them. For example, to pass the 15-character LastName data item described above to a C subroutine:

   ```
   01  LastName-Arg-to-C        PIC X(16).
   ---
       MOVE FUNCTION CONCATENATE(LastName,X'00') TO LastName-Arg-to-C
   ```

   And then pass LastName-Arg-to-C to the C subprogram!

3. When a COBOL program needs to process string data prepared by a C program, the embedded null character must be accounted for. This can easily be accomplished with an INSPECT statement (see Section 7.8.26 [INSPECT], page 344) such as the following:

   ```
   INSPECT Data-From-a-C-Program
       REPLACING FIRST X'00' BY SPACE
                   CHARACTERS BY SPACE AFTER INITIAL X'00'
   ```

11.8.3 Matching C Data Types with GnuCOBOL USAGE's

Matching up GnuCOBOL numeric Usage's with their C language data type equivalents is possible via the following chart:

COBOL	C
BINARY-CHAR UNSIGNED	unsigned char
BINARY-CHAR [SIGNED]	signed char
BINARY-SHORT UNSIGNED	unsigned
	unsigned int
	unsigned short
	unsigned short int
BINARY-SHORT [SIGNED]	int
	short
	short int
	signed int
	signed short
	signed short int
BINARY-LONG UNSIGNED	unsigned long
	unsigned long int
BINARY-LONG [SIGNED]	long
BINARY-INT	long int
	signed long
	signed long int
BINARY-C-LONG [SIGNED]	long
BINARY-DOUBLE UNSIGNED	unsigned long long
	unsigned long long int
BINARY-DOUBLE [SIGNED]	long long int
BINARY-LONG-LONG	signed long long int
COMPUTATIONAL-1	float
COMPUTATIONAL-2	double
N/A (no GnuCOBOL equivalent)	long double

These sizes conform to the COBOL standard and the minimum sizes of the COBOL types are the same as the minimum sizes of the corresponding C data types. There's no official compatibility between them. Note that values in square braces '[]' are the defaults.

11.8.4 GnuCOBOL Main Programs CALLing C Subprograms

Here's a sample of a GnuCOBOL program that CALLs a C subprogram.

```
COBOL Calling Program                    C Called Program
================================= ==============================
IDENTIFICATION DIVISION.                 #include <stdio.h>
PROGRAM-ID. maincob.                     int subc(char *arg1,
DATA DIVISION.                                   char *arg2,
WORKING-STORAGE SECTION.                         unsigned long *arg3) {
01 Arg1        PIC X(7).                   char nu1[7]="New1";
01  Arg2       PIC X(7).                   char nu2[7]="New2";
01 Arg3        USAGE BINARY-LONG.          printf("Starting subc\n");
PROCEDURE DIVISION.                        printf("Arg1=%s\n",arg1);
000-Main.                                  printf("Arg2=%s\n",arg2);
    DISPLAY 'Starting maincob'             printf("Arg3=%d\n",*arg3);
    MOVE Z'Arg1'     TO Arg1               arg1[0]='X';
    MOVE Z'Arg2'     TO Arg2               arg2[0]='Y';
    MOVE 123456789 TO Arg3                 *arg3=987654321;
    CALL 'subc'                            return 2;
        USING BY CONTENT    Arg1,        }
              BY REFERENCE Arg2,
```

```
                BY REFERENCE Arg3
 DISPLAY 'Back'
 DISPLAY  'Arg1='  Arg1
 DISPLAY  'Arg2='  Arg2
 DISPLAY 'Arg3=' Arg3
 DISPLAY   'Returned   value='
           RETURN-CODE
 STOP RUN
 -
```

The idea is to pass two string and one full-word unsigned arguments to the subprogram, have the subprogram print them out, change all three and pass a return code of 2 back to the caller. The caller will then re-display the three arguments (showing changes only to the two BY REFERENCE arguments), display the return code and halt.

While simple, these two programs illustrate the techniques required quite nicely.

Note how the COBOL program ensures that a null end-of-string terminator is present on both string arguments.

Since the C program is planning on making changes to all three arguments, it declares all three as pointers in the function header and references the third argument as a pointer in the function body. It actually had no choice for the two string (char array) arguments — they must be defined as pointers in the function even though the function code references them without the leading '*' that normally signifies pointers.

These programs are compiled and executed as follows.

```
$ cobc -x maincob.cbl subc.c
$ maincob
Starting maincob
Starting subc
Arg1=Arg1
Arg2=Arg2
Arg3=123456789
Back
Arg1=Arg1
Arg2=Yrg2
Arg3=+0987654321
Returned value=+000000002
$
```

Remember that the null characters are actually in the GnuCOBOL Arg1 and Arg2 data items. They don't appear in the output, but they are there.

Did you notice the output showing the contents of Arg1 after the subroutine was called? Those contents were unchanged! The subroutine definitely changed that argument, but since the COBOL program passed that argument BY CONTENT, the change was made to a copy of the argument, not to the Arg1 data item itself.

11.8.5 C Main Programs Calling GnuCOBOL Subprograms

Now, the roles of the two languages in the previous section will be reversed, having a C main program execute a GnuCOBOL subprogram.

```
C Calling Program                                GNU-COBOL Called Program
================================================ ================================
#include <libcob.h> /* COB RUN-TIME */           IDENTIFICATION DIVISION.
#include <stdio.h>                               PROGRAM-ID. subcob.
int main (int argc, char **argv) {               DATA DIVISION.
    int returnCode;                              LINKAGE SECTION.
```

```
    char arg1[7] = "Arg1";
    char arg2[7] = "Arg2";
    unsigned long arg3 = 123456789;
    printf("Starting mainc...\n");
    cob_init (argc, argv); /* COB RUN-TIME */
    returnCode = subcob(arg1,arg2,&arg3);
    printf("Back\n");
    printf("Arg1=%s\n",arg1);
    printf("Arg2=%s\n",arg2);
    printf("Arg3=%d\n",arg3);
    printf("Returned value=%d\n",returnCode);
    return returnCode;
}
```

```
01    Arg1          PIC X(7).
01    Arg2          PIC X(7).
01    Arg3          USAGE BINARY-LONG.
PROCEDURE DIVISION USING
      BY VALUE        Arg1,
      BY REFERENCE Arg2,
      BY REFERENCE Arg3.
000-Main.
      DISPLAY 'Starting cobsub.cbl'
      DISPLAY  'Arg1='  Arg1
      DISPLAY  'Arg2='  Arg2
      DISPLAY 'Arg3=' Arg3
      MOVE  'X' TO Arg1 (1:1)
      MOVE 'Y' TO Arg2 (1:1) MOVE
      987654321 TO Arg3 MOVE 2
      TO RETURN-CODE GOBACK
      -
```

Since the C program is the one that will execute first, before the GnuCOBOL subroutine, the burden of initializing the GnuCOBOL run-time environment lies with that C program; it will have to invoke the cob_init function, which is part of the libcob library. The two required C statements are shown highlighted.

The arguments to the cob_init routine are the argument count and value parameters passed to the main function when the program began execution. By passing them into the GnuCOBOL subprogram, it will be possible for that GnuCOBOL program to retrieve the command line or individual command-line arguments. If that won't be necessary, cob_init(0,NULL); could be specified instead.

Since the C program wants to allow arg3 to be changed by the subprogram, it prefixes it with a '&' to force a CALL BY REFERENCE for that argument. Since arg1 and arg2 are strings (char arrays), they are automatically passed by reference.

Here's the output of the compilation process as well as the program's execution. The example assumes a Windows system with a GnuCOBOL build that uses the GNU C compiler on that system; the technique works equally well regardless of which C compiler and which operating system you're using.

```
C:\Users\Gary\Documents\Programs> cobc -S subcob.cbl C:\Users\Gary\Documents\Programs>
gcc mainc.c subcob.s -o mainc.exe -llibcob C:\Users\Gary\Documents\Programs> mainc.exe
Starting mainc...
Starting cobsub.cbl
Arg1=Arg1
Arg2=Arg2
Arg3=+0123456789
Back
Arg1=Xrg1
Arg2=Yrg2
Arg3=987654321
Returned value=2
C:\Users\Gary\Documents\Programs>
```

Note that even though we told GnuCOBOL that the 1st argument was to be BY VALUE, it was treated as if it were BY REFERENCE anyway. String (char array) arguments passed from C callers to GnuCOBOL subprograms will be modifiable by the subprogram. It's best to pass a copy of such data if you want to ensure that the subprogram doesn't change it.

The third argument is different, however. Since it's not an array you have the choice of passing it either BY REFERENCE or BY VALUE.

————————————————————————

End of Chapter 11 — Sub-Programming

12 Programming Style Suggestions

This chapter deals with a variety of stylistic issues that may be of interest to someone who is just starting out learning and using COBOL. Much of this chapter makes recommendations and suggestions for how to write your own programs. The sample programs in the Sample Programs document (see Sample Programs) were coded using almost all of these recommendations.

There's no particular order of importance to the topics presented here.

12.1 Marking Changes in Programs

Historically in the early 60's programs were first punched on to paper tape and by the mid 60's that was replaced almost totally, by punched cards although paper tape was still used by programmers for the odd few changes to their sources held on magnetic tape or disk as a portable paper tape punch could be put in your pocket. Now the problem with punched cards were there was 2,000 cards per box and that they could and did, get dropped. So, cc (column) 1 through 6 had the card sequence number in and that way if a box was dropped they could be feed in to a card sorter to be fixed. This was after the cards was cleaned up so that they were all in the same direction which one corner cut out helped.

In the late 70's cards was also on its way out to the point where P.C's started being used (and no they were not made by IBM), so these columns could be used for other purposes including cc 73 - 80 instead of indicating the 8 character program name which was the maximum size allowed on a IBM system.

For quite a while now (back to the late 1970's), the sequence number area' of a COBOL statement (columns 1-6) has come to be used as a change indicator area. Programmers would place a code in columns 1-6 of every line they changed in a program. The author works in a COBOL shop where change indicators of the form "xxmmyy" are required on every altered line of a program — "xx" is the initials of the programmer while "mmyy" are the month and two-digit year of the date the change was made. This is frequently accompanied by a comment block at or near the top of a COBOL program providing general documentation of what changes were made and what change indicator was used to mark that change.

The GCic sample program source listing (see Section "GCic" in GnuCOBOL Sample Programs) provides an excellent example of such documentation.

This technique of using columns 1-6 as a change indicator will only work if fixed source-record format is in effect.

Some COBOL shops prefer to use the eight-character Program Name Area (columns 73-80) as a change code area.

Marking changes becomes more of a challenge when free-format source code is in effect. Creating a top-of-program comment block to generically describe changes that have been made isn't difficult, even in free-form. What is difficult, however, is coming up with a scheme for per-statement mark up of changes that doesn't introduce a ridiculously excessive number of source lines to the program. I'm not sure there is a good answer to this problem (if a reader has one, please let me know). Generally, I've noticed that shops using free-format conventions for their COBOL source tend to stick with just the top-of-program comment block combined with minimal comment blocks sprinkled throughout the program noting areas that underwent major changes.

12.2 Data Item Coding and Naming Conventions

When programs get very large, it becomes more and more challenging to keep track of the data items that will be used in the program. Here, in no particular order of importance, are a variety of conventions that can simplify that problem.

Remember that the points described here are intended to make things easier for you, the programmer. No COBOL compiler cares one way or another whether any of these suggestions are followed.

1. Avoid the use of level 77 data items in new programs. Once (1968 and before) there were valid reasons for creating level-77 data items, but since the 1974 ANSI standard of COBOL there really hasn't been any reason why an elementary level-01 data item couldn't have been used instead of a level-77 item.
2. Allocate level-01 data items in alphabetical sequence in the program source wherever practical. This will make it vastly easier to locate the definitions of 01-level items in the program source without having to resort to a compilation cross-reference listing and/or text editor "find" command to locate them.
3. *Consider* prefixing data items with an indication of where in the program structure they were created. For example:
   - Start everything defined in the file section with "F-"
   - Start everything defined in working-storage with "WS-"
   - Start everything defined in local-storage with "LS-"
   - Start everything defined in the linkage section with "L-"
   - Start everything defined in the screen section with "S-"
   - Start everything defined in the report section with "R-"

   A convention such as this makes it simple, when you're reviewing code in the procedure division, to know in which section of the data division you should look in when locating the detailed description of a data item. Once you're in the right division, coding convention #2 will assist in locating the data item definition.
4. Consider including a trailing descriptor of the nature of all data items in their names. The following chart presents a variety of such descriptors the author has encountered and used through the years.

-ADDR	The data item contains all or a part of an Address (City-ADDR, State-ADDR, Street-ADDR, . . .)
-BOOL	A level-88 data item (which only has the value TRUE or FALSE)
-CD	A code whose value denotes information content above and beyond that of the mere value itself. Some examples could be Error-CD, Status-CD, Billing-CD
-CHR	A data item containing a single character of data.
-CONST	A constant, specified as a level-78 data item, a level-01 item with the CONST attribute
-DT	The data item contains a complete or partial date (Birth-DT, Birth-Month-DT, Birth-Year-DT, . . .)
-DTTM	A data item containing both a date and a time
-FILE	A file name. Note that these items would probably also have a "F-" prefix.
-IDX	A data item used as a table index (see section 12.3)

-NM All or a portion of a person's name. These could be extended to include business names, product names, etc.

-PTR A data item whose USAGE is POINTER

-NUM A generic numeric data item that doesn't fit into any of the other categories

-QTY A count of something

-REC An 01-level item defined in the FILE SECTION (constituting the layout of a record within a file). Note that these items would probably also have a "F-" prefix.

-SCR The data item contains a complete or partial screen description (appropriate for SCREEN SECTION 01-level data items).

-SUB A numeric item used as a table subscript (see section 12.3)

-TEL All or part of a telephone number

-TM The data item contains a complete or partial time value

-TXT The data item contains generic alphanumeric text that doesn't fit into any of the other categories.

The above is by no means an exhaustive list, but good programmers will use as few of these descriptors as possible as having too many defeats any benefits of such classification/documentation efforts.

5. Consider including an acronym to be inserted into the name of any data item defined directly or indirectly subordinate to an 01-level item, typically to be specified after any section-level tag, if you're using them. For example, consider the names used in the following structure:

```
01  WS-File-Status-Message-TXT.
    05 FILLER                    PIC X(13) VALUE 'Status Code: '.
    05 WS-FSM-Status-CD          PIC 9(2).
    05 FILLER                    PIC X(11) VALUE ', Meaning: '.
    05 WS-FSM-Msg-TXT            PIC X(25).
----
01  WS-OI-SUB                   PIC  99  COMP.
01  WS-OI-IDX                   PIC 99   COMP.
```

The "-FSM-" acronyms make it easier to locate the description of the 01-item the status code and message text items belong to.

12.3 Table Subscripting versus Table Indexing

The elements of a table may be referenced either using a subscript or an index. Syntactically, this is coded using parenthesis, as per the following three examples, all of which store the letter 'A' into the 17th occurrence of a data item named WS-Output-Image-TXT:

1. MOVE 'A' TO WS-Output-Image-TXT (17)
2. MOVE 17 TO WS-OI-SUB
   MOVE 'A' TO WS-Output-Image-TXT (WS-OI-SUB)
3. SET WS-OI-IDX TO 17
   MOVE 'A' TO WS-Output-Image-TXT (WS-OI-IDX)

The 1st and 2nd examples are referred to as Subscripting while the 3rd is known as Indexing. The distinction is fairly simple.

Indexing is the process of referencing an element of a table utilizing a data item with an explicitly or implicitly defined USAGE (see Section 6.9.59 [USAGE], page 228) of INDEX to select the desired occurrence, while . . .

Subscripting is the process of referencing an element of a table utilizing either a numeric constant or an unedited numeric data item to select the desired occurrence.

Various implementations of COBOL generate object code that is quite different in each of these three situations, and GnuCOBOL is no exception.

In general, table references such as example #1 (constant subscript) generate the smallest, simplest and fastest object code while table references such as example #2 (numeric data item subscript) generate the largest, most-complicated and slowest object code.

Table references such as example #3 (table indexing) generate object code that falls in the middle of the other two but is far closer in efficiency to example #1 than #2.

Some COBOL statements (SEARCH (see Section 7.8.41 [SEARCH], page 376), SEARCH ALL (see Section 7.8.42 [SEARCH ALL], page 377) and the table-based SORT (see Section 7.8.44.2 [Table SORT], page 392)) require you to index the affected table and to utilize that index with those statements. With any other references to tables, the choice is left to the programmer as to which approach should be used. In general, follow these rules:

1. Use constant subscripts (example #1) wherever possible/practical.
2. If references to table elements are going to be performed many, many times (tens or hundreds of thousands of times or more) during program execution, you will probably see a noticeable reduction in program execution time if you use indexing versus subscripting.

It's impossible to perform any arithmetic operation against an index data item directly (other than a simple incrementation or decremental operation via the SET UP/DOWN statement (see Section 7.8.43.5 [SET UP/DOWN], page 383)). Situations where any non-trivial computations are required to calculate the effective occurrence number for a table reference will require you to use a conventional unedited numeric data item as the receiving field for the calculation. That calculated value would then need to be saved into the index data item via a SET Index statement.

If you only need to use the computed occurrence number once, you might as well just use the computed occurrence number data item as a subscript. If, however, you will need to use a computed "subscript" many more times than once, the run-time overhead of converting that occurrence value to an index (via SET Index) will be worth the coding effort.

Whew!

If references to table elements are not going to be performed many, many times it probably won't make much difference whether you use indexing or subscripting.

If you are comfortable with the C programming language, you might find the following simple GnuCOBOL program useful in exploring the differences between subscripting and indexing:

```
IDENTIFICATION      DIVISION.
PROGRAM-ID.   SUBVSINDEX.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    WS-TABLE-SUB                BINARY-LONG.
01    WS-TABLE.
      05 WS-TABLE-ENTRY            OCCURS  20 TIMES
                                  INDEXED BY WS-TABLE-IDX
                                  PIC X(1).
PROCEDURE DIVISION.
000-Main SECTION.
E1. MOVE 'A' TO WS-TABLE-ENTRY (17)
```

```
-
E2. MOVE 17 TO WS-TABLE-SUB
    MOVE 'A' TO WS-TABLE-ENTRY (WS-TABLE-SUB)
-
E3. SET WS-TABLE-IDX TO 17
    MOVE 'A' TO WS-TABLE-ENTRY (WS-TABLE-SUB)
-
```

Compile this program as follows (the assumption is made that you are executing the cobc command from the directory in which the above program source code (subvsindex.cbl) exists.

```
cobc -C -save-temps subvsindex.cbl\
```

After this command is executed, the file subvsindex.c will contain the procedure division C code and subvsindex.c.1.h will contain the working-storage C code. Compare the generated C code for each of the three MOVE statements.

12.4 Copybook Naming Conventions and Usage

Since the intent of a copybook is to introduce COBOL code into a particular spot in a program via the COPY statement (see Section 3.2 [COPY], page 63), it is always a good idea to prefix copybook names with a character sequence that identifies where in a program its contents are intended to be COPYed.

For example:

IDxxxxxxxx
: Copybooks containing code intended for the identification division. These will be rare as you almost never encounter copied code in the identification division.

EDxxxxxxxx
: Copybooks containing code intended for use in the environment division. These copybooks are generally used for predefined SPECIAL-NAMES (see Section 5.1.3 [SPECIAL-NAMES], page 90) or FILE-CONTROL (see Section 5.2 [INPUT-OUTPUT SECTION], page 101) syntax, Copybooks

DDxxxxxxxx
: that contain data definitions.

PDxxxxxxxx
: Copybooks that contain executable instructions.

