UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL BUENOS AIRES

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Algoritmos y Estructuras de Datos

Teoría y Práctica con Lenguaje C/C++

Ejercicios para resolver

Apuntes del Lic. Hugo A. Cuello

Enlaces a archivos con Diagramas de Estructuras, Código Fuente en C++ y en Python.

ÍNDICE COMPACTO

Unidad I – Capítulo I - Una Metodología de Resolución de Problemas 5
Unidad I – Capítulo II..... 21
Unidad I – Capítulo III 35
Unidad I – Capítulo IV 43
Unidad I – Capítulo V 69
Guía de Ejercicios #1..... 88
Unidad II – Capítulo I - Cadenas de Caracteres y Punteros:..... 91
Unidad II – Capítulo II - El tipo struct 136
Unidad II – Capítulo III - Lenguaje de programación..... 151
Unidad II – Capítulo IV - Arreglos 161
Unidad II – Capítulo V - La clase vector de C++..... 240
Guía de ejercicios #2 243
Unidad III – Capítulo I 247
Unidad III – Capítulo II 251
Unidad III – Capítulo III..... 257
Unidad III – Capítulo IV - Conjuntos en C++. La Clase set 261
Guía de ejercicios #3 297
Unidad IV – Capítulo I - Archivos 301
Unidad IV – Capítulo II - Archivos: FILE archivos en C/C++..... 351
Unidad IV – Capítulo III - Archivos: STREAM en C++ 377
Guía de ejercicios #4 439
Unidad V – Capítulo I - Punteros 445
Unidad V – Capítulo II - Estructuras Dinámicas de Datos..... 457
Unidad V – Capítulo III - Recursividad 483
Guía de ejercicios #5 501
ANEXOS..... 557
TEMPLATES 565
Aplicaciones de puntero a función para estructuras dinámicas de datos..... 580
TABLAS – CADENAS ESTILO C 591
TABLAS – CADENAS CLASE STRING..... 595
TABLAS – CARACTERES..... 607
TABLAS – ARCHIVOS FILE*..... 611
TABLAS ARCHIVOS ESTILO C++ STREAMS..... 618
TIPO ENUM 620
TABLAS PARA ARCHIVOS DEL LENGUAJE C..... 625
TABLAS PARA ARCHIVOS DEL LENGUAJE C++ 678
AGREGADOS ADICIONALES - Función fatoi convierte cadena a entero 696
Enlaces a archivos Diagramas de Estructuras, Lenguajes C++ y Python 697

ÍNDICE DE CONTENIDO

Índice Detallado

Unidad I – Capítulo I - Una Metodología de Resolución de Problemas 5
Etapas de la Metodología 5
Ejemplo 1..... 6
Ejemplo 2..... 6
Datos..... 6
Acciones 6
Resultado 6
Enteros..... 7
Ejemplos de constantes enteras con o sin signo: 7
Formato binario de punto fijo con signo 7
Formato binario de punto fijo sin signo..... 7
Ejemplos de tipos enteros y modificadores: 8
Reales. 8
Ejemplos de constantes reales de punto fijo..... 8
Ejemplos de constantes reales de punto flotante en notación matemática 9
Ejemplos de constantes reales de punto flotante en computación..... 9
Ejemplos de constantes reales convertidas para normalizarlas 9
Ejemplos de tipos reales 9
Boolean o Lógicos..... 9
Ejemplo de variables booleanas 9
Caracteres 9
Ejemplos de tipo caracter 10
Ejemplos de constantes carácter imprimibles..... 10
Ejemplos de constantes carácter no imprimibles o de control..... 10
Cadenas de caracteres o Sartas 10
Ejemplo de cadenas 11
Ejemplos de constantes de cadenas de caracteres 11
void 12
Modificadores de tipo..... 12
Ejemplos de enteros con signo y modificadores 12
Ejemplos de enteros sin signo y modificadores..... 12
Modificadores de acceso 12
Ejemplo de const 12
Ejemplo de volatile..... 13
Especificaciones de clase de almacenamiento..... 13
Tipos de Datos Primitivos 14
Solución codificada de ejemplos 15

Ejemplo 1 Hallar hipotenusa triángulo rectángulo 15
Codificación G1Ej01 Hallar hipotenusa triángulo rectángulo 15
Ejemplo 2 Determinar el máximo entre dos valores distintos 16
Codificación G1Ej02 Mayor entre 2 valores distintos 16
Ejemplo 3 Hallar superficie del cuadrado 16
Codificación G1Ej03 Hallar superf. del cuadrado s/ la medida de la hipotenusa 17
Ejemplo 1 modularizado versión 1 17
Codificación G1Ej01 Modularizado 17
Ejemplo 1 modularizado versión 2 18
Codificación G1Ej01 Modularizado Versión II 18
Ejercitación 19
Unidad I – Capítulo II 21
Diseñar una ESTRATEGIA 21
El concepto de *caja negra* 22
El concepto de caja transparente 22
Refinar la estrategia: El ALGORITMO y su prueba 22
Todo algoritmo debe ser 23
Eficiencia y complejidad de los algoritmos 23
Pseudocódigo 23
Diagramas de estructuras 23
Programación estructurada 25
Terminologías 25
Codificación usando fmod, modf, div 28
Codificación de uso de señales 32
Unidad I – Capítulo III 35
Asignación interna 35
Tabla de tipos de datos primitivos en compiladores Turbo V3.0/ Borland V5.5. 35
Asignación externa de entrada 35
Tabla de formato para scanf() 36
Asignación externa de salida 38
Tabla de formato para printf() 38
Códigos de barra invertida o caracteres de control 39
Tabla de barra invertida \ o caracteres de control 39
Codificación de indicadores de formato 40
Modificadores de la Clase ios 40
Tabla de identificadores de formato para setf(), unsetf() y flags() 40
Unidad I – Capítulo IV 43
Clasificación de Sentencias 43
Ejemplos de asignaciones internas 44
Asignación externa de entrada 44
Asignación externa de salida 45
Sentencias de procedimientos equivalente función que retorna void 46
Ejemplos de invocaciones a módulos 46

Sentencias Estructuradas 46
Ejemplo de concatenación de sentencias 46
Ejemplo de selecciones anidadas y en secuencia 48
Ejemplo con selección simple anidadas 53
Con selección simple 53
Con selección múltiple 53
Ejemplos repetición exacta 58
Ejemplos de repeticiones 1-x 59
Ejemplo de repetición 1-x 61
Ejemplos de sentencias simples y estructuradas de ejercicios resueltos de la guía 1 62
Codificación G1Ej05 Potencia de enteros 62
Codificación G1Ej06 Factorial 63
Codificación G1Ej07 Número e 63
Codificación G1Ej07 Número e Optimización 64
Codificación G1Ej08 Sumatoria primeros 100 naturales 65
Codificación G1Ej09 Sumatoria primeros 100 Naturales Optimizada 65
Codificación G1Ej10 Sucesión de Fibonacci 66
Codificación G1Ej15 Primeros 3 números Perfectos 67
Unidad I – Capítulo V 69
Funciones 69
Ejemplo de funciones 73
Conversión de cadena a número 77
Ejemplos de conversión de cadena a número 77
Conversión de entero a cadena 77
Conversión de real –float, double- a cadena 78
Funciones definidas por el usuario 79
Ejemplos de funciones y diseño modular 79
Codificación G1Ej17 Función iif(expLog,x,y) 79
Codificación G1Ej18 Algoritmo de Euclides Función mcd(x,y) 80
Codificación G1Ej19 Función signo(x) 80
Codificación G1Ej20 81
Codificación G1Ej21 Función EsDivis(x,y) 82
Codificación G1Ej22 Examen Matemática Triángulos 82
Codificación G1Ej23 Ternas de valores 85
Guía de Ejercicios #1 88
Unidad II – Capítulo I - Cadenas de Caracteres y Punteros: 91
Punteros 91
Ejemplo de punteros 91
* operador de indirección desreferencia o contenido. 91
Aritmética de punteros 92
Tratamientos de cadenas de caracteres en C / C++ 93
Cadenas como arreglo de caracteres 93
Cadenas como puntero a caracter 95

Ejemplo de new con arreglo de caracteres 95
Funciones para el tratamiento de Cadenas de Caracteres de la librería string.h 96
strcat(cad1,cad4)..... 96
Codificación Cadenas Función: strcat(cad1,cad4) 96
strcmp(cad3,cad4)..... 97
Codificación Cadenas Función: strcmp(cad3,cad4) 97
strcpy(cad1,cad4)..... 97
char *strerror(int *numErr*) 98
strlen(cad3) 98
Codificación Cadenas Función: strlen(cad3)..... 98
strstr(cad3,cad4) 99
Ejemplos de strstr: 99
Codificación Cadenas Función: strstr(cad3,cad4)..... 99
strtok(cad1,cad2) 99
Codificación Cadenas Función: strtok(cad1,cad2)..... 99
strchr(cad3,car)..... 100
Codificación Cadenas Función: strchr(cad3,car)..... 100
strrchr(cad3,car)..... 101
Codificación Cadenas Función: strrchr(cad3,car) 101
strncpy(cad1,cad4,n)..... 101
Codificación Cadenas Función: strncpy(cad1,cad4,n) 101
int strcmpi(const char * cad1, const char * cad2)..... 102
char *strnset(char * cad, char car, size_t n)..... 102
strcspn(cad,subcad) 102
strnicmp(cad1,cad2) 103
strpbrk(cad3,cad4)..... 103
Codificación Cadenas Función: strpbrk(cad3,cad4)..... 103
strncat(cad1,cad3,n)..... 104
Codificación Cadenas Función: strncat(cad1,cad3,n) 104
strncmp(cad3,cad4,n)..... 104
Codificación Cadenas Función: strncmp(cad3,cad4,n) 104
strspn(cad2,cad4)..... 105
Codificación Cadenas Función: strspn(cad2,cad4) 105
strlwr(cad1) –NO ANSI- 105
Codificación Cadenas Función: strlwr(cad1) 105
strupr(cad1) –NO ANSI- 106
Codificación Cadenas Función: strupr(cad1) 106
strrev(cad1) –NO ANSI- 106
Codificación Cadenas Función: strrev(cad1) 106
tolower(car) <ctype.h> 106
Codificación Cadenas Función: tolwr(car)..... 107
toupper(car) <ctype.h> 107
Codificación Cadenas Función: toupper (car)..... 107

Funciones de cadenas definidas por el usuario..... 108
Tratamiento de cadenas al estilo de C++..... 112
Clase string 112
Ejemplos de definición de variables de la clase string 115
Codificación clase string Constructor 115
Codificación clase string Operador de asignación 116
Codificación clase string Función begin - end 117
Codificación clase string Función rbegin - rend..... 117
Codificación clase string Función size() / length() es equivalente..... 118
Codificación clase string Función resize() 118
reserve (size_t n =0); ejemplo: str.reserve(1520);..... 119
max_size(); retorna el valor 4.294.967.291 el cual equivale a 4 Gbytes..... 119
capacity(); retorna el espacio de almacenamiento str.capacity() >= str.size(). 119
Codificación clase string Función clear()..... 119
Codificación clase string Función de conversión c_str()..... 120
Codificación clase string Función find() 120
Codificación clase string función substr()..... 121
Codificación clase string Función replace() 122
Codificación clase string Función append() 123
Codificación clase string Función getline()..... 124
Codificación clase string Operador de concatenación + ... 124
Codificación clase string Función erase() 125
Codificación clase string Función append() 126
Codificación clase string Función copy()..... 127
Codificación clase string Función rfind()..... 127
Codificación clase string Función compare()..... 128
Funciones de Verificación y Conversión de caracteres, librería ctype.h..... 129
isalpha(car) : 129
isupper(car) :..... 129
islower(car) :..... 129
isdigit(car) : 129
isalnum(car) :..... 129
isspace(car) :..... 129
isascii(car) : 129
iscntrl(car) : 129
isgraph(car) :..... 129
isprint(car) : 129
ispunct(car) :..... 129
isxdigit(car) : 129
toupper(car) : 129
tolower(car) : 129
Codificación de cadenas definidas por el usuario 130
Sub-cadena e insertar..... 134

Funciones de conversión de arreglo de caracteres a número..... 135
Funciones de conversión de número a arreglo de caracteres..... 135
Funciones que retornan direcciones de memoria. 135
Unidad II – Capítulo II - El tipo struct 136
Definición y características del tipo struct 136
Pasaje de parámetros de tipo registro 137
Codificación el tipo struct..... 137
Herencia por agragación o composición 143
Herencia por extensión 143
Operador -> (se lo puede leer como operador flecha) 143
Inicializar una estructura 144
Tipo de dato ordinal definido por el usuario 144
enum IdTipo {lista de constantes con nombre}; 144
Ejemplo de tipo por enumeración..... 144
Declaración de variables de tipo enum..... 144
Codificación G1Ej24 Vendedores el tipo struct 146
Unidad II – Capítulo III - Lenguaje de programación..... 151
Los lenguajes de programación se clasifican en..... 151
Lenguaje máquina 151
Lenguajes de bajo nivel 151
Lenguajes de medio nivel 151
Lenguajes de alto nivel 152
Etapas para la generación de código máquina..... 152
Estructura de un programa en C/C++ 153
Ejemplos de include 153
Ejemplos de define 155
Ejemplos de const..... 155
typedef tipoDato IdAlias 156
Ejemplos de typedef 156
Ejemplos de using..... 156
La forma de la función principal es la siguiente:..... 158
Comentarios internos en el programa..... 159
Unidad II – Capítulo IV - Arreglos 161
Tipo de dato estructurado: arreglo..... 161
Punteros con subíndice 164
Relaciones entre matriz y puntero 164
Inicializar..... 165
Arreglos de apuntadores 166
Matriz 171
Elementos de la matriz linealizada 171
Pasaje de parámetros de estructuras de datos combinadas, arreglos-struct:..... 172
Tabla invocaciones y cabecera con estructuras de datos combinadas..... 172
Ejemplo recorrer matriz triangular superior 173

Definición de un arreglo 174
Ejemplo ordenamiento por Intercambio Burbuja optimizada 175
Ejemplo ordenamiento por Selección 175
Ejemplo ordenamiento por Inserción 176
Ejemplo ordenamiento Shell 176
Ejemplo ordenamiento Quick 177
Método de Búsqueda Binaria o Dicotómica para arreglos ordenados 178
Insertar elementos en forma ordenada, para evitar ordenarlos posteriormente 178
Invocación 178
Ejemplo de invocación y cabecera de estructuras combinadas 179
Operaciones con arreglos 179
Ordenamiento de arreglos 179
qsort Función de ordenamiento indirecto 182
Ejemplo con tipo int 183
Ejemplo con tipo float 184
Ejemplo con tipo cadena 184
Ejemplo con tipo struct campo int 185
Ejemplo con tipo struct campo cadena 186
Búsqueda secuencial o lineal 187
Puntero a función 187
Invocación en la Instancia I 188
Invocación en la Instancia II 188
Invocación en la Instancia III 188
Búsqueda Binaria o Dicotómica 191
bsearch función primitiva de búsqueda binaria 192
Ejemplo con tipo int 194
Ejemplo con tipo float 195
Ejemplo con tipo struct 196
Ejemplo con tipo struct con campo int 197
Ejemplo con tipo struct con campo cadena 198
Arreglos paralelos 200
Desplazamiento de elementos en un arreglo 200
Ejemplo de módulo de InsertarEnOrden 202
Ejemplos codificados en C++ 202
Codificación G4Ej03 Generar al azar sin repetir 206
Operaciones de conjuntos utilizando arreglos 209
Codificación G4Ej05 Fusión de arreglos 214
Codificación G4Ej07 Encuesta Discográfica 216
Codificación G4Ej07 Encuesta Discográfica versión stream 220
Codificación G4Ej08 Viajes de Camioneros 223
Codificación G4Ej08 Viajes de Camioneros versión stream 225
Codificación G4Ej17 Rutas Argentinas 228
Codificación G4Ej17 Rutas Argentinas versión stream 234

Unidad II – Capítulo V - La clase vector de C++ 240
Definición del tipo vector 240
Definición de un vector de registros: 240
Capacidad 241
Acceso a elementos: 241
Guía de ejercicios #2 243
Ejercicios con Arreglos. 243
Unidad III – Capítulo I 247
CONJUNTOS 247
Diagramas de Venn o esquema de Euler 247
Las operaciones básicas entre conjunto son 247
UNIÓN 247
INTERSECCIÓN 247
DIFERENCIA 247
DIFERENCIA SIMÉTRICA 248
COMPLEMENTO 248
Relaciones entre conjuntos 248
Pertenencia de un elemento en un conjunto 248
Conjuntos disjuntos 249
Leyes de las operaciones de Conjuntos 249
Unidad III – Capítulo II 251
Tipo de dato estructurado: set of de Turbo Pascal 251
Representación interna en una estructura de datos de tipo conjunto 251
Operadores de conjuntos 254
Las prioridades son: 255
Unidad III – Capítulo III 257
Alternativa al tipo set of a nivel de bit 257
Operaciones de conjuntos a nivel de bits 258
Tabla de Operaciones de Conjuntos a nivel de bits 258
Unión 259
Intersección 259
Diferencia 259
Diferencia simétrica 259
Complemento 259
Pertenencia 260
SubConjunto 260
Unidad III – Capítulo IV - Conjuntos en C++. La Clase set 261
Tipo de dato estructurado de la clase: set 261
Ejemplos: 262
Observaciones: 262
Incorporar elementos al conjunto 262
insert() 262
Eliminar elementos a un conjunto 263

erase() 263
Limpiar o vaciar un conjunto..... 263
clear()..... 263
Determinar si un elemento pertenece a un conjunto 263
contains() 263
Determinar si un conjunto es vacío 263
empty() 263
Recorrer un conjunto de izquierda a derecha 264
Recorrer un conjunto de derecha a izquierda 264
iterator..... 264
begin()..... 264
end()..... 264
reverse_iterator 265
rbegin() 265
rend() 265
Operaciones entre conjuntos..... 265
El complemento lo definimos de la siguiente manera:..... 266
Unión 266
Intersección..... 266
Diferencia 266
Diferencia Simétrica..... 266
Diagrama de Venn..... 267
Ejemplo operaciones entre conjuntos..... 267
G3_Ej01 Operaciones con Conjuntos Encuesta 3 Preguntas 270
G3_Ej02 Encuesta 100 Personas combinación consumo bebidas en un pub 275
G3_EJ03 Operaciones entre Conjuntos por Consola 278
G3_Ej04 M.C.D..... 282
G3_Ej05 Criba de Eratóstenes..... 285
G3_Ej06 Examenes finales AyED, LyED y Fis1..... 286
Sobrecarga de Operadores aplicado al tipo set..... 290
Tabla de Funciones Miembro para el tipo set..... 294
Guía de ejercicios #3 297
Unidad IV – Capítulo I - Archivos 301
Definición y características de un archivo..... 301
Ventajas y desventajas con los archivos..... 301
Clasificación de archivos de acuerdo a su función de uso 302
Datos..... 302
Maestros 302
Novedades o Transacciones 302
Históricos..... 303
Tablas 303
Índices..... 303
Auxiliares 303

Informes o Reportes 304
Seguridad 304
Programas 304
Fuentes 304
Objetos 304
Ejecutables 304
Otros: 305
Documentos 305
Imágenes gráficas 305
Audio 305
Miscelánea 305
Clasificación de lenguajes 305
Bajo nivel i++; 305
Medio nivel i += 1; 305
Alto nivel i = i + 1; 305
Traductores 306
Intérprete 306
Compilador 306
Etapas de procesos de los archivos en el tiempo 307
Modo de apertura de archivos 307
Entrada –Input- 307
Salida –Output- 307
Entrada / Salida -Input/Output- 307
Secuencial 307
Al azar 307
Organizaciones de archivos 308
Secuencial 308
Indexada 308
Alumnos.Idx 309
Relativa 312
Tipos de direccionamiento en una organización relativa 312
Direccionamiento directo 313
Direccionamiento indirecto 313
A continuación se darán varios casos de estudio 313
Estado del puntero al archivo en situación de leer o grabar 315
Procesos clásicos con archivos 315
Corte de Control 315
Ejercicio de Corte de Control de Universidades y Facultados 317
Técnica de HIGH_VALUE o LOW_VALUE 323
Ejemplo de Apareo de archivos 324
Apareo de archivos con técnica de High Value 327
Búsqueda Lineal o Secuencial 331
Ejemplos de la G2Ej4 Crear archivo Articulos 334

Ejemplo de la G2Ej5 Actualizar Precio en Articulos 336
Ejemplo de la G2Ej06 Mayores Precios en Articulos grabar en archivo Mayores 337
Relaciones entre archivos 339
Ejemplo de la G2Ej11 de Facturas a Clientes en Cta./Cte. 339
Ejemplo G2Ej13 Listado de Gastos ordenado Mes-Dia ImpAcum 347
Unidad IV – Capítulo II - Archivos: FILE archivos en C/C++ 351
Clasificación de archivos en el lenguaje C/C++ 351
Estilo FILE * 351
Archivo de texto 351
fopen() 352
Modo de apertura 353
freopen() 354
Funciones de lectura en archivo de texto 355
Funciones de escritura en archivo de texto 355
Ejemplo de archivo de texto 355
Agrega nuevas componentes al archivo luego lo recorre para leer 355
Lee archivo de texto y muestra en pantalla 356
Ingresa datos por teclado y graba en archivo de texto 358
Lee archivo de texto y graba en archivo binario 359
Lee archivo binario y muestra en pantalla 360
Lee archivo binario y graba en archivo de texto 361
Archivo binario 362
Abrir un archivo 364
fopen() 364
Modo de apertura 365
Detectar marca fin de archivo 365
feof() 365
Ejemplo: 366
Determinar posición del puntero y el tamaño del archivo 366
ftell() 366
Calcular el tamaño en cantidad de componentes 367
Ejemplos 367
Mover el puntero del archivo a una nueva posición 367
fseek() 367
Mover el puntero del archivo al comienzo 368
rewind 368
Ubicar el puntero a una posición n 368
Para ubicarnos al final del archivo, la sentencia será 368
Ubicar el puntero sobre la última componente grabada 368
Determinar el tamaño del archivo 368
Función de usuario determina el tamaño del archivo en cantidad de bytes: 368
Operación de lectura en un archivo binario 369
fread() 369

Operación de escritura en un archivo binario 370
fwrite() 370
Cerrar el archivo 370
fclose() 370
Cerrar todos los archivos abiertos 370
fcloseall() 370
Renombrar un archivo 370
rename() 370
Eliminar un archivo 371
remove() 371
Función de usuario equivalente a la sentencia de Pascal: truncate() 371
Definimos las siguientes variables puntero a FILE, local al módulo main: 371
Invocamos al módulo desde la función main, para abrir estos archivos: 371
Módulo Abrir, cabecera y cuerpo de la función: 371
Ejemplo de posición del puntero: 372
Propiedades de los stream 374
Acceso para Leer / Grabar -Read/Write- 374
Texto/ Binario -*Text / Binary*- 374
Buffer 375
Indicadores 375
Indicador de Error 375
Indicador Fin-De-Archivo 375
Indicador de posición 375
Funciones streams 375
Unidad IV – Capítulo III - Archivos: STREAM en C++ 377
Stream o flujo de datos externo en C++ 377
Difference Table: 378
Indicadores y Manipuladores 378
Manipuladores definidos en <iomanip.h> 378
Ejemplos de uso manipuladores iomanip 379
Manipuladores sin parámetros definidos en la clase ios 379
Ejemplos de manipuladores sin parámetros de la clase ios 380
Codificación de manipuladores sin parámetros de la clase ios 380
Funciones miembro definidas en la clase ios 381
Máscaras de modificadores 381
Indicadores de la clase ios 382
Algunas funciones miembros <ios.h> 382
definidas en las clases istream, ostream, iostream 382
cout.precision() 383
cout.fill() 384
Indicadores y Manipuladores de Entrada / Salida 385
Entrada/Salida con formato <iostream> 385
Tabla de Indicadores con cout.setf(), cout.unsetf(), cout.flags() 385

Ejemplos de indicadores 385
Manipuladores sin argumentos 387
Ejemplos 387
Manipuladores parametrizados de <iomanip> 387
Tabla Manipuladores de <iomanip> 387
Codificaciòn de Manipuladores en <iomanip.n> 392
Funciones miembro booleanas 393
Tabla de estado de I-O 394
Ejemplos 394
Funciones útiles: good(), fail(), eof(), bad(), rdstate() o clear() 394
Ejemplo de fichero previamente abierto 395
Entradas y Salidas de archivos stream 395
Funciones miembro para abrir archivo 396
Nomenclatura utilizada 396
Apertura del archivo 396
La sintaxis de apertura de archivo de Entrada es 397
Apertura de archivos en modo Entrada 397
Ejemplos 397
La sintaxis de apertura de archivo de Salida es 398
Apertura de archivo en modo Salida 398
La sintaxis de apertura de archivo de Entrada-Salida es 398
Apertura de archivos en modo Entrada-Salida 398
Operaciones de lectura en archivos de texto –incluye dispositivo por omisión- 399
Funciones interesantes de cin 400
Formatear la entrada 400
Funciones manipuladoras con parámetros 400
Manipulador setw 400
Manipulador setbase 402
Manipuladores setiosflags y resetiosflags 402
Manipuladores sin parámetros 403
Manipuladores dec, hex y oct 403
Permite cambiar la base de numeración de las entradas de enteros: 403
El cambio persiste hasta un nuevo cambio de base. 403
Función ws 404
Función width() 405
Función setf() 405
Función unsetf() 405
Función flags() 406
Función get 406
Función getline 407
Función peek() 408
Función putback() 408
Función get() 408

Archivos Binarios en C++ 409
Modos de Apertura de archivos binarios 409
Modificadores de apertura de archivos binarios 410
Operaciones de lectura y grabación 411
Leer en archivo binario 411
Proceso de lectura secuencial a todas sus componentes 411
Grabar en archivo binario 412
Leer y / o grabar en el mismo archivo binario 412
Cerrar archivo 412
Funciones boolean 413
Obtener la cantidad de bytes o caracteres leídos 413
Mover el ptr a otra posición con direccionamiento directo 413
Determinar la ubicación del puntero 414
Limpiar flag de error 414
Comandos del sistema para archivos binarios 414
Redirecciona la entrada a un archivo 415
Redirecciona la salida a un archivo 415
Versión stream G4Ej04 Creación archivo de Artículos 415
Versión stream G4Ej05 Actualización Precio en Artículos 416
G4Ej06 Precios Mayores de Articulos grabar en Mayores 417
Versión Corte de Control con stream 419
Versión apareo de archivos técnica High Value y stream 422
Versión stream ejercicio Facturación a Clientes en Cta./Cte. 425
Versión stream de Gastos anuales 434
Guía de ejercicios #4 439
Unidad V – Capítulo I - Punteros 445
MAPA DE MEMORIA 445
Punteros 447
Puntero Cercano 447
Puntero Lejano 447
Puntero Externo 447
Puntero Empotrado 447
Puntero Colgado o Desbocado 448
Puntero indefinido 448
Puntero Nulo 448
Puntero Genérico 448
Puntero a un tipo de dato 448
Puntero constante 448
Puntero a constante 448
Puntero constante a constante 449
Puntero a puntero 449
& Operador de dirección 449
Operador * desreferencia 450

Variables estáticas y variables dinámicas 450
Punto importante a tomar en cuenta 451
Modelos de Memoria 451
TINY (diminuto) 452
SMALL (pequeño) 452
COMPACT (compacto) 452
MEDIUM (medio) 452
LARGE (grande) 453
HUGE (enorme) 453
FLAT (plano) 453
NODO 453
NULL 453
Palabras reservadas que crean y eliminan variables dinámicas 454
Operador new 454
Operador delete 455
Uniones 455
Unidad V – Capítulo II - Estructuras Dinámicas de Datos 457
Estructuras Dinámicas de Datos 457
Con restricciones 458
Pila (Stack) –L.I.F.O.- 458
Funciones para la Pila 459
Cola (Queue) – F.I.F.O.- 460
Funciones para la Cola 461
Sin restricciones 462
Lista (List) –Freedom- 462
Funciones para la Lista 463
Los procesos básicos para el manejo de la estructura del tipo lista son 465
Estructuras dinámicas lineales 470
Cola circular 470
Listas doblemente enlazadas 472
Estructuras dinámicas no lineales 477
Árbol binario 477
Unidad V – Capítulo III - Recursividad 483
Directa o simple 483
Indirecta 483
Ventajas y desventajas en la recursividad 485
Ejemplo axb 486
Ejemplo BusBin 487
Ejemplo Crazy 488
Ejemplo DolorCabeza 489
Ejemplo DolorCabeza2 490
Ejemplo Factorial 491
Ejemplo Fibonacci 492

Ejemplo Indirecta 493
Ejemplo Insensata 493
Ejemplo maxComDiv 495
Ejemplo Mistery 495
Ejemplo Palíndromo 497
Ejemplo Pot2 498
Ejemplo Puzzle 499
Guía de ejercicios #5 501
Ejercicios con Recursividad 510
Funciones para la Pila –Stack- 511
Funciones para la Cola –Queue- 512
Funciones para la Lista –List- (primera parte) 513
Módulos para Listas (segunda parte) 515
Vector de punteros a Listas de días con importes acumulados 517
DG5E04GtArcAuxMesLstDias.hpp 521
Lista única de meses y dias 523
DG5E04GtLstU.hpp 526
Lista y SubListas Opc. 1 527
DG5E04GtOpc1.hpp 530
Lista y SubListas Opc. 2 532
DG5E04GtOpc2.hpp 536
Lista y Sublistas Opc. 3 539
DG5E04GtOpc3.hpp 542
Cola Circular 545
Lista doblemente enlazada 547
Codificación G5Ej Suma de Polinomios 552
ANEXOS 557
Codificación Ventas Indexado con Árbol Binario Búsqueda (ABB) 562
TEMPLATES 565
Versiones de Pila, Cola y Lista con templates 565
Pila con template 565
Cola con template 569
Lista con template 572
Punteros a funciones 577
Ejemplo de uso de puntero a función 578
Ejemplo de Operaciones aritméticas con punteros a funciones 578
Ejemplo de punteros a función como parámetros de otra función 579
Aplicaciones de puntero a función para estructuras dinámicas de datos 580
Funciones para criterios 580
Instancias 580
Desde la Invocación 580
Desde la Cabecera 581
Desde el cuerpo 581

Templates y Criterios con Pilas, Colas y Listas 581
TABLAS – CADENAS ESTILO C 591
Resumen de funciones de string C++ 594
TABLAS – CADENAS CLASE STRING 595
TABLAS – CARACTERES 607
Obtener la Fecha y Hora del Sistema 609
TABLAS – ARCHIVOS FILE* 611
TABLAS ARCHIVOS ESTILO C++ STREAMS 618
Input/Output library 618
Tabla funciones miembro para archivos de Texto y Binario estilo C++ stream 619
TIPO ENUM 620
Conclusión 623
Sitios web de información empleada en el documento 624
TABLAS PARA ARCHIVOS DEL LENGUAJE C 625
Operaciones con archivos 628
TABLAS PARA ARCHIVOS DEL LENGUAJE C++ 678
Funciones miembro de la clase ios 679
AGREGADOS ADICIONALES - Función fatoi convierte cadena a entero 696
Enlaces a archivos 697
Diagramas de Estructuras en Visio: 697
Guía #1 Algoritmos. 697
Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto. 697
Guía #4 Estr. Datos: Arhivos Binarios. 697
Guía #5 Estr. Datos: Dinámicas Lineales y No Lineales. 697
Código Fuente en C++ Borland / cygwin GCC v 20 697
Guía #1 Algoritmos. 697
Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto. 697
Guía #3 Conjuntos. 697
Guía #4 Archivos Binarios. 697
Guía #5 Estr. Datos: Dinámicas Lineales y No Lineales. 697
Código Fuente en Python V 12. 697
Guía #1 Algoritmos. 697
Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto. 697
Guía # 3: Conjuntos. 697
Guía # 4: Archivos Binarios. 697
Guía # 5: Estr. Datos: Dinámicas Lineales y No Lineales. 697
Código Fuente en Embarcadfero C++ Builder 12.1 697
Guía #1 Algoritmos VCL Proyectos. 697
Guía #1 Algoritmos CONSOLA Proyectos. 697
Guía #2 Estr. Datos Registro + Arreglos + Archivo de Texto VCL Proyectos 697
Guía #2 Estr. Datos Registro + Arreglos + Archivo de Texto CONSOLA Proyectos. 697

Unidad I — Capítulo I

Unidad I – Capítulo I - Una Metodología de Resolución de Problemas 1

Etapas de la Metodología

- **COMPRENDER el problema.**
- **Diseñar una ESTRATEGIA.**
- **Refinar la estrategia: El ALGORITMO y su prueba.**
- **CODIFICAR el algoritmo en un Lenguaje de computadora.**
- **Corregir errores de compilación.**
- **EJECUTAR el programa:** Con una muestra de datos bien elaborada, que incluya todos los caminos del algoritmo, con valores de datos que incluyan puntos límites, incluso sin ningún dato, valores críticos y no tanto cantidad sino variedad.
- **Depurar errores de LÓGICA:** Esto implica volver hacia atrás en las etapas, reacomodar la situación, debido a errores cometidos anteriormente, produciendo un mayor costo a si se hubiese detectado en su momento oportuno.

Gráfico I

Gráfico II

[1] **UNA METODOLOGÍA DE RESOLUCIÓN DE PROBLEMAS.** Jorge D. Muchnik..ex Prof. Titular de Sintaxis y Semántica de Lenguajes de la U.T.N. F.R.B.A. Q.E.P.D.

En el **gráfico I** se observa un mayor retroceso en las etapas, debido a errores ocasionados en etapas previas y detectados en etapas posteriores. En cambio en el **gráfico II** se observa que los errores cometidos se han detectado en la misma etapa en que ocurrieron, es decir al pasar a una próxima etapa no debemos arrastrar errores de etapas previas. **Nuestros proyectos deben corresponderse al gráfico II.**

COMPRENDER el problema

Problema Elemental

Un problema es elemental cuando es sencillo, simple o *evidente*, para poder comprenderlo. Se destacan las siguientes partes: **Datos**, **Acciones** y **Resultados**. Un problema elemental puede ser de:

Evaluación

Con una **ACCIÓN**, p.e. un cálculo, obtenemos la solución.

Ejemplo 1

Dados los catetos de un triángulo rectángulo, hallar la medida de su hipotenusa.

Decisión

Es necesario establecer una pregunta para obtener la alternativa adecuada, la cual podrá resultar en verdadero o falso.

Ejemplo 2

Dados 2 valores a y b enteros y distintos, emitir un cartel apropiado que dé a conocer cuál de ellos es mayor.

Identificar las PARTES PRINCIPALES del problema.

Datos

¿Cuáles son los **DATOS**?, ¿cuántos?, ¿en qué instante se conocen?

Acciones

¿Cuáles son las **ACCIONES** a realizar y en qué orden lógico?

Resultado

¿Cuáles son los **RESULTADOS**?, ¿cuántos?, ¿en qué instante se informan?

Observación: el orden lógico de **Datos, Acciones y Resultados**, para su comprensión puede ser modificado por el siguiente orden práctico de **Datos, Resultados y Acciones**; ya que los datos como los resultados los obtenemos desde el mismo enunciado, mientras que las acciones deben ser pensadas por nosotros.

Establecer el concepto de TIPO o CLASE o DOMINIO de datos y/o resultados.

No se deben considerar a los datos y/o resultados como un caso único, particular o singular, sino, mas bien, como pertenecientes a una clase o tipo, es decir, se deben generalizar, pluralizar, para que nuestro *algoritmo* funcione para ese conjunto de datos. A continuación se presentan los *tipos de datos primitivos* que maneja generalmente una computadora:

Enteros

Constituidos solamente por su parte entera, vale decir, no debe existir la coma o punto decimal. Presentan dos variantes.

Con signo

Pierde un bit para poder establecer el signo del número. La convención es: se utiliza el bit de la extrema izquierda o sea el de mayor peso; un cero indica que el número es positivo, un uno que es negativo y se encuentra representado en complemento a dos, es decir, se reemplazan ceros por unos y unos por ceros y luego se le suma uno, el resultado final corresponde al valor real. Notar la diferencia en 1, de los valores negativos con respecto a los positivos. Ver Tabla 1.

Sin signo

Debido a que no se requiere el signo del número se aprovechan todos los bits del formato para representar el valor. Ver Tabla 2.

Ejemplos de constantes enteras con o sin signo:

4675, -29, 0, -25678, 3, 0x4b3a, 123, 027

Las constantes enteras indicadas anteriormente representan valores en base diez (decimal), los cuales no van precedidos por el valor cero, en cambio las que van precedidas por cero son valores representados en base ocho (octal), mientras las que van precedidas por un cero y una letra x o X, representan valores en base dieciséis (hexadecimal).

Formato binario de punto fijo con signo

Tabla 1.

TIPO	Cant. Bytes	Desde	Hasta	Resultado	
				Desde	Hasta
short	2	-2^{15}	$2^{15} - 1$	-32768	32767
int, long	4	-2^{31}	$2^{31} - 1$	-2147483648	2147483647

Formato binario de punto fijo sin signo

Tabla 2.

TIPO	Cant. Bytes	Desde	Hasta	Resultado	
				Desde	Hasta
unsigned short	2	0	$2^{16} - 1$	0	65535
unsigned int, unsigned long	4	0	$2^{32} - 1$	0	4294967295

bit de signo (positivo).

0	1	0	0	1	0	1	1

los bits restantes corresponden a la representación del número dado en forma directa.
1001011 = 1+2+8+64 = **+75**

bit de signo (negativo).

1	1	0	0	1	0	1	1

los bits restantes corresponden a la representación del número dado en complemento a 2.
1001011 se reemplazan 0's x 1's y 1's x 0's
0110100 luego sumamos 1 = 0110101 =
= 1 + 4 + 16 + 32 = **-53**

En el formato **sin signo** todos los bits se utilizan para representar al número

0	1	1	0	1	0	1	1

Nro. = 1+2+8+32+64 = **107**

1	1	0	0	1	0	1	0

Nro. = 2+8+64+128 = **202**

Ejemplos de tipos enteros y modificadores:

short a;
int b, c;
long d;
unsigned e, f, g;
unsigned short h;
unsigned long i;

Reales.

Los números reales presentan dos partes, una parte entera, que se sitúa a la izquierda de la coma o punto decimal, y de otra parte decimal, que se sitúa a la derecha, del carácter separador. Los números reales son con signo y presentan dos variantes.

Notación de punto fijo

Es la notación real en el que la coma o punto decimal se localiza en su posición indicada.

Ejemplos de constantes reales de punto fijo

3.1415926, -3493.87, 3.0 0.0032, 15.66, -21.492012

Notación en punto flotante o científica

Se lo representa constituido por una parte denominada parte **fraccionaria o mantisa** y otra parte denominada **característica o exponencial** y ambas partes su signo puede ser positivo o negativo. El valor numérico suele expresarse normalizado, es decir, su parte entera es siempre cero, y su parte fraccionaria el dígito adyacente a la coma o punto decimal es distinto de cero. Es la notación en el que la coma o punto decimal localizado en una posición debe ser desplazado hacia la izquierda si el exponente tiene signo negativo, o desplazarse hacia la derecha si el exponente posee un signo positivo. Esta notación es utilizada para expresar valores muy pequeños o muy grandes, como una forma de abreviar su notación.

Ejemplos de constantes reales de punto flotante en notación matemática

$0.3217 * 10^4$, $0.6 * 10^6$, $-0.4 * 10^{-3}$, $-0.732 * 10^6$.

En la notación computacional se reemplaza el * y el 10 por el símbolo e o E.

La parte a la izquierda del símbolo e o E representa la parte **fraccionaria o mantisa**, mientras que la parte a la derecha del mismo símbolo representa la parte **exponencial o característica**.

Ejemplos de constantes reales de punto flotante en computación

0.3217e4, 0.6E6, -0.4e-3, -0.732E6, 3.725e-3, 1.0e6

Son ejemplos equivalentes a la notación tradicional matemática.

Ejemplos de constantes reales convertidas para normalizarlas

732.5051 normalizado es $0.7325051 * 10^3 = 0.7325051e3$
-0.005612 normalizado es $-0.5612 * 10^{-2} = -0.5612e^{-2}$

Tabla 3.

TIPO	Cant. Bytes	Desde	Hasta	Dígitos Significativos
float	4	$1.5x10^{-45}$	$3.4x10^{38}$	7 - 8
double	8	$5.0x10^{-324}$	$1.7x10^{308}$	15 - 16
long double	10	$3.4x10^{-4932}$	$1.1x10^{4932}$	19 - 20

Ejemplos de tipos reales

float a, b;
double c;
long double d, e;

Boolean o Lógicos

Sólo dos valores son posibles: **false** o **true**, es decir, para indicar, falso o verdadero. El tipo de dato se lo indica con la palabra reservada bool y el tamaño que ocupa en la memoria es de un byte. Un valor de **false** está representado por el **valor cero**, mientras un valor de **true** es un **valor distinto de cero**. **Asignar false** a una variable de tipo bool **es asignar el valor cero**; mientras que **asignar un valor true** a una variable de tipo bool es **asignar el valor uno**.

Ejemplo de variables booleanas

bool existe, negativo = **false**;
enum bool {false, true};

Caracteres

Utiliza un byte y el carácter debe ir encerrado entre apóstrofos, como en los siguientes ejemplos: **'A', '3', '+', ' '**, en el último ejemplo entre los apóstrofos el carácter es el espacio en blanco. Los caracteres o símbolos que utiliza la computadora son representados c/u. de ellos por un código numérico. Por ejemplo la letra 'A' tiene el código 65, la 'B' 66, ..., la 'Z' 90, la 'a' tiene el código 97, la 'b' 98, ..., la 'z' 122. El

dígito ‘0’ tiene el código 48, el ‘1’ 49, ..., el ‘9’ 57. Los demás símbolos clasificados como aritméticos, gramaticales o símbolos especiales tienen sus respectivos códigos. Otros códigos son utilizados para indicar controles a diferentes dispositivos periféricos, como ser, el teclado, la impresora, la pantalla, el modem. El código *ASCII* –Código de Intercambio de Información Standard Americano- Figura 1, es utilizado por las PC’s y mainframes de distintos fabricantes de computadoras, en cambio IBM en sus mainframes adoptó un código particular, el *EBCDIC*. El código ASCII originalmente estaba formado por 7 bits, esto daba una posibilidad de 128 combinaciones diferentes, luego se amplió a 8 bits, por lo cual se expandió a 256 posibles combinaciones o representaciones de símbolos diferentes. Los primeros 128 símbolos corresponden al ASCII STANDARD y sus símbolos son siempre los mismos para distintos fabricantes de computadoras, en cambio los 128 símbolos restantes se denominan ASCII EXTENDIDO y cada fabricante representa los símbolos que crea conveniente utilizar. Del ASCII STANDARD los primeros 32 símbolos –0 al 31- no son utilizados para representar caracteres concretos, sino, mas bien, representan controles a diferentes dispositivos periféricos y son llamados CÓDIGOS TRANSPARENTES, p.e. el código 13 indica la tecla ENTER del teclado, el 27 ESCAPE, el 8 RETROCESO –BACKSPACE-, el 9 TABulación, el 10 AVANCE DE LÍNEA, el 12 AVANCE DE PÁGINA. Utilizando la tecla Alt izquierda en combinación con el keypad numérico se obtienen las representaciones en la pantalla del carácter correspondiente al código tipeado, p.e. Alt izq.+ 92, luego, al soltar la tecla Alt se emite en el editor de texto que estemos utilizando el carácter ‘\’ -barra invertida-.

Ejemplos de tipo caracter

char car = ‘A’, car2;

Ejemplos de constantes carácter imprimibles

‘A’, ‘a’, ‘3’, ‘+’, ‘ ’

Ejemplos de constantes carácter no imprimibles o de control

‘\n’, ‘\t’, ‘\b’, ‘\f’, ‘\r’, ‘\v’

Cadenas de caracteres o Sartas

Se construyen a partir del tipo de dato carácter, por lo tanto, una cadena es una **disposición o colección o arreglo de caracteres**. La longitud de la cadena queda establecida entre su origen y el valor de una marca o centinela, siendo este valor indicado por la constante ‘\0’ que se corresponde con el código ASCII cero. Cada carácter ocupa una posición que es un byte, siendo la primer posición cero, la segunda uno y así sucesivamente. Luego cuando la cadena finalice y después del último carácter de ella se le agrega en el byte adyacente posterior al último carácter, el carácter nulo ‘\0’, suponiendo que este carácter se ubicó en la posición diez, los caracteres de la cadena van de cero a nueve, los cuales son 10 caracteres.

Una constante de cadena va encerrada entre comillas, *p.e.*: **“esto es una cadena”, “3+2”, “”**, el último ejemplo indica la **cadena vacía**, entre las comillas no hay ningún carácter. El carácter nulo ‘\0’ se ubica en la posición cero.

Hay 3 formas para declarar tipos de cadenas, dos de ellas se corresponden a las formas clásicas del lenguaje C y la otra de uso exclusivo para el lenguaje C++, el que además puede utilizar las otras 2 variantes.

En el lenguaje C / C++ los tipos de datos primitivos como los modificadores, son palabras reservadas.

Ejemplo de cadenas

Lenguaje C / C++

char cad[20 + 1]; //disposición o arreglo de caracteres –estático-

char *pcad; //puntero a char –dinámico- y de tamaño fijo decidido en el instante de crear la var.din. que se crea con **new** y se elimina con **delete**. El tamaño asignado a pcad es de 4 bytes por ser una variable de tipo puntero lejano.

Lenguaje C++ exclusivamente

string str1,str2 ("Cadena de caracteres"); //dinámica y tamaño variable. No debe ser utilizada para grabar datos de cadenas en archivos binarios.

Ejemplos de constantes de cadenas de caracteres

"Una cadena de caracteres", "3 + 2", "", "valor a + 3 = x", "3", "$32,95"

void

Es un tipo de dato primitivo que indica vacío o nulo, en otras palabras void establece un tipo especial de dato, que incluye al invocar a una función que no debe retornar ningún valor o en los parámetros para indicar ausencia de ellos, también el tipo void es utilizado para indicar cuando una función debe retornar un puntero genérico.

Modificadores de tipo

signed
unsigned
long
short

signed es el modificador por omisión si no se lo especifica. Establece para los tipos enteros y char, utilizará un bit de signo, representado por el bit de la extrema izquierda y del byte de la extrema izquierda si el tipo de dato su formato tuviera más de un byte. Para indicar signo positivo se utiliza el valor cero y el valor se lo representa en forma directa, en cambio, si el valor fuera negativo el bit de signo se lo indica con el valor uno y el valor del número se encuentra complementado a dos, vale decir, el patrón de bits se lo complementa, reemplazando ceros por unos y unos por ceros y a este complemento se le suma uno, el patrón resultado es el valor representado en el formato.

Ejemplos de enteros con signo y modificadores

short es equivalente a indicarlo como **short int**.
long es equivalente a indicarlo como **long int**.
signed char, signed short, signed int, signed long es equivalente a **char, short, int, long** respectivamente.
En cambio si quisiéramos indicar el valor entero sin signo, se utiliza el modificador *unsigned*, en estos casos el bit de signo, se lo utilizará para representar el número, por lo que se agrega una posición más y es la de mayor peso.

Ejemplos de enteros sin signo y modificadores

unsigned int representa valores enteros entre 0 y $2^{32} – 1$
unsigned short representa valores enteros entre 0 y $2^{16} – 1$
unsigned long representa valores enteros entre 0 y $2^{32} – 1$
unsigned char representa valores entre 0 y 2^{8} - 1

Cabe aclarar que los tipos reales **float**, **double** o **long double** siempre son con signo.

Modificadores de acceso

const
volatile
const volatile

const establece para las variables y/o parámetros que no podrán ser modificados sus valores, solo está permitdo inicializarlas.

Ejemplo de const

```
const int a = 5;
```

luego si hacemos

```
a = 7;
```

será un error.

volatile indica que al compilador que el valor de una variable puede cambiar por medios no explícitos indicados en el programa.

Ejemplo de volatile

```
volatile unsigned short a = *puerto = 0x40;
const volatile unsigned char *port = 0x30;
```

Especificaciones de clase de almacenamiento

extern
static
register
auto

Variables *extern*

Son utilizadas cuando en un proyecto se deben enlazar diferentes programas y aquellas variables globales que requieran ser utilizadas por los distintos programas de este proyecto se las establece como ***extern*** y serán visibles por cada uno de estos módulos compilados por separado. Al igual que las variables globales dos identificadores no deben repetirse, sucede lo mismo con las expecificiones extern. Cuando una declaración crea almacenamiento para una variable se lo denomina definición. Las sentencias extern son declaraciones, pero no definiciones. Simplemente indica que las definiciones existen en alguna otra parte del programa.

Variables *static*

Mantienen sus valores entre llamadas, es decir, aún saliendo de un módulo, aunque no sean reconocidas fuera de ellos. Dicho en otras palabras, mantienen sus valores, son permanentes como las variables globales. Las variables static pueden ser locales o globales.

En el primer caso una variable static funciona en forma semejante a una variable local, con la diferencia en que la variable static solo es conocida dentro de su ámbito declarada. Al igual que una variable global, las variables static retienen su valor entre las diferentes llamadas al módulo.

En el segundo caso otros programas enlazados no la reconocerán, solo por el programa que la declaró.

Variables *register*

Utilizan los registros de la CPU en lugar de posiciones de memoria de la RAM, por lo que el acceso resulta más veloz. Su uso más común está indicado en las variables de control de ciclos exactos, for en el lenguaje C/C++.

```
for (register i = 0; i < MAXVAL; i++)
    cout << i << ' ';
```

Variables *auto*

Representan a las variables locales dentro de los módulos es el valor por defecto u omisión en caso de no indicarse. El sistema se encarga de crear las variables locales y de eliminarlas al abandonar el módulo, dejando el lugar de almacenamiento, del stack disponible para otras variables cuando se invoque a otros módulos. Vale decir, no están ocupando lugar permanentemente. El programador solo menciona que variables locales va a utilizar cuando se invoque al módulo, y estas variables locales serán creadas en forma automática por el sistema cuando se invoque al módulo, las que serán eliminadas automáticamente cuando se abandone el módulo por el mismo sistema.

```
int miFunc() {
  static unsigned int x = 0;
  auto int y = 5;
  long z; // es auto long

  z = 3;
  y++;
  cout << y << ' ';
  x++;
  return x;
}
```

Invocación al módulo *miFunc*, con declaraciones de variables static y auto.

```
  for (unsigned i = 1; i <= 10; i++)
    cout << setw(4) << miFunc() << ' ';
```

Tipos de Datos Primitivos

Tabla de datos primitivos en C++

Tipo	Cant. Bytes	Desde	Hasta	Resultado		Dígitos Signif.
				Desde	Hasta	
char (signed)	1	-2^{7}	$2^{7} - 1$	-128	127	-----
short	2	-2^{15}	$2^{15} – 1$	-32768	32767	-----
int, long	4	-2^{31}	$2^{31} – 1$	-2147483648	2147483647	-----
unsigned short	2	0	$2^{16} – 1$	0	65535	-----
unsigned char	1	0	$2^{8} - 1$	0	255	-----
unsigned int **unsigned long**	4	0	$2^{32} – 1$	0	4294967295	-----
float	4	$\pm1.5x10^{-45}$	$\pm3.4x10^{+38}$	$\pm3.4E^{-38}$	$\pm3.4E^{+38}$	7 - 8
bool	1	false	true	0	1[2]	-----
double	8	$\pm5.0x10^{-324}$	$\pm1.7x10^{+308}$	$\pm1.7E^{-308}$	$\pm1.7E^{+308}$	15 - 16
long double	10	$\pm3.4x10^{-4932}$	$\pm1.1x10^{+4932}$	$\pm3.37E^{-4932}$	$\pm1.18E^{+4932}$	19 - 20
void	0	-----	-----	-----	-----	-----

Problema compuesto

[2] *true* asigna siempre el valor 1 a una variable de tipo **bool**, sin embargo, cualquier valor distinto de cero es considerado como verdadero.

Un problema compuesto está formado por la sucesión de 2 o más problemas elementales. Por lo tanto, si determinamos que un problema es compuesto, lo que debemos hacer es dividirlo en problemas elementales, ya que éste último siempre es más fácil de comprender.

Ej.3: Dados los catetos de un triángulo rectángulo, hallar la superficie de un cuadrado, sabiendo que uno de sus lados es la medida de la hipotenusa, caso contrario, es el doble. Se observa en este ejemplo, que las acciones corresponden a dos problemas elementales, uno de evaluación y el otro de decisión.

Realizar un dibujo o una representación gráfica

Es posibilitar una mayor comprensión al problema dado, ya que un dibujo o gráfico colabora a comprender una idea, pero este dibujo deberá estar acompañado de rótulos o leyendas aclaratorias, como en los siguientes ejemplos de la Figura 2:

Figura 2.

Reenunciar el problema

Es enunciarlo con otras palabras, o cambiando el orden de las frases originales, pero sin modificar ni sacar o agregar los datos dados originalmente o los resultados solicitados*; p.e.*: Hallar la hipotenusa de un triángulo rectángulo, conociendo las medidas de sus catetos.

Solución codificada de ejemplos

Ejemplo 1 Hallar hipotenusa triángulo rectángulo

Codificación G1Ej01 Hallar hipotenusa triángulo rectángulo

```
/*
        Id.Programa: G1Ej01.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dadas las medidas de los catetos de un triangulo
            rectangulo, hallar la medida de su hipotenusa.
        Versión modularizada.
*/

#include <fstream>
```

```
using namespace std;

main() {
  float a,
        b,
        c;

  cout << "Calcular hipotenusa con datos de catetos a y b" << endl;
  cout << "Cateto a: ";
  cin >> a;
  cout << "Cateto b: ";
  cin >> b;
  c = sqrt(pow(a,2) + pow(b,2));
  cout << "Hipotenusa: " << c;
  return 0;
}
```

Ejemplo 2 Determinar el máximo entre dos valores distintos

Codificación G1Ej02 Mayor entre 2 valores distintos

```
/*
          Id.Programa: G1Ej02.cpp
          Autor......: Lic. Hugo Cuello
          Fecha......: julio-2013
          Comentario.: Dados 2 valores enteros y distintos emitir cartel apropiado
                       que informe el mayor entre ellos.
*/

#include <fstream>
using namespace std;

main() {
  int a,
        b;

  cout << "Mostrar cartel que informe el mayor entre 2 enteros diferentes" << endl;
  cout << "Entero a: ";
  cin >> a;
  do {
     cout << "Entero b: ";
     cin >> b;
  }
  while (a == b);
  if (a > b)
     cout << "a > b";
  else
     cout << "b > a";
  return 0;
}
```

Ejemplo 3 Hallar superficie del cuadrado

Codificación G1Ej03 Hallar superf. del cuadrado s/ la medida de la hipotenusa

```
/*
        Id.Programa: G1Ej3.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Conociendo sus catetos a y b, hallar la superficie de
           un cuadrado con lado igual a la medida de la
           hipotenusa de un triangulo rectangulo si es > 100,
           sino el doble de su valor.
*/

#include <fstream>
using namespace std;

main() {
 float a,
       b,
       c,
       SupCuad;

 cout << "Emitir Sup.Cuad con hip.>100 o 2*hip si es <= 100" << endl;
 cout << "Cateto a: ";
 cin >> a;
 cout << "Cateto b: ";
 cin >> b;
 c = sqrt(pow(a,2) + pow(b,2));
 if (c > 100)
   SupCuad = pow(c,2);
 else
   SupCuad = pow(2*c,2);
 cout << "Sup.Cuad.= " << SupCuad << " c/Hipot. = " << c;0
 return 0;
}
```

Ejemplo 1 modularizado versión 1

Codificación G1Ej01 Modularizado

```
/*
        Id.Programa: G1Ej01a.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dados las medidas de los 2 catetos de un triangulo
                     rectangulo, hallar la medida de su hipotenusa.
                     Versión modularizada.
*/

#include <fstream>
using namespace std;
```

```
void ObtDatos(float &cata, float &catb) {
	cout << "Calculo hipotenusa con datos de catetos a y b" << endl;
	cout << "Cateto a: ";
	cin >> cata;
	cout << "Cateto b: ";
	cin >> catb;
}  // ObtDatos

float CalcHipot(float cata, float catb) {
	return sqrt(pow(cata,2) + pow(catb,2));
} // CalcHipot

void EmiteHipot(float hipot) {
	cout << "Hipotenusa: " << hipot << endl;
} // EmiteHipot

main() {
	float a,
		b,
		c;

	ObtDatos(a,b);
	c = CalcHipot(a,b);
	EmiteHipot(c);
	return 0;
}
```

Ejemplo 1 modularizado versión 2

Codificación G1Ej01 Modularizado Versión II

```
/*
	Id.Programa: G1Ej01a2.cpp
	Autor......: Lic. Hugo Cuello
	Fecha......: julio-2013
	Comentario.: Dadas las medidas de los 2 catetos de un triangulo rectangulo,
			hallar la medida de su hipotenusa.
		Version modularizada 2.
*/

#include <fstream>
using namespace std;

void ObtDatos(float &cata, float &catb) {
	cout << "Calculo hipotenusa con datos de catetos a y b" << endl;
	cout << "Cateto a: ";
	cin >> cata;
	cout << "Cateto b: ";
	cin >> catb;
}  // ObtDatos
```

```
float CalcHipot(float cata, float catb) {
        return sqrt(pow(cata,2) + pow(catb,2));
} // CalcHipot

void EmiteHipot(float hipot) {
        cout << "Hipotenusa: " << hipot << endl;
} // EmiteHipot

main() {
        float a,
              b;

        ObtDatos(a,b);
        EmiteHipot(CalcHipot(a,b));
        return 0;
}
```

Ejercitación

En la Guía de ejercicios #1 deberá responder a las siguientes cuestiones referidas a la metodología aprendida:

1) determine por cada uno de los ejercicios propuestos lo siguiente:
   a) Si es un problema simple de evaluación o de decisión o si es un problema compuesto.
   b) Cuales son las partes principales del problema. Detecte cantidad y momentos en que deben ser conocidos los datos y cantidad y momentos en que deban ser mostrados los resultados. Además indique las acciones como quedan definidas en cuanto al orden lógico de los acontecimientos.
   c) A qué clase o tipo de dato corresponden cada uno de los objetos variables.
2) Realizar un dibujo que se ajuste al problema a resolver.
3) Reenuncie el problema.

Unidad I Capítulo II

Unidad I – Capítulo II - Una Metodología de Resolución de Problemas II.

Diseñar una ESTRATEGIA

Es elaborar un plan de acción global. Una planificación en la cual se establece qué debe hacerse. Es identificar las acciones relevantes. Es ver el bosque antes que el propio árbol. Es mirar de arriba hacia abajo, es decir, de lo general a lo particular. El bosque representa lo general, mientras que el árbol representa lo particular, los **detalles**. Por lo tanto, *"en el nivel de estrategia se deben establecer las acciones más importantes del proceso"*, sin entrar en los detalles para no vernos abrumados por ellos, ya que podemos perder el rumbo, desorientarnos y hacer fracasar nuestro objetivo a resolver. La metodología que va de lo general a lo particular se conoce como *diseño descendente* o ***TOP – DOWN***.

La estrategia será pues un conjunto de **acciones relevantes** interrelacionadas, dispuestas en un orden lógico, que tendrá como meta alcanzar un objetivo.

El concepto de Sistema es algo complejo de entender. Lo que hacemos para poder asimilarlo es, dividirlo en problemas menores, más fácil de comprender; si aún algunas de estas partes fuera complejo, volvemos a dividir nuevamente, hasta alcanzar un punto en el cual ya no es necesario seguir dividiendo, debido a su evidencia. La solución del Sistema queda definido cuando c/u. de estas partes brinden la solución esperada y queden establecidas las interrelaciones, conectadas por medio de interfaces, que posibilitan la comunicación entre ellas.

Divide y vencerás es el axioma de la estrategia.

Si tomamos como Sistema un Automóvil, vemos que este es bastante difícil de comprender. Pues bien, nuestra tarea consiste entonces en dividirlo en subsistemas, los cuales podrían ser: Eléctrico, Mecánico, Frenos, Dirección, Chasis entre otros. Notamos que el Sistema Eléctrico, para nosotros es aún complejo, por lo cual decidimos volver a dividir en los siguientes subsistemas: Batería, Alternador, Bujías, Luces y Cableado. Aún podemos continuar con la Batería en: Bornes, Nivel del Líquido, etc..

Sistema AUTOMÓVIL

1. S. ELÉCTRICO
   - 1.1. S. BATERÍA
     - 1.1.1. BORNES
     - 1.1.2. NIVEL DEL LÍQUIDO
   - 1.2. S. ALTERNADOR
   - 1.3. S. BUJÍAS
   - 1.4. S. LUCES Y CABLEADO
2. S. MECÁNICO
3. S. FRENOS
4. S. DIRECCIÓN
5. S. CHASIS-CARROCERÍA

El concepto de *caja negra*

Representa para los científicos una abstracción de detalles, conoce su interfaz, sabe lo que realiza, pero no le interesa como lo hace; es este un concepto, muy importante, ya que ayuda a simplificar notablemente la idea de algún tema.

El concepto de caja transparente

Es lo opuesto, en este caso está representando el poder ver como funcionan las cosas por dentro, los detalles de su mecanismo de acción. Al simple automovilista que solo le interesa manejar su vehículo, ve a este en cada una de sus partes como una caja negra, mientras que el mecánico necesita verlo como una caja transparente. Un televisor es también una caja negra, para el simple usuario que requiere ver un programa televisivo, él sabe qué finalidad cumple dicho aparato, también sabe que requiere de entradas y salidas, digamos alimentación eléctrica, la señal desde la antena o cable o satélite y las salidas del aparato de, imagen y sonido. Por otro lado el técnico en TV verá a ese aparato como a una caja transparente, debido a que requiere poder entender todo su mecanismo interno de funcionamiento para darle solución en caso de algún problema técnico que se presente.

Refinar la estrategia: El ALGORITMO y su prueba

Algoritmo

Es una secuencia de acciones dadas en un orden lógico que conociendo ciertos **datos** debemos obtener ciertos **resultados esperados**. Ciertas acciones deberán ir antes que otras, y en determinados casos ciertas acciones podrán ir antes o después de otras

indistintamente. La palabra deriva del nombre del matemático y astrónomo árabe Alkhôwarîzmi[3] del siglo IX.

Todo algoritmo debe ser

- ***Preciso***, se debe indicar el orden de ejecución en cada paso.
- ***Definido***, cada vez que se lo realice, se deben obtener los mismos resultados.
- ***Finito***, tiene un número determinado de acciones, debe terminar.

Eficiencia y complejidad de los algoritmos

La eficiencia establece un grado en la calidad del algoritmo, considerando los recursos que utiliza, los cuales podrán ser memoria y tiempo. Un algoritmo podrá ser medido de acuerdo a una escala de grados de eficiencia, y tomando un valor N = 10000, establecer el tiempo requerido para finalizar un proceso. La siguiente tabla muestra estos grados de eficiencia.

Tabla de grados de eficiencia de mejor a peor.

Eficiencia	Función	Bucles	Tiempo estimado	Ecuación Propiedades	Proceso
Constante	O(1)	1	Nanosegundos		Intercambio
Logarítmica	$O(Log_2 N)$	17	Microsegundos		Búsqueda binaria
Lineal	O(N)	10000	0.016 segundos	$\sum_{1=1}^{N} c = c * n$	Ciclos simples
Logarítmica lineal	$O(N * \log_2 N)$	133000			Ordenamientos más veloces
Cuadrática	$O(N^2)$	10000^2	0.270 segundos	$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$	Dos ciclos anidados, matrices.
Cúbica	$O(N^3)$	10000^3		$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$	Tres ciclos anidados, tensores.
Polinómica	$O(N^k)$	10000^k	Horas		
Exponencial	$O(2^N)$	2^{10000}	Inmedible		Recursividad
Factorial	O(N!)	10000!	Inmedible	N (N – 1)!	

Las 2 herramientas más comúnmente empleadas para diseñar algoritmos son:

Pseudocódigo

Utiliza palabras en castellano, es decir, en lenguaje natural, para indicar las acciones a realizar.

Diagramas de estructuras

[3] Escribió un tratado sobre manipulación de números y ecuaciones *titulado Kitab al-jabr w'almugabala.*

Utiliza una representación gráfica, denominados símbolos o bloques que indican las acciones a realizar. **Nassi-Schneiderman** también conocido com **Chapin**.

El diseño del algoritmo se basa en dos etapas

- ***Análisis***
- ***Diseño.***

En la etapa de *análisis* se determina *qué* debe realizar.
En la etapa de diseño se determina *cómo* lo debe hacer.

La realización de un problema complejo se lleva a cabo dividiendo el problema original en sub-problemas de menor complejidad, continuando con estas divisiones hasta alcanzar un nivel en el cual ya no es necesario seguir avanzando. Este método se conoce como **diseño descendente** o *modular*. El proceso de **explotar** el problema en etapas y expresar cada paso en forma más detallada se denomina **refinamiento sucesivo**.

