LIBRO BLANCO SOBRE COBOL

REV 1.0 beta

Gran parte de la información de este libro blanco la he traducido al castellano y su fuente es Wikipedia en Inglés. Actualmente no disponible en castellano.
He añadido más información de otras fuentes y de cosecha personal.
Deseo que os sea al menos de interés.

COBOL es el acrónimo de Lenguaje Orientado a Negocios, del inglés Common Business Oriented Language). Es un lenguaje de programación para ordenadores compilado diseñado para propósitos empresariales y de negocio.

Es un lenguaje imperativo, procedural y desde el 2002 orientado a objetos.

Fue diseñado en 1959 por el CODASYL (Conference on Data Systems Languages) y se basó en gran parte en un lenguaje de programación anterior diseñado por Grace Hopper, comúnmente conocida como "La Abuela del COBOL".

La vida de Grace Murray Hopper es bastante interesante y merece una mención.

Falleció en 1992 con 85 años. Fue una científica americana experta en ordenadores y contraalmirante de la Marina de los Estados Unidos. Como pionera en este campo, fue la primera programadora del ordenador Hardward Mark I e inventó el primer compilador para un lenguaje de programación. Popularizo la idea de la independencia de los lenguajes respecto a la maquina lo que le guió al desarrollo de COBOL, como a la creación del primer lenguaje de programación de alto nivel. A ella se le atribuye la población del término "debugging" (que no bug) para detectar fallos informáticos (inspirado en la polilla-bug que se otros científicos encontraron en el Mark II).

Tras su experiencia con FLOW-MATIC, Hopper pensó que podía crearse un lenguaje de programación que usara órdenes en inglés y que sirviera para aplicaciones de negocios. Con esta idea, las bases para COBOL habían sido establecidas, y 2 años después se creó el comité que diseño este lenguaje.

Aunque Hopper no tuvo un papel preponderante en el desarrollo del lenguaje, fue miembro del comité original para crearlo, y el FLOW-MATIC fue una influencia tan importante en el diseño de COBOL, que Hopper ha pasado a la historia de la informática como su creador.

COBOL fue el primer lenguaje que ofreció una auténtica interfaz a los recursos disponibles en el ordenador, de forma que el programador no tenía que conocer los detalles específicos. Además, los programas desarrollados para una plataforma concreta podían ser ejecutados en un ordenador diferente a aquél en el cual se habían programado sin necesidad de hacer cambios.

Al final de su carrera profesional participó en los comités de estandarización de los lenguajes de programación COBOL y FORTRAN.

Lo que se pretendía con COBOL era crear un lenguaje portable para el procesado de datos. Usado temporalmente como recurso provisional, el departamento de defensa forzó a los fabricantes a suministrarlo, lo que popularizo y extendió la adopción del lenguaje.

En el año 1968 se estandarizó y desde entonces ha sido revisado 4 veces. Las expansiones o extensiones incluyen soporte para programación orientada a objeto y estructurada. Es estándar actual es el ISO/IEC 1989:2014

COBOL posee una sintaxis basada en la lengua inglesa y fue diseñado para documentarse a sí mismo mientras se programa así como ser fácilmente interpretado y leído. Sin embargo es muy verboso y amplio, emplea unas 300 palabras reservadas.

Por ejemplo, en la notación moderna de muchos lenguajes empleamos:

x=y;

En cambio cobol usa MOVE x TO Y

Como podemos apreciar es bastante más natural.

Cobol divide su código en 4 partes o divisiones (DIVISIONS) que son la Identification, environment, data y procedure).

La identificación, el entorno, los datos y para finalizar procedimientos. Contiene unas secciones, párrafos y sentencias muy rígidas en jerarquía.

Para evitar una biblioteca enorme, el estándar especifica 43 estamentos, 87 funciones y sólo una clase.

El lenguaje COBOL ha sido criticado a lo largo de su vida. Ha sido criticado por su verbosidad, su proceso de diseño y un pobre soporte para la programación estructurada, el cual ha llevado a sus detractores a comentar que se crean programas incomprensibles y monolíticos. Ha sido separado de la ciencia computacional desde su creación.

Un poco de Historia y Especificación

El fondo.

Los usuarios de ordenadores y fabricantes estuvieron preocupados sobre el creciente coste de crear programas. En 1959 una encuesta determinó que para muchos proyectos el procesamiento de datos, sus costes de programación al menos eran de 800.000 dólares y la traducción de programas para que funcionara en un nuevo hardware costaría unos 600.000 dólares. En una época donde los nuevos lenguajes de programación estaban proliferando a un ritmo cada vez mayor, la misma encuesta sugería que, si se empleaba un lenguaje orientado a la empresa o al negocio la conversión o adaptación sería, mucho más barata y rápida.

En abril de 1959 representantes de los fabricantes, usuarios, centros de formación convocaron un encuentro en la Universidad de Pennsylvania para organizar unas jornadas sobre los lenguajes y su orientación al entorno empresarial. Entre estos representantes estaba incluida Grace Hopper, creadora del lenguaje FLOW-MATIC, Jean Sammet y Saul Gorn.

El grupo solicitó al DoD (Departamento de Defensa) que lo patrocinara, como un esfuerzo para crear un lenguaje común orientado a los negocios. Esto impresionó a Charles A. Philips, director del área técnica de investigación de sistemas de datos del DoD. El DoD que estaba operando con más de 225 ordenadores, tenía unas futuras previsiones de ampliación de otros 175 ordenadores y un gasto de 200 millones de dólares en implementación de programas para ellos. Esto provocó él interés, estas nuevas ideas, sobre todo su objetivo para simplificar gastos y mejorar la implementación. Es por ello que accedieron al patrocinio del proyecto.

COBOL 60

El 28 y 29 de Mayo de 1959 (exactamente un año después del encuentro de Zürich sobre ALGOL 58), se celebró una reunión en el Pentágono para debatir sobre la creación de un lenguaje común de programación orientado a los negocios. A ella asistieron 41 personas y fue presidido por Philips. El DoD estaba preocupado sobre si podía ejecutar los mismos programas en diferentes ordenadores. FORTRAN, era el único lenguaje principal por aquella época y carecía de las características necesarias para escribir este tipo de programas.

Representantes describieron con entusiasmo un lenguaje que podría trabajar con una amplia variedad de entornos, desde la banca, seguros, control de inventarios, etc. Ellos acordaron por unanimidad que más personas deberían ser capaces de programar y que este nuevo lenguaje debería no ser restrictivo por las limitaciones de la tecnología contemporánea. Una mayoría acordó que el lenguaje debiera usar al máximo en inglés, ser capaz de cambiar, ser independiente de la máquina y sobre todo fácil de usar, incluso a expensas de sus capacidades o poder.

Del meeting resultó la creación de un comité directivo y 3 comités adicionales. El corto, el intermedio y el de largo alcance. Al comité corto se le dio 3 meses desde septiembre para producir las especificaciones de un lenguaje intermedio que luego sería mejorada por los otros comites. Su misión oficial, sin embargo, fue identificar las fortalezas y debilidades de los lenguajes existentes de programación, y no especificar directamente la creación de un lenguaje.

El plazo se cumplió con incredulidad por el comité corto. Un miembro, Betty Holberton, describió los tres meses de plazo con puro obtimismo y puso en duda que el lenguaje creado fuera provisional.

El comité directivo se reunió el 4 de Junio y acordó el nombre de toda la actividad como Comité de Sistemas de datos y lenguajes (CODASYL) y formar un comité ejecutivo.

El comité corto estaba formado por miembros de 6 fabricantes de ordenadores y 3 agencias de gobierno. Los 6 fabricantes de computadoras fueron Burroughts Corporation, IBM, Minneapolis-Honeywell, RCA, Sperry Rand y Syldania Electric Products. Las 3 agencias de gobierno fuero la fuerza aérea de Estados Unbidos, David Tayloir Model Basin y la oficina nacional de Estándares (ahora el Instituto de Tecnología y estándares). El comité fue presidido por Joseph Wegstein de la oficina americana de estándares. El trabajo empezó con la investigación de la descripción de los datos, estamentos, aplicaciones existentes y experiencias de usuario.

El comité principalmente investigó los lenguajes FLOW-MATIC, AIMACO y COMTRAN. El lenguaje FLOW-MATIC fue particularmente influyente. Grace Hopper creadora de FLOW-MATIC actuó como consejera técnica en el comité. Las mayores contribuciones de FLOW-MATIC a COBOL fueron los nombres de variables, las palabras en inglés para los comandos y la separación de las descripciones de datos y las instrucciones.

El lenguaje de IBM llamado COMTRAM, creado por Bob Bemer, fue una respuesta al potencial del Flow-Matic. Algunas de sus características no se incorporaron a COBOL para que no pareciera que hubiera dominado el proceso de diseño. En 1981 Jean Sammet dijo que había habido un "un fuerte prejuicio anti IBM" de algunos miembros del comité (incluida ella misma). Cuando Roy Goldfinger, autor del manual Comtran y miembro del comité de alcance intermedio, asistió a una reunión de la Subcomisión para mantener su lenguaje y fomentar el uso de expresiones algebraicas, Grace Hopper envió un memorándum a la comisión de corto alcance reiterando los esfuerzos de Sperry Rand para crear un lenguaje basado en Inglés.

En 1980, Grace Hopper comentó que COBOL 60 es 95% FLOW-MATIC y que Comtran había tenido una influencia "Extremadamente pequeña". Las características que incorpora COMTRAN dentro de COBOL son las formulas, la cláusula PICTURE y el estamento IF mejorado que obviamente necesitaba de GOTRO y de una gestión de ficheros más robusta.

La utilidad del trabajo del comité fue sujeto de un gran debate. Mientras que algunos miembros pensaron que el lenguaje tuvo muchos compromisos y fue resultado de un diseño por comité, otros consideraron que era el mejor lenguane de los 3. Algunos les pareció que el lenguaje fue muy complejo, en cambio otros muy simple. Alguns características polémicas que algunos condideraron inútiles o aspectos demasiado complejos para el procesamiento de datos. Las expresiones booleanas, formulas y subscipts de tablas. Otro aspecto de controversia fue hacer keywords sensibles al contexto y el efecto que podría tener sobre la legibilidad. Aunque las keywords fueron al final rechazadas, este enfoque se empleó más tarde en PL/I y parcialmente en COBOL desde el 2002. Poco se tuvo en consideración la interactividad, interacciones con los sistemas operativos (pocos por entonces) y funciones (considerado como puramente matemático y no de utilidad para el procesamiento de datos.