12.5 PROCEDURE DIVISION Sections Versus Paragraphs

The issue of whether to use section and/or paragraph names (collectively referred to as procedure names) within the procedure division is one of near religious significance with many COBOL programmers.

COBOL programming standards used by many organizations that use the language generally call for procedure names to:

1. Contain a leading numeric component (for example: 2000-Update-Customer), AND. . .
2. Be defined in the procedure division in non-decreasing sequence of that numeric component.

When you are looking at or editing any large COBOL program that has been created with programming standards that include these two rules, it is always a simple thing to know whether a reference to a procedure is being made to code that exists before or after your current location in the program, simply by comparing the numeric component of the current procedure's name with the one in question.

Technically, GnuCOBOL does not require ANY procedure names be defined unless:

1. You are using the ALTER statement (see Section 7.8.4 [ALTER], page 289) (the use of which should be avoided at all costs)
2. You are using a procedural PERFORM statement (see Section 7.8.33.1 [Procedural PERFORM], page 360)
3. You are using a GO TO statement (see Section 7.8.22 [GO TO], page 335)
4. You are using a MERGE statement (see Section 7.8.29 [MERGE], page 351) with an OUTPUT PROCEDURE
5. You are using a SORT statement (see Section 7.8.44 [SORT], page 388) with either (or both) an INPUT PROCEDURE or OUTPUT PROCEDURE
6. You are using DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250)

Since it is difficult to write any non-trivial COBOL program that uses none of the above, lets assume you will be including at least one section or paragraph in your GnuCOBOL programs.

I like to use procedure division paragraphs and sections as follows:

1. The very first procedure defined in the procedure division of my programs, assuming no DECLARATIVES (see Section 7.5 [DECLARATIVES], page 250) are defined, will be a section named 000-Main. The declaration of this procedure will immediately follow the procedure division header (or END DECLARATIVES if DECLARATIVES are used).
2. Any procedures referenced by MERGE, PERFORM, or SORT statements will be defined as sections.
3. Any procedures referenced by GO TO statements will be defined as paragraphs, and those paragraphs will be defined in the same section as the GO TO statements that reference them. In other words, GO TO statements may not be used to transfer control to a point in a different section. This is not a GnuCOBOL rule — this is my own personal programming practice intended to improve the readability and maintainability of my programs.
4. I always include a numeric prefix to all procedure names I define, for the reasons stated earlier.
5. I do not use THRU on any MERGE, PERFORM or SORT statement unless the programming standards of the shop in which I am working require it. My reasoning for this is that it is too easy to accidentally introduce a new procedure into the scope of a THRU.

12.6 COMPUTE Versus ADD-SUBTRACT-MULTIPLY-DIVIDE

Over the years, there has been much debate over the efficiency and arithmetic accuracy of using the COMPUTE statement (see Section 7.8.9 [COMPUTE], page 297) rather than the four basic arithmetic operation statements.

Here are the facts — draw your own conclusions as to which approach is more appropriate under which circumstances.

1. The COMPUTE statement supports exponentiation (via the '**' operator) — there is no equivalent basic arithmetic statement. Although you could simulate integral exponentiation (raising a value to the third power, for example) using MULTIPLY statements, and you may use the SQRT intrinsic function (see Section 8.1.87 [SQRT], page 519) to find a square root, there's just no (easy) way to find the cube-root of a value without using the COMPUTE statement.
2. For non-trivial computations, COMPUTE statements "read" better. Take this, for example:

```
COMPUTE R = (A + B * C) / D
```

As compared to:

```
MULTIPLY B BY C GIVING TEMP ADD
A TO TEMP
DIVIDE TEMP BY D GIVING R
```

For non-trivial computations, COMPUTE statements may execute faster than the equivalent chain of basic arithmetic statements. For example, the COMPUTE statement shown above executes about 25% faster on my computer using GnuCOBOL than does the MULTIPLY- ADD-DIVIDE sequence.

3. For trivial computations, on the other hand, I prefer the inherent readability of a statement such as this:

```
ADD 1 TO WSS-Input-Trans-QTY
```

to this:

```
COMPUTE WS-Input-Trans-QTY = WS-Input-Trans-QTY + 1
```

————————————————————————

End of Chapter 12 — Programming Style Suggestions

13 Programming for XFD

This chapter deals with the initial support for ODBC and OCI as file-handlers (so far PostgrSQL, MySQL, SQLite, MSSQL) and OCI, along with new directory COB SCHEMA DIR containing the necessary internal schema files to match the file definition to the database table all though usage of $XDF.

13.1 GnuCobol use SQL for files

The ACUCOBOL compiler has a defined syntax for describing records that can be stored in SQL tables. This is referred to as XFD (extended file descriptor). The various XFD directives are intermixed in the record description of a data file as either $XFD (similar to $SET) or as comments *((XFD)). Each directive precedes the data item it is affecting. When a program is compiled that has these directives and the compiler is directed by a compile option, then the record layout is written out to a text file which is later used at run-time to process the data records. i.e., facilitate mapping the COBOL data fields to/from SQL columns.

The goal of this project is for GnuCOBOL to accept directives similar in syntax to the ACUCOBOL XFD syntax1 (plus a few additions as required) and develop run-time modules to handle all of the COBOL I/O verbs while the data is stored in a SQL table.

The purpose for doing this is to allow legacy COBOL to have data stored in SQL tables without rewriting the COBOL code. If a COBOL based application is already using EXEC SQL this feature of having INDEXED files stored in SQL tables is really not of any use. Once the data is in SQL tables it may be accessed by other SQL tools. To get access to more advanced SQL features, the COBOL code will need to be recoded to use EXEC SQL, but this can be done at points which provide the most advantage.

At present both INDEXED and RELATIVE files are supported using either an ODBC or OCI (Oracle Call Interface) interface. RELATIVE files are supported by adding a special column that gets a unique number as the RELATIVE KEY. The intention is that you could store the data for the master data files of an application in an SQL database. Often COBOL based applications have intermediate data files and it would be a waste of time to place these into the database. Also keep in mind that the performance of ISAM will be much better than using an SQL database. However, SQL provides the opportunity to use ad hoc inquiry and other third party tools to process your data.

When directed by the XFD directives, the GnuCOBOL compiler will write out the CREATE TABLE statements that could be used to represent the file. The GnuCOBOL compiler will also write out a data description file which is read at runtime and used to manage the conversion of data between the COBOL record view and the SQL column view. The format of this data description file is not exactly the same as what ACUCOBOL had used since the file is easily regenerated. The more important feature is to accept similar XFD directives intermixed in the record description.

XFD Directives

The following directives are accepted intermixed in the COBOL record description. The KEY IS clauses from SELECT statements will be used to generate SQL INDEX statements.

$XFD Name	**XFD Directive Descriptions**
ALL	Indicates this is a special module defining all files in an application system so all files in this module should be defined for use via SQL

FILE	Name to be used for SQL table, filename.ddl and filename.xd instead of guessing from the SELECT/ASSIGN. If FILE is not provided then if ASSIGN filename is used that is taken, other wise the SELECT name is used for the table name and the .ddl and .xd files.
ALPHA	Next field is handled as a single CHAR type. If a group item the entire group is a single CHAR column. May optionally have alternate name.
BINARY	Next field handled as BINARY (RAW) so arbitrary hex data is stored. If a group item the entire group becomes a single BINARY column. May optionally have alternate name
CHAR	Next field is handled as a single CHAR type. If a group item the entire group is a single CHAR column. May optionally have alternate name.
DATE	Next field is a date in the defined format M - month (01 12), Y - year (2 or 4 digit), D - day of month (01-31) H - hour (00 -23), N minute, S second, T - hundredths of a second J - Julian day, E - day of year YY%nn nn is the pivot year YYY+nnnn nnnn is added to YYY to compute the actual year
GROUP type	The next field or group item is taken as a single SQL column of the declared type
NAME	Name to be used for the next SQL column
NUMERIC	Next field is handled as NUMERIC, may optionally have alternate name
USE type	The word USE is essentially ignore and the next word defines the field type
VARCHAR	Emit next field as VARCHAR instead of CHAR
VAR-LENGTH	Emit next field as VARCHAR instead of CHAR
WHEN fld op val	fld is some previous field in the record, val is a numeric value of alpha value in quotes, op is one of =, !=, >, >=, <, <=. val may be other to indicate other cases. If the condition is true then the following field/group format is used.
AND fld op val	A continuation of the previous WHEN logically connect as an AND condition
OR fld op val	A continuation of the previous WHEN logically connect as an OR condition

Name	**GnuCOBOL Compile time Option Descriptions**
-fsqldb=name	name indicates which database is to be used. Choose one of ODBC, MySQL, MSSQL, OCI, Oracle10, Oracle11, Oracle12.
-fsqlschema=name	name is sub-directory into which the filename.ddl holding CREATE TA- BLE is written as well as the filename.xd holding the data mapping infor- mation. If omitted then the ddl/xd files are written to the base schema dir directory.

Environment Variable	runtime.cfg	GnuCOBOL Run time Option Descriptions
COB_SCHEMA_DIR	schema_dir	Define location where the filename.xd files are stored. Default: $COB_CONFIG_DIR/schema
COB_SCHEMA_NAME	schema_name	Database schema name
COB_SCHEMA_UID	schema_uid	Database User-id for connecting to the database
COB_SCHEMA_PWD	schema_pwd	Password for database User-Id
COB_SCHEMA_DSN	schema_dsn	DSN name for database

COB_DEBUG_LOG Parameters for GnuCOBOL logging feature.
Module type is db

Implementation Plan - GnuCOBOL Developer Notes

To support $XFD directives changes were made to pplex.l, ppparse.y and scanner.l. Also codegen.c and field.c will need changes.

Additional flags may have to be added to cb field to retain the XFD directive information.

In codegen.c (output file initialization) depending on use of XFD and compile options is the location to emit the CREATE TABLE and the data description file. The cob file will get some new flags for io routine to indicate the file is really handled by ODBC/OCI.

A new fileio routine (fodbc.c and later foci.c) were developed following the recently developed interface for multiple I/O handlers. The cob load xfd routine loads the data description file into internal structures. Multiple schemas could have the same table name/structure for develop- ment, QA, production, etc. This could be indicated via runtime.cfg options and/or environment variables. IO asgname may have format=odbc or format=oci to indicate which method is used to access the SQL table for the file. You may also specify table=sql table name to override the default SQL table.

At run-time when the file is opened the file description is loaded into memory for fast access and conversion of the data between COBOL data type and SQL column data. Since SQL tables are usually collected under a database schema well need a compile option to generate the external file description under either a default directory or under a given schema name. At run-time an environment variable and/or runtime.cfg option could be used to define which database schema is to be used along with the database user-id and password. Often a database user-id has a default schema associated with it.

The module fsqlxfd.c contains routines common to both fodbc.c and foci.c such as loading of the filename.xd information and conversion of the data between SQL and COBOL data types. When the file is OPENed and the table is found to not exist, then cob load ddl loads the filename.ddl into memory for the OCI/ODBC routine to submit the CREATE TABLE/INDEX statements as required. As the table is loaded, if it contains column types not supported by the current database then there is a translation done during the load process.

Format of filename.xd

GnuCOBOL has its own format instead of using what ACUCOBOL had. The first character on each line indicates what the line is. Lines starting with # or * are comments. Each field is separated by a comma.

Name	**Header Line Description**
H	Is a header
1	Version number
table	SQL table name
,	For character used as decimal, default ,
.	For character used as comma, default .
Hex	Indicate how numeric signs are handled
num	File type, 3 for INDEXED, 2 for RELATIVE

Name	**Label Description**
L	Is a Label
Num	Numeric label, may be referenced by goto or condition

Name	Goto Description
G	Is a Goto
Num	Numeric label to transfer to

Name	Condition Description
C	Is a Condition to be tested (Appear in postfix sequence) Complex WHEN clauses become multiple Condition tests
Label	If condition test is true, then goto this label, 0 indicates more coming, non-zero indicates end of the WHEN condition to be tested
Opcode	>=, >, <=, <, =, !=, &&, ||, !
Operand	Usually the data field
Value	String in quotes or numeric value

Name	Date Format Description
D	Date format definition
num	Unique number for this format
date	Date format string
num	Total number of digits in this date field
num	1 if DATE present else 0
num	1 if TIME present else 0
yy rule	+ means add yyAdj to YY, % defines pivot point
num	Value for adjusting the Year
P:L	Year Position and Length within field
P:L	Month Position and Length within field
P:L	Day Position and Length within field
P:L	Hours Position and Length within field
P:L	Minutes Position and Length within field
P:L	Seconds Position and Length within field
P:L	Century date Position and Length within field

You may define how your application stores DATE information. The database will always expect dates to be in a full YYYYMMDD format and date/time to be in YYYYMMDDHHMISS format. Date fields could be defined like the following:

```
$XFD DATE "YYYYMMDD"
$XFD DATE "YY%60MMDD"
$XFD DATE "YYY+1800MMDD"
$XFD DATE "YYMMDDCC"
```

The Y is a place holder for YEAR, MM for month, DD for day, HH hour, MI minutes, SS seconds, CC for century. If the Ys are followed by % then the digits after the % defines a pivot year used to map the YY value into a 4 digit year. In the above example if the YY value is below 60 then it is 19YY else 20YY

If the Y's are followed by + then the digits after the + are added to the Y value. In above example, the year is 1800 + YYY value. There is a limit of 16 different DATE formats per record.

If the day is defined like DDD (3 Ds) then it is taken to be the day of the year. For example:

```
$XFD DATE "YYYYDDD"
```

Name	**Data Description**
F	Define data Field
num	Offset from record to start of COBOL data field
num	Bytes occupied by COBOL data field
num	Type of data field
	1 Arbitrary Binary Data
	2 PIC S9 COMP-5
	3 PIC 9 COMP-5
	4 PIC 9 COMP-6
	5 PIC S9 BINARY/COMP/COMP-4
	6 PIC 9 BINARY/COMP/COMP-4
	7 PIC x COMP-X
	8 COMP-1/COMP-2
	9 PIC S9 COMP-3/PACKED DECIMAL
	10 PIC 9 COMP-3/PACKED DECIMAL
	11 PIC S9 SIGN LEADING
	12 PIC S9 SIGN LEADING SEPARATE
	13 PIC S9
	14 PIC 9 SIGN TRAILING
	15 PIC 9 SIGN TRAILING SEPARATE
	16 PIC 9
	17 PIC A
	18 PIC X National characters
	19 PIC X Wide characters
	20 PIC X
	21 PIC X - VARCHAR
num	Bytes needed to store as SQL data
num	Digits in numeric field
num	Scale (or decimal places)
num	Unique id of date format (see D)
num	COBOL level number
name	SQL Column name used for data field

Name	**Key Definitions Description**
K	Define an index
num	Index number (1 16)
dup	Y if duplicates allowed, else N
sup	Y if key may be suppressed, else N
char	Suppress character either as c or 0xHH or string
columns	Comma separate list of column names in order that make up the index

Example Records to Table With the following:

```
SELECT OPTIONAL TSPFILE
ASSIGN TO "testsql"
ORGANIZATION INDEXED ACCESS DYNAMIC RECORD
KEY IS PRIME-KEY
SOURCE IS CM-CUST-NUM
```

```
ALTERNATE RECORD KEY IS SPLIT-KEY2
SOURCE IS CM-TELEPHONE,CM-MACHINE WITH DUPLICATES
        SUPPRESS WHEN "900"

ALTERNATE RECORD KEY IS SPLIT-KEY3
SOURCE IS CM-DISK,CM-DP-MGR,CM-MACHINE WITH DUPLICATES
        SUPPRESS WHEN ALL "*"
FILE STATUS IS CUST-STAT
-

SELECT FLATFILE ASSIGN "relfile"
ORGANIZATION RELATIVE
ACCESS IS RANDOM RELATIVE KEY IS REC-NUM FILE
STATUS IS CUST-STAT.

SELECT FLATSEQ ASSIGN "relfile"
ORGANIZATION RELATIVE
ACCESS IS SEQUENTIAL RELATIVE KEY IS REC-NUM FILE
STATUS IS CUST-STAT.
```

Note that FLATFILE and FLATSEQ are the same file called relfile with a different ACCESS. Given the following record description:

```
$XFD  ALL
 FD   FLATFILE
      BLOCK CONTAINS 5 RECORDS.

 01   FLAT-RECORD.
      10   RL-CUST-NUM                    PICTURE  X(8).
      10   RL-COMPANY                     PICTURE  X(25).
      10   RL-TRAILER                     PICTURE  X(16).

 FD   FLATSEQ
      BLOCK CONTAINS 5 RECORDS.

 01   RS-RECORD.
      10   RS-CUST-NUM                    PICTURE  X(8).
      10   RS-COMPANY                     PICTURE  X(25).
      10   RS-TRAILER                     PICTURE  X(16).

 FD   TSPFILE
      BLOCK CONTAINS 5 RECORDS.

$XFD  NAME=tspfilex
 01   TSPFL-RECORD.
      05  TSPFL-REC.
$XFD  USE  GROUP  C U S T N U M
      10  CM-CUST-NUM.
```

```
            15    CM-CUST-PRE                          PICTURE X(3).
            15    CM-CUST-NNN                          PICTURE X(5).
        10    CM-STATUS                                PICTURE   X.
        10    CM-COMPANY                               PICTURE   X(25).
$XFD USE GROUP VAR_LENGTH custaddr
        10    CM-ADDRESS.
            15    CM-ADDRESS-1                         PICTURE X(25).
            15    CM-ADDRESS-2                         PICTURE X(25).
            15    CM-ADDRESS-3                         PICTURE X(25).
          10    CM-TELEPHONE                           PICTURE 9(10).
          10    CM-DP-MGR                              PICTURE X(25).
        10    CM-MACHINE                               PICTURE   X(8).
        10    CM-MEMORY                                PICTURE   X(4).
$XFD WHEN (CM-STATUS = 'A' && CM-TELEPHONE > 100)
$XFD AND CM-MACHINE = 'B' || CM-COMPANY = ' '
        10    CM-MEMORYX REDEFINES CM-MEMORY.
            15    CM-MEMSZ                             PICTURE 9(2).
      15    CM-MEMUNIT                               PICTURE   X(2).
          10    CM-DISK                                PICTURE X(8).
          10    CM-TAPE                                PICTURE X(8).
$XFD WHEN CM-STATUS = 'X'
        10    CM-TAPEX  REDEFINES CM-TAPE              PICTURE 9(8).
$XFD WHEN CM-STATUS = 'Y'
        10    CM-TAPEY  REDEFINES CM-TAPE              PICTURE   9(6)V99.
        10    CM-NO-TERMINALS                          PICTURE   9(5)  BINARY.
        10    CM-COMPX                                 PICTURE   XXX COMP-X.
        10    CM-COMP5                                 PICTURE   9(7)  COMP-5.
        10    CM-COMP1                                 COMP-1.
        10    CM-COMP2                                 COMP-2.
        10    CM-PRICE                                 PICTURE   9(3)V99  COMP-3.
        10    CM-PRICES                                PICTURE   S9(5)V99.
$XFD DATE "MMDDYYYY"
        10    CM-DATE                                 PICTURE 9(8) COMP-3.
$XFD DATE "YYMMDDCC"
        10    CM-DATE2                                 PICTURE   9(8)  COMP-3.
```