Todo programa bien diseñado consta de un **bloque principal** –es el módulo de nivel más alto- y es el que llama a otros **módulos** –de nivel más bajo- los cuales pueden llamar a otros módulos. Los algoritmos diseñados de esta forma se dice que tienen un *diseño modular*. Cada módulo puede ser realizado, probado, depurado en forma independiente del resto.

BLOQUE	SIGNIFICADO
	C, F, R indica Comienzo, Fin del algoritmo y Retorno de un módulo.
	Acción, Asignación interna.
	Asignación externa de entrada.
	Asignación externa de salida.
	Selección simple.
	Selección múltiple.
	Repetición con pre-condición.
	Repetición con post-condición.
	Módulo, invocación o llamada CALL.
	Módulo, cabecera o interfaz.

Programación estructurada

La programación estructurada significa escribir un programa de acuerdo a las siguientes reglas:

- **El algoritmo tiene un diseño modular.**
- **Los módulos son diseñados de modo descendente.**
- **Cada módulo se codifica utilizando las tres estructuras de control básicas:**
  - **Concatenación**
  - **Selección**
  - **Repetición**

Además la programación estructurada es el conjunto de técnicas que han evolucionado desde los primeros trabajos de *Edsgar Dijkstra* y que incorporan:

- **Recursos abstractos.**
- **Diseño descendente.**
- **Estructuras de control básicas.**

El teorema de la programación estructurada establecido en mayo de 1966 por *Böhm* y *Jacopini*, demostró que un *programa propio* puede ser escrito utilizando solamente tres tipos de estructuras de control: la **Concatenación**, la **Selección** y la **Repetición**.

Un programa se define como propio si cumple las siguientes características:

- Posee un solo punto de entrada y uno de salida.
- Existen caminos desde la entrada hasta la salida que se pueden seguir y que pasan por todas las partes del programa.
- Todas las acciones son ejecutables y no existen ciclos o iteraciones infinitas.

Terminologías

Palabras Reservadas

Es un conjunto reducido de alrededor de 50 palabras, las cuales no pueden ser utilizadas para otro propósito excepto para uso exclusivo del compilador, el cual le indicará acciones específicas que realizar. Algunas de las más comunes palabras reservadas son las siguientes: **break, case, char, class, const, continue, default, delete, do, double, else, enum, float, for, if, int, long, new, private, public, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, while**, entre otras.

Identificadores

Un identificador es una palabra creada por el programador, para poder identificar diferentes objetos o elementos presentes en un algoritmo, como ser variables, módulos, constantes con nombres, librerías, campos, tipos de datos. El lenguaje impone

determinadas reglas para estos nombres de identificadores que debemos respetar, las cuales para el lenguaje C/C++ son:

1. El nombre de un identificador debe comenzar con una letra.
2. Luego puede continuar con letras, dígitos o símbolo de subrayado.
3. Existe diferencias entre las mayúsculas de las minúsculas, recordemos que en el lenguaje Pascal estas diferencias no existen.
4. Un identificador no puede ser una palabra reservada.
5. Se reconocen alrededor de n caracteres, más de esta cantidad los restantes caracteres que sobrepasen el límite serán ignorados por el compilador, en el caso de C++ más caracteres son reconocidos, no obstante, a la hora de dar un nombre de identificador, debemos ajustar esta medida a que no sea corta o extensa, sino balanceada a un tamaño conveniente, p.e.: Apellido_y_Nombre, aN, apeNom, de todas ellas la más conveniente sea la última ya que simplifica el nombre por lo que lo hace más aconsejable al momento de escribir este identificador en el algoritmo, cosa que podría aparecer varias veces y además es fácilmente interpretado o significativo dicho nombre, ya que no debe dejar dudas al respecto.

Constante

Es un valor que no cambia a lo largo de un algoritmo. Las constantes pueden ser de distinto tipos como ser:

Enteras con o sin signo: 45, -1234, 0, 3, 28976, -8976, 08, 0x2A.

Reales de punto fijo o de punto flotante –notación científica: 1543.56, -432.007, 0.0, 0.00043, 3.0, 2.34e02, -1.9876E-05, 1E6.

Caracteres: ‘a’, ‘B’, ‘3’, ‘+’, ‘(‘, ‘\0’, ‘\n’, ‘ ‘.

Booleanos o lógicos: false, true.

Cadenas de caracteres: “Cadena de caracteres”, “3+2”, “3”, “”, “abCD12-4”.

Variable

Una variable contiene un valor que podrá ser modificado en el trancurso de ejecución. Las variables deben ser definidas y declaradas. Al **declararlas** le correspondemos un **tipo de dato**, y al **definirlas** le estamos asignando de un **espacio de almacenamiento** en la memoria interna que se corresponde al tipo de dato declarado, además queda establecido otras dos cosas más; que valores puede recibir o aceptar la variable y que tipo de operaciones pueden ser realizadas.

Expresiones

Una expresión está formada por operandos y operadores. Los operandos pueden ser, valores constantes, variables, expresiones propiamente dichas o funciones. Los operadores a su vez pueden ser, según la siguiente clasificación:

Aritméticos

Son los operadores que permiten realizar las operaciones de cálculo aritmético, como ser, suma (+), resta (-), multiplicación (*), división (/), resto (%). El **resultado de evaluar una expresión aritmética produce un número**.

Las **prioridades** de estos operadores son las siguientes:

1. **Suma y resta como operador unario (+, -)**
2. **Multiplicación, división y resto (*, /, %)**
3. **Suma y resta como operador binario (+, -)**

Obtener el valor de x:
x = 8 / 2 * (2 + 2)
Responder:
x =

La función **fmod**, retorna el resto entre un numerador y un denominador, p.e.: r = fmod(7,3) asigna a r de tipo double el valor 1. (math.h)

a % b el resultado es entero si ambos operandos son enteros, caso contrario es real.

Ejemplos

7 / 2 = 3

7 / 2.0 = 3.5 b / float (a)

La función **div** retorna un tipo **div_t** el cociente entero en el campo **quot** y el resto en el campo **rem**, p.e.: x = div(10,3), retorna en x.quot el valor 3 y en x.rem el valor 1; en donde, x es de tipo div_t. (stdlib.h)

Para alterar el orden de prioridad se utilizan los paréntesis (). A igualdad de prioridad se evalúa la expresión de izquierda a derecha.

Ejemplos

2 + 3 * 5 = 17
(2 + 3) * 5 = 25.

Codificación usando fmod, modf, div

```
#include <fstream>
using namespace std;
int main() {
  div_t x;
  double r, pd, pe;
  int c;

  x = div(7,3);
  r = fmod(7,3); // asigna 1
  cout << x.quot << ' ' << x.rem << endl << r << endl; // x.quot es 2; x.rem es 1
  r = 7 % 3;
  cout << r << endl // emite 1
  pd = modf(3.2, &pe); // pd es 0.2; pe es 3
  c = 7 / 3.0; // c es 2
  return 0;
}
```

Relacionales

Son los operadores que permiten realizar las operaciones de comparación, como ser, igualdad (==), distinto (!=), mayor (>), menor (<), mayor o igual (>=), menor o igual (<=). El resultado de evaluar una expresión relacional produce un **resultado lógico de verdadero o falso**.

Las **prioridades** de estos operadores son las siguientes:

1. **Mayor (>), menor (<), mayor o igual (>=), menor o igual (<=).**
2. **Igual (==), distinto (!=).**

Para alterar el orden de prioridad se utilizar los paréntesis (). A igualdad de prioridad se evalúa la expresión de izquierda a derecha.

Los operadores relacionales son de menor prioridad que los operadores aritméticos.

Ejemplos

3 > x es verdadero si 3 es mayor a x, caso contrario es falso.
$x^2 – 2 <= 2 * z$ es falso si el resultado de 2*z es menor a x^2-2.

Lógicos

Son los operadores que permiten realizar las operaciones lógicas, como ser, conjunción (&&), disyunción (||) y la negación (!). El **resultado de evaluar una expresión lógica produce un resultado lógico de verdadero o falso**.

Las **prioridades** de estos operadores son las siguientes:

1. **Negación (!).**
2. **Conjunción (&&).**
3. **Disyunción (||).**

En las **prioridades generales** con los anteriormente indicados para la negación (!) en el mismo nivel que los operadores unarios (+, -), en cambio para los operadores lógicos restantes están en más bajo nivel que los relacionales, la disyunción u o inclusivo (||) tiene un nivel más bajo que la conjunción o y lógico(&&).

Cabe aclarar que en C/C++ un resultado de falso está indicado por un valor entero igual a cero, mientras que un resultado de verdadero está indicado por un valor distinto de cero. La constante boolean false asigna a una variable siempre un valor de cero, mientras que la constante boolean true asigna a una variable siempre un valor de uno.

Ejemplos

3 / x + 1 > h – 2 ^ a + b < x
~existe
m – 2 = sqrt(x) – sqr(2) v z + 3 <> t * g

Prioridades generales
1ro.: +, -, ! (operadores unarios)
2do.: *, /, %
3ro.: +, - (operadores binarios)
4to.: >, <, >=, <=
5to.: ==, !=
6to.: && (operador compuesto conjunción)
7mo.: || (operador compuesto disyunción)

Inicializar

Es asignar un valor inicial a una variable. No todas las variables deben ser inicializadas. Casos típicos de inicializar son los acumuladores –contador, sumador-, señales, cadenas de caracteres que se deban operar en la concatenación.

Ejemplo

```
c = s = 0;          // representa la asignación múltiple
cad = “”;           // asigna la cadena vacía
existe = false;
```

*Ver página 75 por funciones matemáticas primitivas.

Prioridades de operadores

Orden	Categoría	**Operador**	**Representa o hace**
1	Más alta prioridad	() [] ->
:: .	Llamar a función. Índice en arreglo. Selector componente indirección en C++. Acceso de alcance/resolución. Selector componente directo.

Orden	Categoría	**Operador**	**Representa o hace**
2	Unario	! ~ + - ++ -- & *, ->(tipo) sizeof new delete	Negación lógica (NOT). Complemento a nivel de bit. Más unario. Menos unario. Pre incremento o pos incremento. Pre decremento o pos decremento. Dirección. Indirección o des referencia. Molde. Retorna tamaño del operando en bytes. Crea una variable dinámica en C++. Elimina una variable dinámica en C++.
3	Multiplicativo	* / %	Multiplicación. División. Resto (módulo).
4	Adición	+ -	Suma binario. Resto binario.
5	Desplazamiento	<< >>	Desplazamiento a izquierda. Desplazamiento a derecha.
6	Relacional	< <= > >=	Menor que. Menor o igual a. Mayor que. Mayor o igual a.
7	Igualdad	== !=	Igual a. Distinto de.
8	Nivel de bit	&	AND a nivel de bit.
9	Nivel de bit	^	XOR a nivel de bit.
10	Nivel de bit	|	OR a nivel de bit.
11	Lógico conjunción	&&	AND lógico.
12	Lógico disyunción	||	OR lógico.
13	Condicional	? :	(a?x:y representa si a entonces x sino y).
14	Asignación	= *= /= %= += -= &= ^= |= <<= >>=	Asignación simple o interna. Asignación producto. Asignación división. Asignación resto o módulo. Asignación suma. Asignación diferencia. Asignación AND a nivel de bit. Asignación XOR a nivel de bit. Asignación OR a nivel de bit. Asignación desplazamiento a izquierda. Asignación desplazamiento a derecha.
15	Coma	,	Evaluar.

Acumulador

Son variables utilizadas para contar o sumar, siendo sus características las siguientes:

Contador

Son variables que aparecen a ambos miembros o lados de una asignación interna a la que se le suma un valor constante.

Ejemplo

c = c + 1;

Sumador

Son variables que aparecen a ambos miembros o lados de una asignación interna a la que se le suma un valor variable.

Ejemplo

s = s + c;

Auxiliar

Son variables que nos auxilian en ciertos casos triviales, como lo es en el intercambio entre dos variables.

Ejemplo

```
a = 3;
b = 7;
aux = a;
a = b;
b = aux;
```

Señales

Son variables que permiten establecer si ocurrió o no algún evento esperado. Tiene tres instancias, una de inicialización, dos durante un proceso determinar si ocurre un evento esperado, si es así entonces se cambia el valor de inicialización por otro y tres actuar una vez finalizado el proceso de acuerdo al estado en que quedo la variable utilizada como señal.

Ejemplo

Luego de ingresar una lista de valores se solicita informar si se ingresó algún valor negativo. Se procede de la siguiente manera: antes de ingresar algún valor se inicializa una variable, digamos negativo ← false, luego se ingresan de a uno por vez los valores y por c/u. de ellos se averigua si es < 0, si es así entonces se cambia el valor de negativo ← true, por último al finalizar el ingreso de todos los valores se determina, si negativo es verdadero, se emite 'hubo algún valor negativo ingresado' o caso contrario este otro cartel 'no se ingresó ningún valor negativo'.

Codificación de uso de señales

```
negativo = falso;  // inicializa
for (int i = 1; i < 10; i++) {
   cout << "Valor: ";
   cin >> valor;
   if (valor < 0)          //determina el evento
      negativo = true;
}
if (negativo)              //actuar según el evento
   cout << "se ingresó algún valor negativo";
else
   cout << "NO se ingresó ningún valor negativo";
```

Centinela

Son variables que como contenido adoptan un valor esperado en un momento oportuno, p.e. cuando no sabemos la cantidad de datos a ingresar por teclado, en estos casos pautamos que cuando uno de los datos a ingresar su valor sea igual a un valor previamente establecido, indicaremos que a continuación ya no se seguirán ingresando datos para el lote actual. Ejemplo: Se ingresan los apellidos de alumnos y su nota correspondiente en un examen, procesar estos datos, para obtener la nota más alta, la más baja y el promedio del curso, el proceso finaliza al ingresar un apellido cuyo valor sea igual a un '*'.

Máximos y Mínimos

Son variables que utilizamos para guardar el valor de un máximo o mínimo de una lista de valores. Procedemos de la siguiente manera: si el problema es de máximo inicializamos con un valor ficticio que esté por fuera de los posibles valores reales a ingresar en el proceso y que esté por debajo de esos valores reales, de tal manera que aseguremos que al ingresar el primer valor real sea reemplazado por éste, luego los próximos valores que se vayan conociendo podrán reemplazar o no al valor guardado en ese instante en la variable que hemos elegido como máximo, realizando una pregunta si el valor actual ingresado es mayor al valor guardado en la variable máximo. Al terminar el proceso estamos en condiciones de emitir o de realizar cualquier otra acción necesaria con dicha variable. Tratándose de un problema de mínimo presenta características similares, pero inicializando con un valor mayor que cualquier valor real que se ingrese en el proceso y preguntando durante el momento en que se vayan conociendo c/u. de los valores reales si es menor que el valor guardado en la variable elegida para contener el mínimo en el momento actual. Otro método a elegir podría asignarse a la variable máximo o mínimo el primer valor real ingresado, luego se procede de la manera indicada anteriormente, con los restantes datos.

Ej.: Se ingresan edades de personas, informar la persona con mayor edad. Inicializar MayEdad ← 0, luego por cada edad conocida se pregunta Edad > MayEdad por verdad se modifica MayEdad ← Edad, caso contrario no hacemos nada. Al finalizar el ingreso de todas las edades, informamos que la persona de mayor edad tiene MayEdad años.

Otro ejemplo podría ser las temperaturas registradas en alguna región geográfica.

Promedios y Porcentajes

Son cálculos que debemos realizar en el proceso. Por ejemplo se pide los promedios de notas en un examen de Algoritmos, en el cual se saben la cantidad de alumnos y las notas alcanzadas por c/u. de ellos. Se requiere uso de un sumador, el cual vaya sumando las notas de cada alumno, al finalizar de ingresar todas las notas, se obtiene el promedio asignando a Prom ← SumNotas / N, en donde SumNotas es la suma de todas las notas y N la cantidad de alumnos. El siguiente ejemplo solicita obtener los porcentajes de alumnos aprobados y de alumnos reprobados en el mismo examen. Necesitamos un contador de alumnos que aprueban. Este contador se inicializa en cero, luego tras conocer c/u. de las notas de cada alumno y de comparar si es mayor o igual a 4 se sumará uno en dicho contador.

Finalmente, al término de todas las notas, se calculan los porcentajes, de la siguiente forma: PorcApr ← CantApr / N x 100; PorcRepr ← (N – CantApr) / N x 100.

Unidad I Capítulo III

Unidad I – Capítulo III - Sentencias de asignaciones:

Asignación interna

```
var = exp;
var = var2 = exp;  // asignación interna múltiple
```

Asigna exp a var, en donde: var debe ser solamente una variable y exp puede ser alguna de las siguientes formas: cte., var., exp. p.d. o función; siendo **cte.** un valor constante sea numérica entera con o sin signo o real o, no numérica; car., booleana; indicando, car. un valor de tipo char y booleana un valor de tipo bool.

Los tipos enteros son: **int** (4 bytes), **long** (4 bytes), **short** (2 bytes) a los que se le puede aplicar el modificador de signo, **signed** o **unsigned**,

Los tipos reales son: **float** (4 bytes), **double** (8 bytes) o **long double** (10 bytes).

El tipo carácter es: **char** (1 byte).

El tipo booleano es: **bool** (1 byte).

El tipo de variable debe concordar con el tipo de valor.

Se permite la asignación múltiple, evaluándose de derecha a izquierda.

Ejemplo

```
a = b = 0;   // Asigna el valor cero a las variables a y b.
esCerox = x == 0;  //si x es igual a cero, entonces, verdadero asigna 1, sino, asigna cero.
```

Tabla de tipos de datos primitivos en compiladores Turbo V3.0/ Borland V5.5.

Tipo de dato	Expresión	Intervalo	Turbo C/C++ V3.0	Borland C/C++ V5.5
char	$-2^7 .. 2^7 – 1$	-128 .. 127	1	1
unsigned char	$0 .. 2^8 – 1$	0 .. 255	1	1
signed char	$-2^7 .. 2^7 – 1$	-128 .. 127	1	1
int	$-2^{15} .. 2^{15} – 1$	-32768 .. 32767, -147483648..2147483647	2	4
unsigned	$0 .. 2^{16} – 1$	0 .. 65535, 0 .. 4294967295	2	4
signed	$-2^{15} .. 2^{15} – 1$	-32768 .. 32767, 2147483648..2147483647	2	4
short	$-2^{15} .. 2^{15} – 1$	-32768 .. 32767	2	2
unsigned short	$0 .. 2^{16} – 1$	0 .. 65535	2	2
signed short	$-2^{15} .. 2^{15} – 1$	-32768 .. 32767	2	2
long	$-2^{31} .. 2^{31} – 1$	-2147486648 .. 2147483647	4	4
signed long	$-2^{31} .. 2^{31} – 1$	-2147486648 .. 2147483647	4	4
unsigned long	$0..2^{32} - 1$	0 .. 4294967295	4	4
float	3.4E-38 .. 3.4E+38	3.4E-38 .. 3.4E+38	4	4
double	1.7E-308 .. 1.7E+308	1.7E-308 .. 1.7E+308	8	8
long double	3.4E-4932 .. 1.1E+4932	3.4E-4932 .. 1.1E+4932	10	10
bool	En C++	0 es falso, distinto 0 es verdadero	1	1
void	-----	-----	0	0

Asignación externa de entrada

Permite el ingreso de datos desde el exterior por medio de un periférico de entrada de datos, por omisión es el teclado. La ejecución del algoritmo se detiene a espera de ingresar desde el teclado un valor, luego de oprimir la tecla Enter, el valor tipeado se asigna a la variable indicada en su argumento. Debe existir correspondencia entre el tipo de valor tipeado y el tipo declarado por la variable. En C la sentencia a utilizar es: **scanf**, el ingreso de datos es con formato y debe existir una correspondencia entre el formato y el tipo de valor a ingresar desde el teclado. El formato va encerrado entre comillas conteniendo un primer carácter de % que indica el formato y de otro carácter inmediatamente a continuación del % que establece el tipo de formato, siendo estos formatos los siguientes: **scanf**(cad,&var,…);

Tabla de formato para scanf()

Formato	Indica
%d	entero con signo o int.
%i	idem anterior.
%hd	entero corto.
%u	entero sin signo.
%hu	entero corto sin signo.
%x	valor hexadecimal sin signo.
%X	idem anterior.
%o	valor octal sin signo.
%f	real o flotante de 4 bytes.
%L	real long double o doble largo, 10 o 12 bytes según la implementación de uso.
%e	real o notación exponencial.
%E	idem anterior.
%lf	real de doble precisión.
%Lf	real long double.
%c	caracter o char.
%ld	entero largo.
%lu	entero largo sin signo.
%s	cadena de caracteres o char*, es decir, puntero a char, se agrega terminador '\0'.
%p	puntero.
%%	literal símbolo de %.

Ejemplos

```
scanf("%d",&entero); flush(stpPCL);
scanf("%c%f%d",&car,&xReal,&ent); fflush(stpPCL);
scanf("%s",&cad);  fflush(stpPCL); // lee caracteres hasta encontrar un espacio en blanco o tab.
```

La sentencia fflush(stpPCL) vacía el buffer por si quedó algo allí. Después de un scanf es necesario utilizar un fflush y evitar problemas posteriores.

Existen otros formatos para el ingreso de datos para controlar su entrada, p.e.: %[A-Za-z]; o su complemento, %[^a-zA-Z], para ingresar no letras, %[^0-9] para ingresar no dígitos.

En C/C++ existe la posibilidad de ingresar cadenas por medio de la sentencia gets en lugar de ingresar datos desde el exterior con scanf, la diferencia es que no debemos indicar el formato con gets, ya que asume directamente el dato de tipo cadena de

caracteres incorporándole al final del ingreso de la cadena el carácter de terminación '\0'. Además incluye en la cadena los espacios en blanco que separan las palabras.

Ejemplos

```
char cad[20];
gets(cad); // más adelante se verá otra manera mejorada con cin.get()
```

Si tecleamos la frase: esto es una cadena, la cadena cad contendrá la frase indicada incluyendo los espacios que separan cada una de las palabras de la frase.

En C++ el ingreso de datos por defecto u omisión que es el teclado, se utiliza la sentencia **cin**, que se la considera más versátil que scanf, siendo su formato el siguiente:

```
cin >> var1 >> var2 …;
```

cin permite el ingreso de datos desde el exterior al programa, cuyo periférico por omisión es el teclado, la ejecución del programa se detiene en espera del ingreso de los datos, por cada dato tipeado deberá oprimirse la tecla Enter, asignándose el ingreso a la variable indicada por var1 para el primer ingreso de datos, de existir otras variables como argumento de cin se procederá en forma análoga al primer ingreso, asignándose los siguientes datos a las variables indicadas como var2, … respectivamente. Por lo que solo deberán ser indicadas objetos variables en cada uno de los argumentos de cin. Observar además que no es necesario preceder cada variable con el símbolo & ya que el pasaje de parámetros es pasado directamente por dirección, no así con la sentencia scanf que debemos pasar cada variable por referencia anteponiendo el símbolo &. La conversión de tipo es realizada directamente por el proceso, razón por la cual la sentencia cin es sin formato. El **símbolo >> se denomina extractor**, ya que extrae caracteres de un flujo (stream).

Ejemplos

```
int a;
float b;
char c;
cin >> a  >> b >> c;
```

Desde el teclado tipeamos 3 valores el primero un entero, el segundo un valor real y para el tercero un carácter, para cada dato oprimimos la tecla Enter, luego de los cual cada uno de esos valores tipeados son asignados a sus variables correspondientes.

234 Enter
3.14 Enter
M Enter

Luego de esto los contenidos de las variables a, b y c serán respectivamente 234, 3.14 y M, es decir, es como si se hubiese asignado internamente para a = 234; para b = 3.14; y para c = 'M'; salvo que en este caso se solicitan los datos al usuario desde el exterior a través del teclado.

Asignación externa de salida

Permite el egreso de datos hacia el exterior por medio de un periférico de salida de datos, por omisión es la pantalla. La ejecución del algoritmo se detiene a espera de enviar los valores hacia la pantalla.

En C la sentencia a utilizar es: **printf**(cad,exp,…), el egreso de datos es con formato y debe existir una correspondencia entre el formato y el tipo de valor a mostrar hacia la pantalla. El formato va encerrado entre comillas conteniendo un primer carácter de % que indica el formato y de otro carácter que indica el tipo de dato, por otro lado, expresión puede ser un valor constante, o una variable, o una expresión propiamente dicha.v En la siguiente tabla se indican los formatos de datos para **printf**(cad,exp,…)

Tabla de formato para printf()

Formato	Indica
%d	entero con signo o int.
%i	idem anterior.
%hd	entero corto.
%u	entero sin signo.
%hu	Entero corto sin signo.
%x	valor hexadecimal sin signo.
%X	idem anterior.
%o	valor octal sin signo.
%f	real o flotante de 4 bytes.
%L	real long double o doble largo, 10 o 12 bytes según la implementación de uso.
%e	real o notación exponencial.
%E	idem anterior.
%lf	real de doble precisión.
%lu	entero largo sin signo.
%Lf	real long double.
%ld	entero largo.
%c	caracter o char.
%s	cadena de caracteres o char*, es decir, puntero a char, se agrega terminador '\0'.
%p	puntero.
%%	literal símbolo de %.

Ejemplos

```
printf("valor de a es: %d\n",a);
printf("Suma de a+b+c= %d %d %d %d\n",a,b,c,a+b+c);
printf("El carácter es: %c\n",car);
```

Se permite indicar la longitud del valor a mostrar en cantidad de posiciones, como ilustra el siguiente ejemplo:

```
printf("%5d\n",123);  // completa 5 posiciones con 2 espacios en blanco antes del valor.
printf("%05d\n",456);  // completa con 5 posiciones con 2 ceros antes del valor.
printf("%8.2f\n",6789.12);  // completa con 3 esp. en blanco a izq. antes del valor y 2 decimales.
printf("%20s\n",cad);  // alinea la cadena a derecha, completando con espacios a izquierda.
```

```
printf("%-20s\n",cad);  // alinea la cadena a izquierda, completando con espacios a derecha.
```

Códigos de barra invertida o caracteres de control

carácter inmediatamente a continuación del % que establece el tipo de formato, siendo estos formatos los siguientes:

Tabla de barra invertida \ o caracteres de control

Formato	Indica
\b	retroceso.
\f	salto de página.
\n	salto de línea.
\r	retorno de carro.
\t	tabulación horizontal.
\v	tabulación vertical.
\a	Alerta, campana o Bell.
\0	nulo.
\\	literal barra invertida.
\o	constante numérica octal.
\x	constante numérica hexadecimal.
\"	literal comillas.
\'	literal apóstrofos.

En C++ el egreso de datos por defecto u omisión que es la pantalla, se utiliza la sentencia **cout**, que se la considera más versátil que printf, siendo su formato el siguiente:

```
cout << exp1 << exp2 …;
```

cout permite mostrar información hacia el exterior al programa, cuyo periférico por omisión es la pantalla, la ejecución del programa se detiene en espera de poder mostrar la salida la información, por cada argumento indicado como exp1, exp2. La forma de los argumentos indicado como expn, puede tomar la forma de un valor constante o una variable o una expresión propiamente dicha. Si es una constante emite ese propio valor constante, si es una variable emite el contenido de esa variable y si es una expresión propiamente dicha, se resuelve su resultado final, emitiéndose ese resultado final. La conversión de tipo es realizada directamente por el proceso, razón por la cual la sentencia cout es sin formato. El **símbolo << se denomina insertor**, ya que inserta caracteres hacia un flujo (stream).

Indicadores de Formato: Tres funciones miembro (width, precision y fill) que fijan formato de anchura, precisión y carácter de relleno. Es necesario fijar la anchura, precisión y carácter de relleno antes de cada sentencia de escritura.

ANCHURA: cout.**width**(ancho);
DECIMALES: cout.**precision**(nº digitos);
RELLENO: cout.**fill**('carácter');

Ejemplo

Codificación de indicadores de formato

```
// Formato.cpp
#include<iostream.h>
using namespace std;

int main() {
    double numero=123.1234567;

    cout<< "hola" <<"\n";
    cout.width(15);
    cout<< "hola" <<"\n";
    cout.width(15);
    cout.fill('*');
    cout<< "hola"<<"\n";
    cout<<numero <<"\n";
    cout.precision(4);
    cout<<numero <<"\n";
    cout.precision(10);
    cout<<numero;
    return 0;
}
```

Modificadores de la Clase ios

Estos modificadores son pertenecientes a la clase ios. Cuando se activa su valor se mantiene, es decir hay que desactivarlos para volver al formato de salida original.

Fijar indicador

cout.**setf**(ios::identificador|ios::identificador2);

Anular identificador

cout.**unsetf**(ios::identificador|ios::identificador2);

Tabla de identificadores de formato para setf(), unsetf() y flags()

Identificador	Descripción
oct	Devuelve un entero en octal.
hex	Devuelve un entero en hexadecimal.
scientific	Devuelve un número en formato científico.
showpoint	Muestra 6 decimales aunque no sea necesario
showpos	Muestra el signo + en los valores positivos
left	Ajusta la salida a la izquierda.
skipws	Omite los espacios a la izquierda de la salida.
uppercase	Muestra el texto en mayúsculas.

Ejemplo

Codificación de ejemplos con setf(), unsetf()

```
// FormatoIdentificador.cpp

#include <stdio.h>
#include <conio.h>
#include <iostream.h>

void main() {
    float num = 200.0;
    int num1 = 200;

    clrscr();
    cout << num << "\n";
    cout.setf(ios::showpos|ios::showpoint);
    cout << num << "\n";
    cout.setf(ios::scientific);
    cout << num << "\n";
    cout.unsetf(ios::scientific|ios::showpoint|ios::showpos);
    cout << num << "\n";
    cout.setf(ios::hex);
    cout << num1 << "\n";
    getch();
}
```

Ejemplos

```
int a;
float b;
char c;

cout << “El valor de a es: “ << a << endl;
cout << “a+b= “ << (a+b) << endl;
cout << “El precio es: “ << 238.58 << endl;
```

```
// AreaTriangulo.cpp

// Calcula el área de un triángulo conociendo sus tres lados.

#include <iostream.h>
#include <conio.h>
// CalculoAreaTriangulo.cpp
```

```
#include <math.h>

int main() {
  // Definición de variables locales
  float a,
        b,
        c;
  float p,
        at;

  cout << "Ingrese el valor del lado a: ";
  cin >> a;
  cout << "Ingrese el valor del lado b: ";
  cin >> b;
  cout << "Ingrese el valor del lado c: ";
  cin >> c;
  p = (a + b + c) / 2.0;
  at = sqrt(p * (p - a) * (p - b) * (p - c));
  cout << “\nEl área del triángulo de lados “ << a << b << c << “es: “ << at;
  getch();
}
```

```
// CalcFunc.cpp

// Calcula diferentes expresiones para un solo valor

#include <iostream.h>
#include <conio.h>
#include <math.h>

int main() {
  float t;
  double x,y,z;

  cout << "Ingrese el valor de t: ";
  cin >> t;
  x = pow(t,3) – 8 * t + 4;
  y = sin(t) + cos(2 * t);
  z = exp(3 * t + 7);
  cout << "El valor de x es: " << x;
  cout << "\nEl valor de y es: " << y;
  cout << "\nEL valor de z es: " << z;
  getch();
}
```

Unidad I — Capítulo IV

Unidad I – Capítulo IV - Sentencias: Simples y Estructuradas.

Clasificación de Sentencias

Las sentencias son acciones algorítmicas que pueden ser ejecutadas. Hay dos tipos principales de sentencias:

- Sentencias simples
- Sentencias estructuradas

Sentencias simples

Una sentencia simple es una sentencia que no contiene a ninguna otra sentencia.

Sentencia de asignación

<u>Asignación interna</u>

Es utilizada para almacenar valores en variables.

Referencia a variable representa una **variable** y **expresión** representa una **constante** o una **variable** o una **expresión propiamente dicha**. El símbolo ← representa la asignación interna. Se codifica en C/C++ con el símbolo '='.

Se evalúa la expresión y el resultado es asignado a la ubicada en el miembro izquierdo de la asignación. Debe existir una correspondencia de tipos entre, el resultado de la expresión, y el tipo definido para la variable, caso contrario se generará un error de incompatibilidad de tipos. La expresión podrá estar formada por, una constante, una variable, una expresión propiamente dicha, o la invocación a una función; en cambio en el miembro izquierdo de la asignación solo podrá especificarse un objeto variable.

Cualquier valor previo que tuviera la variable en la parte izquierda de la asignación es reemplazado por el nuevo valor asignado, por lo tanto, la **asignación es destructiva**. Cualquier variable situada en la parte derecha de la asignación deberá tener un valor previamente asignado, debido a que deberá leerse su contenido.

Ejemplos de asignaciones internas

a ← 4	a toma el valor constante entero 4
b ← a	b toma el valor de a
b ← b + 3	b toma el valor de b + 3
apeNom ← "Perez Juan""	ApeNom toma el valor constante de cadena
x ← sqrt(25)	¿de qué tipo definiría a x?
car ← 'h'	
cociente ← b div a	
resto ← b mod a	
existe ← true	existe y negativo son de tipo boolean.
negativo ← x < 0	

Codificación

```
a = 4;
b = a;
b = b + 3; // b += 3; alternativa más compacta
strcpy(apeNom,"Juan Perez");
x = sqrt(25);
car = 'h';
cociente = b / a;
resto = b % a;
existe = true;
negativo = x < 0;
```

Funciones de entradas y salidas por omisión

Asignación externa de entrada

var deberá ser una variable. El valor asignado a la variable indicada en el bloque se lee desde un dispositivo periférico, siendo el teclado el dispositivo por defecto. El dato a conocer puede ser leído desde otro dispositivo como ser un archivo en disco. En cualquier caso la ejecución del proceso entra en espera, a efecto de que pueda leerse el dato solicitado. Si la entrada es ingresada por teclado, luego de tipear el valor se deberá presionar la tecla ENTER tras lo cual se reanuda la ejecución del proceso con la

próxima acción que continúa en la secuencia. Deberá existir coincidencia de tipos entre el valor leído y el tipo de la variable. Una cadena de caracteres o un carácter deberá ser ingresado sin comillas o apóstrofos. Si el ingreso es desde un archivo también existe una espera, solo que en estos casos dicha espera es menor, ya que el ingreso del dato se realiza electrónicamente y no requiere de la intervención manual del usuario. Los puntos suspensivos indican que se podrá escribir más de una variable separadas por comas. La ventaja de la asignación externa de entrada con respecto a la asignación interna, es que en distintas ejecuciones del programa podemos ingresar diferentes valores sin modificar el código.

Codificación

```
cin >> a; // el símbolo compuesto >> se denomina extractor.
cin >> b >> c;
cin >> cad; // si el ingreso de la cadena contiene espacios trunca inmediatamente.
gets(cad); // admite cadenas que contengan espacios
cin.get(cad,n); // más versatil que gets(), se aconseja usar esta forma.
cin.ignore(1000,'\n'); // limpia el buffer, usar si el ingreso es por teclado.
```

Asignación externa de salida

Exp puede ser una constante, una variable o bien una expresión propiamente dicha. El resultado de la expresión es enviado al dispositivo periférico que corresponda, por defecto la pantalla, otros dispositivos pueden ser archivos en disco, la impresora. Al igual que la asignación externa de entrada, también existe una espera, a efectos de dar tiempo a que se envíen los resultados, una vez que esto suceda, la ejecución del proceso continúa su marcha con la próxima acción de la secuencia. Se podrá enviar un valor constante, el contenido de una variable o bien el resultado de una expresión propiamente dicha. Los diferentes objetos van separados por comas. Si la salida va destinada a la pantalla, impresora o disco; el término empleado es emitir-mostrar, listar y grabar respectivamente.

Codificación

```
cout << 3; // el símbolo compuesto << se denomina insertor.
cout << a;
cout << "b: " << b << endl; //luego de emitir el endl fuerza un retorno y avance de línea.
cout << endl << endl << "expresión = " << ( c * (a + b) * 2) << endl;
cout << setw(5) << sum << endl; // si sum es menor a 5 díg. rellena con espacios a su izquierda.
cout << cout.fill('*') << setw(5) << sum << endl; // ídem anterior pero con asteriscos.
```

Sentencias de procedimientos equivalente función que retorna void

Ejemplos de invocaciones a módulos

Llamada o invocación para que se ejecute
Imprimir
InterCambio(a, b)
Encontrar(Lista, Nombre, posic)

Sentencias Estructuradas

Las sentencias estructuradas son construcciones compuestas por otras sentencias que son ejecutadas en secuencia, condicionalmente o repetidamente.

Como lo indica la programación estructurada, solo son necesarias tres Estructuras de Control de Programas, y estas son:

1. Concatenación
2. Selección
3. Repetición

Concatenación

La concatenación de sentencias es la unión de dos o más sentencias encerradas dentro de un bloque, denominado bloque de concatenación, este es una unidad lógica de información, vale decir, que **las n sentencias encerradas dentro de un bloque de concatenación, se ve como si fuera una sola sentencia**. Este concepto de bloque es sumamente importante, como se verá a continuación con respecto a las estructuras de la selección y repetición, ya que tiene que ver con el ámbito o alcance que tienen dichas estructuras de control de programas.

En C/C++ un bloque de concatenación se establece por medio de llave abre { el cual indica el comienzo de un nuevo bloque y llave cierra } el cual indica el fin del bloque.

Ejemplo de concatenación de sentencias

```
{ // Comienzo del bloque
   Sentencia-1;
   Sentencia-2;
         .
   Sentencia-n;
} // Fin del bloque
```

Selección

La estructura de control de programa de la Selección permite efectuar decisiones o preguntas a efectos de poder bifurcar la lógica del proceso. Clasificamos a la selección en:

Selección Simple, el cual puede ser incompleta o completa.
Selección Múltiple, el cual puede ser también incompleta o completa.
Operador condicional (? :), es el único operador ternario.

En los casos de la selección incompleta, se estima que en la rama derecha ninguna acción se realizará.

Selección simple

Luego de ejecutar la acción *alfa*, en el punto de entrada se evalúa la *condición*, si el resultado es verdadero, se ejecuta la acción indicada por la acción *beta* –rama izquierda- caso contrario, se ejecuta la acción *gamma* –rama derecha- en ambos casos luego en el punto de salida se ejecuta la acción *delta*. Si la acción *gamma* no está presente –se representa por una línea en diagonal- en ese caso, nada se ejecuta, nos encontramos con una **selección simple incompleta**, de lo contrario sera una **selección simple completa**.

Las acciones *alfa* y *delta* siempre se ejecutan por encontrarse fuera del **alcance** o **ámbito** de la estructura de control de programa. En cambio la acción *beta* solo se realizará si la condición resulta verdadera y *gamma* no se realizará. Por otro lado, solo se ejecutará la acción *gamma* si la condición resulta falsa en cuyo caso la acción *beta* no se realizará.

Ejemplo 1: Asumiendo que *condición* resulta verdadera entonces las acciones que se ejecutan son: alfa, beta y delta.

Ejemplo 2: Asumiendo que *condición* resulta falsa y gamma es vacía, entonces las acciones que se ejecutan son: alfa y delta.

Ejemplo 3: Asumiendo que *condición* resulta falsa y gamma no es vacía, entonces las acciones que se ejecutan son: alfa, gamma y delta.

Con este tipo de estructura es posible realizar cualquier grado de complejidad de alternativas o decisiones o selecciones.

Es válido armar alternativas encadenadas o alternativas dentro de alternativas –en estos casos se denominan *selecciones anidadas*-.

Ejemplo de selecciones anidadas y en secuencia

Sentencia en C/C++ para la selección simple incompleta

```
α;
if (exp.lóg.)
   β;
δ;
```

La sintaxis de la sentencia **if** mostrada anteriormente indica lo siguiente:

Alfa y delta son sentencias que se encuentran fuera del alcance de la sentencia if, mientras que la sentencia beta sí se encuentra dentro de su alcance. También observamos que el alcance es de solo una sentencia, por lo que si queremos ejecutar más de una sentencia debemos encerrarlas dentro de un bloque de concatenación.

La sentencia beta solamente se ejecutará si la exp.lóg. es verdadera, caso contrario, no se ejecutará.

Si exp.lóg. es verdadera entonces las sentencias a ejecutar según el modelo presentado son:

Alfa;
Beta;
Delta;

Si en cambio exp.lóg. su resultado fuera falso entonces las sentencias a ejecutar serán:

Alfa;
Delta;

Sentencia en C/C++ para la selección simple completa

```
α;
if (exp.lóg.)
   β;
else
   γ;
δ;
```

La sintaxis de la sentencia if mostrada anteriormente indica lo siguiente:

Alfa y delta son sentencias que se encuentran fuera del alcance de la sentencia if, mientras que la sentencia beta y gamma sí se encuentran dentro de su alcance. También observamos que el alcance es de solo una sentencia en ambas ramas de la selección, es decir, tanto por el camino de verdad de la exp.lóg. como por el camino de falso de la exp.lóg., en ambos casos si deseamos ejecutar más de una sentencia estamos obligados a encerrar estas sentencia dentro de un bloque de concatenación.

Si exp.lóg. es verdadera entonces las sentencias a ejecutar según el modelo presentado son:

Alfa;
Beta;
Delta;

Si en cambio exp.lóg. su resultado fuera falso entonces las sentencias a ejecutar serán:

Alfa;
Gamma;
Delta;

Observación si Gamma es la sentencia vacía entonces es la selección simple y en la codificación en C/C++ no se indica la partícula else.

La selección simple es **autosuficiente**, es decir, no requiere de ninguna otra sentencia de selección para resolver problemas lógicos en que haya que tomar decisiones. No obstante, observamos que además de la selección simple existe la selección múltiple, el uso de esta última sentencia se debe a otros motivos que serán expuestos más adelante.

Las selecciones simples pueden establecerse en secuencia o bien de manera anidada. Esto lo establecerá la lógica del proceso.

Si una selección simple incompleta va anidada dentro de una selección completa, en estos casos solo existe una partícula else, y la pregunta es, esa única partícula else ¿con qué if se corresponde?. El propio lenguaje nos da la respuesta, se corresponde con el **último if abierto**.

En el diagrama observamos que Beta solo se ejecutará si las condiciones Cond1 y Cond2 ambas son verdaderas, en cambio, Gamma solo se ejecutará si la condición Cond1 es falsa. Observamos también que la selección externa es completa mientras que la selección cuya condición Cond2 está anidada dentro de la selección externa es incompleta. La forma correcta de codificar en C/C++ es la siguiente para estos casos:

```
α;
if (Cond1) {
   if (Cond2)
      β;
}
else
   γ;
δ;
```

Observamos el uso de la concatenación obligada aunque solo sea una única sentencia a ejecutar a efectos de acomodar la lógica del proceso como fuera indicado en el diagrama de estructuras.

De esta manera el único **else** se corresponde no con el último **if** abierto, sino con el primer **if** abierto, vale decir, es el caso contrario de la Cond1.

Selección múltiple

Luego de ejecutar la acción *alfa* en el punto de entrada se evalúa la *expresión ordinal*, el resultado es **comparado por igual** con cada uno de los valores constantes de izquierda a derecha, en cuanto ocurra la primera igualdad se ingresa por la rama correspondiente y se ejecuta la acción *beta* indicada por esa rama, en caso contrario, se ejecuta la acción *gamma*, en caso de estar presente. En todos los casos en el punto de salida se ejecuta la acción *delta*.

Esta estructura de control de programa está **limitada a ciertos procesos**, debido a que la expresión debe ser de tipo ordinal. Los valores constantes también deben ser del mismo tipo ordinal que el de la expresión.

Sentencia en C/C++ para la selección múltiple incompleto

```
α;
switch (exp.ord.) {
   case cte1: β1; …;
   case cte2: β2; …;
   case cte3: β3; …;
         .
   case cten: βn; …;
}
δ;
```

La sintaxis de la sentencia **switch** mostrada anteriormente indica lo siguiente:

Alfa y delta son sentencias que se encuentran fuera del alcance de la sentencia switch, mientras que las sentencias betas sí se encuentra dentro de su alcance. También observamos que el alcance es de varias sentencias, por lo que si queremos ejecutar más de una sentencia no es necesario encerrarlas dentro de un bloque de concatenación.

La sentencias betas solamente se ejecutarán si la exp.ord. coincide con algunos de los valores de caso indicado como cte, caso contrario, no se ejecutará ninguna acción si ningún caso resultó igual al valor de la exp.ord.. La exp.ord. indica una expresión

ordinal, siendo valores ordinales los enteros, el tipo carácter o bool. No son valores ordinales los valores de tipo real ni las cadenas. Debido a esto, la sentencia de selección múltiple no permite realizar cualquier tipo de lógica requerida por un proceso, vale decir, su uso está limitado solo para los casos en que la expresión sea ordinal exclusivamente.

Si la exp.ord. resulta igual a uno de los valores ctes, comenzando desde arriba y hacia abajo, entonces ingresa por esa rama o camino, se ejecutan las sentencias Betas indicadas por dicha rama y luego se continúa ingresando por las restantes ramas o caminos, observar que este accionar es muy diferente a su equivalente en Pascal la sentencia Case. Si por el contrario no queremos ingresar por las siguientes ramas entonces por la rama que hayamos ingresado, la última acción será ejecutar la sentencia **break**. Esta sentencia obliga a forzar la salida del switch, sin ingresar por las ramas posteriores. Si hay una sentencia **return** se abandona, saliendo no solo de esta estructura de control de programa, sino, que además se abandona el módulo.

Sentencia en C/C++ para la selección múltiple completa

```
α;
switch (exp.ord.) {
   case cte1: β1; …;
   case cte2: β2; …;
   case cte3: β3; …;
         .
   case cten: βn; …;
   default: γ;
}
δ;
```

La sintaxis de la sentencia **switch** mostrada anteriormente indica lo siguiente:

Alfa y delta son sentencias que se encuentran fuera del alcance de la sentencia switch, mientras que las sentencias betas sí se encuentra dentro de su alcance, como así también la sentencia gamma. También observamos que el alcance es de varias sentencias, por lo que si queremos ejecutar más de una sentencia no es necesario encerrarlas dentro de un bloque de concatenación.

La sentencias betas solamente se ejecutarán si la exp.ord. coincide con algunos de los valores de caso indicado como **cte**, caso contrario, se ejecutará la acción gamma si ningún caso resultó igual al valor de la exp.ord.. La exp.ord. indica una expresión ordinal, siendo valores ordinales los enteros, el tipo carácter o bool. No son valores ordinales los valores de tipo real ni las cadenas. Debido a esto, la sentencia de selección múltiple no permite realizar cualquier tipo de lógica requerida por un proceso, vale decir, su uso está limitado solo para los casos en que la expresión sea ordinal exclusivamente.

Si la exp.ord. resulta igual a uno de los valores ctes, comenzando desde arriba y hacia abajo, entonces ingresa por esa rama o camino, se ejecutan las sentencias Betas

indicadas por dicha rama y luego se continúa ingresando por las restantes ramas o caminos, observar que este accionar es muy diferente a su equivalente en Pascal la sentencia Case. Si por el contrario no queremos ingresar por las siguientes ramas entonces por la rama que hayamos ingresado, la última acción será ejecutar la sentencia **break**. Esta sentencia obliga a forzar la salida del switch, sin ingresar por las ramas posteriores. Cabe aclarar que por cada una de las ramas de la selección múltiple o switch pueden ejecutarse varias sentencias sin necesidad de armar un bloque de concatenación.

Esta estructura de control de programa de la selección múltiple brinda claridad y simplificación en lugar de utilizar selecciones simples anidadas, solo para aquellos casos en que se pueda utilizar esta alternativa de selección.

Ejemplo con selección simple anidadas

Se ingresa dato de codVehiculo valores entre 1 y 4, indicando 1: auto, 2: camión, 3: moto, 4: camioneta. Se pide emitir un cartel apropiado con el nombre del código de vehículo.

Con selección simple

Con selección múltiple

Se observa la claridad del algoritmo en este último caso.

La codificación en C/C++ para ambos casos es:

```
if (codVehi == 1)
   cout << "auto" << endl;
else
   if  (codVehi == 2)
      cout << "camión" << endl;
   else
       if (codVehi == 3)
          cout << "moto" << endl;
      else
          cout << "camioneta" << endl;
switch (codVehi) {
   case 1:
       cout << "auto" << endl;
       break;
   case 2:
      cout << "camión" << endl;
      break;
   case 3:
      cout << "moto" << endl;
     break;
   default:
      cout << "camioneta" << end;
}
```

Operador condicional (? :)

Es el único operador ternario, es decir, requiere tres operandos. Los operandos de izquierda a derecha son, una expresión condicional, la cual se encuentra antes del operador (?), el valor a realizarse si la expresión condicional es verdedera, la cual se encuentra entre del operador (?) y el operador (:) y, el tercer operando que es el valor a realizarse si la expresión condicional es falsa, la cual se encuentra luego del operador dos puntos (:). La sintaxis correspondiente a la *expresión condicional* es:

exp.lóg. ? β : γ

en donde:

exp. lóg. Es una expresión lógica, cuya evaluación produce un resultado de verdadero o falso.

β representa la acción a ejecutar si la exp. lóg. resulta ser verdadera.

γ representa la acción a ejecutar si la exp. lóg. resulta ser falsa.

Ejemplos

```
int a = 23;
cout << (a > 12 ? "si a es mayor" : "a no es mayor") << endl; // emite: si a es mayor
a = 3;
cout << (a > 12 ? "si a es mayor" : "a no es mayor") << endl; //emite: a no es mayor
```

En ambos casos después de emitir, el señalador de escritura en la pantalla, regresa y avanza una línea, como si fuese un punto y aparte en un lenguaje natural.

Los paréntesis que encierra la expresión condicional son necesarios por la sentencia cout.

```
int x, b = 3, c = 2, h = 5, z = 4;
x = (b * c + 1 <= pow(z,2) / 2) ? b + 2 : z * (c + 5) - h;   // x = 5
```

Repetición

La estructura de control de programa de la repetición, permite elaborar un ciclo o iteración o bucle para que un conjunto de sentencias se ejecuten repetidamente o bien una cantidad indefinida de veces o una cantidad conocida de veces.

Presenta diversos casos que presentamos a continuación:

1. **Repetición pre-condición**
   - *1.1. Indefinida, 0-x , Mientras.*
   - *1.2. Exacta*
     - *1.2.1. Ascendente*
     - *1.2.2. Descendente*
2. **Repetición pos-condición**
   - *2.1. Indefinida, 1-x, Hasta*

Pre-Condición: Indefinida, 0-X, o Mientras

Luego de ejecutarse la acción *alfa* en el punto de entrada se evalúa la *condición*, si el resultado es verdadero se ingresa al ciclo y ejecuta la acción *beta*, luego en el punto

de salida se vuelve otra vez a la cabecera para ser evaluada nuevamente la condición, en caso que resulte falsa se abandona el ciclo por el punto de salida y a continuación se ejecuta la acción *delta*. Puede ocurrir que en el punto de entrada la condición resulte falsa, en ese caso la acción *beta* jamás será ejecutada. Hay que asegurar que si se ingresa al ciclo lograr que la condición en algún momento se haga falsa, para poder abandonar la estructura cíclica. La condición podrá estar controlada por un contador, un sumador, un valor centinela, un valor boolean, una situación de fin de archivo, una condición compuesta, entre otras. Con este tipo de ciclo se pueden llevar a cabo cualquier complejidad de situaciones que requieran el uso de una estructura cíclica. Es válido construir estructuras cíclicas anidadas o en secuencia.

Codificación

```
while (a < 10)
   a++;
while (x + 2 < h && b) {
   cout << x + 2 << endl;
   ++x;
   if (h – x == 1)
      B = true;
}
```

Repetición Exacta o Para ascendente

Luego de ejecutarse la acción *alfa*, en el punto de entrada se realizan las siguientes acciones –realizados por la propia estructura de control-; si el resultado de la *expr.ini*. es **menor o igual** al resultado de la *expr.fin*. entonces el resultado de la *expr.ini*. es asignado a *var*. –variable de control, es una variable contadora del ciclo- y se ingresa al ciclo, ejecutándose la acción *beta*. En el punto de salida, se llevan a cabo las siguientes acciones, -también propias de la estructura de control- si la variable de control *var* es menor al resultado de la *expr.fin*., entonces *var* toma el valor ordinal sucesor del que contiene y se vuelve a ejecutar la acción *beta*. Se abandona el ciclo una vez que *var* haya alcanzado al valor de la *expr.fin*. ejecutándose la acción *delta*. Por otro lado,

estando en el punto de entrada si el valor de la *expr.ini* fuera mayor al valor de la *expr.fin*. directamente se busca el punto de salida y se ejecuta la acción *delta*. El tipo de *var*, la *expr.ini*. y la *expr.fin*. deben ser todas del mismo tipo ordinal.

Este tipo de ciclo es de uso limitado, ya que solo sirve para valores de tipo ordinal. De todas maneras en ciertos casos es más aconsejable su uso cuando se conozca de antemano, la cantidad de veces que deba ser ejecutado. Una de las ventajas es que no se debe llevar la cuenta del contador, ya que lo gestiona la propia estructura de control.

La cantidad de veces que se repetirán las acciones dentro del ciclo, queda establecida de la siguiente manera:

Cant.Rep.Asc. ← valor expr.fin. – valor expr.ini. + 1

Repetición Exacta o Para descendente

Luego de ejecutarse la acción *alfa*, en el punto de entrada, se realizan por la propia estructura de control, las siguientes acciones; si el resultado de la *expr.ini*. es **mayor o igual** al resultado de la *expr.fin*. entonces el resultado de la *expr.ini*. es asignado a *var* –variable de control, es una variable contadora del ciclo- y se ingresa al ciclo, ejecutándose la acción *beta*. En el punto de salida, se llevan a cabo las siguientes acciones, -también propias de la estructura de control- si la variable de control *var* es mayor al resultado de la *expr.fin*., entonces *var* toma el valor ordinal predecesor del que contiene y se vuelve a ejecutar la acción *beta*. Se abandona el ciclo una vez que *var* haya alcanzado al valor de la *expr.fin*. ejecutándose la acción *delta*. Por otro lado, estando en el punto de entrada si el valor de la *expr.ini*. fuera menor al valor de la *expr.fin*. directamente se busca el punto de salida y se ejecuta la acción *delta*. El tipo de *var*, la *expr.ini*. y la *expr.fin*. deben ser todas del mismo tipo ordinal.

Este tipo de ciclo es de uso limitado, ya que solo sirve para valores de tipo ordinal. De todas maneras en ciertos casos es más aconsejable su uso cuando se conozca de antemano, la cantidad de veces que deba ser ejecutado. Una de las ventajas es que no se debe llevar la cuenta del contador, ya que lo gestiona la propia estructura de control.

La cantidad de veces que se repetirán las acciones dentro del ciclo, queda establecida de la siguiente manera:

Cant.Rep.Desc. ← valor expr.ini. – valor expr.fin. + 1

Ejemplos repetición exacta

```
for (unsigned i = 0; i < 10; i++)
   cout << i << endl;
for (auto i = 1; i < 5 || b) {
   for (register j = 10; j > 0; j--)
      a += - (j * i);
   cout << "a: " << a << endl;
   if (a > 3)
      b = true;
}
```

Repetición post-condición

Repetición 1-X o Hasta

Luego de ejecutar la acción *alfa*, en el punto de entrada se ingresa a la estructura cíclica, ejecutándose la acción *beta*, luego en el pié de la estructura se evalúa la *condición*, si el resultado es **falso, permanece en el ciclo,** volviéndose a ejecutar la acción *beta*, **hasta** que la *condición* cambie de estado, es decir, cuando su resultado se haga **verdadero, abandona el ciclo** y en el punto de salida se ejecute la acción *delta*. La *condición* podrá ser lo más compleja que se desee. Este tipo de ciclo requiere un uso específico ya que la acción *beta* se podrá ejecutar al menos una vez, por eso a este ciclo también se lo denomina ciclo 1-x. Existen casos puntuales para su uso, a saber, **creación de menú de opciones, validación de datos, ordenamiento de burbuja optimizado**, y otros casos especiales.

Ejemplos de repeticiones 1-x

```
do {

}
while (b);
do
   a = 5;
while (a < 4);
c = 15;
do {
   c--;
   cout << c << endl;
}
while (c > 10);
```

Ejemplo

```
a = 0;
while (a < 3)
   cout << a << endl;
// jamás salimos del ciclo (infinito), por lo tanto, no es un algoritmo, puede decir ¿por qué?.
```

la situación anterior se soluciona como se indica a continuación:

Ejemplo

```
a = 0;
while (a < 3) {
   a = a + 1;
   cout << a << endl;
}
```

La **variable a** es inicializada con el valor cero, en la cabecera del ciclo la condición establece que si el contenido de la **variable a** es menor que tres y como es verdadero ingresamos al ciclo, observamos que hay dos sentencias, razón por la cual las encerramos dentro de un bloque de concatenación ({ }), dentro del mismo se incrementa en uno el contenido de la **variable a**, ahora su contenido es 1 (ya que 0 + 1 = 1), se emite su contenido, es decir, 1 y el endl indica que luego de emitir este valor hay un avance y retorno de línea. Nuevamente en la cabecera del ciclo la condición continúa siendo verdadera, ya que (1 < 3), se incrementa en uno la **variable a** y se emite 2 y así sucesivamente mientras **a < 3**, cuando el contenido de la **variable a** sea igual a 3, se emite este valor y al volver a la cabecera del ciclo, ahora la condición falla (ya que **3 < 3** es falso) y se abandona el ciclo.

(V.1)
```
a = 0;
while (a < 3) {
   a += 1;
   cout << a << endl;
}
```

(V.2)
```
a = 0;
while (a < 3) {
   a++;
   cout << a << endl;
}
```

(V.3)
```
a = 0;
while (a < 3) {
   ++a;
   cout << a << endl;
}
```

(V.4)
```
a = 0;
while (a++ < 3)
(    cout << a << endl;
```

(V.5)
```
for (a = 1; a < 3; a++)
   cout << a << endl;
```

Pr ueba de escritorio

	a	Emite
(V.1)	0	
	1	**1**
	2	**2**
	3	**3**
(V.2)	0	
	1	**1**
	2	**2**
	3	**3**
(V.3)	0	
	1	**1**
	2	**2**
	3	**3**
(V.4)	0	
	1	**1**
	2	**2**
	3	**3**
(V.5)	1	1
	2	2
	3	3

En los cinco ejemplos precedentes, observamos diferentes formas de codificación para hacer lo mismo, todas las versiones indicadas emiten valores [1; 3]. Incluso el ejemplo precedente a estas versiones también es equivalente. De lo indicado podemos decir que el lenguaje C/C++ ofrece distintas posibilidades de codificar nuestros algoritmos.

Repetición con pos-condición indefinida: Hasta, 1-X

```
α;
do
   β;
while (exp.lóg.)
δ;
```

Se debe utilizar el par de llaves {} para indicar un bloque en los casos en que β sea la sentencia vacía o un conjunto de sentencias mayor a uno.

Alfa y delta están fuera del alcance del ciclo, mientras que la sentencia beta se encuentra dentro de su alcance. Luego de ejecutarse la sentencia alfa, en el punto de entrada al ciclo se ingresa directamente, ejecutándose la sentencia beta. En el pie del ciclo se evalúa la condición, si resulta ser **verdadera** entonces **se continúa en el ciclo** volviéndose a ejecutar la sentencia beta. En algún instante del proceso cuando la **condición se torne falsa**, en ese caso **se abandona el ciclo**, ejecutándose la sentencia delta. Este tipo de ciclo ofrece la ventaja de que al menos una vez la sentencia beta será ejecutada y cuanto más una cantidad X de veces.

Este tipo de ciclo debe ser utilizado discretamente ya que la mayoría de las veces empleamos el ciclo while. No obstante, existen casos que hacen de su uso más aconsejable, por citar algunos de ellos mencionamos, cuando haya que validar datos, cuando creamos un menú de opciones, en el método de ordenamiento de arreglos, entre otros casos puntuales. Su alcance es de una sola sentencia, por lo que si necesitamos ejecutar más de una sentencia estamos obligados a encerrarlas dentro de un bloque de concatenación.

Ejemplo de repetición 1-x

```
do {
   cout "Ingrese un valor entero entre 1 y 9: ";
   cin >> valor;
}
while (!(valor >= 1 && valor <= 9)); // o su equivalente: (valor < 1 || valor > 9)
```

Ejemplo:

```
// Cálculos multiplicativos desde 1 * 1 hasta 1 * 9 y hasta llegar a 9 * 1 hasta 9 * 9.
for (i = 1; i < 10; i++) {
   for (j = 1; j < 10, j++)
      cout << i * j << " ";
   cout << endl;
}
```

Luego de haber conocido cada una de la estructuras de control de programa, es de esperar que el alumno utilice la mejor herramienta para llevar a cabo el algoritmo. No siempre la forma en que el alumno ve la situación será la mejor de ellas, un cierto grado de experiencia va a afianzar más el criterio a adoptar en cada situación que se presente.

Cuando determinamos que hay que utilizar una estructura repetitiva, luego nos planteamos cuál de ellas es la más adecuada, hay casos por ejemplo que aparentemente en un primer momento creemos que es un ciclo exacto, entonces no dudamos y utilizamos el ciclo exacto, pero luego vemos que en realidad no estamos obligados a alcanzar la cantidad total de repeticiones debido a que existe una segunda condición que nos hará abandonar el ciclo antes de haber alcanzo el final máximo, ahora nos damos cuenta que el ciclo apropiado es un ciclo indefinido con dos condiciones, mientras no hayamos encontrado un valor que estamos buscando y mientras no hayamos alcanzado la cantidad de repeticiones máximas, si el valor no aparece.

En toda estructura de control condicional y repetitiva existen distintos momentos, un momento previo al punto de entrada, un momento durante entre los puntos de entrada y salida y un momento después del punto de salida. Si la estructura es repetitiva indefinida el momento previo es el de preparar acciones previas, inicializar variables, conocer primer dato; el momento durante es el de procesar los datos y conocer próximo dato y el momento después es el de terminar, emitir, cálculos a totales mayores, promedios, porcentajes, grabar, etc.. En cambio en una estructura repetitiva exacta, en el momento previo inicializamos variables pero no conocemos primer dato, sino que en el momento durante, primero conocemos dato y luego procesamos, y en el momento después realizamos las acciones de terminar. Cualquier variable que aparezca en la condición de un ciclo deberá tener un valor previamente asignado en un momento anterior.

Lo mismo puede suceder con las alternativas simples y múltiples para determinar cuál será la estructura condicional más conveniente bajo ciertas circunstancias.

¿Serán ciclos en secuencia, o anidados?. ¿Serán condiciones en secuencia o anidadas?. ¿Serán ciclos dentro de decisiones o decisiones dentro de ciclos?. ¿Las selecciones tendrán acciones por cada una de las ramas o solo en una de ellas?. Estas son tan solo algunas de las preguntas que nos debemos formular al momento de resolver un problema computacional.