Las especificaciones se presentaron al comité ejecutivo el 4 de septiembre. Ellos estuvieron a la altura de las expectativas: Joseph Wegstein señalo que "contiene puntos ásperos y requiere algunas adiciones" y Bob Bemer tarde los describió como una "mezcolanza". El subcomité se le dio un plazo hasta diciembre para pulirlo.

En una reunión a mediados de septiembre, el comité debatió sobre el nuevo nombre del lenguaje. Nombre sugeridos “BUSY” (Business System), “INFOSYL” (Information System Language) y COCOSYL (Common Computer Systems Language). El nombre de “COBOL” fue sugerido por Bob Bemer.

En Octubre, el comité intermedio recibió copias de lenguaje de facto creadas por Roy Nutt. Sus características impresionaron al comité, por lo que les motivo para aprobar una resolución para basar COBOL en esas especificaciones. Esto fue un golpe para el comité corto, que había realizado un buen progreso en la especificación. Pero a pesar de que las especificaciones de facto eran técnicamente superiores, no fueron pensadas en base a la portabilidad y consenso del usuario, también carecía de aplicativo demostrable, lo que supuso su revocación por parte de los partidarios de COBOL basado en FLOW-MATIC.

Pronto se hizo evidente que el comité era demasiado grande para que el proceso avanzara rápidamente.

Como curiosidad un frustrado Howard Bromberg compró una lápida por 1$5 con "COBOL" grabado en ella y se la envió a Charles Phillips para demostrar su descontento (la lápida se encuentra actualmente en el Museo de Historia de los Computadores).

Un subcomité se creó para analizar los lenguajes existentes. Estaba formado por seis personas:

William Selden y Gertrude Tierney de IBM

Howard Bromberg y Howard Discount de RCA

Vernon Reeves y Jean E. Sammet de Sylvania Electric Products

El subcomité hizo la mayor parte del trabajo de creación de la especificación, dejando el comité de corto alcance para revisar y modificar su trabajo antes de producir la especificación final.

Las especificaciones fueron aprobadas por el Comité Ejecutivo el 3 de enero de 1960, y se enviaron a la oficina de imprenta del gobierno, que las imprimieron como COBOL 60. Los objetivos perseguidos del idioma deberían permitir programas eficientes y portátiles, para ser escritos fácilmente, y para permitir a los usuarios mover los nuevos sistemas sin esfuerzo y con coste mínimo, siendo adecuado para los programadores inexpertos. El Comité Ejecutivo CODASYL más tarde creó el Comité de Mantenimiento COBOL para responder preguntas de los usuarios y los proveedores y para mejorar y ampliar las especificaciones.

Durante 1960, fue en aumento la lista de fabricantes que planearon construir compiladores COBOL. En septiembre, cinco nuevos fabricantes se habían unido CODASYL (Bendix, Control Data Corporation, General Electric (GE), National Cash Register y Philco) y todos los fabricantes representados habían anunciado los compiladores de COBOL. GE e IBM planean integrar COBOL en sus propios idiomas, GECOM y Comtran, respectivamente. Por el contrario, IBM planeo sustituir su propio idioma, CODEL, por COBOL.

Mientras tanto, RCA y Sperry Rand trabajaron en la creación de compiladores de COBOL.

El primer programa COBOL corrió el 17 de agosto en una RCA 501.

El 6 y 7 de diciembre, el mismo programa COBOL (aunque con cambios menores) corrió en un equipo RCA y un ordenador Remington-Rand Univac, lo que demuestra que la compatibilidad podría ser lograda.

COBOL-61 a COBOL 65

Liderado por Charles Katz, se comentó que muchos defectos de lógica se encontraron en COBOL 60. Se advirtió que en ocasiones no se podía interpretar con certeza el código. Un comité a corto plazo promulgo una serie de mejoras relacionadas con la limpieza de su sintaxis, similar a ALGOL para eliminar las ambigüedades.

Los primeros compiladores COBOL fueron lentos y primitivos. En 1962 una evaluación de la US Navy testeo velocidades de compilación de 3 a 11 estamentos por minuto. Posteriormente, a mediados de 1964 se incrementaron de 11 a 1000 instrucciones por minuto. Se observó que el incremento en la memoria podría mejorar la velocidad de forma drástica, y los costes de compilación podrían variar ampliamente: el coste por estamento era de 0.23$ a 18,91$.

A finales de 1962, IBM anunció que COBOL sería su lenguaje principal de desarrollo y que el desarrollo de COMTRAN cesaría.

COBOL-60 fue reemplazado en 1961 por COBOL-61. Especificaciones extendidas en 1963 introdujeron ordenaciones y facilidades para la generación de informes. Las mejoras añadidas corrigieron los fallos identificados por HoneyWell a finales del 1959 en una carta a el comité corto. La edición de COBOL de 1965, trajo nuevas aclaraciones a las especificaciones e introdujo mejoras para el manejo de archivos de almacenamiento y tablas.

COBOL-68

Los esfuerzos comenzaron y se trató de estandarizar COBOL para superar las incompatibilidades entre versiones. A finales de 1962, las normas ISO y America Standards Institute (ANSI actualmente) crearon grupos para crear normas estándar. ANSI produjo el USA Standard COBOL X3.23 en agosto de 1968 y se convirtió en la piedra angular para las versiones posteriores. Esta versión fue conocida como Norma Nacional Americana (ANS) COBOL y fue adoptado por la ISO en 1972.

COBOL-74

En 1970, COBOL se había convertido en el lenguaje de programación más utilizado en el mundo.

Independientemente del comité ANSI, el Comité CODASYL estuvo trabajando en la mejora del lenguaje. Describieron nuevas versiones en 1968, 1969, 1970 y 1973, incluyendo cambios como nueva comunicación entre programas, la depuración y facilidades para el merging de archivos, así como la mejora de las características de manejo de series y de inclusión de librerías. Aunque CODASYL era independiente del ANSI comité, el CODASYL Journal of Developent fue utilizado por ANSI para identificar las características que eran lo suficientemente populares como para justificar su aplicación. El Comité del lenguaje de programación también se puso en contacto con ECMA y el comité japonés COBOL estándar.

En 1974, ANSI publicó una versión revisada de (ANS) COBOL, que contiene nuevas características, como las organizaciones de archivos, la sentencia DELETE y el módulo de segmentación. Las características suprimidas incluyen la declaración NOTE, la declaración EXAMINE (que fue sustituida por INSPECT) y el módulo de acceso aleatorio (que fue reemplazado por los nuevos módulos secuenciales y relación e / S). Fueron 44 cambios que chocan con la nueva norma existente. ISO después adoptó la actualización para el estándar en 1978.

COBOL-85

En junio de 1978, se comenzó a trabajar en la revisión de COBOL-74. El estándar propuesto (comúnmente llamado COBOL-80) fue significativamente diferente del anterior, legando a provocar preocupaciones acerca de los costos de incompatibilidad y de conversión.

En enero de 1981, Joseph T. Brophy, Vice-Presidente de Travelers Insurance, amenazó con demandar a la comisión estándar porque no era compatible hacia atrás con COBOL-74. Brophy describió previas conversiones en su base de código de unas 40 millones de líneas como "no productivas" y una "pérdida total de nuestros recursos de programación".

Ese mismo año, la Asociación de Gestión de Procesos de Datos (DMPA) dijo que estaban "muy opuestos" a la nueva norma, citándola como "prohibitiva" para los costos de conversión y mejoras impuesto de forma "forzosa para el usuario".

Durante el primer período de revisión pública, el comité recibió 2.200 respuestas, de las cuales 1.700 eran cartas con respuesta negativa para el modelo. Otras respuestas fueron un análisis detallado de los efectos que COBOL-80 tendría en sus sistemas.; los costos de conversión se estimaron por lo menos en 50 centavos de dólar por línea de código. Menos de una docena de las respuestas estaban a favor del estándar.

En 1983, la DPMA retiró su oposición al estándar, citando la capacidad de respuesta de la comisión a las preocupaciones públicas. En ese mismo año, un estudio de la Oficina Nacional de Normas concluyó que la norma propuesta presentaría algunos problemas. Un año más tarde, el compilador COBOL-80 fue liberado a los usuarios del DEC VAX, quien señaló que la conversión de programas COBOL- 74 planteaban pocos problemas. La nueva declaración EVALUATE y PERFORM en línea fueron particularmente bien recibidos y mejoraron la productividad, gracias a flujo de control simplificado y depuración.

La segunda borrador de opinión pública atrajo otras 1.000 respuestas (principalmente negativas), mientras que el último borrador apunto a sólo 25, momento el cual, se constató que muchas preocupaciones se habían abordado.

A finales de 1985, ANSI publicó la norma revisada. 60 características fueron modificadas o se establecieron en desuso y muchos cambios se añadieron:

- Finalizadores de ámbito (END-IF, FIN-PERFORM, FIN-READ, etc.)
- subprogramas anidados.
- CONTINUE, un estamento de no-operación
- EVALUATE, una sentencia switch
- INITIALIZE, una declaración que puede establecer grupos de datos a sus valores predeterminados.

- inline PERFORM para bucles - anteriormente, los cuerpos de bucle tuvieron que ser especificados en un procedimiento separado.
- modificación de referencia, que permite el acceso a subcadenas.
- Códigos de E / S

La norma fue aprobada por la ISO el mismo año. Dos enmiendas siguieron en 1989 y 1993, los primeros que introducen funciones intrínsecas y las otras correcciones. ISO aprobó las enmiendas en 1991 y 1994, respectivamente.

COBOL 2002 y la orientación a objetos en COBOL

A principios de 1992 se decidió añadir la OPP u orientación a objetos en su próxima revisión. Las características de orientación a objetos se tomaron de C++ y Smalltalk. La estimación inicial fue disponer de esta revisión completada en 1997 por un comité ISO a modo Draft o borrador. Algunas industrias (incluyendo Micro Focus, Fujitsu e IBM) presentaron su sintaxis orientada a objetos basada en los borradores. El estandar ISO final se aprobó y publico a finales del 2002.