And the resulting SQL table(s) would look like:

tspfilex.ddl --

```
CREATE TABLE tspfilex (
    custnum                                CHAR(8) NOT NULL,
    status                                 CHAR(1),
    company                                CHAR(25),
    custaddr                               VARCHAR(75),
    telephone                              DECIMAL(10) NOT NULL,
    dp_mgr                                 CHAR(25)  NOT NULL,
    machine                                CHAR(8) NOT NULL,
    memory                                 CHAR(4),
```

```
    memsz                                   DECIMAL(2),
    memunit                                 CHAR(2),
    disk                                    CHAR(8) NOT NULL,
    tape                                    CHAR(8),
    tapex                                   DECIMAL(8),
    tapey                                   DECIMAL(8,2),
    no_terminals                            DECIMAL(5),
    compx                                   DECIMAL(8),
    comp5                                   DECIMAL(7),
    comp1                                   FLOAT(23),
    comp2                                   FLOAT(53),
    price                                   DECIMAL(5,2),
    prices                                  DECIMAL(7,2),
    date_x                                  DATE,
    date2                                   DATE
);
CREATE UNIQUE INDEX pk_tspfilex ON tspfilex (custnum); CREATE
INDEX k1_tspfilex ON tspfilex (telephone,machine); CREATE INDEX
k2_tspfilex ON tspfilex (disk,dp_mgr,machine);
```

relfile.ddl --

```
CREATE TABLE relfile (
    cust_num                                CHAR(8),
    company                                 CHAR(25),
    trailer                                 CHAR(16),
    rid_relfile                             BIGINT PRIMARY KEY
);
```

The tspfilex.xd description for this follows:

```
H,1,tspfilex,2,',','.',0,3
D,1,'MMDDYYYY',8,1,0,,0,4:4,0:2,2:2,0:0,0:0,0:0,0:0
D,2,'YYMMDDCC',8,1,0,,0,0:2,2:2,4:2,0:0,0:0,0:0,6:2
F,0000,0008,20,0009,0,0,,10,custnum
F,0008,0001,20,0002,0,0,,10,status
F,0009,0025,20,0026,0,0,,10,company
F,0034,0075,21,0076,0,0,,10,custaddr
F,0109,0010,16,0013,10,0,,10,telephone
F,0119,0025,20,0026,0,0,,10,dp_mgr
F,0144,0008,20,0009,0,0,,10,machine
C,0,=,status,'A'
C,0,>,telephone,100
C,0,&&
C,0,=,machine,'B'
C,0,=,company,' '
C,0,||
C,2,&&
F,0152,0004,20,0005,0,0,,10,memory
```

```
G,3
L,2
F,0152,0002,16,0005,2,0,,15,mems
z
F,0154,0002,20,0003,0,0,,15,memu
nit L,3
F,0156,0008,20,0009,0,0,,10,disk
C,5,=,status,'X'
C,6,=,status,'Y'
F,0164,0008,20,0009,0,0,,10,tape
G,7
L,5
F,0164,0008,16,0011,8,0,,10,tapex
G,7
L,6
F,0164,0008,16,0011,8,2,,10,tapey
L,7
F,0172,0004,06,0013,5,0,,10,no_terminals
F,0176,0003,07,0011,8,0,,10,compx
F,0179,0004,03,0013,7,0,,10,comp5
F,0183,0004,08,0036,15,8,,10,comp1
F,0187,0008,08,0036,34,17,,10,comp2
F,0195,0003,10,0008,5,2,,10,price
F,0198,0007,13,0010,7,2,,10,prices
F,0205,0005,10,0032,8,0,1,10,date_x
F,0210,0005,10,0032,8,0,2,10,date2
K,0,N,N,,custnum
K,1,Y,Y,"900",telephone,machine
K,2,Y,Y,0x2A,disk,dp_mgr,machine
```

The relfile.xd description for this follows:

```
H,1,relfile,0,',','.',0,2
F,0000,0008,20,0009,0,0,,10,cust_nu
m
F,0008,0025,20,0026,0,0,,10,compan
y F,0033,0016,20,0017,0,0,,10,trailer
F,0049,0004,03,0015,12,0,,00,rid_relf
ile
K,0,N,N,,rid_relfile
```

Application Schema

Sometimes COBOL programs will have local views of the data by copying a record to some other group item which redefines the data differently. In some cases these hidden redefines will cause problems for SQL columns. For example, if a field is defined as PIC 9(x) but sometimes the same location in the record contains non-numeric data, then SQL will reject the storing of such data as it violates the strict data typing. By using the $XFD WHEN you are able to define which a portion of the record has one view other another. The I/O logic will then only copy the valid column of data.

It may therefore be a good idea to collect all of the file definitions into a single module, verify that all possible redefines are identified. The $XFD ALL directives indicates that such a module is

being compiled. Think of this module as defining all of the master data files in an application which will be migrated into SQL tables.

Check the DDL

It is a good idea to check the DDL generated by the GnuCOBOL compiler and make sure it suits your local database guidelines. You may also need to add local directives such as a STORAGE clause for Oracle. You should then use the appropriate tool (eg. Mysql or sqlplus) and manually create all of the tables to be used by an application. There may also be considerations for what database user-id has permission to create tables, read/write tables etc.

Automatic Table Creation

At OPEN time, the filename.xd is processed and if not present that is considered a serious error (30) is returned.

At OPEN time, a check is done to see if the table exists and if not, the run-time code then reads the filename.ddl and submits it to the database to create the table and indexes. If the filename.ddl is not present then OPEN is given an error (30). If runtime.cfg option create_table is true then the runtime will attempt to recreate filename.ddl from the information in filename.xd.

Taken from Development Notes by Ron Norman February 2020.

————————————————————

End of Chapter 13 — Programming for XFD

Appendix A Glossary of Terms

Alphabetic Data Item

A data item whose PICTURE clause allows it to contain only upper- and/or lower-case letters. See Section 6.9.37 [PICTURE], page 195.

Alphanumeric Data Item

A data item whose PICTURE clause allows it to contain absolutely any charac- ter whatsoever. See Section 6.9.37 [PICTURE], page 195. Group items (see Section 2.1.7 [Structured Data], page 13) are also implicitly considered to be al- phanumeric data items.

Alphanumeric Literal

A string of characters enclosed within a pair of quotation marks ('"') or apostrophes ("'). See Section 2.1.19.2 [Alphanumeric Literals], page 33.

Called Program

Another way to refer to a subprogram. Note that a called program may also be a calling program.

Calling Program

A program that executes a subprogram. Note that a calling program may also be a called program.

Collating Sequence

The sequence in which the characters that are acceptable to a computer are ordered for purposes of all types of sorting, merging, comparing, and processing. Gnu- COBOL programs may utilize standard character-set collating sequences (such as that defined by the ASCII or EBCDIC character sets) or programmer-defined cus- tom sequences as specified in the OBJECT-COMPUTER paragraph (section 4.1.2) and defined in the SPECIAL-NAMES paragraph (section 4.1.4).

Compilation Group

The collection of all compilation units being compiled by a single execution of the GnuCOBOL compiler.

Compilation Unit

A single source file being compiled by the GnuCOBOL compiler. A compilation unit may contain one or more programs.

Control Break

An event that is triggered when a control field on an RWCS-generated report changes value. It is these events that trigger the generation of control heading and control footing groups.

Control Field

A field of data being presented within a detail group; as the various detail groups that comprise the report are presented, they are presumed to appear in sorted sequence of the control fields contained within them. As an example, a department-by-department sales report for a chain of stores would probably be sorted by store number and – within like store numbers – be further sorted by department number. The store number will undoubtedly serve as a control field for the report, allowing control heading groups to be presented before each sequence of detail groups for the same store and control footing groups to be presented after each such sequence.

Control Footing

A report group that appears immediately after one or more detail groups of an RWCS-generated report. Such are produced automatically as a result of a control

break. This type of group typically serves as a summary of the detail group(s) that precede it, as might be the case on a sales report for a chain of stores, where the detail groups documenting sales for each department (one department per detail group) from the same store might be followed by a control footing that provides a summation of the department-by-department sales for that store.

Control Heading
A report group that appears immediately before one or more detail groups of an RWCS-generated report. Such are produced automatically as a result of a control break. This type of group typically serves as an introduction to the detail group(s) that follow, as might be the case on a sales report for a chain of stores, where the detail groups documenting sales for each department (one department per detail group) from the same store might be preceded by a control heading that states the full name and location of the store.

Control Hierarchy
The natural hierarchy of control breaks within a RWCS-controlled report based upon the manner in which the data the report is being generated from is sorted.

Copybook
A segment of program code that may be utilized by multiple programs simply by having that program use the COPY statement to import that code into the program. Although similar to the "include" facility present in many other programming languages, the COBOL copybook mechanism is actually considerably more powerful. See Section 2.1.6 [Copybooks], page 13, for a general discussion. See Section 3.2 [COPY], page 63, for the specifics of the COPY statement.

Data Item
A contiguous area of storage within the memory space of a program that may be referenced, by name, in a COBOL program. Other programming languages use the term variable, property or attribute to describe the same concept. See Section 2.1.7 [Structured Data], page 13.

Detail Group
A report group that contains the detailed data being presented for the report.

Detail Report
An RWCS-generated report to which at least one type of detail group is presented.

Division A collection of zero, one, or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. Each di- vision consists of the division header and the related division body. There are four divisions in a GnuCOBOL program: Identification, Environment, Data, and Procedure (coded in that sequence). See Section 2.1.17 [Program Structure], page 30.

Dynamic Subprogram
A subprogram whose executable object code is contained in a different executable file as its calling program. Dynamic subprograms are therefore loaded into memory as needed.

Elementary Item
A data item that isn't itself comprised of other data items. See Section 2.1.7 [Structured Data], page 13.

Entry-point
A spot in the procedure division where a program may begin execution when it is executed from the operating system, invoked as a user-defined function or called by another program.

Every program has at least one entry-point — known as the primary entry-point — which corresponds to the first executable statement in the procedure division following the declaratives area, if any.

Additional entry-points may be defined via the ENTRY statement (see Section 7.8.14 [ENTRY], page 319).

Entry-point Name

Every entry-point has a name. That name must be unique for all programs that comprise an executable program. Entry-point names are defined using a subroutine's PROGRAM-ID paragraph, a user-defined function's FUNCTION-ID paragraph or via ENTRY (see Section 7.8.14 [ENTRY], page 319) statements coded in a subprogram's procedure division.

Executable File

The GnuCOBOL compiler can create operating-system appropriate files that may be executed directly from the operating system environment. On Windows systems, these will be .exe files whereas on UNIX systems they will have no specific exten- sions. The compiler's -x switch is used to create executable files. Only main programs should be compiled in this manner.

Execution Thread

The complete set of executable code that is run during the execution of a program. This includes the main program as well as all executed subprograms, including those that are both dynamically and statically loaded.

Figurative Constants

GnuCOBOL, like other COBOL implementations, supports a number of reserved words that may be used to represent a specific literal value. These are known as figurative constants. See Section 2.1.19.3 [Figurative Constants], page 34, for more information.

Fixed Format Mode

A mode of the GnuCOBOL compiler's operation where source statements are constrained to meeting the pre-2002 standard of limiting COBOL statements to 80 columns, with various columns having limitations as to what sort of COBOL syntax could be specified in them. See Section 2.1.16 [Format of Program Source Lines], page 27, for more information.

Free Format Mode

A mode of the GnuCOBOL compiler's operation where source statements are allowed to be as long as 255 characters, with no restrictions or requirements as to in which columns various syntax elements must appear. See Section 2.1.16 [Format of Program Source Lines], page 27, for more information.

Group Item

A hierarchical data structure where the group item — itself a data item — actually consists of two or more other contiguously allocated data items. For example, Employee-Name could be a 35-character data item consisting of a 20-character Last-Name data item followed by a 14-character First-Name and a 1-character Middle-Initial. See Section 2.1.7 [Structured Data], page 13.

Hexadecimal Alphanumeric Literal

These are alphanumeric literals whose character sequence is specified by hexadecimal value. These literals are formed by a quote- or apostrophe-delimited sequence of an even number of hexadecimal digits (upper- or lower-case), prefixed with the letter

'X' (also upper- or lower-case). For example, the character string "Demo" could be specified as the hexadecimal alphanumeric literal X'44656D6F', assuming the ASCII character set. See Section 2.1.19.2 [Alphanumeric Literals], page 33.

Hexadecimal Numeric Literal

A numeric literal whose value is specified by hexadecimal value. These literals are formed by a quote- or apostrophe-delimited sequence of from 1 to 16 hexadecimal digits (upper- or lower-case), prefixed with the letter 'H' (also upper- or lower-case). For example, the number 123456 could be specified as the hexadecimal numeric literal H'01E240'. See Section 2.1.19.1 [Numeric Literals], page 33.

Identifiers

These are data items a COBOL program will be working with. The vast majority of identifiers are defined by the user (programmer) while a few are pre-defined by the GnuCOBOL compiler. Identifiers pre-defined by the compiler are referred to as special registers. Other programming languages generally refer to identifiers as "variables".

Imperative Statement

Either a statement that begins with a non decision-making verb and specifies an unconditional action to be taken or a conditional verb such as IF or EVALUATE, delimited by its explicit scope terminator (such as END-IF or END-EVALUATE). An imperative statement can consist of a sequence of imperative statements.

Intrinsic Function

A built-in routine that accepts arguments and returns a value; syntactically, these may be used most places where GnuCOBOL identifiers are valid. See Appendix D [List of Intrinsic Functions], page 787, for documentation on all the GnuCOBOL intrinsic functions.

Level Number

A 1- or 2-digit number that indicates the hierarchical position of a data item in a group item or the special properties of a data description entry.

Level numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level numbers in the range 1 through 9 can be written either as a single digit or as a zero followed by the significant digit.

Level numbers 66, 77, 78 and 88 identify special properties of a data description entry.

Literal A generic term used for a constant value coded in a program that may be either a numeric literal or an alphanumeric literal.

Main program

A program that is executed directly from an operating system or shell event. Main programs are not executed from other programs (i.e. they are not called programs).

National Character set

A character set that supports symbols using other than the traditional Roman alphabet symbols used by the ASCII character set. Typically, such a character set uses a UTF-16 (i.e. 16 bits-per-character) encoding of the Unicode character set.

Support for national character sets in GnuCOBOL is currently only partially imple- mented, and the compile- and run-time effect of using the N symbol in a PICTURE (see Section 6.9.37 [PICTURE], page 195) clause to define a field as containing national characters is the same as if X(2) had been coded, with the additional effect that such a field will qualify as a NATIONAL or NATIONAL-EDITED field on an INITIALIZE (see Section 7.8.24 [INITIALIZE], page 339) statement.

Numeric Data Item
A data item whose PICTURE clause allows it to contain only the numeric digit characters 0-9 (signed or unsigned), or a data item whose PICTURE/USAGE combination allow it to contain actual binary numbers in integer, fixed-point, floating-point or packed-decimal format. Numeric data items are the only ones that may be used as table subscripts or as source arguments on arithmetic statements. PICTURE (see Section 6.9.37 [PICTURE], page 195), or USAGE (see Section 6.9.59 [USAGE], page 228).

Numeric Edited Data Item
An otherwise numeric data item whose PICTURE (see Section 6.9.37 [PICTURE], page 195) clause also contains any of the editing symbols '$', '*', '+', ',', '-', '.', '/', '0' (zero), 'B', 'CR', 'DB' or 'Z'. Numeric edited data items are not eligible to serve as table subscripts or source arguments on arithmetic statements.

Numeric Literal
A numeric constant. See Section 2.1.19.1 [Numeric Literals], page 33.

Page Footing
A report group that appears at the bottom of every page of an RWCS-generated report. Information typically found within such a report group might be:

- The date the report was generated
- The current page number of the report

Page Heading
A report group that appears at the top of every page of an RWCS-generated report. Information typically found within such a report group might be:

- A title for the report
- The date the report was generated
- The current page number of the report
- Column headings describing the fields within the detail group(s)

Primary Entry-Point
See entry-point.

Procedure
All executable code statements within a single procedure division paragraph or section.

Procedure name
A programmer-defined section or paragraph name in the procedure division assigned to a procedure. Procedure names serve as a means by which a statement may refer to the statements that follow the procedure name.

Program A GnuCOBOL main program or subprogram.

Qualification
The process of establishing a unique reference to a data item whose name is duplicated in a program. This takes the form of using the duplicated data name and the name of any of its parent data items, connected by OF or IN such that the combination of those two data names is unique within the program.

Record A group item that is not part of a higher-level group item. See Section 6.1 [Data Definition Principles], page 120. An elementary item with a level number of 01 can also be referred to as a record if its definition occurs in the file section, provided

that its definition does not include the CONSTANT attribute. See Section 6.2.2 [FILE-SECTION-Data-Item], page 126.

Report Footing
A report group that occurs only once in an RWCS-generated report — as the very last presented report group of the report. These typically serve as a visual indication that the report is finished.

Report Group
One or more consecutive lines on a report that serve a common informational purpose or function. For example, lines of text that are displayed at the top or bottom of every printed page of a report.

Report Heading
A report group that occurs only once in an RWCS-generated report — as the very first presented report group of the report. These typically serve as an introduction to the report.

Reserved Word
A word coded in a GnuCOBOL program without any quote or apostrophe characters around it (which would have transformed that sequence of characters into a literal string) which has a very specific meaning to the compiler. See Section 2.1.1 [Language Reserved Words], page 9, for a general discussion of the concept. Appendix C for a complete list of GnuCOBOL reserved words.

Sentence An arbitrarily long sequence of statements terminated by a period.

Special Registers
Special data items that are automatically defined for your use by the GnuCOBOL compiler. See Section 7.7 [Special Registers], page 260, for a complete list.

Statement
A single executable COBOL instruction. All statements start with a verb (DISPLAY, IF, MOVE, ...) which is followed by the operands and additional syntax elements that describe the actions to be performed.

Static Subprogram
A subprogram whose executable object code is part of the same executable file as its calling program. Static subprograms are therefore loaded into memory at the same time as their caller.

Subprogram
A program invoked directly by another program in such a manner that it may return control back to the other program, directly back to the point where the subprogram was invoked.

Subroutine
A subprogram executed from another via a GnuCOBOL CALL (see Section 7.8.5 [CALL], page 290) statement (or the equivalent in whatever programming language that other program was written in).

Summary Report
An RWCS-generated report to which no detail groups are presented.

User-Defined Function
A subprogram written in GnuCOBOL that is executed in a syntactically-similar manner to that by which the various built-in intrinsic functions are executed.

User-Defined Names
: Either the name of an identifier or a procedure in the program. GnuCOBOL limits user-defined names to a maximum of 31 characters taken from the set of numeric digits, upper- and lower-case letters, hyphens and underscores. A user-defined name may neither begin nor end with a hyphen or underscore. User-defined names used as file names may additionally not begin with a digit although - unlike many other programming languages - user-defined names used as identifiers or procedure names may. A vendor specific feature has been introduced into gnucobol to allow extending the 31 character limit to 63 when using -std=default. It may be added also to other conf files if required. NOTE: This feature will prevent you ever migrating source code compiled for gnucobol to work with other Cobol compilers (unless they also have such a feature) without a lot of changes to all such names.

Verb
: The first reserved word of a COBOL statement.

Zero-Delimited Alphanumeric Literals
: An alphanumeric literal prefixed with an upper- or lower-case 'Z' character — for example, Z'ABC'. These literals are one character longer than the value within apostrophes or quotes would make them appear. The extra character (the last character) will be a null character (comprised entirely of zero bits). These literals are ideal when defining or assigning values to alphanumeric data items that will be passed as arguments to a C subroutine. See Section 2.1.19.2 [Alphanumeric Literals], page 33.

———————————————————————

End of Appendix A — Glossary of Terms

Appendix B Reserved Word List

See Appendix C for the complete lists of ALL reserved words of all types.

————————————————————

End of Appendix B — Reserved Word Lists

Appendix C Grouped Word Lists by feature and function

The following is the complete list of ALL reserved words in the 05 February 2025 at 00:20 GMT. build of GnuCOBOL 3.2 - Final. Even though the functionality behind some of these words may not be implemented in this version of GnuCOBOL, none may be used as any user-defined name. This list includes ALL reserved, intrinsics, mnemonics and system and shows some 1100+ words in total. In addition there are the arithmetic and relational symbols as well as extra words that can be added and existing words removed by the use of the –std file content see the specific one used for each sub-set.