Como vemos la forma de combinar estas pocas **estructuras de control de programa** es lo que nos da una gran diversidad de variantes, el esqueleto que formará nuestro algoritmo. No obstante, la forma esquelética de nuestros algoritmos se determinará en base a otro tipo de estructuras, las **estructuras de datos**, este será entonces nuestro primer paso, a partir de allí, pensaremos como han de ser las estructuras de control de programa. Finalmente decimos que las **estructuras de control de programa esta determinada por las estructuras de datos.**

Ejemplos de sentencias simples y estructuradas de ejercicios resueltos de la guía 1

Codificación G1Ej05 Potencia de enteros

```
/*
        Id.Programa: G1Ej05.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dados 2 valores enteros >= 0, hallar la potencia de a ^ b.
            Ambos simultaneamente distinto de cero.
*/

#include <fstream>
using namespace std;

main() {
  int  a,
       b,
       i;
  long pot = 1;

  cout << "Hallar la potencia de a ^ b" << endl;
  do {
```

```
        cout << "Entero a: ";
        cin >> a;
        cout << "Entero b: ";
        cin >> b;
  }
 while (!(a + b > 0));
 for (i = 1; i <= b; i++)
    pot *= a;
 cout << a << " ^ " << b << " = " << pot;
 return 0;
}
```

Codificación G1Ej06 Factorial

```
/*
        Id.Programa: G1Ej06.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dado un valor entero n, hallar su factorial.
*/

#include <fstream>
using namespace std;

main() {
 int  n,
      i;
 long fact = 1;

 cout << "Hallar el factorial de n" << endl;
 cout << "Entero n: ";
 cin >> n;
 for (i = n; i >= 1; i--)
    fact *= i;
    cout << n << " != " << fact;
    return 0;
}
```

Codificación G1Ej07 Número e

```
/*
        Id.Programa: G1Ej07.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dado un valor n, generar el valor irracional e.
*/

#include <fstream>
using namespace std;
```

```
main() {
  int n,
       i,
       j;
  long fact;
  float e = 1;

  cout << "Hallar el numero irracional e" << endl;
  cout << "Entero n: ";
  cin >> n;
  for (i = 1; i <= n; i++) {
      fact = 1;
      for (j = i; j >= 1; j--)
          fact *= j;
      e += 1.0 / fact;
    }
    cout << "e~= " << e;
    return 0;
}
```

Codificación G1Ej07 Número e Optimización

```
/*
        Id.Programa: G1Ej07Ef.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dado un valor n, generar el valor irracional e.
*/

#include <fstream>
using namespace std;

main() {
        int n,
            i;
        long fact = 1;
        float e = 1;

        cout << "Hallar el numero irracional e" << endl;
        cout << "Entero n: ";
        cin >> n;
        for (i = 1; i <= n; i++) {
                fact *= i;
                e += 1.0 / fact;
        }
        cout << "e ~=  " << e;
        return 0;
}
```

Codificación G1Ej08 Sumatoria primeros 100 naturales

```
/*
        Id.Programa: G1Ej08.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dado un valor entero n > 0, generar la sumatoria
                desde 1 hasta n.
*/

#include <fstream>
using namespace std;

void main() {
        int  n,
             i;
        long sum = 0;

        cout << "Hallar la sumatoria desde 1 hasta n" << endl;
        do {
                cout << "Entero n: ";
                cin >> n;
        }
        while (!(n > 0));
        for (i = 1; i <= n; i++)
                sum += i;
        cout << "Sumatoria desde 1 hasta " << n << " = " << sum;
}
```

Codificación G1Ej09 Sumatoria primeros 100 Naturales Optimizada

```
/*
        Id.Programa: G1Ej09.cpp
        Autor……...: Lic. Hugo Cuello
        Fecha……....: julio-2013
        Comentario.: Dado un valor entero n > 0, generar la sumatoria
                                                        desde 1 hasta n.
*/
#include <fstream>
using namespace std;

void main() {
        int n;
        long sum;

        cout << "Hallar la sumatoria desde 1 hasta n" << endl;
        do {
                cout << "Entero n: ";
                cin >> n;
```

```
        }
        while (!(n > 0));
        sum = n * (n + 1) / 2;
        cout << "Sumatoria desde 1 hasta " << n << " = " << sum;
}
```

Codificación G1Ej10 Sucesión de Fibonacci

```
/*
        Id.Programa: G1Ej10.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Dado un valor entero n, hallar la sucesion de Fibonacci.
*/
#include <fstream>
using namespace std;

main() {
  int  i,
       n;
  long anteult,
       ult,
       sum;

        cout << "Hallar la sucesion de Fibonacci desde 0 hasta n" << endl;
        cout << "Entero n: ";
        cin >> n;
        switch (n) {
          case 0:
                cout << 0 << endl;
                break;
          case 1:
                cout << 0 << ", " << 1;
                break;
          default:
                anteult = 0;
                ult = 1;
                cout << anteult << ", " << ult << ", ";
                for (i = 2; i <= n; i++) {
                   sum = ult + anteult;
                   anteult = ult;
                   ult = sum;
                   cout << sum << ", ";
                }
        }
        cout << '\b' << '\b' << " ";
        return 0;
}
```

Codificación G1Ej15 Primeros 3 números Perfectos

```
/*
        Id.Programa: G1Ej15.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Generar los 3 primeros numeros perfectos.
*/

#include <fstream>
using namespace std;

main() {
  int i,
      sum,
      num = 0,
      k = 0;

  cout << "Generar los 3 primeros numeros perfectos" << endl;
  while (k < 3) {
     num += 2;
     sum = 0;
     for (i = 1; i <= num / 2; i++)
         if (num % i == 0)
            sum += i;
         if (num == sum) {
            k++;
            cout << k << ": " << num << endl;
         }
    }
    return 0;
}
```

Página en blanco dejada a propósito.

Unidad I Capítulo V

Unidad I – Capítulo V - Modularización:

Funciones

Una función es un módulo que presenta las siguientes partes:

- **Una cabecera también llamada interfaz**
- **Un cuerpo**

La cabecera presenta las siguientes partes:

- **El tipo de valor que retornará la función.**
- **El nombre de la función**
- **Una lista de parámetros, que podría estar vacía**

También una función puede NO retornar valor, en este caso el tipo de la función será **void**.

El cuerpo de la función es un bloque, dentro del cual se escriben las sentencias. En caso que la función retorne un valor deberá existir alguna sentencia **return** el cual retorne el resultado de una expresión que debe ser de igual tipo que el valor retornado por la función. Al finalizar la función, el último valor asignado, será el valor que retorne. Una función la podemos ver como una caja negra –no nos interesa saber su interior- o como una caja transparente –si nos interesa saber su interior-. Para que una función pueda ser ejecutada, se deberá **llamarla o invocarla**. Los **parámetros formales** –aquellos que se definen en la cabecera de la función, son pasados por **valor**; en estos casos, se trabaja con una copia del valor pasado desde la invocación, cualquier modificación que se realice en los parámetros formales, no altera a los **parámetros actuales** –los indicados en el momento de la invocación- o por **referencia** en este último caso se antepone al nombre del parámetro el símbolo ampersand & y cualquier cambio en este parámetro formal repercute en su argumento correspondiente de la llamada, ya que se trabaja en la misma posición de memoria de su argumento.

Ventajas y / o desventajas

Simplificar el código, dividir las tareas, reusabilidad (ventajas), pero a costa de un mayor tiempo de ejecución por el uso de la Pila del Sistema (desventaja).

Una función se la puede invocar en **forma indirecta**, es decir, como perteneciente a una sentencia, como p.e., la asignación interna, o como formando parte de una condición, etc..o puede ser invocada en **forma directa**, en el caso que la función retorne un valor (el tipo de la función no es void), ese valor se perderá.

Si el tipo de valor de la función se la definió como **void**, en estos casos, ningún valor retornará la función y dentro de su cuerpo **no se utilizará** la palabra **return**.

Una función puede necesitar trabajar con variables internas a su ámbito, en estos casos esas variables definidas dentro de la función son denominadas **variables locales**, también conocida como **variables auto, por automáticas**, ya que el sistema debe crear estas variables cuando se invoca a la función y también las debe eliminar al salir. Fuera de la función en que han sido definidas no existen, salvo en los casos en que la función invoque a un módulo definido dentro de su ámbito, allí serán reconocidas. La comunicación entre el módulo que invocó a la función y la función invocada se establece por medio de los parámetros.

Estructura de una función: -definición-

lst_par_formales presenta por cada parámetro formal dos componentes y cada componente se separan con comas. La primer componente indica el tipo del parámetro y la segunda componente indica el nombre del parámetro. Además se debe indicar por cada parámetro el pasaje de parámetro, los cuales pueden ser: por valor/contenido o por referencia/dirección.

Llamada o invocación a una función

Ejemplo

```
var ← NomFunc(lst_argumentos)  // forma indirecta.
NomFunc(lst_argumentos)        // forma directa.
```

<u>Observación</u>: Los parámetros en la invocación también suelen llamarse **parámetros actuales**. En estos casos los parámetros en la cabecera de los módulos se los puede indicar como **parámetros formales**.

Las **funciones** son módulos que deben ser invocados para que su código pueda ser ejecutado, ya que, si no se invocan están como en un estado de **letargo** o **hibernación**. La comunicación entre la llamada y el módulo se logra por medio de los parámetros. En el punto de la llamada se denominan **parámetros actuales o argumentos**, mientras que en la cabecera del módulo se denominan **parámetros formales**. Una función está estructurada en las siguientes partes:

- Una cabecera que incluye:
  - El tipo de dato que retornará la función.
  - El identificador o nombre de la función.
  - Una lista de parámetros, que podrán ser pasados por **valor**, o por **referencia**; en este último caso deberá estar precedido con un asterisco que indica puntero o en la versión C++ por el símbolo ampersand **&**.
- Un área de definiciones y declaraciones de objetos locales.
- Un cuerpo encerradas entre los símbolos de concatenación { }, el cual deberá tener al menos una sentencia que retorne un valor del tipo indicado en su cabecera o interfaz, por medio de la palabra reservada return cuyo valor indicado a continuación es del mismo tipo que el indicado en su cabecera. Cuando se ejecute esta sentencia hará que la función que se está ejecutando finalice y retorne al punto de la llamada o invocación.

```
tipo IdFunc( tipo2 lista de parámetros formales) {
   var's –Locales–

   sentencia; ...
   return expresión;
}
```

en donde: la lista de parámetros formales pueden ser:

```
tipo Identificador
tipo *Identificador
tipo &Identificador   // C++
```

Pasaje de parámetros

En el primer caso, decimos que el parámetro es pasado por **valor**, esto es, con el valor pasado se hace una copia en otra región de memoria de la RAM llamada Stack o Pila y cualquier cambio que se hiciere, en realidad no afectará al parámetro actual, por estar trabajando con la copia, y no con el valor original. Este tipo de pasaje de parámetros es utilizado cuando el valor se conoce antes de invocar al módulo y deseamos pasar el valor al módulo para que realice algún proceso. Los parámetros pasados por valor son parámetros de entrada y sirve para inicializar al parámetro formal. El objeto que ocupa la misma posición, el parámetro actual, puede ser una expresión, es decir, puede ser una **constante**, una **variable** o una **expresión propiamente dicha** o una **función**. No obstante, hay situaciones en que el argumento pasado es un arreglo, en estos casos el pasaje de parámetros es pasado automáticamente por referencia, por lo que, el parámetro en la cabecera del módulo no se precederá con el asterisco '*' o en el caso de C++ con el ampersand '&'.

Si el parámetro está precedido por un asterisco o el ampersand &, entonces el parámetro es pasado por **referencia**, es el método nativo de C, esto es, se pasa la dirección del objeto, un puntero recibe la dirección del parámetro actual pasado correspondiente al parámetro formal, por lo tanto, este objeto, del parámetro actual, **solo debe ser una variable**. El parámetro formal recibe la dirección, y al momento en que se le asigne un valor dentro del módulo al parámetro formal, en realidad se lo está asignando al parámetro actual, a través del parámetro formal. Imaginemos que este parámetro formal es una catapulta que envía lo que recibe a un único punto, al punto al que apunta, es decir en donde está ubicada la variable –el parámetro actual-, cuya dirección es la que recibió el parámetro formal. Este tipo de pasaje de parámetros es utilizado cuando conocemos el valor dentro del módulo y queremos dárselo al bloque que lo llamó a través de su parámetro actual o bien conociendo previamente su valor, dentro del módulo invocado quizás sufra alguna modificación y queremos informarlo al bloque que invocó al módulo, por medio de su parámetro actual. Este tipo de parámetros pasados por referencia son **parámetros de salida** o bien de **entrada-salida**. Dentro del módulo la notación empleada para asignar un valor a estos tipo de parámetros pasados por referencia, se debe indicar la des referencia por medio del símbolo asterisco que precede al parámetro por referencia que es un puntero, como se indica a continuación:

Si pi está definido como parámetro formal de la siguiente forma, **int *pi**; entonces asignar un valor a la dirección apuntada por pi se lo indicará de la siguiente manera:

```
*pi = *pi + 1;
```

Por otro lado en C++ decimos que si el parámetro formal está precedido por el símbolo &, entonces el parámetro formal es pasado por referencia, esto es, se pasa la dirección del objeto, pero a diferencia del caso anterior, esta forma solo está indicado para C++ y su denotación se asemeja más al lenguaje Pascal logrando una mayor claridad en su manejo. De todas formas, se pasa la dirección del objeto o parámetro actual, por lo tanto, este objeto, **solo debe ser una variable**. El parámetro formal recibe la dirección, y al momento en que se le asigne un valor dentro del módulo al parámetro formal, en realidad se lo está asignando al parámetro actual, a través del parámetro

formal. Imaginemos que este parámetro formal es una catapulta que envía lo que recibe a un único punto, al punto al que apunta, es decir en donde está ubicada la variable –el parámetro actual-, cuya dirección es la que recibió el parámetro formal. Este tipo de pasaje de parámetros es utilizado cuando conocemos el valor dentro del módulo y queremos dárselo al bloque que lo llamó a través de su parámetro actual o bien conociendo previamente su valor, dentro del módulo invocado quizás sufra alguna modificación y queremos informarlo al bloque que invocó al módulo, por medio de su parámetro actual. Este tipo de parámetros pasados por referencia son parámetros de salida o bien de entrada-salida. Dentro del módulo la notación empleada para asignar un valor a estos tipo de parámetros pasados por referencia, solo se debe indicar el parámetro formal, vale decir sin indicar nada que lo preceda como fue en el caso anterior. El siguiente ejemplo ilustra este último caso de pasaje de parámetros por referencia al estilo de C++:

Si pRef está definido como parámetro formal de la siguiente forma, **int &pRef**; entonces asignar un valor a la dirección apuntada por pRef se lo indicará de la siguiente manera:

pRef = pRef + 1;

Otra forma de ilustrar el pasaje de parámetros por referencia o dirección es el ejemplo del jefe que le dice a su **cadete**, ve a tal lugar y regrésame la información que te van a brindar. Si el cadete luego de ir al lugar de destino regresa a la oficina y se lo informa al jefe de la información, es el caso de pasaje por referencia o dirección. Si en cambio el cadete no regresa a la oficina o no comenta la información recibida en el lugar de destino a su jefe, estamos en el caso de un pasaje por valor o contenido.

Siguiendo con los ejemplos para asimilar en concepto de pasaje de parámetros por valor o contenido versus por referencia o dirección, tomemos la actividad realizada por un **cartero**. A primera hora del día el cartero se dirige a la estafeta postal a retirar su mochila cargada de cartas, luego se dirige a la zona de reparto a efectos de repartir las cartas a los domicilios indicados en el sobre. Mencionado de otra manera, cuando el cartero recibe las cartas no se las queda para él, sino que va a repartirlas, pero no a cualquier lado, sino donde indica la dirección en el sobre, al entregar dicho sobre le está entregando la información brindada al destinatario y precisamente ese destinatario es el parámetro actual o argumento, el cual decimos que debe ser si o si una variable, ya que las variables ocupan posiciones de memoria fijas.

El **pasaje de parámetros por valor o contenido**, significa que lo que se está pasando es el valor del objeto enviado que puede ser un valor constante, el contenido de una variable o el resultado de una expresión; el valor recibido por el parámetro formal realiza una copia de ese valor en un área de almacenamiento temporal el **Stack** o Pila, eliminándose en el momento de abandonar el módulo; cualquier cambio que se produzca sobre ese objeto, el cambio se lo está realizando en la copia, por lo tanto el parámetro actual jamás se verá afectado de esas modificaciones.

En cambio el **pasaje de parámetros por referencia o dirección**, también llamado **parámetro variable**, significa que lo que se está pasando es la **dirección** del objeto enviado, que **sólo podrá ser una variable**. El parámetro formal correspondiente recibe la dirección de ese objeto y, cualquier valor que se le asigne, en realidad se lo está asignando al parámetro actual correspondiente, por lo tanto, no se trabaja con una copia sino con el propio objeto. La forma de establecer si un parámetro formal debe ser por referencia es indicando antes de la lista de parámetros el símbolo **&**. Todos aquellos parámetros que deban ser pasados por valor, entonces no se debe anteponer ese símbolo denominado ampersand.

¿Cuándo pasar parámetros por valor o por referencia?

Usamos parámetros por valor cuando conocemos previamente el valor de esos parámetros y deseamos pasarlo al módulo para que realice algún tratamiento con esos datos; aún si se cambiara su valor dentro del módulo invocado, no afecta al parámetro actual. En cambio usamos parámetros por referencia cuando el dato se va a conocer en el módulo o sabiendo su valor debemos modificarlo y se requiere saberlo en el ámbito en que fue invocado Los parámetros actuales cuando son pasados por valor pueden ser valores constantes, variables, expresiones o funciones. En cambio los parámetros actuales cuando son pasados por referencia solo pueden ser objetos variables.

Ejemplo de funciones

```
// versión C.

#include <fstream>
#include <math.h>
#include <conio.h>

void ObtenerCatetos(float *, float *);
void CalcularHipotenusa(float, float, float *);
void Emitir(float);
void main () {

   float a,
         b,
         c;

   ObtenerCatetos(&a,&b);
   CalcularHipotenusa(a,b,&c);
```

```
    Emitir(c);
}

void Obtener Catetos(float *x, float *y) {
    cout << "Cateto a: ";
    cin >> *x;
    cout << "Cateto b: ";
    cin >> *y;
}

void CalcularHipotenusa(float x, float y, float *z) {
    *z = sqrt(pow(x, 2) + pow(y, 2))
}

void Emitir(float z) {
    cout << "Hipotenusa c: " << z << endl;
    getch();
}
```

Alternativa de solución en el módulo CalcularHipotenusa, en la invocación a esta función lo modificamos por la siguiente forma:

```
    c = CalcularHipotenusa(a, b);
```

también debemos modificar el prototipo de esta función por la nueva interfaz:

```
float CalcularHipotenusa(float , float);
```

y el desarrollo de la función por la siguiente forma:

```
float CalcularHipotenusa(float x, float y) {
  return sqrt(pow(x, 2) + pow(y, 2));
}

// versión C++.

#include <fstream>
#include <math.h>
#include <conio.h>

void ObtenerCatetos(float &, float &);
void CalcularHipotenusa(float, float, float &);
void Emitir(float);

void main () {
    float a,
          b,
          c;

    ObtenerCatetos(a,b);
```

```
    CalcularHipotenusa(a,b,c);
    Emitir(c);
}

void Obtener Catetos(float &x, float &y) {
    cout << "Cateto a: ";
    cin >> x;
    cout << "Cateto b: ";
    cin >> y;
}

void CalcularHipotenusa(float x, float y, float &z) {
    z = sqrt(pow(x, 2) + pow(y, 2))
}

void Emitir(float z) {
    cout << "Hipotenusa c: " << z << endl;
    getch();
}
```

Funciones y/o Macros Numéricas de la librería math.h

El lenguaje C/C++ brinda de un conjunto de funciones o macros matemáticas, definidas en la librería **<math.h>** en el cuál, el programador solo necesitará conocer su interface y su propósito, actúa como el concepto de **Caja Negra**, solo debemos conocer sus entradas y salidas y nada de su parte interior.

A continuación se presenta una lista de estas funciones o macros:

abs(x) : Retorna el valor absoluto de x, el tipo de la entrada x es int, el tipo de valor que retorna también es int. El abs(x) es x, si x >= 0, sino, es –x. Está definida como macro.
fabs(x) : Idem anterior pero el tipo de la entrada x es double y, el tipo de valor que retorna también es double. Está definida como función.
acos(x) : Función inversa del coseno, o dicho de otro modo, es el arco coseno(x). Tanto el argumento como el valor retornado por la función son de tipo double.
acosl(x) : Idem anterior pero para tipo long double, tanto para el argumento como para el valor retornado por la función. -1 <= x <= 1 R[0, π]
asin(x) : Función inversa del seno o arco seno(x).
asinl(x) : Idem anterior pero de tipo long double.
atan(x) : Función inversa de la tg o arco tg(x). atan(1) * 180 / PI = 45
atanl(x) : Idem anterior pero de tipo long double.
cos(x) : Función coseno(x), x en radianes, para convertir grados a radianes x * PI / 180.
sin(x) : Función seno(x), x en radianes, para convertir grados a radianes x * PI / 180.
tan(x) : Función tangente(x), para convertir grados a radianes x * PI / 180.
exp(x) : Función exponencial o anti logaritmo(x) e^x. Ambos de tipo double, base e. Para convertir a otra base se aplica la siguiente expresión: exp(x * log(n)), en donde, n indica la base a pasar. exp(1) = e ~= 2.718282…exp(3 * log(10)) = 1000.
expl(x) : Idem anterior pero de tipo long double.
*Ver página 28 por temas relacionados con funciones.

Ver página 28 por tema del resto.

log(x) : Función logaritmo(x) de base e o natural, x > 0. Ambos de tipo double. Para convertir a otra base se aplica la siguiente expresión: log(x) / log(n), en donde, n indica la base a pasar. log(100) = 4.60157…log(100) / log(10) = 2.
logl(x) : Idem anterior pero de tipo long double.
log10(x) : Función logaritmo(x) de base 10. Ambos de tipo double.
log10l(x) : Idem anterior pero de tipo long double.
div(x,y) : Divide x por y y retorna 2 valores, el cociente entero y el resto, esto es posible por que el tipo de valor retornado es de tipo **div_t** definido por el lenguaje, siendo un registro –**struct**- con 2 campos de nombre **quot** (cociente) y **rem** (resto).
Ejemplo: **div_t** x; x = **div**(7,3); cout >> x.**quot** << ' ' << x.**rem**: // Emite: 2 y 1.
fmod(x,y) : Función que retorna el módulo o resto entre x e y. Todos de tipo double.
fmodl(x,y) : Idem anterior pero de tipo long double.
modf(x,y) : Particiona el valor de x de tipo double a sus partes entera en y el cual debe ser pasado por referencia y la parte decimal que el de tipo double retornado por la función. double x = 37.1854; // 37.0 0.1854
pd = modf(x, &y); cout << "Parte entera: " << y << "Parte decimal: " << pd;
double frac, entero, numero = 1234.567;
frac = modf(numero, &entero);
cout << numero << " " << entero << " " << frac << endl;
pow(x,y) : Función potencia en donde x es la base e y es el exponente, x^y. De tipos double. Si x = 0 entonces y > 0.
pow10(x) : Función potencia en donde x es el exponente y 10 es la base. Tipos double, 10^x.
sqrt(x) : Función que extrae la raíz cuadrada positiva de x, x>=0. Ambos double.
ceil(x) : Función de redondeo hacia arriba, siendo x de tipo double. ceil(3.2) = 4.0; ceil(3.7) = 4.0; ceil(-3.2) = -3.0; ceil(-3.7) = -3.0
floor(x) : Función de redondeo hacia abajo, siendo x de tipo double. floor(3.2) = 3.0; floor(3.7) = 3.0; floor(-3.2) = -4.0; floor(-3.7) = -4.0
random(x) : genera valores enteros pseudoaleatorios [0, x) o [0, x – 1] Ejemplo: arrojar un dado el cual genera valores entre [1, 6]. random(6) + 1. Generar valores en el [a, b] con b >= a. random(b – a + 1) + a. Si a = 12 y b = 35 random(b – a + 1) + a = random(24) + 12. Arrojar el dado 10 veces: Run1 3 2 6 1 1 4 3 5 6 2, Run2 idem anterior,…. Run n veces idem a los anteriores.
rand() : Función que retorna un valor entero [0, RAND_MAX] RAND_MAX = 32767
randomize() : Asigna una semilla al generador de números aleatorios en ramdom.
srand(x) : Asigna una semilla al generador de números aleatorios de rand(). x es un unsigned. x = time(NULL); srand(x); // **srand(time(NULL)); rand() % n;**
rand() % (MAX – MIN + 1) + MIN. Incluir librerías **stdlib.h o cstdlib y ctime.h.**
// Se debe incluir las librerías **#include<ctime> #include <cstdlib>**
hypot(x, y) : Función que retorna la hipotenusa de un triángulo rectángulo.
h = hypot(4,3); asigna 5 a la variable h. Es equiv. sqrt(pow(4)+pow(3)) = 5.0
poly(x,n,c) : Función que retorna el resultado de operar con el polinomio de grado n, de una variable x y de coeficientes c. El tercer parámetro c es un arreglo de valores double que se corresponde al exponente de x en el $término_i$ con i >= 0. x es la variable representado en el primer parámetro de la función. Ejemplo: Px = **poly**(2,3,vecCoef); vecCoef = {3,1,5,2}. Px = 41. (Px = 2x2^3 + 5x2^2 + 1x2 + 3 = 16 + 20 + 2 + 3 = 41).

Funciones de la librería <stdlib.h>
Conversión de cadena a número

int atoi(const char* cad)

convierte una cadena de caracteres a valor de tipo entero. Descarta espacios en blanco al inicio. Considera los signos +, -. Otros caracteres corta retornando cero. Cualquier carácter extraño que no sea dígito corta inmediatamente, dejando los caracteres previos hasta ese punto, convirtiéndolos a número.

Ejemplos de conversión de cadena a número

```
char cad = "12a34";
int num;
num = atoi("3142") + 5; // asigna a num el valor 3147.
num = atoi(cad) * 2; // asigna a num el valor 24.
num = atoi("    -34871"); // asigna a num el valor -34871.
num = atoi("  *+1234") * 4; // asigna a num el valor cero.
```

long atol(char* cad)

convierte una cadena a entero largo. Los mecanismos de control de los caracteres en cad se analizan de la misma manera que atoi.

```
long numL;

numL = atol("21546374"); // asigna a numL el valor 21546374.
numL = atol("  -3245671") + 2; // asigna a numL el valor -3245673.
numL = atol("  uno134523"); // asigna a numL el valor cero.
```

double atof(char* cad**)**

convierte la cadena cad a un valor de tipo double. Acepta signo +, -, digitos de 0-9 punto decimal, las letras e, E, con el formato sgn dígitos punto dígitos e o E sgn dígitos.

Ejemplos

```
double numD;

numD = atof("   1.76387e+2); // asigna a numD el valor 176.387.
numD = atof("   1.76*387e+4); // asigna a numD el valor 1.76.
numD = atof("+1.76387e+2"); // asigna a numD el valor 176.387.
```

Conversión de entero a cadena

char* itoa(int num**, char*** cad**, int** base**)**

char* itoa(long num**, char*** cad**, int** base**)**

convierte el entero num a un valor de tipo cadena, en cad, el tercer argumento corresponde al sistema de numeración a ser convertido y es un valor entre 2 y 36. Si base es 2, entonces es en binario, si base es 10 la conversión es en decimal, si base es 16, la base a ser convertido es en hexadecimal. Si la base es 10 y el num es negativo, entonces se incorpora el signo menos. Para otras bases es sin signo.

```
char cad[31];
int num;
```

```
num = 2167;
cout << itoa(num,cad,10); // emite la cadena de caracteres en decimal, 2167.
printf("dec. %s \n",cad); // ídem anterior.
num = 67;
cout << itoa(num,cad,2); // emite la cadena de caracteres en binario, 1000011.
printf("binario; %s \n",cad); // ídem anterior.
num = 139;
cout << itoa(num,cad,16); // emite la cadena de caracteres en hexadecimal, 8b.
printf("hexadec. %s: \n",cad); // ídem anterior.
```

Conversión de real –float, double- a cadena

```
double numDb = 123412341234.123456789;

 char output[50];
 snprintf(output, 50, "%f", numDb);
 strcat(output," Hola");
 printf("%s", output); // emite 123412341234.123456789 Hola
```

Aplicación: Dados dos valores enteros a y b, encontrar la fracción irreducible de a / b.

Funciones varias

system(cad);

Ejemplo

system("pause") // Produce una pausa hasta que se oprima una tecla.
system("dir"); // obtiene el directorio actual.

Funciones definidas por el usuario

Ejemplos de funciones y diseño modular

Codificación G1Ej17 Función iif(expLog,x,y)

```
/*
   Id.Programa: G1Ej17.cpp
   Autor……...: Lic. Hugo Cuello
   Fecha.........: julio-2013
   Comentario.: Crear una funcion que recibe un valor logico y 2 enteros y
                retorne el primer entero si el valor logico es verdadero,
                caso contrario, retorne el segundo entero.
*/

#include <fstream>
using namespace std;

int iif(bool expLog, int ent1, int ent2) {
  if (expLog)
    return ent1;
  else
    return ent2;
} // iif

main() {
  int a,
      b;

  cout << "Funcion que retorna un entero u otro segun el resultado " <<
          "de una expLog" << endl;
  cout << "Entero a: ";
  cin >> a;
  cout << "Entero b: ";
  cin >> b;
  cout << iif(a > b,a,b);
  return 0;
}
```

Codificación G1Ej18 Algoritmo de Euclides Función mcd(x,y)

```
/*
   Id.Programa: G1Ej18.cpp
   Autor……...: Lic. Hugo Cuello
   Fecha.........: julio-2013
   Comentario.: Crear una funcion que recibe 2 enteros y retorne el mcd.
*/

#include <fstream>
using namespace std;

int mcd(int x, int y) {
   int res;

   while(y) {
      res = x % y;
      x = y;
      y = res;
   }
   return x;
} // mcd

main() {
   int a,
       b,
      maxComDiv;

   cout << "Funcion que retorna el m.c.d. entre 2 valores enteros" << endl;
   cout << "Entero a: ";
   cin >> a;
   cout << "Entero b: ";
   cin >> b;
   maxComDiv = mcd(a,b);
   cout << "m.c.d.: " << maxComDiv << endl;
   cout << "Aplicacion del m.c.d.: " << endl;
   cout << a << "/" << b << " = " << a / maxComDiv << "/" << b / maxComDiv;
   cout << " que es la fraccion irreducible al dividir " << endl;
   cout << "numerador y denominador por el m.c.d. = " << maxComDiv;
   return 0;
}
```

Codificación G1Ej19 Función signo(x)

```
/*
   Id.Programa: G1Ej19.cpp
   Autor.........: Lic. Hugo Cuello
   Fecha.........: julio-2013
  Comentario.: Crear una funcion que recibe un valor entero x, y retorne
                el signo de x. A diferencia de la funcion matematica,
                se acepta el cero y en este caso se debe retornar cero.
```

```
*/

#include <fstream>
using namespace std;

short int sgn(int x) {

   if(x == 0)
      return 0;
   else
      return x / abs(x);
} // sgn

void main() {
   float a;

   cout << "Funcion que retorna el signo de x" << endl;
   cout << "Real a: ";
   cin >> a;
   cout << "sgn x: " << sgn(a);
}
```

Codificación G1Ej20

```
/*
   Id.Programa: G1Ej20.cpp
   Autor……...: Lic. Hugo Cuello
   Fecha.........: julio-2013
   Comentario.: Crear una funcion que recibe dos valores enteros y retorne
                el resto de la division entera.
*/

#include <fstream>
using namespace std;

int resto(int x, int y) {
   return x - x / y * y;
} // resto

void main() {
   int a,
       b;

   cout << "Funcion que retorna el resto de una division entre enteros" << endl;
   cout << "Entero a: ";
   cin >> a;
   cout << "Entero b: ";
   cin >> b;
   cout << "resto(" << a << ", " << b << ") = " << resto(a,b);
}
```

Codificación G1Ej21 Función EsDivis(x,y)

```
/*
   Id.Programa: G1Ej21.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: julio-2013
   Comentario.: Crear una funcion que recibe dos valores enteros y retorne
                un valor booleano de verdadero si el primero es divisible
                por el segundo, caso contrario retorne falso.
*/

#include <fstream>
using namespace std;

bool EsDivis(int x, int y) {

   return bool(x % y == 0);
} // EsDivis

main() {
   int a,
       b;

   cout << "Funcion que retorna verdadero si el primer entero es divisible ";
   cout << "por el segundo o falso si no lo es" << endl;
   cout << "Entero a: ";
   cin >> a;
   cout << "Entero b: ";
   cin >> b;
   cout << boolalpha;
   cout << "EsDivis(" << a << ", " << b << ") = " << EsDivis(a,b);
   return 0;
}
```

Codificación G1Ej22 Examen Matemática Triángulos

```
/*
   Id.Programa: G1Ej22.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: julio-2013
   Comentario.: Examen de Matematica, tema Triangulos, para responder por
                cada alumno su perimetro y tipo de triangulo
                (Isos,Escal,eQuil), conociendo las medidas de sus lados.
*/

#include <fstream>
using namespace std;

#define CENTINELA "*"
const MIN = 23,
      MAX = 72;
```

```
typedef unsigned uint;
typedef char str20[21];

void EmiteInic(uint &tRespC, uint &tRespI) {
   cout << "Examen de Matematica, tema Triangulos perimetro y tipo de Triang.";
   tRespC = tRespI = 0;
   randomize();
} // EmiteInic

void ObtNomAlu(str20 nAlu) {
   cout << "Nombre Fin = *: ";
   cin >> nAlu;
} // ObtNomAlu

uint iif(bool expLog, uint ent1, uint ent2) {
   if (expLog)
      return ent1;
   else
      return ent2;
} // iif

void ObtLados(uint &x, uint &y, uint &z) {
   uint mini,
        maxi;

   x = random(MAX - MIN + 1) + MIN;
   y = random(MAX - MIN + 1) + MIN;
   mini = iif(abs(x - y) <= MIN,MIN,abs(x - y) + 1);
   maxi = iif(x + y >= MAX,MAX,x + y - 1);
   z = random(maxi - mini + 1) + mini;
} // ObtLados

void RespAlu(uint x, uint y, uint z, uint &pAlu, char &tAlu) {

   cout << "Dados los lados de un triangulo con medidas: " << x;
   cout << ", " << y << ", " << z;
   cout << " responder su perimetro y ";
   cout << "tipo de triangulo (Isos,Escal,eQuil)" << endl;
   cout << "Perimetro: ";
   cin >> pAlu;
   do {
      cout << "Tipo de triangulo (I,E,Q): ";
      cin >> tAlu;
   }
   while(!(strcmp(&tAlu,"I") || strcmp(&tAlu,"E") || strcmp(&tAlu,"Q")));
} // RespAlu

void RespMaq(uint x, uint y, uint z, uint &pMaq, char &tMaq) {
   pMaq = x + y + z;
```

```
   if(x == y && y == z)
      tMaq = 'Q';
   else
      if(x == y || x == z || y == z)
         tMaq = 'I';
      else
         tMaq = 'E';
} // RespMaq

void CalcVerifEmite(str20 nAlu, uint pAlu, uint pMaq,
                    char tAlu, char tMaq, uint &tRespC, uint &tRespI) {

   char Cartel[11];

   if(pAlu == pMaq && tAlu == tMaq) {
      tRespC++;
      strcpy(Cartel,"O.K.");
   }
   else {
      tRespI++;
      strcpy(Cartel,"Estudie+++");
   }
   cout << nAlu << " " << Cartel << endl;
} // CalcVerifEmite

void EmiteTot(int tRespC, int tRespI) {
   cout << "Cantidad de respuestas correctas..: " << tRespC << endl;
   cout << "Cantidad de respuestas incorrectas: " << tRespI;
} // EmiteTot

int main() {
   uint   totRespCor,
          totRespIncor,
          a,
          b,
          c,
          perAlu,
          perMaq;
   str20 NomAlu;
   char  triAlu,
          triMaq;

   EmiteInic(totRespCor,totRespIncor);
   ObtNomAlu(NomAlu);
   while(strcmp(NomAlu,CENTINELA)) {
      ObtLados(a,b,c);0
      RespAlu(a,b,c,perAlu,triAlu);
      RespMaq(a,b,c,perMaq,triMaq);
      CalcVerifEmite(NomAlu,perAlu,perMaq,triAlu,triMaq,totRespCor,totRespIncor);
      ObtNomAlu(NomAlu);
```

```
    }
    EmiteTot(totRespCor,totRespIncor);
    return 0;
}
```

Codificación G1Ej23 Ternas de valores

```
/*
    Id.Programa: G1Ej23.cpp
    Autor...........: Lic. Hugo Cuello
    Fecha...........: julio-2013
    Comentario.: Dado un valor n y a continuacion n ternas de valores enteros,
                 emitir c/u. de las ternas ordenadas descendente y por fin
                 de proceso el valor minimo y maximo de todas las ternas.
*/

#include <values>
#include <fstream>
using namespace std;
typedef unsigned uint;

void EmiteInic(uint &vMin, uint &vMax) {

    cout << "Ternas de valores enteros ordenar descendente" << endl;
    cout << "Val.Min. y Val.Max" << endl;
    vMin = MAXINT;
    vMax = -MAXINT - 1;
} // EmiteInic

uint ObtDato() {
    uint cTer;

    cout << "Cantidad de ternas: ";
    cin >> cTer;
    return cTer;
} // ObtDato

void ObtTerna(uint &x, uint &y, uint &z) {

    cout << "Valores enteros de una terna" << endl;
    cout << "Entero 1: ";
    cin >> x;
    cout << "Entero 2: ";
    cin >> y;
    cout << "Entero 3: ";
    cin >> z;
} // ObtTerna

void IntCmb(uint &elem1, uint &elem2) {
                                uint aux = elem1;
```

```
   elem1 = elem2;
   elem2 = aux;
} // IntCmb

void OrdTerna(uint &x, uint &y, uint &z) {

   if(y > x)
      IntCmb(x,y);
   if(z > x)
      IntCmb(x,z);
   if(z > y)
      IntCmb(y,z);
} // OrdTerna

void VerifMinMax(uint a, uint c, uint &vMin, uint &vMax) {

   if(a > vMax)
      vMax = a;
   if(c < vMin)
      vMin = c;
} // VerifMinMax

void EmiteTerna(uint x, uint y, uint z) {

cout << "Terna(" << x << "," << y << "," << z << ")" << endl;
} // EmiteTerna

void EmiteMinMax(uint vMin, uint vMax) {

   cout << "Valor minimo de ternas: " << vMin << endl;
   cout << "Valor maximo de ternas: " << vMax;
} // EmiteTot

main() {
   uint a,
         b,
         c,
         cantTernas,
         valMin,
         valMax;

   EmiteInic(valMin,valMax);
   cantTernas = ObtDato();
   for (uint i = 1; i <= cantTernas; i++) {
      ObtTerna(a,b,c);
      OrdTerna(a,b,c);
      VerifMinMax(a,c,valMin,valMax);
      EmiteTerna(a,b,c);
   }
   EmiteMinMax(valMin,valMax);
```

```
   return 0;
}
```

Funciones NO Borland

cbrt(x) : retorna la raíz cúbica de x. Ej.: cout << cbrt(27); emitre 3.
exp2(x) : retorna el antilogaritmo o exponencial de x de base 2. Ej.: cout << exp2(2); emite 256.
log2(x) : retorna el log en base 2 de x. Ej.: cout << log2(1024); emite 10.
logb(x) : retorna el log en base binaria de x. Ej.: cout << logb(1024); emite 10.
trunc(x) : retorna el entero de x. Ej.: cout << trunc(3.732); emite 3.
round(x) : retorna la parte entera redondeado al entero más cercano. Ej.: cout trunc(3.6); // 4.
remainder(x,y) : retorna el resto entero entre x e y.
fdim(x,y) : si x < y retorna cero.
fmax(x,y) : retorna el máximo valor x si x > y, sino, y.
fmin(x,y) : retorna x si x < y, sino, y.
isgreater(x,y) : retorna 1 si x > y, sino, 0.
isless(x,y) : retorna 1 si x < y, sino, 0.
isgreaterequal(x,y) : retorna 1 si x >= y, sino, y0
islessequal(x,y) : retorna 1 si x <= y, sino, 0.

valor	round	floor	ceil	trunc	rint
2.3	2.0	2.0	3.0	2.0	2.0
3.8	4.0	3.0	4.0	3.0	4.0
5.5	6.0	5.0	6.0	5.0	6.0
-2.3	-2.0	-3.0	-2.0	-2.0	-2.0
-3.8	-4.0	-4.0	-3.0	-3.0	-4.0
-5.5	-6.0	-6.0	-5.0	-5.0	-6.0

π : 4 – 4 / 3 + 4 / 5 – 4 / 7 + 4 / 9 … Serie de Leibniz.

$$\prod_{n=1}^{\infty} \frac{2n-1}{2n} * \frac{2n+1}{2n} = \frac{2}{\pi}$$

$$\frac{1 * 3 * 3 * 5 * 5 * 7 * 7 * 9}{2 \quad 2 \quad 4 \quad 4 \quad 6 \quad 6 \quad 8 \quad 8}$$

$$\prod_{n=1}^{\infty} 2 * \left(\frac{2n}{2n-1} * \frac{2n}{2n+1} \right) = \pi$$

Producto de Wallis

$$\frac{2 * 2 * 4 * 4 * 6 * 6 * 8}{1 \quad 3 \quad 3 \quad 5 \quad 5 \quad 7 \quad 7}$$

Problema de Basilea

$$1 + 1 + 1 + 1 + \ldots$$

$$1^2 \quad 2^2 \quad 3^2 \quad 4^2$$

U.T.N. F.R.B.A. Algoritmos y Estructuras de Datos Prof.: Hugo A. Cuello
Guía de ejercicios Nro. 1
Estructuras de Control de Programas, Datos Primitivos y Módulos –Funciones que retornan o no valores-.

Aplicar **en los siguientes ejercicios, comprensión del problema, diseño de la estrategia y desarrollar el algoritmo.**

Guía de Ejercicios #1

1. Dados los catetos a y b ambos reales y positivos, de un triángulo rectángulo, se solicita hallar la medida de su hipotenusa.
2. Dados dos valores a y b, enteros y distintos, se pide emitir un cartel apropiado que informe cual es el mayor entre ellos.
3. Dados los catetos de un triángulo rectángulo, hallar la superficie de un cuadrado, siendo uno de sus lados la medida de su hipotenusa, si ella es mayor a 100; caso contrario, es el doble de su hipotenusa.
4. Hallar la medida de la hipotenusa de cada triángulo rectángulo, sabiendo las medidas de sus catetos, el proceso finaliza cuando se ingrese un valor menor o igual a cero. Además se requiere informar, cantidad de triángulos y porcentajes con hipotenusa mayor a 100, promedio de las medidas de las hipotenusas, cantidad de triángulos y porcentajes con hipotenusa menor o igual a 100.
5. Dados dos valores a y b, enteros positivos incluido el cero, hallar la potencia de a^b. Sí a = 0 entonces b > 0, sí b = 0 entonces a > 0.
6. Dado un valor n, entero positivo incluido el cero, hallar el factorial de n.
7. Hallar el número irracional e, ingresando previamente un valor n, que indica la cantidad de términos a calcular, con n>=0.
8. Ingresar un valor n, luego calcular la sumatoria de los números naturales hasta n incluido y emitir su resultado.
9. Encontrar otra solución al punto anterior para optimizar el tiempo de proceso.
10. Dado un valor n, entero positivo incluido el cero, obtener cada uno de los términos de la sucesión de Fibonacci.
11. Dado un valor n, entero positivo, obtener la sumatoria de la sucesión de Fibonacci hasta encontrar un valor mayor a n.
12. Dada una lista de n, valores enteros, encontrar el mayor valor y su posición.
13. Dada una lista de valores enteros y un valor x también entero, determinar si el valor x está en la lista dando un mensaje apropiado e informar en qué posición fue encontrado; caso contrario informar con otro mensaje apropiado.
14. De c/u. de los participantes del Rally Paris-Dakar se conocen, al finalizar una etapa, los siguientes datos:

    Código(0: fin del proceso, 1: auto, 2:camión, 3: moto), número del vehículo, apellido y nombre del piloto y del copiloto, nombre de la escudería y tiempo utilizado. Se debe informar para cada tipo de vehículo los datos del ganador.
15. Desarrollar un algoritmo para:
    a) Obtener y emitir los tres primeros números perfectos.
    b) Ídem anterior, pero aplicando el siguiente método:
       1 + 2 = 3; 3 * 2 = 6.

3 + 4 = 7; 7 * 4 = 28.
7 + 8 = 15.
15 + 16 = 31; 31 * 16 = 496.

c) Ídem anterior, pero aplicando el siguiente método:

$2^{n-1} * (2^n - 1)$

Probar con n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, …

16. Realizar una prueba de escritorio para seguir las acciones del siguiente algoritmo e indicar que emite, luego modularizar el bloque 1, 2 y 3:

17. Crear una función que recibe tres parámetros, el primero es un valor booleano, y los otros 2, valores enteros. La función deberá retornar el primer valor entero, si el valor booleano es verdadero, caso contrario se retorna el segundo valor entero.

18. Crear una función que recibe dos valores enteros y retorne el **M.C.D.**
Nota: Aplicar la siguiente estrategia:

Repetir mientras el segundo parámetro sea distinto de cero
Obtener el resto(parámetro1, parámetro2)
Asignar al parámetro1 lo del parámetro2
Asignar al parámetro2 lo del resto
FinRepetir
Retornar el valor del parámetro1

19. Crear la función sgn(x). A diferencia de la función matemática, sí x = 0, deberá retornar cero.

20. Crear una función que recibe como parámetros dos valores enteros y retorne el resto de la división entera.

21. Crear una función que recibe como parámetros dos valores enteros y retorne verdadero si el primero es divisible por el segundo; caso contrario, retornar falso.

22. En un curso se toman exámenes a los alumnos por medio de una computadora. Las preguntas que se realizan son: Calcular el perímetro de un triángulo y su tipo (**I**sósceles, **E**scaleno, E**Q**uilátero), sabiendo como datos las medidas de sus lados.
Se solicita que se informe por cada alumno su nombre y un mensaje apropiado si respondió bien o mal -se responde bien si contestó correctamente ambas preguntas-.
Al finalizar el proceso se deberá emitir cantidad de respuestas correctas e incorrectas. Indique ud. el valor centinela. Se generará al azar las medidas de los lados y:

a. Se asume que forman triángulo.
b. Validar que formen triángulo. (Si (a< b+c) ^ (b<a+c) ^ (c<a+b))

23. Dado un valor entero n, y a continuación n ternas de valores enteros sin orden, se deberán emitir cada una de esas ternas en forma ordenada decreciente en las mismas variables. Por fin de proceso, se debe emitir el menor y el mayor valor de todas las ternas ingresadas.
24. Se conocen los siguientes datos:
    a. *Código de vendedor, (3 díg.)*
    b. *Cantidad vendida, (4 díg.)*
    c. *Descripción de artículo, (str20)*
    d. *Precio unitario. (5 díg.,2 díg.)*

    por cada una de las ventas realizadas. Los datos están agrupados por *código de vendedor* y el proceso finaliza cuando se ingresa un código de vendedor igual a 0.

    Se pide emitir:

    Por cada vendedor su *código* y, por cada *venta* realizada; la *cantidad*, descripcion del *artículo*, el *precio* y el *importe total de ese ítem*.

    Al finalizar ese vendedor se deberá emitir el *total de importe vendido*.

    Por fin del proceso emitir el *importe total vendido por todos los vendedores y el vendedor que más vendió en importe*.

ANEXOS

Ejercicios a resolver exclusivamente por el estudiante aplicando los temas vistos en la Unidad I: Metodología (Análisis, Diseño, Desarrollo, Codificación y Ejecución), datos primitivos, estructuras de Control de Programas (E.C.P.), Módulos.

1. La *Conjetura de Collatz*:
   Se ingresa por única vez un valor entero positivo, a continuación se aplica una de las siguientes reglas:
   Regla #1: Si el valor entero es impar, entonces, se lo multiplica por 3 y se le suma 1.
   Regla #2: Si el valor entero es par, entonces, se lo divide por 2.
   Luego a cada nuevo valor obtenido, se vuelven aplicar las mismas reglas.
   El proceso finaliza cuando se alcance el valor 1.
   Observación: ¿qué sucede si a este valor 1 el cual será el menor valor obtenido, se vuelve aplicar las reglas indicadas?
   Para todo valor entero n ¿se alcanzará llegar a 1?
   ¿qué aplicación útil, si es que la tiene, podría ser posible?
   ¿Existe algún patrón que permita obtener alguna posible aplicación práctica?
2. Dados los coeficientes de una función cuadrática emitir sus raíces, acompañadas de un cartel apropiado, si ambas raíces son iguales, si son distintas o complejas conjugadas, si tiene un máximo o mínimo, las coordenadas del vértice, si el eje y (eje de ordenada) es simétrico, en caso que no lo sea, si está desplazado a la izquierda o a la derecha.
3. Dado un valor n entero determinar el valor aproximado de la constante irracional pi, aplicando algún proceso matemático.
4. Dibujar la gráfica de la función sen *x* en el intervalo de x = [0, 2π].

Triángulos según sus ángulos

Rectángulo: $a^2 + b^2 = c^2$

Si $a^2 + b^2 – c^2 = 0$ v $a^2 + c^2 – b^2 = 0$ v $b^2 + c^2 – a^2 = 0$ entonces es rectángulo.

$A = b^2 + c^2 – a^2$, $B = a^2 + c^2 – b^2$, $C = a^2 + b^2 – c^2$.

Si (A = 0 v B = 0 v C = 0) entonces es rectángulo. |__

Si (A > 0 v B < 0 v C < 0) entonces es acutángulo. /__

Si (A < 0 v B < 0 b C < 0) entonces es obtusángulo. __

Unidad II Capítulo I

Unidad II – Capítulo I - Cadenas de Caracteres y Punteros:

Punteros

Antes de entrar de lleno al tratamiento de cadenas, es conveniente conocer otros temas que serán tratados con mayor profundidad más adelante, estos temas tienen que ver precisamente con los punteros. Un puntero es una variable cuyo contenido es una dirección de memoria y se dice, que ese puntero apunta a la dirección de memoria que contiene. El lenguaje C/C++ ofrece grandes posibilidades para trabajar con punteros, ya que, el lenguaje está muy enriquecido en el manejo de los punteros, para ello ofrece una aritmética de punteros muy versatil. El tamaño de una variable puntero es de 4 bytes.

& operador de dirección.

El operador & extrae la dirección de un objeto indicado a continuación, es un operador monario.

Ejemplo de punteros

```
int   x,
    *pc;

pc = &x;
*pc = 5;
cout << x << endl; // Emite: 5.
```

En la posición de memoria donde está ubicada la variable x se le asignó el valor 5 por medio del puntero pc, el cual apunta a esa posición de memoria ya que se le asignó previamente la dirección de x.

```
*pc += 3;
cout << *pc << endl; // Emite: 8.
cout << x << endl; // Emite: 8.
```

*** operador de indirección desreferencia o contenido.**

int i, *p, *q, j, *r;
p = &i; // asigna a p la dirección de i, por lo que se dice que p apunta a i.
q = &j; // asigna a q la dirección de j, es decir, q apunta a j.
r = p; // asignación entre punteros, solo válido si apuntan a igual tipos de datos.
*p = 2; // asigna a la dirección apuntada por p el valor 2. Como p apunta a la var. i,

Agregado:
system(cad); p.e.: system(“dir”); system(“pause”);
system(color nn); p.e.: system(color 5);
// entonces y pasa a contener el valor 2.
*q = *p + 3; // asigna a la dirección apuntada por q, esto es, la dirección en donde se encuentra la var. j, el resultado de sumar al contenido de la dirección apuntada por p,

que es, 2, ya que, p apunta a la dirección de la var. i, con el valor constante 3, es decir, 5. Por lo tanto la var. j, pasa a tomar el valor 5 como contenido.

Para tomar en cuenta

int *p;
double *q;
void *r;
r = q; // ok. porque r es puntero genérico.
p = r; // error ya que son punteros a distinto tipos de datos.
*r = 3; // error ya que no se puede asignar un valor de tipo específico a un puntero sin tipo o genérico.
p = r; // error ya que son punteros a distinto tipo de datos.
p = (int *) q // ok. se aplica un casting o molde y siendo p un puntero a un tipo int, el miembro derecho de la asignación aplicando el cast convierte el tipo puntero de double forzado a puntero a int, por lo que la asignación es correcta ahora, vale decir, son punteros al mismo tipo de datos o en tal caso el miembro derecho de la asignación es una expresión que genera un resultado de puntero de tipo int.

Aritmética de punteros

int *p;

int i, j;

p = &i; // asigna al contenido de p la dirección en donde se localiza la var. i.

Si el compilador asignó los siguientes 4 bytes contiguos de i a la var. j, entonces si hacemos p++, decimos que p ahora apunta a j.

p++; // incrementa la dirección a la que apunta p en tantos byte como el tamaño del tipo al que apunta, en este caso particular, si el tipo int son 4 bytes, entonces el incremento es 4.

Otras formas de hacer lo mismo

```
p = p + 1;
++p;
p += 1;

int a, b, c;
int *p, *q;
int **r; // puntero a puntero.
p = &a;
*p = 1;
q = &b;
Pq = 2;
p = q;

*p = 0;
q = &c;
*q = 3;
cout << a << b << c << endl;
```

Emite: 1, 0, 3. ¿por qué?

```
p = &q;
*p = q;
*q = 1;
cout << a << b << c << endl;
```

Emite: 1, 0, 1. ¿por qué?

Tratamientos de cadenas de caracteres en C / C++

Las cadenas de caracteres son tan importantes como los números en el tratamiento de los procesos que deba realizar una computadora. Ejemplos de tratamiento de cadenas pueden ser muy diversos, desde el reconocimiento de patrones conocido como *scanner*, que deba realizar una máquina teórica como lo son los Autómatas Finitos o una máquina de Pila para llevar a cabo el análisis sintáctico, conocido como *parser*, hasta los casos de búsqueda de palabras o frases, como lo es al escribir una línea de comandos, o buscar en la internet, la concatenación de cadenas como una de las operaciones esenciales y hacer más simple una comparación de dos o más claves, así también como ordenar una base de datos por diferentes criterios, mucho de los cuales incluyen cadenas de caracteres. Por lo tanto, hay muy buenas razones para darle importancia al tratamiento de cadenas.

El lenguaje C / C++ ofrece por medio de la librería **string.h** un conjunto de funciones para el tratamiento de cadenas de caracteres y por otro lado el lenguaje C++ en forma exclusiva, ofrece por medio de la clase string un conjunto de operadores, métodos, iteradores, constructores para el tratamiento de cadenas dinámicas con longitud variable.

Por lo tanto, el siguiente esquema establece estas formas diversas de definir tipos de string o sartas o cadenas de caracteres.

```
cin.ignore();
cin.get(cad,size);
cin.ignore(1000,'\n'); //x ingreso x teclado.
Desactivar con // si se pegan.
```

Forma "clásica" utilizada por el lenguaje C

Disposición o arreglo de caracteres

1. char **cad[n];**

Puntero a char:

2. char ***pcad;**

```
cin.get(cad,size); //Preferible
cin.ignore(1000,'\n'); //Usar ingreso por teclado.
                       // No usar si se pegan los datos.
```

Forma "clase string" utilizada por el lenguaje C++ exclusivamente

3. string **str;**

Cadenas como arreglo de caracteres

En la forma de arreglo de caracteres el espacio de almacenamiento para la cadena queda establecida en tiempo de compilación, el indicador n es un valor constante, no variable que indica la cantidad de caracteres que contendrá la cadena, la cual incluye un

valor centinela o marca de fin de la cadena representado por el carácter '\0' o NULL. Por lo tanto la cadena contendrá realmente n – 1 caracteres.

Una cadena en el lenguaje C/C++ es una secuencia u ocurrencia, también llamada disposición o arreglo de caracteres, el cual finaliza cuando se encuentre un carácter especial denominado carácter nulo o NULL, representado por '\0'. En muchas ocaciones una cadena de caracteres está representado en el lenguaje C/C++ por un arreglo de caracteres, el cual debemos indicar la cantidad máxima que contendrá incluido el carácter de terminación de la cadena o carácter nulo. Existe una relación muy estrecha entre un arreglo y un puntero, ya que el arreglo apunta al primer byte de ese conjunto de bytes que pertenecen a esa disposición de elementos. Por otra lado, estamos en condiciones de afirmar que el lenguaje no ofrece herramientas para manejar directamente una cadena de caracteres o vulgarmente conocida como strings o sartas. Si queremos copiar cadenas, o comparar cadenas, o determinar la longitud de una cadena entre otras operaciones estamos obligados a aplicar la función específica para llevar a cabo estas operaciones, como se verá más adelante. Las siguientes definiciones establecen un arreglo de caracteres o char y un puntero a carácter o char.

Ejemplos de tipo cadenas

char cad[21], *pcad;

La variable **cad** es una disposición de caracteres, es un espacio asignado por el compilador de 21 bytes, las posiciones van desde 0 (cero) hasta la 19, el cual nos indica que la cadena más larga serán de 20 caracteres para este ejemplo y el espacio extra, el de la posición 20 reservado para el carácter nulo '\0', siempre y cuando la cadena contenga la cantidad máxima de caracteres que para este ejemplo es de 20 caracteres. La cadena vacía es la cadena que no contiene caracteres, y la marca de fin de la cadena '\0', se ubica en la posición cero.

Ejemplo

strcpy(cad,"abcde");

strcpy(cad**,"");** // **NO usar '\0' o NULL**, no es error de sintaxis, pero, produce efectos adversos; en cambio usar comillas seguido de comillas, para asignar la cadena vacía.

En este ejemplo observamos que la función primitiva *strcpy* copia una cadena en una variable de tipo cadena de caracteres. Las posiciones que van desde cero hasta cuatro contendrán los caracteres 'abcde' y la posición cinco tendrá la marca de fin de la cadena '\0', y esta marca se asigna de manera automática.

El siguiente gráfico detalla la situación de la variable cad:

Para leer cadenas de consola con espacios usar mejor: cin.get(cad,n); cin.ignore(N,'\n');

Longitud física de 21 bytes, establecida en tiempo de compilación, es estática.

Longitud lógica, establecida en tiempo de ejecución, en este momento es de 5.

cad

a	b	c	d	e	\0															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Si queremos averiguar el tamaño o espacio de almacenamiento físico para la variable cad, la función sizeof nos ofrece esta posibilidad:

cout << sizeof(cad) << endl; // Emite: 21.

Si quisieramos averigüar la longitud lógica de la cadena cad, la función primitiva empleada es strlen:

cout << strlen(cad) << endl; // Emite: 5.

Cadenas como puntero a caracter

Por otro lado el ejemplo para la variable **pcad**, decimos que es un puntero a char, no tiene espacio reservado para asignarle una cadena, solo se reserva el espacio para el puntero **pcad** en tiempo de compilación de **4 bytes** y es el programador el que deba establecer ese espacio en tiempo de ejecución para asignar una cadena. Para hacer esto se utiliza la palabra reservada **new** debemos indicarle el espacio a reservar, y si se localiza un espacio de memoria contiguo de ese tamaño retorna una dirección de memoria referida al primer byte del tamaño o bloque a reservar. Este espacio de almacenamiento se realiza en una parte de la memoria principal de la RAM, denominada HEAP o montículo, ya que es un área de memoria restante disponible para estos casos. Podría suceder que no se encuentre memoria disponible al tamaño requerido, aunque realmente haya espacio pero no contiguo. En ese caso **new** retornará un valor de dirección especial NULL o puntero que no apunta a nada.

Ejemplo de new con arreglo de caracteres

pcad = **new** char [21]; // se reserva en el heap un espacio de 21 bytes y retorna la dirección del primer byte de ese bloque de memoria contigua.

strcpy(pcad,"abcde");

```
cout << pcad << endl; // Emite: abcde
```

En cambio:

```
cout << *pcad << endl; // Emite: 'a'.   // Oops!!!… pcad vs. *pcad
cout << *++pcad << endl; // Emite: 'b'.
cout << *--pcad << endl; // Emite: 'a'.
cout << *(pcad+3) << endl; // Emite: 'd'.
```

Oops!

RAM que corresponde al heap o montículo

dirección de memoria en donde está el carácter 'a', en este caso hipotético:

AA00FF50

pcad entonces apunta a esa posición de memoria.

a	b	c	d	e	\0															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

cout << pcad[3] << endl; // Emite: ‘d’.

cout << *(pcad+4) << endl; // Emite: ‘e’.

Convertir cadena C++ (string str) a cadena C (char cad[n], char *cad)

strcpy(cad, str.**c_str**());

Funciones para el tratamiento de Cadenas de Caracteres de la librería string.h

A continuación se presentan algunas de ellas, la convensión adoptada para los parámetros son: cad1 y cad3 son **char***, cad2 y cad4 son **const char***, car es **int**, n es de tipo **size_t**.

strcat(cad1,cad4)

Función que concatena la cadena cad4 al final de la cadena cad1. Retorna char* un puntero a char.

Codificación Cadenas Función: strcat(cad1,cad4)

```
// Uso de la función strcat
#include <fstream>
using namespace std;

int main () {
  char str[80];

  strcpy (str,"Estas ");
  strcat (str,"cadenas ");
  strcat (str,"estàn ");
  strcat (str,"concatenadas.");
  puts (str);
  return 0;
}
```

strcmp(cad3,cad4)

Función que compara las dos cadenas, si la cadena cad3 es menor que la cadena cad4, la función strcmp retorna un valor menor que cero, si son iguales, retorna cero y si es mayor la cadena cad3 que la cadena cad4, la función retorna un valor mayor que cero.

Codificación Cadenas Función: strcmp(cad3,cad4)

```
// Uso de la función strcmp
#include <fstream>
using namespace std;

main () {
  char cad1[21],
         cad2[21];

  strcpy(cad1,"Cadena uno");
  strcpy(cad2,"cadena uno");
  if (strcmp(cad1,cad2) == 0)
    cout << "Las cadenas cad1 y cad2 son iguales" << endl;
  else
    cout << "Las cadenas cad1 y cad2 son distintas" << endl;
   return 0;
}
```

strcpy(cad1,cad4)

Función que copia una cadena cad4 en la cadena cad1 hasta encontrar un NULL, '\0'. La función retorna un puntero a cad1.

Codificación Cadenas Función: strcpy(cad1,cad4)

```
// Uso de la función strcpy
#include <fstream>
using namespace std;

main() {
  char cad1[20];
         *espacio = " ";
         *cad2 = "Tratamiento"; // literar localizada por el compilador en t.ejecución
         *cad3 = "de cadenas";

  strcpy(cad1,cad2);
  strcpy(cad1,espacio);
  strcpy(cad1,cad3);
  cout << cad1 << endl;
  return 0;
}
```

```
// Uso de la función strcpy
#include <fstream>
using namespace std;

main () {
  char cad1[21],
        cad2[21];

  if (strcmp(strcpy(cad1,"Cadena uno"),strcpy(cad2,"Cadena uno")) == 0)
    cout << "Las cadenas cad1 y cad2 son iguales" << endl;
  else
    cout << "Las cadenas cad1 y cad2 son distintas" << endl;
  return 0;
}
```

char *strerror(int *numErr*)

Convierte un número de error en una cadena de caracteres que indica el error el cual depende de la implementación en uso.

```
// Uso de la función strerror

#include <fstream>
#include <iomanip>
using namespace std;
main() {

  for (short i = 0; i < 49; i++)
    cout << setw(3) << i << "  " << strerror(i) << endl;
  return 0;
}
```

strlen(cad3)

Función que retorna un valor entero que indica la cantidad de caracteres que contiene la cadena cad3.

cout << strlen("La cadena cuantos caracteres tiene") << endl;

Emite 34, ¿por qué?

Codificación Cadenas Función: strlen(cad3)

```
// Uso de la función strlen
#include <fstream>
using namespace std;

int main () {
```

```
  char szInput[256];

  cout << "Enter a sentence: ";
  gets (szInput);
  cout << “The sentence entered is “ <<  "((unsigned) strlen(szInput))
        << “ characters long.” << endl;
  return 0;
}
```

strstr(cad3,cad4)

Busca la primer ocurrencia de una subcadena cad4, en una cadena cad3. Retorna un puntero a char si existió la ocurrencia de cad4 en cad3. Si no ocurrió entonces retorna NULL.

Ejemplos de strstr:

Codificación Cadenas Función: strstr(cad3,cad4)

```
// Uso de la función  strstr
#include <fstream>
using namespace std;

main() {
  char *cad1 = “Argentina pertenece al continente Americano”;
        *cad2 = “continente”;
        *pCad;

  pCad = strstr(cad1,cad2);
  cout << “La subcadena es: “ << pCad << endl;
  return 0;
}
```

strtok(cad1,cad2)

Retorna un puntero a la siguiente cadena en cad1. La cad2 representan separadores entre las cadenas. La primer llamada es cad1 pero en las sucesivas llamadas es NULL.

Codificación Cadenas Función: strtok(cad1,cad2)

```
// Uso de la función strtok
#include <fstream>
using namespace std;

int main () {
  char str[] ="- Esto, es un ejemplo de cadenas.";
  char * pch;
```

```
  cout << "Partiendo cadenas \"" << str << "\" en tokens:" << endl;
  pch = strtok (str," ,.-");
  while (pch != NULL) {
    cout << pch << endl;
    pch = strtok (NULL, " ,.-");
  }
  return 0;
}

/* Salida:
  Partiendo cadenas "- Esto, es un ejemplo de cadenas." en tokens:
  Esto
  es
  un
  ejemplo
  de
  cadenas
*/
```

strchr(cad3,car)

Retorna un puntero a la primer aparición de car en cad3, sino NULL.

Codificación Cadenas Función: strchr(cad3,car)

```
// Uso de la función strchr
#include <fstream>
using namespace std;

int main () {
  char str[] = "Esto es un ejemplo de cadenas";
  char * pch;

  cout << "Buscando el carácter  's' en: \"" << str << "\"..." << endl;
  pch=strchr(str,'s');
  while (pch!=NULL) {
    cout << "encontrado en " << pch - str+1 << endl;
    pch = strchr(pch + 1,'s');
  }
  return 0;
}

/* Salida:
  Buscando el caracter ‘s‘ en: " Esto es un ejemplo de cadenas"...
  found at 4
  found at 7
  found at 11
  found at 18
*/
```

strrchr(cad3,car)

Retorna un puntero a la última aparición de car en cad3, sino NULL.

Codificación Cadenas Función: strrchr(cad3,car)

```
// Uso de la función strrchr
#include <fstream>
using namespace std;

int main () {
  char str[] = "This is a sample string";
  char * pch;

  pch=strrchr(str,'s');
  printf ("Last occurence of 's' found at %d \n",pch-str+1);
  return 0;
}

/* Salida:
  Last occurrence of 's' found at 18
*/
```

strncpy(cad1,cad4,n)

Copia los primero n caracteres de cad4 en cad1, si cad4 tiene menos caracteres que lo indicado por n entonces rellena con '\0'.