Fujitsu / GTSoftware, Micro Focus y introdujeron compiladores COBOL orientados a objetos para el .NET Framework.

Había gran cantidad de nuevas características, muchas de estas habían sido incluidas por CODASYL COBOL "Journal of Development" desde 1978, se había perdido la ocasión de ser incluidas en COBOL-85. Estas características incluyen:

- Código de forma libre (free-form code).
- Funciones definidas por el usuario.
- Recursividad
- Procesamiento basado en configuración local.
- Soporte para Unicode
- Procesamiento en coma flotante y tipos de datos binarios (hasta entonces los elementos binarios se truncaban basados en su declaración base -10)
- Resultados aritméticos portables.
- Tipos de datos bit y booleano.
- Punteros y sintaxis para liberar almacenamiento.
- La SCREEN SECTION para interfaces de usuario basados en texto.
- La característica VALIDATE
- Una mejor interoperatividad con otros lenguajes de programación y frameworks como .NET y JAVA.

COBOL 2014

Entre 2003 y 2009, 3 informes técnicos se crearon para describir la finalización de objetos, el procesamiento XML y las colecciones de clases para COBOL.

COBOL 2002 padeció de un pobre soporte, no había compiladores que soportaran el estándar. Micro Focus constato que fue causa de la falta de demanda de las nuevas características por parte de los usuarios y debido a la abolición de la suite de testeo del NIST que se había utilizado para comprobar la estandarización del compilador. El proceso de estandarización fue lento y conto con pocos recursos.

COBOL 2014 incluye los siguientes cambios.

- Los resultados aritméticos portables han sido reemplazados por tipos de datos IEEE 754
- Las características principales se han vuelto opcionales, como la orientación a objetos, la facilidad VALIDATE, el generador de informe y las opciones de gestión de la pantalla.
- Sobrecarga de métodos.
- Tablas de capacidad dinámica (una característica eliminada del borrador de COBOL 2002)

EL legado

Los programas COBOL son empleados a nivel mundial por los gobiernos y empresas, y se ejecutan en diversos sistemas operativos como Z/OS, VME, Windows, Linux. En 1997, el grupo Gartner informó que el 80% de los negocios empresariales ejecutan COBOL y su lógica posee más de 200 billones de líneas de código y 6 billones más son escritas anualmente.

A finales del siglo XX, con el problema que existió con el famoso efecto 2000 (Y2K), los programas COBOL tuvieron que ser adaptados en masa de forma significativa, muchos de ellos por los mismos programadores que los habían efectuado décadas antes. El nivel de esfuerzo requerido para reparar el código COBOL se ha atribuido a la cantidad extensa de negocios orientados a COBOL. Muchos de estas aplicaciones emplean las fechas de forma notable y ha sido necesario reparar los campos de datos de longitud fija. Después de estos cambios, muchos negocios actuales continúan con las aplicaciones basadas, adaptadas y totalmente funcionales realizadas en COBOL.

Una encuesta de Computerworld mostró que en los años del 2006 hasta el 2012, el 60% de organizaciones usan COBOL (bastante más que C++ y Visual Basic .NET) a nivel mayoritariamente interno. El 36% de los directivos opinaron que les gustaría migrar de COBOL y el 25% de ellos si fuera más barato. En contra, otras empresas han migrado sus aplicaciones hacia COBOL, han cambiado los sistemas, o han invertido grandes cantidades en mainframes que soportan COBOL.

CARACTERISTICAS DEL LENGUAJE

Sintaxis

COBOL tiene una sintaxis en inglés que es usada para describir casi todo. Por ejemplo, una condición puede ser expresada como:

X IS GREATER THAN y (x es mayor que y).

Y más concisamente x GREATER y ó x>y.

Las condiciones más complejas pueden ser abreviadas eliminando las condiciones repetidas y variables. Por ejemplo:

a>b AND a>c OR a=d

Puede ser abreviado como:

a>b AND c OR d.

Como consecuencia de su sintaxis inglesa COBOL tiene unas 300 palabras reservadas o keywords. Sin embargo las extensiones del compilador de muchas implementaciones pueden tener muchas más: alguna de ellas puede reconocer unas 1.100 keywords.

Algunas Keywords son versiones alternativas de una misma palabra, una forma pluralizada de expresar lo mismo. Ejemplo: las keywords IN y OF pueden usarse indistintamente, como la IS y ARE y VALUE o VALUES. Esto es una muestra de la riqueza sintáctica del lenguaje.

Los elementos sintácticos de un programa COBOL son “palabras”, “literales” y “puntuación”. Las palabras incluyen keywords reservados, identificadores definidos por el usuario, y etiquetas y deben estar separadas de otras palabras por espacios, saltos de línea o signos de puntuación. Los identificadores (para tipos de datos y ficheros, así como para párrafos y secciones tipo label) distinguen entre mayúsculas y minúsculas y pueden contener guiones para facilitar la lectura, con hasta 30 caracteres de longitud. Los elementos literales incluyen constantes numéricas y caracteres (strings).

Esto es muy importante y marca el estilo COBOL:

Un programa COBOL se encuentra dividido en 4 divisiones:
La identification division, La environment division, la data division, y la procedure division

La identification división define el nombre y tipo de los elementos origen y es donde se especifican las clases e interfaces.

La environment especifica las características del programa que dependen del sistema donde se ejecuta, tipos de ficheros y juego de caracteres.

La data es usada para declarar variables y parámetros.

La procedure contiene los estamentos y sentencias del programa.

Cada división esta a su vez subdividida en subsecciones que están compuestas por párrafos.

Formato del código

COBOL se puede escribir en dos formatos.

Fijo (por defecto) o libre.
En formato fijo, el código debe estar alienado o indentado para encajar en determinadas áreas. Hasta COBOL 2002 estas áreas eran:

Columnas 1 -6. Área de números de sentencia. Originalmente usadas por las tarjetas o números de línea. Esta área es ignorada por el compilador.

Columna 7. Área de indicador.
Los caracteres siguientes son permitidos:

- El * o asterisco para comentar una línea.
- La / para comentar una línea que será impresa en una nueva página del listado de código.
- El – o guion medio para continuar una línea de código.
- La letra D para activar el modo depuración.

Columnas 8-11 el área A. Contiene la DIVISON, SECTON y las cabeceras de procedimiento. Lis números 01 y 77 son para descriptores de ficheros e informes.

Columnas 12-72, Área B. Cualquier otro código no permitido en el área A.

Columna 73.Area del nombre del programa. Históricamente hasta la columna 80. Se utiliza para identificar el programa o secuencia que pertenece la tarjeta.

En COBOL 2002, las áreas A y B se fusionaron y se extendieron hasta la columna 255. Además se eliminó el área del nombre del programa.

COBOL 2002 también introdujo el código de formato libre. El código con formato libre puede ser colocado en cualquier columna del archivo, como en los lenguajes C y Pascal. Los comentarios se pueden especificar usando *>, pudiendo colocarse en cualquier parte, además este comentario puede ser usado con el formato fijo. Las líneas de continuación se han eliminado y la directiva >>PAGE reemplaza al indicador /.

Identification division

La identification indica al ordenador el nombre del programa y proporciona otras informaciones relativas al autor del programa, cuando ha sido escrito, cuando ha sido compilado, a que está destinado, etc. De hecho, sólo el nombre del programa es de carácter obligatorio.

Environment division

Esta sección contiene la configuración o parámetros del entorno y la sección de los ficheros de entrada y salida (I/O). Se emplea para especificar varias características como la configuración regional, monedas, formatos, conjunto de caracteres. Información relacionada con el manejo de archivos.

Archivos

COBOL soporta tres formatos de ficheros, u organizaciones: secuencial, indexada y relativa.

En los ficheros secuenciales, los registros son contiguos y deben ser recorridos secuencialmente, de forma similar a una lista enlazada.

Los archivos indexados tienen uno o más índices que permiten acceder a los registros al azar y que se puedan ordenar. Cada registro debe tener una clave única, pero claves de alternarivas no tienen que ser únicas.

Las implementaciones de los archivos indexados varían entre los vendedores, aunque las implementaciones más comunes comunes, como la C-ISAM y VSAM, se basan en ISAM de IBM.

Los archivos relativos, como los archivos indexados, necesita tener una clave de registro única, pero no poseen claves alternativas. La Clave de un registro relativo es su posición ordinal; por ejemplo, el décimo disco tiene una clave de 10. Esto significa que la creación de un registro con una clave de 05 de mayo requerirá la creación de previos registros (vacios) . Los ficheros relativos permiten tanto el acceso secuencial y el aleatorio.

Una extensión no estándar pero común es la organización secuencial de línea (line sequiential), se utiliza para procesar archivos de texto. Los registros en un archivo se cierran mediante un salto de línea y pueden ser de longitud variable.

Data división

La data se divide en **6 secciones** que declaran diferentes elementos:

- La sección de archivos (file section), para los registros de archivos.
- La sección de almacenamiento de trabajo (working storage section), para las variables estáticas.
- La sección de almacenamiento local (local storage section), para las variables automáticas.
- La sección de enlazado (linkage section), para los parámetros y el valor de retorno.
- La sección del informe (report section)
- La sección de la pantalla (screen section), para interfaces de usuario basadas en texto.

Agregación de los datos (aggregated data)

Los elementos de datos o ítems en COBOL se declaran jerárquicamente a través del uso del nivel de números que indican si un elemento de datos es parte de otro.

Un item con un número superior está subordinado a un elemento con uno más bajo. Los ítems que contienen elementos subordinados que no se subordinan a otro item se llaman registros. Los items que no tienen elementos de datos subordinados se denominan items elementales (elementaty ítems); los que lo hacen son los llamados elementos de grupo (group ítems).

```
01  some-record.
        05  num             PIC 9(10).
        05  the-date.
            10  the-year    PIC 9(4).
            10  the-month   PIC 99.
            10  the-day     PIC 99.
```

En el ejemplo superior, num y the-date están subordinados al registro some-record, mientras que el año, el mes y el día son parte del elemento del grupo the-date.