The following list of reserved words was extracted from cobc --list-reserved and shows the reserved words, an implementation

Please notice: This list is highly specific to the option -std=dialect and reserved word options (-freserved=word, -fno-reserved=word) in effect. You can get the list for a given dialect by calling cobc -std=dialect --list-reserved.

C.1 Common reserved words

Note: (C/S) stands for Context-sensitive words (words reserved only depending on a specific contexts)

Reserved word	Implemented	Aliases
3-D	Yes (C/S)	
ABSENT	Yes	
ACCEPT	Yes	
ACCESS	Yes	
ACTION	Yes (C/S)	
ACTIVATING	No (C/S)	
ACTIVE-CLASS	No	
ACTIVE-X	Yes (C/S)	
ACTUAL	Yes (C/S)	
ADD	Yes	
ADDRESS	Yes	
ADJUSTABLE-COLUMNS	Yes (C/S)	
ADVANCING	Yes	
AFTER	Yes	
ALIGNED	No	
ALIGNMENT	Yes (C/S)	
ALL	Yes	
ALLOCATE	Yes	
ALLOWING	Yes (C/S)	
ALPHABET	Yes	
ALPHABETIC	Yes	
ALPHABETIC-LOWER	Yes	
ALPHABETIC-UPPER	Yes	
ALPHANUMERIC	Yes	
ALPHANUMERIC-EDITED	Yes	
ALSO	Yes	
ALTER	Yes	
ALTERNATE	Yes	
AND	Yes	
ANUM	No (C/S)	

ANY	Yes	
ANYCASE	No	
APPLY	Yes (C/S)	
ARE	Yes	
AREA	Yes	AREAS
AREAS	Yes	AREA
ARGUMENT-NUMBER	Yes	
ARGUMENT-VALUE	Yes	
ARITHMETIC	Yes (C/S)	
AS	Yes	
ASCENDING	Yes	
ASCII	Yes (C/S)	
ASSIGN	Yes	
AT	Yes	
ATTRIBUTE	Yes (C/S)	
ATTRIBUTES	Yes (C/S)	
AUTHOR	Yes (C/S)	
AUTO	Yes (C/S)	AUTO-SKIP, AUTOTERMINATE
AUTO-DECIMAL	Yes (C/S)	
AUTO-SKIP	Yes	AUTO, AUTOTERMINATE
AUTO-SPIN	Yes (C/S)	
AUTOMATIC	Yes	
AUTOTERMINATE	Yes	AUTO, AUTO-SKIP
AWAY-FROM-ZERO	Yes (C/S)	
B-AND	Yes	
B-NOT	Yes	
B-OR	Yes	
B-SHIFT-L	Yes	
B-SHIFT-LC	Yes	
B-SHIFT-R	Yes	
B-SHIFT-RC	Yes	
B-XOR	Yes	
BACKGROUND-COLOR	Yes (C/S)	BACKGROUND-COLOUR
BACKGROUND-COLOUR	Yes	BACKGROUND-COLOR
BACKGROUND-HIGH	Yes	
BACKGROUND-LOW	Yes	
BACKGROUND-STANDARD	Yes	
BACKWARD	Yes (C/S)	
BAR	Yes (C/S)	
BASED	Yes	
BEEP	Yes	BELL
BEFORE	Yes	
BELL	Yes (C/S)	BEEP
BINARY	Yes	
BINARY-C-LONG	Yes	
BINARY-CHAR	Yes	
BINARY-DOUBLE	Yes	BINARY-LONG-LONG
BINARY-INT	Yes	BINARY-LONG
BINARY-LONG	Yes	BINARY-INT
BINARY-LONG-LONG	Yes	BINARY-DOUBLE
BINARY-SEQUENTIAL	Yes (C/S)	

BINARY-SHORT	Yes	
BIT	Yes	
BITMAP	Yes (C/S)	
BITMAP-END	Yes (C/S)	
BITMAP-HANDLE	Yes (C/S)	
BITMAP-NUMBER	Yes (C/S)	
BITMAP-START	Yes (C/S)	
BITMAP-TIMER	Yes (C/S)	
BITMAP-TRAILING	Yes (C/S)	
BITMAP-TRANSPARENT-COLOR	Yes (C/S)	
BITMAP-WIDTH	Yes (C/S)	
BLANK	Yes	
BLINK	Yes (C/S)	
BLOCK	Yes	
BOOLEAN	No	
BOTTOM	Yes	
BOX	Yes (C/S)	
BOXED	Yes (C/S)	
BULK-ADDITION	Yes (C/S)	
BUSY	Yes (C/S)	
BUTTONS	Yes (C/S)	
BY	Yes	
BYTE-LENGTH	Yes (C/S)	
C	Yes (C/S)	
CALENDAR-FONT	Yes (C/S)	
CALL	Yes	
CANCEL	Yes	
CANCEL-BUTTON	Yes (C/S)	
CAPACITY	Yes (C/S)	
CARD-PUNCH	Yes (C/S)	
CARD-READER	Yes (C/S)	
CASSETTE	Yes (C/S)	
CCOL	Yes (C/S)	
CD	Yes	
CELL	Yes (C/S)	CELLS
CELL-COLOR	Yes (C/S)	
CELL-DATA	Yes (C/S)	
CELL-FONT	Yes (C/S)	
CELL-PROTECTION	Yes (C/S)	
CELLS	Yes	CELL
CENTER	Yes (C/S)	
CENTERED	Yes (C/S)	
CENTERED-HEADINGS	Yes (C/S)	
CENTURY-DATE	Yes (C/S)	
CF	Yes	
CH	Yes	
CHAIN	No	
CHAINING	Yes	
CHANGED	Yes (C/S)	
CHARACTER	Yes	
CHARACTERS	Yes	

CHECK-BOX	Yes (C/S)	
CLASS	Yes	
CLASS-ID	No	
CLASSIFICATION	Yes (C/S)	
CLEAR-SELECTION	Yes (C/S)	
CLINE	Yes (C/S)	
CLINES	Yes (C/S)	
CLOSE	Yes	
COBOL	Yes (C/S)	
CODE	Yes	
CODE-SET	Yes	
COL	Yes	
COLLATING	Yes	
COLOR	Yes	
COLORS	Yes (C/S)	COLOURS
COLOURS	Yes	COLORS
COLS	Yes	
COLUMN	Yes	
COLUMN-COLOR	Yes (C/S)	
COLUMN-DIVIDERS	Yes (C/S)	
COLUMN-FONT	Yes (C/S)	
COLUMN-HEADINGS	Yes (C/S)	
COLUMN-PROTECTION	Yes (C/S)	
COLUMNS	Yes	
COMBO-BOX	Yes (C/S)	
COMMA	Yes	
COMMAND-LINE	Yes	
COMMIT	Yes	
COMMON	Yes	
COMMUNICATION	Yes	
COMP	Yes	COMPUTATIONAL
COMP-0	Yes	COMPUTATIONAL-0
COMP-1	Yes	COMPUTATIONAL-1
COMP-10	Yes	COMP-10, DOUBLE, FLOAT-LONG
COMP-15	Yes	COMP-15, DOUBLE, FLOAT-LONG
COMP-2	Yes	COMPUTATIONAL-2
COMP-3	Yes	COMPUTATIONAL-3
COMP-4	Yes	COMPUTATIONAL-4
COMP-5	Yes	COMPUTATIONAL-5
COMP-6	Yes	COMPUTATIONAL-6
COMP-9	Yes	FLOAT, FLOAT-SHORT
COMP-N	Yes	COMPUTATIONAL-N
COMP-X	Yes	COMPUTATIONAL-X
COMPUTATIONAL	Yes	COMP
COMPUTATIONAL-0	Yes	COMP-0
COMPUTATIONAL-1	Yes	COMP-1
COMPUTATIONAL-2	Yes	COMP-2
COMPUTATIONAL-3	Yes	COMP-3
COMPUTATIONAL-4	Yes	COMP-4
COMPUTATIONAL-5	Yes	COMP-5
COMPUTATIONAL-6	Yes	COMP-6

COMPUTATIONAL-N	Yes	COMP-N
COMPUTATIONAL-X	Yes	COMP-X
COMPUTE	Yes	
CONDITION	Yes	
CONFIGURATION	Yes	
CONSTANT	Yes	
CONTAINS	Yes	
CONTENT	Yes	
CONTINUE	Yes	
CONTROL	Yes	
CONTROLS	Yes	
CONVERSION	Yes (C/S)	
CONVERTING	Yes	
COPY	Yes	
COPY-SELECTION	Yes (C/S)	
CORE-INDEX	Yes (C/S)	
CORR	Yes	CORRESPONDING
CORRESPONDING	Yes	CORR
COUNT	Yes	
CRT	Yes	
CRT-UNDER	Yes	
CSIZE	Yes (C/S)	
CURRENCY	Yes	
CURRENT	No	
CURSOR	Yes	
CURSOR-COL	Yes (C/S)	
CURSOR-COLOR	Yes (C/S)	
CURSOR-FRAME-WIDTH	Yes (C/S)	
CURSOR-ROW	Yes (C/S)	
CURSOR-X	Yes (C/S)	
CURSOR-Y	Yes (C/S)	
CUSTOM-PRINT-TEMPLATE	Yes (C/S)	
CYCLE	Yes (C/S)	
CYL-INDEX	Yes (C/S)	
CYL-OVERFLOW	Yes (C/S)	
DASHED	Yes (C/S)	
DATA	Yes	
DATA-COLUMNS	Yes (C/S)	
DATA-POINTER	No	
DATA-TYPES	Yes (C/S)	
DATE	Yes	
DATE-COMPILED	Yes (C/S)	
DATE-ENTRY	Yes (C/S)	
DATE-MODIFIED	Yes (C/S)	
DATE-WRITTEN	Yes (C/S)	
DAY	Yes	
DAY-OF-WEEK	Yes	
DE	Yes	
DEBUGGING	Yes	
DECIMAL-POINT	Yes	
DECLARATIVES	Yes	

DEFAULT	Yes	
DEFAULT-BUTTON	Yes (C/S)	
DEFAULT-FONT	Yes	
DELETE	Yes	
DELIMITED	Yes	
DELIMITER	Yes	
DEPENDING	Yes	
DESCENDING	Yes	
DESTINATION	Yes	
DESTROY	Yes	
DETAIL	Yes	
DISABLE	Yes	
DISC	Yes (C/S)	
DISK	Yes (C/S)	
DISP	Yes (C/S)	
DISPLAY	Yes	
DISPLAY-COLUMNS	Yes (C/S)	
DISPLAY-FORMAT	Yes (C/S)	
DIVIDE	Yes	
DIVIDER-COLOR	Yes (C/S)	
DIVIDERS	Yes (C/S)	
DIVISION	Yes	
DOTDASH	Yes (C/S)	
DOTTED	Yes (C/S)	
DOUBLE	Yes	FLOAT-LONG
DOWN	Yes	
DRAG-COLOR	Yes (C/S)	
DROP-DOWN	Yes (C/S)	
DROP-LIST	Yes (C/S)	
DUPLICATES	Yes	
DYNAMIC	Yes	
EBCDIC	Yes (C/S)	
EC	Yes	
ECHO	Yes	
EDITING	No	
EGI	Yes	
ELEMENT	Yes (C/S)	
ELSE	Yes	
EMI	Yes	
EMPTY-CHECK	Yes	REQUIRED
ENABLE	Yes	
ENCODING	Yes (C/S)	
ENCRYPTION	Yes (C/S)	
END	Yes	
END-ACCEPT	Yes	
END-ADD	Yes	
END-CALL	Yes	
END-CHAIN	No	
END-COLOR	Yes (C/S)	
END-COMPUTE	Yes	
END-DELETE	Yes	

END-DISPLAY	Yes	
END-DIVIDE	Yes	
END-EVALUATE	Yes	
END-IF	Yes	
END-JSON	Yes	
END-MODIFY	Yes (C/S)	
END-MULTIPLY	Yes	
END-OF-PAGE	Yes	EOP
END-PERFORM	Yes	
END-READ	Yes	
END-RECEIVE	Yes	
END-RETURN	Yes	
END-REWRITE	Yes	
END-SEARCH	Yes	
END-SEND	Yes	
END-START	Yes	
END-STRING	Yes	
END-SUBTRACT	Yes	
END-UNSTRING	Yes	
END-WRITE	Yes	
END-XML	Yes	
ENGRAVED	Yes (C/S)	
ENSURE-VISIBLE	Yes (C/S)	
ENTRY	Yes	
ENTRY-CONVENTION	Yes (C/S)	
ENTRY-FIELD	Yes (C/S)	
ENTRY-REASON	Yes (C/S)	
ENVIRONMENT	Yes	
ENVIRONMENT-NAME	Yes	
ENVIRONMENT-VALUE	Yes	
EO	No	
EOL	Yes (C/S)	
EOP	Yes	END-OF-PAGE
EOS	Yes (C/S)	
EQUAL	Yes	EQUALS
EQUALS	Yes	EQUAL
ERASE	Yes (C/S)	
ERROR	Yes	
ESCAPE	Yes	
ESCAPE-BUTTON	Yes (C/S)	
ESI	Yes	
EVALUATE	Yes	
EVENT	Yes	
EVENT-LIST	Yes (C/S)	
EVERY	Yes (C/S)	
EXCEPTION	Yes	
EXCEPTION-OBJECT	No	
EXCEPTION-VALUE	Yes (C/S)	
EXCLUSIVE	Yes	
EXCLUSIVE-OR	No	
EXHIBIT	Yes	

EXIT	Yes	
EXPAND	Yes (C/S)	
EXPANDS	No (C/S)	
EXTEND	Yes	
EXTENDED-SEARCH	Yes (C/S)	
EXTERN	Yes (C/S)	
EXTERNAL	Yes	
EXTERNAL-FORM	Yes	
F	Yes (C/S)	
FACTORY	No	
FALSE	Yes	
FD	Yes	
FH--FCD	Yes (C/S)	
FH--KEYDEF	Yes (C/S)	
FILE	Yes	
FILE-CONTROL	Yes	
FILE-ID	Yes	
FILE-LIMIT	Yes (C/S)	
FILE-LIMITS	Yes (C/S)	
FILE-NAME	Yes (C/S)	
FILE-POS	Yes (C/S)	
FILL-COLOR	Yes (C/S)	
FILL-COLOR2	Yes (C/S)	
FILL-PERCENT	Yes (C/S)	
FILLER	Yes	
FINAL	Yes	
FINALLY	No	
FINISH-REASON	Yes (C/S)	
FIRST	Yes	
FIXED	Yes	
FIXED-FONT	Yes	
FIXED-WIDTH	Yes (C/S)	
FLAT	Yes (C/S)	
FLAT-BUTTONS	Yes (C/S)	
FLOAT	Yes	FLOAT-SHORT
FLOAT-BINARY-128	No	
FLOAT-BINARY-32	No	
FLOAT-BINARY-64	No	
FLOAT-DECIMAL-16	Yes	
FLOAT-DECIMAL-34	Yes	
FLOAT-EXTENDED	No	
FLOAT-INFINITY	No	
FLOAT-LONG	Yes	DOUBLE
FLOAT-NOT-A-NUMBER	No (C/S)	
FLOAT-SHORT	Yes	FLOAT
FLOATING	Yes	
FONT	Yes	
FOOTING	Yes	
FOR	Yes	
FOREGROUND-COLOR	Yes (C/S)	FOREGROUND-COLOUR
FOREGROUND-COLOUR	Yes	FOREGROUND-COLOR

FOREVER	Yes (C/S)	
FORMAT	No	
FRAME	Yes (C/S)	
FRAMED	Yes (C/S)	
FREE	Yes	
FROM	Yes	
FULL	Yes (C/S)	LENGTH-CHECK
FULL-HEIGHT	Yes (C/S)	
FUNCTION	Yes	
FUNCTION-ID	Yes	
FUNCTION-POINTER	No	
GENERATE	Yes	
GET	No	
GIVING	Yes	
GLOBAL	Yes	
GO	Yes	
GO-BACK	Yes (C/S)	
GO-FORWARD	Yes (C/S)	
GO-HOME	Yes (C/S)	
GO-SEARCH	Yes (C/S)	
GOBACK	Yes	
GRAPHICAL	Yes (C/S)	
GREATER	Yes	
GRID	Yes (C/S)	
GROUP	Yes	
GROUP-USAGE	No	
GROUP-VALUE	Yes (C/S)	
HANDLE	Yes	
HAS-CHILDREN	Yes (C/S)	
HEADING	Yes	
HEADING-COLOR	Yes (C/S)	
HEADING-DIVIDER-COLOR	Yes (C/S)	
HEADING-FONT	Yes (C/S)	
HEAVY	Yes (C/S)	
HEIGHT-IN-CELLS	Yes (C/S)	
HEX	No (C/S)	
HIDDEN-DATA	Yes (C/S)	
HIGH-COLOR	Yes (C/S)	
HIGH-VALUE	Yes	HIGH-VALUES
HIGH-VALUES	Yes	HIGH-VALUE
HIGHLIGHT	Yes (C/S)	
HOT-TRACK	Yes (C/S)	
HSCROLL	Yes (C/S)	
HSCROLL-POS	Yes (C/S)	
I-O	Yes	
I-O-CONTROL	Yes	
ICON	Yes (C/S)	
ID	Yes	
IDENTIFICATION	Yes	
IDENTIFIED	Yes	
IF	Yes	

IGNORE	Yes	
IGNORING	Yes (C/S)	
IMPLEMENTS	No (C/S)	
IN	Yes	
INDEPENDENT	Yes (C/S)	
INDEX	Yes	
INDEXED	Yes	
INDICATE	Yes	
INHERITS	No	
INITIAL	Yes	
INITIALISE	Yes	INITIALIZE
INITIALISED	Yes	INITIALIZED
INITIALIZE	Yes	INITIALISE
INITIALIZED	Yes (C/S)	INITIALISED
INITIATE	Yes	
INPUT	Yes	
INPUT-OUTPUT	Yes	
INQUIRE	Yes	
INSERT-ROWS	Yes (C/S)	
INSERTION-INDEX	Yes (C/S)	
INSPECT	Yes	
INSTALLATION	Yes (C/S)	
INTERFACE	No	
INTERFACE-ID	No	
INTERMEDIATE	Yes (C/S)	
INTO	Yes	
INTRINSIC	Yes (C/S)	
INVALID	Yes	
INVOKE	No	
IS	Yes	
ITEM	Yes (C/S)	
ITEM-TEXT	Yes (C/S)	
ITEM-TO-ADD	Yes (C/S)	
ITEM-TO-DELETE	Yes (C/S)	
ITEM-TO-EMPTY	Yes (C/S)	
ITEM-VALUE	Yes (C/S)	
JSON	Yes	
JUST	Yes	JUSTIFIED
JUSTIFIED	Yes	JUST
KEPT	Yes	
KEY	Yes	
KEYBOARD	Yes (C/S)	
LABEL	Yes	
LABEL-OFFSET	Yes (C/S)	
LARGE-FONT	Yes	
LARGE-OFFSET	Yes (C/S)	
LAST	Yes	
LAST-ROW	Yes (C/S)	
LAYOUT-DATA	Yes (C/S)	
LAYOUT-MANAGER	Yes	
LC_ALL	No (C/S)	