Codificación Cadenas Función: strncpy(cad1,cad4,n)

```
// Uso de la función strncpy
#include <fstream>
using namespace std;

int main () {
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy to sized buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );
  /* partial copy (only 5 chars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';   /* null character manually added */
  puts (str1);
  puts (str2);
  puts (str3);
  return 0;
}
```

```
/* Salida:
  To be or not to be
  To be or not to be
  To be
*/
```

int strcmpi(const char * cad1, const char * cad2)

Compara ambas cadenas sin hacer diferencias entre mayúsculas de minúscula.

```
// Uso de la funcion strcmpi
// No hace diferencias de mayusculas de minusculas.
// Retorna cero si ambas cadenas son iguales.

#include <fstream>
using namespace std;

main( ) {
  char str1[] = "geeks";
  char str2[] = "GeEks";
  int j = strcmpi ( str1, str2 );

  cout << "La funcion retorna: " << j;
  return 0;
}
```

char *strnset(char * cad, char car, size_t n)

Reemplaza los primeros n caracteres de cad, por el mismo car.

char *strset(char * cad, char car)

Reemplaza todos los caracteres de cad, por el mismo car.

```
// Uso de la funcion strset y strnset

#include <fstream>
using namespace std;

main( ) {
 char cad[] = "abcdefgh";

 cout << strnset(cad,'X',4) << endl; // Reemplaza por 4 X's los primeros 4 caracteres.
 cout << strset(cad,'X');  // Reemplaza todos los caracteres por X's.
 return 0;
}
```

strcspn(cad,subcad)

Retorna la posición de la cadena cad en donde se encontró algún carácter contenido en subcad. También podemos interpretar como que los primeros caracteres hasta la posición menos uno no apareció ninguno de los caracteres indicados por subcad.

Ejemplo:
si cad continene “La cadena contine estos mismos caracteres”
si subcad contiene “xolzt”
La función retorna: 11

strnicmp(cad1,cad2)

Retorna un entero, si ambas cadenas son equivalentes cero, si cad1 es menor que cad2 menor a cero y si cad1 es mayor que cad2 mayor a cero. Se aclara que no hay diferencias entre las mayúsculas de las minúsculas, de ahí que si la función retorna cero, no son iguales, sino, equivalentes.

Ejemplo:
Si cad1 contiene “ESTA ES UNA CADENA DE CARACTERES”
SI cad1 contiene “esta es una cadena de caracteres”
La función retorna: 0 (cero):

strpbrk(cad3,cad4)

Retorna un puntero a la primer aparición de algún carácter de cad4 en cad3, sino NULL.

Codificación Cadenas Función: strpbrk(cad3,cad4)

```
// Uso de la función strpbrk
#include <fstream>
using namespace std;

int main () {
  char str[] = "This is a sample string";
  char key[] = "aeiou";
  char * pch;

  printf ("Vowels in '%s': ",str);
  pch = strpbrk (str, key);
  while (pch != NULL) {
     printf ("%c " , *pch);
     pch = strpbrk (pch+1,key);
  }
  printf ("\n");
  return 0;
}

/* Salida
  Vowels in 'This is a sample string': i i a a e i
*/
```

strncat(cad1,cad3,n)

Agrega los primeros n caracteres de la cadena cad3 a la cadena de destino cad1, incorpora el terminador nulo.

Si la longitud de la cadena origen cad3 es menor que n, solo copia los caracteres de cad3 e incorpora el terminador nulo.

Codificación Cadenas Función: strncat(cad1,cad3,n)

```
// Uso de la función strncat
#include <fstream>
using namespace std;

int main () {
  char str1[20];
  char str2[20];

  strcpy (str1,"To be ");
  strcpy (str2,"or not to be");
  strncat (str1, str2, 6);
  puts (str1);
  strcpy(str1,“To be “);
  strncat(str1,str2,15);
  printf(str1);
  return 0;
}

/* Salida
  To be or not
  To be or not to be
*/
```

strncmp(cad3,cad4,n)

Compara los primeros n caracteres entre cad3 y cad3, la comparación continúa si las posiciones correspondientes en ambas cadenas los caracteres son iguales, en cuanto ocurra una diferencia, o alcance la posición indicada o una de las cadenas finalizó, entonces detiene la comparación.

Codificación Cadenas Función: strncmp(cad3,cad4,n)

```
// Uso de la función strncmp
#include <fstream>
using namespace std;

int main () {
  char str[][5] = { "R2D2" , "C3PO" , "R2A6" };
  int    n;
```

```
  puts ("Looking for R2 astromech droids...");
  for (n=0 ; n<3 ; n++)
    if (strncmp (str[n],"R2xx",2) == 0)
      printf ("found %s\n",str[n]);
  return 0;
}

/* Salida
  Looking for R2 astromech droids...
  found R2D2
  found R2A6
*/
```

strspn(cad2,cad4)

Retorna la cantidad de caracteres contenidos en cad2 que son parte de cad4.

Codificación Cadenas Función: strspn(cad2,cad4)

```
// Uso de la función strspn
#include <fstream>
using namespace std;

int main () {
  int i;
  char strtext[] = "129th";
  char cset[] = "1234567890";

  i = strspn (strtext,cset);
  printf ("The initial number has %d digits.\n",i);
  return 0;
}

/* Salida
  The initial number has 3 digits.
*/
```

strlwr(cad1) –NO ANSI-

Convierte la cad1 a minúsculas. Retorna un puntero a cad1.

Codificación Cadenas Función: strlwr(cad1)

```
// Uso de la función strlwr
#include <fstream>
using namespace std;

main() {
   char cad[51];
```

```
   strcpy(cad,"La CaDEnA 12+15 CONVERTIDA en miNUsCUlA");
   strlwr(cad);
   printf(cad);
   return 0;
}
```

strupr(cad1) –NO ANSI-

Convierte la cad1 a mayúsculas. Retorna un puntero a cad1.

Codificación Cadenas Función: strupr(cad1)

```
// Uso de la función strupr
#include <fstream>
using namespace std;

main() {
   char cad[51];

   strcpy(cad,"La CaDEnA 12+15 convertida en mAYuscULa");
   strupr(cad);
   printf(cad);
   return 0;
}
```

strrev(cad1) –NO ANSI-

Invierte la cad1. Retorna un puntero a cad1.

Codificación Cadenas Función: strrev(cad1)

```
// Uso de la función strrev
#include <fstream>
using namespace std;

main() {
  char cad[51];

  strcpy(cad,"Plateria");
  strrev(cad);
  printf(cad);
  return 0;
}
```

tolower(car) <ctype.h>

Retorna car a minúscula si es letra mayúscula, sino retorna el mismo carácter car.

Codificación Cadenas Función: tolwr(car)

```
// Uso de la función tolower
#include <fstream>
using namespace std;

int main () {
  int i=0;
  char str[]="Test String.\n";
  char c;

  while (str[i]) {
    c=str[i];
    putchar (tolower(c));
    i++;
  }
  return 0;
}

/* Salida
  test string.
*/
```

toupper(car) <ctype.h>

Retorna car a mayúscula si es letra minúscula, sino retorna el mismo carácter car.

Emite “continente Americano”, ¿por qué?

Codificación Cadenas Función: toupper (car)

```
// Uso de la función toupper
#include <fstream>
using namespace std;

int main () {
  int i=0;
  char str[]="Test String.\n";
  char c;

  while (str[i]) {
    c=str[i];
    putchar (toupper(c));
    i++;
  }
  return 0;
}
```

```
/* Salida
  TEST STRING.
*/
```

Funciones de cadenas definidas por el usuario

Al igual que con las funciones numéricas en que el usuario puede crear sus propias funciones, sucede lo mismo con las cadenas de caracteres. A continuación se presentan algunas funciones de cadenas.

Función que retorna el carácter enviado convertido a minúscula si es alfabético mayúscula, sino el mismo carácter.

Función que retorna invertida una palabra enviada:

Una posible aplicación de la función *InvPal* sería comprobar si una palabra es palíndromo, es decir, capicúa, p.e.: capicua ← pal = InvPal(pal), habiendo declarado a capicua de tipo boolean, contendrá false si la palabra pal no es capicúa, sino será true. Pero, ¿podría ocurrir que al invertir una cadena, palabra de un lenguaje, la nueva cadena creada sea una palabra que pertenezca a ese mismo lenguaje?.

p.e.: palreves ← invpal('lámina'), obviando el tilde, asigna 'animal', o esta otra, palreves ← invpal('roma'), asigna 'amor'.

Función que dado un caracter car y un número n, retorna el carácter car repetido n veces, es decir, retorna una cadena de longitud n, en donde cada componente es el caracter car.

También podría realizarse la siguiente funciones espacios en blanco:

Una posible aplicación de la función space o replicate podría ser rellenar una variable de cadena con espacios a derecha, a partir de la longitud lógica + 1 hasta el final, p.e.: cad ← ‘Algoritmos’. cad ← cad + space(pred(sizeof(cad)) – length(cad)).

```
// Uso de la función strlen2.
#include <iostream.h>
#include <conio.h>

int strlen2(char *cad) {
  char *pCad = cad;

  while (*pCad != ‘\0’)
    pCad++;
  return pCad – cad:
} // strlen2

void main() {
  char pal[] = “La cadena tiene una cantidad de caracteres”;

  cout << “La longitud de la cadena es de “ << strlen2(pal) << “ caracteres” << endl;
  getch();
}
// Uso de la función strcpy2.
#include <iostream.h>
#include <conio.h>
```

```
int strcpy2(char *cad1, char *cad2) {
  char *cad3 = cad1;

  while (*cad3++ = *cad2++) ;
  return cad1;
} // strcpy2

void main() {
  char pal1[31],
         pal2[] = "Cadena hasta 30 caracteres";

  clrscr();
  cout << "La copia en cad1 de la cad2 es: " << strcpy2(pal1,pal2) << endl;
  getch();
}
```

```
// Uso de la función strcmp2 versión con arreglos.
#include <iostream.h>
#include <conio.h>

int strcmp2(char *cad1, char *cad2) {
  int i;

  for (i = 0; cad1[i] == cad2[i]; i++ )
    if (cad1[i] == '\0')
      return 0;
  return cad1[i] - cad2[i];
} // strcmp2

void main() {
  char pal1[] = "Cadena hasta 40 caracteres de longitud",
         pal2[] = "Cadena hasta 30 caracteres";
  int cmpPal;

  clrscr();
  cmpPal = strcmp2(pal1,pal2);
  if (cmpPal == 0)
    cout << "Las palabras son iguales" << endl;
  else
    if  (cmpPal < 0)
      cout  << "La palabra pal1 es menor que la palabra pal2" << endl;
    else
      cout << "La palabra pal1 es mayor que la palabra pal2" << endl;
```

```
  getch();
}
```

```
// Uso de la función strcmp2 versión con punteros.
#include <iostream.h>
#include <conio.h>

int strcmp2(char *cad1, char *cad2) {

  for (; *cad1 == *cad2; cad1++, cad2++)
    if (*cad1 == '\0')
      return 0;
  return *cad1 - *cad2;
} // strcmp2

void main() {
  char pal1[] = "Cadena hasta 40 caracteres de longitud",
         pal2[] = "Cadena hasta 30 caracteres";
  int cmpPal;

  clrscr();
  cmpPal = strcmp2(pal1,pal2);
  if (cmpPal == 0)
    cout << "Las palabras son iguales" << endl;
  else
    if  (cmpPal < 0)
      cout  << "La palabra pal1 es menor que la palabra pal2" << endl;
    else
      cout << "La palabra pal1 es mayor que la palabra pal2" << endl;
  getch();
}

char Lower(char car) { // (int car) es =
  if (car >= 'A' && car <= 'Z')
    return car + 'a' - 'A';
  else
    return car;
} // Lower
void Invertir(char *cad) {
  char aux;

  for (unsigned i = 0, j = strlen(cad) - 1; i < j; i++, j--)
      aux = cad[i],
      cad[i] = cad[j],
      cad[j] = aux;
} // Invertir

char *InvPal(char cad[]) {
  char aux;
```

```
  for (unsigned i = 0, j = strlen(cad) - 1; i < j; i++, j--)
      aux = cad[i],
      cad[i] = cad[j],
      cad[j] = aux;
  return cad;
} // InvPal
char *CmbCar(char cad[], char car1, char car2) {
  for (unsigned i = 0; i < strlen(cad); i++)
      if (cad[i] == car1)
          cad[i] = car2;
  return cad;
} // CmbCar
string replicate(char car, unsigned n) {
  string cad = "";

  for (unsigned i = 1; i <= n; i++)
      cad = cad + car;
  return cad;
} // replicate

char *replicate2(char car,unsigned n) {
   char *pcad;
   unsigned i;

   pcad = new char [n];
   for ( i = 0; i < n; i++)
       pcad[i] = car;
   pcad[i] = '\0';
   return pcad;
} // replicate2

char *space(unsigned n) {
   char *pcad;
   unsigned i;

   pcad = new char [n];
   for ( i = 0; i < n; i++)
       pcad[i] = ' ';
   pcad[i] = '\0';
   return pcad;
}//space
```

Tratamiento de cadenas al estilo de C++

Clase string

El lenguaje C++ ofrece otra variante al tratamiento de cadenas que lo hace semejante a otros lenguajes de programación como el Pascal o Basic, en el cual al momento de copiar o comparar o concatenar como así también otras operaciones entre cadenas, no es necesario utilizar funciones para su tratamiento, como lo es en los casos anteriormente visto para el lenguaje C.

Ejemplos de la clase string

Definición de variables de la clase string

```
string str1, str2, str3;
```

Copiar una cadena

```
str1 = "Copiar cadena";
```

Concatenar una cadena

```
str1 = str1 + " de caracteres";

if (str1 == str2)
  cout << "Las cadenas son iguales" << endl;
```

Además diremos que la definición de las variables de tipo string serán punteros a un espacio en la memoria dinámica (heap) para almacenar el contenido al que apunta y además este espacio para el puntero al string también podrá crecer o disminuir según las necesidades del momento.

A diferencia de la forma clásica que poseen un valor centinela al final de la cadena para indicar la finalización, representado por el carácter NULL o '\0', las cadenas de la clase string no terminan con ningún valor centinela.

Si averiguamos el espacio designado a la definición de una variable de la clase string, a través del operador **sizeof**, este nos devolverá que su tamaño es de **16 bytes**, para arquitectura y sofware de **32 bits,** pero, para arquitectura y software de **64 bits**, **sizeof** devolverá que su tamaño es de **32 bytes.**

El tamaño de almacenamiento de un objeto string es de 16 bytes

Podríamos interpretar que el contenido de una variable **string** pudiera contener la dirección en donde apunta al heap o memoria dinámica, al inicio de la cadena, pensemos que esta dirección pesa 4 bytes, de otra dirección también al heap que apunta al final o incluso más allá del final de la cadena de caracteres asignada al heap, entonces tenemos otros cuatro 4 bytes más. Con esto completamos solo la mitad del peso de la variable string, es decir, 16 – (4 **puntero al inicio** + 4 **puntero al final**) = 8 bytes que restan del espacio máximo asignado. Otro aspecto importante de información será cuál es el tamaño actual de la cadena de caracteres, y además el espacio sobrante antes de determinar el redimensionamiento del área de almacenamiento. Suponiendo que estas otras dos componentes más debe ser determinado por valores de tipo **int**, el cual pesa otros 4 bytes, entonces, tenemos 4 bytes para indicar la **longitud de la cadena**, más otros 4 bytes para indicar la **longitud restante** antes de redimensionar el espacio de almacenamiento, nos dan otros 8 bytes, y que sumados a los 8 bytes de punteros al inicio y final del almacenamiento en el heap que le fuera asignado, completamos los 16 bytes que pesa una variable de tipo string.

Con este tipo de cadenas se evitan ciertas situaciones adversas que podrían ocurrir con el manejo tradicional de cadenas en C, por ejemplo, quedarnos sin espacio de almacenamiento al asignar una cadena a una variable de arreglo de caracteres, o de asignar una cadena a una variable puntero a char el cual no se le asigno anticipadamente de un espacio de almacenamiento, por medio de la función de disponibilidad de memoria en el heap, como lo es con malloc en el lenguaje C o con new en C++.

Las cadenas de caracteres en C++ son más versátiles que las cadenas en C. Nos otorgan mayor seguridad al momento de operar con las cadenas. Permiten crear espacios de almacenamiento en forma dinámica y este espacio dinámico una vez asignado, puede más adelante crecer o disminuir de acuerdo a las necesidades del momento; por lo que este espacio decimos que es dinámico y variable, contrario, al manejo en C que serán dinámico pero una vez creado ese espacio no podrá ser ni ampliado ni reducido, ya que será fijo.

Las cadenas en C++ son mas seguras y confiables

El espacio asignado es dinámico y variable

No obstante, este tipo de cadenas no podrán ser empleadas como tipo de datos para archivos binarios, ya que se espera de ellos, cada una de las componentes tengan longitud fija. A pesar de esto, se podrá trabajar internamente en la memoria RAM con el tipo de cadenas string y al momento de necesitar guardar en el archivo, tenemos una herramienta muy necesaria, y es la que convierte una cadena de la forma en C++, o sea, string, a la forma tradicional del lenguaje C del tipo c-string, por medio de la función brindada por C++, el cual es, c_str(), que es una función miembro. Su uso se muesta en el siguiente ejemplo:

```
char cad[21];
string str;
str = “abcdefghijklmnopqrstuvwxyz”;
strncpy(cad,str.c_str(),20);
cout << ‘*’ << cad << ‘*’ << endl; // Emite: *abcdefghijklmnopqrst*
str = “abcdefghij”;
strncpy(cd,str.c_str(),20);
cout << ‘*’ << cad << ‘*’ << endl; // Emite: *abcdefghij*
```

Sleep(n); se debe incluir la librería <windows.h>

<string.h>
strcat(char *, const char *s2);
strchr(const char *s, ubt c);
strcmp(const char *s1, const char *s2);const char *s);

```
const char *s);
strcpy(char *s1, const char *s2);
strspn(const chars1, const char *s2);
strlen(const char *s);
strncat(char *s, const char *s2, size_t n);
strncmp(const char *s1, const char *s2, size_t n);
strncpy(char *s1, const char *s2, size_t n);
strcpy(char *s1, const char *s2);
strpbrk(const char *s1, const char *s2);
strrchr(const char *s, int c);
strspn(const char *s1, const char *s2);
strstr(const char *s1, const char *s2);
strtok(char *x1, const char *s2);
strrerror(int i);
strscspn(const char *s1, const char *s2);
```

<ctype.h>

```
isalnum(car);
isalpha(car);
isdigit(car);
islower(car);
isupper(car);
tolower(car);
toupper(car);
```

Ejemplos de definición de variables de la clase string

```
string str1, str2;
string str3  ("Hola, mundo");
string str4 = "Todo Ok";
string str5 (10,'*');
string str6 = 'a';
```

Ejemplos de la clase string

Codificación clase string Constructor

```
// string constructor

/*
default (1)          string();
copy (2)             string (const string& str);
substring (3)        string (const string& str, size_t pos, size_t len = npos);
from c-string (4)    string (const char* s);
from sequence (5)    string (const char* s, size_t n);
```

```
fill (6)                    string (size_t n, char c);
range (7)                   template <class InputIterator>
                               string  (InputIterator first, InputIterator last);
*/

#include <fstream>
using namespace std;

int main () {
  string s0 ("Initial string");

  // constructors used in the same order as described above:
  string s1;
  string s2 (s0);
  string s3 (s0, 8, 3);
  string s4 ("A character sequence", 6);
  string s5 ("Another character sequence");
  string s6a (10, 'x');
  string s6b (10, '*');      // 42 is the ASCII code for '*'
  string s7 (s0.begin(), s0.begin()+7);
  cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;
  cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;
  cout << "\ns6b: " << s6b << "\ns7: " << s7 << '\n';
  return 0;
}

/*  Salida:
  s1:
  s2: Initial string
  s3: str
  s4: A char
  s5: Another character sequence
  s6a: xxxxxxxxxx
  s6b: **********
  s7: Initial
*/
```

Codificación clase string Operador de asignación

```
// string assigning

/*
string (1)                      string& operator= (const string& str);
c-string (2)                 string& operator= (const char* s);
character (3)                string& operator= (char c);
*/

#include <fstream>
using namespace std;
```

```
int main () {
  string str1, str2, str3;
  str1 = "Test string: ";   // c-string
  str2 = 'x';               // single character
  str3 = str1 + str2;       // string

  cout << str3  << '\n';
  return 0;
}

/* Salida:
  Test string: x
*/
```

Codificación clase string Función begin - end

```
// string::begin/end

/*
      iterator begin();
      const_iterator begin() const;

      iterator end();
      const_iterator end() const;
*/

#include <fstream>
using namespace std;

int main ()
{
  string str ("Test string");
  for ( string::iterator it=str.begin(); it!=str.end(); ++it)
    cout << *it;
  cout << '\n';

  return 0;
}

/*
Salida:
Test string
*/
```

Codificación clase string Función rbegin - rend

```
// string::rbegin/rend

/*
      reverse_iterator rbegin();
      const_reverse_iterator rbegin() const;

      reverse_iterator rend();
       const_reverse_iterator rend() const;
*/

#include <fstream>
using namespace std;

int main () {
  string str ("now step live...");
  for (string::reverse_iterator rit=str.rbegin(); rit!=str.rend(); ++rit)
    cout << *rit;
  return 0;
}

/*
The actual output is:
...evil pets won
*/
```

Codificación clase string Función size() / length() es equivalente.

```
// string::size

/*
size_t size() const;
*/

#include <fstream>
using namespace std;

int main () {
  string str ("Test string");
  cout << "The size of str is " << str.size() << " bytes.\n";
  return 0;
}
/*
Salida:
The size of str is 11 bytes
*/
```

Codificación clase string Función resize()

```
// resizing string

/*
```

```
void resize (size_t n);
void resize (size_t n, char c);
*/
#include <fstream>
using namespace std;

int main () {
  string str ("I like to code in C");

  cout << str << '\n';
  unsigned sz = str.size();
  str.resize (sz+2,'+');
  cout << str << '\n';
  str.resize (14);
  cout << str << '\n';
  return 0;
}

/* Salida:
I like to code in C
I like to code in C++
I like to code
*/
```

reserve (size_t n =0); ejemplo: str.reserve(1520);

max_size(); retorna el valor 4.294.967.291 el cual equivale a 4 Gbytes.

capacity(); retorna el espacio de almacenamiento str.capacity() >= str.size().

Codificación clase string Función clear()

```
// string::clear

/*
void clear();
*/

#include <fstream>
using namespace std;

int main () {
  char c;
  string str;

  cout << "Please type some lines of text. Enter a dot (.) to finish:\n";
  do {
    c = cin.get();
    str += c;
    if (c=='\n')
    {
      cout << str;
```

```
      str.clear();
    }
  } while (c!='.');
  return 0;
}
```

Codificación clase string Función de conversión c_str()

```
// strings and c-strings

/*
const char* c_str() const;
*/

#include <fstream>
using namespace std;

int main () {
  string str ("Please split this sentence into tokens");

  char * cstr = new char [str.length()+1];
  strcpy (cstr, str.c_str());

  // cstr now contains a c-string copy of str

  char * p = strtok (cstr," ");
  while (p!=0)
  {
    std::cout << p << '\n';
    p = strtok(NULL," ");
  }

  delete[] cstr;
  return 0;
}

/* Salida
Please
split
this
sentence
into
tokens
*/
```

Codificación clase string Función find()

```
// string::find

/*
string (1)                    size_t find (const string& str, size_t pos = 0) const;
c-string (2)                  size_t find (const char* s, size_t pos = 0) const;
```

```
buffer (3)                    size_t find (const char* s, size_t pos, size_t n) const;
character (4)                 size_t find (char c, size_t pos = 0) const;
*/

#include <fstream>
using namespace std;

int main () {
  string str ("There are two needles in this haystack with needles.");
  string str2 ("needle");

  // different member versions of find in the same order as above:
  size_t found = str.find(str2);
  if (found!=std::string::npos)
    cout << "first 'needle' found at: " << found << '\n';

  found=str.find("needles are small",found+1,6);
  if (found!=std::string::npos)
    cout << "second 'needle' found at: " << found << '\n';

  found=str.find("haystack");
  if (found!=std::string::npos)
    cout << "'haystack' also found at: " << found << '\n';

  found=str.find('.');
  if (found!=std::string::npos)
    cout << "Period found at: " << found << '\n';

  // let's replace the first needle:
  str.replace(str.find(str2),str2.length(),"preposition");
  cout << str << '\n';

  return 0;
}

/*
Notice how parameter pos is used to search for a second instance of the same search
string. Output:
first 'needle' found at: 14
second 'needle' found at: 44
'haystack' also found at: 30
Period found at: 51
There are two prepositions in this haystack with needles.
*/
```

Codificación clase string función substr()

```
// string::substr

/*
string substr (size_t pos = 0, size_t len = npos) const;
*/

#include <fstream>
using namespace std;

int main () {
  string str="We think in generalities, but we live in details.";
                                    // (quoting Alfred N. Whitehead)

  string str2 = str.substr (3,5);     // "think"

  size_t pos = str.find("live");      // position of "live" in str

  string str3 = str.substr (pos);     // get from "live" to the end

  cout << str2 << ' ' << str3 << '\n';

  return 0;
}

/* Salida
think live in details.
*/
```

Codificación clase string Función replace()

```
// replacing in a string

/*
string (1)        string& replace (size_t pos,  size_t len,  const string& str);
                  string& replace (iterator i1, iterator i2, const string& str);
substring (2)     string& replace (size_t pos,  size_t len,  const string& str,
                  size_t subpos, size_t sublen);
c-string (3)      string& replace (size_t pos,  size_t len,  const char* s);
                  string& replace (iterator i1, iterator i2, const char* s);
buffer (4)        string& replace (size_t pos,  size_t len,  const char* s, size_t n);
                  string& replace (iterator i1, iterator i2, const char* s, size_t n);
fill (5)          string& replace (size_t pos,  size_t len,  size_t n, char c);
                  string& replace (iterator i1, iterator i2, size_t n, char c);
range (6)         template <class InputIterator>
                  string& replace (iterator i1, iterator i2,
                      InputIterator first, InputIterator last);
*/

#include <fstream>
using namespace std;
```

```
int main () {
  string base="this is a test string.";
  string str2="n example";
  string str3="sample phrase";
  string str4="useful.";

  // replace signatures used in the same order as described above:

  // Using positions:                 0123456789*123456789*12345
  string str=base;          // "this is a test string."
  str.replace(9,5,str2);          // "this is an example string." (1)
  str.replace(19,6,str3,7,6);     // "this is an example phrase." (2)
  str.replace(8,10,"just a");     // "this is just a phrase."     (3)
  str.replace(8,6,"a shorty",7);  // "this is a short phrase."    (4)
  str.replace(22,1,3,'!');        // "this is a short phrase!!!"  (5)

  // Using iterators:                                    0123456789*123456789*
  str.replace(str.begin(),str.end()-3,str3);                    // "sample phrase!!!"      (1)
  str.replace(str.begin(),str.begin()+6,"replace");             // "replace phrase!!!"     (3)
  str.replace(str.begin()+8,str.begin()+14,"is coolness",7);    // "replace is cool!!!"    (4)
  str.replace(str.begin()+12,str.end()-4,4,'o');                // "replace is cooool!!!"  (5)
  str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());// "replace is useful."    (6)
  scout << str << '\n';
  return 0;
}

/*
Salida:
replace is useful.
*/
```

Codificación clase string Función append()

```
// appending to string

/*
string (1)                    string& append (const string& str);
substring (2)                 string& append (const string& str, size_t subpos, size_t
sublen);
c-string (3)                  string& append (const char* s);
buffer (4)                    string& append (const char* s, size_t n);
fill (5)                      string& append (size_t n, char c);
range (6)                     template <class InputIterator>
          string& append (InputIterator first, InputIterator last);
*/

#include <fstream>
using namespace std;

int main () {
  string str;
```

```
  string str2="Writing ";
  string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                    // "Writing "
  str.append(str3,6,3);                // "10 "
  str.append("dots are cool",5);       // "dots "
  str.append("here: ");                // "here: "
  str.append(10u,'.');                 // "........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  //str.append<int>(5,0x2E);           // "....." genera error
  cout << str << '\n';
  return 0;
}

/*
Salida:
Writing 10 dots here: ......... and then 5 more.....
*/
```

Codificación clase string Función getline()

```
// extract to string getline

/*
(1)                    istream& getline (istream& is, string& str, char delim);
(2)                    istream& getline (istream& is, string& str);
*/

#include <fstream>
using namespace std;

int main () {
  string name;

  cout << "Please, enter your full name: ";
  getline (cin,name);
  scout << "Hello, " << name << "!\n";

  return 0;
}

/* Salida
  Hello, John AC!
*/
```

Codificación clase string Operador de concatenación +

```
// concatenating strings
```

```
/*
string (1)                    string operator+ (const string& lhs, const string& rhs);
c-string (2)                  string operator+ (const string& lhs, const char*   rhs);
          string operator+ (const char*   lhs, const string& rhs);
character (3)                 string operator+ (const string& lhs, char          rhs);
          string operator+ (char          lhs, const string& rhs);
*/

#include <fstream>
using namespace std;

main () {
  string firstlevel ("com");
  string secondlevel ("cplusplus");
  string scheme ("http://");
  string hostname;
  string url;

  hostname = "www." + secondlevel + '.' + firstlevel;
  url = scheme + hostname;

  cout << url << '\n';

  return 0;
}

/* Salida
  http://www.cplusplus.com
*/
```

Codificación clase string Función erase()

```
// string::erase

/*
sequence (1)                  string& erase (size_t pos = 0, size_t len = npos);
character (2)                 iterator erase (iterator p);
range (3)                     iterator erase (iterator first, iterator last);
*/

#include <fstream>
using namespace std;

int main () {
  string str ("This is an example sentence.");
  cout << str << '\n';
                              // "This is an example sentence."
  str.erase (10,8);                 //          ^^^^^^^^
  cout << str << '\n';
                              // "This is an sentence."
  str.erase (str.begin()+9);        //          ^
```

```
  cout << str << '\n';
                                    // "This is a sentence."
  str.erase (str.begin()+5, str.end()-9);  //       ^^^^^
  cout << str << '\n';
                                    // "This sentence."
  return 0;
}

/* Salida
This is an example sentence.
This is an sentence.
This is a sentence.
This sentence.
*/
```

Codificación clase string Función append()

```
// appending to string

/*
string (1)                          string& append (const string& str);
substring (2)                     string& append (const string& str, size_t subpos, size_t
sublen);
c-string (3)                      string& append (const char* s);
buffer (4)                          string& append (const char* s, size_t n);
fill (5)                            string& append (size_t n, char c);
range (6)                           template <class InputIterator>
              string& append (InputIterator first, InputIterator last);
*/

#include <fstream>
using namespace std;

int main () {
  string str;
  string str2="Writing ";
  string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                    // "Writing "
  str.append(str3,6,3);                  // "10 "
  str.append("dots are cool",5);          // "dots "
  str.append("here: ");                  // "here: "
  str.append(10u,'.');                  // "........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  //str.append<int>(5,0x2E);               // "....." genera error
  cout << str << '\n';
  return 0;
}

/* Salida
```

```
Writing 10 dots here: ..... and then 5 more.....
*/
```

Codificación clase string Función copy()

```
// copy

/*
size_t copy (char* s, size_t len, size_t pos = 0) const;
*/

#include <fstream>
#include <string>

int main () {
  char buffer[20];
  string str ("Test string...");
  size_t length = str.copy(buffer,6,5);
  buffer[length]='\0';
  cout << "buffer contains: " << buffer << '\n';
  return 0;
}

/* Salida
 buffer contains: string
*/
```

Codificación clase string Función rfind()

```
// string::rfind

/*
string (1)          size_t rfind (const string& str, size_t pos = npos) const;
c-string (2)        size_t rfind (const char* s, size_t pos = npos) const;
buffer (3)          size_t rfind (const char* s, size_t pos, size_t n) const;
character (4)       size_t rfind (char c, size_t pos = npos) const;
*/

#include <fstream>
using namespace std;

#include <cstddef>

int main () {
  string str ("The sixth sick sheik's sixth sheep's sick.");
  string key ("sixth");

  size_t found = str.rfind(key);
  if (found!=string::npos)
    str.replace (found,key.length(),"seventh");
```

```
  cout << str << '\n';

  return 0;
}

/* Salida
  The sixth sick sheik's seventh sheep's sick.
*/
```

Codificación clase string Función compare()

```
// comparing apples with apples

/*
string (1)          int compare (const string& str) const;
substrings (2)   int compare (size_t pos, size_t len, const string& str) const;
                     int compare (size_t pos, size_t len, const string& str,
                         size_t subpos, size_t sublen) const;
c-string (3)       int compare (const char* s) const;
                     int compare (size_t pos, size_t len, const char* s) const;
buffer (4)          int compare (size_t pos, size_t len, const char* s, size_t n) const;
*/

#include <fstream>
using namespace std;

int main () {
  string str1 ("green apple");
  string str2 ("red apple");

  if (str1.compare(str2) != 0)
    cout << str1 << " is not " << str2 << '\n';

  if (str1.compare(6,5,"apple") == 0)
    cout << "still, " << str1 << " is an apple\n";

  if (str2.compare(str2.size()-5,5,"apple") == 0)
    cout << "and " << str2 << " is also an apple\n";

  if (str1.compare(6,5,str2,4,5) == 0)
    cout << "therefore, both are apples\n";

  return 0;
}
/* Salida
green apple is not red apple
still, green apple is an apple
and red apple is also an apple
therefore, both are apples
*/
```

Funciones de Verificación y Conversión de caracteres, librería ctype.h

isalpha(car) : Función que verifica si el contenido del caracter car es un caracter letra mayúscula o minúscula, esto es, A-Z, a-z, en estos casos la función retorna cero por falso o 1 por verdadero.

isupper(car) : Función que verifica si el contenido del caracter car es un caracter letra mayúscula, esto es, A-Z, en estos casos la función retorna cero por falso o 1 por verdadero.

islower(car) : Función que verifica si el contenido del caracter car es caracter letra minúsucla, esto es, a-z, en estos casos la función retorna cero por falso o 1 por verdadero.

isdigit(car) : Función que verifica si el contenido del caracter car es dígito, esto es, 0-9, en estos casos la función retorna cero por falso o 1 por verdadero.

isalnum(car) : Función que verifica si el contenido del caracter car es letra A-A, a-z o dígito 0-9, en estos casos la función retorna cero por falso o 1 por verdadero.

isspace(car) : Función que verifica si el contenido del caracter car es espacio, tabulador, nueva línea, retorno, avance de línea o tabulador vertical, en estos casos la función retorna cero por falso o 1 por verdadero.

isascii(car) : Función que verifica si el contenido del carácter car está entre 0 y 0x7F, en este caso retorna un valor distinto de cero, sino cero.

iscntrl(car) : Función que verifica si el contenido del carácter car está entre 0 y 0x1F o 0x7F (tecla DEL), en este caso retorna un valor distinto de cero, sino cero.

isgraph(car) : Función que verifica si el carácter car es cualquier carácter imprimible distinto de espacio, en este caso retorna un valor distinto de cero, sino cero.

isprint(car) : Función que verifica si el carácter car está en el intervalo [0x20, 0x7e].

ispunct(car) : Función que verifica si el carácter car es un carácter de puntuación o un espacio, en este caso retorna un valor distinto de cero, sino cero.

isxdigit(car) : Función que verifica si el contenido del carácter car es un dígito hexadecimal 0-9, A-F, a-f

toupper(car) : Función que convierte a letra mayúscula, si el contenido del carácter car es letra minúsucla, retornando el carácter convertido, o en caso contrario, el mismo carácter.

tolower(car) : Función que convierte a letra minúscula, si el contenido del carácter car es letra mayúscula, retornando el carácter convertido, o en caso contrario, el mismo carácter.

Codificación de cadenas definidas por el usuario

```
/*
   Id. Programa: MisLibsCad.cpp
   Autor.......: Lic. Hugo Cuello
   Fecha.......: ago-2014
   Comentario..: Funciones para el tratamiento de cadenas
*/

#include<iostream>
using namespace std;

char *mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL",
                  "MAYO","JUNIO","JULIO","AGOSTO",
                  "SEPTIEMBRE","OCTUBRE","NOVIEMBRE","DICIEMBRE"
               };

char LowCase(char car) {

  if (car >= 'A' && car <= 'Z')
    return car + 'a' - 'A';
  else
    return car;
} // LowCase

string replicate(char car, unsigned n) {
  string cad = "";

  for (unsigned i = 1; i <= n; i++)
    cad += car;
  return cad;
} // replicate

char *replicate(char car, int n) {
  char *cad;
  int i = 0;

  cad = new char [n + 1];
  for ( ; i < n; i++)
    cad[i] = car;
  cad[i] = '\0';
  return cad;
} //replicate

char *space(int n) {
  char *cad;
  int i = 0;

  cad = new char [n + 1];
  for ( ; i < n; i++)
    cad[i] = ' ';
```

```
  cad[i] = '\0';
  return cad;
} //space

string CmbCar(string cad, char car1, char car2) {

  for (unsigned i = 0; i <= cad.length(); i++)
    if (cad[i] == car1)
      cad[i] = car2;
  return cad;
} // CmbCar

string InvPal(string cad) {
  string palInv = "";

  for (unsigned i = 0; i <= cad.length(); i++)
    palInv = cad[i] + palInv;
  return palInv;
} // InvPal

char *InvPal(char *pal) {
  char aux;
  int j;

  for (int i = 0, j = strlen(pal) - 1; i < j; i++, j--) {
    aux = pal[i];
    pal[i] = pal[j];
    pal[j] = aux;
  }
  return pal;
} //InvPal

char *InvPalV(char pal[]) {
  char *pcad,
       *pcadF,
       aux;
  int j;

  pcadF = pal;
  while (*pcadF)
    pcadF++; //recorre pal hasta el final
  pcadF--;
  for (pcad = pal; pcad < pcadF; pcad++, pcadF--) {
    aux = *pcad;
    *pcad = *pcadF;
    *pcadF = aux;
  }
  return pal;
} //InvPalV
```

```
char *strlcat(char *cadDes, char *cadOri) {
  char *cadAux;

  cadAux = new char [strlen(cadDes) + strlen(cadOri) + 1];
  strcpy(cadAux,cadOri);
  strcat(cadAux,cadDes);
  strcpy(cadDes,cadAux);
  delete cadAux;
  return cadDes;
} //strlcat

string MesStr(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
    case  8: return "Agosto";
    case  9: return "Septiembre";
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStr

char *MesStrP(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
    case  8: return "Agosto";
    case  9: return "Septiembre";
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStrP

char *NomMes(unsigned int mes) {
```

```
   char *MesStr;

   MesStr = new char [11];
   switch (mes) {
      case  1: strcpy(MesStr,"Enero"); break;
      case  2: strcpy(MesStr,"Febrero"); break;
      case  3: strcpy(MesStr,"Marzo"); break;
      case  4: strcpy(MesStr,"Abril"); break;
      case  5: strcpy(MesStr,"Mayo"); break;
      case  6: strcpy(MesStr,"Junio"); break;
      case  7: strcpy(MesStr,"Julio"); break;
      case  8: strcpy(MesStr,"Agosto"); break;
      case  9: strcpy(MesStr,"Septiembre"); break;
      case 10: strcpy(MesStr,"Octubre"); break;
      case 11: strcpy(MesStr,"Noviembre"); break;
      case 12: strcpy(MesStr,"Diciembre"); break;
   }
   return MesStr;
} // NomMes

main() {
   char *pCad;

   pCad = NomMes(12);
   cout << pCad << endl;
   delete [] pCad;
   return 0;
}
```

```
/* strcpy example */
//char * strcpy ( char * destination, const char * source );

#include <fstream>
using namespace std;

int main () {
  char str1[] = "Sample string";
  char str2[40];
  char str3[40];

  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  cout << "str1: " << str1 << endl << "str2: " << str2 << endl << "str3: " << str3 << endl;
  return 0;
}

/* Salida
  str1: Sample string
  str2: Sample string
  str3: copy successful
```

```
*/
```

```
/* strncpy example */

#include <fstream>
using namespace std;

int main () {
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy hasta el tamaño del buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );

  /* copia parcial (solo 5 cars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';   /* caracter null agregado manualmente */

  puts (str1);
  puts (str2);
  puts (str3);

  return 0;
}

/* Salida
  To be or not to be
  To be or not to be
  To be
*/
```

Sub-cadena e insertar

```
/* strstr example */
// const char * strstr ( const char * str1, const char * str2 );

#include <fstream>
using namespace std;

int main () {
  char str[] ="This is a simple string";
  char * pch;

  pch = strstr (str,"simple");
  strncpy (pch,"sample",6);
  puts (str);
  return 0;
}
```

```
/*
This example searches for the "simple" substring in str and replaces that word for "sample".

Salida
  This is a sample string
*/
```

Funciones de conversión de arreglo de caracteres a número.

double atof(const char *s); // convierte a double.
int atoi(const char *s); // convierte a int.
long atol(const char *s); convierte a long int.

Funciones de conversión de número a arreglo de caracteres.

double strtod(const char*s, char **end p);
long strtol(const char *s, cgar **end p, int base);
unsigned long strtoul(const char *s, char ** end p, int base);
bsearch(const void *keyVal, const void *base);
qsort(void *bse, size_t n, size_t size, int (*cmp) (const void *, const void *));

Funciones que retornan direcciones de memoria.

```
#include<iostream>
using namespace std;
short & f (short * pS, short pos) {
   return pS[pos + 1];
} // f

main() {
   short vec[ ] = { 0,1,2,3,4,5,6,7,8,9};
   f(vec, 2) += 3;
   for (short i = 0; i < 10; i++)
      cout << vec[i] << ‘ ‘;
   return 0;
} // Emite: 0,1,2,6,4,5,6,7,8,9
```

Unidad II Capítulo II

Unidad II – Capítulo II - El tipo struct

Definición y características del tipo struct

El tipo registro es una estructura de datos cuyas componentes se denominan **campos**, los cuales entre ellos pueden ser de diferentes tipos, por lo tanto, el tipo registro es una estructura **heterogénea**. Podemos referirnos a todo el conjunto de datos bajo un solo nombre, esto es, la variable de tipo registro, además podemos referirnos a solo una componente, usando la siguiente notación: *NombreRegistro.NombreCampo***,** en donde: *NombreRegistro* es una variable de tipo registro y *NombreCampo* es una de las componentes del registro. El tipo registro es una estructura de datos estática, debido a que el compilador asignará una cantidad de byte en la memoria, del tamaño de la longitud del registro, esto es la sumatoria de las longitudes de cada componente. Para definir un tipo registro se utiliza la palabra reservada **struct**. Los campos del registro se definen dentro de su ámbito, por lo cual, los nombres de los campos serán de uso global, es decir, en cualquier lugar del programa, siempre nos referiremos a un campo con los mismos nombres definidos. No así con respecto al nombre del registro que podremos dar distintos nombres según en el lugar del programa en que nos encontremos.

La asignación entre registros es válida siempre y cuando sean del mismo tipo, si regA y regB son de tipo Reg, con campos cmp1, cmp2, ..., cmpn, entonces:
regB ← regA, es válida, y es lo mismo que hacer:
regB.cmp1 ← regA.cmp1; regB.cmp2 ← regA.cmp2; ...; regB.cmpn ← regA.cmpn.

Cada campo de un tipo registro podrá ser de tipo simple o bien de tipo estructurado como es el tipo registro, en estos casos, los subcampos se definen bajo otro apartado struct y la notación será, *NomReg.NomCmp.NomSubCmp*. El caso típico es un campo fecha y los subcampos dia, mes y año.

Ejemplo de sub-struct

rAlumno.FecNac.aa, en donde: *rAlumno* es una variable de tipo RegAlu, *FecNac* es un campo de tipo Fecha y *aaaa* es un subcampo de tipo short. A continuación se presenta una vista gráfica de lo que podría ser un registro de una transacción o novedad de una venta de un vendedor:

sVen rVendedor			
ushort **codVen** 2 b.	ushort **cantVen** 2 b.	str20 **descripArt** 21 b.	float **preUni** 4 b.

T.R.: 29 bytes

Observar los rótulos:
sVen indica el tipo struct, acompañado por la variable rVendedor.
Para cada uno de los campos se indica el tipo del campos y su nombre.
El T.R. (Tamaño del Registro) siendo la suma de sus campos.

```
typedef unsigned short ushort;
typedef char str5[6];
struct sVen {
   ushort  codVen,
           cantVen;
```

```
        str5      codArt;
        float     preUni;
    };
```

Pasaje de parámetros de tipo registro

Las variables de tipo registro pueden ser pasadas como parámetros a los módulos por valor o por referencia, tanto el registro completo como una de sus componentes simples o estructuradas. A continuación se lista un programa que muestra el uso del tipo registro en Turbo Pascal y su equivalente en C++:

Codificación el tipo struct

```
Program EjemploRegistro; {Turbo Pascal}
Uses
  Crt;

Type
  str20 = string[20];
  Fecha = record
     aaaa    : word;
     mm,
     dd      : byte
  end;
  RegAlu = record
        NroLeg    : longint;
        ApeNom,
        Domic,
        Local       : str20;
        FecNac     : Fecha;
        EstCiv      : char;
        Trabaja     : boolean;
        NroDoc    : longint
  end;

procedure ProcAlum(var rAlu : RegAlu);
begin
  writeln(rAlu.NroLeg);
  writeln('Ing. Apellido y nombre: ');
  readln(rAlu.ApeNom)
end;

procedure EmiteAlum(ApeN : str20);
begin
  writeln('Apellido y Nombre: ',ApeN);
  delay(2000)
end;

  var
     rAlumno,
     rAlumno2 : RegAlu;
```

```
begin
  rAlumno.NroLeg := 123456;
  writeln(rAlumno.NroLeg);
  rAlumno.FecNac.mm :=  5;
  rAlumno2 := rAlumno;
  if rAlumno.NroLeg = 123456 then
    ProcAlum(rAlumno)  {Pasa todo el registro}
  Else
    EmiteAlum(rAlumno.ApeNom) {Pasa solo el campo ApeNom}
end.
```

```
// Programa en C++

#include <fstream>
using namespace std;

typedef char  str20[21];
typedef unsigned short word;
typedef unsigned char byte;
struct sFec {
     word aa;
     byte mm,
            dd;
}
struct sAlu {
        long NroLeg;
        str20 ApeNom,
                Domic,
                 Local;
       tsFec  FecNac;
       char   EstCiv;
        bool  Trabaja;
        long   NroDoc;
  };

void ProcAlum(sAlu &rAlu) {
    cout << rAlu.NroLeg;
    cout << “Ing. Apellido y nombre: “;
    cin >> rAlu.ApeNom;
}

void EmiteAlum(str20 ApeN) {
    cout << “Apellido y Nombre: ‘ << ApeN”;
    delay(2000);
}

main() {
     sAlu rAlumno,
             rAlumno2;
```

```
    rAlumno.NroLeg = 123456;
    cout << rAlumno.NroLeg << endl;
    rAlumno.FecNac.mm =  5;
    rAlumno2 = rAlumno;
    if (rAlumno.NroLeg == 123456)
        ProcAlum(rAlumno)  //Pasa todo el registro
    else
        EmiteAlum(rAlumno.ApeNom) //Pasa solo el campo ApeNom
}
```

El tipo de dato **struct** es el equivalente a Pascal la estructura de dato de tipo **record**.

El tipo struct de C/C++ permite que bajo un mismo nombre haga referencia a una colección de datos cuyas componente pueden ser de distinto tipo entre sí, por lo que decimos que el tipo struct es una estructura heterogénea a estos elementos los llamamos **campos** de la estructura y que son utilizados como variables y que no pueden ser reemplazados por alias en los módulos, lo que significa que se deben llamarse a estos campos tal cual fuera especificado en la propia estructura cuando se lo definió.

Podemos referirnos a toda la colección de datos con su nombre, es decir la variable de tipo struct, y además si queremos referirnos a un elemento de esta colección la notación a emplear es con el nombre de la variable del tipo estructura seguido de un punto y a continuación el nombre del campo. Si nomStruct es una variable de tipo struct y sus campos fueron nominados como cmp1, cmp2 y cmp3, entonces la notación para referirse a toda la estructura o a un elemento de ella, sería en estos casos la siguiente:

nomStruct
nomStruct.cmp1 o
nomStruct.cmp2 o
nomStruct.cmp3

El área de almacenamiento de una estructura ocupa bytes contiguos y el tamaño de la estructura está dada por la suma de los pesos de cada una de sus componentes en bytes, por ejemplo si el cmp1 es de tipo float, el cmp2 es de tipo int y el cmp3 es de tipo char, entonces el tamaño de la estructura nomStruct es de 4 b. + 4 b. + 1 b. = 9 bytes en una implementación en que el tamaño de un float fuese de 4 bytes y el de un int fuese de 4 bytes y el tipo char de 1 byte. Además una componente puede ser a su vez de tipo estructurado, en este caso de tipo struct. Por ejemplo, el campo fecha podría definirse de tipo struc tFecha, en donde, tFecha es una estructura de tipo struct con tres componente o campos, siendo estos, el año, mes y día.

El tratamiento de manejo de los campos de una estructura es semejante al tratamiento que le damos a las variables de tipo simple, vale decir, que podremos emitir el contenido de un campo de una estructura o asignar el valor de un campo a otra variable o de una variable a un campo o realizar una operación relacional con un campo, entre otras cosas.

También podremos pasar como parámetro a una función una estructura completa o bien uno de sus campos, y esta invocación podrá ser pasada por valor o por referencia.

Una de las ventajas que tiene el uso de estructuras está en el hecho de que simplifica el pasaje de parámetros a los módulos.

El tipo estructura es ampliamente utilizado en combinación con otras estructuras de datos que se verán más adelante, por citar ciertos casos, podemos indicar que el tipo struct es utilizado en combinación con archivos binarios, con arreglos y con estructuras dinámicas de datos cuyos elementos los denominamos nodos.

Podremos realizar asignaciones entre estructuras de datos siempre y cuando ambas estructuras sean del mismo tipo de dato.

Ejemplo

Definición de una estructura de tipo Ventas:

```
struct sVenta {
   int codVend;
   char codArt[20];
   int cant;
   float preUni;
} UnVendedor;
```

luego de esta definición podremos declarar variables del tipo sVenta. UnVendedor es una variable de tipo struct sVenta.

```
sVenta OtroVendedor;

UnVendedor.codVend = 1234;
strcpy(UnVendedor.codArt,"ABCDE");
UnVendedor.cant = 120;
UnVendedor.preUni = 1567.89;
OtroVendedor = UnVendedor;
OtroVendedor.cant = UnVendedor.cant + 3;
ObtenerDatos(&UnVendedor);   // versión C pasado como referencia, su cabecera debería indicar el pasaje de parámetros como puntero a la estructura de datos, (sVenta *Vend).
```

En cambio este último ejemplo si se utiliza en la versión C++ la invocación no debe indicarse el símbolo &, debería invocarse como ObtenerDatos(UnVendedor) y su cabecera de la función tener la siguiente forma la lista de parámetro: (sVenta &Vend).

MostrarCampo2(UnVendedor.codArt);

MostrarCampo3(UnVendedor.cant);

Ejemplo

```
#include <conio.h>
#include <fstream>
using namespace std;

typedef  unsigned int word;

struct sVenta {
  word codVend;
  char codArt[20];
  int cant;
  float preUni;
} UnVendedor;

void MostrarDatos(sVenta);
void ObtenerDatos(sVenta &);
void MostrarCmp2(char []);
void MostrarCmp3(int);

void main() {
  sVenta OtroVendedor;

  UnVendedor.codVend = 1234;
  strcpy(UnVendedor.codArt ,"ABCDE");
  UnVendedor.cant  = 120;
  UnVendedor.preUni = 1567.89;
  OtroVendedor = UnVendedor;
  OtroVendedor.cant = UnVendedor.cant + 3;
  MostrarDatos(UnVendedor);
  MostrarDatos(OtroVendedor);
  ObtenerDatos(UnVendedor);
  MostrarCmp2(UnVendedor.codArt);
  MostrarCmp3(UnVendedor.cant);
  getch();
}

void MostrarDatos(sVenta Vend) {
  clrscr();
  gotoxy(10,5);
  cout << Vend.codVend << " "
         << Vend.codArt << " "
         << Vend.cant << " "
        << Vend.preUni << endl;
  getch();
}

void ObtenerDatos(sVenta &Vend) {
  clrscr();
```

```
  gotoxy(10,5);
  cout << "Cod.Vend.: ";
  cin >> Vend.codVend;
  gotoxy(10,7);
  cout << "Cod.Art…: ";
  cin >> Vend.codArt;
  gotoxy(10,9);
  cout << "Cant….: ";
  cin >> Vend.cant;
  gotoxy(10,11);
  cout << "Pre.Uni.: ";
  cin >> Vend.preUni;
}
void MostrarCmp2(char cArt[]) {
  clrscr();
  gotoxy(10,5);
  cout << cArt;
  getch();
}
void MostrarCmp3(int cantidad) {
  clrscr();
  gotoxy(10,5);
  cout << cantidad;
  getch();
}
```

Ejemplo

```
struct tAlumno {
  char Nom[21];
  char Dir[31];
  unsigned long NroMat;
  unsigned long Telefono;
};
float notas[10];
tAlumno UnAlumno, Curso[100];

Curso[i].Nom; // es el alumno de la posición i, su nombre.
tAlumno *pAlum;

pAlum = &UnAlumno;
```

Herencia por agragación o composición

```
struct tsAlum {
   int    a;
   char b[20];
};

struct tsAlum2 {
   tsAlum x;
   int        c;
} P3;
```

Herencia por extensión

```
struct tsAlu {
   int   a;
   char b[20];
};

struct tsAlu2 : tsAlu {
       int c;
} P4;
```

```
  tsAlum rAlum;
  tsAlum2 rAlum2;
  tsAlu rAlu;
  tsAlu2 rAlu2;

  rAlum.a = 23;
  strcpy(rAlum.b,"Jose");
  rAlum2.x.a = 32;
  strcpy(rAlum2.x.b,rAlum.b);
  strcat(rAlum2.x.b," Maria");
  cout << rAlum.a << ' ' << rAlum.b << ' ' << rAlum2.x.a << ' ' << rAlum2.x.b << endl;
```

Emite: 23 Jose 32 Jose Maria

Operador -> (se lo puede leer como operador flecha)

```
pAlum->Telefono; // Operador flecha.
(*pAlum).Telefono;
```

Se permiten los modos *auto*, *extern* y *static*.

Inicializar una estructura

```
struct tAlum {
   char apeNom[21],
          domic[21];
   long nro1,
         nro2;
   short notas[10];
} UnAlum = {"Juan Perez","Cordoba 1234",643219,48025};
         // Al no asignarse valores para notas estos se inicializan con valor cero.
```

Tipo de dato ordinal definido por el usuario

Por medio de la palabra reservada *enum* se pueden declarar tipos de datos definidos por el usuario. Su sintaxis es:

enum IdTipo {lista de constantes con nombre};

en donde:

IdTipo es un identificador válido definido por el usuario.

Lista de constantes con nombre, son nombres de identificadores válidos definidos por el usuario. Cada elemento de la lista tiene una posición ordinal, para el primer elemento le corresponde la posición ordinal 0, al segundo elemento, la posici{on ordinal 1, y así sucesivamente, no obstante, existe la posibilidad de romper esta secuencia igualando un elemento a un valor entero mayor al anterior, y los próximos elementos tendrán posición ordinal consecutivo al valor indicado expresamente, hasta tanto no se vuelva a modificar otra constante con nombre con otros valores. El siguiente ejemplo tratará de aclarar lo dicho anteriormente.

Ejemplo de tipo por enumeración

```
enum eColores {rojo, amarillo,azul,naranja=10,violeta,verde,marron=20,blanco,negro};
```

Las posiciones ordinales de las constantes con nombre indicadas en enum son las siguientes: {0,1,2,10,11,12,13,20,21,22}.

Las restricciones que poseen estos tipos de datos son básicamente dos:

No pueden ser asignados por asignaciones externas de entrada, esto es no pueden ingresarse con un scanf, o cin, por ejemplo.

No puden ser asignados por asignaciones externas de salida, esto es no pueden emitirse con un printf, o cout, por ejemplo; aunque en estos casos, en vez de emitir el nombre de un color, emite la posición ordinal, p.e.: cout >> rojo; emite 0.

Declaración de variables de tipo enum

```
eColores UnColor, OtroColor = verde;

  UnColor = rojo;
  if (UnColor == OtroColor)
      cout << “Son iguales ambos colores”;
  else
      cout << “Los colores son distintos”;
  switch(UnColor) {
     case rojo:case azul:case amarillo: cout << “El color es Primario”; break;
     case naranja:case violeta:case verde: cout << “El color es Secundario”; break;
     case marron: cout << “El Color es Terciario”; break;
     default: cout << “Es un Tono”;
}
```

Observar que los nombres de constantes por enumeración no van encerradas entre apóstrofos, si lo estuvieran no serían constantes por enumeración, sino, constantes de cadena y no es el caso que lo fueran.

Para ingresar datos desde el exterior se realiza el siguiente artilugio, se define una variable de tipo entero, en el ejemplo MiColor, luego:

```
int MiColor;
cout << “Ing. un valor entero 0:rojo, 1:amarillo, 2:azul, 10:naranja, 11:violeta,
12:verde, 20:marron,21:blanco,22:negro : “;
cin >> MiColor;
switch(MiColor) {
    case 0: UnColor = rojo; break;
    case 1: UnColor = amarillo; break;
    case 2: UnColor = azul; break;
    case 10: UnColor = naranja; break;
    case 11: UnColor = violeta; break;
    case 12: UnColor = verde; break;
    case 20: UnColor = marron; break;
    case 21: UnColor = blanco; break;
    case 22: UnColor = negro;
}
if (UnColor == rojo)
    cout << "si es rojo" << endl;
switch(UnColor) {
    case 0: cout << "rojo"; break;
    case 1: cout <<  "amarillo"; break;
    case 2: cout <<  "azul"; break;
    case 10: cout <<  "naranja"; break;
    case 11: cout << "violeta"; break;
    case 12: cout << "verde"; break;
    case 20: cout << "marron"; break;
```

```
   case 21: cout << "blanco"; break;
   case 22: cout << "negro";
}
```

Codificación G1Ej24 Vendedores el tipo struct

```
/*
        Id.Programa: G1Ej24.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: julio-2013
        Comentario.: Datos ventas de vendedores agrupados por Cod.Ven.
            Emitir por cada vendedor c/u. de sus ventas y al
                                finalizar un vendedor el Importe Total vendido, y
                                por fin del proceso el Importe General vendido y el
                                Cod.Ven. que mas vendio en Importe.
*/

#include <iomanip>
#include <fstream>
using namespace std;

typedef char str5[6];
struct RegVen {
        unsigned codVen,
         cant;
        str5    codArt;
        float   preUni;
};

void EmiteInic(float &tGral, float &iMaxVta) {

        cout << "Ventas de vendedores agrupados por codVen" << endl;
        cout << "Totales por codVen y Total General + codVenMaxVta" << endl;
        tGral = iMaxVta = 0.0;
} // EmiteInic

void ObtDatos(RegVen &rVen) {

        cout << "Datos de Vendedor: ";
        cout << "Cod.Ven. FIN = 0: ";
        cin >> rVen.codVen;
        if(rVen.codVen) {
                cout << "Cantidad........: ";
```

```
                cin >> rVen.cant;
                cout << "Cod.Art.........: ";
                cin >> rVen.codArt;
                cout << "Precio Unitario.: ";
                cin >> rVen.preUni;
        }
} // ObtDatos

void EmiteInicCabVen(unsigned int cVen, unsigned int &cVenAnt, float &tVen) {

        cout << "Cod.Ven.: " << cVen << endl;
        cout << "   Cant.   Cod.Art.   Pre.Uni.    Tot.Item" << endl;
        tVen = 0.0;
        cVenAnt = cVen;
} // EmiteINicCabVen

void CalcDet(RegVen rVen,float &tItm, float &tVen) {

        tItm = rVen.cant * rVen.preUni;
        tVen += tItm;
} // CalcDet

void EmiteDet(RegVen rVen, float tItm) {

        cout << setw(4) << "" << setw(4) << rVen.cant
            << setw(6) << "" << setw(5) << rVen.codArt
            << setw(3) << "" << setw(8) << rVen.preUni
            << setw(3) << "" << setw(9) << tItm << endl;
} // EmitDet

void CalcVerifMaxVtaPieVen(float tVen, float &tGral, float &iVtaMax,
                    unsigned int &cVenMax, unsigned int cVenAnt) {

        tGral += tVen;
        if(tVen > iVtaMax) {
                iVtaMax = tVen;
                cVenMax = cVenAnt;
        }
} // CalcPieVen

void EmitePieVen(float tVen) {

        cout << "Total Vendedor:" << tVen << endl;
```

```
} // EmitePieVen

void EmitePieLst(int tGral, unsigned int cVenMax) {

        cout << "Total General: " << tGral << endl;
        cout << "Cod.Ven.Max.Vta.: " << cVenMax;
} // EmitePieLst

void main() {
        float    totVen,
         totGral,
                        impMaxVta,
                        totItem;
        unsigned codVenAnt,
         codVenMax;
        RegVen   rVendedor;

        cout << setprecision(2);
        cout << fixed;
        //o cout.setf(ios::fixed);
        cout.fill('*');
        //o cout << setfill('+');
        //cout.setf(ios::showpoint);
        EmiteInic(totGral,impMaxVta);
        ObtDatos(rVendedor);
        while(rVendedor.codVen) {
                EmiteInicCabVen(rVendedor.codVen,codVenAnt,totVen);
                while(rVendedor.codVen == codVenAnt) {
                        CalcDet(rVendedor,totItem,totVen);
                        EmiteDet(rVendedor,totItem);
                        ObtDatos(rVendedor);
                }

        CalcVerifMaxVtaPieVen(totVen,totGral,impMaxVta,codVenMax,codVenAnt);
                EmitePieVen(totVen);
        }
        EmitePieLst(totGral,codVenMax);
}
```

```
typedef unisgned short ushort;
struct sDatos {
   short i;
   ushort j;
};
```

```
union miUni {
   int a;
   sDatos rDatos;
   char cad[4];
};

miUni u,
        *pu;
```

Acceso a miembros:

Directo..: u.a;
Indirecto: pu-> a;

Uniones anónimas

```
union {
   int x;
   float y;
};

X = 537; y = 3.1415;
Cout << x << ‘ ‘ << y;
```

Sus miembros son accedidos directamente, sin usar los operadores de acceso.

```
typedef unsigned char uchar;
union tUnion {
   char car;
   bool signo;
   uchar b;
};

tUnion car = ‘A’;
cout << tUnion.sino << endl;
cout << boolalpha << tUnion.signo << endl;
cout << tUnion.car << endl;
```

Unidad II Capítulo III

Unidad II – Capítulo III - Lenguaje de programación

Los lenguajes de programación se clasifican en

Lenguaje máquina

Escritos a nivel de bits, hace que sea una tarea bastante tediosa al programador aunque directamente reconocible por la computadora –cpu-.

Lenguajes de bajo nivel

–*ensambladores*-: En lugar de escribir las ordenes impartidas a la máquina con ceros y unos, se utiliza un lenguaje simbólico, que establece una relación 1 a 1, por lo tanto es extraordinariamente rápida su ejecución, la desventaja es que estos lenguajes están muy comprometidos con la máquina en donde deban ser ejecutados, por lo tanto no son transportables prácticamente a otros equipos, esto hace entonces que cada hardware diferente cuente con un código especialmente escrito para cada equipo. Los lenguajes de bajo nivel, las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos, p.e.:

Código Máquina	**Código Mnemotécnico**
0010	ADD
0001	LOAD
0100	STG

Luego de escribir un programa en lenguaje assembler, se necesita de un **programa ensamblador** que lo traduzca a código máquina.

Lenguajes de medio nivel

Son lenguajes que se encuentran en un paso intermedio entre los lenguajes de bajo nivel y los lenguajes de alto nivel. Los lenguajes de bajo nivel se encuentran más cerca de la máquina que del programador, en cambio los lenguajes de alto nivel son más cercanos al programador que a la máquina. Los lenguajes de nivel medio por lo tanto están en un punto intermedio entre la máquina y el programador. El lenguaje C es considerado como un lenguaje de nivel medio, ya que posibilita escribir el código de diferentes maneras, el que se elija hará que esté más cerca de la máquina o del programador o inclusive en un punto medio, entre estos dos, p.e.: c = c + 1 o c += 1 o c++, son distintas maneras de hacer lo mismo, aunque el código máquina que genere no sea el mismo, esto determinará que algunas de estar órdenes al traducirse a código máquina sea más óptima que otras, es decir, se ejecutrá más rápidamente.

Lenguajes de alto nivel

Aquí encontramos muchos de los lenguajes de programación, p.e.: Pascal, Basic, Cobol, Fortran, Algol, Ada, Modula, etc. Esta categoría de lenguajes hace que se encuentren más cercanos al programador que de la máquina. Esto significa que al alejarse más de la máquina requerirá de mayor tiempo de ejecución, pero serán más fáciles de escribir por utilizar palabras similares al lenguaje natural. Los programas escritos en un lenguaje de alto nivel son transportables, es decir, se pueden ejecutar en distintas máquinas, ya que no son tan dependientes del hardware. El código escrito debe ser traducido al lenguaje de máquina y para llevar a cabo esta tarea se utilizan los compiladores o bien los intérpretes.