Los números de nivel que se utilizan para describir los elementos de datos estándar están entre 01 y 49. Los elementos subordinados pueden no ser ambiguos con las palabras clave o keywords IN (ó OF). Por ejemplo:

```
01  sale-date.
        05  the-year        PIC 9(4).
        05  the-month       PIC 99.
        05  the-day         PIC 99.
```

Los nombres the-year, the-month y the-day son ambiguos de por si, porque existen 3 con el mismo número. Para especificar un ítem de datos concreto, por ejemplo, uno de los items contenidos en el grupo sale-date, el programador podría usar the-year IN sale-date (o su equivalente semántica the-year OF sale-date). Esta sintaxis es similar a la notación punto “dot notation” soportada por muchos lenguajes parecidos a C u otros orientados a objetos.

Otros niveles de datos (números de nivel concretos)

El número de nivel 66 es usado para declarar una reagrupación de ítems previamente definidos, con independencia de como esos ítems están estructurados.

```
01  customer-record.
           05  cust-key            PIC X(10).
           05  cust-name.
               10  cust-first-name PIC X(30).
               10  cust-last-name  PIC X(30).
           05  cust-dob            PIC 9(8).
           05  cust-balance        PIC 9(7)V99.

       66  cust-personal-details   RENAMES cust-name THRU cust-dob.
       66  cust-all-details        RENAMES cust-name THRU cust-
balance.
```

El número de nivel 77 indica que el elemento es independiente, y en dicha situación es equivalente a el nivel número 01. Por ejemplo, el siguiente código declara dos ítems de datos 77 property-name y sales-region, que son ítems no agrupados de datos independientes de otros elementos.

```
77  property-name       PIC X(80).
       77  sales-region        PIC 9(5).
```

El número de nivel 88 declara un tipo de condición. Cuando el tipo de dato contiene uno de los valores especificados en esa condición. Es una simple lista de valores. Por ejemplo, el siguiente código define dos condiciones 88 que son verdaderos o falsos dependiendo del valor de wage-type. Cuando el ítem contiene el valor de 'H', la condición wage-is-hourly es verdadera, mientras que cuando el valor de wage-is-yearly es 'S' o 'Y', el valor es true, en caso contrario es falso.

```
   01  wage-type           PIC X.
           88  wage-is-hourly VALUE 'H'.
           88  wage-is-yearly VALUE 'S', 'Y'.
```

Tipos de datos de COBOL Standard

El estándar de COBOL proporciona los siguientes tipos de datos.

Tipo de datos	Declaración de ejemplo	Notas/Observaciones
Alfabético	`PIC A(30)`	Sólo puede contener letras o espacios
Alfanumérico	`PIC X(30)`	Puede contener cualquier caracter
Booleano/lógico	`PIC 1 USAGE BIT`	Almacena datos en forma de 0s y 1s, como un número binario
Index	`USAGE INDEX`	Usado para referenciar elementos de tabla
Nacional	`PIC N(30)`	Similar al alfanumérico, pero usando un carácter extendido, e.g. UTF-8
Numérico	`PIC 9(5)V9(5)`	Puede contener sólo números
Objeto	`USAGE OBJECTREFERENCE`	Puede referenciar a un objeto o ser `NULL`
Puntero	`USAGE POINTER`	

La Cláusula PICTURE

Una cláusula PICTURE (o PIC) es una cadena de caracteres o string, cada uno de los cuales representa una parte del elemento de datos o data item.

Algunos caracteres PIC especifican el tipo de item y el número de caracteres o dígitos que ocupa en la memoria. Por ejemplo, un 9 indica un dígito decimal, y un S indica que el item es con signo.

Otros caracteres PIC (llamados de inserción y edición) especifican cómo debe formatear un elemento. Por ejemplo, una serie de caracteres + definen las posiciones, el carácter no numérico más a la derecha contendrá signo del elemento, mientras que otras posiciones de caracteres + a la izquierda de esta posición contendrán un espacio. Caracteres repetidos se pueden especificar de manera más concisa especificando un número entre paréntesis después de un carácter de imagen; por ejemplo, 9 (7) es equivalente a 9999999. Las especificaciones PIC que contienen sólo dígito (9) y el signo (S) pueden definir elementos de datos puramente numéricos, mientras que las especificaciones PIC que contienen alfabético (A) o alfanumérico (X) definen elementos de datos alfanuméricos.

Esta explicación se puede apreciar mejor en la tabla de ejemplo inferior:

Clausula PICTURE	Valor in	Valor out
PIC 9(5)	100	00100
	"Hola"	"Hola" *(esto es correcto pero el resultado puede tener un comportamiento no definido)*
PIC +++++	-10	" -10" *(observa los espacios iniciales)*
PIC 99/99/9(4)	31042003	"31/04/2003"
PIC *(4)9.99	100.50	"**100.50"
		"****0.00"
PICX(3)BX(3)BX(3)	"ABCDEFGHI"	"ABC DEF GHI"

La cláusula USAGE

La cláusula USAGE especifica el formato de datos a almacenar. En función del tipo de datos, puede o bien complementar o ser utilizada en lugar de una cláusula PICTURE.

Si bien puede utilizarse para declarar punteros y referencias de objetos, se orienta principalmente hacia la especificación de tipos numéricos.

Estos formatos numéricos son:

Binaria, donde el tamaño mínimo es especificado por la cláusula PICTURE o por una cláusula USAGE, como BINARY-LONG.

USAGE COMPUTATIONAL, donde los datos pueden ser almacenados en el formato que la aplicación ofrece; a menudo equivalente a USAGE BINARY.

USAGE DISPLAY el formato predeterminado, donde los datos se almacenan como una cadena.

De punto flotante, ya sea en un formato dependiente de la implementación o según IEEE 754.

USAGE NATIONAL, donde los datos se almacenan como una cadena utilizando un juego de caracteres extendidos.

USAGE PACKED-DECIMAL, donde los datos se almacenan en el formato decimal más pequeño posible (normalmente en decimal codificado en binario)

Report Writer

El Report Writer es interesante para crear informes. El programador sólo necesita especificar el diseño del informe y los datos necesarios para generarlo. De esta manera nos liberamos de escribir código, y gestionar otras cosas como los saltos de página, formato de la página, encabezados y pies.

Los reports o informes se asocian a ficheros. Estos ficheros pueden ser escritos a través de estamentos REPORT.

Por ejemplo:

```
    FD  report-out REPORT sales-report.
```

Cada informe se define en la report section de la data division. Un informe se divide en grupos de informes que definen las cabeceras del informe, los pies y los detalles. Los informes funcionan mediante puntos de control jerárquicos. Los “Control breaks” se producen cuando una variable cambia de valor, por ejemplo, cuando creamos un informe a partir de los pedidos de los clientes, los cambios o breaks podrían ocurrir cuando se procesan los pedidos de otro cliente.

Este ejemplo, es un informe que muestra las ventas del comercial y muestra información de los registros o entradas no válidas.

```
RD  sales-report
           PAGE LIMITS 60 LINES
           FIRST DETAIL 3
           CONTROLS seller-name.

       01  TYPE PAGE HEADING.
           03  COL 1                    VALUE "Sales Report".
           03  COL 74                   VALUE "Page".
           03  COL 79                   PIC Z9 SOURCE PAGE-COUNTER.

       01  sales-on-day TYPE DETAIL, LINE + 1.
           03  COL 3                    VALUE "Sales on".
           03  COL 12                   PIC 99/99/9999 SOURCE
sales-date.
           03  COL 21                   VALUE "were".
           03  COL 26                   PIC $$$$9.99 SOURCE sales-
amount.

       01  invalid-sales TYPE DETAIL, LINE + 1.
           03  COL 3                    VALUE "INVALID RECORD:".
           03  COL 19                   PIC X(34) SOURCE sales-
record.

       01  TYPE CONTROL HEADING seller-name, LINE + 2.
           03  COL 1                    VALUE "Seller:".
           03  COL 9                    PIC X(30) SOURCE seller-
name.
```

El informe anterior se procesa con la siguiente distribución:

```
Sales Report
Page  1

Seller: Howard Bromberg
  Sales on 10/12/2008 were $1000.00
  Sales on 12/12/2008 were    $0.00
  Sales on 13/12/2008 were   $31.47
  INVALID RECORD: Howard Bromberg             XXXXYY

Seller: Howard Discount
...
Sales Report
Page 12

  Sales on 08/05/2014 were  $543.98
  INVALID RECORD: William Selden      12O52014FOOFOO
```

```
  Sales on 30/05/2014 were    $0.00
```

Utilizando el report writer en la procedure, existen 4 declaraciones o estamentos.

INICIATE, que prepara el generador para la impresión.
GENERATE, imprime un grupo de informes.
SUPPERSS, que elimina impresión de un grupo de informes.
TERMINATE, termina el procesamiento de informes.

Por ejemplo, la procedure division podría tener este aspecto:

```
OPEN INPUT sales, OUTPUT report-out
           INITIATE sales-report

           PERFORM UNTIL 1 <> 1
               READ sales
                   AT END
                       EXIT PERFORM
               END-READ

               VALIDATE sales-record
               IF valid-record
                   GENERATE sales-on-day
               ELSE
                   GENERATE invalid-sales
               END-IF
           END-PERFORM

           TERMINATE sales-report
           CLOSE sales, report-out
           .
```

Procedure Division

Procedimientos o procedures

En ella encontramos todos los procesos necesarios para que el programa funcione, que haga para la que fue concebido.

Todo esto se realiza con instrucciones (ordenes, verbos, comandos, etc..). Cada uno de ellos con un formato y una solución que resolver.

Las secciones y párrafos en la procedure division (llamados de forma conjunta procedures) pueden ser usados como etiquetas o labels y como rutinas simples. A diferencia de otras divisiones, los párrafos no necesitan estar en secciones. La ejecución se efectúa a través de procedimientos de un programa hasta que finaliza. Para utilizar procedures como subrutinas se emplea PERFORM. Esto transfiere el control al procedure especificado y retorna el control al final de su ejecución.

Los procedimientos pueden ser llamados de 3 maneras:

Mediante PERFORM
A través de GO TO
O mediante flujo.

```
PROCEDURE DIVISION (USING Variable, Variable ...).
DECLARATIVES.
Nombre-seccion SECTION.
USE AFTER ERROR PROCEDURE ON tipo.
Nombre-parrafo.
Sentencias.
.....
END DECLARATIVES.
Nombre-seccion SECTION.
Nombre-parrafo.
Sentencias.
.....
```

Este sería groso modo el formato general de una Procedure, pero incluso se puede omitir si en un programa no vamos a realizar ningún proceso.