LC_COLLATE	No (C/S)	
LC_CTYPE	No (C/S)	
LC_MESSAGES	No (C/S)	
LC_MONETARY	No (C/S)	
LC_NUMERIC	No (C/S)	
LC_TIME	No (C/S)	
LEADING	Yes	
LEADING-SHIFT	Yes (C/S)	
LEAVE	Yes (C/S)	
LEFT	Yes	
LEFT-JUSTIFY	No	
LEFT-TEXT	Yes (C/S)	
LEFTLINE	Yes	
LENGTH	Yes	
LENGTH-CHECK	Yes	FULL
LESS	Yes	
LIKE	Yes	
LIMIT	Yes	
LIMITS	Yes	
LINAGE	Yes	
LINAGE-COUNTER	Yes	
LINE	Yes	
LINE-COUNTER	Yes	
LINE-SEQUENTIAL	Yes (C/S)	
LINES	Yes	
LINES-AT-ROOT	Yes (C/S)	
LINKAGE	Yes	
LIST-BOX	Yes (C/S)	
LM-RESIZE	Yes	
LOC	Yes (C/S)	
LOCAL-STORAGE	Yes	
LOCALE	Yes	
LOCATION	No	
LOCK	Yes	
LOCK-HOLDING	Yes (C/S)	
LONG-DATE	Yes (C/S)	
LOW-COLOR	Yes (C/S)	
LOW-VALUE	Yes	LOW-VALUES
LOW-VALUES	Yes	LOW-VALUE
LOWER	Yes (C/S)	
LOWERED	Yes (C/S)	
LOWLIGHT	Yes (C/S)	
MAGNETIC-TAPE	Yes (C/S)	
MANUAL	Yes	
MASS-UPDATE	Yes (C/S)	
MASTER-INDEX	Yes (C/S)	
MAX-LINES	Yes (C/S)	
MAX-PROGRESS	Yes (C/S)	
MAX-TEXT	Yes (C/S)	
MAX-VAL	Yes (C/S)	
MEDIUM-FONT	Yes	

MEMORY	Yes (C / S)
MENU	Yes
MERGE	Yes
MESSAGE	Yes
MESSAGE-TAG	No
METHOD	No
METHOD-ID	No
MIN-VAL	Yes (C/S)
MINUS	Yes
MODE	Yes
MODIFY	Yes
MODULES	Yes (C/S)
MOVE	Yes
MULTILINE	Yes (C/S)
MULTIPLE	Yes
MULTIPLY	Yes
NAME	Yes (C/S)
NAMED	Yes (C/S)
NAMESPACE	Yes (C/S)
NAMESPACE-PREFIX	Yes (C/S)
NAT	No
NATIONAL	Yes
NATIONAL-EDITED	Yes
NATIVE	Yes
NAVIGATE-URL	Yes (C/S)
NEAREST-AWAY-FROM-ZERO	Yes (C / S)
NEAREST-EVEN	Yes (C/S)
NEAREST-TOWARD-ZERO	Yes (C / S)
NEGATIVE	Yes
NESTED	Yes
NEW	Yes
NEXT	Yes
NEXT-ITEM	Yes (C/S)
NO	Yes
NO-AUTO-DEFAULT	Yes (C / S)
NO-AUTOSEL	Yes (C/S)
NO-BOX	Yes (C/S)
NO-DIVIDERS	Yes (C/S)
NO-ECHO	Yes
NO-F4	Yes (C/S)
NO-FOCUS	Yes (C/S)
NO-GROUP-TAB	Yes (C / S)
NO-KEY-LETTER	Yes (C/S)
NO-SEARCH	Yes (C/S)
NO-UPDOWN	Yes (C / S)
NOMINAL	Yes (C/S)
NONE	No (C/S)
NONNUMERIC	Yes (C/S)
NORMAL	Yes (C / S)
NOT	Yes
NOTAB	Yes (C/S)

NOTHING	Yes	
NOTIFY	Yes (C/S)	
NOTIFY-CHANGE	Yes (C/S)	
NOTIFY-DBLCLICK	Yes (C/S)	
NOTIFY-SELCHANGE	Yes (C/S)	
NULL	Yes	NULLS
NULLS	Yes	NULL
NUM-COL-HEADINGS	Yes (C/S)	
NUM-ROWS	Yes (C/S)	
NUMBER	Yes	
NUMBERS	Yes	
NUMERIC	Yes	
NUMERIC-EDITED	Yes	
OBJECT	Yes	
OBJECT-COMPUTER	Yes	
OBJECT-REFERENCE	No	
OCCURS	Yes	
OF	Yes	
OFF	Yes	
OK-BUTTON	Yes (C/S)	
OMITTED	Yes	
ON	Yes	
ONLY	Yes	
OPEN	Yes	
OPTIONAL	Yes	
OPTIONS	Yes	
OR	Yes	
ORDER	Yes	
ORGANISATION	Yes	ORGANIZATION
ORGANIZATION	Yes	ORGANISATION
OTHER	Yes	
OTHERS	Yes (C/S)	
OUTPUT	Yes	
OVERFLOW	Yes	
OVERLAP-LEFT	Yes (C/S)	OVERLAP-TOP
OVERLAP-TOP	Yes (C/S)	OVERLAP-LEFT
OVERLINE	Yes	
OVERRIDE	No	
PACKED-DECIMAL	Yes	
PADDING	Yes	
PAGE	Yes	
PAGE-COUNTER	Yes	
PAGE-SETUP	Yes (C/S)	
PAGED	Yes (C/S)	
PARAGRAPH	Yes (C/S)	
PARENT	Yes (C/S)	
PARSE	Yes (C/S)	
PASCAL	Yes (C/S)	
PASSWORD	Yes (C/S)	
PERFORM	Yes	
PERMANENT	Yes (C/S)	

PF	Yes	
PH	Yes	
PHYSICAL	Yes	
PIC	Yes	PICTURE
PICTURE	Yes	PIC
PIXEL	Yes (C/S)	PIXELS
PIXELS	Yes	PIXEL
PLACEMENT	Yes (C/S)	
PLUS	Yes	
POINTER	Yes	
POP-UP	Yes (C/S)	
POS	Yes	
POSITION	Yes	
POSITION-SHIFT	Yes (C/S)	
POSITIVE	Yes	
PREFIXED	No (C/S)	
PRESENT	Yes	
PREVIOUS	Yes (C/S)	
PRINT	Yes (C/S)	
PRINT-NO-PROMPT	Yes (C/S)	
PRINT-PREVIEW	Yes (C/S)	
PRINTER	Yes (C/S)	
PRINTER-1	Yes (C/S)	
PRINTING	Yes	
PRIORITY	Yes	
PROCEDURE	Yes	
PROCEDURE-POINTER	Yes	PROGRAM-POINTER
PROCEDURES	Yes	
PROCEED	Yes	
PROCESSING	Yes (C/S)	
PROGRAM	Yes	
PROGRAM-ID	Yes	
PROGRAM-POINTER	Yes	PROCEDURE-POINTER
PROGRESS	Yes (C/S)	
PROHIBITED	Yes (C/S)	
PROMPT	Yes	
PROPERTIES	Yes (C/S)	
PROPERTY	Yes	
PROTECTED	Yes (C/S)	
PROTOTYPE	No	
PURGE	Yes	
PUSH-BUTTON	Yes (C/S)	
QUERY-INDEX	Yes (C/S)	
QUEUE	Yes	
QUOTE	Yes	QUOTES
QUOTES	Yes	QUOTE
RADIO-BUTTON	Yes (C/S)	
RAISE	Yes	
RAISED	Yes (C/S)	
RAISING	No	
RANDOM	Yes	

RD	Yes	
READ	Yes	
READ-ONLY	Yes (C/S)	
READERS	Yes (C/S)	
RECEIVE	Yes	
RECEIVED	Yes	
RECORD	Yes	
RECORD-DATA	Yes (C/S)	
RECORD-OVERFLOW	Yes (C/S)	
RECORD-TO-ADD	Yes (C / S)	
RECORD-TO-DELETE	Yes (C/S)	
RECORDING	Yes	
RECORDS	Yes	
RECURSIVE	Yes (C/S)	
REDEFINES	Yes	
REEL	Yes	
REFERENCE	Yes	
REFERENCES	Yes	
REFRESH	Yes (C/S)	
REGION-COLOR	Yes (C/S)	
RELATION	No (C/S)	
RELATIVE	Yes	
RELEASE	Yes	
REMAINDER	Yes	
REMARKS	Yes (C/S)	
REMOVAL	Yes	
RENAMES	Yes	
REORG-CRITERIA	Yes (C/S)	
REPEATED	Yes	
REPLACE	Yes	
REPLACING	Yes	
REPORT	Yes	
REPORTING	Yes	
REPORTS	Yes	
REPOSITORY	Yes	
REQUIRED	Yes (C / S)	EMPTY-CHECK
REREAD	Yes (C/S)	
RERUN	Yes (C/S)	
RESERVE	Yes	
RESET	Yes	
RESET-GRID	Yes (C/S)	
RESET-LIST	Yes (C/S)	
RESET-TABS	Yes (C/S)	
RESUME	No	
RETRY	Yes	
RETURN	Yes	
RETURNING	Yes	
REVERSE	Yes	
REVERSE-VIDEO	Yes (C/S)	
REVERSED	Yes	
REWIND	Yes	

REWRITE	Yes
RF	Yes
RH	Yes
RIGHT	Yes
RIGHT-ALIGN	Yes (C/S)
RIGHT-JUSTIFY	No
RIMMED	Yes (C/S)
ROLLBACK	Yes
ROUNDED	Yes
ROUNDING	Yes (C/S)
ROW-COLOR	Yes (C / S)
ROW-COLOR-PATTERN	Yes (C / S)
ROW-DIVIDERS	Yes (C/S)
ROW-FONT	Yes (C/S)
ROW-HEADINGS	Yes (C/S)
ROW-PROTECTION	Yes (C/S)
RUN	Yes
S	Yes (C/S)
SAME	Yes
SAVE-AS	Yes (C/S)
SAVE-AS-NO-PROMPT	Yes (C / S)
SCREEN	Yes
SCROLL	Yes (C/S)
SCROLL-BAR	Yes (C/S)
SD	Yes
SEARCH	Yes
SEARCH-OPTIONS	Yes (C/S)
SEARCH-TEXT	Yes (C/S)
SECONDS	Yes (C/S)
SECTION	Yes
SECURE	Yes (C/S)
SECURITY	Yes (C/S)
SEGMENT	Yes
SEGMENT-LIMIT	Yes
SELECT	Yes
SELECT-ALL	Yes (C/S)
SELECTION-INDEX	Yes (C/S)
SELECTION-TEXT	Yes (C/S)
SELF	No
SELF-ACT	Yes (C/S)
SEND	Yes
SENTENCE	Yes
SEPARATE	Yes
SEPARATION	Yes (C/S)
SEQUENCE	Yes
SEQUENTIAL	Yes
SET	Yes
SHADING	Yes (C/S)
SHADOW	Yes (C / S)
SHARING	Yes
SHORT-DATE	Yes (C/S)

SHOW-LINES	Yes (C/S)	
SHOW-NONE	Yes (C/S)	
SHOW-SEL-ALWAYS	Yes (C/S)	
SIGN	Yes	
SIGNED	Yes	
SIGNED-INT	Yes	
SIGNED-LONG	Yes	
SIGNED-SHORT	Yes	
SIZE	Yes	
SMALL-FONT	Yes	
SORT	Yes	
SORT-MERGE	Yes	
SORT-ORDER	Yes (C/S)	
SOURCE	Yes	
SOURCE-COMPUTER	Yes	
SOURCES	No	
SPACE	Yes	SPACES
SPACE-FILL	No	
SPACES	Yes	SPACE
SPECIAL-NAMES	Yes	
SPINNER	Yes (C/S)	
SQUARE	Yes (C/S)	
STACK	No (C/S)	
STANDARD	Yes	
STANDARD-1	Yes	
STANDARD-2	Yes	
STANDARD-BINARY	Yes (C/S)	
STANDARD-DECIMAL	Yes (C/S)	
START	Yes	
START-X	Yes (C/S)	
START-Y	Yes (C/S)	
STATEMENT	No (C/S)	
STATIC	Yes (C/S)	
STATIC-LIST	Yes (C/S)	
STATUS	Yes	
STATUS-BAR	Yes (C/S)	
STATUS-TEXT	Yes (C/S)	
STDCALL	Yes (C/S)	
STEP	Yes (C/S)	
STOP	Yes	
STRING	Yes	
STRONG	No (C/S)	
STYLE	Yes (C/S)	
SUB-QUEUE-1	Yes	
SUB-QUEUE-2	Yes	
SUB-QUEUE-3	Yes	
SUBTRACT	Yes	
SUBWINDOW	Yes	
SUM	Yes	
SUPER	No	
SUPPRESS	Yes	

SYMBOL	No (C/S)	
SYMBOLIC	Yes	
SYNC	Yes	SYNCHRONISED, SYNCHRONIZED
SYNCHRONISED	Yes	SYNC, SYNCHRONIZED
SYNCHRONIZED	Yes	SYNC, SYNCHRONISED
SYSTEM-DEFAULT	Yes	
SYSTEM-INFO	Yes (C/S)	
SYSTEM-OFFSET	Yes	
TAB	Yes (C/S)	
TAB-TO-ADD	Yes (C/S)	
TAB-TO-DELETE	Yes (C/S)	
TABLE	Yes	
TALLYING	Yes	
TAPE	Yes (C/S)	
TEMPORARY	Yes (C/S)	
TERMINAL-INFO	Yes (C/S)	
TERMINATE	Yes	
TERMINATION-VALUE	Yes (C/S)	
TEST	Yes	
TEXT	Yes	
THAN	Yes	
THEN	Yes	
THREAD	Yes	
THREADS	Yes	
THROUGH	Yes	THRU
THRU	Yes	THROUGH
THUMB-POSITION	Yes (C/S)	
TILED-HEADINGS	Yes (C/S)	
TIME	Yes	
TIME-OUT	Yes (C/S)	TIMEOUT
TIMEOUT	Yes	TIME-OUT
TIMES	Yes	
TITLE	Yes (C/S)	
TITLE-POSITION	Yes (C/S)	
TO	Yes	
TOP	Yes	
TOP-LEVEL	No (C/S)	
TOWARD-GREATER	Yes (C/S)	
TOWARD-LESSER	Yes (C/S)	
TRACK	Yes (C/S)	
TRACK-AREA	Yes (C/S)	
TRACK-LIMIT	Yes (C/S)	
TRACKS	Yes (C/S)	
TRADITIONAL-FONT	Yes	
TRAILING	Yes	
TRAILING-SHIFT	Yes (C/S)	
TRAILING-SIGN	No	
TRANSFORM	Yes	
TRANSPARENT	Yes (C/S)	
TREE-VIEW	Yes (C/S)	
TRUE	Yes	

TRUNCATION	Yes (C/S)	
TYPE	Yes	
TYPEDEF	Yes	
U	Yes (C/S)	
UCS-4	Yes (C/S)	
UNBOUNDED	Yes (C/S)	
UNDERLINE	Yes (C/S)	
UNFRAMED	Yes (C/S)	
UNIT	Yes	
UNIVERSAL	No	
UNLOCK	Yes	
UNSIGNED	Yes	
UNSIGNED-INT	Yes	
UNSIGNED-LONG	Yes	
UNSIGNED-SHORT	Yes	
UNSORTED	Yes (C/S)	
UNSTRING	Yes	
UNTIL	Yes	
UP	Yes	
UPDATE	Yes	
UPDATERS	Yes (C/S)	
UPON	Yes	
UPPER	Yes (C/S)	
USAGE	Yes	
USE	Yes	
USE-ALT	Yes (C/S)	
USE-RETURN	Yes (C/S)	
USE-TAB	Yes (C/S)	
USER	Yes (C/S)	
USER-DEFAULT	Yes	
USING	Yes	
UTF-16	Yes (C/S)	
UTF-8	Yes (C/S)	
V	Yes (C/S)	
VAL-STATUS	No	
VALID	No	
VALIDATE	Yes	
VALIDATE-STATUS	No	
VALIDATING	Yes (C/S)	
VALUE	Yes	VALUES
VALUE-FORMAT	Yes (C/S)	
VALUES	Yes	VALUE
VARIABLE	Yes (C/S)	
VARIANT	Yes	
VARYING	Yes	
VERTICAL	Yes (C/S)	
VERY-HEAVY	Yes (C/S)	
VIRTUAL-WIDTH	Yes (C/S)	
VOLATILE	Yes	
VPADDING	Yes (C/S)	
VSCROLL	Yes (C/S)	

VSCROLL-BAR	Yes (C/S)	
VSCROLL-POS	Yes (C/S)	
VTOP	Yes (C/S)	
WAIT	Yes	
WEB-BROWSER	Yes (C/S)	
WHEN	Yes	
WIDTH	Yes (C/S)	
WIDTH-IN-CELLS	Yes (C/S)	
WINDOW	Yes	
WITH	Yes	
WORDS	Yes	
WORKING-STORAGE	Yes	
WRAP	Yes (C/S)	
WRITE	Yes	
WRITE-ONLY	Yes (C/S)	
WRITE-VERIFY	Yes (C/S)	
WRITERS	Yes (C/S)	
X	Yes (C/S)	
XML	Yes	
XML-DECLARATION	Yes (C/S)	
XML-SCHEMA	Yes (C/S)	
XOR	No	
Y	Yes (C/S)	
YYYYDDD	Yes (C/S)	
YYYYMMDD	Yes (C/S)	
ZERO	Yes	ZEROES, ZEROS
ZERO-FILL	No (C/S)	
ZEROES	Yes	ZERO, ZEROS
ZEROS	Yes	ZERO, ZEROES

C.2 Extra (obsolete) context sensitive words

AUTHOR, DATE-COMPILED, DATE-MODIFIED, DATE-WRITTEN, INSTALLATION, REMARKS, SECURITY

C.3 Internal registers

Register	Implemented	Definition
'ADDRESS OF' phrase	Yes	USAGE POINTER
COB-CRT-STATUS	Yes	PICTURE 9(4) USAGE DISPLAY VALUE ZERO
DEBUG-ITEM	Yes	PICTURE X(n) USAGE DISPLAY
'LENGTH OF' phrase	Yes	CONSTANT USAGE BINARY-LONG
NUMBER-OF-CALL-PARAMETERS	Yes	USAGE BINARY-LONG
RETURN-CODE	Yes	GLOBAL USAGE BINARY-LONG VALUE ZERO
SORT-RETURN	Yes	GLOBAL USAGE BINARY-LONG VALUE ZERO
TALLY	Yes	GLOBAL PICTURE 9(5) USAGE BINARY VALUE ZERO

WHEN-COMPILED	Yes	CONSTANT PICTURE X(16) USAGE DISPLAY
XML-CODE	Yes	GLOBAL PICTURE S9(9) USAGE BINARY VALUE 0
XML-EVENT	Yes GLOBAL USAGE DISPLAY PICTURE X(30) VALUE SPACE	
XML-INFORMATION	Yes GLOBAL PICTURE S9(9) USAGE BINARY VALUE 0	
XML-NAMESPACE	Yes GLOBAL PIC X ANY LENGTH	
XML-NAMESPACE-PREFIX	Yes GLOBAL PIC X ANY LENGTH	
XML-NNAMESPACE	Yes GLOBAL PIC N ANY LENGTH	
XML-NNAMESPACE-PREFIX	Yes GLOBAL PIC N ANY LENGTH	
XML-NTEXT	Yes GLOBAL PIC N ANY LENGTH	
XML-TEXT	Yes GLOBAL PIC X ANY LENGTH	
JSON-CODE	Yes	GLOBAL PICTURE S9(9) USAGE BINARY VALUE 0
JSON-STATUS	Yes	GLOBAL PICTURE S9(9) USAGE BINARY VALUE 0

Appendix D List of Intrinsic Functions

The following list of intrinsic functions was extracted from cobc --list-intrinsics and shows the names of the available functions, an implementation note and the number of parameters.