Etapas para la generación de código máquina

1. **Edición** del programa, utilizando la **consola** –*teclado, pantalla*-, a través de un editor de texto o que genere código ASCII puro, una vez finalizado se graba en disco con un nombre elegido por el programador y de una extensión que dependerá del lenguaje que hayamos seleccionado para escribir nuestro programa, p.e.: Pascal **.Pas**, Basic .**Bas**, C **.c**, Cobol **.Cob** o **.Cbl**, dBase o Clipper o Fox .**prg**, etc.. Este programa recibe el nombre de **programa fuente** y es muy importante conservarlo, debido a que en caso de necesitar realizar modificaciones, se realizará con el código fuente.
2. **Compilar** el programa fuente, -etapa anterior-, utilizando el compilador correspondiente al lenguaje empleado, por lo tanto, el compilador tiene una entrada que es el lenguaje fuente, p.e.:

   Tp MiProg Tp, Compilador Turbo Pascal de Borland
   Tc MiProg Tc, Compilador Turbo C de Borland
   Cobol MiProg Cobol, Compilador Cobol.

   El archivo generado recibe el nombre de **programa objeto** y tiene como extensión **.obj**, -object-. Este programa contiene la traducción del programa fuente, es decir, su código es el de máquina, pero aún no podrá ser ejecutado debido a que habrá que enlazarle las librerías a que hace referencia.
3. **Vincular** el programa objeto –resultado de la etapa de compilación-, utilizando la aplicación que contiene las librerías que requiera el programa objeto, generalmente con el nombre de Link o similar, p.e.:

   Link MiProg
   Blink MiProg
   Tlink MiProg

Generalmente podrán incorporarse cuantos módulos **.obj** necesitemos para general un solo programa que ahora sí contendrán los códigos de las librerías que requería el .obj. El resultado del enlzador es un **programa ejecutable** cuya extención es **.exe**, p.e.: MiProg.exe, y es este programa autosuficiente de ser ejecutado.

Estructura de un programa en C/C++

Un programa en C/C++ consta de las siguientes partes:
Un área de inclusión de **librerías**, la cual presenta la siguiente forma:

#include "nomLib.h"
#include <nomLib.h>

Para cada librería requerida en un programa deberá incluir una de la forma indicada anteriormente.

Si el argumento está entre comillas, se sigue la ruta indicada, vale decir, busca la librería indicada en la ruta establecida. En cambio, si el argumento de include está entre signo de menor y mayor solo buscará la librería indicada en el directorio por defecto u omisión.

Ejemplos de include

```
#include <stdio.h> o #include<cstdio>
#include <conio.h>
#include <math.h> o #include <cmath>
```

```
#include <stdlib.h> o #include<cstdlib>
#include <string.h> o #include<cstring>
#include <alloc.h>
#include<iostream>
#include<iomanip>
#include <ctype.h> o #include <cctype>
#include <dos.h>
#include "c:\Lenguajes\cpp\lib\miLib.h"
#include <fstream> // C++
using namespace std; // C++
```

La librería **stdio.h** incluye un conjunto de funciones relacionadas con la entrada (i) y salida (o) estándar (std), entre las que podemos mencionar: fclose, fcloseall, feof, fflush, fopen, fprintf, fread, fscanf, fseek, fsetpos, ftell, fwrite, remove, rename, rewind, entre otras.

La librería **conio.h** propia del compilador de Borland C/C++, incluye un conjunto de funciones relacionadas con la consola (con) periféricos formado por la pantalla y el teclado (i) entrada por teclado, y salida (o) por pantalla. Algunas de las funciones perteneciente a esta librería son: *clreol, clrscr, getch, gotoxy, putch, textbackground, textcolor, wherex, wherey, window*.

La librería **math.h** incluye un conjunto de funciones relacionadas con la matemática (math) entre las cuales podemos encontrar: abs, acos, asin, atan, ceil, cos, exp, floor, fmod, log, log10, pow, pow10, sin, sqrt, tan.

La librería **stdlib.h** incluye un conjunto de funciones relacionadas con la librería (lib) estándar (std) entre las cuales podemos indicar: atof, atoi, atol, div, free, itoa, itoa, malloc, max, min, qsort, rand, random, randomize, strtod, strtol, strtoul.

La librería **string.h** incluye un conjunto de funciones relacionadas con la librería (string) de cadenas de caracteres entre las cuales podemos indicar: strcat, strchr, strcmp, strcpy, strlen, strlwr, strstr, strupr, entre otras.

La librería **alloc.h** incluye un conjunto de funciones relacionadas con la memoria dinámica entre las cuales podemos indicar: free, malloc y la constante NULL y el tipo size_t.

La librería **ctype.h** incluye un conjunto de funciones relacionadas con los caracteres entre las cuales podemos indicar: isalnum, isalpha, isascii, isdigit, islower, isspace, isupper, tolower, toupper, entre otras.

La librería **dos.h** incluye un conjunto de funciones relacionadas con los servicios del sistema operativo d.o.s. (hoy en desuso) entre las cuales podemos indicar: getdate, gettime, nosound, setdate, settime, sound entre otras.

La librería **iostream** incluye un conjunto de funciones relacionadas con mejoras a ciertas funciones de entradas y/o salidas entre las cuales podemos indicar: cin, entrada estandar a través del teclado, (se usa en C++ en lugar de scanf), cout, salida estandar a través de la pantalla, (se usa en C++ en lugar de printf), cerr, salida estandar de error sin buffer, escribe directamente al dispositivo de salida, clog, salida estandar con buffer.

Solo se han indicado algumas de las librerías brindadas por el lenguaje C/C++, hemos indicado las de uso más habitual en un programa C/C++.

Continuando a esta lista de librerías cuyo código se incorporará a nuestro programa en elaboración, podremos hacer uso de **definiciones** como ser de constantes con nombres o macros.

#define NOM_CTE valor
#define NOM_MACRO especificación

Ejemplos de define

```
#define FALSO 0
#define IVA 0.21
#define CENTINELA "***"
#define maxi(A,B) ((A) > (B) ? (A) : (B))
#define BELL '\007'
#define CAMPANA 'x7'
#define MAX_ELEM 100
#define record struct
```

Prosiguiendo a las definiciones indicaremos si fuera necesario (generalmente sí lo es) declaramos los prototipos de funciones, o indicaciones al preprocesador de C/C++; vale decir, aquellas funciones definidas por el usuario deberán indicarse como prototipo al pre-procesador, esto es, indicar tipo de valor devuelto por la función, nombre de la función y encerrado entre paréntesis cada uno de los parámetros formales cada uno precedido por el tipo de valor y como es el pasaje de parámetro, en la versión C como puntero (*) en C++ además podremos indicarlo con el símbolo &. Por lo que declarar el prototipo informe al pre-procesador es indicar solo su interfaz, su cabecera de la función y nada más que eso, para cada una de las funciones implementadas por el programa en curso.

En la sección **const** definimos constantes con nombres y es de uso global en el programa. En vez de utilizar valores constantes en nuestro programa, en ciertas situaciones es conveniente utilizar este tipo de objetos, que por diversas razones lo hace más legible y dinámico a la hora de modificar el programa, p.e. en vez de utilizar la constante 21 del IVA, podríamos definir una constante con nombre denominada IVA y hacerla igual al valor 21, luego en los distintos puntos del programa en donde dicho valor podría aparecer, pondríamos este identificador. Si a futuro debemos realizar un cambio de valor solo iremos al único lugar en donde podemos modificarlo. Por conveniencia utilizamos nombres escrito todo en mayúscula a efectos de darnos cuenta en el momento de leer un listado fuente de que esos identificadores son objetos constantes y no variables. El único lugar en donde podemos cambiar el valor de una constante con nombre es en la sección **const**. La sintaxis es:

const
IdConst = expresión;

Ejemplos de const

const

```
IVA = 21;
MIN = 12;
MAX = MIN * 2 – 1;
VAL_ABS = abs(-MAX);
COC = MAX div MIN;
RES = MAX mod MIN;
CAD  = “esto es una cadena de caracteres”;
LONG = length(cad);
C = #67;
VERDADERO = true;
ENT = trunc(23.74);
CODASCCI = ord(‘A’);
```

En la sección **typedef** definimos alias a los tipos de datos primitivos o estructurados como una forma de simplificar un nombre compuesto por uno más simple. La sintaxis es:

typedef tipoDato IdAlias

tipoDato es el tipo de dato simple o estructurado que queremos simplificarlo por un alias.
IdAlias es un identificador válido.

Ejemplos de typedef

```
typedef unsigned short ushort;
typedef char str20[21];
typedef struct tNodo *tLista;
typedef int mat[5][7];
```

Una alternativa al typedef es utilizar using, el cual existe en últimas versiones de C++, cuya sintaxis es:

using nvoNomTipo = nomTipoExistente;

Ejemplos de using

```
using car = char;
using word = unsigned int;
using pChar = char *;
using field = char [50];
```

El lenguaje C++ es **fuertemente tipado** -*strongly typed*-, esto quiere decir que todas las variables deben estar previamente definidas correspondiendo a un tipo de dato al que pertenecen. Otra clase de lenguajes que verifican el tipo de variables según el valor que se le asigne por primera vez se denominan **auto tipados** –*self typed*- mientras que otros

lenguajes podrían permitir que una variable tome valores de distinto tipo durante la ejecución del programa, en este caso se denomina **dinámicamente tipados** – *dinamically typed*-.

Definir a una variable como perteneciente a un tipo de dato, circunscribe al conjunto de valores que puede tener, el espacio de memoria asignado al tipo de dato establecido y las operaciones que pueden realizarse.

Luego de estas definiciones podremos declarar variables globales, las cuales tienen un alcance global y su espacio reservado se lo define en tiempo de compilación, son variables estáticas y su espacio de almacenamiento se reserva hasta la finalización del programa en curso.

La forma de declarar variables es la siguiente:

tipo nomVar, …;

tipo indica algunos de los tipos básicos brindados por el lenguaje o incluso tipos definidos por el usuario, una forma de simplificar nombres estructurados.

Estos tipos definidos por el lenguaje, a los que llamamos **tipos primitivos** de datos simples, los clasificamos en numéricos, enteros con o sin signo, reales, caracteres, booleanos en el caso de C++.

Ejemplos

```
int a,b;
long c;
float d,e;
double f;
char g,h;
bool i;
char *pchar;
int vec[10];
```

Además en la misma declaración podremos inicializar variables.

```
int m = 0, t = 1, *punt = NULL;
short vEnt[] = {35,12,89,23,15,142,45}; // vEnt[ ] es equiv. y preferible a vEnt[7].
```

Cabe mencionar que hay otros tipos de variables declaradas y definidas dentro de las funciones o módulos indicadas como **static**, estas variables poseen la característica de tomar un primer valor al invocar al módulo la primera vez, y dentro del módulo este valor inicial es modificado por el proceso, saliendo del módulo con este último valor almacenado en la variable declarada como static dentro del módulo. Luego, al invocar nuevamente a este módulo, la variable que hemos mencionado anteriormente sigue manteniendo el valor al momento de haber abandonado el módulo en una etapa previa, vale decir, no se reinicializa, sino, por el contrario mantuvo su valor anterior. Para que pueda mantener su valor esta variable local al módulo no se almacena en el stack sino en una región de memoria que permita mantener sus valores por cada invocación al módulo, como lo es el área de datos pero separadas de las variables globales.

Ejemplo

```
tipo nomFunc(tipoParam nomParam) {
   static int x = 0;

   x++;
   return x;
}
```

Al invocar por primera vez a nomFunc, se inicializa la variable static x con el valor cero, y antes de abandonar el módulo x se incrementa en uno. Luego en las siguientes invocaciones a este módulo no vuelve a reinicializar su valor en cero, sino, que mantiene el valor de las llamadas anteriores, incrementándose en uno más.

Finalmente y luego de estas definiciones y declaraciones de objetos diversos (include, define, prototipos de funciones, variables globales, constantes e identificadores de tipos definidos por el usuario), implementaremos nuestra función principal –main- función especial cuyo nombre es brindado por el lenguaje, es por donde comenzará a ejecutarse nuestro programa. Generalmente esta función establecerá invocaciones a otros módulos –funciones-, finalizando el programa al encontrar una sentencia **return** o bien cuando se alcance el final del bloque de la función main. Un bloque de sentencias se compone encerradas entre los símbolos { llave abre y } llave cierra.

La forma de la función principal es la siguiente:

```
tipo main(lista parámetros desde el exterior) {
Declaraciones de variables locales a la función principal

   Sentencia1;
   Sentencia2;
   .
   .
   .
   sentenciaN;
}
```

En donde algunas de las sentencias indicadas puede tomar la forma siguiente:

return valor;

tipo si se omite, por defecto se considera **int**.

Cuando la ejecución del programa ejecuta una sentencia return hará que finalice el programa si nos encontramos en la función main.

Si nos encontramos en otra función implementada por el usuario, entonces en estos casos, finaliza la ejecución del módulo, regresando al punto de la llamada a esa función.

El tipo de valor deberá corresponderse con el tipo de valor retornado por la función.

Si una función no requiere retornar valor, entonces, el tipo se lo indica como **void**.

En C++ se pueden declarar nuevas variables intercaladas con sentencias, es decir, rompe el esquema habitual de C, en donde, primero se declaran variables y luego se indican las sentencias.

En C/C++ la función main puede estar implementada en cualquier punto del código, es decir, puede escribirse primero esta función y luego otras funciones o, puede escribirse primero otras funciones y al final la función main o, incluso la función main puede escribirse luego de otras y antes de otras funciones. Por una cuestión de interés definiremos la función main como última, al igual que en el lenguaje Pascal el bloque principal que va al final del programa.

Comentarios internos en el programa

En C un comentario de varias líneas dirigido al usuario, y no al compilador, ya que este lo ignora, se lo escribe entre /* y */, p.e.:

```
/*
   este es un comentario típico en C
   de varias líneas.
*/
```

En C++ además de esta forma, existe la posibilidad de escribir comentarios de una línea precediendo con //, p.e.:

```
a = 5;  // asigna el valor 5 a la variable a.

// Comentario de una sola línea en C++ únicamente.
```

Para mayor información del Lenguaje C/C++ hacer clic en los siguientes enlaces:

en inglés
https://cplusplus.com/doc/tutorial/
en español
https://conclase.net/

Unidad II

Capítulo IV

Unidad II – Capítulo IV - Arreglos

Tipo de dato estructurado: arreglo

Los arreglos son estructuras de datos **homogéneas**, es decir, todas sus componentes son del mismo tipo. Como una estructura de datos estática su espacio debe definirse en tiempo de compilación.

¿Por qué utilizar este tipo de estructura de datos?. Supongamos la siguiente situación; se requiere tener en memoria interna y en forma **simultánea** los datos de los Apellidos, Nombres de 10 alumnos. Mantener todos estos datos al mismo tiempo en la memoria interna requerirá de 10 variables, pero cada una de ellas con un nombre diferente, p.e. ApeNom1, ApeNom2, ..., ApeNom10. Notamos que esta forma de notación haría muy compleja la tarea de programar, imaginemos ingresar estos datos, emitirlos, procesarlos y si en vez de 10 fueran 100 ó 1000 o más, **¡mucho más complicado!**. Entonces, bajo estas situaciones, cuando contemos con una colección de datos todas del mismo tipo emplearemos los arreglos.

Podemos decir, entonces que los arreglos son colecciones o disposiciones de datos homogéneas ubicadas en la memoria interna y de manera **continua** la cual se le asigna un **nombre único o genérico** que hace referencia a esa región de memoria reservada en tiempo de compilación. Ahora bien, para poder referirnos a un elemento o componente se utiliza un **subíndice**, siendo la notación formada por el nombre genérico seguido del subíndice encerrado entre corchetes. Por ejemplo, si tomamos el caso anterior, el nombre genérico podría ser *ApeNom* y el subíndice podría ser *i*; por lo tanto, para indicar un elemento del arreglo lo escribiríamos como se indica a continuación ***ApeNom[i]***. De esta manera al tener que emitir por ejemplo cada uno de los nombres de los alumnos, basta con variar el valor del subíndice para hacer referencia a cada elemento del arreglo.

Conclusión: al momento de utilizar arreglos, se deben tomar ciertas medidas preventivas. La memoria interna puede verse saturada, el subíndice puede tomar valores fuera del intervalo de valores definidos. Por estos motivos se recomienda realizar una estimación necesaria para la cantidad de memoria a utilizar y asegurarse los valores que adopten los índices, ya que, la responsabilidad será exclusivamente del programador.

int a[10]; // define un arreglo con 10 elementos, en donde, los valores de índice que permite señalar cada elemento del arreglo en forma individual, debe estar comprendido en [0, 9]. El tamaño del almacenamiento para la variable a es de 10 * sizeof(int) = 40 bytes. Todos estos 10 elementos se encuentran en **memoria contigua o consecutiva**.

a[0] se refiere a los bytes 0 a 3 de esta disposición de memoria.
a[1] se refieren a los bytes 4 a 7.
a[2] se refieren a los byte 8 a 11, y así sucesivamente.
Es válido hacer si p es puntero a entero, p = a, o p = &a[0]. En ambos casos p apunta al primer elemento del arreglo a, *p = a[0].
char cad[21] = “Mar del Plata”;
char *p;

dirección de cad es **00FA : 00A0** en donde **00FA** es el nro. de **segmento** y **00A0** es el nro. de **desplazamiento**.

p = cad; // La var. puntero p apunta a la dirección de cad, es decir, apunta a 00FA : 00A0.

p = p + 2; // Ahora p apunta 00FA : 00A2.
cout << *p << endl; // Emite: r.
p = &cad[2]; // Ahora p también apunta 00FA : 00A2.
por lo que *p = 'r'.
*p = cad[2] = 'r'.

char aCad[] = "Esto es una cadena de caracteres en un arreglo";

char *pCad = "Esto es una cadena de caracteres apuntada";

La diferencia entre estas dos declaraciones consiste en que en el primer caso podemos cambiar el contenido del arreglo, pero aCad siempre apuntará al primer byte de esta disposición, en cambio, en el segundo caso, sí podremos cambiar la dirección a la que apunta pCad para que apunte a otra posición en la memoria, con la posibilidad de que quede indefinida la constante de cadena si se modificara este valor. Como se mostrará más adelante es muy facil pasar de un modelo con arreglos a un modelo con apuntadores o viceversa. Un arreglo de apuntadores a cadenas, permite que cada una de esas cadenas pueda contener longitudes variables y a la hora de p.e. ordenar el arreglo de punteros a cadenas los que se ordenan no son las cadenas, sino, el cambio de las direcciones de los punteros, haciendo más óptima la solución algorítmica.

```
// Arreglos y Punteros existe una estrecha relación

#include <conio.h>
#include <iostream.h>

void main() {
  char vCar[] = "Este es un arreglo de caracteres";
  char *pCar = "Este otro es un puntero a caracteres";
  char *qCar;
  // Ambas declaraciones asignan al final del string el carácter nulo o NULL, indicado por, '/0'.
```

```
  clrscr();
  cout << vCar << endl; // Emite: Este es un arreglo de caracteres
  cout << *pCar << endl; // Emite: E
  *pCar = '*';
  cout << *pCar << endl; // Emite: *
  qCar = pCar;
  for (; *qCar != '\0'; qCar++)
    cout << *qCar; // Emite: *ste otro es un puntero a caracteres
  getch();
}
```

```
// Índice de un arreglo de tipo char se reemplaza a su código ASCII. Ej. 'A' es 65.

#include <stdio.h>
#include <conio.h>
#include <fstream>

main () {
  int a[10];

  for (char b = 'A'; b <= 'j'; b++) {
    a[b-'A'] = b;
    cout << (b-'A') << " " << (char) a[b-'A'] << " " << endl;
  }
  return 0;
}
```

```
int *p, v[10], m[3][5] = {{ 1,  2,  3,  4,  5},
                          { 6,  7,  8,  9,10},
                          {11,12,13,14,15}};
p = &v[0]; // p apunta a donde apunta el arreglo v.
cout << *(p + 2) << endl; // Emite: el contenido de v[2]
p = &m[0][0]; // p apunta a donde apunta el arreglo m
cout << *(p + 2) << endl; // Emite: el contenido de m[0][2]
cout << *(p + 4) << endl; // Emite: el contenido de m[0][4]
cout << *(p + 5) << endl; // Emite: el contenido de m[1][5]
cout << *(p + 12) << endl; // Emite: el contenido de m[2][2]
p = v; // o p = &v[0];
p++; Eqv. v + 1;
v[i]; Eqv. v + i;
```

Punteros con subíndice

p[3] = p[2] * 2.0; Eqv. v[3] = v[2] * 2.0;
*p; Eqv. v[0]; Eqv. v; Eqv. p[0];

```
int a[N], suma, i, *p;

for (i = 0, suma = 0; i < N; ++i)
  suma += a[i];
for (i = 0, suma = 0; i < N, ++i)
  suma += *(a + i);
for (i = 0, p = a, suma = 0; i < N; ++i)
  suma += p[i];
for (p = a, suma = 0; p < &a[N]; ++p)
  suma += *p;
```

Relaciones entre matriz y puntero

int m[5][3], *p;

m es un puntero al primer elemento de un vector de punteros m[], que contiene los primeros elementos de cada fila, por lo que m, es un puntero a puntero. El vector de punteros se crea en forma automática.

m = &m[0]; m[0] = &m[0][0];

m[1] = &m[1][0];

m[2] = &m[2][0];

la dirección base no es m, sino, &m[0][0]
m + i = m[i]

$$dir_{(i, j)} = dir_{(0, 0)} + i * MaxCol + j$$

p = m;
*p; // es el contenido de m[0]
*(p + 1); es el contenido de m[1]
**p es m[0][0]
**(p + 1) es el contenido de m[1][0]
((p + 1) + 1) es el contenido de m[1][1]

p = &m[0][0] es distinto de p = mat
acceder al elemento m[i][j]
*(p + MaxCol * i + j)
*(m[i] + j); // primer elemento fila i desplazamiento j elementos
(*(m + i)[j]
(((m + i)) + j)

Inicializar

```
float f[] = {1.2, 3.4, 5.1};
int i[100] = {0}; // Se inicializa todo el vector con valor cero
int m[3][2] = {{1,2},{3,4},{5,6}};
```

Arreglos multidimensionales

```
// Ejemplo El lenguaje de Programacion C K&R pag.: 122
// Arreglos multidimensionales
// Ejemplo datos:
// dayYear Año, mes, dia obtiene el nro.del dia acumulado
// monthDay Obtiene su inversa.

#include <fstream>
using namespace std;

int dayYear(char dayTab[][13], int year, int month, int day) {
  int leap;

  leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
  for (short i = 1; i < month; i++)
    day += dayTab[leap][i];
  return day;
} // DayYear

void monthDay(char dayTab[][13], int year, int yearDay, int &pMonth, int &pDay) {
  int i,
      leap;
```

```
  leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
  for (i = 1; yearDay > dayTab[leap][i]; i++)
    yearDay -= dayTab[leap][i];
  pMonth = i;
  pDay = yearDay;
} // monthDAy

main() {
  int year,
      mes,
      dia,
      dia_year;
  char dayTab[2][13] = {{0,31,28,31,30,31,30,31,31,30,31,30,31},
                        {0,31,29,31,30,31,30,31,31,30,31,30,31}};

  cout << "A" << char(164) << "o: ";
  cin >> year;
  cout << "Mes: ";
  cin >> mes;
  cout << "Dia: ";
  cin >> dia;

  dia_year = dayYear(dayTab,year,mes,dia);
  cout << "Dia del a" << char(164) << "o: " << dia_year << endl;
  monthDay(dayTab,year,dia_year,mes,dia);
  cout << "Mes: " << mes << endl;
  cout << "Dia: " << dia << endl;
  return 0;
}
```

Arreglos de apuntadores

```
// Ejemplo: El lenguaje de programación C de K&R pag.: 124
// Arreglo de apuntadores
// Un arreglo de apuntadores contiene direcciones de memoria (4 bytes)
// Los cuales apuntan a los datos alojados en otro lugar en la memoria interna.
// Ejemplo: vector de punteros a cadenas de nombres de los meses de un año.
// El vector solo contiene direcciones de memoria que apuntas a otros sitios
// en la RAM donde se encuentran c/u de los nombres de los meses.
// Estas posiciones pueden generarse en forma directa por el compilador
// o bien en forma explicita por el programador utilizando new.

#include <fstream>
using namespace std;

char *monthName(int n) {
  static char *name[] = {"Mes ilegal", "Enero", "Febrero", "Marzo", "Abril", "Mayo",
      "Junio", "julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"};
  return (n < 1 || n > 12) ? name[0] : name[n];
```

```
} // monthName

main() {
  int nroMes;

  cout << "Nro.Mes: ";
  cin >> nroMes;
  cout << monthName(nroMes) << endl;
  return 0;
}
```

Los arreglos son estructuras de datos interna dispuestos sus elementos en forma contigua, al igual que la estructura de dato **struct**, pero a diferencia de esta, los datos de un arreglo deben ser todos del mismo tipo, por lo cual decimos que es una estructura **homogénea**. El arreglo de C/C++ permite que bajo un mismo nombre haga referencia a una colección de datos cuyas componente como se dijo deben ser del mismo tipo entre sí, a estos elementos los llamamos **componentes o elementos o ítems** de la estructura. La gran ventaja que ofrecen los arreglos es que bajo un nombre genérico, nos referimos a dicha colección de datos, en lugar de asignar nombres diferentes a cada uno de estos valores individuales, cosa que haría bastante imposible referirnos en un proceso determinado, con nombres de variables distintas, p.e. si contamos con diez nombres de personas que debemos conocer y, si no contáramos que la estructura de tipo arreglo, deberíamos utilizar diez nombres de variables distintas para guardar en forma simultánea en la memoria interna estos datos, digamos, a1, a2, a3, … y así sucesivamente, pero si en lugar de diez, fueran 100 o 1000 o muchos más los datos a procesar simultáneamente, esto haría imposible su procesamiento. Sin embargo, con el uso de la estructura de tipo arreglo, estas dificultades se ven superadas al darle un nombre genérico a toda la colección de datos, p.e. ApeNom. Ahora bien, como referirnos a elementos individuales de esta colección, ya que, seguramente vamos a querer referirnos en ciertos momentos a elementos individuales de la colección y no a toda la colección de datos. La solución está brindada por la matemática misma al definir las **variables suscritas**, esto es, variables que van **acompañadas por un subíndice** el cual señala o apunta a un elemento o componente o ítem de la colección de datos. De tal manera, que la notación para referirnos a un elemento individual, se lo indicaría como $\mathbf{ApeNom_i}$, en donde, **i** indica el subíndice. El nombre del arreglo señala al primer byte de la colección de datos, por lo que el índice le sumará a la dirección base un desplazamiento, que observado desde el programador será el incremento en +1, pero que internamente se desplazará la cantidad de byte que pese una componente, p.e., si cada componente es de tipo long y el tamaño de una long es de 4 bytes, esto implica que el incremento del índice en uno, internamente será de 4 bytes el desplazamiento a la próxima componente.

Podemos decir, entonces que los arreglos son colecciones o disposiciones de datos homogéneas ubicadas en la memoria interna y de manera **continua** la cual se le asigna un **nombre único o genérico** que hace referencia a esa región de memoria reservada en

tiempo de compilación. Ahora bien, para poder referirnos a un elemento o componente se utiliza un **subíndice**, siendo la notación formada por el nombre genérico seguido del subíndice encerrado entre corchetes. Por ejemplo, si tomamos el caso anterior, el nombre genérico podría ser *ApeNom* y el subíndice podría ser *i*; por lo tanto, para indicar un elemento del arreglo lo escribiríamos como se indica a continuación ***ApeNom[i]***. De esta manera al tener que emitir por ejemplo cada uno de los nombres de los alumnos, basta con variar el valor del subíndice para hacer referencia a cada elemento del arreglo. Cada uno de los elementos del arreglo contiene una cadena de 20 caracteres, por lo que el tamaño debería ser de 21 bytes, en el cual el último carácter de la cadena no representaría un carácter más de la cadena sino, la finalización de la cadena, este carácter especial es **'\0'** y a este carácter se lo conoce como carácter nulo o NULL.

La declaración de este arreglo en C/C++ sería:

```
char ApeNom[10][21]; // El arreglo ApeNom, contendrá 10 cadenas, cada una de las
                     // cuales pueden contener hasta 20 caracteres más el carácter
                     // nulo.
```

Podríamos representar este concepto de manera gráfica en horizontal para ahorrar espacio, pero, también podríamos dibujarlo en forma vertical:

ApeNom[i] en donde 0 ≤ i ≤ 9 (dibujado en forma horizontal)

El gráfico está indicando varias cosas; el nombre genérico *ApeNom* , la cantidad de elementos 10, cantidad de dimensiones 1, tipo del índice valores enteros, intervalo de valores del índice *[0; 9]*, contenido de cada elemento de tipo cadena, por otro lado se nota que la variable *ApeNom* se encuentra ubicada en la dirección 1000 y que su tamaño es de *21 b. x 10 = 210 bytes*. El área de memoria reservada para la variable *ApeNom* va desde 1000 a 1000 + 210 – 1, es decir de 1000 a 1209.

Si queremos averiguar el tamaño **físico** de un arreglo, esto se puede determinar con la función *sizeof(obj)*, en donde *obj* indicará el tipo arreglo o la variable de ese tipo arreglo. La función retorna un longint indicando la cantidad de byte de ese objeto. Si queremos calcular el espacio ocupado por la variable *ApeNom*, lo estableceríamos de la siguiente manera, *sizeof(ApeNom)*, y si *ApeNom* fue definido de tipo *VecAlu* como un arreglo con 10 componentes cada una de las cuales sería una cadena de 20 caracteres, lo estableceríamos de la siguiente manera, *sizeof(VecAlu)*. Por lo tanto, *sizeof(ApeNom) = sizeof(VecAlu) = 210*.

Por otro lado, si lo que queremos es averiguar cuantas componentes tiene un arreglo, esto se establecería como el tamaño físico del arreglo dividido por el tamaño de una de sus componentes, es decir, *sizeof(ApeNom) / sizeof(ApeNom[1]) = 210 / 21 = 10*.

Si utilizamos un subíndice para que haga referencia a un elemento del arreglo, estará indicando una dirección de memoria dentro de ese intervalo, siempre y cuando el valor

del subíndice esté comprendido entre 0 y 9. Así, p.e. si *i = 0* la dirección del primer elemento del arreglo coincide con la dirección de la variable *ApeNom*, es decir, *d*ApeNom = *d*Apenom[*0*], en donde la letra *d* indica dirección, si *i = 1*, la dirección de inicio del segundo elemento es 1021, en cambio sí *i = 2*, la dirección de inicio del tercer elemento es 1042 y así sucesivamente. Por lo tanto, notamos que un incremento de 1 en el subíndice en realidad es un incremento en 21 bytes, es decir, un incremento de acuerdo al tamaño de cada componente del arreglo.

¿Qué sucederá si el subíndice adoptara un valor inferior o superior a 0 o 9 respectivamente? La respuesta es obvia, estaríamos invadiendo una zona de memoria desconocida, por lo tanto, los efectos que se podrían producir serían impredecibles, algunos de los cuales podrían ser, cambiar los valores a otras variables que estuvieran contiguas a esta región de memoria, que aparecieran caracteres extraños en la pantalla, que se colgara la ejecución del programa, entre otros. Será responsabilidad del programador llevar el control del valor del subíndice.

Un arreglo puede contener valores de tipo simple, como ser enteros con o sin signo, reales, carácter, booleana o incluso cadenas de caracteres y como veremos más adelante punteros o; de tipo estructurado, como ser struct o arreglo.

En cambio, el tipo de valor del subíndice solo deben ser valores de tipo ordinal, es decir, valores de tal manera que dado un valor x, se pueda determinar tanto el valor predecesor como el valor sucesor inmediato y, en esta categoría los tipos ordinales, pueden ser solamente los enteros con o sin signo, el tipo carácter como así también el tipo booleano y tipos de datos definidos por el usuario, denominados de tipo por enumeración.

El valor mínimo del subíndice es cero y el valor máximo queda establecido en la propia definición del arreglo indicado por un valor tope, no obstante, es este valor tope menos uno, p.e. si definimos un arreglo como **int** NroLeg[10]; para referirnos a toda la colección en forma completa basta con indicar el nombre genérico del arreglo, que en este caso es NroLeg, pero, para indicar un elemento de esta colección de datos la notación utilizada es NroLeg[i]; en donde, la variable i puede tomar valores entre cero y 9, o sea, NroLeg[0], NroLeg[1], NroLeg[2], …, NroLeg[9].

Es fácil notar que el manejo del subíndice puede estar controlado por el uso de una estructura cíclica, colaborando de manera simple la notación y paseándonos ágilmente por cada una de las componentes del arreglo.

La gran ventaja del uso de arreglos es que esta reside en la memoria interna por lo que el acceso a un elemento del arreglo se lo mide en nanosegundos y permite un recorrido con **acceso secuencial** o bien un **acceso al azar**, posibilitando acceder a cualquier elemento del arreglo sin tener que recorrer otros elementos que no nos interesan procesar. Los elementos de un arreglo son todos consecutivos, es decir, si contamos con un arreglo de valores enteros y si un entero pesa en memoria 2 bytes, entonces el primer elemento se ubica a partir de una posición base y su siguiente byte contiguo, el segundo elemento ocupan los dos bytes siguientes, el tercer elemento otros dos bytes siguientes y así sucesivamente. El tamaño total del arreglo queda determinado por la cantidad de elementos multiplicado por el peso de un elemento, p.e. int

NroLeg[10], el tamaño de memoria en cantidad de byte ocupada por el arreglo queda determinada por la siguiente expresión:

10 * 2 bytes = 20 bytes o de esta otra manera: 10 * sizeof(int).

En este último caso, si un int ocupa 2 bytes el tamaño total ocupado por el arreglo será de 20 bytes, pero, si un int ocupa 4 bytes, entonces, el tamaño total ocupado por el arreglo será de 40 bytes.

Los arreglos al igual que la struct pueden ser pasados a las funciones como parámetros, tanto el arreglo en forma completa, como en forma individual, como así también pueden ser pasados por valor o por referencia, aunque en este caso debemos aclarar que la forma natural es el envío de la dirección de memoria base del arreglo, es decir, la dirección de memoria del primer byte del arreglo, por lo que concluimos que los arreglos son pasados exclusivamente por referencia. Si no deseamos modificar el contenido del arreglo dentro del módulo, se puede utilizar la palabra reservada **const** el cual asegura no poder modificar el contenido del arreglo.

El tipo arreglo es ampliamente utilizado tanto en forma sencilla como en combinación con otras estructuras de datos como ser el tipo struct.

Un arreglo puede ser definido como de una dimensión o dos dimensiones o tres dimensiones o de n dimensiones. Los arreglos de una dimensión habitualmente se los denominan vectores o arreglos unidimensionales, los arreglos con dos dimensiones son matrices o bidimensionales, los de tres dimensiones son denominados tensores o tridimensionales y los de n dimensiones se los denominan en forma genérica como arreglos n dimensionales o multidimensionales.

En un arreglo bidimensional o matriz, la disposición de los elementos en la memoria interna se disponen estos elementos por filas, vale decir, las primeras posiciones las ocupan los elementos de la primer fila a continuación se disponen los elementos de la segunda fila, luego los elementos de la tercer fila y así sucesivamente, finalizando con los elementos de la última fila en esta linealización.

Una matriz de **m filas x n columnas**, siendo **m ≠ n**, se dice que es una matriz **rectangular**.

Una matriz de **m filas x n columnas**, siendo **m = n**, se dice que es una matriz **cuadrada**.

Los elementos de la matriz cuadrada con **i = j (I)**, se dicen que son elementos pertenecientes a la **diagonal principal**.

Los elementos de la matriz cuadrada con **i + j = n + 1**, y despejando j, queda: **j = n + 1 – i (II)**, se dicen que son elementos pertenecientes a la **diagonal secundaria**, en donde, **n** es la cantidad de filas o columnas.

Los elementos por arriba de la diagonal principal, esto es, con **i < j (III)**, se dicen que sus elementos pertenecen a la **triangular superior**.

Los elementos por debajo de la diagonal principal, eso es, con **i > j (IV)**, se dicen que sus elementos pertenecen a la **triangular inferior**.

Para linealizar una matriz por filas, en donde, i y j inician en la fila y columna cero, el valor m, indica la cantidad de filas y el valor n, la cantidad de columnas de la matriz, se aplica la siguiente expresión:

i * n + j (V)

con: (0 < m) ^ (0 < n)

Por ejemplo, si m = 4 y n = 5, encontrar la posición del elemento [2, 4], aplicando la expresión (V), reemplazando i por 2 y j por 4 obtenemos 14, por lo tanto, el elemento [2, 4] linealizado se localiza en la posición ordinal 14 a partir de la dirección inicial del arreglo en la memoria interna.

En el ejemplo que se detalla a continuación notamos que el elemento [2, 4] = 23 y en la linealización dada a continuación vemos que en la posición 14 el valor es 23.

Matriz

	0	**1**	**2**	**3**	**4**
0	32	18	6	45	23
1	35	3	8	22	1
2	56	12	76	7	23
3	19	83	2	15	4

Elementos de la matriz linealizada

32	18	6	45	23	35	3	8	22	1	56	12	76	7	23	19	83	2	15	4
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

Observación:

Dada la fila i = 1, y la columna j = 3, el valor o contenido en la Matriz $_{i,j}$ = Matriz 1, 3 = 22, aplicando la expresión en (V) reemplazando i por 1 y j por 3, tenemos que:

1 * 5 + 3 = 5 + 3 = 8, notamos que los elementos de la matriz linealizada ocupa la posición 8, corresponde al valor 22 en la Matriz $_{1,3}$.

Asignar valores ceros a los elementos de (I), diagonal principal**:**

```
for (i = 0; i < n; i++)
   Matriz[i] [i] = 0;
```

Asignar valores ceros a los elementos de (II), diagonal secundaria:

```
for i = 0; i < n; i++)
   Matriz[i][n + 1 – i] = 0;
```

Asignar unos a los elementos de (III), triangular superior:

```
for (i = 0; i < m; i++)
   for (j = i + 1; j < m; j++)
      Matriz[i] [j] = 1;
```

Asignar unos a los elementos de (IV), triangular inferior:

```
for (i = 1; i < m; i++)
   for (j = 0; j < i; j++)
      mat[i][j] = 1;
```

Pasaje de parámetros de estructuras de datos combinadas, arreglos-struct:

```
struct sFec {
   long aaaa;
   int mes,
       dia;
};
typedef char str21[21];
typedef double tvCtas[12];
struct sAlu {
   long nroLeg;
   str21 apeNom,
         domic,
         local;
   long  nroDoc;
   char  sexo;
   tFecha fecNac;
   unsigned int codPos;
   bool   trabaja;
   tvCtas vCuotas;
}; // T.R.: 4 + 63 + 4 + 1 + 8 + 2 + 1 + 48 = 131 bytes.
sAlu vrAlumnos[100];
```

Tabla invocaciones y cabecera con estructuras de datos combinadas

Invocación o Llamada	Cabecera del Módulo	#bytes
Mod1(vrAlumnos)	Mod1(tvrAlu vrAlu[])	4
Mod2(vrAlumnos[i])	Mod2(trAlu UnAlum)	131
Mod3(vrAlumnos[i].nroLeg)	Mod3(long nLeg)	4
Mod4(vrAlumnos[i].fecNac)	Mod4(tFecha fNac)	8
Mod5(vrAlumnos[i].fecNac.mes)	Mod5(int mes)	2
Mod6(vrAlumnos[i].vCuotas)	Mod6(tvCtas Ctas)	48
Mod7(vrAlumnos[i].vCuotas[j])	Mod7(double &UnaCta)	4
Mod8(vrAlumnos[i],vrAlumnos[i+1])	Mod8(trAlu &UnAlu,&OtroAlu)	8

Operaciones realizadas sobre arreglos de uso habitual son: inicializar un arreglo, recorrer secuencialmente o al azar un arreglo, ordenar y buscar elementos en un arreglo. Una matriz puede ser recorrida por filas o bien por columnas, ya sea de izquierda a derecha o de derecha a izquierda, como así también de arriba hacia abajo o de abajo hacia arriba, entendiéndose de arriba valores de subíndice menores y abajo valores de subíndice mayores, de igual manera de izquierda valores menores, y derecha valores de subíndice mayores.

Ejemplo recorrer matriz triangular superior

Realizar un proceso que recorra los elementos de la triangular superior y de la triangular inferior, de izquierda a derecha y, de arriba hacia abajo.

```
#include <stdio.h>
#include <conio.h>
#include <fstream>
#define CANT_FC 5
using namespace std;

main() {
   int mat[5][5] = {{0, 1, 2, 3, 4},
                    {1, 0, 5, 6, 7},
                    {2, 3, 0, 8, 9},
                    {4, 5, 6, 0, 10},
                    {7, 8, 9, 10, 0}
                   };
   clrscr();

  // Emitir elementos de la matriz, recorrido por fila de arriba hacia abajo y de izquierda
a derecha.

  for (int i = 0; i < CANT_FC; i++) {
      for (int j = 0; j < CANT_FC; j++)
         cout << mat[i][j] << ' ';
      cout << endl;
   }
   getch();

   // Recorrido de los elementos de la triangular superior.

   for (i = 0; i < CANT_FC; i++) {
      for (int j = i + 1; j < CANT_FC; j++)
         cout << mat[i][j]) << ' ';
      cout << endl;
   }
    getch();

    // Recorrido de los elementos de la triangular inferior.

    for (i = 0; i < CANT_FC; i++) {
       for (int j = 0; j < i; j++)
          cout << mat[i][j]) << ' ';
       cout << endl;
    }
    getch();
    return 0;
}
```

Ejemplo

Definición de un arreglo

```
int a[10], b[20]; // arreglo unidimensional o vector de elementos enteros.
float c[100];
long d[50];
unsigned char e[40];
char f[70];
char g[50][20]; // arreglo bidimensional de 50 filas por 20 columnas de elementos carácter.
double h[20][40]; // arreglo bidimensional de 20 filas por 40 columnas de elementos reales.
typedef unsigned int word;
struct fecha{
   word year;
   short mes,
         dia;
};
typedef float vCtas[12];
struct sAlu{
   long NroLeg ;
   char ApeNom[20];
   fecha FecNac;
   unsigned int CodPos;
   char EstCiv;
   vCtas vCuotas;
};
sAlu vrAlumnos[100]; // arreglo unidimensional de struct con 100 elementos.
int arreglo3[10][20][30]; // arreglo de tres dimensiones o tensor.
```

Los elementos de un arreglo pueden encontrarse en forma ordenada o desordenada. Buscar un elemento en un arreglo ordenado se lo busca en forma secuencial, salvo que contemos con un método de **hashing**. En cambio, un arreglo con elementos ordenados el método de búsqueda utilizado es el método conocido como **búsqueda binaria** o **búsqueda dicotómica**. Si un arreglo se encuentra desordenado, podemos realizar un proceso de ordenamiento, existiendo para estos casos diversos métodos el cual podemos clasificarlos en métodos cuadráticos o métodos logarítmicos. En el primer caso, **método cuadrático o directo**, decimos que son métodos sencillos de programar pero de ejecución más lenta, en cambio, los **métodos logarítmicos** son métodos más complejos que los anteriores, pero de ejecución más veloz y generalmente estos métodos utilizan una técnica de programación denominada métodos recursivos, en contraposición de los métodos iterativos. No obstante, podremos asegurar que con pocos datos, no existen diferencias en tiempo de procesamiento significativos en ambos métodos, por lo que, en

estos casos se aconseja el uso de algunos de los métodos cuadráticos como se verá a continuación por su sencillez del código.

Entre los métodos cuadráticos o directos, a su vez, existen diversas técnicas, entre las que encontramos, intercambio (Burbuja y Optimizado), inserción y selección.

Entre los métodos logarítmicos, a su vez, existen diversas técnicas, entre las que encontramos, shell, quick, heap.

Ejemplo ordenamiento por Intercambio Burbuja optimizada

```
// Ordenamiento directo o cuadrático por intercambio: Método de la Burbuja optimizada.
void IntCmb(int &elem1, int &elem2) {
int aux = elem1;

   elem1 = elem2;
   elem2 = aux;
} // IntCmb

void OrdBurOpt(int *v, int n) {
 bool ordenado;
 int li = 0;

   do {
      li++;
      ordenado = true;
      for (int i = 0; i < n - li; i++)
         if ( v[i] > v[i+1] ) { // compara los valores intercambia los datos
            IntCmb(v[i],v[i+1]);
            ordenado = false;
         }
   } while(!ordenado);
} // OrdBurOpt
```

Ejemplo ordenamiento por Selección

```
// Ordenamiento directo o cuadrático por el método de Selección:
```

```
void IntCmb(int &elem1, int &elem2) {
  int aux = elem1;

  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdSel( int *v, int n) {
    for (int i = 0, j = 0, k = 0; i < n - 1; i++) {
        k = i;
        for (j = i + 1; j < n; j++)
            if ( v[k] > v[j] )
                k = j;
       IntCmb(v[i], v[i + 1]);
    }
}  // OrdSel
```

Ejemplo ordenamiento por Inserción

```
// Ordenamiento directo o cuadrático por el método de Inserción:

void OrdIns( int *v, int n) {
    for (int i = 1, j = 0; i < n; i++) {
        int tem = v[i];
        j = i - 1;
        while ( j >= 0 && tem < v[j]) {
            v[j + 1] = v[j];
            j--;
         }
         v[j + 1] = tem;
    }
} // OrdIns
```

Ejemplo ordenamiento Shell

```
// Ordenamiento logarítmico, el método Shell

void OrdShell(int A[], int size) {
     int j, temp;
     int paso = size / 2;

    while (paso > 0) {
         for (int i = paso; i < size; i++) {
             j = i;
            temp = A[i];
           while ((j >= paso) && (A[j - paso] > temp)) {
```

```
            A[j] = A[j - paso];
            j = j - paso;
          }
         A[j] = temp;
       }
       paso /= 2;
    }
} // OrdShell
```

Ejemplo ordenamiento Quick

```
// Ordenamiento logarítmico por el método Quick

int Particiones(int vec[], int low, int high) {
   int prvotkey = vec[low];

   while (low < high) {
      while (low < high && vec[high] >= prvotkey)
         --high;
       swap(vec,high,low);
       while (low < high && vec[low] <= prvotkey)
          ++low;
       int dummy;
       dummy = vec[low];
        vec[low] = vec[high];
        vec[high] = dummy;
    }
    return low;
}  // Particiones

void qSort(int vec[],int low,int high) {
   int prvotloc;

   if (low < high) {
      prvotloc = Particiones(vec,low,high);
      qSort(vec,low,prvotloc);
      qSort(vec,prvotloc+1,high);
   }
} // qSort

void quickSort(int vec[], int n) {
   qsort(vec,0,n);
}  // quickSort
```

Enlace al siguiente sitio web, relacionado con métodos de ordenamientos internos y externos:

Método de Búsqueda Binaria o Dicotómica para arreglos ordenados

```
int BusBinVec(const int v[], int clave, int card){
  int ult = card - 1;
  int prim = 0;
  int med;

  while (prim <= ult){
     med = (prim + ult) / 2;
     if (v[med] == clave)
        return med;
     else
     if (v[med] < clave)
        pri = med + 1;
     else
        ult = med - 1;
  }
  return -1;
} // BusBinVec
```

Insertar elementos en forma ordenada, para evitar ordenarlos posteriormente

```
void InsertarEnOrdenVec(int v[], int elem, int card) {

  while (card > 0 && elem < v[card – 1]) {
     v[card] = v[card – 1];
     card--;
  }
  v[card] = elem;
}
```

Invocación

```
// Programa: InsertarOrdenado

#include <conio.h>
#include <iostream.h>
#define MAX_ELEM = 10

main() {
  int vec[MAX_ELEM],
       valor;

  for (int i = 0; i <= MAX_ELEM - 1; i++) {
```

```
    cout << "Ing. valor del elemento: ";
    cin >> valor;
    InsertarEnOrdenVec(vec,valor,i)
  }
  for (i = 0; i <= MAX_ELEM – 1; i++)
    cout <<  i << ": " << vec[i] << endl;
  getch();
  return 0;
}
```

A continuación presentamos diversas variantes para pasar como parámetro la variable *vApeNom*, por razones de conveniencia en la escritura, se pasará el arreglo por valor.

Ejemplo de invocación y cabecera de estructuras combinadas

sizeof(rAlu) = 145 y *sizeof(vApeNom) = 14500*

Invocación o Llamada	Cabecera del Módulo	Cant. bytes
Mod1(vApeNom)	Mod1(vrAlu vApeN)	14500
Mod2(vApeNom[i])	Mod2(sAlu rApeN)	145
Mod3(vApeNom[i].NroLeg)	Mod3(long nLeg)	4
Mod4(vApeNom[i].FecNac)	Mod4(rFecha fNac)	4
Mod5(vApeNom[i].FecNac.mm)	Mod5(short mes)	1
Mod6(vApeNom[i].vCuotas)	Mod6(vCuotas vCta)	72
Mod7(vApeNom[i].vCuotas[j])	Mod7(float Impo)	6
Mod8(vApeNom[i],vApeNom[i+1]	Mod8(sAlu rApeN1, sAlu rApeN2)	290

La asignación entre vectores NO es posible en C / C++
Por lo que, para copiar todo un arreglo en otro debería hacerse a través de un ciclo:

Operaciones con arreglos

Ordenamiento de arreglos

Como fue indicado en párrafos previos un arreglo podrá encontrarse ordenado o no. Si no se encuentra ordenado, a efectos de mejorar el rendimiento de un proceso o por motivos de mostrar los datos ordenados bajo un determinado criterio, hacen de la necesidad de ordenar los arreglos. Existen varios métodos para ordenar arreglos, pero básicamente podríamos dividirlos en 2 clases; los métodos cuadráticos o directos (N^2)

siendo la expresión n * (n – 1) / 2 y los métodos avanzados o logarítmicos o indirectos ($N \times log_2 N$).

En el primer grupo encontramos varios métodos, a saber; burbuja (bubble sort), selección, inserción. En el segundo grupo encontramos también varios métodos, a saber; shell, quick sort, ordenación por mezcla (*merge*).

Los métodos directos son fáciles de programar pero de ejecución más lenta. En cambio, los métodos indirectos son más complejos en su programación pero más eficientes para su ejecución ya que corren más rápidamente que los directos. No obstante, con pocos elementos en un arreglo, ambos métodos directos o indirectos funcionan muy parejos.

A continuación se presentará el método de ordenación por burbujeo optimizado. Este método compara elementos adyacentes, es decir, v[i] y v[i+1], cada vez que un elemento de una posición menor sea mayor al adyacente inmediato siguiente se debe realizar un intercambio, por lo tanto, al avanzar desde las posiciones inferiores hacia las posiciones superiores del arreglo, se irá arrastrando el valor mayor hasta alcanzar la última posición, y algunos de los valores menores se irán corriendo hacia las posiciones inferiores. La cantidad de pasadas estará en función de la cantidad de elementos de un arreglo. En el caso de un arreglo con n cantidad de elementos la cantidad de pasadas máximas será de n – 1. Pero este método por ser optimizado tiene la característica de que si en una pasada no se realizó ningún intercambio podemos asegurar que el arreglo ya quedó ordenado y por lo tanto abandonar las restantes pasadas, consiguiendo un ahorro en el tiempo de ejecución. Por cada pasada debemos recorrer cada elemento desde 1 hasta a n – k, en donde, k en la primer pasada vale 1, en la segunda pasada vale 2 y así sucesivamente. Al termino de cada pasada se habrá acomodado el elemento más pesado en la última posición de cada pasada. Seguidamente se presenta el algoritmo de ordenación por burbujeo optimizado.

Los métodos directos de ordenación se miden en cuanto a la cantidad de preguntas que deben de realizarse y está en función del valor N. En el caso de la ordenación por burbuja optimizado, notamos que en la primera pasada la cantidad de comparaciones es de $n - 1$, en la segunda pasada de n-2, n-3 en la tercera pasada, hasta llegar a 1 comparación en la última pasada, por lo tanto:

$$1 + 2 + 3 + ... + (n - 3) + (n - 2) + (n - 1) = n \cdot (n - 1) / 2 = (n^2 - n) / 2$$

El mejor de los casos es que el arreglo se encuentre ordenado, en esta situación solo hará falta una pasada, en cambio, el peor de los casos será que el arreglo se encuentre ordenado invertido, es decir, si queremos ordenarlo en forma ascendente y se encuentra ordenado en forma descendente.

Los métodos vistos en párrafos previos de búsqueda secuencial, búsqueda secuencial optimizada, búsqueda binaria y ordenación por burbujeo, han sido desarrollados para arreglos en la que las componentes son de un valor simple de dato. No obstante, estos mismos métodos con algunas ligeras modificaciones pueden ser aplicados para arreglos cuyas componentes sean de tipo estructurado, lo más común de tipo registro. Bajo esta situación al comparar un elemento con otro modificamos la notación *Arr[i]* por *Arr[i].cmp* en donde, *cmp* es el nombre de una de las componentes del registro, es decir, el nombre de un campo.

qsort Función de ordenamiento indirecto

El lenguaje C/C++ por medio de la librería **stdlib.h** ofrece la función **qsort** que como vimos anteriormente pertenece a la categoría de método indirecto, lo que lo hace un método muy rápido de ordenamiento interno, solo, debemos conocer su interfaz para poder invocarla, además ofrece la posibilidad de utilizarla para diferentes tipos de datos, es decir, es una función adaptable a la necesidad del momento, lo que significa que podemos contar con arreglos de distintos tipos de datos, como ser, short, int, bool, char, cadenas de caracteres, float, double o struct (registro), para efectuar las comparaciones entre dos elementos debemos aplicar una función particular, a la que llamaremos **función criterio**, luego más adelante detallaremos en forma más precisa y con diversos ejemplos de uso.

El prototipo de la función qsort presenta el siguiente formato:

void qsort(**void*** base, *size_t* num, *size_t* tam, **int** (*comp) (**const void***, **const void***))

Como podemos observar en el prototipo la función qsort tiene 4 parámetros formales, que pasaremos a describir:

base es el primer parámetro que representa puntero a un arreglo, el arreglo que debemos ordenar.

num es el segundo parámetro que representa el número de elementos del arreglo.

tam es el tercer parámetro que representa el tamaño en byte del tipo de cada elemento del arreglo.

El cuarto parámetro representa la **función criterio** cuyas partes indican:

int la función retorna un valor de tipo int.

*(*comp)* es un puntero a la función criterio, el nombre representado en forma general es comp, pero, este nombre puede ser reemplazado por cualquier otro nombre de identificador válido y más representativo. Esta función se deberá desarrollar como un módulo función por separado.

*(const void *, const void *)* representan los argumentos de la función criterio.

Por otro lado, en la invocación a la función qsort, el formato sintáctico debe ser:

qsort(arreglo,card,tamTipo,(**int**(*) (**const void** *, **const void** *)) (comp))

<u>en donde</u>:

arreglo representa el identificador variable del arreglo.
card representa la cantidad de elementos del arreglo.
tamTipo representa el tamaño de cada uno de los elementos del arreglo.
***(int(*) (const void** *, **const void** *))* debe ser escrito tal cual se lo indica.
(comp) representa la función que se utilizará para comparar dos elementos del arreglo. Este nombre puede ser cambiado por cualquier otro nombre válido según las reglas de los indentificadores para que sea más representativo.

Por último, la función criterio adoptará el siguiente formato sintáctico:

int comp(**const** tipo &elem1, **const** tipo &elem2)

<u>en donde</u>:

int es el tipo de la función que siempre será de este tipo.

comp es el nombre de la función, recordando como se comentara anteriormente que ese nombre puede ser cualquier otro.
tipo representa el tipo de cada elemento del arreglo, por ejemplo, short, int, float, char, bool, o tipo estructurado (registro), por ejemplo, sArt, o cualquier otro tipo de dato válido.

El cuerpo de la función criterio tendrá la siguiente forma sintáctica:

return elem1 – elem2;

En estos casos que el tipo de dato sea un tipo simple.
En caso que sea una cadena de caracteres debemos aplicar la función strcmp.

return strcmp(elem1,elem2);

En cambio si el tipo de dato fuera un struct, la forma sintáctica será:

return elem1.nomCmp – elem2.nomCmp;

A continuación se presentarán diversos ejemplos de uso, para su mayor comprensión del tema tratado. Sería conveniente comparar los distintos ejemplos entre si, para su mayor entendimiento.

Ejemplo con tipo int

```
/*
   Id.Programa: qsort tipo int.cpp
   Autor..........: Lic. Hugo Cuello
   Fecha..........: enero-2024
   Comentario.: Ordena asc. datos de tipo int
   Compilador.: Borland Vers. 5.5
                función qsort de la libreria stdlib
*/

#include <fstream>
using namespace std;

const
  CARD = 10;

int f_int (const int &i,  const int &j) {
  return i - j; // ord. asc. en cambio, return j - i; ord. desc.
} // comp

void Emite(int n[]) {
  for (short i = 0; i < CARD; i++)
    cout << n[i] << ' ';
  cout << endl;
} // Emite
```

```
main() {
  int num[CARD] = {315,259,812,127,402,706,387,102,321,543};

  Emite(num);
  qsort(num,10,sizeof (int), (int(*) ( const void *,  const void *)) (f_int));
  Emite(num);
  return 0;
}
```

Ejemplo con tipo float

```
/*
   Id.Programa: qsort de tipo float.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Ordena asc. datos de tipo float
   Compilador.: Borland Vers. 5.5
                función qsort de la libreria stdlib
*/

#include <fstream>
using namespace std;

const
  CARD = 10;

int f_float ( const float &i,  const float &j) {
  return i - j; // ord. asc. en cambio, return j - i; ord. desc.
} // comp

void Emite(float n[]) {
  for (short i = 0; i < CARD; i++)
    cout << n[i] << ' ';
  cout << endl;
} // Emite

main() {
  float num[CARD] = {31.5,2.59,81.2,12.7,4.02,70.6,38.7,1.02,32.1,54.3};

  Emite(num);
  qsort(num,10,sizeof (float), (int(*) ( const void *,  const void *)) (f_float));
  Emite(num);
  return 0;
}
```

Ejemplo con tipo cadena

```
/*
   Id.Programa: qsort con tipo cadena.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Ordena asc. datos de tipo cadena
```

```
  Compilador.: Borland Vers. 5.5
              función qsort de la libreria stdlib
*/

#include <fstream>
using namespace std;

const
  CARD = 10;

int f_str( const char * &i,  const char * &j) {
  return strcmp(i,j); // ord. asc. en cambio, return j - i; ord. desc.
} // comp

void Emite(char * n[]) {
  for (short i = 0; i < CARD; i++)
    cout << n[i] << ' ';
  cout << endl;
} // Emite

main() {
  char * num[CARD] = {"hola", "dia", "tarde", "verano", "invierno", "junio",
                      "diciembre", "primavera", "enero", "noche"};

  Emite(num);
  qsort(num,10,sizeof (char *), (int(*) ( const void *,  const void *)) (f_str));
  Emite(num);
  return 0;
}
```

Ejemplo con tipo struct campo int

```
/*
  Id.Programa: qsort registro campo int.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: enero-2024
  Comentario.: Ordena asc. datos en registro por campo int
  Compilador.: Borland Vers. 5.5
              función qsort de la libreria stdlib
*/

#include <fstream>
using namespace std;

const
  CARD = 10;

typedef char str20[21];

struct sArt {
```

```
  int     nLeg;
  str20 apeNom;
};

int f_cmp_int (const sArt &i, const sArt &j) {
  return i.nLeg - j.nLeg; // ord. asc. en cambio, return j - i; ord. desc.
} // comp

void Emite(sArt n[]) {
  for (short i = 0; i < CARD; i++)
    cout << n[i].nLeg << ' ' << n[i].apeNom << endl;
  cout << endl;
} // Emite

main() {
  sArt num[CARD] = {315,"HERNANDEZ",259,"ALVAEZ",812,"PEREZ",
                    127,"QUIROZ",402,"MARTINEZ",706,"LOPEZ",
                    387,"PETROCCELLI",102,"RIOS",
                    321,"CASTILLO",543,"BARRIOS"};

  Emite(num);
  qsort(num,10,sizeof (sArt), (int(*) (const void *,  const void *)) (f_cmp_int));
  Emite(num);
  return 0;
}
```

Ejemplo con tipo struct campo cadena

```
/*
   Id.Programa: qsort registro campo str20.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Ordena asc. datos en registro por campo cadena
   Compilador.: Borland Vers. 5.5
                función qsort de la libreria stdlib
*/

#include <fstream>
using namespace std;

const
  CARD = 10;

typedef char str20[21];

struct sArt {
  int   nLeg;
  str20 apeNom;
};

int f_cmp_str (const sArt &i,  const sArt &j) {
```

```
  return strcmp(i.apeNom,j.apeNom); // ord. asc. en cambio, return j - i; ord. desc.
} // comp

void Emite(sArt n[]) {
  for (short i = 0; i < CARD; i++)
    cout << n[i].nLeg << ' ' << n[i].apeNom << endl;
  cout << endl;
} // Emite

main() {
  sArt num[CARD] = {315,"HERNANDEZ",259,"ALVAREZ",812,"PEREZ",
                    127,"QUIROZ",402,"MARTINEZ",706,"LOPEZ",
                    387,"PETROCCELLI",102,"RIOS",
                    321,"CASTILLO",543,"BARRIOS"};

  Emite(num);
  qsort(num,10,sizeof (sArt), (int(*) (const void *,  const void *)) (f_cmp_str));
  Emite(num);
  return 0;
}
```

Búsqueda secuencial o lineal

La búsqueda secuencial en un arreglo consiste en buscar un valor en el arreglo recorriendo los elementos adyacentes, comenzando desde una posición inicial. El proceso finaliza cuando se encuentre una posición en el cual su valor es el buscado o bien cuando se haya alcanzado al último elemento y no apareció el valor buscado. Por lo tanto habrá dos situaciones de salida, una es cuando se encontró el valor y la otra cuando se alcanzó al último elemento en el arreglo sin haberse encontrado el valor. De esto se desprende que si el valor estuviera en la primera posición, un solo acceso al arreglo sería suficiente y en el peor de los casos tendremos que acceder hasta la última posición n, así, si un arreglo tuviera *n* componentes, la **cantidad de accesos promedio sería de *n / 2***.

A continuación se presenta el siguiente módulo en el cual se buscará el valor de una clave en un arreglo, retornando el módulo la posición encontrada si el valor clave se encontró en el arreglo, caso contrario un valor cero si no se encontró.

Puntero a función

Se da un ejemplo de puntero a función para ser aplicada a un método de ordenamiento el cual necesitamos en distintos puntos del algoritmo ordenar el vector por diferentes **criterios**.

En el ejemplo dado a continuación se aplica a la función *OrdxBur*.

Observar detenidamente la cabecera la fuente en color rojo y en el cuerpo de la función la condición en la selección también la fuente en color rojo.

```
void OrdxBur(tvrCam vrCam, short card, bool (*criterio) (void *, void *) ) {
  bool ordenado;
  short k = 0;

  do {
    ordenado = true;
    k++;

    for (short i = 1; i <= card - k; i++)
      if (criterio (&vrCam[i], &vrCam[i+1]) ) {
        ordenado = false;
        IntCmb(vrCam[i],vrCam[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur
```

La invocación a la función OrdxBur se destaca la fuente también en color rojo.

Invocación en la Instancia I

```
OrdxBur(vrCamioneros,canCam, (bool (*) (void *, void * )) (fcantUniRotas) );
Listado(vrCamioneros,canCam,"Cant.Unid.Rotas");
```

Invocación en la Instancia II

```
OrdxBur(vrCamioneros,canCam, (bool (*) (void *, void * )) (fnroLeg) );
Listado(vrCamioneros,canCam,"Nro.Legajo");
```

Invocación en la Instancia III

```
OrdxBur(vrCamioneros,canCam, (bool (*) (void *, void * )) (fcantViajes) );
Listado(vrCamioneros,canCam,"Cant.Viajes");
```

Las funciones que determinarán cada uno de los criteros de ordenamiento se presentan a continuación en color rojo el nombre de la función:

```
bool fcantUniRotas (trCam &elem1, trCam &elem2) {
  return elem1.CantUniRotas > elem2.CantUniRotas;
}
```

```
bool fnroLeg (trCam &elem1, trCam &elem2) {
  return strcmp(elem1.CodLeg,elem2.CodLeg) > 0;
}
```

```
bool fcantViajes (trCam &elem1, trCam &elem2) {
```

```
  return elem1.CantVjes > elem2.CantVjes;
}
```

El método de búsqueda secuencial sirve tanto si el arreglo está desordenado como si se encontrara ordenado. No obstante en los casos en que el arreglo estuviera ordenado debería optimizarse el algoritmo para los casos en que habiendo llegado a una posición en el arreglo y siendo que su contenido es mayor al valor buscado, bajo esta situación se debería abandonar la búsqueda, ya que si no se lo encontró hasta esa posición, tampoco se lo encontrará más adelante, por ser todos esos valores mayores al buscado, sabiendo que el arreglo se encontraba ordenado en forma ascendente. Este concepto va a ser muy importante cuando se analice en la Unidad V la estructura dinámica lineal Lista, ya que la única manera de acceder a sus nodos es en forma secuencial, y estando la Lista ordenada por el valor de la clave a buscar el recorrido finalizará cuando aparezca un valor mayor al que estamos buscando. En una lista enlazada NO podemos aplicar la búsqueda binaria, el cual aplica el direccionamiento directo a los nodos, por lo que la única manera es recorrerla en forma secuencial.