COBOL 2014 cuenta con 47 estamentos (llamados verbos), los cuales pueden ser agrupados en categorías como control de fujo, entrada y salida, manipulación de datos, generación de informes.

Control del flujo

Los estamentos condicionales de COBOL son:

IF y **EVALUATE**

EVALUATE es como una sentencia Switch en otros lenguajes. Tiene la capacidad de evaluar múltiples valores y condiciones.

Por ejemplo, puede ser usada para implementar tablas decisión. En el código inferior vemos cómo puede manipular el control de un torno CNC.

```
EVALUATE TRUE ALSO desired-speed ALSO current-speed
    WHEN lid-closed ALSO min-speed THRU max-speed ALSO LESS THAN
desired-speed
        PERFORM slow-down-machine
    WHEN lid-closed ALSO min-speed THRU max-speed ALSO GREATER THAN
desired-speed
        PERFORM speed-up-machine
    WHEN lid-open ALSO ANY ALSO NOT ZERO
        PERFORM emergency-stop
    WHEN OTHER
        CONTINUE
END-EVALUATE
```

El estamento PERFORM se emplea para definir loops o búcles, que se ejecutan hasta que una condición es verdadera o true. También se emplea para llamar a procedimientos específicos como he mostrado anteriormente.

CALL e INVOKE puede llamar a subprogramas y métodos respectivamente. El nombre del subprograma o método esta contenido en una cadena de caracteres. Los parámetros pueden ser pasados por referencia, por contenido o por valor. CANCEL descarga los programas de memoria. GO TO permite saltar a un procedimiento determinado.

El estamento GOBACK es una sentencia de retorno y STOP para el programa.

El estamento EXIT puede tener 6 diferentes formatos. Puede ser usado como sentencia de retorno, break o ruptura, continue, como marcador de finalización o para dejar un procedimiento.

Las excepciones son lanzadas por RAISE y manipuladas por un gestor de excepciones definido en DECLARATIVES como parte de la procedure division. Las DECLARATIVES son secciones que empiezan con el estamento USE para manipular los errores. Las excepciones pueden ser nombres u objetos.

RESUME se emplea como declaración de saltos del estamento después de producirse una excepción o una salida de procedimiento de DECLARATIVES. A diferencia de otros lenguajes, las excepciones no capturadas no finalizan el programa y pueden continuar funcionando pero afectado por el error producido.

I/O - entrada y salida

Esta sección se define a sí misma. Entradas y salidas.

Los estamentos OPEN, CLOSE, READ, WRITE y otros 3 más, REWRITE, que actualiza un registro, START, que seleccione registros manipulados con una cierta clave, y UNLOCK, que libera el acceso a los últimos registros. La iteración del usuario se realiza con ACCEPT y DISPLAY.

Manipulación de datos

Los siguientes verbos se emplean para manipular datos:

- `INITIALIZE`, define un conjunto de datos y sus valores por defecto.
- `MOVE`, asigna valores a conjuntos de datos.
- `SET`, con 15 formatos: puede modificar índices, asignar referencias a objetos, tablas y otros muchos más usos
- `ADD`, `SUBTRACT`, `MULTIPLY`, `DIVIDE`, y `COMPUTE`, manipulación aritmética (con `COMPUTE` asocia el resultado de una fórmula a una variable).
- `ALLOCATE` y `FREE`, gestiona memoria dinámica.
- `VALIDATE`, valida y disrubuye datos y descripciones de ítems en la data division.
- `STRING` y `UNSTRING`, concatena y divide cadenas, respectivamente.
- `INSPECT`, recuenta o reemplaza subcadenas de una cadena.
- `SEARCH`, busca en una tabla la primera entrada que cumpla una condición.

Las tablas y ficheros se ordenan usando SORT y MERGE. El verbo RELEASE proporcina los registros a ordenar y RETURN recibe los registros en orden.

Scope termination

Algunos estamentos o sentencias como IF y READ, pueden contener otras sentencias o estamentos. Las sentencias pueden terminar de 2 formas:

Por un periodo (terminación implícita).

```
*> por periodo ("implicit termination")
IF invalid-record
    IF no-more-records
        NEXT SENTENCE
    ELSE
        READ record-file
            AT END SET no-more-records TO TRUE.
```

Por ámbito.

```
*> por ámbito ("explicit termination")
IF invalid-record
    IF no-more-records
        CONTINUE
    ELSE
        READ record-file
            AT END SET no-more-records TO TRUE
        END-READ
    END-IF
END-IF
```

Declaraciones anidadas terminados con un período son una fuente común de errores.

Por ejemplo:

```
IF x
    DISPLAY y.
    DISPLAY z.
```

La intención es mostrar Y y Z si la condición x es true o cierta. Sin embargo, z se mostrará independientemente del valor de x porque la instrucción IF termina erróneamente después de y.

Otro bug o error común es resultado de la ausencia de condiciones de indentado.

```
IF x
    IF y
        DISPLAY a
ELSE
    DISPLAY b.
```

En el ejemplo superior, el ELSE se asocia con el IF Y en vez de X.

Código automodificable.

La especificación original COBOL soportó el estamento

```
ALTER X TO PROCEED TO Y
```

Para que muchos compiladores generasen código seft-modifying.

Muchos compiladores todavía lo soportan, pero se consideran obsoletas en el COBOL 1985 estándar y se eliminaron en el 2002.

Hola mundo

Bueno, faltaba como mostrar el típico Hola mundo en COBOL.

```
IDENTIFICATION DIVISION.
       PROGRAM-ID. HOLA MUNDO.
       PROCEDURE DIVISION.
           DISPLAY 'Hola Mundo'.
           STOP RUN.
```

Esquema de un programa COBOL - Resumen

Como hemos visto, los programas cobol se dividen en partes llamadas "DIVISION" que a su vez se separan en secciones llamadas "SECTION". Vamos a ver el esquema básico que todo programa cobol debe tener:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRUEBA1.
```

Nombre del programa. Debe coincidir con el nombre externo del programa.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
```

Usamos los puntos como separadores de miles y la coma como el separador de decimales.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
```

En esta parte se definen los ficheros en caso de que los haya.

```
   SELECT FICHERO1
```

Nombre de fichero.

```
   ACCES MODE IS SEQUENTIAL
```

Tipo de acceso: SEQUENTIAL para los planos, INDEXED para los indexados.

```
   FILE STATUS IS FS-FICHERO1
```

Variable donde guardará el file-status (código de control de errores de acceso a ficheros).

```
DATA DIVISION.
FILE SECTION.
FD FICHERO1 RECORDING MODE IS F
```
Fichero con longitud fija. Si fuese de longitud variable pondríamos V.

```
    BLOCK CONTAINS 0 RECORDS
    RECORD CONTAINS 129 CHARACTERS.
```
Longitud del fichero.

```
01 REG-FICHERO1 PIC X(129).
```
Donde guardaremos la información.

```
WORKING-STORAGE SECTION.
```
En ella definimos las variables que usaremos en el programa.

```
01 WX-VARIABLE PIC X.
```
Definición de variables lo veremos más adelante.

```
LINKAGE SECTION.
```
Área de comunicación con otros programas

```
01 AREA-LINKAGE PIC X.
PROCEDURE DIVISION. / PROCEDURE DIVISION USING AREA-LINKAGE.
```
Si hay un área definida en la linkage debemos incluir el USING en la procedure.

Aquí es donde va el programa en sí. La estructura general será:

```
0000-PRINCIPAL.
    PERFORM 1000-INICIO

    PERFORM 2000-PROCESO
      UNTIL CUMPLE-CONDICION

    PERFORM 3000-FINAL
    .
```

La numeración de párrafos suele ser esa, pero cada empresa puede tener su propia nomenclatura estándar.

El proceso en un programa se suele repetir n veces, hasta que se cumple la condición indicada en el UNTIL.

En el párrafo de final se incluye la instrucción de finalización de ejecución:
STOP RUN para programas principales.
GOBACK para rutinas.

Se pueden definir tantos párrafos como queramos, pero la estructura de todos ellos será la siguiente:

```
1000-INICIO.
    código con instrucciones a realizar
    .
```

Es decir, todos los párrafos terminan con un punto "." que indica el final de párrafo.

A tener en cuenta:
En COBOL no podemos empezar a escribir donde queramos, pues cada cosa tiene su sitio^^
Un programa cobol mide de ancho 80 posiciones, aunque sólo escribiremos hasta la 72.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
       01 WX-CAMPOS.
          05 WX-CAMPO1 PIC X.
          05 WX-LITERAL PIC X(40) VALUE 'LITERAL TAN GRANDE QUE NO CABE '
      -   'EN UNA LINEA Y LO TENEMOS QUE PARTIR'.
      *
       PROCEDURE DIVISION.
      ******************************************************************
       00000-PRINCIPAL.
      *
            PERFORM 10000-INICIO.
```

* De la posición 1 a la 6: no se codifica nada.
* Posición 7: se escribirá un * si queremos comentar la línea. En caso de que un texto no nos quepa en una línea, se escribirá en esta posición un guión "-" para continuarlo.
* De la posición 8 a la 11: se denomina área A. Aquí escribiremos las "divisiones" y "secciones", los niveles 01 y los descriptores de ficheros "FD".
* De la 12 a la 72: se denomina área B. Aquí se escribirán el resto de instrucciones del programa, y los niveles de variables subordinadas (02 en adelante).
* De la 73 a la 80: no se codifica nada.

Programación orientada a objeto.

Las clases e interfaces se añadieron a COBOL 2002. Las clases tienen objetos Factory, conteniendo métodos de clase y variables, objetos de instancia, conteniendo métodos y variables. La herencia e interfaces proporcionan polimorfismo. Soporte para programación genérica se puede lograr mediante clases parametrizadas. Los objetos pueden ser almacenados como referencias que pueden estar restringidos a cierto tipo. Existen dos formas de llamar a un método: el estamento INVOKE que actúa de forma similar a CALL, o a través de métodos inline que es similar al uso de funciones.

```
*> es equivalente.
INVOKE my-clase "foo" RETURNING var
MOVE my-clase::"foo" TO var *> incovación en línea
```

Observación: No te lies con el “foo”.

Foo es un término genérico ampliamente usado para aludir a cualquier entidad informática cuyo nombre se ignora o no se quiere expresar. Por sí misma, la palabra foo no tiene un significado preciso; solamente es una representación lógica en el sentido en que las letras "x" e "y" se usan en álgebra para representar un número desconocido.