Intrinsic Function	Implemented	Parameters
ABS	Yes	1
ACOS	Yes	1
ANNUITY	Yes	2
ASIN	Yes	1
ATAN	Yes	1
BASECONVERT	No	3
BIT-OF	Yes	1
BIT-TO-CHAR	Yes	1
BOOLEAN-OF-INTEGER	No	2
BYTE-LENGTH	Yes	1 - 2
CHAR	Yes	1
CHAR-NATIONAL	No	1
COMBINED-DATETIME	Yes	2
CONCAT	Yes	Unlimited
CONCATENATE	Yes	Unlimited
CONTENT-LENGTH	Yes	1
CONTENT-OF	Yes	1 - 2
CONVERT	No	3
COS	Yes	1
CURRENCY-SYMBOL	Yes	0
CURRENT-DATE	Yes	0
DATE-OF-INTEGER	Yes	1
DATE-TO-YYYYMMDD	Yes	1 - 3
DAY-OF-INTEGER	Yes	1
DAY-TO-YYYYDDD	Yes	1 - 3
DISPLAY-OF	No	1 - 2
E	Yes	0
EXCEPTION-FILE	Yes	0
EXCEPTION-FILE-N	No	0
EXCEPTION-LOCATION	Yes	0
EXCEPTION-LOCATION-N	No	0
EXCEPTION-STATEMENT	Yes	0
EXCEPTION-STATUS	Yes	0
EXP	Yes	1
EXP10	Yes	1
FACTORIAL	Yes	1
FIND-STRING	No	7 - 2
FORMATTED-CURRENT-DATE	Yes	1
FORMATTED-DATE	Yes	2
FORMATTED-DATETIME	Yes	4 - 5
FORMATTED-TIME	Yes	3 - 4
FRACTION-PART	Yes	1
HEX-OF	Yes	1
HEX-TO-CHAR	Yes	1
HIGHEST-ALGEBRAIC	Yes	1

INTEGER	Yes	1
INTEGER-OF-BOOLEAN	No	1
INTEGER-OF-DATE	Yes	1
INTEGER-OF-DAY	Yes	1
INTEGER-OF-FORMATTED-DATE	Yes	2
INTEGER-PART	Yes	1
LENGTH	Yes	1 - 2
LENGTH-AN	Yes	1
LOCALE-COMPARE Yes 2 - 3		
LOCALE-DATE	Yes	1 - 2
LOCALE-TIME	Yes	1 - 2
LOCALE-TIME-FROM-SECONDS	Yes	1 - 2
LOG	Yes	1
LOG10	Yes	1
LOWER-CASE	Yes	1
LOWEST-ALGEBRAIC	Yes	1
MAX	Yes	Unlimited
MEAN	Yes	Unlimited
MEDIAN	Yes	Unlimited
MIDRANGE	Yes	Unlimited
MIN	Yes	Unlimited
MOD	Yes	2
MODULE-CALLER-ID	Yes	0
MODULE-DATE	Yes	0
MODULE-FORMATTED-DATE	Yes	0
MODULE-ID	Yes	0
MODULE-NAME	No	1
MODULE-PATH	Yes	0
MODULE-SOURCE	Yes	0
MODULE-TIME	Yes	0
MONETARY-DECIMAL-POINT	Yes	0
MONETARY-THOUSANDS-SEPARATOR	Yes	0
NATIONAL-OF	No	1 - 2
NUMERIC-DECIMAL-POINT	Yes	0
NUMERIC-THOUSANDS-SEPARATOR	Yes	0
NUMVAL	Yes	1
NUMVAL-C	Yes	2
NUMVAL-F	Yes	1
ORD	Yes	1
ORD-MAX	Yes	Unlimited
ORD-MIN	Yes	Unlimited
PI	Yes	0
PRESENT-VALUE	Yes	Unlimited
RANDOM	Yes	0 - 1
RANGE	Yes	Unlimited
REM	Yes	2
REVERSE	Yes	1
SECONDS-FROM-FORMATTED-TIME	Yes	2
SECONDS-PAST-MIDNIGHT	Yes	0
SIGN	Yes	1
SIN	Yes	1

SQRT	Yes	1
STANDARD-COMPARE	No	2 - 4
STANDARD-DEVIATION	Yes	Unlimited
STORED-CHAR-LENGTH	Yes	1
SUBSTITUTE	Yes	Unlimited
SUBSTITUTE-CASE	Yes	Unlimited
SUM	Yes	Unlimited
TAN	Yes	1
TEST-DATE-YYYYMMDD	Yes	1
TEST-DAY-YYYYDDD	Yes	1
TEST-FORMATTED-DATETIME	Yes	2
TEST-NUMVAL	Yes	1
TEST-NUMVAL-C	Yes	2
TEST-NUMVAL-F	Yes	1
TRIM	Yes	1 - 2
UPPER-CASE	Yes	1
VARIANCE	Yes	Unlimited
WHEN-COMPILED	Yes	0
YEAR-TO-YYYY	Yes	1 - 3

Appendix E System routines

The following list of system routines was extracted from cobc --list-system and shows the names of the available system routines along with the number of parameters.

System routine	Parameters
C$CALLEDBY	1
C$CHDIR	2
C$COPY	3
C$DELETE	2
C$FILEINFO	2
C$GETPID	0
C$JUSTIFY	1 - 2
C$MAKEDIR	1
C$NARG	1
C$PARAMSIZE	1
C$PRINTABLE	1 - 2
C$SLEEP	1
C$TOLOWER	2
C$TOUPPER	2
CBL_ALARM_SOUND	0
CBL_AND	3
CBL_BELL_SOUND	0
CBL_CHANGE_DIR	1
CBL_CHECK_FILE_EXIST	2
CBL_CLOSE_FILE	1
CBL_COPY_FILE	2
CBL_CREATE_DIR	1
CBL_CREATE_FILE	5
CBL_DELETE_DIR	1
CBL_DELETE_FILE	1
CBL_EQ	3
CBL_ERROR_PROC	2
CBL_EXIT_PROC	2
CBL_FLUSH_FILE	1
CBL_GC_FORK	0
CBL_GC_GETOPT	6
CBL_GC_HOSTED	2
CBL_GC_NANOSLEEP	1
CBL_GC_PRINTABLE	1 - 2
CBL_GC_SET_SCR_SIZE	2
CBL_GC_WAITPID	1
CBL_GET_CSR_POS	1
CBL_GET_CURRENT_DIR	3
CBL_GET_SCR_SIZE	2
CBL_IMP	3
CBL_NIMP	3
CBL_NOR	3
CBL_NOT	2
CBL_OC_GETOPT	6
CBL_OC_HOSTED	2

CBL_OC_NANOSLEEP	1
CBL_OPEN_FILE	5
CBL_OR	3
CBL_READ_FILE	5
CBL_READ_KBD_CHAR	1
CBL_RENAME_FILE	2
CBL_RUNTIME_ERROR	2
CBL_SET_CSR_POS	1
CBL_TOLOWER	2
CBL_TOUPPER	2
CBL_WRITE_FILE	5
CBL_XOR	3
EXTFH	2
SYSTEM	1
X"91"	3
X"E4"	0
X"E5"	0
X"F4"	2
X"F5"	2

Appendix F System names

The following list of system names was extracted from cobc --list-mnemonics and shows the system names categorized by their type.

F.1 System names: device

SYSIN, SYSIPT, STDIN, SYSOUT, SYSLIST, SYSLST, SYSPCH, SYSPUNCH, STDOUT, PRINT, PRINTER, PRINTER-1, SYSERR, STDERR, CONSOLE, ALTERNATE-CONSOLE, ALTERNATE CONSOLE

F.2 System names: feature

C01, C02, C03, C04, C05, C06, C07, C08, C09, C10, C11, C12, S01, S02, S03, S04, S05, CSP, FORMFEED, TOP, CALL-CONVENTION

F.3 System names: switch

SWITCH-0, SWITCH-1, SWITCH-2, SWITCH-3, SWITCH-4, SWITCH-5, SWITCH-6, SWITCH-7, SWITCH-8, SWITCH-9, SWITCH-10, SWITCH-11, SWITCH-12, SWITCH-13, SWITCH-14, SWITCH-15, SWITCH-16, SWITCH-17, SWITCH-18, SWITCH-19, SWITCH-20, SWITCH-21, SWITCH-22, SWITCH-23, SWITCH-24, SWITCH-25, SWITCH-26, SWITCH-27, SWITCH-28, SWITCH-29, SWITCH-30, SWITCH-31, SWITCH-32, SWITCH-33, SWITCH-34, SWITCH-35, SWITCH-36

Appendix G Exceptions

The following list of exceptions names was extracted from cobc --list-exceptions and shows the names by type.

Exception Name
EC-ALL
- EC-ARGUMENT
  - EC-ARGUMENT-FUNCTION (f)
  - EC-ARGUMENT-IMP
- EC-BOUND
  - EC-BOUND-FUNC-RET-VALUE
  - EC-BOUND-IMP
  - EC-BOUND-ODO (f)
  - EC-BOUND-OVERFLOW (f)
  - EC-BOUND-PTR (f)
  - EC-BOUND-REF-MOD (f)
  - EC-BOUND-SET (f)
  - EC-BOUND-SUBSCRIPT (f)
  - EC-BOUND-TABLE-LIMIT
  - (f)
- EC-CONTINUE
  - EC-CONTINUE-IMP
  - EC-CONTINUE-LESS-THAN-ZERO
- EC-DATA
  - EC-DATA-CONVERSION
  - EC-DATA-IMP
  - EC-DATA-INCOMPATIBLE (f)
  - EC-DATA-NOT-FINITE (f) EC-
  - DATA-OVERFLOW (f)
  - EC-DATA-PTR-NULL (f)
- EC-EXTERNAL
  - EC-EXTERNAL-DATA-MISMATCH (f)
  - EC-EXTERNAL-FILE-MISMATCH (f)
  - EC-EXTERNAL-FORMAT-CONFLICT (f)
  - EC-EXTERNAL-IMP
- EC-FLOW
  - EC-FLOW-APPLY-COMMIT (f)
  - EC-FLOW-COMMIT (f)
  - EC-FLOW-GLOBAL-EXIT (f)
  - EC-FLOW-GLOBAL-GOBACK (f)
  - EC-FLOW-IMP
  - EC-FLOW-RELEASE (f)
  - EC-FLOW-REPORT (f)
  - EC-FLOW-RETURN (f)
  - EC-FLOW-ROLLBACK (f)
  - EC-FLOW-SEARCH (f)
  - EC-FLOW-USE (f)
- EC-FUNCTION
  - EC-FUNCTION-ARG-OMITTED (f)
  - EC-FUNCTION-IMP
  - EC-FUNCTION-NOT-FOUND (f)

EC-FUNCTION-PTR-INVALID (f)
EC-FUNCTION-PTR-NULL (f)
EC-I-O
EC-I-O-AT-END
EC-I-O-EOP
EC-I-O-EOP-OVERFLOW
EC-I-O-FILE-SHARING EC-
I-O-IMP
EC-I-O-INVALID-KEY
EC-I-O-LINAGE (f)
EC-I-O-LOGIC-ERROR (f)
EC-I-O-PERMANENT-ERROR (f)
EC-I-O-RECORD-CONTENT (f)
EC-I-O-RECORD-OPERATION
EC-I-O-RECORD-WARNING
EC-IMP
EC-IMP-ACCEPT
EC-IMP-DISPLAY
EC-IMP-UTC-UNKNOWN (f)
EC-IMP-FEATURE-DISABLED
EC-IMP-FEATURE-MISSING
EC-LOCALE
EC-LOCALE-IMP
EC-LOCALE-INCOMPATIBLE
EC-LOCALE-INVALID (f)
EC-LOCALE-INVALID-PTR (f)
EC-LOCALE-MISSING (f)
EC-LOCALE-SIZE (f)
EC-MCS
EC-MCS-ABNORMAL-TERMINATION
EC-MCS-IMP
EC-MCS-INVALID-TAG
EC-MCS-MESSAGE-LENGTH
EC-MCS-NO-REQUESTER
EC-MCS-NO-SERVER
EC-MCS-NORMAL-TERMINATION
EC-MCS-REQUESTOR-FAILED
EC-OO
EC-OO-ARG-OMITTED (f)
EC-OO-CONFORMANCE (f)
EC-OO-EXCEPTION (f) EC-
OO-IMP
EC-OO-METHOD (f)
EC-OO-NULL (f)
EC-OO-RESOURCE (f)
EC-OO-UNIVERSAL (f)
EC-ORDER
EC-ORDER-IMP
EC-ORDER-NOT-SUPPORTED (f)
EC-OVERFLOW
EC-OVERFLOW-IMP

EC-OVERFLOW-STRING
EC-OVERFLOW-
UNSTRING
EC-PROGRAM
EC-PROGRAM-ARG-MISMATCH (f)
EC-PROGRAM-ARG-OMITTED (f) EC-
PROGRAM-CANCEL-ACTIVE (f) EC-
PROGRAM-IMP
EC-PROGRAM-NOT-FOUND (f)
EC-PROGRAM-PTR-NULL (f)
EC-PROGRAM-RECURSIVE-CALL (f)
EC-PROGRAM-RESOURCES (f)
EC-RAISING
EC-RAISING-IMP
EC-RAISING-NOT-SPECIFIED (f)
EC-RANGE
EC-RANGE-IMP
EC-RANGE-INDEX (f)
EC-RANGE-INSPECT-SIZE (f)
EC-RANGE-INVALID
EC-RANGE-PERFORM-VARYING (f)
EC-RANGE-PTR (f)
EC-RANGE-SEARCH-INDEX
EC-RANGE-SEARCH-NO-MATCH
EC-REPORT
EC-REPORT-ACTIVE (f)
EC-REPORT-COLUMN-OVERLAP (f)
EC-REPORT-FILE-MODE (f)
EC-REPORT-IMP
EC-REPORT-INACTIVE (f) EC-
REPORT-LINE-OVERLAP EC-
REPORT-NOT-TERMINATED EC-
REPORT-PAGE-LIMIT
EC-REPORT-PAGE-WIDTH
EC-REPORT-SUM-SIZE (f)
EC-REPORT-VARYING (f)
EC-SCREEN
EC-SCREEN-FIELD-OVERLAP
EC-SCREEN-IMP
EC-SCREEN-ITEM-TRUNCATED
EC-SCREEN-LINE-NUMBER
EC-SCREEN-STARTING-COLUMN
EC-SIZE
EC-SIZE-ADDRESS (f)
EC-SIZE-EXPONENTIATION (f)
EC-SIZE-IMP
EC-SIZE-OVERFLOW (f)
EC-SIZE-TRUNCATION (f)
EC-SIZE-UNDERFLOW (f)
EC-SIZE-ZERO-DIVIDE (f)
EC-SORT-MERGE
EC-SORT-MERGE-ACTIVE (f)

- EC-SORT-MERGE-FILE-OPEN (f)
- EC-SORT-MERGE-IMP
- EC-SORT-MERGE-RELEASE (f)
- EC-SORT-MERGE-RETURN (f)
- EC-SORT-MERGE-SEQUENCE (f)

EC-STORAGE
- EC-STORAGE-IMP
- EC-STORAGE-NOT-ALLOC
- EC-STORAGE-NOT-AVAIL

EC-USER

EC-VALIDATE
- EC-VALIDATE-CONTENT
- EC-VALIDATE-FORMAT
- EC-VALIDATE-IMP
- EC-VALIDATE-RELATION
- EC-VALIDATE-VARYING (f)

EC-XML
- EC-XML-CODESET (f)
- EC-XML-CODESET-CONVERSION (f)
- EC-XML-COUNT (f)
- EC-XML-DOCUMENT-TYPE (f)
- EC-XML-IMPLICIT-CLOSE (f)
- EC-XML-INVALID (f)
- EC-XML-NAMESPACE (f)
- EC-XML-STACKED-OPEN (f)
- EC-XML-RANGE (f)
- EC-XML-IMP (f)

EC-JSON
- EC-JSON-IMP (f)

———————————————————

End of Appendix C — Grouped Reserved Word List

Appendix H GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document free in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTEX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modifi- cation. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly

and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY- SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

———————————————————

End of Appendix D — GNU Free Documentation License

Appendix I Summary of Document Changes

GnuCOBOL is an ever-evolving tool. While all reasonable attempts will be made to maintain the currency of the information in this document, neither the author of this document nor the authors of the GnuCOBOL software extend any warranties of any kind for this document or for the information contained therein.

8th Edition on release of v3.2 with some updates for 4.0+. Removed all update notes for 2021 - 23.

1. 2024
2. 12/01 Updated by lefessan, for mis-spelled SYNCHRONISED / SYNCHRONIZED in various places and other stuff not recorded by him. Added NEW file SYN- DD-SYNCHRONIZED.texi and removed SYN-DD-SYNCRONIZED.texi that he missed.
3. 15/01 Updated 6.9.30 No-ECHO and 6.9.42 for error in descriptions and text. 15/01 Updated 7.8.1.4, 7.8.12.4 for CONTROL, COLOUR, COLOUR & GRAPHICS. 15/01 Updated with examples for ANNUITY and PRESENT-VALUE.
4. 30/01 More details for VALUE OF FILE-ID. Update to function x"91" and added options 11-12, 13-14, 15, 35, 46-49, 69.
5. 03/02 In 2.2.9 Relation Conditions added extra text for <>.
6. 06/02 Spurious item for Accept 7/8/1/4 CONTROL (8) removed.
7. 08/02 Description for OCCURS clause details for 01, 66, 77, 88 changed.
8. 25/02 7.8.1.4 ACCEPT -added CURSOR description, misspelling for storage, under SPECIAL-NAMES added extra sentence for CURSOR.
9. 26/02 Added CBL_GC_SET_SCR_SIZE under supported function and added it into System functions list file cbsyst.tex.
10. 27/02 Additions to Appendix C and D and 8.2 for missing functions but a few have no info as nothing is in NEWS worth speaking of as well as the Quick Ref doc.
11. 02/03 Text for Function Random updated plus an example demo of usage.
12. 03/03 In 6.9.61 USAGE layout, tidied up examples using typedef.
13. 09/03 In 8.2.58 EXTFH update syntax with minor description. Minor update to CBL_OPEN_FILE, CBL_CLOSE_FILE referencing where an example of use can be found in the Contribs folder.
14. 14/03 Updated descriptions for functions HIGHEST-ALGEBRAIC, LOWEST-ALGEBRAIC, B-SHIFT-R, B-SHIFT-RC, B-SHIFT-L and B-SHIFT-LC ditto in Quick Reference.
15. 15/03 Move all B-SHIFT to section 2 (as PD's) from Functions - they are not so. Ditto in QR.
16. 16/03 For B-NOT use operand-3 shown separately.
17. 16/04 Tidy up 7.8.1.4 Color and Control phrases a bit.
18. 23/04 Changed details for UNBOUNDED to only apply to LINKAGE SECTION. Updated diagrams for REPOSITORY. Added example for respository from FAQs.
19. 16/05 More chgs for REPOSITORY and prototypes -missed out 23/04.
20. 18/05 Reverted chg on SYN-ED-REPOSITORY.texi back to function and not prototype. Chg sub & section in 2.2 et al #943. Added JSON GENERATE, PARSE, XML GENER- ATE and PARSE. MORE TEXT REQUIRED <<<<<.
21. 20/05 Sub and sub section changes for 2.1.27 onwards.