Ejemplo módulo de búsqueda secuencial optimizado

El módulo BusLinOpt mejora el rendimiento de la búsqueda con respecto al módulo BusLin, ya que en el segundo caso si se encontró un valor mayor al buscado se abandona la búsqueda de los siguientes elementos restantes.

Pero, **¿será este el mejor método de búsqueda cuando el arreglo se encuentre ordenado, por el valor de la clave a buscar?.** La respuesta es **¡NO!**. Una mejor manera de buscar un valor en un arreglo ordenado será aquel que obtiene un punto medio entre sus extremos y es comparado por el valor a buscar, si es, entonces se encontró y se abandona la búsqueda, en cambio si no lo es, caben dos posibilidades, que haya que seguir buscando en una primera mitad o bien en la segunda mitad, en ambos casos el arreglo queda reducido para continuar la búsqueda en la mitad. El método que realiza estas acciones se conoce como método de búsqueda binario o dicotómica. Este método ya fue presentado en el tema de buscar un elemento en un archivo ordenado. Por lo tanto, diremos que conservará básicamente la estructura lógica vista, la única diferencia notable está en que no hace falta bajar el dato ya que se encuentra en la propia memoria interna. A continuación se presenta el módulo de búsqueda binaria o dicotómica en arreglos.

Búsqueda Binaria o Dicotómica

El método comienza conociendo las posiciones extremas del arreglo, **pri** y **ult**, se obtiene el punto medio med y se compara si el valor de esta posición es el valor clave a buscar, si es entonces se abandona el proceso, sino caben 2 alternativas, es menor, entonces se modifica el extremo inferior pri por el de la posición **med + 1** caso contrario se modifica el extremo superior ult por el de la posición **med – 1**, luego de lo cual se obtiene una nueva posición med. Si después de reiteradas búsquedas no aparece el valor el algoritmo finaliza cuando el valor extremo inferior pri se vuelva mayor al valor del extremo superior ult.

Como vemos por cada comparación realizada se reduce a la mitad el conjunto de valores a consultar, por lo tanto, el análisis será de:

$$n / 2,\ n / 4,\ n / 8;\ \text{es decir},\ n / 2^1,\ n / 2^2,\ n / 2^3$$

El proceso finalizará cuando el tamaño se haga < 1. Por lo tanto, si k es el número mayor de comparaciones $n / 2^k < 1$ establecerá la finalización.

Si tomamos un conjunto de 50000 elementos, la cantidad máxima de comparaciones estará en el orden de $\log_2 N = \log_2 50000 \sim 16$. Por lo tanto, con k = 16 se abandonará la búsqueda si aún no se encontró el valor de la clave en el arreglo.

Ejemplo: Se tiene el siguiente arreglo Arr con los siguientes elementos:

	5	12	27	32	35	43	54	56	61	66	84	96	98	119	152
pos.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

y se requiere buscar el valor 35, los pasos a realizar establecerán los siguientes valores para los extremos y los puntos medios:

pri ← 1, ult ← 15, med ← 8, como el valor de la pos. 8 es 56 y no es igual a 35 la comparación siguiente 56 < 35 es falsa se cambia el extremo superior ult ← med – 1 y med ← 4, como el valor de la pos. 4 es 32 y no es igual a 35 la comparación siguiente 32 < 35 es verdadera se cambia el extremo inferior pri ← med + 1 y med ← 6, notamos que 43 < 35 es falso se modifica ult ← med – 1 y med ← 5, por último vemos que el valor de la pos. 5 35 es igual al valor a buscar que es 35, por lo tanto hemos encontrado el valor clave. Tambien notamos que los extremos en este último caso se hicieron iguales y justamente en esa posición se encontraba el valor a buscar, este sería la última comparación a realizar si el valor no se hubiera encontrado, debido a que en el próximo paso el extremo inferior se volvería mayor al extremo superior, indicando que no debemos seguir procesando la búsaqueda.

bsearch función primitiva de búsqueda binaria

El lenguaje C/C++ por medio de la librería **stdlib.h** ofrece la función **bsearch**, solo, debemos conocer su interfaz para poder invocarla, además ofrece la posibilidad de utilizarla para diferentes tipos de datos, es decir, es una función adaptable a la necesidad del momento, lo que significa que podemos contar con arreglos de distintos tipos de datos, como ser, short, int, bool, char, cadenas de caracteres, float, double o struct (registro), para efectuar las comparaciones entre dos elementos debemos aplicar una función particular, a la que llamaremos **función criterio**, luego más adelante detallaremos en forma más precisa y con diversos ejemplos de uso.

El prototipo de la función bsearch presenta el siguiente formato:

void * bsearch (**const void** * clave, **const void** * base, size_t num, size_t tam,
int (*comp) (**const void** *, **const void** *))

Como podemos observar en el prototipo la función bsearch tiene 5 parámetros formales, que pasaremos a describir:

clave es el primer parámetro que representa puntero al valor clave a buscar en el arreglo.

base es el segundo parámetro que representa puntero a un arreglo, el arreglo que debemos buscar el valor clave, **muy importante, es que el arreglo debe encontrarse ordenado en forma ascendente**.

num es el tercer parámetro que representa el número de elementos del arreglo.

tam es el cuarto parámetro que representa el tamaño en byte del tipo de cada elemento del arreglo.

El quinto parámetro representa la **función criterio** cuyas partes indican:

int la función retorna un valor de tipo int.

*(*comp)* es un puntero a la función criterio, el nombre representado en forma general es comp, pero, este nombre puede ser reemplazado por cualquier otro nombre de identificador válido y más representativo. Esta función se deberá desarrollar como un módulo función por separado.

*(const void *, const void *)* representan los argumentos de la función criterio.

Por otro lado, en la invocación a la función bsearch, el formato sintáctico puede ser, algunos de los casos indicados a continuación, entre otros, si la situación lo amerita:

```
p = bsearch(&clave, arreglo, card, tamTipo, comp);
```

En el caso que el dato a buscar sea una cadena el formato de la invocación presenta la siguiente forma, observar que el nombre de la función para aplicar es la función primitiva de cadenas strcmp, en este caso no debemos desarrolar dicha función.

```
p = bsearch(&clave,arreglo,card,tamTipo,(int(*)(const void*,const void*)) strcmp);
```

En el caso que el dato a buscar sea un registro el formato de la invocación presenta la siguiente forma:

```
p = (sArt*) bsearch (&clave, rArt, 10, sizeof (sArt),
                     (int(*) (const void *, const void *)) f_cmp_int);
```

<u>en donde</u>:

clave representa el valor a buscar en el arreglo.
arreglo representa el identificador variable del arreglo.
card representa la cantidad de elementos del arreglo.
tamTipo representa el tamaño de cada uno de los elementos del arreglo.
comp representa la función que se utilizará para buscar el valor clave en el arreglo. Este nombre puede ser cambiado por cualquier otro nombre válido según las reglas de los indentificadores para que sea más representativo.

Por último, la función criterio en su cabecera o prototipo, adoptará algunos de los siguientes formatos sintáctico:

```
int comp(const void * i, const void * j)
int comp( const tipo &elem1, const tipo &elem2)
```

<u>en donde</u>:

int es el tipo de la función que siempre será de este tipo.
comp es el nombre de la función, recordando como se comentara anteriormente que ese nombre puede ser cualquier otro.
tipo representa el tipo de cada elemento del arreglo, por ejemplo, short, int, float, char, bool, o tipo estructurado (registro), por ejemplo, sArt, o cualquier otro tipo de dato válido.

El cuerpo de la función criterio tendrá la siguiente forma sintáctica:

```
return elem1 – elem2;
return *(float *) elem1 - *(float  *) elem2;
```

En estos casos que el tipo de dato sea un tipo simple.
En caso que sea una cadena de caracteres debemos aplicar la función strcmp.

return strcmp(elem1,elem2);

En cambio si el tipo de dato fuera un struct, la forma sintáctica será:

return elem1.nomCmp – elem2.nomCmp;

Cabecera y cuerpo de una fución criterio de datos vector de registro con el campo a buscar de tipo cadena.

```
int f_busca_str (const str20 &i,  const sArt &j) {
  return strcmp(i,j.apellido); // busca siempre el valor de i en c/u. de los regs. de j
```

A continuación se presentarán para su mayor comprensión del tema tratado diversos ejemplos de uso de la función primitiva de búsqueda binaria **bsearch**, la cual requiere que el arreglo se encuentre previamente ordenado en forma ascendente. Sería conveniente comparar los distintos ejemplos entre si, para su mayor entendimiento

Ejemplo con tipo int

```
/*
   Id.Programa: bsearch int.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Busca el valor clave de tipo int en el arreglo ordenado asc.
   Compilador.: Borland Vers. 5.5
                función bsearch de la libreria stdlib
*/

#include <fstream>
#include <iomanip>
using namespace std;

const
  CARD = 10;

int f_int(const void *elem1, const void *elem2) {
  return *(int *) elem1 - *(int *) elem2;
} // f_int

void Emite(int n[]) {
  for (short i = 0; i < CARD; i++)
    cout << setw(2) << i << setw(4) << n[i] << endl;
  cout << endl;
} // Emite

main() {
  int   num[CARD] = {102,127,259,315,321,387,402,543,706,812};
  int * ptr,
        clave = 315;

  Emite(num);
```

```
  ptr = (int *) bsearch(&clave,num,CARD,sizeof (num[0]),f_int);
  if (ptr)
    cout << "El valor " << clave << " se encuentra en el arreglo en la posici"
         << char(162) << "n: " << ptr - &num[0] << endl;
  else
    cout << "El valor " << clave << " NO se encuentra en el arreglo." << endl;
  return 0;
}
```

Ejemplo con tipo float

```
/*
   Id.Programa: bsearch float.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Busca el valor clave de tipo float en el arreglo ordenado asc.
   Compilador.: Borland Vers. 5.5
                función bsearch de la libreria stdlib
*/

#include <fstream>
#include <iomanip>
using namespace std;

const
  CARD = 10;

int f_float(const void *elem1, const void *elem2) {
  return *(float *) elem1 - *(float  *) elem2;
} // f_float

void Emite(float n[]) {
  for (short i = 0; i < CARD; i++)
    cout << setw(2) << i << setw(8) << n[i] << ' ' << endl;
  cout << endl;
} // Emite

main() {
  float   num[CARD] = {2.59,3.87,7.06,8.12,10.2,12.7,31.5,32.1,40.2,54.3};
  float * ptr,
            clave = 8.12;

  Emite(num);
  ptr = (float *) bsearch(&clave,num,CARD,sizeof (num[0]),f_float);
  if (ptr)
    cout << "El valor " << clave << " se encuentra en el arreglo en la posici"
         << char(162) << "n: " << ptr - &num[0] << endl;
  else
    cout << "El valor " << clave << " NO se encuentra en el arreglo." << endl;
```

```
  return 0;
}
```

Ejemplo con tipo struct

```
/*
   Id.Programa: bsearch str.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Busca la clave de tipo cadena en el arreglo ordenado asc.
                Para que la función bsearch funcione correctamente, el arreglo
                debe encontrarse ordeado en forma ascendente, por lo que,
                antes de buscar la clave en el arreglo, previo es ordenado
                por medio la funcion qsort en forma ascendente.
--> Observacion: Existe una incompatibilidad de bsearch si se utiliza cout,
                el cual requiere de cierta libreria o std:: o using namespace,
                y cualquiera del uso de estos recursos, hará que genere un
                error en la compilacion. En su reemplazo se utilizara el
                comando de salida a la consola de printf.
   Compilador.: Borland Vers. 5.5
                función bsearch de la libreria stdlib
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const
  CARD        = 10,
  MAX_SIZE_STR = 20;

void Emite(char n[][MAX_SIZE_STR]) {
  for (short i = 0; i < CARD; i++)
    printf("%2d %s\n",i,n[i]);
  printf("\n");
} // Emite

main() {
  char lststr[][MAX_SIZE_STR] = {"sagitario","enero","viernes","geminis","invierno",
                                 "junio","noche","dia","primavera","jueves"};
  char * ptrItem;
  char   clave[MAX_SIZE_STR] = "geminis";

  Emite(lststr);
  qsort (lststr, CARD, MAX_SIZE_STR, (int(*)(const void*, const void*)) strcmp);
```

```
  printf("Buscar la cadena %s\n",clave);
  ptrItem = (char*) bsearch (clave, lststr, CARD, MAX_SIZE_STR,
                        (int(*)(const void*,  void*)) strcmp);
  Emite(lststr);
  if (ptrItem)
    printf("La cadena %s se encuentra en el arreglo en la posicion: %d\n",
          ptrItem,(ptrItem - lststr[0]) / MAX_SIZE_STR);
  else
    printf("La cadena %s NO se encuentra en el arreglo.\n",clave);
  return 0;
}
```

Ejemplo con tipo struct con campo int

```
/*
   Id.Programa: bsearch struct cmp int.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Busca la clave de tipo int en el arreglo ordenado asc.
              Para que la función bsearch funcione correctamente, el arreglo
              debe encontrarse ordenado en forma ascendente, por lo que,
              antes de buscar la clave en el arreglo, previamente es
              ordenado por medio la funcion qsort en forma ascendente.
              función bsearch de la libreria stdlib.
--> Observacion: Existe una incompatibilidad de bsearch si se utiliza cout,
              el cual requiere de cierta libreria o std:: o using namespace,
              y cualquiera del uso de estos recursos, hará que genere un
              error en la compilacion. En su reemplazo se utilizara el
              comando de salida a la consola printf.
   Compilador.: Borland Vers. 5.5
*/

#include <stdio>
#include <stdlib>
#include <string>

const
  CARD        = 10,
  MAX_SIZE_STR = 20;

typedef char str20[21];

struct sArt {
  int    nLeg;
```

```
  str20 apeNom;
};

int f_cmp_int (const sArt &i, const sArt &j) {
  return i.nLeg - j.nLeg;
} // comp

void Emite(sArt n[]) {
  for (short i = 0; i < CARD; i++)
    printf("%2d %4d %s\n",i,n[i].nLeg,n[i].apeNom);
  printf("\n");
} // Emite

main() {
  sArt rArt[] = { 315,"HERNANDEZ",259,"ALVAEZ",812,"PEREZ",
                  127,"QUIROZ",402,"MARTINEZ",706,"LOPEZ",
                  387,"PETROCCELLI",102,"RIOS",
                  321,"CASTILLO",543,"BARRIOS" };
  sArt * ptrItem;
  int   clave;

  clave = 543;
  Emite(rArt);
  qsort (rArt,10,sizeof(sArt),(int(*) (const void *, const void *)) f_cmp_int);
  printf("Buscar el nro.leg. %d\n",clave);
  ptrItem = (sArt*) bsearch (&clave, rArt, 10, sizeof (sArt),
                    (int(*) (const void *, const void *)) f_cmp_int);
  Emite(rArt);
  if (ptrItem)
    printf("El nro.leg. %d se encuentra en el arreglo en la posicion: %d\n",
          ptrItem->nLeg,(ptrItem - &rArt[0]));
  else
    printf("El nro.leg. %d NO se encuentra en el arreglo.\n",clave);
  return 0;
}
```

Ejemplo con tipo struct con campo cadena

```
/*
   Id.Programa: bsearch struct cmp str.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: enero-2024
   Comentario.: Busca la clave de tipo cadena en el arreglo ordenado asc.
              Para que la función bsearch funcione correctamente, el arreglo
              debe encontrarse ordenado en forma ascendente, por lo que,
```

```
            antes de buscar la clave en el arreglo, previamente se lo
            ordena por medio de la funcion qsort en forma ascendente.
            función bsearch esta en la libreria stdlib.
--> Observacion: Existe una incompatibilidad de bsearch si se utiliza cout,
            el cual requiere de cierta libreria o std:: o using namespace,
            y cualquiera del uso de estos recursos, hará que genere un
            error en la compilacion. En su reemplazo se utilizara el
            comando de salida a la consola printf.
   Compilador.: Borland Vers. 5.5
*/

#include <stdio>
#include <stdlib>
#include <string>

const
  CARD = 10;

typedef char str20[21];

struct sArt {
  int     nLeg;
  short edad;
  str20  apellido;
};

int f_cmp_str (const sArt &i,  const sArt &j) {
  return strcmp(i.apellido,j.apellido); // ord. asc., en cambio, return j - i; ord. desc.
} // f_cmp_st

int f_busca_str (const str20 &i,  const sArt &j) {
  return strcmp(i,j.apellido); // busca siempre el valor de i en c/u. de los regs. de j
} // f_busca_str

void Emite(sArt vrA[]) {
  for (short i = 0; i < CARD; i++)
    printf("%2d %4d %2d %s\n",i,vrA[i].nLeg,vrA[i].edad,vrA[i].apellido);
  printf("\n");
} // Emite

main() {
  sArt vrArt[] = {315,37,"HERNANDEZ",259,15,"ALVAREZ",812,9,"PEREZ",
              127,25,"QUIROZ",402,53,"MARTINEZ",706,12,"LOPEZ",
              387,36,"PETROCCELLI",102,29,"RIOS",
```

```
            321,42,"CASTILLO",543,7,"BARRIOS"};
  sArt * ptrItem;
  str20  clave;

  strcpy(clave,"QUIROZ");
  Emite(vrArt);
  qsort (vrArt,CARD,sizeof(sArt),(int(*) (const void *, const void *)) f_cmp_str);
  Emite(vrArt);
  printf("Buscar el apellido %s\n",clave);
  ptrItem = (sArt *) bsearch (clave,vrArt,CARD, sizeof (sArt),
                        (int(*) (const void *, const void *)) f_busca_str);
  if (ptrItem)
    printf("El apellido %s se encuentra en el arreglo en la posicion: %d\n",
          ptrItem->apellido,(ptrItem - &vrArt[0]));
  else
    printf("El apellido %s NO se encuentra en el arreglo.\n",clave);
  return 0;
}
```

Arreglos paralelos

Una alternativa a los arreglos de registro son los arreglos paralelos. Se denominan arreglos paralelos debido a que las componentes de un arreglo se corresponden con las componentes de los otros arreglos, es decir, *Arr1[i]* se corresponde con *Arr2[i]*, ... *Arrn[i]*. Los procesos analizados anteriormente, podrían aplicarse tanto a los arreglos de registros como a los arreglos paralelos. Por ejemplo, si estamos ordenando el arreglo, en el caso de arreglos de registro al intercambiar las componentes *Arr[i]* y *Arr[i+1]* se intercambian todos los campos de esas componentes, es decir se mueve el registro completo de una y de la otra componente. En el caso de los arreglos paralelos se deben intercambiar todas las posiciones correspondientes de cada uno de ellos, vale decir, *Arr1[i]* se intercambia con *Arr1[i+1]*, *Arr2[i]* se intercambia con *Arr2[i+1]*, ..., *Arrn[i]* se intercambia con *Arrn[i+1]*.

Desplazamiento de elementos en un arreglo

Un arreglo puede generarse incorporando valores en forma ordenada en el mismo instante en que ocurre el evento. Esta acción implicará realizar un corrimiento de un lugar, de aquellos elementos que ya estuvieran en el arreglo para abrir un lugar al nuevo elemento que se insertará en la posición apropiada para mantener el orden, ya sea ascendente o descendente. La siguiente figura muestra un ejemplo de insertar el valor 43 en un arreglo que ya contiene los siguientes valores:

2	9	12	28	35	41	49	52	87	94					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Según el modelo presentado el valor 43 deberá ser insertado entre los valores 41 y 49. Para poder llevar a cabo esto, los valores mayores a 41 deberán ser desplazados un lugar hacia la derecha. Al hacerlo debemos tener cuidado de no perder ningún valor en el arreglo, está claro que si moviéramos el valor 49 a la posición siguiente perderíamos el valor 52, por lo tanto el desplazamiento debería realizarse desde el fondo hacia arriba o en esta situación desde la derecha hacia la izquierda, esto es, el valor 94 se movería a la posición 11, luego el valor 87 se movería a la posición 10, y así sucesivamente hasta alcanzar el valor 49 que se movería a la posición 8. Por último insertaríamos el valor 43 en la posición 7. Este proceso debe contemplar todos los casos posibles, es decir, al insertar el primer elemento, al insertar el último elemento que completa el arreglo, al insertar al inicio, al insertar al final o en cualquier otra ubicación, incluso con valores que puedan repetirse, insertar ya dado un juego de valores que se ingresarán ordenados ascendente o descendentemente.
El proceso deberá contemplar entonces todos estos casos. A continuación se presenta el algoritmo correspondiente.

o, como el siguiente módulo, el cual antes de invocar se debe determinar que el valor no exista en el arreglo, si es así, se incrementa en 1 la cantidad de elementos en el arreglo y a continuación se invoca al módulo pasando como argumentos el **arreglo**, el **elemento** a incorporar y la **cardinalidad** actualizada previamente.

Ejemplo de módulo de InsertarEnOrden

Ejemplos codificados en C++

```
/*
   Id.Programa: G4Ej99InsertarOrdenado.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: agosto-2014
   Comentario.: Insertar elementos en el arreglo ordenado.
*/

#include <fstream>
using namespace std;

const
  MAX_ELEM = 5;

typedef unsigned char byte;
typedef unsigned short word;
typedef byte tvEnt[MAX_ELEM];

// Prototipos -------------------------
void InsertarEnOrden(tvEnt , byte , byte );
// Fin Prototipos ---------------------

void InsertarEnOrden(tvEnt vE, byte elem, byte card) {

        while (card > 0 && elem < vE[card - 1]) {
                vE[card] = vE[card - 1];
                card--;
        }
        vE[card] = elem;
} // InsertarEnOrden
```

```
main() {
   tvEnt vEnt;
   word valor;

         for(byte i = 0; i < MAX_ELEM; i++) {
                  cout << "Ing. valor del elemento: ";
                  cin >> valor;
                  InsertarEnOrden(vEnt,valor,i);
         }
         for(byte i = 0; i < MAX_ELEM; i++)
                  cout << (short) i << ": " << (short) vEnt[i] << endl;
   return 0;
} // main
```

Codificación G4Ej02 Operaciones con arreglos

```
/*
   Id.Programa: G4Ej02 OperacArreglos.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: ago-2014
   Comentario.: Operaciones con arreglos, opc. de Menu:
                a) Promedio
                b) Maximo
                c) Minimo
                d) Sumatoria
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_ELEM = 100,
  MIN      = 53,
  MAX      = 391;

typedef unsigned char byte;
typedef unsigned short word;
typedef byte tvEnt[MAX_ELEM];

// Prototipos ------------------
byte ObtCard();
void GenVec(tvEnt , byte );
void EmitirVec(tvEnt , byte );
word SumVec(tvEnt , byte );
float PromVec(tvEnt , byte );
byte MaxVec(tvEnt , byte );
byte MinVec(tvEnt , byte );
```

```
void Menu(tvEnt , byte );
// Fin Prototipos -------------

byte ObtCard() {
  word cantElem;

  do {
    gotoxy(5,3); clreol();
    cout << "Ing.Cardinalidad del vector [1.." << MAX_ELEM << "]: ";
    cin >> cantElem;
  } while (!(cantElem >= 1 && cantElem <= MAX_ELEM));
  return cantElem;

} //ObtenerCard

void GenVec(tvEnt vE, byte card) {

  randomize();
  for (byte i = 0; i < card; i++)
    vE[i] = random(MAX - MIN + 1) + MIN;
} //GenVec

void EmitirVec(tvEnt vE, byte card) {

  clrscr();
  for (byte i = 0; i < card; i++)
    cout << setw(8) << (short) vE[i];
  gotoxy(1,24);
  cout << "Oprima Enter para continuar...";
  getche();
} //EmitirVec

word SumVec(tvEnt vE, byte card) {
  word sum = 0;

  for (byte i = 0; i < card; i++)
    sum += vE[i];
  return sum;
} //SumVec

float PromVec(tvEnt vE, byte card) {

  return SumVec(vE,card) / (float) card;
} //PromVec

byte MaxVec(tvEnt vE, byte card) {
  byte maxElem;

  maxElem = vE[0];
  for (byte i = 1; i < card; i++)
```

```
    if (vE[i] > maxElem)
      maxElem = vE[i];
  return maxElem;
} //MaxVec

byte MinVec(tvEnt vE, byte card) {
  byte minElem;

  minElem = vE[0];
  for (byte i = 1; i < card; i++)
    if (vE[i] < minElem)
      minElem = vE[i];
  return minElem;
} //MinVec


void Menu(tvEnt vE, byte card) {
  char opc;

  cout.precision(2);
  cout.setf(ios::fixed);
  do {
    clrscr();
    gotoxy(20, 1); cout << "Operaciones con Arreglos";
    gotoxy(10, 5); cout << "A: Promedio";
    gotoxy(10, 7); cout << "B: Maximo elemento";
    gotoxy(10, 9); cout << "C: Minimo elemento";
    gotoxy(10,11); cout << "D: Sumatoria";
    gotoxy(10,13); cout << "E: Salir";
    do {
      gotoxy(15,16); cout << "Elija su opcion: [ ]";
      gotoxy(wherex() - 2, wherey());
      opc = toupper(getch());
    } while (!(opc >= 'A' && opc <= 'E'));
    gotoxy(20,20);
    switch (opc) {
      case 'A': cout << "Promedio.: " << setw(8) << PromVec(vE,card);
                break;
      case 'B': cout << "Maximo...: " << setw(8) << (short) MaxVec(vE,card);
                break;
      case 'C': cout << "Minimo...: " << setw(8) << (short) MinVec(vE,card);
                break;
      case 'D': cout << "Sumatoria: " << setw(8) << SumVec(vE,card);
                break;
    }
    if (opc != 'E')
      getche();
  } while (!(opc == 'E'));
} //Menu
```

```
main() {
  tvEnt vEnt;
  byte  cantElem;

  cantElem = ObtCard();
  GenVec(vEnt,cantElem);
  EmitirVec(vEnt,cantElem);
  Menu(vEnt,cantElem);
  return 0;
}
```

Codificación G4Ej03 Generar al azar sin repetir

```
/*
    Id.Programa: G4Ej03GeneraAzarSinRep.cpp
    Autor......: Lic. Hugo A. Cuello
    Fecha......: ago-2014
    Comentario.: Generar arreglo con enteros al azar:
                 Caso 1a) Sacar, cantar y meter en la misma bolsa.
                 Caso 1b) Idem anterior, pero optimizando tiempo.
                 Caso 2)  Sacar, cantar y apartar de la bolsa.
                 Caso 2b) Idem anterior pero generando de abajo hacia arriba
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_ELEM = 90;

typedef unsigned short byte;
typedef unsigned int   word;
typedef byte tvEnt[MAX_ELEM + 1]; //pos. cero no se va a usar.
typedef bool tvBool[MAX_ELEM + 1]; //pos. cero no se va a usar.

// Prototipos -----------------------------
byte ObtCard();
void GenVecSinRepCaso1a(tvEnt , byte );
void GenVecSinRepCaso1b(tvEnt , byte );
void IntCmb(byte &, byte &);
void GenVecSinRepCaso2(tvEnt , byte );
void GenVecSinRepCaso2Inv(tvEnt , byte );
void OrdxBur(tvEnt , byte );
void EmiteVec(tvEnt , byte );
void EmiteVecInv(tvEnt , byte );
// Fin Prototipos -------------------------
```

```
byte ObtCard() {
  byte card;

  clrscr();
  gotoxy(10,5);
  cout << "Ing.cardinalidad entre 1 y " << MAX_ELEM << ": ";
  do {
    gotoxy(42,5); clreol();
    cin >> card;
  } while (!(card >= 1 && card <= MAX_ELEM));
  return card;
} //ObtCard

void GenVecSinRepCaso1a(tvEnt vE, byte card) {
  byte j;

  for (byte i = 1; i <= card; i++) {
    vE[i] = random(MAX_ELEM ) + 1;
    j = 1;
    while (j < i)
    if (vE[i] == vE[j]) {
      vE[i] = random(MAX_ELEM) + 1;
      j = 1;
    }
    else
      j++;
  }
} //GenVecSinRepCaso1a

void GenVecSinRepCaso1b(tvEnt vE, byte card) {
  byte   x;
  tvBool vBool;

  for (byte i = 1; i <= MAX_ELEM; i++)
    vBool[i] = false;
  for (byte i = 1; i <= card; i++) {
    do
      x = random(MAX_ELEM) + 1;
    while (vBool[x]);
    vBool[x] = true;
    vE[i] = x;
  }
} //GenVecSinRepCaso1b

void IntCmb(byte &elem1, byte &elem2) {
  byte aux = elem1;

  elem1 = elem2;
  elem2 = aux;
```

```
} //IntCmb

void GenVecSinRepCaso2(tvEnt vE, byte card) {
  byte x;

  for (byte i = 1; i <= MAX_ELEM; i++)
    vE[i] = i;
  for (byte i = 1; i <= card; i++) {
    x = random(MAX_ELEM - i + 1) + i;
    IntCmb(vE[i],vE[x]);
  }
} //GenVecSinRepCaso2

void GenVecSinRepCaso2Inv(tvEnt vE, byte card) {
  byte x;

  for (byte i = 1; i <= MAX_ELEM; i++)
    vE[i] = i;
  for (byte i = 1; i <= card; i++) {
    x = random(MAX_ELEM - i + 1) + 1;
    IntCmb(vE[MAX_ELEM - i + 1],vE[x]);
  }
} //GenVecSinRepCaso2Inv

void OrdxBur(tvEnt vE, byte card) {
  byte k = 0;
  bool ordenado;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (vE[i] > vE[i + 1]) {
        ordenado = false;
        IntCmb(vE[i], vE[i + 1]);
      }
  } while (!ordenado);
} //OrdxBur

void EmiteVec(tvEnt vE, byte card) {

  for (byte i = 1; i <= card; i++)
    cout << setw(4) << vE[i];
  cout << endl << endl;
} //EmiteVec

void EmiteVecInv(tvEnt vE, byte card) {

  for (byte i = card; i >= 1; i--)
    cout << setw(4) << vE[i];
```

```
  cout << endl << endl;
} //EmiteVecInv

main() {
  tvEnt vEnt;
  byte  cantElem;

  cantElem = ObtCard();
  GenVecSinRepCaso1a(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  OrdxBur(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  GenVecSinRepCaso1b(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  OrdxBur(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  GenVecSinRepCaso2(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  OrdxBur(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  GenVecSinRepCaso2Inv(vEnt,cantElem);
  EmiteVec(vEnt,cantElem);
  OrdxBur(vEnt,cantElem);
  EmiteVecInv(vEnt,cantElem);
  return 0;
}
```

Operaciones de conjuntos utilizando arreglos

```
/*
   Id.Programa: G4Ej04OperacCjtoConArr.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: ago-2014
   Comentario.: Operaciones de Conjunto con arreglos:
                a) Union
                b) Interseccion
                c) Diferencia
                d) Diferencia Simetrica
                e) Complemento de A U B
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_ELEM = 90;
```

```
typedef char str20[21];
typedef unsigned char byte;
typedef unsigned short word;
typedef byte tvEnt[MAX_ELEM + 1]; //pos. cero no se va a usar.

// Prototipos ------------------------------------------------
byte ObtCard(str20 );
void IntCmb(byte &, byte &);
void OrdxBur(tvEnt , byte );
char Menu();
byte BusBin(tvEnt , byte , byte );
void Union(tvEnt , tvEnt , tvEnt , byte , byte , byte &);
void Intersec(tvEnt , tvEnt , tvEnt , byte , byte , byte &);
void Difer(tvEnt , tvEnt , tvEnt , byte , byte , byte &);
void DifSim(tvEnt , tvEnt , tvEnt , byte , byte , byte &);
void Cmplto(tvEnt , tvEnt , tvEnt , byte , byte , byte &);
void GenVecSinRepCaso2(tvEnt , byte );
void EmiteVec(tvEnt , byte , str20 );
// Fin Prototipos -------------------------------------------

byte ObtCard(str20 titulo) {
  word card;

  clrscr();
  do {
    gotoxy(10,3);
    cout << titulo << " #: ";
    clreol();
    cin >> card;
  } while (!(card >= 1 && card <= MAX_ELEM));
  return card;
} //ObtCard

void IntCmb(byte &elem1, byte &elem2) {
        byte aux = elem1;

        elem1 = elem2;
        elem2 = aux;
} // IntCmb

void OrdxBur(tvEnt vE, byte card) {
        bool ordenado;
        byte k = 0;

        do {
                ordenado = true;
                k++;
                for (byte i = 1; i <= card - k; i++)
                        if (vE[i] > vE[i + 1]) {
                                IntCmb(vE[i],vE[i + 1]);
```

```
                                    ordenado = false;
                                }
            } while (!ordenado);
} // OrdxBur

char Menu() {
  char opc;

  clrscr();
  gotoxy(10,5);
  cout << endl;
  cout << setw(10) << " " << "A: Union" << endl;
  cout << setw(10) << " " << "B: Interseccion" << endl;
  cout << setw(10) << " " << "C: Diferencia" << endl;
  cout << setw(10) << " " << "D: Diferencia Simetrica" << endl;
  cout << setw(10) << " " << "E: Complemento" << endl;
  cout << setw(10) << " " << "F: Finalizar" << endl;
  cout << endl;
  cout << setw(10) << "Su opcion: ";
  do
    opc = toupper(getch());
  while (!(opc >= 'A' && opc <= 'F'));
  cout << endl;
  return opc;
} //Menu

byte BusBin(tvEnt vX, byte clv, byte ult) {
  byte pri = 1,
         med;

  while (pri <= ult) {
      med = (pri + ult) / 2;
      if (vX[med] == clv)
          return med;
      else
            if (vX[med] < clv)
                pri = med + 1;
            else
                ult = med - 1;
  }
  return 0;
} //BusBin

void Union(tvEnt vP, tvEnt vQ, tvEnt vR, byte cP, byte cQ, byte &cR) {

  for (byte i = 1; i <= cP; i++)
    vR[i] = vP[i];
  cR = cP;
  for (byte i = 1; i <= cQ; i++)
```

```
    if (!BusBin(vP,vQ[i],cP)) {
      cR++;
      vR[cR] = vQ[i];
    }
} //Union

void Intersec(tvEnt vP, tvEnt vQ, tvEnt vR, byte cP, byte cQ, byte &cR) {

  for (byte i = 1; i <= cP; i++)
    if (BusBin(vQ,vP[i],cQ)) {
    cR++;
    vR[cR] = vP[i];
  }
} //Intersec

void Difer(tvEnt v1, tvEnt v2, tvEnt vR, byte c1, byte c2, byte &cR) {

  for (byte i = 1; i <= c1; i++)
    if (!BusBin(v2,v1[i],c2)) {
      cR++;
      vR[cR] = v1[i];
    }
} //Difer

void DifSim(tvEnt vP, tvEnt vQ, tvEnt vR, byte cP, byte cQ, byte &cR) {

  Difer(vP,vQ,vR,cP,cQ,cR);
  Difer(vQ,vP,vR,cQ,cP,cR);
} //DifSim

void Cmplto(tvEnt vP, tvEnt vQ, tvEnt vR, byte cP, byte cQ, byte &cR) {

  for (byte i = 1; i <= MAX_ELEM; i++)
    if (!BusBin(vP,i,cP) && !BusBin(vQ,i,cQ)) {
      cR++;
      vR[cR] = i;
    }
} //Cmplto

void GenVecSinRepCaso2(tvEnt vE, byte card) {
  byte x;

  for (byte i = 1; i <= MAX_ELEM; i++)
    vE[i] = i;
  for (byte i = 1; i <= card; i++) {
    x = random(MAX_ELEM - i + 1) + i;
    IntCmb(vE[i],vE[x]);
  }
} //GenVecSinRepCaso2
```

```
void EmiteVec(tvEnt vE, byte card, str20 titulo) {

  cout << titulo << endl;
  cout << "#" << titulo << ": " << setw(3) << (short) card << endl;
  for (byte i = 1; i <= card; i++)
    cout << setw(4) << (short) vE[i];
  cout << endl;
  getche();
} //EmiteVec

main() {
  tvEnt p,
        q,
        r;
  byte  m,
        n,
        k;
  char  opc;

  m = ObtCard("Cjto. P");
  n = ObtCard("Cjto. Q");
  GenVecSinRepCaso2(p,m);
  GenVecSinRepCaso2(q,n);
  OrdxBur(p,m);
  OrdxBur(q,n);
  do {
    opc = Menu();
    k = 0;
    switch (opc) {
      case 'A': Union(p,q,r,m,n,k);
                break;
      case 'B': Intersec(p,q,r,m,n,k);
                break;
      case 'C': Difer(p,q,r,m,n,k);
                break;
      case 'D': DifSim(p,q,r,m,n,k);
                break;
      case 'E': Cmplto(p,q,r,m,n,k);
                break;
    }
    if (opc != 'F') {
     EmiteVec(p,m,"Cjto. P");
     EmiteVec(q,n,"Cjto. Q");
     EmiteVec(r,k,"Cjto. R");
    }
  } while (!(opc == 'F'));
  return 0;
}
```

Codificación G4Ej05 Fusión de arreglos

```
/*
   Id.Programa: G4Ej05FusionArreglos.cpp
   Autor..........: Lic. Hugo A. Cuello
   Fecha..........: ago-2014
   Comentario.: Fusion o Mezcla (Merge) de arreglos
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_ELEM = 90;

typedef char str20[21];
typedef unsigned char byte;
typedef unsigned short word;
typedef byte tvEnt[MAX_ELEM + 1]; //pos. cero no se va a usar.
typedef byte tvEntD[2 * MAX_ELEM + 1]; //pos. cero no se va a usar.

// Prototipos ------------------------------------------------
byte ObtCard(str20 );
void IntCmb(byte &, byte &);
void OrdxBur(tvEnt , byte );
void Asignar(byte &, byte , byte &, byte &, byte , bool &);
void Intercalar(tvEnt , tvEnt , tvEntD , byte , byte );
void GenVecSinRepCaso2(tvEnt , byte );
void EmiteVec(tvEnt , byte , str20 );
// Fin Prototipos ---------------------------------------------

byte ObtCard(str20 titulo) {
  word card;

  clrscr();
  do {
    gotoxy(10,3);
    cout << titulo << " #: ";
    clreol();
    cin >> card;
  } while (!(card >= 1 && card <= MAX_ELEM));
  return card;
} //ObtCard

void IntCmb(byte &elem1, byte &elem2) {
          byte aux = elem1;
```

```
        elem1 = elem2;
        elem2 = aux;
} // IntCmb

void OrdxBur(tvEnt vE, byte card) {
        bool ordenado;
        byte k = 0;

        do {
                ordenado = true;
                k++;
                for (byte i = 1; i <= card - k; i++)
                        if (vE[i] > vE[i + 1]) {
                                IntCmb(vE[i],vE[i + 1]);
                                ordenado = false;
                        }
        } while (!ordenado);
} // OrdxBur

void Asignar(byte &elemR, byte elemX, byte &indX, byte &indR, byte cX, bool
&fdvX) {

  elemR = elemX;
  indX++;
  indR++;
  fdvX = indX > cX;
} //Asignar

void Intercalar(tvEnt vP, tvEnt vQ, tvEntD vR, byte cP, byte cQ) {
  byte i,
       j,
       k;
  bool fdvP = false,
       fdvQ = false;

  i = j = k = 1;
  while (!fdvP || !fdvQ) {
    if (!fdvP && !fdvQ)
      if (vP[i] <= vQ[j])
        Asignar(vR[k],vP[i],i,k,cP,fdvP);
      else
        Asignar(vR[k],vQ[j],j,k,cQ,fdvQ);
    else
      if (!fdvP)
        Asignar(vR[k],vP[i],i,k,cP,fdvP);
      else
        Asignar(vR[k],vQ[j],j,k,cQ,fdvQ);
  }
} //Intercalar
```

```
void GenVecSinRepCaso2(tvEnt vE, byte card) {
  byte x;

  for (byte i = 1; i <= MAX_ELEM; i++)
    vE[i] = i;
  for (byte i = 1; i <= card; i++) {
    x = random(MAX_ELEM - i + 1) + i;
    IntCmb(vE[i],vE[x]);
  }
} //GenVecSinRepCaso2

void EmiteVec(tvEnt vE, byte card, str20 titulo) {

  cout << titulo << endl;
  cout << "#" << titulo << ": " << setw(3) << (short) card << endl;
  for (byte i = 1; i <= card; i++)
    cout << setw(4) << (short) vE[i];
  cout << endl;
  cin.get();
} //EmiteVec

int main() {
  tvEnt  p,
         q;
  tvEntD r;
  byte   m,
         n;

  m = ObtCard("Cjto. P");
  n = ObtCard("Cjto. Q");
  GenVecSinRepCaso2(p,m);
  GenVecSinRepCaso2(q,n);
  OrdxBur(p,m);
  OrdxBur(q,n);
  Intercalar(p,q,r,m,n);
  EmiteVec(p,m,"Cjto. P");
  EmiteVec(q,n,"Cjto. Q");
  EmiteVec(r,m + n,"Cjto. R");
  return 0;
}
```

Codificación G4Ej07 Encuesta Discográfica

```
/*
   Id.Programa: G4Ej07DiscoEnc.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: jun-2014
   Comentario.: Ranking de temas musicales (1..30).
                1. Listado de Ranking.
                2. Nro.Tema +Votado por H30+.
```

```
                3. Nros. de Temas no votados por las Mujeres.
*/

#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_TEMAS = 30;

typedef unsigned char byte;
typedef unsigned short word;

struct sEnc {
  char Sexo;
  byte Edad,
         NroTemas[3];
};
struct sRank {
  byte NroTema;
  word Votos,
       H30;
  bool Mujer;
};

typedef sRank tvrRank[MAX_TEMAS + 1]; //tvrRank es el tipo arreglo de registro
                                      //el +1 es un ajuste para no empezar pos. 0.

// Prototipos ---------------------
void Abrir(FILE **);
void IniTbl(tvrRank );
void AcumVotos(tvrRank , sEnc );
void IntCmb(sRank &,sRank &);
void OrdxBur(tvrRank ,byte );
byte MayNroTemaH30(tvrRank );
void EmitePtoB(byte );
void EmitePtoC(tvrRank );
void EmitePtoA(tvrRank );
// Fin Prototipos -----------------

void Abrir(FILE **Enc) {
  *Enc = fopen("Encuestas.Dat","rb");
} //Abrir

void IniTbl(tvrRank vrRank) {
  trRank rRank;

  rRank.Votos = rRank.H30 = 0;
  rRank.Mujer = false;
```

```
  for (byte i = 1; i <= MAX_TEMAS; i++) { //pos.0 en vrRank no se toma en cuenta.
    rRank.NroTema = i;
    vrRank[i] = rRank;
  }
} //IniTbl

void AcumVotos(tvrRank vrRank, sEnc rEnc) {
  byte nTema;

  for (byte i = 0; i <= 2; i++) {
    nTema = rEnc.NroTemas[i];
    vrRank[nTema].Votos++;
    if (rEnc.Sexo == 'F')
      vrRank[nTema].Mujer = true;
    else
      if (rEnc.Edad >= 30)
        vrRank[nTema].H30++;
  }
} //AcumVotos

void IntCmb(sRank &elem1, sRank &elem2) {
  sRank aux = elem1;

  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdxBur(tvrRank vrRank,byte card) {
  bool ordenado;
  byte  k = 0;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (vrRank[i].Votos < vrRank[i + 1].Votos) {
        ordenado = false;
        IntCmb(vrRank[i],vrRank[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

byte MayNroTemaH30(tvrRank vrRank) {
  byte nTema = 1;

  for (byte i = 2; i <= MAX_TEMAS; i++)
    if (vrRank[i].H30 > vrRank[nTema].H30)
      nTema = i;
  return nTema;
```

```
} //MayNroTemaH30

void EmitePtoB(byte nTema) {

  freopen("Encuestas30TemasDisco.Lst","w",stdout);
  cout << "El nro. de tema + votado por los H30+: " << (short) nTema << endl;
} //EmitePtoB

void EmitePtoC(tvrRank vrRank) {

  cout << endl << "Nros. de temas no votados por Mujeres" << endl;
  cout << "   Nro.Tema" << endl;
  for (byte i = 1; i <= MAX_TEMAS; i++)
    if (!vrRank[i].Mujer)
      cout << "     " << setw(2) << (short) i << endl;
} //EmitePtoC

void EmitePtoA(tvrRank vrRank) {

  cout << endl << "Listado de Ranking de Temas Musicales" << endl;
  cout << "  Nro.Ord.  Nro.Tema Cant.Votos" << endl;
  for (byte i = 1; i <= MAX_TEMAS; i++)
    cout << "     " << setw(2) << (short) i << "        "
                << setw(2) << (short) vrRank[i].NroTema << setw(6) << ""
                << setw(4) << vrRank[i].Votos << endl;
  freopen("CON","w",stdout);
} //EmitePtoA

main() {
  FILE   *Encuestas;
  sEnc   rEncuesta;
  tvrRank vrRanking;

  Abrir(&Encuestas);
  IniTbl(vrRanking);
  while (fread(&rEncuesta,sizeof (rEncuesta),1,Encuestas))
    AcumVotos(vrRanking,rEncuesta);
  EmitePtoB(MayNroTemaH30(vrRanking));
  EmitePtoC(vrRanking);
  OrdxBur(vrRanking,MAX_TEMAS);
  EmitePtoA(vrRanking);
  fclose(Encuestas);
  return 0;
}
```

Codificación G4Ej07 Encuesta Discográfica versión stream

```
/*
   Id.Programa: G4Ej07DiscoEnc.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: jun-2014
   Comentario.: Ranking de temas musicales (1..30).
                1. Listado de Ranking.
                2. Nro.Tema +Votado por Hom30+.
                3. Nros. de Temas no votados por las Mujeres.
*/

#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_TEMAS = 30;

typedef unsigned char byte;
typedef unsigned short word;

struct sEnc {
  char Sexo;
  byte Edad,
       NroTemas[3];
};

struct sRank {
  byte NroTema;
  word Votos,
       Hom30;
  bool Mujer;
};

typedef sRank tvsRank[MAX_TEMAS + 1]; //tvsRank es el tipo arreglo de registro
                                      //el +1 es un ajuste para no empezar pos. 0.

// Prototipos ---------------------
void IniTbl(tvsRank);
void AcumVotos(tvsRank , sEnc );
void IntCmb(sRank &,sRank &);
void OrdxBur(tvsRank ,byte );
byte MayNroTemaHom30(tvsRank );
void EmitePtoB(byte );
void EmitePtoC(tvsRank );
void EmitePtoA(tvsRank );
// Fin Prototipos -----------------

void IniTbl(tvsRank vrRank) {
```

```
  sRank rRank;

  rRank.Votos = rRank.Hom30 = 0;
  rRank.Mujer = false;
  for (byte i = 1; i <= MAX_TEMAS; i++) { //pos.0 en vrRank no se toma en cuenta.
    rRank.NroTema = i;
    vrRank[i] = rRank;
  }
} //IniTbl

void AcumVotos(tvsRank vrRank, sEnc rEnc) {
  byte nTema;

  for (byte i = 0; i <= 2; i++) {
    nTema = rEnc.NroTemas[i]; //estamos haciendo una copia
    vrRank[nTema].Votos++;
    if (rEnc.Sexo == 'F')
      vrRank[nTema].Mujer = true;
    else
      if (rEnc.Edad >= 30)
        vrRank[nTema].Hom30++;
  }
} //AcumVotos

void IntCmb(sRank &elem1,sRank &elem2) {
  tsRank aux = elem1;

  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdxBur(tvsRank vrRank,byte card) {
  bool ordenado;
  byte  k = 0;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (vrRank[i].Votos < vrRank[i + 1].Votos) {
        ordenado = false;
        IntCmb(vrRank[i],vrRank[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

byte MayNroTemaHom30(tvsRank vrRank) {
  byte nTema = 1;
```

```
  for (byte i = 2; i <= MAX_TEMAS; i++)
    if (vrRank[i].Hom30 > vrRank[nTema].Hom30)
      nTema = i;
  return nTema;
} //MayNroTemaHom30

void EmitePtoB(byte nTema) {

  cout << "Por ahora sale en la Pantalla" << endl;
  freopen("Encuestas30TemasDisco.Lst","w",stdout);
  cout << "El nro. de tema + votado por los Hom30+: " << (short) nTema << endl;
} //EmitePtoB

void EmitePtoC(tvsRank vrRank) {

  cout << endl << "Nros. de temas no votados por Mujeres" << endl;
  cout << "   Nro.Tema" << endl;
  for (byte i = 1; i <= MAX_TEMAS; i++)
    if (!vrRank[i].Mujer)
      cout << "     " << setw(2) << (short) i << endl;
} //EmitePtoC

void EmitePtoA(tvsRank vrRank) {

  cout << endl << "Listado de Ranking de Temas Musicales" << endl;
  cout << "  Nro.Ord.  Nro.Tema Cant.Votos" << endl;
  for (byte i = 1; i <= MAX_TEMAS; i++)
    cout << "     " << setw(2) << (short) i << "        "
                << setw(2) << (short) vrRank[i].NroTema << setw(6) << ""
                << setw(4) << vrRank[i].Votos << endl;
  freopen("CON","w",stdout);
  cout <<"Vuelvo a la pantalla";
} //EmitePtoA

main() {
  ifstream Encuestas("Encuestas.Dat",ios::binary);
  sEnc   rEncuesta;
  tvsRank vrRanking;
  //sRank  vrRanking[MAX_TEMAS+1];

  IniTbl(vrRanking); // NroTema: Pos i, Votos: 0, Hom30: 0, Mujer: false.
  while (Encuestas.read((char *) &rEncuesta,sizeof (rEncuesta)))
    AcumVotos(vrRanking,rEncuesta);

  EmitePtoB(MayNroTemaHom30(vrRanking));
  EmitePtoC(vrRanking);
  OrdxBur(vrRanking,MAX_TEMAS);
  EmitePtoA(vrRanking);
  Encuestas.close();
```

```
  return 0;
}
```

Codificación G4Ej08 Viajes de Camioneros

```
/*
    Id.Programa: G4Ej08ViajesCam.cpp
    Autor......: Lic. Hugo A. Cuello
    Fecha......: agosto-2014
    Comentario.: Viajes de Camioneros con material fragil (1..Max.100).
                 1. Listado de Camioneros ord. x Cant.Unid.Rotas asc.

*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_CAM = 100;

typedef char str5[6];
typedef unsigned char byte;
typedef unsigned short word;

struct sVia {
  str5 CodLeg;
  byte CodPcia;
  word CantRotas;
};

struct sCam {
  str5 CodLeg;
  byte CantVjes;
  word CantUniRotas;
};

typedef sCam tvrCam[MAX_CAM + 1]; //tvrRank es el tipo arreglo de registro

// Prototipos -----------------------------
void Abrir(FILE **);
void AcumViajes(tvrCam , sVia , byte &);
void IntCmb(sCam &, sCam &);
void OrdxBur(tvrCam , byte );
void Listado(tvrCam , byte );
// Fin Prototipos -------------------------

void Abrir(FILE **Vje) {
  *Vje = fopen("Viajes.Dat","rb");
```

```
} //Abrir

byte BusLinVec(tvrCam vrCam, str5 nLeg, byte card) {
  byte i = 1;

  while (i <= card && strcmp(nLeg,vrCam[i].CodLeg) != 0)
    i++;
  return i;
} //BusLinVec

void AcumViajes(tvrCam vrCam, sVia rVia, byte &cCam) {
  byte posCam;

  posCam = BusLinVec(vrCam,rVia.CodLeg,cCam);
  if (posCam > cCam) {
    cCam++;
    strcpy(vrCam[posCam].CodLeg ,rVia.CodLeg);
    vrCam[posCam].CantVjes = 0;
    vrCam[posCam].CantUniRotas = 0;
  }
  vrCam[posCam].CantVjes++;
  vrCam[posCam].CantUniRotas += rVia.CantRotas;
} //AcumViajes

void IntCmb(sCam &elem1, sCam &elem2) {
  sCam aux = elem1;

  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdxBur(tvrCam vrCam, byte card) {
  bool ordenado;
  byte k = 0;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (vrCam[i].CantUniRotas > vrCam[i + 1].CantUniRotas) {
        ordenado = false;
        IntCmb(vrCam[i],vrCam[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

void Listado(tvrCam vrCam, byte cCam) {

  freopen("ViajesCamioneros.Lst","w",stdout);
```

```
  cout << setw(5) << ""
       << "Listado de Camioneros ordenado por Cant. Unid. rotas"
       << endl << endl;
  cout << "        Cod. Leg.     Cant. Viajes       Cant.Uni.Rotas" << endl;
  for (byte i = 1; i <= cCam; i++) {
    cout.setf(ios::left);
    cout << setw(10) << "" << setw(5) << vrCam[i].CodLeg;
    cout.unsetf(ios::left);
    cout << setw(15) << (short) vrCam[i].CantVjes
         << setw(19) << vrCam[i].CantUniRotas << endl;
  }
  freopen("CON","w",stdout);
} //Listado

main() {
  FILE  *Viajes;
  sVia  rViaje;
  tvrCam vrCamioneros;
  byte   canCam = 0;

  Abrir(&Viajes);
  while (fread(&rViaje,sizeof (rViaje),1,Viajes))
    AcumViajes(vrCamioneros,rViaje,canCam);
  OrdxBur(vrCamioneros,canCam);
  Listado(vrCamioneros,canCam);
  fclose(Viajes);
  return 0;
}
```

Codificación G4Ej08 Viajes de Camioneros versión stream

```
/*
    Id.Programa: G4Ej08ViajesCam.cpp
    Autor......: Lic. Hugo A. Cuello
    Fecha......: agosto-2014
    Comentario.: Viajes de Camioneros con material fragil (1..Max.100).
                 1. Listado de Camioneros ord. x Cant.Unid.Rotas asc.

*/

#include<iomanip>
#include<fstream>
using namespace std;

const
  short MAX_CAM = 100;

typedef char str5[6];
typedef unsigned short word;
```

```
struct sVia {
  str5    CodLeg;
  short  CodPcia;
  word  CantRotas;
};

struct sCam {
  str5   CodLeg;
  short CantVjes;
  word CantUniRotas;
};

typedef sCam tvrCam[MAX_CAM + 1]; //tvrCam es el tipo arreglo de registro

// Prototipos -----------------------------
void Abrir(FILE **);
void AcumViajes(tvrCam , trVia , short &);
void IntCmb(sCam &, sCam &);
void OrdxBur(tvrCam , short);
void Listado(tvrCam , short );
// Fin Prototipos -------------------------

short BusLinVec(tvrCam vrCam, str5 nLeg, short card) {
  short i = 1;

  while (i <= card && strcmp(nLeg,vrCam[i].CodLeg) != 0)
    i++;
  return i;
} //BusLinVec

void AcumViajes(tvrCam vrCam, sVia rVia, short &cCam) {
  short posCam;

  posCam = BusLinVec(vrCam,rVia.CodLeg,cCam);
  if (posCam > cCam) {
    cCam++;
    strcpy(vrCam[posCam].CodLeg ,rVia.CodLeg);
    vrCam[posCam].CantVjes = 0;
    vrCam[posCam].CantUniRotas = 0;
  }
  vrCam[posCam].CantVjes++;
  vrCam[posCam].CantUniRotas += rVia.CantRotas;
} //AcumViajes

void IntCmb(trCam &elem1,trCam &elem2) {
  trCam aux = elem1;

  elem1 = elem2;
  elem2 = aux;
```

```
} // IntCmb

void OrdxBur(tvrCam vrCam, short card) {
  bool ordenado;
  short k = 0;

  do {
    ordenado = true;
    k++;
    for (short i = 1; i <= card - k; i++)
      if (vrCam[i].CantUniRotas > vrCam[i + 1].CantUniRotas) {
        ordenado = false;
        IntCmb(vrCam[i],vrCam[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

void Listado(tvrCam vrCam, short cCam) {

  freopen("ViajesCamioneros.Lst","w",stdout);
  cout << setw(5) << ""
       << "Listado de Camioneros ordenado por Cant. Unid. rotas"
       << endl << endl;
  cout << "        Cod. Leg.     Cant. Viajes       Cant.Uni.Rotas" << endl;
  for (short i = 1; i <= cCam; i++) {
    cout.setf(ios::left);
    cout << setw(10) << "" << setw(5) << vrCam[i].CodLeg;
    cout.unsetf(ios::left);
    cout << setw(15) << (short) vrCam[i].CantVjes
         << setw(19) << vrCam[i].CantUniRotas << endl;
  }
  freopen("CON","w",stdout);
} //Listado

main() {
  ifstream Viajes("Viajes.Dat",ios::binary);
  sVia  rViaje;
  tvrCam vrCamioneros;
  short   canCam = 0;

  while (Viajes.read((char *) &rViaje,sizeof (rViaje)))
    AcumViajes(vrCamioneros,rViaje,canCam);
  OrdxBur(vrCamioneros,canCam);
  Listado(vrCamioneros,canCam);
  Viajes.close();
  return 0;
}
```

Codificación G4Ej17 Rutas Argentinas

```
/*
   Id.Programa: G4Ej17Rutas.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: septiembre-2014
   Comentario.: Consultas de Localidades.

   *Estructuras de Datos

   Vector de Localidades con nomLoc (str20) CON MAX_LOC ord. ascendente.
   Matriz cuadrada de Rutas con Distancias_NroRutas con MAX_LOC:
     Inicializar en cero la matirz con la cantidad de Localidades canLoc.
     procesadas en el archivo de Localidades.
     Distancias en la triangular inferior (i > j)
     Nros. de Rutas en la triangular superior (i < j)
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

#define readf(file,buf) fread(&(buf),sizeof((buf)),1,(file))

const
  MAX_LOC = 100;

typedef char str20[21];
typedef unsigned char byte;
typedef unsigned short word;

struct sRut {
  str20 nomLoc1,
        nomLoc2;
  word  dist,
        nroRuta;
};

typedef str20 tvLoc[MAX_LOC + 1]; //tvLoc es el vector de Localidades, pos. 0 no se
usa
typedef word  tmRut[MAX_LOC + 1][MAX_LOC + 1]; //tmRut es la matriz de Rutas
                                //(Tri.Inf.: Dist., Tri.Sup.: NroRuta) pos. 0 no se usa

// Prototipos ------------------------------
void Abrir(FILE **, FILE ** );
void IntCmb(str20, str20 );
void IntCmb(byte &, byte & );
void OrdxBur(tvLoc, byte );
void GenLocal(FILE *, tvLoc, byte & );
```

```
void InicRut(tmRut, byte );
byte BusBinVec(tvLoc, str20, byte );
void GenRutas(FILE *, tmRut, tvLoc, byte );
void EmiteMat(tmRut, tvLoc, byte );
void ObtLocal(byte &, byte &, tvLoc, byte );
void EncontroLoc(word &, word &, str20, bool &,word, word, word );
void CalcDist(tmRut, byte, byte, byte, word &, word &, str20, word &, word &  );
void EmiteDist(word, word, str20, tvLoc, word, word );
void Cerrar(FILE *, FILE * );
// Fin Prototipos -------------------------

void Abrir(FILE **Loc, FILE **Rut) {
  *Loc = fopen("Localidades.Dat","rb");
  *Rut = fopen("Rutas.Dat","rb");
} //Abrir

void IntCmb(str20 elem1, str20 elem2) {
  str20 aux;

  strcpy(aux,elem1);
  strcpy(elem1,elem2);
  strcpy(elem2,aux);
} // IntCmb

void IntCmb(byte &elem1, byte &elem2) {
  byte aux;

  aux = elem1;
  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdxBur(tvLoc vLoc, byte card) {
  bool ordenado;
  byte k = 0;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (strcmp(vLoc[i],vLoc[i + 1]) > 0) {
        ordenado = false;
        IntCmb(vLoc[i],vLoc[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

void GenLocal(FILE *Loc, tvLoc vLoc, byte &cLoc) {
```

```
  cLoc = 0;
  while (readf(Loc,vLoc[++cLoc]));
  cLoc--;
  OrdxBur(vLoc,cLoc);
 } //GenLocal

void InicRut(tmRut mRut, byte cLoc) {

  for (byte i = 1; i <= cLoc; i++)
    for (byte j = 1; j <= cLoc; j++)
      mRut[i][j] = 0;
} //InicRut

byte BusBinVec(tvLoc vLoc, str20 cLoc, byte ult) {
  byte pri = 1,
       med;

  while (pri <= ult) {
     med = (pri + ult) / 2;
     if (!strcmp(vLoc[med],cLoc))
         return med;
     else
         if (strcmp(vLoc[med],cLoc) < 0)
             pri = med + 1;
         else
             ult = med - 1;
  }
  return 0;
} // BusBinVec

void GenRutas(FILE *Rut, tmRut mRut, tvLoc vLoc, byte cLoc) {
  sRut rRut;
  byte pos_i,
       pos_j;

  InicRut(mRut,cLoc);
  while (readf(Rut,rRut)) {
    pos_i = BusBinVec(vLoc,rRut.nomLoc1,cLoc);
    pos_j = BusBinVec(vLoc,rRut.nomLoc2,cLoc);
    if (pos_i > pos_j) {
      mRut[pos_i][pos_j] = rRut.dist;
      mRut[pos_j][pos_i] = rRut.nroRuta;
    }
    else {
      mRut[pos_j][pos_i] = rRut.dist;
      mRut[pos_i][pos_j] = rRut.nroRuta;
    }
  }
 } //GenRutas
```

```
void EmiteMat(tmRut mRut, tvLoc vLoc, byte cLoc) {
  char nLoc[5];

  //Matriz Triangular Superior indica Nro.de Rutas entre 2 Localidades
  //Matriz Triangular Inferior indica Dist. entre 2 Localidades
  cout << setw(21) << "";
  for (byte i = 1; i <= cLoc; i++) {
    strncpy(nLoc,vLoc[i],4);
    nLoc[4] = '\0';
    cout << setw(4) << nLoc << ' ';
  }
  cout << endl;
  for (byte i = 1; i <= cLoc; i++) {
    cout << setw(20) << vLoc[i];
    for (byte j = 1; j <= cLoc; j++)
      cout << setw(5) << mRut[i][j];
    cout << endl;
  }
} //EmiteMat

void ObtLocal(byte &d1, byte &d2, tvLoc vLoc, byte cLoc) {
  str20 nLoc1,
        nLoc2;
  char  *pcad;

  do {
    gotoxy(5,15); clreol();
    cout << "Ing.Nom.Loc.1: ";
    strupr(gets(nLoc1));
    d1 = BusBinVec(vLoc,nLoc1,cLoc);
  }
  while (d1 == 0);
  do {
    gotoxy(5,17); clreol();
    cout << "Ing.Nom.Loc.2: ";
    strupr(gets(nLoc2));
    d2 = BusBinVec(vLoc,nLoc2,cLoc);
  }
  while (d2 == 0);
} //ObtLocal

void EncontroLoc(word &dist, word &locInt, str20 mnsj, bool &existe, word km1,
                 word km2, word x, word &ckm1, word &ckm2) {

  dist = km1 + km2;
  ckm1 = km1;
  ckm2 = km2;
  strcpy(mnsj,"Dist. pasando por ");
```

```
  locInt = x;
  existe = true;
  cout << "Encontro la Localidad... " << endl;
} //EncontroLoc

void CalcDist(tmRut mRut, byte cLoc, byte i, byte j, word &dist,
              word &locInt, str20 mnsj, word &km1, word &km2) {
  bool existe;
  byte pos_j,
       k,
       aux;

  if (j > i)
    IntCmb(i,j);
  dist = locInt = 0;
  strcpy(mnsj,"No hay ruta");
  if (mRut[i][j]) {
    dist = mRut[i][j];
    strcpy(mnsj,"Distancia directa ");
  }
  else {
    existe = false;
    cLoc++;
    k = i + 1;
    pos_j = 1;
    while ((i > pos_j || cLoc > k) && !existe) {
      if (i > pos_j)
      //Recorrer fila mientras fila i > columna j
        if (mRut[i][pos_j])
          if (pos_j <= j)
            if (mRut[pos_j][j])

EncontroLoc(dist,locInt,mnsj,existe,mRut[i][pos_j],mRut[j][pos_j],pos_j,km1,km2);
            else
              pos_j++;
          else
            if (mRut[j][pos_j])

EncontroLoc(dist,locInt,mnsj,existe,mRut[i][pos_j],mRut[pos_j][j],pos_j,km1,km2);
            else
              pos_j++;
        else
          pos_j++;
      else  // ( i < j )
        //Recorrer Columna j desde fila i + 1 hasta ultima fila
        if (mRut[k][pos_j])
          if (mRut[k][j])
            EncontroLoc(dist,locInt,mnsj,existe,mRut[k][pos_j],mRut[k][j],k,km1,km2);
          else
            k++;
```

```
      else
        k++;
    }
  }
} //CalcDist

void EmiteDist(word dist, word locInt, str20 mnsj, tvLoc vLoc, word km1, word km2) {

  if (!dist)
    cout << mnsj << " hasta un nivel";
  else
    if (!locInt)
      cout << mnsj << " es: " << dist;
    else
      cout << mnsj << vLoc[locInt] << ' ' << km1 << " + " << km2 << " es: " << dist;
  getch();
} //EmiteDist

void Cerrar(FILE *Loc, FILE *Rut) {

  fclose(Loc);
  fclose(Rut);
} //Cerrar

main() {
  FILE  *Localidades,
        *Rutas;
  byte  i,
        j,
        cantLocal = 0;
  word  localInt,
        distan,
        dist1,
        dist2;
  str20 mens;
  tvLoc vLocal;
  tmRut mRutas;

  Abrir(&Localidades,&Rutas);
  GenLocal(Localidades,vLocal,cantLocal);
  GenRutas(Rutas,mRutas,vLocal,cantLocal);
  EmiteMat(mRutas,vLocal,cantLocal);
  ObtLocal(i,j,vLocal,cantLocal);
  CalcDist(mRutas,cantLocal,i,j,distan,localInt,mens,dist1,dist2);
  EmiteDist(distan,localInt,mens,vLocal,dist1,dist2);
  Cerrar(Localidades,Rutas);
  return 0;
} // main
```