La palabra foo aparece en el idioma inglés como un neologismo dada su popularidad en describir conceptos en las ciencias informáticas y muchas personas la consideran un ejemplo canónico de una variable metasintáctica. Se usa de forma amplia en la literatura informática anglosajona, generalmente en los ejemplos de programación y pseudocódigo. La Real Academia Española no reconoce esta palabra como parte del idioma español.

El origen de tal palabra no está muy claro porque tiene antecedentes muy complicados, incluyendo una larga historia en los guiones de cómics y caricaturas.

COBOL no proporciona una forma de ocultar métodos. La clase data puede ser ocultada, sin embargo declararla sin la cláusula PROPERTY la puede dejar sin acceso. La sobrecarga de métodos se añadió en COBOL 2014.

La crítica y la defensa

La falta de estructura

En la década de 1970, los programadores comenzaron a moverse lejos del no estructurado "código espagueti" al paradigma de la programación estructurada.

Una de las causas de código spaghetti era la instrucción GO TO.

Los intentos de eliminar GO TO del código COBOL, sin embargo, dieron lugar a programas enrevesados y reducción de la calidad del código. Los GO TO fueron reemplazados en gran medida por el PERFORM para a declaración de procedimientos, lo que promovió la programación modular y dio un fácil acceso a poderosas capacidades y saltos.

En 1975 Edsger Dijkstra en una carta a un editor titulada "¿Cómo decir verdades que podrían lastimar?" el que era crítico de varios de los contemporáneos de COBOL, informático y ganador del Premio “Turing” comentó que "El uso de COBOL paraliza la mente; su enseñanza debe, por lo tanto, ser considerada como un delito penal".

En su respuesta disidente al artículo de Dijkstra y las anteriores "declaraciones ofensivas," el informático Howard E. Tompkins defendió el COBOL estructurado:

"Los programas COBOL con un flujo de control complicado de hecho tienden a "paralizar la mente "," pero esto es debido a que "hay demasiados programas empresariales escritos por programadores que nunca han tenido el placer de comprender el beneficio de un COBOL estructurado bien enseñado ...”

Lo que considero, es que es mejor aportar y mejorar, porque puede que cáigalos en errores de ego de Dikstra. Alan Kay expuso que en informática, la arrogancia se mide en nanodijkstras. :-)

No obstante no es de extrañar, puesto que el físico-científico Dikstra tenía un carácter árido y ácido, conocidas era su oposición a la instrucción GOTO y además al lenguaje BASIC («mutila la mente más allá de toda recuperación»). En fin sigamos…

Los programas COBOL eran tristemente célebres por ser monolíticos y carentes de modularización. El código COBOL sólo podía ser modularizado través de procedimientos, que resultaron ser insuficientes para sistemas grandes. Era imposible ocultar los datos, es decir, un procedimiento podría acceder y modificar cualquier elemento de datos. Por otra parte, no había manera de pasar parámetros a un procedimiento, una omisión Jean Sammet considerado como el mayor error de la comisión.

Otra complicación fue la capacidad de PERFORM a un rango de procedimientos. Esto significaba que el control podría saltar y volver de cualquier procedimiento, creando de flujo de control complicado y permitiendo a un programador romper la regla "única entrada, única salida".

Esta situación mejoró cuando COBOL adoptó más características. COBOL-74 añadió subprogramas, dando a los programadores la capacidad de controlar los datos de cada parte del programa que podría tener acceso.

COBOL-85 añadió subprogramas anidados, permitiendo a los programadores esconder subprogramas. Además el control sobre los datos y el código se produjo en el 2002, cuando la programación orientada a objetos, funciones definidas por el usuario y tipos de datos definidos por el usuario se incluyeron.

Aspectos de compatibilidad

COBOL fue pensado para ser altamente portable, un lenguaje "común". Sin embargo, para el año 2001, se habían creado alrededor de la friolera de 300 dialectos.

COBOL-85 no era plenamente compatible con las versiones anteriores, y su desarrollo fue polémico. Joseph T. Brophy, el CIO de Travelers Insurance, se esforzó en informar a los usuarios de COBOL de los costes de reprogramación pesados en la aplicación de la nueva norma.

En consecuencia, el Comité COBOL ANSI recibió más de 2.200 cartas del público, en su mayoría negativas, lo que requirió de la comisión para hacer cambios. Por otro lado, se pensó en la conversión a COBOL-85 para aumentar la productividad en los próximos años, lo que justifica los costos de conversión.

Sintaxis verbosa

La sintaxis de COBOL a menudo ha sido muy criticada por su verbosidad. Sin embargo, los defensores señalan que se creó de forma intencional en el proceso de diseño del lenguaje. Uno de los propósitos del lenguaje es crear código auto-documentado, lo que facilita el mantenimiento del programa.

COBOL fue diseñado para a ser fácil para los programadores en el proceso de aprendizaje y uso, además de ser legible para el personal NO técnico y gestores (a pesar de que no exista evidencia de que esto fuese útil). El deseo de legibilidad es debido a que COBOL tiene sintaxis y elementos estructurales en inglés, como sustantivos, verbos, cláusulas, frases, secciones y divisiones. No obstante en 1984, los gestores de programas COBOL estaban luchando para hacer frente a código "incomprensible" y los principales cambios en COBOL-85 se crearon para para ayudar a facilitar el mantenimiento.

Jean Sammet, un miembro del comité de corto alcance, señaló que "se hizo un intento para satisfacer al programador profesional, de hecho la gente cuyo principal interés es programar tienden a ser muy infelices con COBOL" lo que se atribuye a su detallada sintaxis.

La alienación de la comunidad de la informática

La comunidad COBOL siempre ha estado aislada de la comunidad informática. No hay apenas científicos informáticos que participaron en el diseño de COBOL; todos los miembros del comité fueron del área comercial o gubernamental.

Esto era debido a los diferentes intereses de los informáticos de la época, que estaban más interesados en otros campos como el análisis numérico, la física y la programación del sistema, que en los problemas de procesamiento empresarial que abordó el desarrollo de COBOL.

Jean Sammet atribuye la impopularidad inicial de COBOL a una "snob reaction" debido a su falta de elegancia, la falta de informáticos influyentes que participaron en el proceso de diseño y a un desdén por el procesamiento de datos empresariales.

En castellano Esnob es un anglicismo derivado de la palabra «snob» con el cual se denomina a una persona que imita con afectación las maneras, opiniones, etc. de aquellos a quienes considera distinguidos o de clase social alta para aparentar ser igual que ellos. Su plural es «esnobs».

La especificación COBOL utiliza una "notación" única, o metalenguaje, para definir su sintaxis y no la nueva forma de Backus-Naur. Sólo algunos miembros del comité habían oído hablar de esto lo que dio lugar a la una crítica "severa".

La notación de Backus-Naur, también conocida por sus denominaciones inglesas Backus-Naur form (BNF),Backus-Naur formalism o Backus normal form, es un metalenguaje usado para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales.

Posteriormente, COBOL sufrió de una gran escasez de soporte; hubo que esperar hasta 1963 para que apareciesen los libros introductorios. En 1985, existían el doble de libros sobre Fortran y cuatro veces más sobre BASIC que del lenguaje COBOL en la Biblioteca del Congreso.

Donald Nelson, el presidente del comité de CODASYL COBOL dijo en 1984 que "los académicos... odiaban COBOL" y que los graduados en informática tenían repulsión al lenguaje.

Una encuesta del 2013 realizada por Micro Focus constató que el 20% de los formadores universitarios pensaba que COBOL era anticuado o un lenguaje muerto y que el 55% consideraba que sus estudiantes estaban convencidos que el lenguaje COBOL era anticuado y no tenía futuro. La misma encuesta también mostraba que sólo el 25% de los planes académicos tenían programación COBOL en su plan de estudios a pesar de que el 60% pensaba que era necesario enseñarlo.

El dato contundente aparece en el año 2003. COBOL aparece en el 80% de los planes de estudio de sistemas de información en los Estados Unidos, la misma proporción que los lenguajes C ++ y Java.

Preocupaciones acerca del proceso de diseño

Había dudas sobre la eficacia del proceso de diseño (a veces de quienes tomaron parte del mismo). El Miembro del comité de corto plazo Howard Bromberg, dijo que había "poco control" sobre el proceso de desarrollo y que estaba "plagado de discontinuidad del personal y ... de falta de talento".

Las normas COBOL han sufrido repetidamente retrasos: COBOL-85 llegó cinco años más tarde de lo esperado, COBOL 2002 también con cinco años de retraso, y COBOL 2014 se retrasó hasta 6 años.

Para combatir los retrasos, el comité de la norma permitió la creación de agendas opcionales para añadir características más rápidamente antes de la espera de la próxima revisión del estándar. Sin embargo, algunos miembros del comité expresaron su preocupación por las incompatibilidades entre implementaciones y frecuentes modificaciones de la norma.

Otras defensas

Las estructuras de datos de COBOL fueron influenciadas por los lenguajes de programación posteriores. Su estructura de registro y archivo fueron influenciados por PL / I y Pascal, y la cláusula REDEFINES fue un predecesor a los registros de la variante de Pascal. Las definiciones de la estructura de archivos precedieron el desarrollo de sistemas de gestión de base de datos y los datos agregados fueron un avance significativo respecto a las matrices de Fortran.

La característica COPY de COBOL, aunque se consideró "primitiva", influyó en el desarrollo de las directivas include.

La estandarización de los programas gracias a la normalización significaba que las aplicaciones escritas en COBOL son portables y ha permitido que el lenguaje se haya extendido a una amplia variedad de plataformas de hardware y sistemas operativos. Además, la estructura jerárquica rígida restringe la definición de referencias externas a la Enviroment Division, lo que simplifica los cambios en una plataforma en particular.

IDEs

Fujitsu NetCobol (anteriormente conocido como PowerCOBOL) de la empresa GTSoftware.
http://www.gtsoftware.com/netcobol-net/

Este producto está disponible para .NET, Linux, Solaris.
NetCOBOL es un compilador COBOL que elimina el coste de ejecución por completo.
Al preservar código COBOL original, NetCOBOL asume el riesgo de la modernización de aplicaciones heredadas.
Permite conectar con gestores de bases de datos como SQL Server, Oracle o mediante ODBC.