22. 23/05 Updated text for ACCEPT ident. from LINE or COLUMN.
23. 26/05 Again updated text in .6 ACCEPT.
24. 27/05 In 5.2.1 SELECT option b should refer to IBM not mf, thanks Mark (Ruthe).
25. 03/06 Added note in Appendix A regarding user defined names allowed to be as long as 63 characters. Inline perform - remove optional tags [] as it is required.
26. 05/06 Correct CBL_GET_SCR_SIZE to use BINARY-CHAR UNSIGNED same for all others and they are also changed namely CBL_GC_SET_SCR_SIZE, CBL_GET_SCR_SIZE, CBL_GET_CSR_POS, CBL_SET_CSR_POS.
27. 12/06 at 7.8.8 COMPUTE added for boolean expression and updated text #974.
28. 29/06 Removed B-SHIFT from intrinstic lists. Clean up prime texi'files i.e., gnu-cobpg,pr,qr,sp) removing Appendixes E & F)
29. 04/07 Updated Occurs depending on formats 2 & 3. VERB/END VERB update to last example. TIMEOUT typo, 'tp' should be to. Example of picture P value should be 128000000 and not 000000128. Extra example for comma's showing usage of PICTURE Z. plus correc- tion for first one. Corrected text regarding multiple floating symbols as gnucobol behaviour same as many other compilers. Corrections all thanks to Michael F Gleason.
30. 11/08 Typo's in section 6, 7 & 8 fixed thanks to Michael F Gleason.
31. 06/09 C$SLEEP change + example 2 and removed if/but type grammar as ambiguous. 6.9.50 SIZE add some text :)
32. 13/09 In section 4 for PROGRAM-ID and FUNCTION-ID extra text to warn NOT to use same names as in clib but to change case to upper ONLY, if really needed.
33. 28/09 In 10.2.3.5 Added COB_HIDE_CURSOR, In 7.8.43.1 SET ENVIRONMENT and 7.8.1.3 ACCEPT FROM ENVIRONMENT added extra texts. FR #479. Added routines_CBL_GC_SCR_DUMP & CBL_GC_SCR_RESTORE to 8.2.FR #473.
34. 02/11 In appendix C remove reference to 1.3.15 - no longer exists.
35. 05/11 In Ch 10 for cobc help for –list-reserved, added 'and internal registers'
36. 11/11 In Identification Division within the OPTIONS text there were missing 2 periods and therefore Square braces missing.
37. 16/11 Updated layout for EXAMINE, covering optional elements more clearly.
38. 17/11 Updated Declaratives texts for readability etc.
39. 01/12 Changes for NO-ECHO and BEFORE-TIME clauses in sect 6.
40. 03/12 ACCEPT data-name & DISPLAY data-name, added EXCEPTION|ESCAPE for option 4. Changes to CALL so ALL variants are within one texi file deleting SYN-PD- CALL-arg.texi. For ENTRY ditto for ENTRY-Arg.texi. Text in 7.texi changed accordingly. In 7.4 INPUT used twice for SORT instead of 2nd as OUTPUT.
41. 07/12 Changes to SS-Data-Item, DD-Column-1/2, DD-Line-1/2, DD-TYPE diagrams. New: SPECIAL-NAMES in Data Division with SYN-DD-SPECIAL-NAMES.texi added & SYN-DD-Overview-Tex.texi deleted having changed 6.texi.
42. 09/12 Changes to SYN-PD-OPEN.texi, REWRITE and WRITE adding new text.
43. 13/12 Additions to SYN-DD-SS-Data-Item.texi - missing sub clauses, update to SYN-PD-ACCEPT-4.texi missing clauses. bugs 1033/35 - comment regarding DEFAULT is ignored as not true - default is overwrite. DD-DEFAULT added. Updated SYN-PD-DISPLAY- 4.texi.
44. 12/24 INCLUDE added to CDF COPY clause. 78 NEXT added + example program.
45. 2025

46. 08/01 Removed all update notes for period 2021 to 2023. Removed references in section 6 for NO UPDATE and UPDATE as not used within SCREEN SECTION, ditto files SYN-DD- UPDATE.texi amd -NO-UPDATE.texi. Remove NO UPDATE from ACCEPT data-item
    (4) also change text for UPDATE as section 6 ref removed. Chg to SYN-DD-SS-Data-Item.texi.
47. 11/01 Minor textual change to ACCEPT Data-Item for UPDATE and DEFAULT - see as on 08/01.
48. 13/01 CDF >>DEFINE OFF has an additional sentence added.
49. 22/01 Removed SCROLL UP / DOWN as not available in screen section in sections 6 & 7 points for ACCEPT.
50. 23/01 Removed reference to OPENCOBOL in chapter 10. In 10.2.3 included def. for "OS". Removed incorrect [ ] in Chap 6 - there are a lot more though in other chapters :(.
51. 05/02 Updated SYN-PD-ACCEPT-4 and DISPLAY-4.

Index

"

" (Quote) 33

$

$ Directives 79

&

& (Literal Concatenation) 34

'

' (Apostrophe) 33

*

* (Multiplication) 40
* In Column 7 (Comment) 32
** (Exponentiation) 40
*> (Comment) 32

+

+ (Addition) 41
+ (Unary Sign Retention) 40

,

, (Punctuation) 35

-

- (Character in Words/Names) 10
- (Subtraction) 41
- (Unary Sign Reversal) 40
- In Column 7 (Continuation) 34
-b Compiler Switch 722
-conf Compiler Switch 691
-debug Compiler Switch 251, 453, 454, 541
-fdebugging-line Compiler Switch 87, 251
-ffold-call Compiler Switch 699
-ffold-copy Compiler Switch 70, 690
-fintrinsics Compiler Switch 425
-fintrinsics=ALL Compiler Switch 99
-fixed Compiler Switch 28, 70, 72
-foptional-file Compiler Switch 103
-free Compiler Switch 28, 70, 72
-fsyntax-extension Compiler Switch 97
-ftrace Compiler Switch 369, 371, 700
-ftraceall Compiler Switch 371, 453, 454, 541, 700
-g Compiler Switch 453, 454, 541
-I Compiler Switch 690
-m Compiler Switch 684, 698, 721
-o Compiler Switch 684, 698
-O Compiler Switch 718
-O2 Compiler Switch 718
-Os Compiler Switch 718
-Wobsolete Compiler Switch 82
-x Compiler Switch 683, 698, 721, 757

.

. (Punctuation) 35

/

/ (Division) 40
/ In Column 7 (Comment) 32

;

; (Punctuation) 35

>

>>CALL-CONVENTION 62
>>D 74
>>D (Debugging Line) 33
>>DEFINE 67
>>DISPLAY 75
>>ELIF 69
>>ELSE 69
>>END-IF 69
>>IF 68
>>LEAP-SECONDS 78
>>LISTING 77
>>PAGE 76
>>SET 70
>>SOURCE 72
>>TURN 73

_

_ (Character in user-defined words) 10

0

01-Level Constants 145

6

66-Level Data Items 149

7

77-Level Data Items 150
78-Level Data Items 151

8

88-Level Data Items 153

A

A Sample Screen ... 21
ABS ... 429
ACCEPT ... 263
ACCEPT data-item ... 267
ACCEPT FROM COMMAND-LINE ... 264
ACCEPT FROM CONSOLE ... 263
ACCEPT FROM DATE/TIME ... 277
ACCEPT FROM ENVIRONMENT ... 266
ACCEPT FROM EXCEPTION STATUS ... 281
ACCEPT FROM Runtime-Info ... 279
ACCEPT FROM Screen-Info ... 278
ACCEPT OMITTED ... 280
ACCESS MODE DYNAMIC ... 112, 115
ACCESS MODE RANDOM ... 112, 115
ACCESS MODE SEQUENTIAL ... 109, 110, 114
ACOS ... 430
ADD CORRESPONDING ... 286
ADD GIVING ... 284
ADD TO ... 282
Additional Reference Sources ... 3
ADD ... 282
ADDRESS OF ... 381
ADVANCING PAGE ... 411
AFTER ... 361
AFTER ADVANCING ... 411
AFTER EXCEPTION CONDITION ... 250
ALL ... 99, 240, 406
ALL INTRINSIC ... 99
ALL OTHER ... 102
ALLOCATE ... 287
Alphabet-Name-Clause ... 94
Alphabetic Data Item ... 755
Alphabetic Data Items ... 195
ALPHABET ... 94
ALPHABETIC ... 339
Alphanumeric Data Item ... 755
Alphanumeric Data Items ... 195
Alphanumeric Literal ... 755
Alphanumeric Literal (Hexadecimal) ... 34
Alphanumeric Literal (Zero-Delimited) ... 34
Alphanumeric Literals ... 33
ALPHANUMERIC ... 339
ALPHANUMERIC-EDITED ... 339
ALSO ... 66, 322
Alternate Entry Points ... 720
ALTER ... 289
ALTERNATE RECORD KEY ... 115
An Example ... 656
ANNUITY ... 431
ANY ... 322
ANY LENGTH ... 154
ANY NUMERIC ... 155
Area 'A' (Columns 8-11) ... 29
Area 'B' (Columns 12-72) ... 29
ARGUMENT-NUMBER ... 264, 303
ARGUMENT-VALUE ... 264
Arithmetic Expressions ... 40
ASCENDING KEY ... 377
ASCII ... 94
ASIN ... 433
AT END ... 376
AT END + NOT AT END ... 253
ATAN ... 434
AUTHOR ... 81
AUTO ... 156
AUTO-SKIP ... 157
AUTOTERMINATE ... 158

B

BACKGROUND-COLOR ... 159
BASED ... 161
BEEP ... 162
BEFORE ADVANCING ... 411
BEFORE TIME ... 160
BELL ... 163
Binary Truncation ... 715
BIT-OF ... 435
BIT-TO-CHAR ... 436
BLANK ... 164
BLANK WHEN ZERO ... 165
BLINK ... 166
BLOCK CONTAINS ... 123
Boolean Expressions ... 43
BOOLEAN-OF-INTEGER ... 537
Built-In System Subroutines ... 546
BY CONTENT ... 293, 725, 726
BY REFERENCE ... 244, 292, 725, 726
BY VALUE ... 244, 293, 726
BYTE-LENGTH ... 145, 437

C

C Main Programs Calling GnuCOBOL Subprograms ... 733
C$CALLEDBY ... 548
C$CHDIR ... 549
C$COPY ... 550
C$DELETE ... 551
C$FILEINFO ... 552
C$GETPID ... 553
C$JUSTIFY ... 554
C$MAKEDIR ... 555
C$NARG ... 556
C$PARAMSIZE ... 557
C$PRINTABLE ... 558
C$SLEEP ... 559
C$TOLOWER ... 561
C$TOUPPER ... 562
Call Environment ... 705
Called Program ... 719, 755
Called Program Considerations ... 726
Calling Program ... 719, 755
Calling Program Considerations ... 725
CALL ... 290
CALL-CONVENTION ... 91
CANCEL ... 294
Case Insensitivity ... 10
CBL_ALARM_SOUND ... 563
CBL_AND ... 564
CBL_BELL_SOUND ... 565
CBL_CHANGE_DIR ... 566
CBL_CHECK_FILE_EXIST ... 567
CBL_CLOSE_FILE ... 568
CBL_COPY_FILE ... 569

CBL_CREATE_DIR 570
CBL_CREATE_FILE 571
CBL_DELETE_DIR 572
CBL_DELETE_FILE 573
CBL_EQ 574
CBL_ERROR_PROC 575
CBL_EXIT_PROC 577
CBL_FLUSH_FILE 579
CBL_GC_FORK 580
CBL_GC_GETOPT 582
CBL_GC_HOSTED 584
CBL_GC_NANOSLEEP 587
CBL_GC_PRINTABLE 588
CBL_GC_SCR_DUMP 589
CBL_GC_SCR_RESTORE 591
CBL_GC_SET_SCR_SIZE 592
CBL_GC_WAITPID 593
CBL_GET_CSR_POS 594
CBL_GET_CURRENT_DIR 595
CBL_GET_SCR_SIZE 596
CBL_IMP 597
CBL_NIMP 598
CBL_NOR 599
CBL_NOT 600
CBL_OC_GETOPT 601
CBL_OC_HOSTED 602
CBL_OC_NANOSLEEP 603
CBL_OPEN_FILE 604
CBL_OR 605
CBL_READ_FILE 606
CBL_READ_KBD_CHAR 607
CBL_RENAME_FILE 608
CBL_RUNTIME_ERROR 609
CBL_SET_CSR_POS 610
CBL_TOLOWER 611
CBL_TOUPPER 612
CBL_WRITE_FILE 613
CBL_XOR 614
CDF - Compiler Directing Facility 61
CHAR 438
CHAR-NATIONAL 538
Class Conditions 45
Class-Definition-Clause 96
CLASSIFICATION 88
CLOSE 295
COB-CRT-STATUS 91
COB_CC 687
COB_CFLAGS 687
COB_CONFIG_DIR 688
COB_CONFIG_DIR Environment Variable 691
COB_COPY_DIR 688
COB_CPY_DIR Environment Variable 690
COB_DISPLAY_WARNINGS 700
COB_LDADD 688
COB_LDFLAGS 688
COB_LIBRARY_PATH 700
COB_LIBRARY_PATH
Environment Variable 699, 722
COB_LIBS 688
COB_LOAD_CASE 700
COB_LOAD_CASE Environment Variable 722
COB_PHYSICAL_CANCEL 700
COB_PHYSICAL_CANCEL
Environment Variable 294, 722
COB_PRE_LOAD 700
COB_PRE_LOAD Environment Variable 292, 722
COB_SCREEN_ESC 701
COB_SCREEN_ESC Environment Variable 270
COB_SCREEN_EXCEPTIONS 701
COB_SCREEN_EXCEPTIONS
Environment Variable 270
COB_SET_DEBUG 700
COB_SET_DEBUG Environment Variable 251
COB_SET_TRACE 700
COB_SET_TRACE Environment Variable 369, 371
COB_SORT_MEMORY 701
COB_SORT_MEMORY Environment Variable 389
COB_SWITCH_n 701
COB_SWITCH_n Environment Variable 97
COB_SYNC 701
COB_TRACE_FILE 701
cobc - The GnuCOBOL Compiler 673
cobc option -Xref an example 684
COBCPY 688
COBCPY Environment Variable 690
cobcrun - Command-line Execution 698
COBOL Fundamentals 9
CODE-SET 124
Collating Sequence 755
COLLATING SEQUENCE 88, 102, 352
Color Palette 24
Color Palette and Video Attributes 21
COLOUR 309
Column 7 (Indicator Area) 28
Columns 1-6 (Sequence Number Area) 28
Columns 12-72 (Area B) 29
Columns 73-80 (Program Name Area) 29
Columns 8-11 (Area A) 29
COLUMN 167
COLUMNS 278
Combined Conditions 51
COMBINED-DATETIME 439
Combining GnuCOBOL and C Programs 730
COMMAND-LINE 264
Comments 32
COMMIT 296
Common Clauses on Executable Statements 253
COMMON 82
Compilation Group 683, 755
Compilation Time Environment Variables 687
Compilation Unit 683, 755
Compiler Configuration Files 691
Compiler Switches, -b 722
Compiler Switches, -conf 691
Compiler Switches, -debug 251, 453, 454, 541
Compiler Switches, -fdebugging-line 87, 251
Compiler Switches, -ffold-call 699
Compiler Switches, -ffold-copy 70, 690
Compiler Switches, -fintrinsics 425
Compiler Switches, -fintrinsics=ALL 99
Compiler Switches, -fixed 28, 70, 72
Compiler Switches, -foptional-file 103
Compiler Switches, -free 28, 70, 72
Compiler Switches, -fsyntax-extension 97
Compiler Switches, -ftrace 369, 371, 700

Compiler Switches, -ftraceall . . 371, 453, 454, 541, 700
Compiler Switches, -g ... 453, 454, 541
Compiler Switches, -I ... 690
Compiler Switches, -m ... 684, 698, 721
Compiler Switches, -o ... 684, 698
Compiler Switches, -O ... 718
Compiler Switches, -O2 ... 718
Compiler Switches, -Os ... 718
Compiler Switches, -Wobsolete ... 82
Compiler Switches, -x ... 683, 698, 721, 757
Compiling Programs ... 673
COMPUTE ... 297
COMPUTE Versus ADD-SUBTRACT-MULTIPLY-DIVIDE ... 742
CONCAT ... 440
CONCATENATE ... 441
Concurrent Access to Files ... 56
Condition ... 748
Condition Names ... 44, 153
Conditional Expressions ... 44
CONFIGURATION SECTION ... 86
CONSOLE ... 92
CONSOLE IS CRT ... 91
CONSTANT ... 67, 70, 126, 169
Contained Subprograms ... 719
CONTENT ... 320
CONTENT-LENGTH ... 442
CONTENT-OF ... 443
Continuation (- in Column 7) ... 34
CONTINUE ... 300
Control Break ... 656, 755
Control Field ... 755
Control Footing ... 755
Control Heading ... 756
Control Hierarchy ... 654, 756
Control Hierarchy (Revisited) ... 668
CONTROL ... 137, 273, 306
CONVERSION ... 268, 306
CONVERTING ... 344
Copybook ... 13, 756
Copybook Naming Conventions and Usage ... 741
Copybooks ... 13
COPY ... 63
CORRESPONDING ... 254
COS ... 444
COUNT ... 407
Cross Reference listing using cobxref ... 686
CRT STATUS ... 91
CRT STATUS Codes ... 270
CURRENCY SIGN ... 91
CURRENCY-SYMBOL ... 445
current character pointer ... 397
CURRENT-DATE ... 446
CURSOR ... 269
CURSOR IS ... 91

D

D In Column 7 (Debugging Line) ... 33
Data ... 656
Data Definition Principles ... 120
Data Description Clauses ... 154
Data Initialization ... 25
Data Item ... 13, 756
Data Item Coding and Naming Conventions ... 738
DATA DIVISION ... 119
DATA RECORD ... 123
DataTypes ... 748
Date Format ... 748
DATE-COMPILED ... 81
DATE-MODIFIED ... 81
DATE-OF-INTEGER ... 447
DATE-TO-YYYYMMDD ... 448
DATE-WRITTEN ... 81
DAY-OF-INTEGER ... 449
DAY-TO-YYYYDDD ... 450
DB HOME ... 701
DB HOME Environment Variable ... 58
DEBUG-ITEM Special Register ... 251
DEBUGGING MODE ... 87
DECIMAL POINT IS COMMA ... 91
DECLARATIVES ... 250
DEFAULT ... 170, 269, 340
DEFINED ... 69
DELETE ... 301
DELIMITED BY ... 397, 406
DELIMITER ... 407
DEPENDING ON ... 192
DESCENDING KEY ... 377
Detail Group ... 756
detail report ... 333
Detail Report ... 756
Direct Execution ... 698
DISPLAY ... 302
DISPLAY data-item ... 305
DISPLAY data-item (Microsoft v1-v2) ... 312
DISPLAY UPON COMMAND-LINE ... 303
DISPLAY UPON device ... 302
DISPLAY UPON ENVIRONMENT-NAME ... 304
DISPLAY-OF ... 539
DIVIDE ... 313
DIVIDE BY GIVING ... 317
DIVIDE INTO ... 313
DIVIDE INTO GIVING ... 315
Division ... 756
Divisions ... 31
Divisions Organize Programs ... 13
DUPLICATES ... 351, 388, 392
Dynamic Subprogram ... 721, 756
Dynamic vs Static Subprograms ... 721
Dynamically Loaded Subprograms ... 699