Codificación G4Ej17 Rutas Argentinas versión stream

```
/*
   Id.Programa: G4Ej17Rutas.cpp
   Autor......: Lic. Hugo A. Cuello
   Fecha......: septiembre-2014
   Comentario.: Consultas de Localidades.

   *Estructuras de Datos

   Vector de Localidades con nomLoc (str20) CON MAX_LOC ord. ascendente.
   Matriz cuadrada de Rutas con Distancias_NroRutas con MAX_LOC:
     Inicializar en cero la matirz con la cantidad de Localidades canLoc.
     procesadas en el archivo de Localidades.
     Distancias en la triangular inferior (i > j)
     Nros. de Rutas en la triangular superior (i < j)
*/

#include<conio.h>
#include<iomanip>
#include<fstream>
using namespace std;

const
  MAX_LOC = 100;

typedef char str20[21];
typedef unsigned char byte;
typedef unsigned short word;

struct sRut {
  str20 nomLoc1,
        nomLoc2;
  word  dist,
        nroRuta;
};

typedef str20 tvLoc[MAX_LOC + 1]; //tvLoc es el vector de Localidades, pos. 0 no se
usa
typedef word  tmRut[MAX_LOC + 1][MAX_LOC + 1]; //tmRut es la matriz de Rutas
                                //(Tri.Inf.: Dist., Tri.Sup.: NroRuta) pos. 0 no se usa

// Prototipos ------------------------------
void Abrir(ifstream &, ifstream & );
void IntCmb(str20, str20 );
void IntCmb(byte &, byte & );
void OrdxBur(tvLoc, byte );
void GenLocal(ifstream &, tvLoc, byte & );
void InicRut(tmRut, byte );
byte BusBinVec(tvLoc, str20, byte );
```

```
void GenRutas(ifstream &, tmRut, tvLoc, byte );
void EmiteMat(tmRut, tvLoc, byte );
void ObtLocal(byte &, byte &, tvLoc, byte );
void EncontroLoc(word &, word &, str20, bool &,word, word, word );
void CalcDist(tmRut, byte, byte, byte, word &, word &, str20, word &, word &  );
void EmiteDist(word, word, str20, tvLoc, word, word );
void Cerrar(ifstream &, ifstream & );
// Fin Prototipos -------------------------

void Abrir(ifstream &Loc, ifstream &Rut) {
  Loc.open("Localidades.Dat",ios::binary);
  Rut.open("Rutas.Dat",ios::binary);
} //Abrir

void IntCmb(str20 elem1, str20 elem2) {
  str20 aux;

  strcpy(aux,elem1);
  strcpy(elem1,elem2);
  strcpy(elem2,aux);
} // IntCmb

void IntCmb(byte &elem1, byte &elem2) {
  byte aux;

  aux = elem1;
  elem1 = elem2;
  elem2 = aux;
} // IntCmb

void OrdxBur(tvLoc vLoc, byte card) {
  bool ordenado;
  byte k = 0;

  do {
    ordenado = true;
    k++;
    for (byte i = 1; i <= card - k; i++)
      if (strcmp(vLoc[i],vLoc[i + 1]) > 0) {
        ordenado = false;
        IntCmb(vLoc[i],vLoc[i + 1]);
      }
  }
  while (!ordenado);
} // OrdxBur

void GenLocal(ifstream &Loc, tvLoc vLoc, byte &cLoc) {

  cLoc = 0;
```

```
  while (Loc.read((char *) vLoc[++cLoc], sizeof vLoc[0]));
  cLoc--;
  OrdxBur(vLoc,cLoc);
 } //GenLocal

void InicRut(tmRut mRut, byte cLoc) {

  for (byte i = 1; i <= cLoc; i++)
    for (byte j = 1; j <= cLoc; j++)
      mRut[i][j] = 0;
} //InicRut

byte BusBinVec(tvLoc vLoc, str20 cLoc, byte ult) {
  byte pri = 1,
       med;

  while (pri <= ult) {
    med = (pri + ult) / 2;
          if (!strcmp(vLoc[med],cLoc))
            return med;
  else
            if (strcmp(vLoc[med],cLoc) < 0)
                    pri = med + 1;
    else
                      ult = med - 1;
  }
  return 0;
} // BusBinVec

void GenRutas(ifstream &Rut, tmRut mRut, tvLoc vLoc, byte cLoc) {
  sRut rRut;
  byte pos_i,
       pos_j;

  InicRut(mRut,cLoc);
  while (Rut.read((char *) &rRut, sizeof rRut)) {
    pos_i = BusBinVec(vLoc,rRut.nomLoc1,cLoc);
    pos_j = BusBinVec(vLoc,rRut.nomLoc2,cLoc);
    if (pos_i > pos_j) {
      mRut[pos_i][pos_j] = rRut.dist;
      mRut[pos_j][pos_i] = rRut.nroRuta;
    }
    else {
      mRut[pos_j][pos_i] = rRut.dist;
      mRut[pos_i][pos_j] = rRut.nroRuta;
    }
  }
 } //GenRutas

void EmiteMat(tmRut mRut, tvLoc vLoc, byte cLoc) {
```

```
  char nLoc[5];

  //Matriz Triangular Superior indica Nro.de Rutas entre 2 Localidades
  //Matriz Triangular Inferior indica Dist. entre 2 Localidades
  cout << setw(21) << "";
  for (byte i = 1; i <= cLoc; i++) {
    strncpy(nLoc,vLoc[i],4);
    nLoc[4] = '\0';
    cout << setw(4) << nLoc << ' ';
  }
  cout << endl;
  for (byte i = 1; i <= cLoc; i++) {
    cout << setw(20) << vLoc[i];
    for (byte j = 1; j <= cLoc; j++)
      cout << setw(5) << mRut[i][j];
    cout << endl;
  }
} //EmiteMat

void ObtLocal(byte &d1, byte &d2, tvLoc vLoc, byte cLoc) {
  str20 nLoc1,
        nLoc2;
  char  *pcad;

  do {
    gotoxy(5,15); clreol();
    cout << "Ing.Nom.Loc.1: ";
    strupr(gets(nLoc1));
    d1 = BusBinVec(vLoc,nLoc1,cLoc);
  }
  while (d1 == 0);
  do {
    gotoxy(5,17); clreol();
    cout << "Ing.Nom.Loc.2: ";
    strupr(gets(nLoc2));
    d2 = BusBinVec(vLoc,nLoc2,cLoc);
  }
  while (d2 == 0);
} //ObtLocal

void EncontroLoc(word &dist, word &locInt, str20 mnsj, bool &existe, word km1,
                 word km2, word x, word &ckm1, word &ckm2) {

  dist = km1 + km2;
  ckm1 = km1;
  ckm2 = km2;
  strcpy(mnsj,"Dist. pasando por ");
  locInt = x;
  existe = true;
```

```
  cout << "Encontro la Localidad... " << endl;
} //EncontroLoc

void CalcDist(tmRut mRut, byte cLoc, byte i, byte j, word &dist,
              word &locInt, str20 mnsj, word &km1, word &km2) {
  bool existe;
  byte pos_j,
       k,
       aux;

  if (j > i)
    IntCmb(i,j);
  dist = locInt = 0;
  strcpy(mnsj,"No hay ruta");
  if (mRut[i][j]) {
    dist = mRut[i][j];
    strcpy(mnsj,"Distancia directa ");
  }
  else {
    existe = false;
    cLoc++;
    k = i + 1;
    pos_j = 1;
    while ((i > pos_j || cLoc > k) && !existe) {
      if (i > pos_j)
      //Recorrer fila mientras fila i > columna j
        if (mRut[i][pos_j])
          if (pos_j <= j)
            if (mRut[pos_j][j])

EncontroLoc(dist,locInt,mnsj,existe,mRut[i][pos_j],mRut[j][pos_j],pos_j,km1,km2);
            else
              pos_j++;
          else
            if (mRut[j][pos_j])

EncontroLoc(dist,locInt,mnsj,existe,mRut[i][pos_j],mRut[pos_j][j],pos_j,km1,km2);
            else
              pos_j++;
        else
          pos_j++;
      else  // ( i < j )
        //Recorrer Columna j desde fila i + 1 hasta ultima fila
        if (mRut[k][pos_j])
          if (mRut[k][j])
            EncontroLoc(dist,locInt,mnsj,existe,mRut[k][pos_j],mRut[k][j],k,km1,km2);
          else
            k++;
        else
          k++;
```

```
    }
  }
} //CalcDist

void EmiteDist(word dist, word locInt, str20 mnsj, tvLoc vLoc, word km1, word km2) {

  if (!dist)
    cout << mnsj << " hasta un nivel";
  else
    if (!locInt)
      cout << mnsj << " es: " << dist;
    else
      cout << mnsj << vLoc[locInt] << ' ' << km1 << " + " << km2 << " es: " << dist;
  getch();
} //EmiteDist

void Cerrar(ifstream &Loc, ifstream &Rut) {

  Loc.close();
  Rut.close();
} //Cerrar

main() {
  ifstream Localidades,
               Rutas;
  byte  i,
        j,
        cantLocal = 0;
  word  localInt,
        distan,
        dist1,
        dist2;
  str20 mens;
  tvLoc vLocal;
  tmRut mRutas;

  Abrir(Localidades,Rutas);
  GenLocal(Localidades,vLocal,cantLocal);
  GenRutas(Rutas,mRutas,vLocal,cantLocal);
  EmiteMat(mRutas,vLocal,cantLocal);
  ObtLocal(i,j,vLocal,cantLocal);
  CalcDist(mRutas,cantLocal,i,j,distan,localInt,mens,dist1,dist2);
  EmiteDist(distan,localInt,mens,vLocal,dist1,dist2);
  Cerrar(Localidades,Rutas);
  return 0;
} // main
```

Unidad II Capítulo V

Unidad II – Capítulo V - La clase vector de C++

#include <vector>

Definición del tipo vector

vector<tipoDato> IdentificadorVar (cantElem, valInic);

(cantElem, valInic) su uso es opcional.

Ejemplo

```
vector<int> vEnteros;
struct tAlumno {
  long nroLeg;
  char apeNom[21],
        domic[21],
        local[21];
       char sexo;
       bool trabaja;
       tFecha fNac;
      float cuota;
} UnAlumno;
```

Definición de un vector de registros:

vector<tAlumno> vrAlumnos(100);

Iteradores

begin()

end()

El siguiente ejemplo recorre secuencialmente el arreglo myarray desde el primer elemento indicado por begin() 2 hasta el último elemento indicado por end() 50. La variable it es el puntero que se desplaza en cada pasada del ciclo for entre los límites indicados anteriormente.

```
// array::begin example
#include <iostream>
// Clase vector
#include <iostream>
#include <vector>
using namespace std;

main () {
```

```
  vector<int>vec(5,0);// (cant,valIni)su uso es opcional
  for (int i=1; i <= 5; i++)
    vec.push_back(random(90)+10);
  cout << "El vector vec contiene: ";
  for(vector<int>::iterator it=vec.begin();it!=vec.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';
  return 0;
}
```

En este otro ejemplo emite los valores del vector desde 2 hasta 4 y no desde 1 hasta 5. ¿por qué?.

```
// vector::begin()/end(), iterator, push_back()
#include <iostream>
#include <vector>
using namespace std;
main () {
 vector<int> myvector;
 for (int i=1; i<=5; i++)
  myvector.push_back(i);

 cout << "myvector contains:";
 for (vector<int>::iterator it = myvector.begin() + 1 ; it != myvector.end() - 1; ++it)
  cout << ' ' << *it;
 cout << '\n';
 return 0;
}
```

Capacidad

size
std::vector<(int) aEnteros;
std::cout << aEnteros.**size**() << std::endl; // Emite: 5

Acceso a elementos:

operador [ ] // Accede a un elemento.
at // Idem anterior.
front // Accede al primer elemento.
back // Accede al último elemento.
std::vector<int> vEnt;
for(short i = 0; i <=5; i++) vEnt[i] = random(90) + 10;
std::cout << vEnt[0]; // Emite:
std::cout << vEnt[1]; // Emite:
std::cout << vEnt.**at**(2); // Emite:
std::cout << vEnt.**front**(); // Emite:
std::cout << vEnt.**back**(); // Emite:

El lector podrá buscar una lista completa de las funciones miembro de la clase vector consultando en la internet escribiendo por ejemplo en el buscador, clase vector c++. Algunos sitios muy importante son: cplusplus.com, c.conclase.net.

https://cplusplus.com/reference/vector/vector/?kw=vector

U.T.N. F.R.B.A. Algoritmos y Estructuras de Datos Prof.: Hugo A. Cuello
Guía de ejercicios Nro.2.
Estructuras estáticas I. Cadenas, Registros, Arreglos.

Guía de ejercicios #2

Ejercicios con tratamiento de cadenas de caracteres.

1. Crear la función **char** * *Replicate*(**char** car, **short** n); que retorna el carácter dado n veces.
2. Crear la función **char** * *FillBlank*(char * cad, **short** n); que rellena la cadena cad de longitud física n, con espacios en blanco a derecha.
3. Crear la función **char** *LowCase*(**char** car) : char; que convierte el carácter dado a minúscula, si está en mayúscula, caso contrario retorna el mismo carácter.
4. Crear la función **char** * *InvCad*(**char** * cad; que invierte una cadena.
5. Crear la función **void** *CmbCar*(**char** * cad, **char** car1, **char** car2); que reemplaza cada aparición del car1 en cad, por el car2.

Ejercicios con Arreglos.

6. Se efectúa la inscripción de los alumnos de primer año, de acuerdo al orden en que presentaron sus solicitudes. Por cada alumno se confeccionó un registro en el archivo de **Ingresantes**, conteniendo: Legajo (5 car.), Apellido y Nombre (30 car.), División elegida (1..50). Tras conocerse la cantidad de vacantes existentes en las N divisiones, se procede a tratar los datos mencionados. Mientras haya vacantes, a cada solicitud de un alumno, se le asignará la división elegida. Una vez completado el tratamiento inicial, y si han quedado solicitudes sin satisfacer (es decir, no quedan vacantes en las divisiones elegidas), se deberán procesar nuevamente estos datos para asignarles divisiones en las que si haya disponibilidad. Finalmente, se deberá emitir las vacantes sobrantes en cada división o los legajos de los alumnos que han quedado sin asignación de división, según corresponda. Se sabe que no hay más de 3000 solicitudes.
7. Obtener un vector de N elementos, para calcular y emitir según un menú lo siguiente:
   a) El promedio.
   b) El máximo.
   c) El mínimo.
   d) La sumatoria.
8. Generar un vector con N elementos en forma al azar sin repetir en el intervalo de 1 a 90.
   Caso 1: simulando sacar una bolilla cantarla y meterla en la misma bolsa.
   Caso 2: simulando sacar una bolilla cantarla y apartarla.
9. Generar dos vectores p y q con valores enteros en el intervalo 1 a 90, **sin repetir** de m y n elementos respectivamente. Ordenarlos creciente en el mismo arreglo.

   Se pide :

   Crear un menú con las opciones indicadas más abajo.
   Generar un tercer vector r, que es el resultado de :
   a) La Unión de p y q.
   b) La Intersección de p y q.
   c) La Diferencia de p y q.
   d) La Diferencia Simétrica de p y q.
   e) El Complemento de p + q.

10. Dados dos vectores p y q de cardinalidad m y n respectivamente, generarlos con valores enteros al azar, máximo 90 elementos, luego ordenarlos en forma creciente. Por último intercalarlos en un tercer vector r. Emitir los tres vectores.

11. Quince tiradores participan en un campeonato de tiro al blanco, efectuando cada uno de ellos tres disparos. Como datos se ingresan 15 registros desordenados, cada uno de los cuales contiene : Número del tirador (1..15), y tres puntajes (real 0..10) correspondientes a los tres disparon realizados. La suma de los 3 puntajes es el puntaje total del respectivo tirador. Se pide, emitir la clasificación del campeonato ordenada por puntaje total y de la siguiente manera :

Posic.	**Tirador**	**Puntaje**
1	99	99.99
2	99	99.99

12. Una empresa discográfica realiza una encuesta sobre las preferencias musicales del público. Por cada encuestado se ha preparado un registro en el que se indica : Sexo ('F', 'M'), Edad (15..50), 3 temas preferidos (cada uno: 1..30).

Se pide :

a) Emitir la lista ordenada, según preferencias, de los códigos de los 30 temas musicales que han sido encuestados.
b) El código del tema más votado por los hombres de 30 años o más.
c) Los códigos de los temas que no fueron elegidos por ninguna mujer.

13. Una empresa tiene una flotilla de máximo 100 camiones que realizan viajes al interior del país transportando mercaderías muy frágiles. Por esta razón, los directivos de la empresa han decidido otorgar una bonificación a todos los camioneros cuya cantidad de roturas en el último mes haya sido inferior a 20 unidades.

    Para ello se cuenta con un archivo de **Viajes** sin orden y con repetición de camioneros, conteniendo : Nro. Leg. (5 car.), Cód. Pcia. (1..24), Cantidad de unidades rotas.

    Únicamente tendrán derecho a la bonificación aquellos conductores que hayan realizado un mínimo de cuatro viajes. El listado a emitir ordenado por cantidad de unidades rotas, -se deben emitir todos los camioneros con al menos un viaje realizado- deberá contener:

Listado de Camioneros ordenado por Cant. unid. rotas

Nro. Leg.	**Cant. Viajes**	**Cant. Unid. Rotas**
X(5)	999	9999

14. Al finalizar el campeonato Apertura-Clausura de la 1ra. División A de la A.F.A. se requiere emitir la tabla de posiciones. Para ello se cuenta con un archivo de partidos jugados en todo el campeonato sin orden, conteniendo : Nombre del Equipo Local (20 car.), Cantidad de goles (0..9), Nombre del Equipo Visitante, Cantidad de goles, (elija Ud. el mejor orden de los campos del registro de este archivo).

    Se pide :

    a) Ingresar por teclado la leyenda del campeonato finalizado **Apertura** o **Clausura**.
    b) Emitir la tabla de posiciones ordenado decreciente por el campo puntaje
    c) Emitir la tabla de posiciones ordenado decreciente por el campo puntaje y diferencia de goles.

    En ambos casos el diseño de salida es :

Campeonato X(8) de la A.F.A.

Equipos	**PJ**	**PG**	**PE**	**PP**	**GF**	**GC**	**PTS**
X(20)	99	99	99	99	99	99	99

15. Una fábrica de calzados elabora 7 modelos en 5 colores diferentes. Esta fábrica tiene registradas las ventas realizadas por cada modelo y por cada color en un archivo de ventas, conteniendo : Nro. modelo (1..7), Color ('A'..'E'), Cantidad vendida (3 díg.).

Se pide : Emitir un listado que informe los totales de cada uno de los modelos y por cada uno de los colores, como así también los totales por cada modelo, los totales por cada color y el total general.

16. Dada una matriz cuadrada con n = 5, en el cual las posiciones indicadas poseen los siguientes valores:
    $a_{2,1}$ = 8; $a_{1,4}$ = 12; $a_{3,1}$ = 1; $a_{3,4}$ = 20; $a_{4,3}$ = 18; $a_{5,2}$ = 4; completar las posiciones restantes de acuerdo al movimiento de un caballo del tablero de ajedrez; por supuesto, respetando los valores prefijados.

17. Cargar una matriz cuadrada con n = k, en donde k es impar, de tal manera que la suma de cada fila, de cada columna y de cada diagonal (primaria y secundaria), dan el mismo valor. Para resolver este problema y en caso que la fila inicial sea 1 y la columna inicial también sea 1, aplicar las siguientes reglas:

    (1) El valor 1 se ubica en la fila superior (i = 1), columna central (j = n / 2 + 1).

    (2) El siguiente valor de la secuencia 2 se ubica en la fila (i – 1), columna (j + 1).

    (3) En caso que la fila no exista, se retoma por (i = n), columna (j) según (2).

    (4) En caso de la columna no exista, se retoma por (j = 1), fila (i) según (2).

    (5) En caso que la celda esté ocupada, la fila es (i + 2) columna (j – 1). ¿por qué?

    (6) En caso que la fila y/o columna no exista (ángulo superior derecho) aplicar la regla (5).

18. Se cuenta con un archivo de **Localidades** desordenado, máximo 100 registros, conteniendo las localidades (cad20) y de un archivo de **Rutas**, sin orden, para cada par de ciudades unidas por una ruta –sin repetir el par, vale decir, si las ciudades fueran a y b, solo estará el par a-b o el par b-a, solo uno-, conteniendo: Nombres del par de localidades (cad20, para c/u.), el número de ruta que las une (short) y la distancia (short).

    Se pide:

    1. Volcar a memoria interna los datos de ambos archivos.
    2. Responder a diversas consultas, como ser:
        a. Dadas dos localidades, informar si están unidas en forma directa, su distancia y nro. de ruta.
        b. Dada una localidad, informar con cuales localidades está unida en forma directa.
        c. Dada una localidad y una distancia, informar las localidades unidas en forma directa, pero que están dentro del radio dado.
        d. Dadas dos localidades, buscar un nexo, si no están unidas en forma directa, informando, distancia entre cada ciudad –incluye el nexo-, nros. de rutas y el nombre de la ciudad nexo.
        e. Dada una localidad, informar la localidad más cercana unida directamente, su distancia y nro. de ruta.
        f. Elabore ud. 5 preguntas adicionales.

    Recurso disponible: Memoria estática disponible para arreglos: la más adecuada. Optimizar la búsqueda de localidades, utilizando métodos de ordenamiento y búsqueda binaria.

Ejercicio adicional

Normalizar un vector de longitud 1, utilizado para simplificar los cálculos, no modifica la dirección solo la longitud (módulo).

Longitud (módulo) del vector: $\begin{pmatrix} 5 \\ 12 \\ 7 \end{pmatrix} = \sqrt{5^2 + 12^2 + 7^2} = 14{,}765$

Luego para Normalizar el vector es multiplicarlo por $\frac{1}{14{,}765} \cdot \begin{pmatrix} 5 \\ 12 \\ 7 \end{pmatrix} = \begin{pmatrix} 0{,}339 \\ 0{,}813 \\ 0{,}474 \end{pmatrix}$

Unidad III - Estructura de Datos Conjunto Capítulo I

CONJUNTOS

Diagramas de Venn o esquema de Euler

Permiten representar de manera gráfica resultados de las operaciones del álgebra de conjuntos.

Las operaciones básicas entre conjunto son

UNIÓN

A ∪ B se lee A unión B **{∀x / x ∈ A v ∀x / x ∈ B}**

INTERSECCIÓN

A ∩ B se lee A intersección B **{∀x / x ∈ A ^ ∀x / x ∈ B}**

DIFERENCIA

A - B se lee A diferencia con B **{∀x / x ∈ A ^ ∀x / x ∉B}**

DIFERENCIA SIMÉTRICA

A Δ B se lee A diferencia simétrica con B

$\{\forall x / x \in A \wedge \forall x / x \notin B\} \vee \{\forall x / x \in B \wedge \forall x / x \notin A\}$

COMPLEMENTO

~A se lee complemento de A $\{\forall x / x \in \mathcal{U} \wedge \forall x / x \notin A\}$

Relaciones entre conjuntos

A ⊆ B A incluido en B, A es un subconjunto de B, A es una parte de B.
si y solo si, para todo elemento x que pertenezca al conjunto A, entonces también pertenece al conjunto B.

$\forall x : (x \in A \Rightarrow x \in B)$

B ⊇ A B incluye a A

$$A \subset B \Leftrightarrow \begin{cases} \forall x \in A \Leftrightarrow x \in B \\ \exists y \in B \Leftrightarrow y \notin A \end{cases}$$

A ⊂ B se lee subconjunto propio.

La observación es que si $A \subset B \Rightarrow A \neq B$

A = B si tienen ambos conjuntos los mismos elementos.

$A \subset B \wedge B \subset A$

$\{x \in A \Rightarrow x \in B \wedge x \in B \Rightarrow x \in A\}$

A ≠ B $\{$**si algún** $x \in A \Rightarrow x \notin B$ **v si algún** $x \in B \Rightarrow x \notin A\}$

Pertenencia de un elemento en un conjunto

$x \in A$

{} es el conjunto vacío = ∅

A = {x / x es número par y número impar} = ∅

B = {x / x ≠ x} = ∅

~ $\mathcal{U}$ = ∅

$\mathcal{U}$ es el conjunto universal
$\sim\varnothing = \mathcal{U}$

Conjuntos disjuntos

Si ningún elemento de A está en B y ningún elemento de B está en A entonces A y B son disjuntos.

Si $\forall x / x \in A \wedge \forall x / x \notin B$
Si $\forall x / x \in B \wedge \forall x / x \notin A$

Leyes de las operaciones de Conjuntos

Ley	Expresión
Idempotencia	a) $A \cup A = A$ b) $A \cap A = A$
Asociativas	a) $(A \cup B) \cup C = A \cup (B \cup C)$ b) $(A \cap B) \cap C = A \cap (B \cap C)$
Conmutativas	a) $A \cup B = B \cup A$ b) $A \cap B = B \cap A$
Distributivas	a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ b) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identidad	a) $A \cup \varnothing = A$ b) $A \cup \mathcal{U} = \mathcal{U}$ c) $A \cap \mathcal{U} = A$ d) $A \cap \varnothing = \varnothing$
Complemento	a) $A \cup \sim A = \mathcal{U}$ b) $\sim\sim A = A$ c) $A \cap \sim A = \varnothing$ d) $\sim\mathcal{U} = \varnothing$ e) $\sim\varnothing = \mathcal{U}$
Leyes de De Morgan	a) $\sim(A \cup B) = \sim A \cap \sim B$ b) $\sim(A \cap B) = \sim A \cup \sim B$

Unidad III Capítulo II

Unidad III – Capítulo II - Conjuntos en Turbo Pascal

Tipo de dato estructurado: set of de Turbo Pascal

Turbo Pascal es uno de los pocos lenguajes de computadoras que brinda una cantidad de herramientas para las operaciones entre **conjuntos**. Dada la importancia que tiene este concepto no solo para la matemática sino también para otras disciplinas, a continuación se tratará este tema.

Un conjunto es una colección de elementos bien definidos. En el caso de Turbo Pascal, esos elementos deben ser todos del mismo tipo y ese tipo debe ser *ordinal*. La cantidad máxima de elementos está limitada a 256 y los valores ordinales estarán comprendido en el intervalo 0 a 255. El tipo conjunto permite manipular con un solo nombre una serie de valores o elementos, siendo estos, las componentes de la estructura conjunto. La manera en que Turbo Pascal determina si un elemento pertenece o no al conjunto, es a través de un **enmascaramiento** a nivel de bit. En principio si un bit está en 0 significa que el elemento no está presente y si un bit está en 1 significa que el elemento sí está presente. Una estructura de tipo conjunto ocupará una cantidad n de bytes, en donde n depende de los valores extremos del tipo base del conjunto. La máxima cantidad de bytes será de 32, ya que 32 x 8 = 256, de esta manera por cada bit se podrá determinar si un elemento de entre 256 máximos estará o no presente en el conjunto. Las siguientes expresiones determinan:

Determina	Fórmula
Cantidad de bytes a usar por la estructura	*(MAX **div** 8) - (MIN **div** 8) + 1*
Número de byte de un valor x	*(x **div** 8) - (MIN **div** 8)*
Número de bit de un valor x	*x **mod** 8*

en donde:

MIN es el valor extremo inferior.
MAX es el valor extremo superior.
x representa un valor del tipo base

Representación interna en una estructura de datos de tipo conjunto

A continuación se considera el siguiente caso:

1. un tipo base en el intervalo 0 a 15
2. un conjunto A = {2,5,0,12,8,15,9,11}

MIN = 0
MAX = 15
Cantidad de bytes a usar por la estructura: 15 div 8 – MIN div 8 + 1 = 2

Número de byte de un valor x= 2: 2 div 8 – 0 div 8 = 0
Número de bit de un valor x=2: 2 mod 8 = 2

Número de byte de un valor x= 5: 5 div 8 – 0 div 8 = 0
Número de bit de un valor x=5: 5 mod 8 = 5

Número de byte de un valor x= 0: 0 div 8 – 0 div 8 = 0
Número de bit de un valor x=0: 0 mod 8 = 0

Número de byte de un valor x=12: 12 div 8 – 0 div 8 = 1
Número de bit de un valor x=12: 12 mod 8 = 4

Número de byte de un valor x= 8: 8 div 8 – 0 div 8 = 1
Número de bit de un valor x=8: 8 mod 8 = 0

Número de byte de un valor x=15: 15 div 8 – 0 div 8 = 1
Número de bit de un valor x=15: 15 mod 8 = 7

Número de byte de un valor x= 9: 9 div 8 – 0 div 8 = 1
Número de bit de un valor x=9: 9 mod 8 = 1

Número de byte de un valor x=11: 11 div 8 – 0 div 8 = 1
Número de bit de un valor x=11: 11 mod 8 = 3

Un bit en 0, en un número de byte y de bit correspondiente, indica que el número no pertenece al conjunto. En cambio un bit en 1, en un número de byte y de bit correspondiente, indica que el número pertenece al conjunto. De esta manera en la representación gráfica de más arriba está indicando que los elementos 2, 5, 7, 8, 11, 12, 14 y 15 pertenecen al conjunto A.

En la sección **type** primero vamos a definir un tipo conjunto por medio de las palabras reservadas *set of* y a continuación el tipo base de cada elemento del conjunto, de la siguiente manera:

type
  tCjto = **set of** *tipo_base*;

El *tipo_base* debe ser de tipo **ordinal**, pero no cualquier ordinal, solamente aquellos cuyos valores ordinales estén comprendidos en *(0; 255)*. Por lo tanto, no solo los reales y las cadenas quedan excluidos, sino también el tipo integer, shortint, longint, word. Así que, los tipos definidos por Turbo Pascal han de ser el byte, char, boolean y aquellos definidos por el usuario por enumeración y por intervalo.

Ejemplos de tipos bases:

Por comprensión **Por extensión o enumeración**

tipoBase = {x / x es número par ^ 25 < x < 71} = {26, 28, 30, 32, ..., 70}
tipoBase = {x / x es letra del alfabeto inglés} = {‘A’,’B’,’C’, ...’Z’,’a’,’b’,’c’,...’z’}
tipoBase = {x / x es dígito decimal} = {0,1,2, ..., 9}
tipoBase = {x / x es número primo ⇔ x|±x ^ x|±1} = {±2,±5,±7,±11,±13,±17,±19, ...}

```
type
  Cjto = set of byte;
```

Se ha definido un tipo *Cjto* que es igual a set of byte, en donde **set of** establece el tipo conjunto y *byte* dice que los elementos son todos del tipo base byte.

```
type
  tFelinos = (leon,tigre,gato,lince,jaguar,guepardo,pantera);
  tAlgunFelino = gato..guepardo;
  tFieras = set of tFelinos;
  tAlgunaFiera = set of tAlgunFelino;
```

Los tipos definidos por el usuario no pueden ser leídos o emitidos por medio de los procedimientos *read* o *write*, debido a que estos valores no son conocidos por anticipado. Una forma de evitar este inconveniente es utilizar una estructura de selección múltiple, como se indica a continuación:

```
Case random(7) of
    0: Felix:= leon;
    1: Felix:= tigre;
    2: Felix:= gato;
    3: Felix:= lince;
    4: Felix:= jaguar;
    5: Felix:= guepardo;
    6: Felix:= pantera
end;
Case Felix of
  leon: writeln(‘leon’);
  tigre: writeln(‘tigre’);
  gato: writeln(‘gato’);
  lince: writeln(’lince’);
  jaguar: writeln(‘jaguar’);
  guepardo: writeln(‘guepardo’);
  pantera: writeln(‘pantera’)
end;
```

```
var
  Cfieras        : tFieras;
  CalgunaFiera : tAlgunaFiera;
```

Los tipos definidos por el usuario por enumeración o por intervalo son ordinales.

tFelinos es un tipo de dato definido por el usuario, siendo sus elementos valores constantes y no pueden repetirse en otra definición. Cada elemento ocupa una posición ordinal comenzando desde 0 y siguiendo con incremento de 1. Por lo tanto, leon = 0, tigre = 1, gato = 3 y así sucesivamente.

Esto permite entonces establecer una relación leon < tigre < gato < ... < pantera.

tAlgunFelino es un tipo también definido por el usuario en la que se establece indicando un valor que indica el extremo inferior y otro valor que indica el extremo superior.

Cfieras es una variable de tipo *tFieras*, por lo tanto, los valores que podrá contener son elementos del tipo base *tFelinos*, es decir, *leon, tigre, gato...*

CalgunaFiera es una variable de tipo *tAlgunaFiera*, por lo tanto, los valores que podrá contener son elementos del tipo base *tAlgunFelino*, es decir, *gato, lince, jaguar, leopardo*.

- Las funciones ***Succ, Pred, Ord, y Chr*** pueden aplicarse a objetos de tipo <u>ordinal</u>.
- Las constantes de conjuntos se encierran entre corchetes y separadas por coma.

Ejemplos:

```
[3,9..12,5,31,15]
['a'..'z','A'..'Z']
['+','-','*','/']
```

También es posible que los elementos estén referidos por expresiones:

[12,a..b,i+2,ord('A')..ord('C'),x div 2,ord(cad[k])+2,Pred(b)+5..Succ(b)+5,2*i]

si a=5, b=8, x=52, i=15, k=3 y cad= 'algoritmos', entonces los elementos que pertenecen al conjunto son: 12, 5, 6, 7, 8, 17, 65, 66, 67, 26, 105, 12, 13, 14, 30 y con #14.

Una constante de conjunto especial es el **conjunto vacío {}** representado por [ ].

El **conjunto universal** $\mathcal{U}$ está formado por todos los elementos del tipo base del conjunto, p.e. si el tipo base es byte, entonces el $\mathcal{U}$ = [0..255].

Podemos asignarles un nombre a las constantes conjuntos en la sección const:

```
const
  CJTO_VACIO = [];
  CJTO_UNIVERSAL = [0..255]; son todos los elementos del tipo base.
  OPER_ARIT = ['+','-','*','/']
```

Operadores de conjuntos

+ para la unión entre conjuntos: A + B
* para la intersección entre conjuntos: A * B
- para la diferencia entre conjuntos: A – B

Las prioridades son:

1º. *
2º. +, -.

La diferencia simétrica se obtiene de la siguiente manera:

(A + B) – (A * B) o por
(A – B) + (B – A)
El complemento de A, indicado por **~A**, es igual al $\mathcal{U}$ **– A**, es decir:

~A = $\mathcal{U}$ **– A**

Turbo Pascal brinda además de los operadores + y – los procedimientos **Include** y **Exclude** para incluir o excluir un elemento de un conjunto.

Include(A,x)
Exclude(A,x)

Como la unión, la diferencia se refiere entre conjuntos, para incluir o excluir un elemento de un conjunto utilizando los operadores + o – debemos encerrar el elemento a incluir o excluir encerrado entre corchetes, de esta manera, lo estaremos transformando en conjunto, como se muestra a continuación:

A := A + [x]
A := A – [x]

Include(A,x) es más eficiente que hacer A:= A + [x]
Exclude(A,x) es más eficiente que hacer A:= A – [x]

La pertenencia $\in$ de un elemento en un conjunto se establece de la siguiente manera:

x **in** A el resultado será de tipo boolean si el elemento x pertenece al conjunto A el resultado es true, caso contrario es false.

La no pertenencia $\notin$ de un elemento en un conjunto se establece de la siguiente manera:

not (x **in** A)

El resultado de esta evaluación es verdadero si el valor que contenga x está en el conjunto, caso contrario será falso.

Ejemplo:

```
x:= 4;
A:= [2..7,20,12];
if x in A then
  writeln('el valor ',x,' pertenece al conjunto A)
else
  writeln('el valor ',x,' no pertenece al conjunto A');
```

Emite: el valor 4 pertenece al conjunto A. **¿por qué?**

Dos conjuntos pueden ser comparados por: =, <>, >=, <=.

Ejemplos

A = B

Si A = {7, 3, 9, 12} y B = {9, 3, 12, 7} entonces comparar si A = B es verdadero.

A <> B

Si A = {5, 2, 8} y B = {5, 2, 8, 6} entonces comparar si A <> B es verdadero.

A >= B es falso si A = {5, 2, 8} y B = {5, 2, 8, 6}

A <= B es verdadero

Si A = {} y B = {3, 1}

A <= B es verdadero

Para emitir los elementos que pertenecen a un conjunto debemos emplear una variable del tipo base y que recorra cada uno de los valores del conjunto universal y preguntar si pertenece al conjunto, como se muestra a continuación:

```
A:= [];
for x:= 1 to 100 do
  Include(A,2*x);
for x:= 0 to 255 do
    if x in A then
      write(x,' ');
```

La cardinalidad de un conjunto, indicado por #, es la cantidad de elementos que contiene. Para averiguar la cardinalidad de un conjunto en Turbo Pascal establecemos un ciclo como en el ejemplo anterior, pero contando con incremento en 1 en una variable cada vez que un elemento pertenece al conjunto, como se muestra a continuación:

```
card:= 0;
for x:= 0 to 255 do
    if x in A then
      inc(card);
writeln('#A= ',card);
```

Unidad III Capítulo III

Unidad III – Capítulo III - Conjuntos a nivel de bit

Alternativa al tipo set of a nivel de bit

Aquellos lenguajes de computadoras que no brinden los beneficios del tipo conjunto podrán efectuarse de una forma alternativa a **nivel de bit**. Para ello, deberá contener algún operador para el tratamiento de operaciones a nivel de bit. Esos operadores que trabajan con operandos de tipo entero y producen resultados de tipo entero, **Turbo Pascal** los contiene y son:

Operaciones a nivel de bit	Operador
Operador lógico negación: -operador unario-	**not**
Desplazamiento de bit a izquierda:	**shl**
Desplazamiento de bit a derecha:	**shr**
Operador lógico conjunción:	**and**
Operador lógico disyunción exclusivo:	**xor**
Operador lógico disyunción:	**or**

El **lenguaje C / C++** posee la alternativa de manejo a nivel de bits, ya que contiene los operadores anteriormente indicados, los cuales son:

Operaciones a nivel de bit	Operador
Operador lógico negación: -operador unario-	**!**
Desplazamiento de bit a izquierda:	<<
Desplazamiento de bit a derecha:	>>
Operador lógico conjunción:	**&**
Operador lógico disyunción exclusivo:	^
Operador lógico disyunción:	|

Las operaciones **i shl j** y **i shr j** o i << j y i >> j desplazan el valor de i a la izquierda o a la derecha j bits ingresando por el extremo opuesto ceros. El tipo de resultado es del mismo tipo de i.

Ejemplo

x ← 01010011

Si se hace **x ← x shl 3 o x ← x << 3** los bits se desplazan de derecha a izquierda, ingresan ceros por el extremo derecho y los bits del extremo izquierdo se van perdiendo, por lo tanto, el resultado de esa operación será que en x su valor ha sido cambiado a: 10011000, es decir, los 3 dígitos de la extrema izquierda se perdieron e ingresaron 3 ceros por la extrema derecha, representado de esta otra manera

Una de las características más destacadas de estos tipos de operaciones a nivel de bit es que se pueden realizar operaciones de multiplicación y división en enteros muy rápidamente cuando la base es 2, el primer operando establece el exponente de dicha base. Si los bits son desplazados j posiciones hacia la izquierda estamos multiplicando y si se desplazan j posiciones hacia la derecha estamos dividiendo. Por ejemplo si a = 2, realizar la siguiente operación **a shl 3 o a << 3** = 16, ya que a x 2^3 = 2 x 2^3 = 2 x 8 = 16. De la misma manera si a = 16, realizar **a shr 3 o a >> 3** = 2, ya que a / 2^3 = 16 / 2^3 = 16 / 8 = 2.

Las siguientes operaciones a nivel de bit con los operadores lógicos **and, or, not** o &, |, ^ que se indican a continuación son para ilustrar como se opera y el resultado que se obtiene, un valor de 0 = false y un valor de 1 = true:

Ejemplos

	bin.	dec.		bin.	dec.		bin.	dec.
	10010011	147		10010011	147	**not, !**	10010011	147
And, &	00110110	54	**or, ^**	00110110	54		01101100	108
	00010010	18		10110111	183			

Operaciones con conjuntos a nivel de bit en donde A y B son variables de tipo word, por lo tanto el elemento más pequeño es 0 y el elemento más grande es 15. El conjunto vacío es poner todos los bits en 0, como se indica a continuación: **VACIO ← 0**
El conjunto universal es poner a todos los bits en 1, como se indica a continuación:
UNIVERSAL ← $FFFF en donde $ indica notación sistema de numeración hexadecimal.

Operaciones de conjuntos a nivel de bits

Tabla de Operaciones de Conjuntos a nivel de bits

Operación	Expresión
Unión	A **or** B
Intersección	A **and** B
Diferencia	A **and not** B
Diferencia simétrica	A **xor** B
Complemento	U **and not** A
Pertenencia	(A **shr** x) **mod** 2 <> 0
Subconjunto	A and B = A verdadero $A \subseteq B$
Insertar	A ← A + 1 **shl** x si $x \notin A$
Eliminar	A ← A – 1 **shl** x si $x \in A$

Operación	Expresión
Unión	A | B
Intersección	A **&** B
Diferencia	A **& !** B
Diferencia simétrica	A ^ B
Complemento	U **& !** A

Pertenencia	(A >> x) f**mod** 2 != 0
Subconjunto	A & B == A verdadero $A \subseteq B$
Insertar	A ← A + 1 << x si $x \notin A$
Eliminar	A ← A – 1 << x si $x \in A$

Ejemplos

Unión

Unión: C = A or B; C = A | B

```
        0110011010011101   A = {0,2,3,4,7,9,10,13,14}
Or, |   1100111011101110   B = {1,2,3,5,6,7,9,10,11,14,15}
        1110111011111111   C = {0,1,2,3,4,5,6,7,9,10,11,13,14,15}
```

Intersección

Intersección: C = A and B; C = A & B

```
          0110011010011101   A = {0,2,3,4,7,9,10,13,14}
And, &    1100111011101110   B = {1,2,3,5,6,7,9,10,11,14,15}
          0100011010001101   C = {0,2,3,7,9,10,14}
```

Diferencia

Diferencia: C = A – B; C = A & ! B

```
                  0110011010011101   A = {0,2,3,4,7,9,10,13,14}
and not, & !      1100111011101110   B = {1,2,3,5,6,7,9,10,11,14,15}
```

Hacemos primero B ← **not** B; B ← !B B = {0,4,8,12,13}
quedando 0011000100010001 Se complementaron los bits de B, es decir, 0 x 1 y 1 x 0.
por último hacemos el and, & :

```
                  0110011010011101   A = {0,2,3,4,7,9,10,13,14}
        And, &    0011000100010001   B = {0,4,8,12,13}
                  0010000000010001   C = {0,4,13}
```

Diferencia simétrica

Diferencia simétrica: C = A xor B; C = A ^ B

```
        0110011010011101   A = {0,2,3,4,7,9,10,13,14}
xor     1100111011101110   B = {1,2,3,5,6,7,9,10,11,14,15}
        1010100001110011   C = {0,1,4,5,6,8,11,13,15}
```

Complemento

Complemento: C = U and not A; C = U & ! A

```
                 0110011010011101   A = {0,2,3,4,7,9,10,13,14}
not A, !A        1001100101100010   A ← not A; A ← !A;  A = {1,5,6,8,11,12,15}
                 1111111111111111   U = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
And, &           1001100101100010   not A; !A
        1001100101100010   C = {1,5,6,8,11,12,15}
```

Pertenencia

Pertenencia: (A shr x) mod 2 <> 0; (A >> x) fmod 2 != 0

0110011010011101 A = {0,2,3,4,7,9,10,13,14}

Determinar si x = 4 pertenece al conjunto A. Desplazamos los bits del conjunto A, 4 posiciones hacia la derecha resultando 0000011001101001, notamos que se perdieron los 4 bits de la extrema derecha del conjunto A y se incorporaron 4 ceros por la extrema izquierda. Luego al dividir por 2, nos queda un resto de 1 y como es distinto de 0 decimos que $4 \in A$.

SubConjunto

Subconjunto: A and B = A; A & B == A verdadero $A \subseteq B$

	0101110010011110	A = {1,2,3,4,7,10,11,12,14}
And, &	1101111011111110	B = {1,2,3,4,5,6,7,9,10,11,12,14,15}
	0101110010011110	C = {1,2,3,4,7,10,11,12,14}

luego notamos que el conjunto C es igual al conjunto A, por lo tanto decimos que el conjunto A es subconjunto de B.

Unidad III Capítulo IV

Unidad III – Capítulo IV - Conjuntos en C++. La Clase set

Tipo de dato estructurado de la clase: set

El lenguaje C++ brinda una cantidad de herramientas para las operaciones entre **conjuntos** por medio de la clase **set**. Dada la importancia que tiene este concepto no solo para la matemática sino también para otras disciplinas, a continuación se tratará este tema.

Un conjunto es una colección de elementos bien definidos. En el caso de C++, esos elementos deben ser todos del mismo tipo y ese tipo **a diferencia del lenguaje Turbo Pascal pueden ser de cualquier tipo**, es decir, tanto ordinales, como no ordinales, en este último caso incluimos a los tipos reales, **float, doble, long double**, como así también las cadenas de carateres, tanto en la versión del lenguaje C como **arreglo de caracteres** como al estilo de C++ de la clase **string**. La cantidad máxima de elementos está limitada a una cantidad de elementos determiada por la función miembreo **max_size(),** mientras que la cantidad de elementos que contiene una variable conjunto en un instante dado está determinada por la función miembro **size().** El tipo conjunto permite manipular con un solo nombre una serie de valores o elementos, siendo estos, las componentes de la estructura conjunto. **Los elementos de un conjunto de tipo set se ubicar en un <u>orden ascendente</u> y <u>sin repetición de sus valores</u>**.

Un conjunto de la clase set, posee las siguientes cualidades:

1. **Contenedor** sus valores están contenidos en un espacio denominado el contenedor del conjunto.
2. **Reversible** porque podemos recorrer sus elementos en orden ascendene de izquierda a derecha o bien en orden descendente si lo recorremos de derecha a izquierda.
3. **Ordenado** por defecto un conjunto de la clase set sus elemenos se ubican en un orden ascendente según el valor de la clave. Por medio de una función de comparación se podrá invertir el orden de sus elementos. El valor por defecto es ***less***<Key>, que podrá ser cambiado por ***greater***<Key>.
4. **Único** porque sus valores claves no pueden repetirse, en caso de insertar un nuevo valor y este ya se encuentra en el conjunto, entonces no será insertado como duplicado, es decir, se descarta.

La clase set se lo define como plantilla, por lo que podremos pasar a una función de la clase conjunto el tipo genérico, representado por ejemplo como <T> el tipo de datos que manejará el conjunto.

Si queremos que sus valores se <u>repitan</u> en el conjunto se debe utiliar la **clase multiset**. La manera en que el lenguaje C++ determina si un elemento pertenece o no al conjunto, es a través de una función miembro **count(valor)** en donde valor es el elemento que queremos saber **si pertenece al conjunto, en este caso retorna un valor de 1, caso contrario, retorna 0.**

Para utilizar variables o mejor dicho objetos de tipo conjunto, debemos indicarle al pre-procesador a través de un *include* la clase set.

#include <set>

Ejemplos:

```
set<int> CjtoInt;
set<float> CjtoReal = {3.14, 2.7182, 1.4142, 0.956, -185.37, 1234.8562};
set<string> CjtoNom = {"Jose", "Maria",  "Raquel", "Pedro", "Juan", "Ezequel"};
set<int, greater<int>> CjtoEnt = {4, 12, 87, 32, 15, 76}; // ordena descendente.
set<float, less<float>> CjtoFloat; // ordena ascedente, valor por defecto.
cout << CjtoReal.size() << endl;  // informa 6.
cout << CjtoReal.count(1.4142);  // informa 1, es decir, existe.
```

Observaciones:

En la definición del CjtoReal y del CjtoNom, en ambos casos, se asignan los valores literales a dichos conjuntos.

En la versión Borland V 5.5 no podemos hacer esto, ya que, no está definido por el lenguaje este tipo de sintaxis, en otras versiones no Borland como ser el compilador g++ si podremos utilizar ese formato sintáctico.

A cambio en la versión Borland podremos utilizar una alternativa utilizando un arreglo estático.

```
float vecR[] = {3.14, 2.7182, 1.4142, 0.956, -185.37, 1234.8562};
```

luego haremos lo siguiente:

Incorporar elementos al conjunto

```
CjtoReal.insert(vecR, vecR + 6);
CjtoReal.insert(3.452);  // inserta otro elemento en CjtoReal.
```

insert()

Es una función miembro de la clase set, su tarea es insertar elementos al cunjunto, en este ejemplo, **inserta todos los elementos contenidos en el vector vecR**, y en el segundo ejemplo inserta otro elemento más al CjtoReal.
Podremos emitir cada uno de los elementos como se indica a continuación

```
set <float>::iterator it;

for (it = CjtoReal.begin(); it != CjtoReal.end(); it++)
  cout << *it << endl;
```

En versiones más actuales del lenguaje C++ existe una forma más simple de emitir los elementos de un conjunto. Para invertir el orden declaramos además:

```
set<float, greater<float>> CjtoReal2 = {CjtoReal.begin(), CjtoReal.end()};
```

```
for (auto const &i : CjtoReal)
   cout << i << endl;
// Emite ascendentemente.
```

```
for (auto const &i : CjtoReal2)
   cout << i << endl;
// Emite descendentemente.
```

Eliminar elementos a un conjunto

erase()

Es una función miembro de la clase set, su tarea es eliminar elementos de un conjunto.

Ejemplo:

```
CjtoReal.erase(3.452);
CjtoReal.erase(CjtoR.begin(), CjtoR.end());
```

Limpiar o vaciar un conjunto

clear()

La función miembro de la clase set limpia todos los elementos contenidos en el conjunto, dicho de otra manera, deja al conjunto vacío.

Ejemplo:

```
CjtoReal.clear();
```

Determinar si un elemento pertenece a un conjunto

count()

La función miembro count nos dice si un valor pertenece o no a un conjunto. Retorna el valor 1 si pertenece, sino, retorna el valor 0. En realidad retorna la cantidad de repeticiones del valor indicado y tratándose de la clase set si está presente, solo se encuentra una sola vez, caso contrario, cero vez.

Ejemplo:

```
CjtoReal.count(1.4142);
CjtoReal.count( 0.0);
```

contains()

En versión de actualización C++20 existe además la función contains.

Ejemplo:

```
CjtoReal.contains(1.4142);
```

Determinar si un conjunto es vacío

empty()

Función miembro que informa si un conjunto está vacío.

Ejemplo:

```
CjtoReal.clear();
if (CjtoReal.empty())
  cout << "El conjunto CjtoReal está vacío.";
else
  cout << "No está vacío el conjunto CjtoReal.";
```

Recorrer un conjunto de izquierda a derecha

```
  set<float>::iterator it;

  for (it = CjtoReal.begin(); it != CjtoReal.end(); it++)
     cout << *it << ' ';
```

Recorrer un conjunto de derecha a izquierda

```
  set<float>::reverse_iterator rit;

  for (rit = CjtoReal.rbegin(); rit != CjtoReal.rend(); rit++)
     cout << *rit << ' ';
```

iterator

Un iterador es un objeto que puede iterar sobre los elementos de un contenedor de la biblioteca estándar de C++ y proporcionar acceso a los elementos individuales. Para recorrer los elementos de izquierda a derecha se utilizan las funciones miembre begin() y end().

Ejemplo:

```
for (it = CjtoReal.begin(); it != CjtoReal.end(); it++)
     cout << *it << endl;
```

it es el objeto que recorre el contenedor CjtoReal de izquierda a derecha de sus elementos contenidos, siendo el primer elemento indicado por begin(), y cuando se alcance la posición end() indicado por el iterador it, la condición si it != CjtoReal.end() se tornará falsa, abandonando el ciclo. También se observa que el iterador it se comporta como un puntero apuntando en cada secuencia del ciclo al próximo elemento, esto se logra incrementando el valor del iterador it en uno más.

begin()

Permite asignar a un iterator el punto de inicio a los elementos del contenedor.

end()

Permite determinar a un iterator si alcanzó el punto final a los elementos del contenedor.

reverse_iterator

```
for (rit = CjtoReal.rbegin(); rit != CjtoReal.rend(); rit++)
    cout << *rit << endl;
```

Permite recorrer los elementos de un contenedor en orden inverso es decir de derecha a izquierda de sus elementos contenidos, siendo el primer elemento indicado por rbegin(), y cuando se alcance la posición rend() indicado por el iterador rit, la condición si rit != CjtoReal.rend() se tornará falsa, abandonando el ciclo. También se observa que el iterador rit se comporta como un puntero apuntando en cada secuencia del ciclo al próximo elemento, esto se logra incrementando el valor del iterador rit en uno más. Observar que en vez de restar en uno menos, se mantiene el incremento del iterator rit en incrementarlo en uno más.

rbegin()

Permite asignar a un iterator el punto de inicio a los elementos del contenedor, en realidad es el último elemento del contenedor al que va a apuntar el iterador rit.

rend()

Permite determinar si el iterador rit alcanzó el punto final de los elementos contenidos en el contenedor, en realidad, este apunta al primer elemento del contenedor.

Operaciones entre conjuntos

Las operaciones entre conjuntos que desarrollaremos en en resto del capítulo serán:

- ✓ **UNION**
- ✓ **INTERSECCION**
- ✓ **DIFERENCIA**
- ✓ **DIFERENCIA SIMÉTRICA**
- ✓ **COMPLEMENTO**

Para cada una de las operaciones entre conjuntos utilizaremos los algoritmos proporcionados por el lenguaje C++. Para ello debemos indicar con un include la librería algorithm.

```
#include <algorithm>
set_union(Cjto1.begin(),Cjto1.end(),
            Cjto2.begin(),Cjto2.end(),inserter(CjtoR,CjtoR.begin()));
set_intersection(Cjto1.begin(),Cjto1.end(),
                    Cjto2.begin(),Cjto2.end(),inserter(CjtoR,CjtoR.begin()));
set_difference(Cjto1.begin(),Cjto1.end(),
                   Cjto2.begin(),Cjto2.end(),inserter(CjtoR,CjtoR.begin()));
set_simmetric_difference(Cjto1.begin(),Cjto1.end(),
Cjto2.begin(),Cjto2.end(),inserter(CjtoR,CjtoR.begin()));
```

El complemento lo definimos de la siguiente manera:

```
CjtoUniv – (Cjto1 U Cjto2)
```

Por razones de simplificación en la escritura cuando utilizemos estos procesos algorítmicos emplearemos funciones para cada una de las operaciones indicadas, simplificando la invocación con solo 2 argumentos, los dos conjuntos de datos, y la función retorna el conjunto resultado. Se implementan con plantillas, para, de esta manera generalizar los conjuntos a diferentes tipos de datos.

A continuación se muestran estos módulos.

Unión

```
template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
  return Z;
} // Union
```

Intersección

```
template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                inserter(Z, Z.begin()));
  return Z;
} // Interseccion
```

Diferencia

```
template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
               inserter(Z, Z.begin()));
  return Z;
} // Diferencia
```

Diferencia Simétrica

```
template <typename T>
set<T> DiferenciaSimetrica(set<T> X, set<T> Y) {
  set<T> Z;

  set_simmetric_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                           inserter(Z, Z.begin()));
  return Z;
} // DiferenciaSimetrica
```

Diagrama de Venn

Permite representar en forma gráfica como interactúan los conjuntos entre si, estos inmersos como sub-conjuntos, en el conjunto Universal o Referencial

Operación		Áreas
A U B	=	I + II + III
A ∩ B	=	III
A – B	=	I
B – A	=	II
A Δ B	=	I + II
U – (A U B)	=	IV

Ejemplo operaciones entre conjuntos

```
/*
  Id.Programa: Operaciones entre conjuntos.cpp
  Autor……: Lic. Hugo Cuello
  Fecha……: febrero-2024
  Comentario..: Operaciones entre conjuntos de:
                Unión, Intersección, Diferencia,
                Diferencia Simétrica, Complemento.
  Compilador..: g++17 o g++20
*/

#include<iostream>
#include<set>
#include<algorithm>
using namespace std;

template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
  return Z;
} // Union

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
               inserter(Z, Z.begin()));
  return Z;
} // Interseccion

template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
              inserter(Z, Z.begin()));
  return Z;
} // Diferencia

template <typename T>
set<T> DiferenciaSimetrica(set<T> X, set<T> Y) {
  set<T> Z;

  set_symmetric_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                       inserter(Z, Z.begin()));
  return Z;
} // DiferenciaSimetrica
```

```
template <typename T>
set<T> Complemento(set<T> X, set<T> Y, set<T> U) {
  set<T> Z,
        R;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
              inserter(Z, Z.begin()));
  set_difference(U.begin(),U.end(), Z.begin(),Z.end(),
              inserter(R,R.begin()));
  return R;
} // Complemento

main() {
  set<int> Universal,
             Vacio;
  set<int> A = {5, 19, 12, 524, 105, 896, 14, 23, 79, 238, 515, 22, 3, 55, 76};
  set<int> B = {623, 105, 973, 412, 22, 35, 86, 702, 14, 36, 896, 21};
  set<int> C;

  for (int i = 1; i <= 1000; i++)
     Universal.insert(i);

  C = Union(A,B);
  cout << "AUB - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  C = Interseccion(A,B);
  cout << "A∩B - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  C = Diferencia(A,B);
  cout << "A-B - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  C = Diferencia(B,A);
  cout << "B-A - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  C = DiferenciaSimetrica(A,B);
  cout << "A/B - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  C = Complemento(A,B);
  cout << "U – (AUB) - #C: " << C.count() << endl;
  for (auto const &i : C)
     cout << i << endl;
  return 0;
}
```

Produce los siguientes resultados, de acuerdo a la muestra presentada.
#(A+B) es #C: 23
3 5 12 14 19 21 22 23 35 36 55 76 79 86 105 238 412 515 524 623 702 896 973
#(A*B) es #C: 4
14 22 105 896
#(A-B) es #C: 11
3 5 12 19 23 55 76 79 238 515 524 B-A - #C: 8
21 35 36 86 412 623 702 973
#(A/B) es #C: 19
3 5 12 19 21 23 35 36 55 76 79 86 238 412 515 524 623 702 973
#U – (A+B) es #C: 977
También emite la lista de los 977 valores que para simplificar la escritura diremos que son los x / x ϵ Universal – (A+B), es decir, #Universal - #(A+B) = 1000-23 = 977.