Puede ser considerado como uno de los grandes compiladores existentes de COBOL.

IBM VisualAge
http://www-03.ibm.com/software/products/es/cobocompfami

Otro de los grandes.
El compilador IBM COBOL da soporte a IBM z/OS, IBM VSE/ESA, IBM z/VM e IBM AIX.

Con las herramientas necesarias para ampliar el desarrollo de su programa y aprovechar las aplicaciones existentes, y así poder centrar su proceso de desarrollo de aplicaciones en el mercado actual, en constante evolución.

ICOBOL 5
http://www.icobol.com/

Interactivo COBOL (ICOBOL) es un COBOL de propósito general basado en ANSI-74 COBOL con muchas funciones ANSI-85 y extensiones útiles. Está disponible para los sistemas Microsoft Windows y Linux. El producto incluye modos de compatibilidad para varios diferentes dialectos COBOL, incluyendo AOS / VS COBOL.

Visual COBOL de Microfocus
http://www.microfocus.es/soluciones/cobol/index.aspx

El lanzamiento de Visual COBOL 2.2.2 tiene muchas nuevas mejoras, incluyendo soporte completo para Microsoft® Visual Studio 2013 y nuevas herramientas para desarrolladores de COBOL.

Los compiladores y las herramientas de Micro Focus permiten a los desarrolladores crear, mantener y mejorar las aplicaciones COBOL empresariales existentes. Ahora puede reducir los tiempos de desarrollo gracias a herramientas como Visual Studio o Eclipse, que permiten una mayor productividad a los desarrolladores, así como utilizar otras tecnologías para optimizar las aplicaciones COBOL, como WPF, WCF, JavaFX, HTML5 o Silverlight, entre otros, dentro del mismo entorno. para mejorar sus aplicaciones COBOL.

- La plataforma cubre todas las necesidades para Windows, Linux and zLinux (incluye Windows 8)
- Soporte de base de datos para Oracle, Microsoft, IBM y open source.
- Desarrollo de aplicaciones COBOL sin cambiar ninguna línea de código con Java Aplication Servers o Microsoft .Net
- Desarrollo de aplicaciones de escritorio visualmente impresionantes, para cer aplicaciones de Internet, aplicaciones COBOL con acceso desde dispositivos móviles, de forma rápida y fácilmente integrando COBOL con otros lenguajes.

TinyCOBOL
http://tiny-cobol.sourceforge.net/

TinyCOBOL es un compilador COBOL está siendo desarrollado por los miembros de la comunidad del software libre.La misión es producir un compilador COBOL basado en los estándares COBOL 85.

TinyCOBOL está disponible para la arquitectura Intel (IA32) y procesadores compatibles en las siguientes plataformas:

- FreeBSD .
- Linux .
- MinGW en Windows.

Todos los sistemas anteriores utilizan el compilador GCC.

TinyCOBOL se distribuye bajo las siguientes licencias:

- El compilador está licenciado bajo la GNU General Public License .
- Las bibliotecas de tiempo de ejecución están licenciados bajo la Licencia Pública General de GNU.

Fuentes en Wikipedia e Internet que he traducido al castellano.

1. Referencias

Radin, George (1978). Wexelblat, Richard L., ed. "The early history and characteristics of PL/I". History of Programming Languages. Academic Press (published 1981). p. 572. doi:10.1145/800025.1198410. ISBN 0127450408.

2. Robinson, Brian (9 July 2009). "Cobol remains old standby at agencies despite showing its age". *FCW*. Public Sector Media Group. Retrieved 26 April 2014.
3. Porter Adams, Vicki (5 October 1981). "Captain Grace M. Hopper: the Mother of COBOL". *InfoWorld* **3** (20): 33. ISSN 0199-6649.
4. Betts, Mitch (6 Jan 1992). "Grace Hopper, mother of Cobol, dies". *Computerworld***26** (1): 14. ISSN 0010-4841.
5. Lohr, Steve (2008). *Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists, and Iconoclasts--The Programmers Who Created the Software Revolution*. Basic Books. p. 52. ISBN 978-0786730766.
6. Ensmenger, Nathan L. (2009). *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. MIT Press. p. 100. ISBN 978-0262050937. LCCN 2009052638.
7. "ISO/IEC 1989:2014". ISO. 26 May 2014. Retrieved 7 June 2014.
8. Beyer 2009, p. 282.
9. Beyer 2009, pp. 281–282.
10. Sammet 1978a, p. 200.
11. Beyer 2009, p. 283.
12. Beyer 2009, p. 284.
13. "Early Meetings of the Conference on Data Systems Languages". *IEEE Annals of the History of Computing* **7** (4): 316. 1985. doi:10.1109/MAHC.1985.10047. edit
14. ^ Jump up to:[a] [b] [c] [d] [e] Sammet 2004, p. 104.
15. Beyer 2009, p. 286.
16. ^ Jump up to:[a] [b] Conner 1984, p. ID/9.
17. Sammet 1978a, p. 201.
18. ^ Jump up to:[a] [b] [c] [d] Bemer 1971, p. 132.
19. Beyer 2009, p. 288.
20. Sammet 1978a, p. 203.
21. CODASYL 1969, § I.2.1.1.
22. Sammet 1978a, p. 204.
23. CODASYL 1969, § I.1.2.
24. Beyer 2009, p. 290.
25. Sammet, Jean (1978). "The Early History of COBOL". *ACM SIGPLAN Notices*(Association for Computing Machinery, Inc.) **13** (8): 121–161.doi:10.1145/960118.808378. Retrieved 14 January 2010.
26. Sammet 1978a, p. 217.
27. ^ Jump up to:[a] [b] Beyer 2009, p. 292.

28. Bemer 1971, p. 131.
29. Beyer 2009, p. 296.
30. Sammet 1978a, p. 221.
31. Beyer 2009, p. 291.
32. "Oral History of Captain Grace Hopper" (PDF). Computer History Museum. December 1980. p. 37. Retrieved 28 June 2014.
33. Sammet 1978a, p. 218.
34. Marcotty 1981, p. 268.
35. Sammet 1978a, pp. 205–206.
36. ^ Jump up to:a b Sammet 1978a, Figure 8.
37. Sammet 1978a, pp. 230–231.
38. ISO/IEC JTC 1/SC 22/WG 4 2001, p. 846.
39. Sammet 1978a, p. 220.
40. Sammet 1978a, p. 228.
41. Sammet 1978a, p. 210.
42. Sullivan, Patricia (25 June 2004). "Computer Pioneer Bob Bemer, 84". *The Washington Post*. p. B06. Retrieved 28 June 2014.
43. Bemer, Bob. "Thoughts on the Past and Future". Archived from the original on 16 May 2014. Retrieved 28 June 2014.
44. Beyer 2009, p. 293.
45. Beyer 2009, p. 294.
46. ^ Jump up to:a b "The Story of the COBOL Tombstone" (PDF). *The Computer Museum Report*(The Computer Museum) **13**: 8–9. Summer 1985. Archived from the original on 3 April 2014. Retrieved 29 June 2014.
47. Bemer 1971, p. 130.
48. "COBOL Tombstone". Computer History Museum. Retrieved 29 June 2014.
49. Beyer 2009, p. 289.
50. CODASYL 1969, § I.1.1.
51. Brown 1976, p. 47.
52. ^ Jump up to:a b c Bemer 1971, p. 133.
53. ^ Jump up to:a b Beyer 2009, p. 297.
54. Williams, Kathleen Broome (10 November 2012). *Grace Hopper: Admiral of the Cyber Sea*. US Naval Institute Press. ISBN 978-1612512655. OCLC 818867202.
55. Garfunkel, Jerome (11 November 1984). "In defense of Cobol". *Computerworld* **18**(24): ID/19.
56. ^ Jump up to:a b Bemer 1971, p. 134.
57. Brown 1976, p. 48.
58. CODASYL 1969, § I.2.2.4.
59. CODASYL 1969, § I.2.3.
60. ^ Jump up to:a b c d Follet, Robert H.; Sammet, Jean E. (2003). Ralston, Anthony; Reilly, Edwin D.; Hemmendinger, David, eds. *Programming language standards*. *Encyclopedia of Computer Science* (4th ed.) (Wiley). p. 1467. ISBN 0470864125.

61. ^ Jump up to:[a] [b] Beyer 2009, p. 301.
62. ^ Jump up to:[a] [b] Brown 1976, p. 49.
63. Brown 1976, p. 52.
64. Triance, J. M. (1974). *Programming in COBOL: A Course of Twelve Television Lectures*. Manchester University Press. p. 87. ISBN 0719005922.
65. Klein 2010, p. 16.
66. Baird, George N.; Oliver, Paul (May 1977). "1974 Standard (X3.23–1974)".*Programming Language Standards — Who Needs Them?* (Technical report). pp. 19–21. Archived from the original on 7 January 2014. Retrieved 7 January2014.
67. Culleton, John R., Jr. (23 July 1975). "'Spotty' Availability A Problem...".*Computerworld* **9** (30): 17. ISSN 0010-4841.
68. Simmons, Williams B. (18 June 1975). "Does Cobol's Report Writer Really Miss the Mark?". *Computerworld* **9** (25): 20. ISSN 0010-4841.
69. Shoor, Rita (26 January 1981). "User Threatens Suit Over Ansi Cobol-80".*Computerworld* **15** (4): 1, 8. ISSN 0010-4841.
70. Shoor, Rita (26 October 1981). "DPMA Takes Stand Against Cobol Draft".*Computerworld* **15** (43): 1–2. ISSN 0010-4841.
71. ^ Jump up to:[a] [b] [c] Gallant, John (16 September 1985). "Revised Cobol standard may be ready in late '85". *Computerworld* **19** (37): 1, 8. ISSN 0010-4841.
72. ^ Jump up to:[a] [b] "Expert addresses Cobol 85 standard". *Computerworld* **19** (37): 41, 48. 16 September 1985. ISSN 0010-4841.
73. Paul, Lois (15 March 1982). "Responses to Cobol-80 Overwhelmingly Negative".*Computerworld* **16** (11): 1, 5. ISSN 0010-4841.
74. Paul, Lois (25 April 1983). "Study Sees Few Problems Switching to Cobol-8X".*Computerworld* **17** (17): 1, 6.
75. Gillin, Paul (19 November 1984). "DEC users get head start implementing Cobol-80". *Computerworld* **18** (47): 1, 6. ISSN 0010-4841.
76. Garfunkel 1987, p. 150.
77. Roy, M. K.; Dastidar, D. Ghost (1 June 1989). "Features of COBOL-85". *COBOL Programming: Problems and Solutions* (2nd ed.). McGraw-Hill Education. pp. 438–451. ISBN 978-0074603185.
78. ^ Jump up to:[a] [b] Saade, Henry; Wallace, Ann (October 1995). "COBOL '97: A Status Report".*Dr. Dobb's Journal*. Retrieved 21 April 2014.
79. Arranga, Edmund C.; Coyle, Frank P. (February 1998). *Object-Oriented COBOL*.Cambridge University Press. p. 15. ISBN 978-0132611404. Object-Oriented COBOL's style reflects the influence of Smalltalk and C++.
80. ^ Jump up to:[a] [b] "COBOL Standards". Micro Focus. Archived from the original on 31 March 2004. Retrieved 2 September 2014.
81. "NetCOBOL for .Net". *netcobol.com*. GTSoftware. 2013. Archived from the original on 8 July 2014. Retrieved 29 January 2014.
82. "A list of Codasyl Cobol features". *Computerworld* **18** (37). 10 September 1984. p. ID/28. ISSN 0010-4841. Retrieved 8 June 2014.
83. ISO/IEC JTC 1/SC 22/WG 4 2001, Annex F.
84. Klein 2010, p. 21.