E

E ... 451
Elementary Item ... 14, 756
ELSE ... 338
EMPTY-CHECK ... 171
END-OF-PAGE ... 411
Entry-point ... 756
Entry-point Name ... 757
ENTRY ... 319
Environment Variables, COB_CONFIG_DIR ... 691
Environment Variables, COB_CPY_DIR ... 690
Environment Variables,
COB_LIBRARY_PATH ... 699, 722
Environment Variables, COB_LOAD_CASE ... 722
Environment Variables,
COB_PHYSICAL_CANCEL ... 294, 722
Environment Variables, COB_PRE_LOAD ... 292, 722
Environment Variables, COB_SCREEN_ESC ... 270
Environment Variables,
COB_SCREEN_EXCEPTIONS ... 270
Environment Variables, COB_SET_DEBUG ... 251
Environment Variables,
COB_SET_TRACE ... 369, 371
Environment Variables,
COB_SORT_MEMORY ... 389
Environment Variables, COB_SWITCH_n ... 97
Environment Variables, COBCPY ... 690
Environment Variables, DB_HOME ... 58
Environment Variables, LANG ... 445, 496, 497, 498, 499
Environment Variables, LD_LIBRARY_PATH ... 683
Environment Variables, PATH ... 722
Environment Variables, TEMP ... 125, 390
Environment Variables, TMP ... 125, 390
Environment Variables, TMPDIR ... 125, 390
Environment Variables: Compilation-Time ... 687
Environment Variables: Run-Time ... 700
ENVIRONMENT ... 266
ENVIRONMENT DIVISION ... 85
ENVIRONMENT-NAME ... 266
ENVIRONMENT-VALUE ... 266
EOL ... 172
EOS ... 172
ERASE ... 172
Error Exception Codes ... 455
Error Type Strings ... 455
ERROR ... 395
ESCAPE KEY ... 278
EVALUATE ... 321
EXAMINE ... 325
EXCEPTION STATUS ... 279
EXCEPTION-FILE ... 452
EXCEPTION-FILE-N ... 540
EXCEPTION-LOCATION ... 453
EXCEPTION-LOCATION-N ... 541
EXCEPTION-STATEMENT ... 454
EXCEPTION-STATUS ... 455
Exceptions ... 795
Executable File ... 757
Executing Dynamically-Loadable Libraries ... 698
Execution Thread ... 757
EXHIBIT ... 327
EXIT ... 329
EXP ... 457
EXP10 ... 458
EXTEND ... 359
EXTERNAL ... 173
EXTERNAL Data Items ... 727
EXTFH ... 615

F

FACTORIAL ... 459
FALSE ... 174
Figurative Constants ... 34, 757
File I/O ... 706
File I/O Environment Variables
and/or dictionary file ... 714
File OPEN Modes ... 359
File Sharing ... 56
File Status Codes ... 105
File-Based SORT ... 388
File/Sort-Description ... 123
FILE SECTION ... 122
FILE STATUS ... 105
FILE-SECTION-Data-Item ... 126
Files ... 14
FILLER ... 121
FINAL ... 137
FIRST DETAIL ... 136
Fixed Format Mode ... 27, 72, 757
FOLDCOPYNAME ... 70
FOOTING ... 136
FOOTING AT ... 124
FOREGROUND-COLOR ... 175
FOREVER ... 360
Format of Program Source Lines ... 27
FORMATTED-CURRENT-DATE ... 460
FORMATTED-DATE ... 461
FORMATTED-DATETIME ... 462
FORMATTED-TIME ... 463
FRACTION-PART ... 464
Free Format Mode ... 27, 72, 757
FREE ... 332
FROM ... 176, 370, 373, 410
FROM CRT ... 268
FULL ... 177
FUNCTION ... 99
FUNCTION-ID ... 82
Functions ... 425

G

General Environment ... 703
General instructions ... 702
Generated Report Pages ... 662
GENERATE ... 333
GIVING ... 352, 390, 395, 400
GLOBAL ... 178
GLOBAL Data Items ... 726
Glossary of Terms ... 755
GNU Free Documentation License ... 799
GnuCobol use SQL for files ... 745
GnuCOBOL Compile time Options ... 746
GnuCOBOL Main Programs
CALLing C Subprograms ... 732

GnuCOBOL Run-Time Library Requirements . . . 730
GnuCOBOL Statements ... 263
GO TO ... 335
GO TO DEPENDING ON ... 336
GOBACK ... 334
Goto ... 747
GRAPHICS ... 307
Group Item ... 14, 757
GROUP INDICATE ... 179
Grouped Word Lists by feature and function ... 765

H

Header line ... 747
HEADING ... 136
HEX-OF ... 465
HEX-TO-CHAR ... 466
Hexadecimal Alphanumeric Literal ... 34, 757
Hexadecimal Numeric Literal ... 33, 758
HIGHEST-ALGEBRAIC ... 467
HIGHLIGHT ... 180
How RWCS Builds Report Pages ... 653

I

I-O ... 359
IDENTIFICATION DIVISION ... 81
Identifiers ... 758
IF ... 338
Imperative Statement ... 758
Independent Subprograms ... 719
Independent vs Contained vs
Nested Subprograms ... 719
INDEXED BY ... 193, 377
Indexing ... 739
Indicator Area (Column 7) ... 28
INITIAL ... 82, 723
INITIALIZE ... 339
INITIALIZED ... 287
INITIATE ... 343
Inline PERFORM ... 362
INPUT ... 359
INPUT PROCEDURE ... 389
INPUT-OUTPUT SECTION ... 101
INSPECT ... 344
INSTALLATION ... 81
INTEGER ... 468
INTEGER-OF-BOOLEAN ... 542
INTEGER-OF-DATE ... 469
INTEGER-OF-DAY ... 470
INTEGER-OF-FORMATTED-DATE ... 471
INTEGER-PART ... 472
Interfacing to Other Environments ... 35
Interfacing With The OS ... 673
INTO ... 366, 368, 372
Intrinsic Function ... 758
Intrinsic Functions ... 425
INTRINSIC ... 99
Introducing COBOL ... 4
Introduction ... 3
INVALID KEY + NOT INVALID KEY ... 255

J

JSON GENERATE ... 349
JSON PARSE ... 350
JUSTIFIED ... 181

K

Key definition ... 749
KEY ... 193, 352, 368

L

Label ... 747
LABEL RECORD ... 123
LANG Environment Variable ... 445, 496, 497, 498, 499
Language Reserved Words ... 9
LAST CONTROL ... 136
LAST DETAIL ... 136
LD_LIBRARY_PATH ... 688
LD_LIBRARY_PATH Environment Variable ... 683
LEADING ... 64, 66, 212, 532
LEFTLINE ... 183
LENGTH ... 145, 473
LENGTH-AN ... 474
LENGTH-CHECK ... 184
Level Number ... 758
LINAGE IS n LINES ... 124
LINAGE-COUNTER Special Register ... 124, 411
LINE ... 185
LINE ADVANCING ... 15
LINE-COUNTER ... 214
LINE-COUNTER Special Register ... 137, 343, 402, 654
LINES ... 278
LINES AT BOTTOM ... 124
LINES AT TOP ... 124
LINKAGE SECTION ... 132
List of Intrinsic Functions ... 787
Literal ... 758
Literal Concatenation (&) ... 34
Literals ... 33
Literals (Alphanumeric) ... 33
Literals (Numeric) ... 33
LOCAL-STORAGE SECTION ... 130
LOCALE ... 91
LOCALE Names ... 91
LOCALE-COMPARE ... 475
LOCALE-DATE ... 476
LOCALE-TIME ... 477
LOCALE-TIME-FROM-SECONDS ... 478
Locating Copybooks ... 690
LOCK ... 108, 295
LOCK ON ... 58
LOG ... 479
LOG10 ... 480
LOWER ... 70, 187
LOWER-CASE ... 481
LOWEST-ALGEBRAIC ... 482
LOWLIGHT ... 188

M

Main program ... 758
Main Program ... 719
Marking Changes in Programs ... 737
Matching C Data Types with GnuCOBOL USAGE's ... 731
MAX ... 483
MEAN ... 484
MEDIAN ... 485
MEMORY SIZE ... 88
MERGE ... 351
MIDRANGE ... 486
MIN ... 487
MOD ... 488
MODE IS BLOCK ... 268
MODULE-CALLER-ID ... 489
MODULE-DATE ... 490
MODULE-FORMATTED-DATE ... 491
MODULE-ID ... 492
MODULE-PATH ... 493
MODULE-SOURCE ... 494
MODULE-TIME ... 495
MONETARY-DECIMAL-POINT ... 496
MONETARY-THOUSANDS-SEPARATOR ... 497
MOVE ... 354
MOVE CORRESPONDING ... 355
multiple record locking ... 58
MULTIPLE FILE ... 117
MULTIPLY ... 356
MULTIPLY BY ... 356
MULTIPLY GIVING ... 357

N

National Character set ... 195, 758
NATIONAL ... 339
NATIONAL-EDITED ... 340
NATIONAL-OF ... 543
NATIVE ... 94
Negated Conditions ... 52
Nested Subprograms ... 720
NEXT ... 189, 365
NEXT GROUP ... 189
NEXT PAGE ... 185, 189
NEXT SENTENCE ... 338
NO ADVANCING ... 302
NO REWIND ... 295, 358
NO-ECHO ... 190
NOFOLDCOPYNAME ... 70
NORMAL ... 395
NOT ... 52
NOT INVALID KEY ... 256
NOT ON EXCEPTION ... 256
NOT ON OVERFLOW ... 257
NOT ON SIZE ERROR ... 257
Null Terminated Literals ... 33
Numeric Data Item ... 759
Numeric Data Items ... 195
Numeric Edited ... 197
Numeric Edited Data Item ... 759
Numeric Literal ... 759
Numeric Literal (Hexadecimal) ... 33
Numeric Literals ... 33
NUMERIC ... 339
NUMERIC SIGN TRAILING SEPARATE ... 92
NUMERIC-DECIMAL-POINT ... 498
NUMERIC-EDITED ... 339
NUMERIC-THOUSANDS-SEPARATOR ... 499
NUMVAL ... 500
NUMVAL-C ... 501
NUMVAL-C-2 ... 502
NUMVAL-F ... 503

O

OBJECT-COMPUTER ... 88
OCCURS ... 191
OFF ... 67
OFF STATUS ... 97
OMITTED ... 245
ON EXCEPTION + NOT ON EXCEPTION ... 256
ON OVERFLOW + NOT ON OVERFLOW ... 256
ON SIZE ERROR + NOT ON SIZE ERROR ... 257
ON STATUS ... 97
OPEN ... 358
OPTIONAL ... 103, 245
ORD ... 504
ORD-MAX ... 505
ORD-MIN ... 506
ORGANISATION INDEXED ... 16
ORGANISATION LINE SEQUENTIAL ... 14
ORGANISATION RELATIVE ... 16
ORGANISATION SEQUENTIAL ... 15
ORGANIZATION INDEXED ... 114
ORGANIZATION LINE SEQUENTIAL ... 110
ORGANIZATION RELATIVE ... 112
ORGANIZATION SEQUENTIAL ... 109
OS ... 701
OUTPUT ... 359
OUTPUT PROCEDURE ... 352, 390
overflow condition ... 398
OVERLINE ... 194
OVERRIDE ... 67

P

Page Footing ... 759
Page Heading ... 759
PAGE ... 189
PAGE LIMITS ... 136
PAGE-COUNTER ... 214
PAGE-COUNTER Special Register ... 137, 343, 402
PARAMETER ... 67
PATH ... 702
PATH Environment Variable ... 722
perform scope ... 360
PERFORM ... 360
PI ... 507
PICTURE ... 141, 195
POINTER ... 397
Predefined Compilation Variables ... 688
PRESENT WHEN ... 202
PRESENT-VALUE ... 508
PREVIOUS ... 365
Primary Entry-Point ... 759
PRINTER ... 93

PRINTING 63
Procedural PERFORM 360
Procedure 759
Procedure name 759
Procedure Names 249
PROCEDURE DIVISION 243
PROCEDURE DIVISION CHAINING 246
PROCEDURE DIVISION RETURNING 248
PROCEDURE DIVISION Sections
and Paragraphs 249
PROCEDURE DIVISION Sections
Versus Paragraphs 741
PROCEDURE DIVISION USING 244
Procedures 249
Program 658, 759
Program Arguments 715
Program Name Area (Columns 73-80) 29
Program Structure 30
PROGRAM-ID 82
Programmer Productivity 6
Programming for XFD 745
Programming Style Suggestions 737
PROMPT 203
PROTECTED 204
Punctuation 35

Q

Qualification 759
Qualification of Data Names 38

R

Random READ 367
RANDOM 510
RANGE 512
Readability of Programs 11
READ 365
READY TRACE 369
Record 126, 759
Record Locking 58
RECORD CONTAINS 124
RECORD DELIMITER 102
RECORD IS VARYING 124
RECORD KEY 115, 301
RECORDING MODE 123
Recursive Subprogram 728
Recursive Subprograms 728
RECURSIVE 82
REDEFINES 205
REEL 295
Reference Modifiers 39
REFERENCE 320
Relation Conditions 49
RELATIVE KEY 112, 301
RELEASE 370
REM 513
REMAINDER 315, 317
REMARKS 81
RENAMES 206
REPLACE 65
REPLACING 64, 65, 340, 344
Report Footing 760
Report Group 760
Report Group Definitions 138
Report Heading 760
Report I/O 713
Report Writer Features 24
Report Writer Usage 651
REPORT IS 124
REPORT SECTION 135
REPORT SECTION Data Items 140
REPOSITORY 99
REQUIRED 207
Reserved Word 760
Reserved Word List 763
Reserved Words 9, 765
RESERVE 102
RESET 216
RESET TRACE 371
RETURN 372
RETURN-CODE Special Register . . 291, 395, 546, 548,
549, 550, 551, 552, 553, 555, 557, 564, 566, 567, 568,
569, 570, 571, 572, 573, 574, 575, 577, 595, 597, 598,
599, 600, 604, 605, 606, 608, 613, 614, 719, 723, 724
RETURNING 287, 293, 395, 723
REVERSE 514
REVERSE-VIDEO 208
REVERSED 358
REWRITE 373
ROLLBACK 375
ROUNDED 257
Run Time Environment Variables 700
Running Programs 698
RUN 395
RWCS Lexicon 651

S

SAME AS 209
SAME RECORD AREA 116
SAME SORT 116
SAME SORT-MERGE 116
Screen Formatting Features 20
Screen I/O 711
SCREEN SECTION 142
SCROLL 269
Search Index 376
SEARCH 376
SEARCH ALL 377
SECONDS-FROM-FORMATTED-TIME 515
SECONDS-PAST-MIDNIGHT 516
SECURE 211
SECURITY 81
SEGMENT-LIMIT 88
SELECT 102
Sentence 53, 760
SEPARATE CHARACTER 212
Sequence Number Area (Columns 1-6) 28
Sequential READ 365
SET 69, 379
SET ADDRESS 381
SET ATTRIBUTE 386
SET Condition Name 384
SET ENVIRONMENT 379
SET Index 382

SET LAST EXCEPTION 387
SET Program-Pointer 380
SET Switch 385
SET UP/DOWN 383
Sharing Data Between Calling and Called Programs 725
SHARING 108, 359
Sign Conditions 47
SIGN 517
SIGN IS 212
Simple GO TO 335
Simple MOVE 354
single record locking 58
SIN 518
SIZE 213
So What is GnuCOBOL? 7
Sorting and Merging Data 18
SORT 388
SORT STATUS 105
Source Line Format, Fixed 27, 72
Source Line Format, Free 27, 72
SOURCE 214
SOURCE IS 115
SOURCE-COMPUTER 87
Sparse Keys 115
Special Data Items 145
Special Registers 260, 760
Special Registers, DEBUG-ITEM 251
Special Registers, LINAGE-COUNTER 124, 411
Special Registers, LINE-COUNTER 137, 343, 402, 654
Special Registers, PAGE-COUNTER 137, 343, 402
Special Registers, RETURN-CODE 291, 395, 546, 548, 549, 550, 551, 552, 553, 555, 557, 564, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 577, 595, 597, 598, 599, 600, 604, 605, 606, 608, 613, 614, 719, 723, 724
SPECIAL-NAMES 90
SPECIAL-NAMES. 215
Split Keys 115
SQRT 519
STANDARD-1 94
STANDARD-2 94
STANDARD-COMPARE 544
STANDARD-DEVIATION 520
START 393
Statement 53, 760
Static Subprogram 721, 760
STATIC 291
STATUS 395
STDCALL 291
STDERR 93
STDIN 92
STDOUT 93
STEP 193
STOP 395
STORED-CHAR-LENGTH 521
String Allocation Differences Between GnuCOBOL and C 730
String Manipulation 18
STRING 397
Structured Data 13
Sub-Programming 719
Subprogram 719, 760
Subprogram Arguments 725
Subprogram Execution Flow 722
Subprogram Types 719
Subroutine 719, 760
Subroutine Execution Flow 722
Subscripting 739
SUBSTITUTE 522
SUBSTITUTE-CASE 523
SUBTRACT 399
SUBTRACT CORRESPONDING 401
SUBTRACT FROM 399
SUBTRACT GIVING 400
Summary of Document Changes 805
Summary Report 760
summary report 333
SUM 524
SUM OF 216
SUPPRESS 63, 402
SUPPRESS WHEN 115
Switch-Definition-Clause 97
Switch-Status Conditions 48
SWITCH-n 97
SWn 97
Symbolic-Characters-Clause 98
SYNCHRONIZED 217
Syntax Diagram Conventions 26
SYSERR 93
SYSIN 92
SYSIPT 92
SYSLIST 93
SYSLST 93
SYSOUT 93
System names 793
System routines 791
SYSTEM 644

T

Table Handling 17
Table References 37
Table SORT 392
Table Subscripting versus Table Indexing 739
TALLYING 344
TAN 525
TEMP 702
TEMP Environment Variable 125, 390
TERMINATE 403
TEST-DATE-YYYYMMDD 526
TEST-DAY-YYYYDDD 527
TEST-FORMATTED-DATETIME 528
TEST-NUMVAL 529
TEST-NUMVAL-C 530
TEST-NUMVAL-F 531
The Anatomy of a Report 652
The Anatomy of a Report Page 652
The COBOL Language - The Basics 9
THRU 322
TIME OUT 219
TIMEOUT 269
TIMES 360
TMP 688, 702
TMP Environment Variable 125, 390
TMPDIR 688, 702

TMPDIR Environment Variable 125, 390
TO 220
TRAILING 64, 66, 212, 532
TRANSFORM 404
TRIM 532
Turning PHYSICAL Page Formatting Into LOGICAL Formatting 670
TYPE 221
TYPEDEF 225

U

UNDERLINE 226
UNLOCK 405
UNSIGNED 236
UNSTRING 406
UNTIL 360
UNTIL EXIT 360
UPDATE 269
UPON 216, 302
UPON CRT 306
UPON CRT-UNDER 306
UPPER 70, 227
UPPER-CASE 533
USAGE 228
Use of Periods 53
Use of VERB/END-VERB Constructs 54
USE AFTER STANDARD ERROR PROCEDURE 252
USE BEFORE REPORTING 251
USE FOR DEBUGGING 251
User-Defined Function 719, 760
User-Defined Function Execution Flow 723
User-Defined Names 761
User-Defined Words 10
USER NAME 279
USING 238, 244, 292, 319, 352, 389

V

VALUE 239, 320, 340
VALUE OF 123
VARIANCE 534
VARYING 193, 363, 376
Verb 54, 761

W

WHEN 321, 377
WHEN OTHER 321, 322
WHEN-COMPILED 535
Why YOU Should Learn COBOL 4
WITH FILLER 340
WITH IGNORE LOCK 59
WITH LOCK 359
WITH TEST 361
WORKING-STORAGE SECTION 128
WRITE 410

X

X"91" 645
X"E4" 647
X"E5" 647
X"F4" 648
X"F5" 649
XFD Directives 745
XML GENERATE 413
XML PARSE 424

Y

YEAR-TO-YYYY 536

Z

Zero-Delimited Alphanumeric Literals 34, 761