85. ^ Jump up to:a b "JTC1/SC22/WG4 - COBOL". ISO. 30 June 2010. Retrieved 27 April 2014.
86. Billman, John; Klink, Huib (27 February 2008). "Thoughts on the Future of COBOL Standardization" (PDF). Retrieved 14 August 2014.
87. ISO/IEC JTC 1/SC 22/WG 4 2010, Annex E.
88. Schricker, Don (2 December 1998). "J4: COBOL Standardization". Micro Focus. Archived from the original on 24 February 1999. Retrieved 12 July 2014.
89. Kizior, Ronald J.; Carr, Donald; Halpern, Paul. "Does COBOL Have a Future?".*The Proceedings of the Information Systems Education Conference 2000* **17** (126). Retrieved 2012-09-30.
90. Carr & Kizior 2003, p. 16.
91. Carr & Kizior 2003, p. 10.
92. ^ Jump up to:a b Mitchell, Robert L. (4 October 2006). "Cobol: Not Dead Yet". *Computerworld*. Retrieved 27 April 2014.
93. "Cobol brain drain: Survey results". *Computerworld*. 14 March 2012. Retrieved27 April 2014.
94. ISO/IEC JTC 1/SC 22/WG 4 2010, § 8.9.
95. "Reserved Words Table". *Micro Focus Visual COBOL 2.2 COBOL Language Reference*. Micro Focus. Retrieved 3 March 2014.
96. ^ Jump up to:a b ISO/IEC JTC 1/SC 22/WG 4 2001, § F.2.
97. ISO/IEC JTC 1/SC 22/WG 4 2010, § D.2.
98. ISO/IEC JTC 1/SC 22/WG 4 2010, § D.18.2.
99. ISO/IEC JTC 1/SC 22/WG 4 2010, § D.18.
100. ISO/IEC JTC 1/SC 22/WG 4 2010, p. 100.
101. ISO/IEC JTC 1/SC 22/WG 4 2010, p. 871.
102. ISO/IEC JTC 1/SC 22/WG 4 2010, § D.2.1.
103. "File Organizations". *File Handling*. Micro Focus. 1998. Retrieved 27 June 2014.
104. Cutler 2014, Appendix A.
105. McCracken & Golden 1988, § 19.9.
106. Cutler 2014, § 5.8.5.
107. ISO/IEC JTC 1/SC 22/WG 4 2010, § 8.5.2.
108. ^ Jump up to:a b ISO/IEC JTC 1/SC 22/WG 4 2010, § 14.9.24.
109. ISO/IEC JTC 1/SC 22/WG 4 2010, § 14.9.35.
110. ISO/IEC JTC 1/SC 22/WG 4 2010, § 13.18.39.
111. ISO/IEC JTC 1/SC 22/WG 4 2010, § 13.18.59.3.
112. Brown 1976, p. 64.
113. ISO/IEC JTC 1/SC 22/WG 4 2010, p. 835.
114. ISO/IEC JTC 1/SC 22/WG 4 2010, § 14.4.
115. ISO/IEC JTC 1/SC 22/WG 4 2010, § 14.6.3.
116. Field, John; Ramalingam, G. (September 1999). "Identifying Procedural Structure in Cobol Programs" (PDF). PASTE '99. doi:10.1145/381788.316163.ISBN 1581131372.

117. Veerman, Niels; Verhoeven, Ernst-Jan (November 2006). "Cobol minefield detection" (PDF). *Software—Practice and Experience* (Wiley) **36** (14).doi:10.1002/spe.v36:14. Archived from the original on 6 March 2007.
118. ISO/IEC JTC 1/SC 22/WG4 2010, § 14.9.
119. ISO/IEC JTC 1/SC 22/WG 4, §§ 14.9.4, 14.9.22.
120. ISO/IEC JTC 1/SC 22/WG 4 2010, § D.6.5.2.2.
121. ISO/IEC JTC 1/SC 22/WG 4 2010, p. 481.
122. ISO/IEC JTC 1/SC 22/WG 4 2010, pp. 557–559.
123. ISO/IEC JTC 1/SC 22/WG 4 2010, p. 873.
124. ^ Jump up to:a b McCracken & Golden 1988, § 8.4.
125. Examples of compiler support for `ALTER` can be seen in the following:
- Tiffin, Brian (18 September 2013). "September 2014". *GNU Cobol*. Retrieved5 January 2014.
- "The ALTER Statement". *Micro Focus Visual COBOL 2.2 for Visual Studio 2013 COBOL Language Reference*. Micro Focus. Retrieved 5 January 2014.
- "ALTER Statement (Nucleus)" (PDF). *COBOL85 Reference Manual*. Fujitsu. November 1996. p. 555. Retrieved 5 January 2014.
- "ALTER Statement". *Enterprise COBOL for z/OS Language Reference*. IBM. June 2013. Retrieved 5 January 2014.

126. ISO/IEC JTC 1/SC 22/WG 4 2001, § F.1.
127. ^ Jump up to:a b Riehle 1992, p. 125.
128. Shneiderman 1985, pp. 349–350.
129. Dijkstra, Edsger W. (2006). "E. W. Dijkstra Archive: How do we tell truths that might hurt? (EWD498)". University of Texas at Austin. Retrieved August 29, 2007.
130. Tompkins, H. E. (1983). "In defense of teaching structured COBOL as computer science". *ACM SIGPLAN Notices* **18** (4): 86. doi:10.1145/948176.948186. edit
131. Coughlan, Michael (16 March 2014). *Beginning COBOL for Programmers*. Apress. p. 4. ISBN 1430262532. Retrieved 13 August 2014.
132. Sammet 1978b, p. 258.
133. Riehle 1992, p. 126.
134. Riehle 1992, p. 127.
135. Lämmel, Ralf; Verhoef, Chris (November–December 2001). "Cracking the 500-language problem" (PDF). *IEEE Software* **18** (6): 79. doi:10.1109/52.965809.
136. Garfunkel 1987, p. 11.
137. Garfunkel 1987.
138. Raymond, Eric S. (1 October 2004). "COBOL". *The Jargon File, version 4.4.8*.Archived from the original on 30 August 2014. Retrieved 13 December 2014.
139. Brown 1976, p. 53.
140. CODASYL 1969, § II.1.1.
141. Shneiderman 1985, p. 350.
142. Sammet 1961, p. 381.
143. ^ Jump up to:a b Conner 1984, p. ID/10.
144. Marcotty 1981, p. 263.
145. Conner 1984, p. ID/14.

146. Sammet 1961, p. 380.
147. Marcotty 1981, p. 266.
148. Sammet 1978b, p. 255.
149. CODASYL 1969, § 2.2.5.
150. Shneiderman 1985, pp. 348–349.
151. ^ Jump up to:[a] [b] [c] Shneiderman 1985, p. 349.
152. Shneiderman 1985, p. 351.
153. "An interview: Cobol defender". *Computerworld* **18** (37). 10 September 1984. pp. ID/29–ID/32. ISSN 0010-4841. Retrieved 8 June 2014.
154. "Academia needs more support to tackle the IT skills gap" (Press release). Micro Focus. 7 March 2013. Retrieved 4 August 2014.
155. Carr & Kizior 2003, p. 13.
156. Cook, Margaret M. (June 1978). Ghosh, Sakti P.; Liu, Leonard Y., eds. "Data Base Facility for COBOL 80" (PDF). 1978 National Computer Conference. Anaheim, California: AFIPS Press. pp. 1107–1112. Retrieved 2 September 2014. The earliest date that a new COBOL standard could be developed and approved is the year 1980 [...].
157. "Resolutions from WG4 meeting 24 - June 26-28, 2003 Las Vegas, Nevada, USA"(doc). 11 July 2003. p. 1. Retrieved 29 June 2014. a June 2008 revision of the COBOL standard
158. Babcock, Charles (14 July 1986). "Cobol standard add-ons flayed". *Computerworld***20** (28): 1, 12.
159. Marcotty, Michael (1978). Wexelblat, Richard L., ed. "Full text of all questions submitted". History of Programming Languages. Academic Press (published 1981). p. 274. doi:10.1145/800025.1198371. ISBN 0127450408.
160. This can be seen in:
    - "Visual COBOL". *IBM PartnerWorld*. IBM. 21 August 2013. Archived from the original on 12 July 2014. Retrieved 5 February 2014. Micro Focus Visual COBOL delivers the next generation of COBOL development and deployment for Linux x86-64, Linux for System z, AIX, HP/UX, Solaris, and Windows.
    - "COBOL Compilers family". *ibm.com*. IBM. Archived from the original on 23 February 2014. Retrieved 5 February 2014.
    - Tiffin, Brian (4 January 2014). "What platforms are supported by GNU Cobol?". Archived from the original on 14 December 2013. Retrieved 5 February 2014.
161. Coughlan, Michae