G3_Ej01 Operaciones con Conjuntos Encuesta 3 Preguntas

```
/*
  Id.Programa: G3_Ej01 Encuesta 3 Preguntas.cpp
  Autor..........: Lic. Hugo Cuello
  Fecha..........: febrero-2024
  Comentario.: Encuesta 100 personas, si respondieron bien o mal a
                 las 3 preguntas de un total de 100 personas. Los casos
                 que se deben realizar son: Cantidad de personas que
                 respondieron bien a las 3 preguntas, las que respondieron
                 bien a solo 2 preguntas, las que respondieron bien
                 solo a una pregunta o las que no respondieron bien a las
                 3 preguntas.
  Temas......: Estructura de Datos Conjunto.
          Herramientas utilizadas:
          Clases:
            set, algorithm.
          Funciones miembro:
            insert, erase, begin, end, rbegin, rend,
            clear, empty, count, contains, max_size, size.
          Typedef:
            iterator, reverse_iterator.
          Algoritmos:
            set_union, set_intersection,
            set_difference, set_symmetric_difference.
          Opciones en la definicion de un conjunto:
            less (valor por defecto), greater,
          Foreach for especial.
  Compilador.: cygwin GCC g++20.
*/
```

```
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

const
  set<short> CJTO_VACIO;

template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
            inserter(Z, Z.begin()));
  return Z;
} // Union

template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                 inserter(Z, Z.begin()));
  return Z;
} // Diferencia

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                   inserter(Z, Z.begin()));
  return Z;
} // Interseccion

void EmiteCantCjto(set<short> CjtoX, string NomCjto) {

  short CantPers = 0;
  short k = 1;
  while (CjtoX != CJTO_VACIO) {
    if (CjtoX.count(k)) {
      ++CantPers;
```

```
      CjtoX.erase(k);
    }
    ++k;
  }
  cout << "Cantidad de personas del Cjto " << NomCjto << " es: " <<  CantPers << endl;
} // EmiteCantCjto

/* Valores contenidos en los conjuntos: Universal, CjtoPreg1, CjtoPreg2, CjtoPreg3.


-----------------------------------------------------------------------------------------------------
#U100 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
-----------------------------------------------------------------------------------------------------
#A       1,     3, 4, 5, 6, 7,     9,10,          13,     15,16,                 21,22,
#B          2,              7, 8,  9,10,11,12,                              20,21,              25,
#C              3, 4, 5, 6,        9,10,11,12, 13,14,15,16,17,18,19,         22,23,24,25,
-----------------------------------------------------------------------------------------------------
    26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
-----------------------------------------------------------------------------------------------------
#A                   30,    32,33,34,35,36,37,38,39,40,41,42,43,44,45,                  50,
#B                              33,34,35,                                    46,47,48,49,50,
#C 26,27,28,29,30,          33,34,35,                                                   50,
-----------------------------------------------------------------------------------------------------
     51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
-----------------------------------------------------------------------------------------------------
#A                            57,58,59,60,61     ,63,64,                        71,72,73,74,75,
#B              54,                                                  67,68,69,                   75,
#C 51,52,53,      55,56,                                 65,66,      68,     70,                75,
-----------------------------------------------------------------------------------------------------
     76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
-----------------------------------------------------------------------------------------------------
#A 76,77,78,     80,81,82,83,84,85,86,87,88,89,     91                          98
#B                              83,84,85,86,87,88,89,                        96,97,98,99
#C              79,     81,82,                     88,     90,     92,93,94,95,     97,     99,100
*/

main () {
  set<short> CjtoPreg1 = {  1, 3,  4,   5,  6,  7,  9,10,13,15,16,21,22,30,
                            32,33,34,35,36,37,38,39,40,41,42,
                            43,44,45,50,57,58,59,60,61,63,64,
                            71,72,73,74,75,76,77,78,80,81,82,
                            83,84,85,86,87,88,89,91,98};
  set<short> CjtoPreg2 = {2,7,8,9,10,11,12,20,21,25,33,34,35,
                          46,47,48,49,50,54,67,68,69,75,83,84,
```

```
                                85,86,87,88,89,96,97,98,99};
set<short> CjtoPreg3 = {3,4,5,6,9,10,11,12,13,14,15,16,17,18,19,
                                22,23,24,25,26,27,28,29,30,33,34,35,50,
                                51,52,53,55,56,65,66,68,70,75,79,81,82,
                                88,90,92,93,94,95,97,99,100};
set<short> CjtoR1,CjtoR2,CjtoR3,CjtoR4,CjtoR5,CjtoR6,
        CjtoR7,CjtoR8,CjtoR9,CjtoR10,CjtoR11,CjtoR12,
        CjtoR13,CjtoR14,CjtoI4,CjtoI5;
float vecR[] = {3.14, 2.7182, 1.4142, 0.956, -185.37, 1234.8562};
set<float>::iterator it;
set<float> CjtoR;
set<float>::reverse_iterator rit;
set<short> CJTO_UNIVERSAL;
set<float> CjtoReal  = {3.14, 2.7182, 1.4142, 0.956, -185.37, 1234.8562};

for (short i = 1; i <= 100; i++)
  CJTO_UNIVERSAL.insert(i);
CjtoR1  = Interseccion(CjtoPreg1,CjtoPreg2); // CjtoR1 = A * B
CjtoR2  = Interseccion(CjtoR1,CjtoPreg3);    // CjtoR2 = A * B * C
CjtoR3  = Interseccion(CjtoPreg1,CjtoPreg3); // CjtoR3 = A * C
CjtoR4  = Interseccion(CjtoPreg2,CjtoPreg3); // CjtoR4 = B * C
CjtoI4  = Interseccion(CjtoPreg3,CjtoPreg1); // CjtoI4 = C * A
CjtoI5  = Interseccion(CjtoPreg3,CjtoPreg2); // CjtoI5 = C * B
CjtoR5  = Diferencia(CjtoR1,CjtoPreg3);      // CjtoR5 = A * B - C
CjtoR6  = Diferencia(CjtoR3,CjtoPreg2);      // CjtoR6 = A * C - B
CjtoR7  = Diferencia(CjtoR4,CjtoPreg1);      // CjtoR7 = B * C - A
CjtoR8  = Union(CjtoPreg1,CjtoPreg2);        // CjtoR8 = A + B
CjtoR9  = Union(CjtoR8,CjtoPreg3);           // CjtoR9 = A + B + C
CjtoR10 = Union(CjtoPreg1,CjtoPreg3);        // CjtoR10 = A + C
CjtoR11 = Union(CjtoPreg2,CjtoPreg3);        // CjtoR11 = B + C
CjtoR12 = Diferencia(CjtoPreg1,Union(Union(CjtoR1,CjtoR3),CjtoR2));
CjtoR13 = Diferencia(CjtoPreg2,Union(Union(CjtoR4,CjtoR1),CjtoR2));
CjtoR14 = Diferencia(CjtoPreg3,Union(Union(CjtoI4,CjtoI5),CjtoR2));
EmiteCantCjto(CjtoR2,"I. A*B*C");
EmiteCantCjto(Diferencia(CjtoR1,CjtoR2),"II. (A*B)-(A*B*C)");
EmiteCantCjto(Diferencia(CjtoR3,CjtoR2),"III. (A*C)-(A*B*C)");
EmiteCantCjto(Diferencia(CjtoR4,CjtoR2),"IV. (B*C)-(A*B*C)");
EmiteCantCjto(CjtoPreg1,"V. A");
EmiteCantCjto(CjtoPreg2,"VI. B");
EmiteCantCjto(CjtoPreg3,"VII. C");
cout << "CONJUNTOS ADICIONALES AL EJ.01" << endl;
EmiteCantCjto(CjtoR1,"A: A*B");
EmiteCantCjto(Diferencia(CJTO_UNIVERSAL,CjtoR2),"B: Univ-(A*B*C)");
```

```
EmiteCantCjto(Union(Union(CjtoR5,CjtoR6),CjtoR7),"C: (A*B-C)+(A*C-B)+(B*C-A)");
EmiteCantCjto(CjtoR8,"D: A+B");
EmiteCantCjto(CjtoR10,"E: A+C");
EmiteCantCjto(CjtoR11,"F: B+C");
EmiteCantCjto(CjtoR9,"G: A+B+C");
EmiteCantCjto(Diferencia(CJTO_UNIVERSAL,CjtoR9),"H: U-(A+B+C)");
EmiteCantCjto(Diferencia(CJTO_UNIVERSAL,CjtoR2),"I: U-(A*B*C)");
EmiteCantCjto(CjtoR4,"J: B*C");
EmiteCantCjto(CjtoR3,"K: A*C");
EmiteCantCjto(CjtoR1,"L: A*B");
EmiteCantCjto(Union(Union(CjtoR12,CjtoR13),CjtoR14),
                                        "M: A-(A*B+A*C+A*B*C) + "
                                            "B-(B*C+A*B+A*B*C) + "
                                            "C-(C*A+C*B+A*B*C)");
cout << "Existe valor 60 en CjtoPreg1: "
    << CjtoPreg1.count(60) << endl; // retorna 1 si existe.
cout << "Existe valor 25 en CjtoPreg1: "
    << CjtoPreg1.count(25) << endl; // retorna 0 si NO existe.
cout << "La maxima longitud posible del conjunto CjtoR1 es: "
    << CjtoR1.max_size() << "." << endl;
cout << "Cant. elem. CjtoReal es: " << CjtoReal.size() << endl;
cout << "Existe valor 1.4142 en CjtoReal: " << CjtoReal.contains(1.4142) << endl;
CjtoR.insert(vecR, vecR + 6); //inserta en el CjtoR todos los valores de vecR.
cout << "Cant. elem. CjtoR es: " << CjtoR.size() << ' ';
cout << endl << "Emite forClasico CjtoReal: " << endl;
for (it = CjtoReal.begin(); it != CjtoReal.end(); it++)
  cout << *it << ' ';
cout << endl << "Emite foreach CjtoReal: " << endl;
for (auto const &i : CjtoReal)
  cout << i << ' ';
cout << endl << "Cant. de CjtoR: " << CjtoR.size();
cout << endl << "Emite izq a der CjtoR: " << endl;
for (auto const &i : CjtoR)
  cout << i << ' ';
cout << endl << "Emite der a izq CjtoR: " << endl;
for (rit = CjtoReal.rbegin(); rit != CjtoReal.rend(); rit++)
  cout << *rit << ' ';
CjtoReal.clear(); // deja al CjtoReal vacio.
if (CjtoReal.empty())
  cout << endl << "CjtoReal es vacio";
else
  cout << endl << "CjtoReal NO es vacio";
CjtoR.erase(CjtoR.begin(),CjtoR.end());
if (CjtoR.empty())
```

```
  cout << endl << "CjtoR es vacio";
 else
  cout << endl << "CjtoR NO es vacio";

 set<int,greater<int>> s2;
 set<int,greater<int>>::iterator It;
 cout << endl << "greater Inserta elem en orden descendente de > a <" << endl;
 s2.insert( 10 );
 s2.insert( 70 );
 s2.insert( 20 );
 s2.insert( 85 );
 s2.insert( 30 );
 s2.insert( 60 );
 for (It = s2.begin(); It != s2.end(); It++)
  cout << *It << ' ';
 set<int,less<int>> s1;
 set<int,less<int>>::iterator It1;
 cout << endl << "less Inserta elem en orden ascendente de < a >" << endl;
 s1.insert( 10 );
 s1.insert( 70 );
 s1.insert( 20 );
 s1.insert( 85 );
 s1.insert( 30 );
 s1.insert( 60 );
 for (It1 = s1.begin(); It1 != s1.end(); It1++)
  cout << *It1 << ' ';
 return 0;
}
```

G3_Ej02 Encuesta 100 Personas combinación consumo bebidas en un pub

```
/*
 Id.Programa: G3_Ej02 Pub Bebidas 3 Sabores.cpp
 Autor......: Lic. Hugo Cuello
 Fecha......: febrero-2024
 Comentario.: Encuesta limitado a 100 personas, de sus consumos
              de bebidas (NARANJA, ANANA, FRUTILLA), presentando
              los siguientes casos:
               La cantidad de personas que solo bebieron un sabor,
               las que solo bebieron 2 sabores, las que bebieron
               los 3 sabores y las que no bebieron ningún sabor.
 Compilador.: cygwin GCC g++20.
*/
```

```
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

const
  set<short> CJTO_VACIO;

template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
           inserter(Z, Z.begin()));
  return Z;
} // Union

template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                inserter(Z, Z.begin()));
  return Z;
} // Diferencia

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                  inserter(Z, Z.begin()));
  return Z;
} // Interseccion

template <typename T>
set<T> Complemento(set<T> U, set<T> X, set<T> Y) {
  set<T> Z,
              R;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
           inserter(Z, Z.begin()));
  set_difference(U.begin(),U.end(),
                Z.begin(),Z.end(),
```

```
            inserter(R,R.begin()));
  return R;
} // Complemento

void EmiteCantCjto(set<short> CjtoX, string NomCjto) {
  short CantPers = 0;
  short k = 1;

  while (CjtoX != CJTO_VACIO) {
    if (CjtoX.count(k)) {
      ++CantPers;
      CjtoX.erase(k);
    }
    ++k;
  }
  cout << "Cantidad de personas del Cjto " << NomCjto << " es: " <<  CantPers << endl;
} // EmiteCantCjto
/* Valores contenidos en los conjuntos: Universal, CjtoPreg1, CjtoPreg2, CjtoPreg3.
-----------------------------------------------------------------------------------------
#U100    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
-----------------------------------------------------------------------------------------
#A:      1,    3,       6, 7,    9,10,      13,      16,            21,22,
#B:         2, 3,    5,          9,      12,      15,         19,      22,      25,
#C:      1, 2,    4,       7,    9,      12,   14,   16,   18,19,   21,      24,
-----------------------------------------------------------------------------------------
        26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
-----------------------------------------------------------------------------------------
#A                  30,      33,      36,37,         41,42,43,44,45,            50,
#B         27,   29,      32,   34,         38,39    41,   43,   45,   47,
#C         27,28,   30,31,               37,   39,         43,            48,   50,
-----------------------------------------------------------------------------------------
        51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
-----------------------------------------------------------------------------------------
#A                       57,      60,61,   63,64,   66,            71,72,   74,75,
#B            53,   55,   57,   59,      62,            67,   69,      72,
#C      51,   53,         57,      60,         64,      67,       70,        74,
-----------------------------------------------------------------------------------------
        76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
-----------------------------------------------------------------------------------------
#A      76,77,78,   80,81,82,   84,85,86    88,89,   91,                   98,
#B      76,            81,   83,      86,            91,   93,94,               100.
#C         77,   79,80,81,      84,      87,         91,      94,   96,   98
*/

main () {
  set<short> CjtoNaranja  = { 1, 3, 6, 7, 9,10,13,16,21,22,30,33,36,37,
                              41,42,43,44,45,50,57,60,61,63,64,66,71,72,
                              74,75,76,77,78,80,81,82,84,85,86,88,89,91,98};
  set<short> CjtoAnana    = { 2, 3, 5, 9,12,15,19,22,25,27,29,
                              32,34,38,39,41,43,45,47,53,55,57,59,
                              62,67,69,72,76,81,83,86,91,93,94,100};
```

```
  set<short> CjtoFrutilla = { 1, 2, 4, 7, 9,12,14,16,18,19,21,24,27,
                              28,30,31,37,39,43,48,50,51,53,57,60,64,
                              67,70,74,77,79,80,81,84,87,91,94,96,98};
  set<short> CJTO_UNIVERSAL,
             CjtoIntNAF,
             CjtoIntNA,
             CjtoIntNF,
             CjtoIntAF;

  for (short i = 1; i <= 100; i++)
    CJTO_UNIVERSAL.insert(i);
  CjtoIntNAF = Interseccion(Interseccion(CjtoNaranja,CjtoAnana),CjtoFrutilla);
  CjtoIntNA  = Interseccion(CjtoNaranja,CjtoAnana);
  CjtoIntNF  = Interseccion(CjtoNaranja,CjtoFrutilla);
  CjtoIntAF  = Interseccion(CjtoAnana,CjtoFrutilla);
  EmiteCantCjto(CjtoNaranja,"NARANJA: ");
  EmiteCantCjto(CjtoAnana,"ANANA: ");
  EmiteCantCjto(CjtoFrutilla,"FRUTILLA: ");
  EmiteCantCjto(CjtoIntNAF,"NARANJA+ANANA+FRUTILLA: ");
  EmiteCantCjto(Diferencia(CjtoIntNA,CjtoIntNAF),"NARANJA+ANANA: ");
  EmiteCantCjto(Diferencia(CjtoIntNF,CjtoIntNAF),"NARANJA+FRUTILLA: ");
  EmiteCantCjto(Diferencia(CjtoIntAF,CjtoIntNAF),"ANANA+FRUTILLA: ");
  EmiteCantCjto(Diferencia(Diferencia(Diferencia(CjtoNaranja,CjtoIntNA),
                                CjtoIntNF),CjtoIntNAF),"Solo NARANJA: ");
  EmiteCantCjto(Diferencia(Diferencia(Diferencia(CjtoAnana,CjtoIntNA),
                                CjtoIntAF),CjtoIntNAF),"Solo ANANA: ");
  EmiteCantCjto(Diferencia(Diferencia(Diferencia(CjtoFrutilla,CjtoIntNF),
                                CjtoIntAF),CjtoIntNAF),"Solo FRUTILLA: ");
  EmiteCantCjto(Complemento(CJTO_UNIVERSAL,Union(CjtoNaranja,CjtoAnana),
                          CjtoFrutilla),"Ningun sabor: ");
  return 0;
}
```

G3_EJ03 Operaciones entre Conjuntos por Consola

```
/*
  Id.Prog....: G3_03 Operaciones x Consola.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: febrero-2024
  Comentario.: Operaciones entre Conjuntos por Consola:
               Union, Interseccion, Diferencia, Dif.Sim., Complemento.
  Compilador.: Borland V 5.5.
*/
```

```
#include <fstream>
#include <set>
#include <conio.h>
using namespace std;

typedef unsigned short ushort;

const set<ushort> CJTO_VACIO;
typedef set<ushort> tCjtoDias;

template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
           inserter(Z, Z.begin()));
  return Z;
} // Union

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                 inserter(Z, Z.begin()));
  return Z;
} // Interseccion

template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
               inserter(Z, Z.begin()));
  return Z;
} // Diferencia

template <typename T>
set<T> DiferenciaSimetrica(set<T> X, set<T> Y) {
  set<T> Z;

  set_symmetric_difference(X.begin(), X.end(), Y.begin(), Y.end(),
```

```
                    inserter(Z, Z.begin()));
  return Z;
} // DiferenciaSimetrica

template <typename T>
set<T> Complemento(set<T> X, set<T> Y, set<T> U) {
  set<T> Z,
       R;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
  set_difference(U.begin(),U.end(),
              Z.begin(),Z.end(),
              inserter(R,R.begin()));
  return R;
} // Complemento

void ObtCjto(tCjtoDias &CJTO_DIAS, string titulo) {
  ushort n,
       elem;

  cout << "Ing. cant. elementos al " << titulo << ": ";
  cin >> n;
  CJTO_DIAS = CJTO_VACIO;
  for ( short i = 1; i <= n; i++) {
    do {
      cout << "Ing. elemento (dia): ";
      cin >> elem;
    } while (CJTO_DIAS.count(elem));
    CJTO_DIAS.insert(elem);
  }
} // ObtCjto

char Menu() {
  set<char> CJTO_OPC;
  char opc;

  CJTO_OPC.insert('U');
  CJTO_OPC.insert('I');
  CJTO_OPC.insert('D');
  CJTO_OPC.insert('S');
  CJTO_OPC.insert('C');
  CJTO_OPC.insert('F');
  clrscr();
```

```
  gotoxy(10, 3); cout << "Menu de opciones Operaciones entre conjuntos" << endl;
  gotoxy(20, 5); cout << "U: Union";
  gotoxy(20, 6); cout << "I: Interseccion";
  gotoxy(20, 7); cout << "D: Diferencia";
  gotoxy(20, 8); cout << "S: Dif.Sim.";
  gotoxy(20, 9); cout << "C: Complemento";
  gotoxy(20,10); cout << "F: Finalizar";
  gotoxy(20,12); cout << "Su opcion: ";
  do {
    gotoxy(wherex()+1,12);
    clreol();
    opc = toupper(getch());
    cout << opc;
  } while (!CJTO_OPC.count(opc));
  return opc;
} // Menu

void EmiteCjto(tCjtoDias CjtoX, string titulo) {
  ushort elem = 1;

  cout << endl << titulo;
  while (CjtoX != CJTO_VACIO) {
    if (CjtoX.count(elem)) {
      cout << elem << ' ';
      CjtoX.erase(elem);
    }
    elem++;
  }
  cout << endl;
} // EmiteCjto

main() {
  set<ushort> Universal;
  set<ushort> CjtoA,
              CjtoB,
              CjtoC;
  set<char> OPER_CJTO;
  ushort elem;
  char opc;

  for (ushort i = 1; i <= 31; i++)
    Universal.insert(i);
  OPER_CJTO.insert('U');
```

```
  OPER_CJTO.insert('I');
  OPER_CJTO.insert('D');
  OPER_CJTO.insert('S');
  OPER_CJTO.insert('C');
  ObtCjto(CjtoA,"CjtoA");
  ObtCjto(CjtoB,"CjtoB");
  do {
    opc = Menu();
    clrscr();
    string titCjto = " Cjto. A ";
    switch (opc) {
      case 'U':
            CjtoC = Union(CjtoA,CjtoB);
            break;
      case 'I':
            CjtoC = Interseccion(CjtoA,CjtoB);
            break;
      case 'D':
            CjtoC = Diferencia(CjtoA,CjtoB);
            break;
      case 'S':
            CjtoC = DiferenciaSimetrica(CjtoA,CjtoB);
            break;
      case 'C':
            CjtoC = Complemento(CjtoA,CjtoB,Universal);
            break;
    }
    if (OPER_CJTO.count(opc)) {
      EmiteCjto(CjtoA,"Cjto. A: ");
      EmiteCjto(CjtoB,"Cjto. B: ");
      EmiteCjto(CjtoC,titCjto + opc + " Cjto. B: ");
      system("pause");
    }
  } while (!(opc == 'F'));
  return 0;
}
```

G3_Ej04 M.C.D.

```
/*
  Id.Prog...: G3_Ej04 MCD.cpp
  Autor.....: Lic. Hugo Cuello
  Fecha.....: febrero-2024
```

```
  Comentario: Calcula el MCD entre 2 valores, aplicando el siguiente
              metodo:
               a. Por cada nro. se obtienen los divisores, guardandolos
                  en un Cjto.
               b. Se realiza la interseccion de los 2 Cjtos.
               c. Por ultimo obtenemos el maximo divisor del Cjto.Interseccion
  Compilador.: cygwin GCC g++20.
*/
#include <iostream>
#include <set>
using namespace std;
const
  set<int> CJTO_VACIO;

typedef set<int> tCjtoDiv;

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                   inserter(Z, Z.begin()));
  return Z;
} // Interseccion

void Obt2Ent(int &x, int &y) {
  cout << "Calcular el M.C.D. entre 2 valores enteros" << endl;
  cout << "Ing.numerador: ";
  cin >> x;
  cout <<"Ing.denominador: ";
  cin >> y;
} // Obt2Ent

void InicCjto(tCjtoDiv &CjtoX, int x) {
  CjtoX = {1, x};
} // InicCjto

void Divisores(tCjtoDiv &CjtoX, int x) {
  int i;

  for (i = 2; i <= x / 2; i++)
    if (x % i == 0)
      CjtoX.insert(i);
```

```
} // Divisores

void Emite(tCjtoDiv CjtoX, string titulo) {
  int i = 1;

  cout << titulo << ": ";
  while (CjtoX != CJTO_VACIO) {
    if (CjtoX.count(i)) {
      cout << i << ' ';
      CjtoX.erase(i);
    }
    i++;
  }
  cout << endl;
} // Emite

int Max(tCjtoDiv CjtoX) {
  int divi = 1;
  int mcd = 1;

  while (CjtoX != CJTO_VACIO) {
    if (CjtoX.count(divi)) {
      mcd = divi;
      CjtoX.erase(divi);
    }
    divi++;
  }
  return mcd;
} // Max

void Emitemcd(int x, string titulo) {
  cout << titulo << ": " << x;
} // Emitemcd

main() {
  tCjtoDiv Cjto1,
           Cjto2,
           Cjto3;
  int num,
      den;
  Obt2Ent(num,den);
  InicCjto(Cjto1,num);
  InicCjto(Cjto2,den);
  Divisores(Cjto1,num);
```

```
  Divisores(Cjto2,den);
  Cjto3 = Interseccion(Cjto1,Cjto2);
  Emite(Cjto1,"Divisores de num en Cjto1");
  Emite(Cjto2,"Divisores de den en Cjto2");
  Emite(Interseccion(Cjto1,Cjto2),"Divisores comunes en Cjto3");
  Emitemcd(Max(Cjto3),"M.C.D.");
  return 0;
}
```

G3_Ej05 Criba de Eratóstenes

```
/*
  Id.Prog...: G3_05 Criba Eratostenes.cpp
  Autor.....: Lic. Hugo Cuello
  Fecha.....: febrero-2024
  Comentario: Criba de Eratostenes de los 100 primeros Nros. Naturales.
  Compilador.: cygwin GCC g++20.
*/
#include <iostream>
#include <set>
#include <cmath>
using namespace std;
typedef  set<ushort> tCjtoPrimos;
const
  set<ushort> CJTO_VACIO;
  ushort MAX_NUM = 100;
void Cargar(tCjtoPrimos &Criba) {
  for (ushort i = 2; i <= MAX_NUM; i++)
    Criba.insert(i);
} // Cargar

void EliminarNoPrimos(tCjtoPrimos &Criba) {
  ushort Num = 2;

  while (pow(Num,2) <= MAX_NUM) {
    for (ushort k = Num; k <= MAX_NUM / Num; k++)
      Criba.erase(Num * k);
    do
      Num++;
    while (!Criba.count(Num));
  }
} // EliminarNoPrimos

void Emitir(tCjtoPrimos Criba) {
```

```
  ushort Cant = 0,
         k = 2;
  cout << "Nros. primos entre 2 y " << MAX_NUM << endl;
  while (Criba != CJTO_VACIO) {
    if (Criba.count(k)) {
      cout << k << ' ';
      Criba.erase(k);
      Cant++;
    }
    k++;
  }
  cout << endl << "Cant.: " << Cant;
} // Emitir

main() {
  tCjtoPrimos Criba;
  Cargar(Criba);
  EliminarNoPrimos(Criba);
  Emitir(Criba);
  return 0;
}
```

G3_Ej06 Examenes finales AyED, LyED y Fis1

```
/*
  Id.Programa: G3_Ej06 ExamenFinal x 3 asignaturas.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: febrero-2024
  Comentario.: Multi-Conjunto (multiset) de struct, permite
               repetir valores claves, en el caso de las notas.
               Funciones operator<, operator== para criterio de
               los campos del struct sNotas:
                 nroLeg, vNotas[3] de 3 asignaturas
                 0:AlgEstrDatos, 1:LogEstrDiscr, 2:Fisica I
                 con notas [0, 10], en donde cero alumno sin
                 presencia al examen, al menos uno distinto
                 de cero.
  Compilador.: cygwin GCC g++20.
*/

#include <iostream>
#include <set>
using namespace std;

struct sNotas {
  int    nroLeg;
  string asignatura;
  ushort nota; // entre 0 y 10, cero ausente.
```

```
};

/*
bool operator < (const sNotas& lhs, const sNotas& rhs)  {
   string cad1,
          cad2;

   cad1 = to_string(lhs.nroLeg);
   cad2 = to_string(rhs.nroLeg);
   return cad1 + lhs.asignatura < cad2 + rhs.asignatura;
}
*/

bool operator  < (const sNotas& lhs, const sNotas& rhs)  {
  return lhs.nroLeg < rhs.nroLeg;
}

template <typename T>
multiset<T> Union(multiset<T> X, multiset<T> Y) {
  multiset<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
           inserter(Z, Z.begin()));
  return Z;
} // Union

template <typename T>
multiset<T> Interseccion(multiset<T> X, multiset<T> Y) {
  multiset<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                 inserter(Z, Z.begin()));
  return Z;
} // Interseccion

template <typename T>
multiset<T> Diferencia(multiset<T> X, multiset<T> Y) {
  multiset<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
               inserter(Z, Z.begin()));
  return Z;
} // Diferencia

main() {
  multiset<sNotas> CJTO_ALG =  {      {245123, "AyED", 6},{245876, "AyED", 4},
                                {245986, "AyED",10},{246233, "AyED", 2},
                                {245801, "AyED", 4},{242222, "AyED", 7},
                                {243899, "AyED", 7},{241111, "AyED", 3},
                                {245555, "AyED", 9} };
  multiset<sNotas> CJTO_LOG =  {      {245720, "LyED", 5},{241025, "LyED", 7},
```

```
                        {244875, "LyED",10},{245986, "LyED", 9},
                        {243002, "LyED", 2},{245801, "LyED", 4},
                        {250500, "LyED", 7},{243969, "LyED", 3},
                        {245555, "LyED", 9} };
 multiset<sNotas> CJTO_FIS1 = {      {244309, "FIS1", 5},{251032, "FIS1", 6},
                        {245986, "FIS1", 7},{244551, "FIS1", 4},
                        {246963, "FIS1", 8},{246860, "FIS1", 4},
                        {243905, "FIS1", 6},{242222, "FIS1", 8},
                        {246233, "FIS1", 8},{243899, "FIS1", 6},
                        {246804, "FIS1", 6},{245801, "FIS1", 4},
                        {243002, "FIS1", 8},{243969, "FIS1", 2} };
 multiset<sNotas> CJTO_UNIVERSAL = { {245123, "AyED", 6},{245876, "AyED", 4},
                        {245986, "AyED",10},{245801, "AyED", 4},
                        {243899, "AyED", 7},{241111, "AyED", 3},
                        {245555, "AyED", 9},{245720, "LyED", 5},
                        {241025, "LyED", 7},{244875, "LyED",10},
                        {243002, "LyED", 2},{250500, "LyED", 7},
                        {243969, "LyED", 3},{244309, "FIS1", 5},
                        {251032, "FIS1", 6},{244551, "FIS1", 4},
                        {246963, "FIS1", 8},{246860, "FIS1", 4},
                        {243905, "FIS1", 6},{242222, "FIS1", 8},
                        {246233, "FIS1", 8},{246804, "FIS1", 6},
                        {255553, "NADA", 0},{251527, "NADA", 0},
                        {223817, "NADA", 0},{241339, "NADA", 0},
                        {249321, "NADA", 0},{249825, "NADA", 0},
                        {251108, "NADA", 0} };
 multiset<sNotas> CJTO_INT_ALG_LOG_FIS1,
            CJTO_INT_ALG_LOG,
            CJTO_INT_ALG_FIS1,
            CJTO_INT_LOG_FIS1,
            CJTO_SOLO_ALG,
            CJTO_SOLO_LOG,
            CJTO_SOLO_FIS1,
            CJTO_COMPLTO,
            CJTO_UNION_ALG_LOG_FIS1,
            CJTO_ALG_UNION_CJTO_LOG,
            CJTO_SOLO_LAF,
            CJTO_LOG_UNION_FIS1;

 CJTO_ALG_UNION_CJTO_LOG = Union(CJTO_ALG,CJTO_LOG);
 CJTO_UNION_ALG_LOG_FIS1 = Union(CJTO_ALG_UNION_CJTO_LOG,CJTO_FIS1);
 CJTO_COMPLTO = Diferencia(CJTO_UNIVERSAL,CJTO_UNION_ALG_LOG_FIS1);
 cout << "A) Alumnos que no se presentaron a ninguna asignatura..." << endl;
 for (auto& p : CJTO_COMPLTO)
   cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
 cout << endl;
 CJTO_INT_ALG_LOG_FIS1 = Interseccion(Interseccion(CJTO_ALG,CJTO_LOG),CJTO_FIS1);
 CJTO_INT_ALG_FIS1 = Interseccion(CJTO_ALG,CJTO_FIS1);
 CJTO_INT_LOG_FIS1 = Interseccion(CJTO_LOG,CJTO_FIS1);
 CJTO_UNION_ALG_LOG_FIS1 =
Union(Union(CJTO_INT_ALG_FIS1,CJTO_INT_LOG_FIS1),CJTO_INT_ALG_LOG_FIS1);
```

```
  CJTO_SOLO_FIS1 = Diferencia(CJTO_FIS1,CJTO_UNION_ALG_LOG_FIS1); // La U de las 3
Inters.
  cout << "B) Alumnos que solo rindieron FISICA 1..." << endl;
  for (auto& p : CJTO_SOLO_FIS1)
    cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
  cout << "C) Alumnos que aprobaron ALGORITMOS Y ED..." << endl;
  for (auto& p : CJTO_ALG)
    if (p.nota >= 6)
      cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
  cout << "D) Card. Alumnos que aprobaron ALGORITMOS y ED... ";
  short cantAprob = 0;
  for (auto& p : CJTO_ALG)
    if (p.nota >= 6)
      cantAprob++;
  cout << cantAprob << " alumnos" << endl;
  cout << endl;
  CJTO_INT_ALG_LOG_FIS1 = Interseccion(Interseccion(CJTO_ALG,CJTO_LOG),CJTO_FIS1);
  CJTO_INT_ALG_FIS1 = Interseccion(CJTO_ALG,CJTO_FIS1);
  CJTO_INT_LOG_FIS1 = Interseccion(CJTO_LOG,CJTO_FIS1);
  CJTO_INT_ALG_LOG  = Interseccion(CJTO_ALG,CJTO_LOG);
  CJTO_UNION_ALG_LOG_FIS1 =
Union(Union(Union(CJTO_INT_ALG_FIS1,CJTO_INT_LOG_FIS1),CJTO_INT_ALG_LOG_FIS1),CJ
TO_INT_ALG_LOG);
  CJTO_SOLO_ALG = Diferencia(CJTO_ALG,CJTO_UNION_ALG_LOG_FIS1); // La U de las 3
Inters.
  CJTO_SOLO_LOG = Diferencia(CJTO_LOG,CJTO_UNION_ALG_LOG_FIS1); // La U de las 3
Inters.
  CJTO_SOLO_LAF =
Diferencia(Diferencia(CJTO_SOLO_FIS1,CJTO_SOLO_ALG),CJTO_SOLO_LOG);
  CJTO_SOLO_LAF = Union(Union(CJTO_SOLO_LOG,CJTO_SOLO_ALG),CJTO_SOLO_FIS1);
  cout << "E) Card. alumnos que reindieron solamene ALG+LOG+FIS1..." << CJTO_SOLO_LAF.size()
<< endl;
  cout << "     Card. alumnos que reindieron solamene ALG..." << CJTO_SOLO_ALG.size() << endl;
  cout << "     Card. alumnos que reindieron solamene LOG..." << CJTO_SOLO_LOG.size() << endl;
  cout << "     Card. alumnos que reindieron solamene FIS1..." << CJTO_SOLO_FIS1.size() << endl;
  for (auto& p : CJTO_SOLO_LAF)
    cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
  CJTO_LOG_UNION_FIS1 = Union(CJTO_LOG,CJTO_FIS1);
  cout << "F) Alumnos que reindieron LOGICA_ED + FISICA 1..." << CJTO_LOG_UNION_FIS1.size()
<< endl;
  for (auto& p : CJTO_LOG_UNION_FIS1)
    cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
  cout << "G) Card. alumnos que aprobaron solamente LOGICA_ED... ";
  cantAprob = 0;
  for (auto& p : CJTO_SOLO_LOG)
    if (p.nota >= 6)
      cantAprob++;
```

```
  cout << cantAprob << " alumnos" << endl;
  for (auto& p : CJTO_SOLO_LOG)
    if (p.nota >= 6)
      cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
  return 0;
}
```

Sobrecarga de Operadores aplicado al tipo set

```
/*
  Id.Programa: G3_Ej06 ExamenFinal x 3 asignaturas SobrecargaOperadores.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: febrero-2024
  Comentario.: Multi-Conjunto (set) de struct, permite  repetir valores claves, en el caso de las notas.
                Funciones operator<, operator== para criterio de  los campos del struct sNotas:
                nroLeg, vNotas[3] de 3 asignaturas  0:AlgEstrDatos, 1:LogEstrDiscr, 2:Fisica I  con notas
                [0, 10], en donde cero alumno sin   presencia al examen, al menos uno distinto  de cero.
  Compilador.: cygwin GCC g++20.
*/

#include <iostream>
#include <algorithm>
#include <set>
using namespace std;

struct sNotas {
  int    nroLeg;
  string asignatura;
  short  nota; // entre 0 y 10, cero ausente.
};

/*
bool operator < (const sNotas& lhs, const sNotas& rhs)  {
   string cad1,
          cad2;

   cad1 = to_string(lhs.nroLeg);
   cad2 = to_string(rhs.nroLeg);
   return cad1 + lhs.asignatura < cad2 + rhs.asignatura;
}
*/

bool operator < (const sNotas& lhs, const sNotas& rhs)  {
  return lhs.nroLeg < rhs.nroLeg;
}

template <typename T>
set<T> Union(set<T> X, set<T> Y) {
  set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
```

```
  return Z;
} // Union

template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                   inserter(Z, Z.begin()));
  return Z;
} // Interseccion

template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                 inserter(Z, Z.begin()));
  return Z;
} // Diferencia

template <typename T>
set<T> operator + (const set<T> &X, const set<T> &Y) {
  return Union(X, Y);
}

template <typename T>
set<T> operator * (const set<T> &X, const set<T> &Y) {
  return Interseccion(X, Y);
}

template <typename T>
set<T> operator - (const set<T> &X, const set<T> &Y) {
  return Diferencia(X, Y);
}

template <typename T>
set<T> operator / (const set<T> &X, const set<T> &Y) {
  return DiferenciaSimetrica(X, Y);
}

template <typename T>
void EmiteCjto(set<T> CJTO, string titulo) {
  cout << titulo << " #: " << CJTO.size() << endl;
  for (auto& p : CJTO)
    cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
  cout << endl;
} // EmiteCjto

template <typename T>
void EmiteCantAprobados(set<T> CJTO, string titulo) {
```

```
  short cantAprob = 0;

  cout << titulo << " #: de " << CJTO.size() << endl;
  for (auto& p : CJTO)
    if (p.nota >= 6) {
      cantAprob++;
      cout << '(' << p.nroLeg << ',' << p.asignatura << ',' << p.nota << ") " << endl;
    }
  cout << cantAprob << " alumnos aprobados" << endl;
  cout << endl;
} // EmiteCantAprobados

main() {
  set<sNotas> CJTO_ALG =  {          {245123, "AyED", 6},{245876, "AyED", 4},
                              {245986, "AyED",10},{246233, "AyED", 2},
                              {245801, "AyED", 4},{242222, "AyED", 7},
                              {243899, "AyED", 7},{241111, "AyED", 3},
                              {245555, "AyED", 9} };

  set<sNotas> CJTO_LOG =  {          {245720, "LyED", 5},{241025, "LyED", 7},
                              {244875, "LyED",10},{245986, "LyED", 9},
                              {243002, "LyED", 2},{245801, "LyED", 4},
                              {250500, "LyED", 7},{243969, "LyED", 3},
                              {245555, "LyED", 9} };

  set<sNotas> CJTO_FIS1 = {          {244309, "FIS1", 5},{251032, "FIS1", 6},
                              {245986, "FIS1", 7},{244551, "FIS1", 4},
                              {246963, "FIS1", 8},{246860, "FIS1", 4},
                              {243905, "FIS1", 6},{242222, "FIS1", 8},
                              {246233, "FIS1", 8},{243899, "FIS1", 6},
                              {246804, "FIS1", 6},{245801, "FIS1", 4},
                              {243002, "FIS1", 8},{243969, "FIS1", 2} };

  set<sNotas> CJTO_UNIVERSAL = {      {245123, "AyED", 6},{245876, "AyED", 4},
                              {245986, "AyED",10},{245801, "AyED", 4},
                              {243899, "AyED", 7},{241111, "AyED", 3},
                              {245555, "AyED", 9},{245720, "LyED", 5},
                              {241025, "LyED", 7},{244875, "LyED",10},
                              {243002, "LyED", 2},{250500, "LyED", 7},
                              {243969, "LyED", 3},{244309, "FIS1", 5},
                              {251032, "FIS1", 6},{244551, "FIS1", 4},
                              {246963, "FIS1", 8},{246860, "FIS1", 4},
                              {243905, "FIS1", 6},{242222, "FIS1", 8},
                              {246233, "FIS1", 8},{246804, "FIS1", 6},
                              {255553, "NADA", 0},{251527, "NADA", 0},
                              {223817, "NADA", 0},{241339, "NADA", 0},
                              {249321, "NADA", 0},{249825, "NADA", 0},
                              {251108, "NADA", 0} };
  set<sNotas> CJTO_INT_ALG_LOG_FIS1,
               CJTO_INT_ALG_LOG,
               CJTO_INT_ALG_FIS1,
               CJTO_INT_LOG_FIS1,
```

```
            CJTO_SOLO_ALG,
            CJTO_SOLO_LOG,
            CJTO_SOLO_FIS1,
            CJTO_COMPLTO,
            CJTO_UNION_ALG_LOG_FIS1,
            CJTO_ALG_UNION_CJTO_LOG,
            CJTO_SOLO_LAF,
            CJTO_LOG_UNION_FIS1;

 CJTO_ALG_UNION_CJTO_LOG = CJTO_ALG + CJTO_LOG;
 CJTO_UNION_ALG_LOG_FIS1 = CJTO_ALG + CJTO_LOG + CJTO_FIS1;
 CJTO_COMPLTO = CJTO_UNIVERSAL - CJTO_UNION_ALG_LOG_FIS1;
 EmiteCjto(CJTO_COMPLTO,"A) Alumnos que no se presentaron a ninguna asignatura...");
 CJTO_INT_ALG_LOG_FIS1 = CJTO_ALG * CJTO_LOG * CJTO_FIS1;
 CJTO_INT_ALG_FIS1 = CJTO_ALG * CJTO_FIS1;
 CJTO_INT_LOG_FIS1 = CJTO_LOG * CJTO_FIS1;
 CJTO_INT_ALG_LOG  = CJTO_ALG *  CJTO_LOG;
 CJTO_UNION_ALG_LOG_FIS1 =CJTO_INT_ALG_FIS1+ CJTO_INT_LOG_FIS1 +
CJTO_INT_ALG_LOG_FIS1;
 CJTO_SOLO_FIS1 = CJTO_FIS1  - (CJTO_INT_ALG_FIS1 + CJTO_INT_LOG_FIS1 +
CJTO_INT_ALG_LOG_FIS1);
 CJTO_SOLO_ALG  = CJTO_ALG - (CJTO_INT_ALG_FIS1 +  CJTO_INT_ALG_LOG +
CJTO_INT_ALG_LOG_FIS1);
 CJTO_SOLO_LOG  = CJTO_LOG - (CJTO_INT_ALG_LOG + CJTO_INT_LOG_FIS1 +
CJTO_INT_ALG_LOG_FIS1);
 EmiteCjto(CJTO_SOLO_FIS1,"B) Alumnos que solo rindieron FISICA 1...");
 EmiteCantAprobados(CJTO_ALG,"C) Alumnos que aprobaron ALGORITMOS...");
 EmiteCantAprobados(CJTO_FIS1,"D) Alumnos que aprobaron FISICA I... ");
 CJTO_INT_ALG_LOG_FIS1 = CJTO_ALG * CJTO_LOG * CJTO_FIS1;
 CJTO_INT_ALG_FIS1 = CJTO_ALG *  CJTO_FIS1;
 CJTO_INT_LOG_FIS1 = CJTO_LOG * CJTO_FIS1;
 CJTO_INT_ALG_LOG  = CJTO_ALG * CJTO_LOG;
 CJTO_UNION_ALG_LOG_FIS1 = CJTO_INT_ALG_FIS1 + CJTO_INT_LOG_FIS1 +
CJTO_INT_ALG_LOG_FIS1 + CJTO_INT_ALG_LOG;
 CJTO_SOLO_ALG = CJTO_ALG - CJTO_UNION_ALG_LOG_FIS1;
 CJTO_SOLO_LOG = CJTO_LOG - CJTO_UNION_ALG_LOG_FIS1;
 CJTO_SOLO_LAF = CJTO_SOLO_FIS1 - CJTO_SOLO_ALG - CJTO_SOLO_LOG;
 CJTO_SOLO_LAF = CJTO_SOLO_LOG + CJTO_SOLO_ALG + CJTO_SOLO_FIS1;
 EmiteCantAprobados(CJTO_SOLO_LAF,"E) Card. alum. aprobados rindiendo solamente
ALG,LOG,FIS1 ");
 EmiteCjto(CJTO_SOLO_LAF,"E) Alumnos que rindieron solamente ALG+LOG+FIS1...");
 CJTO_LOG_UNION_FIS1 = CJTO_LOG + CJTO_FIS1;
 EmiteCantAprobados(CJTO_LOG_UNION_FIS1,"F) Card. alumnos que rindieron LOGICA_ED +
FISICA 1...");
 EmiteCantAprobados(CJTO_SOLO_LOG,"G) Alumnos que aprobaron solamente LOGICA_ED... ");
```

```
 return 0;
}
```

Tabla de Funciones Miembro para el tipo set

Funciones Miembro	Explicación y ejemplos
#include <**algorithm**>	
#include <**set**>	**set<int>** CjtoA; **set<float>** CjtoR ={2.5,83.14,32.18,3.1415};
Cjto.**size**()	Cardinalidad. **CjtoR.size()**; // 4.

Cjto.**insert**(Cjto.begin(), Cjto.end())	odd.insert(1); odd.insert(3); odd.insert(5); even.insert(2); even.insert(4);
	even.insert(6); odd_even = odd;
	odd_even.insert(even.begin(), even.end());

Cjto.**insert**(valor)	Inserta valor. **CjtoR.insert(24.3);CjtoR.insert(vecR+ 6);**
Cjto.**erase**(valor)	Elimina un valor. **CjtoR.erase(83.14);**
Cjto.**clear**()	Vacía el conjunto. **CjtoR.clear();**
Cjto.**count**(valor)	Pertenencia. **CjtoR.count(3.1415)** retorna 1. **CjtoR(12.25)** retorna 0.
Cjto.**contains**(valor)	Pertenencia (vers.20). **CjtoR.contains(3.1415)** ret. 1. **CjtoR(0.0)** ret. 0.
Cjto.**empty**()	Si vacío ret. 1 sino 0. CjtoR.empty() ret. 1 (true). CjtoA.empty() ret. 0 (false).
set<tipo>**iterator** it; Cjto.**begin**, Cjto.**end**()	Recorrer elementos de izq. a der. **set<float>::iterator** it; **for** (it = **CjtoR.begin();** it != **CjtoR.end(); it++)** cout << *it << endl; **for** (**auto const** &i : CjtoR) cout << i << endl; // vers. +actuales
set<tipo> **reverse_iterator** rit; Cjto.rbegin(), Cjto.rend()	Recorrer elementos de der. a izq. **set<float>::reverse_iterator** rit; **for** (rit = **CjtoR.rbegin();r** it != **CjtoR.rend(); rit++)** cout << *rit << endl; **for** (**auto const** &i : CjtoR) cout << i << endl; // vers. +actuales
Operaciones entre Conjuntos	
set_union (a1,a2,b1,b2, inserter(c,c1)	**set_union** (CjtoR.begin(),CjtoR.end(),CjtoQ.begin(), CjtoQ.end(), ***inserter***(CjtoS, CjtoS.begin()));
set_intersection (a1, a2, b1,b2, inserter(c, c1)	**set_intersection** (CjtoR.begin(),CjtoR.end(), CjtoQ.begin(),CjtoQ.end(), ***inserter***(CjtoS, CjtoS.begin()));
set_difference (a1, a2, b1,b2, inserter(c,c1)	**set_difference** (CjtoR.begin(),CjtoR.end(),CjtoQ.begin(), CjtoQ.end(), ***inserter***(CjtoS, CjtoS.begin()));
set_simmetric_difference (a1,a2,b1,b2, inserter(c, c1)	**set_simmetric_difference** (CjtoR.begin(), CjtoR.end(), CjtoQ.begin(), CjtoQ.end(), ***inserter***(CjtoS,CjtoS.begin()));
Módulos del Usuario	

```
template <typename T>
set<T> Union(set<T> X, set<T> Y) {
   set<T> Z;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
  return Z;
} // Union
template <typename T>
set<T> Interseccion(set<T> X, set<T> Y) {
  set<T> Z;

  set_intersection(X.begin(), X.end(), Y.begin(), Y.end(),
                inserter(Z, Z.begin()));
  return Z;
} // Interseccion
template <typename T>
set<T> Diferencia(set<T> X, set<T> Y) {
  set<T> Z;

  set_difference(X.begin(), X.end(), Y.begin(), Y.end(),
              inserter(Z, Z.begin()));
  return Z;
} // Diferencia
template <typename T>
set<T> DiferenciaSimetrica(set<T> X, set<T> Y) {
  set<T> Z;

  set_symmetric_difference(X.begin(), X.end(), Y.begin(), Y.end(),
                       inserter(Z, Z.begin()));
  return Z;
} // DiferenciaSimetrica
template <typename T>
set<T> Complemento(set<T> X, set<T> Y, set<T> U) {
  set<T> Z,
        R;

  set_union(X.begin(), X.end(), Y.begin(), Y.end(),
          inserter(Z, Z.begin()));
  set_difference(U.begin(),U.end(),
              Z.begin(),Z.end(),
              inserter(R,R.begin()));
  return R;
} // Complemento
```

Invocaciones o Llamadas a los Módulos por el Usuario
C = Union(A,B);
C = Interseccion(A,B);
C = Diferencia(A,B);

C = DiferenciaSimetrica(A,B);
C = Complemento(A,B);
for (int i = 1; i <= 1000; i++) Universal.insert(i);
const set<**short**> CJTO_VACIO;
set<int,**greater**<int>> s2; set<int,**greater**<int>>::iterator It;
set<int,**less**<int>> s1; set<int,**less**<int>>::iterator It1;

void EmiteCantCjto(set<short> CjtoX, string NomCjto) {
short CantPers = 0;
short k = 1;
 	
while (CjtoX != CJTO_VACIO) {
if (CjtoX.count(k)) {
++CantPers;
CjtoX.erase(k);
}
++k;
}
cout << "Cantidad de personas del Cjto " << NomCjto << " es: " << CantPers << endl;
} // EmiteCantCjto

set<int> Cjto = **{1, x}**; // vers. +actual.
merge(a1,a2,b1,b2, *inserter* (c,c1) / **merge**(odd.begin(), odd.end(),even.begin(), even.end(), ***inserter***(odd_even, odd_even.begin()));

Comentarios y Métodos adicionales

La implementación interna de los elementos de un conjunto se realiza a través de un **árbol de búsqueda binario**.

Los elementos de un conjunto una vez insertados **no pueden ser modificados**, pero, si **pueden insertarse** nuevos elementos o **eliminarlos**.

swap()

Intercambia todos los elementos de un conjunto a otro.

```
set<int> Cjto1 = {30, 10, 80, 40, 20};
set<int> Cjto2 = {45, 65, 15, 25};
Cjto1.swap(Cjto2);   // o equiv. swap(Cjto1, Cjto2);
```

find()

Busca el valor indicado por find y retorna un iterador al valor encontrado, sino, retorna un iterador a end().

```
set<int> iterator it;
it = Cjto2.find(20);              // Encuentra el valor 20 del Cjto2.
Cjto2.erase(it);                  // Elimina el valor 20 del Cjto2.
Cjto2.erase(Cjto2.find(40));  // Encuentra y Elimina el valor 40 del Cjto2
```

U.T.N. F.R.B.A. Algoritmos y Estructuras de Datos Prof.: Hugo A. Cuello
Estructuras estáticas II. Conjuntos.

Guía de ejercicios #3

La estructura de datos conjunto en Turbo Pascal de Borland V 7 permite operar en forma fácil muchas de las operaciones entre conjuntos, ya que se cuenta con un conjunto de operadores para realizar las operaciones. Una estructura conjunto no puede tener más de 256 elementos, y además el valor más alto no puede superar 255. Estas son restricciones impuestas por el lenguaje Turbo Pascal. Los elementos podrán ser únicamente valores ordinales cuya magnutid esté comprendida entre 0 y 255, los cuales pueden ser constantes, variables, expresiones y funciones. Los tipos bases de una estructura conjunto pueden ser byte, char, boolean, definidos por el usuario, por enumeración o intervalo. Un tipo conjunto no ocupa más de 32 bytes.

El número de bytes ocupados por un conjunto particular se calcula como:
TamañoByte = (Max div 8) – (Min div 8) + 1.
En donde Min y Max son los límites bajo y alto del tipo base del conjunto.

El número de byte de un elemento x está dado por :
NúmeroByte = (x div 8) – (Min div 8).

El número de bit dentro del byte está dado por:
NúmeroBit = x mod 8, en donde x indica el valor ordinal del elemento.

En el lenguaje C++98 de Borland en su versión 5.5 o en otros compiladores no Borland como por ejemplo g++20 brinda un conjunto de herramientas disponibles en la clase set. Estas herramientas fueron analizadas en la teoría de la Unidad III Capítulo IV.

1. Se realizó una encuesta a 100 personas, las que debieron responder a tres preguntas por si o no. A cada encuestado se le asignó un número entre 1 y 100. Se sabe que :

I.	8	personas	contestaron	bien	las 3 preguntas.
II.	9	“	“	“	la 1ra. y la 2da. solamente.
III.	11	“	“	“	la 1ra. y la 3ra. solamente.
IV.	6	“	“	“	la 2da. y la 3ra. solamente.
V.	55	“	“	“	la 1er. pregunta por lo menos.
VI.	32	“	“	“	la 2da. pregunta por lo menos.
VII.	49	“	“	“	la 3er. pregunta por lo menos.

<u>Se pide</u> :

1) Ingresar los datos por teclado (asignación externa de entrada).

   a) Cardinalidad del conjunto de las personas que respondieron bien la 1er. pregunta.
   b) Cardinalidad del conjunto de las personas que respondieron bien la 2da. pregunta.
   c) Cardinalidad del conjunto de las personas que respondieron bien la 3er. pregunta.
   d) Cuántas personas no contestaron ninguna pregunta.
   e) Cuántas personas contestaron correctamente solo dos preguntas.
   f) Cuántas personas no contestaron las tres preguntas correctamente.
   g) Cuántas personas contestaron una de las tres preguntas correctamente.
   h) Cuántas personas contestaron las tres preguntas correctamente.
   i) Cuántas personas contestaron solamente la 1er. pregunta.
   j) Cuántas personas contestaron solamente la 2da. pregunta.

k) Cuántas personas contestaron solamente la 3er. pregunta.
l) Cuántas personas contestaron solamente la 1er. y 2da. pregunta.
m) Cuántas personas contestaron solamente la 1er. y 3er. pregunta.
n) Cuántas personas contestaron solamente la 2da. y 3er. pregunta.

2) Ingresar los datos internamente (asignación interna).

3) Idem anterior pero emitiendo qué personas (número de encuestado) pertenecen a los conjuntos.

4) Idem anterior pero ingresando las respuestas de cada encuestado sin conocer a priori ningún resultado.

2. En un pub se preparan tragos largos con los siguientes ingredientes : **naranja, frutilla y ananá**, que son entregados como consumisión por el ingreso de cada cliente al pub. Se sabe que la **cantidad máxima** de gente que puede ingresar es de 250 personas, a las que se le da un número de ingreso entre 1 y 250. Puede haber personas que no hayan hecho uso de la consumisión. El fin está indicado por un número de ingreso cero.

Se pide :

a) Ingresar por cada persona qué consumición -trago largo- solicita **solo un sabor**: solo naranja, solo frutilla, solo ananá, **solo 2 sabores**: naranja / frutilla, naranja / ananá, frutilla / ananá, **los 3 sabores**: naranja / frutilla / ananá o **sin consumo**.
Considerar la **cantidad de personas** de cada área según se indica en el gráfico de más abajo, el armado de valores a cada conjunto digamos, CJTO_NARANJA, CJTO_ANANA y CJTO_FRUTILLA, se asigna el número de consumisión asignada a la persona cuando hizo el ingreso al pub, esos valores van en forma correlativa desde 1 hasta el valor máximo de ingreso de personas, cuando mucho será 250. Decimos en este caso que los valores de elementos para cada uno de los conjuntos se conocen a priori.
ALTERNATIVA
Otra alternativa de solución será **no conocer a priori**, estos valores para cada uno de los conjuntos. Esta alternativa será otro proceso en otro programa.
Construir en principio los conjuntos que representan cada uno de los sabores.
b) Determinar los números de personas que corresponden a las áreas **A1**..**A8**.
c) Determinar los porcentajes que corresponden a cada área **A1**..**A8**.
d) Determinar los promedios que corresponden a cada área **A1**..**A8**.
e) Cuántas personas ingresaron al pub.

3. Por medio de un menú de opciones, se solicita realizar las operaciones entre el conjunto A y el conjunto B, considerando que el conjunto universal son todos los

valores enteros comprendidos en el [1, 1000]. Las operaciones a realizar solicitadas a través del menú son: **la Unión, la Intersección, la Diferencia, la Diferencia Simétrica y el Complemento de la unión entre A y B**. Los valores de elementos de cada uno de los conjuntos deberán ser ingresados por el usuario, según la cantidad de elementos que es un dato previa a conocer para cada conjunto A y B. Las opciones del menú deben estar indicadas también en otro conjunto, digamos el Cjto_Opc, {'U','I','D','S','C','F'}.

4. Se solicita por medio de conjuntos determinar el **M.C.D**. Para ello, se deben ingresar 2 valores enteros digamos *num* y *den*, y de cada uno de ellos asignar a cada conjunto sus divisores, hecho esto, el siguiente paso será obtener en otro conjunto la intersección de ambos conjuntos divisores y finalmente el valor máximo de esto último conjunto será el M.C.D.

5. Se solicita utilizando conjunto, la **Criba de Eratóstenes**, esto es, los valores que queden en el conjunto, todos ellos serán números primos. El proceso a llevar en cuenta, será asignar todos los valores al conjunto Criba_Eratostenes, desde 2 en forma consecutiva de uno en uno hasta 100. Luego, se deberán eliminar todos aquellos valores que no son primos. Se asume que el alumno conoce la definición matemática que condiciones debe cumplir para que sea considerado un número primo de los números naturales.

6. Se tiene un archivo de alumnos sin orden, de los que rindieron finales correspondientes a la carrera de Ing. en Sistemas de Información de la UTN.BA, en las siguientes asignaturas : "*Algoritmos y Estructuras de Datos*", "*Física I*" y "*Lógica y Estructuras Discretas*". El archivo contiene : Nro.Leg.(7 díg.), y tres notas enteras (0..10) correspondientes a las asignaturas indicadas en el mismo orden. Se aprueba con nota >= 6. La nota con valor cero indica, no consta en acta, máximo en 2 asignaturas.

   <u>Se pide</u> :

   a) Cuántos alumnos no firmaron, ninguna asignatura sus legajos y notas.
   b) Qué alumnos firmaron solamente "*Física I*" y sus notas.
   c) Qué alumnos aprobaron y las notas obtenidas "*Algoritmos y Estructuras de Datos*".
   d) Cardinalidad de alumnos que aprobaron "*Algoritmos y Estructura de Datos*".
   e) Cardinalidad de alumnos que aprobaron <u>solamente</u> "*Algoritmos y Estructuras de Datos*", "*Lógica y Estructuras Discretas*" y "*Física I*".
   f) Los nros. de legajos y su nota de los alumnos { x / x ε "*Lógica y Estructuras Discretas*" - "*Física I*" }.
   g) Cardinalidad de alumnos que aprobaron "*Lógica y Estructuras Discretas*" solamente mostrar su nro.Leg. y la nota obtenida.

7. Realizar las operaciones entre dos conjuntos de: Unión, Intersección, Diferencia, Diferencia Simétrica y Complemento, mediante un menú apropiado.
8. Generar un conjunto de valores entre 1 y 90. Luego simular el juego del bingo en el cual participan dos jugadores. Informar el ganador al completar un cartón.
9. Obtener un conjunto de n elementos N_0, para calcular y emitir según un menú apropiado lo siguiente:
   a) El promedio.
   b) El máximo.
   c) El mínimo.
   d) La sumatoria.

Unidad IV Capítulo I

Unidad IV – Capítulo I - Archivos

Definición y características de un archivo

El archivo es la **única estructura de datos externa**, ya que en vez de residir en la memoria interna RAM reside en una memoria denominada externa, secundaria, auxiliar o también llamada masiva. Por definición un archivo es una colección de datos que obedece a una misma naturaleza. Sus componente generalmente se denominan **registros = struct**, en el sentido más amplio de la palabra, aunque no necesariamente cada componente deba ser de tipo registro = struct. Si por ejemplo, por cada componente debemos guardar un solo dato, el tipo de ese dato puede ser de tipo simple, como ser entero con o sin signo, real, carácter, booleana o incluso cadena y como veremos más delante de tipo puntero. En cambio, si por cada componente guardamos más de un dato, y cada uno de estos datos fueran de distinto tipo entre sí, en estos casos el tipo de cada componente lo estructuramos de tipo registro = struct. Por otro lado, si por cada componente guardamos más de un dato, y cada uno de estos datos fueran de igual tipo entre sí, en estos casos el tipo de cada componente lo estructuramos de tipo arreglo, que es otra estructura de datos interna como ya fuera visto. Para entender un poco más esta cuestión tomemos un ejemplo para cada caso: si un archivo tiene 10 componentes y por cada componente guardo un valor, en el archivo habrá 10 datos; por otro lado, si un archivo tiene 10 componentes y por cada componente guardo tres valores, en el archivo habrá 10 * 3 = 30 valores.

Ventajas y desventajas con los archivos

Una **primer ventaja** radica en el hecho de que la memoria interna es volátil, esto es, si se corta el suministro de la corriente eléctrica, los datos almacenados allí se pierden, y se deberá volver a ingresar nuevamente en forma manual para su recuperación, tarea inapropiada si contamos con una cantidad importante de datos, por lo cual sería lento y tedioso volver a ingresarlos. En cambio, la memoria auxiliar es independiente del suministro eléctrico ya que este tipo de **memoria es no volátil**, los datos se pueden resguardar y a futuro rápidamente recuperados ya que es un proceso puramente electrónico para llevar a cabo esta tarea de recuperación de los datos.

Una **segunda ventaja** de los archivos, consiste en que la memoria interna es de un espacio reducido, por lo que no podremos dar cabida en forma simultánea a una gran colección de datos, por el contrario, la **memoria masiva presenta una capacidad mucho mayor**, casi podríamos decir inconmensurable, ya que podríamos incorporar memoria adicional según se lo requiera, por lo que contamos y de hecho es posible, guardar una cantidad enorme de datos sin poner el riesgo de quedarnos sin espacio de almacenamiento.

No obstante, debemos aclarar que **existe una desventaja** con los archivos siendo el **tiempo de acceso a los datos**, ya que la unidad que consideramos para acceder a una posición en la memoria RAM es el nanosegundo, esto es, 10^{-9} segundos, o sea, 1.000.000.000 (léase mil millones) de fracciones de segundo, en cambio, para acceder a una posición de memoria externa requiere del orden de 10^{-3} segundos, o sea, la unidad de medida es el milisegundo. Por lo tanto, observamos que existe una gran diferencia de tiempo entre acceder a una posición en la memoria interna con respecto a una posición de memoria externa.

La necesidad de poder **recuperar** los datos en momentos posteriores a su creación, ya sea, por haberse cortado el suministro de la corriente eléctrica o en distintas corridas del programa o en distintos programas, por un lado, y por otro ante la **imposibilidad de poder contar con todos los datos simultáneamente en la memoria interna** debido a su limitación de espacio, son dos motivos que hacen de la necesidad de contar con este tipo de estructuras de datos.

El archivo es la única estructura de datos externa, es decir, ubicadas en un dispositivo externo, al cuál se los denomina memoria auxiliar, o memoria secundaria o memoria externa.

La desventaja principal de este tipo de estructura es el tiempo necesario para recuperar un dato, ya que estos tiempos se miden en milisegundos, esto es *10^{-3} segundos* en comparación con el tiempo empleado para acceder a una posición en la memoria interna cuya unidad de medida es el nanosegundo, esto es *10^{-9} segundos*; **por lo tanto, al momento de requerir un dato desde un archivo, debemos tomar muy en cuenta esta última situación, tratando de minimizar estos tiempos.**

Las componentes de un archivo se denominan registros, y en la mayoría de las situaciones, estas componentes serán de tipo registro. No obstante, en ciertas situaciones podrán ser de un tipo simple de datos como *integer, word, char, boolean, longint o punteros*, como así también de otro tipo estructurado de datos que se verán más adelante.

Un archivo es por lo tanto, una **colección de registros** que responden a una misma naturaleza, p.e. Artículos, Clientes, Proveedores, Empleados, Cuentas Contables, etc..

Clasificación de archivos de acuerdo a su función de uso

Datos

Maestros

Son archivos **permanentes** en el tiempo, es decir, no se eliminan luego de un proceso. Contienen todos los datos necesarios para el desarrollo de las actividades de una organización. Representan al **mundo real**. Con el transcurrir del tiempo deben ser actualizados. Dependiendo del momento en que se actualizó, da un grado de confiabilidad. Ejemplo de archivos maestros pueden ser, los Clientes, Proveedores, Empleados, Artículos, Cuentas Contables, etc.

Novedades o Transacciones

Son archivos transitorios, es decir, luego de ser procesados, no tiene sentido mantenerlos, por lo tanto son eliminados. Su cometido es generalmente la actualización de los archivos maestros. La eliminación se podrá realizar inmediatamente o bien luego de un período de tiempo, por ejemplo, después de una segunda actualización al maestro. No siempre existen estos archivos, esto depende del tipo de proceso que se lleve a cabo.

Por ejemplo si el proceso es *interactivo* en tiempo real, esto es, en el momento en que se conoce la novedad se actualiza en el maestro, no existirá un archivo de novedades. También se podrán generar registros por cada novedad que se presente en un proceso en tiempo real interactivo para control. En cambio, si el proceso es en *batch* o por lotes, primero se capturan las novedades durante el transcurso de un tiempo, -un día, una semana, un mes- se los ordena bajo un cierto criterio.

Históricos

Son archivos cuyo uso generalmente son para fines estadísticos, por ejemplo, las ventas realizadas por mes de un año, el seguimiento de ciertos artículos más solicitados, procesos de períodos anteriores, etc..

Tablas

Son archivos de poco volumen, ya sea en cantidad de registros o con respecto a su longitud del mismo. A efectos de ganar velocidad durante el proceso, estos tipos de archivos pueden ser volcados a la memoria interna –RAM- para acelerar el proceso, debido a que acceder a un registro en un archivo la unidad de medida es el *milisegundo*, esto es, 10^{-3} *seg.*, en cambio acceder a una ubicación en la memoria interna, la unidad de medida es el *nanosegundo*, es decir, 10^{-9} *seg.*, razón por la cual se ve la enorme diferencia existente entre acceder a una u otra fuente. El volcado del archivo se realiza en una pasada secuencial, luego al requerir acceder a un dato se accede en la propia memoria interna. Según los procesos tal vez no sea necesario bajar todos los registros como así también bajar todos los campos del mismo. Por ejemplo un archivo que contenga los códigos de las provincias y un porcentaje que fijan las mismas por las ventas realizadas a esas provincias, un proceso que requiera esos datos, podrían ser volcados a la memoria principal, recorriendo secuencialmente de inicio a fin sobre este archivo, luego cada vez que se requiera averiguar el porcentaje de una provincia se accede a la posición de memoria interna. Más adelante se verá la manera de lograr este cometido con una estructura de datos estática que se estudiará posteriormente.

Índices

Son archivos que contienen 2 ó 3 campos generalmente, un campo denominado *clave* y un campo denominado *referencia* o *dirección* el tercer campo si existe es denominado *estado*. Estos archivos se encuentran ordenados por el campo clave. Su objetivo es indexar al archivo maestro u otros archivos de datos. Esto permite una **búsqueda** más eficiente y establece un orden **lógico** de esos datos. Podrán existir varios archivos de índices para un mismo archivo de datos y c/u. de ellos estará ordenado por el campo clave que corresponda. Para mayor información ver organización indexada.

Auxiliares

Son archivos que crea y elimina el programador y que son necesarios para mejorar la eficiencia de un proceso. Por ejemplo, un archivo de vendedores desordenado y un proceso que requiera acceder a los registros en forma reiterada o en distinto orden al que se grabaron los mismos. Está claro que al buscar un vendedor debemos recorrer el archivo secuencialmente y esto por cada vendedor que requiera el proceso, nada eficiente será el proceso, por lo cual debemos recurrir a alguna técnica que permita optimizar el proceso. Una de las técnicas que podrían emplearse es crear un archivo auxiliar de tal manera que ordene esos registros de alguna manera y permita un mayor acceso a los mismos. Por ejemplo si los vendedores están identificados con un Código

de Vendedor de 1 a 999, se crean anticipadamente esos registros, luego al ir leyendo cada vendedor en el archivo original se lo ubica en la posición física en el archivo auxiliar en la posición indicada por el Código del vendedor leído. Posteriormente se realiza el proceso principal pero esta vez, al buscar un vendedor se accede en el auxiliar en la posición indicada por el código del vendedor, al leer el registro su valor indica la posición en el archivo original del vendedor para acceder a esa ubicación y leer los datos del mismo.

Informes o Reportes

Son archivos cuyo destino original era la impresora, pero debido a que ésta ya estaba ocupada por otro proceso, el sistema operativo lo redireccionó hacia otro dispositivo, es decir, un archivo en disco para que posteriormente cuando la impresora sea liberada y la prioridad le sea asignada ese archivo sea volcado a la impresora, una vez que la tarea se llevó a cabo, el mismo sistema operativo elimina ese archivo. La parte del sistema operativo que realiza este cometido es el **S.P.O.O.L.** (Simultaneous Peripherical Operation On Line), es decir, Operaciones Periféricas simultáneas En Linea, el cual redirecciona las salidas de los procesos a archivos en disco debido a que el destino original, la impresora, estaba ocupada y por ser un recurso no compartido debió ser enviado a otro destino, armando una cola de espera.

Seguridad

Son archivos que se realizaron copias de otros archivos y en caso de pérdida de uno de ellos poder recuperarlos con el otro. Esto se conoce como *back-up*.

Programas

Fuentes

Son los archivos escritos en un lenguaje de computadora y de tipo texto. La extensión de estos archivos se corresponde con el lenguaje utilizado, por ejemplo, *.Pas*, *.C*, *.Cob*, *.Bas*, *.Prg*, etc. Estos archivos son creados utilizando un editor de texto, por ejemplo el Word pero tipo texto, el bloc de notas, no son los más apropiados, otra forma es utilizar el propio editor de texto incorporado en el paquete de software del lenguaje. Así por ejemplo el **Turbo Pascal de Borland** viene un entorno de trabajo denominado **I.D.E.** –Medio ambiente de Desarrollo Integrado- en el cual no solo podremos editar nuestro código fuente, sino además compilar, ejecutar, depurar, entre otros aspectos. Además nos facilita la escritura ya que las palabras reservadas se escriben resaltadas al resto de las otras palabras.

Objetos

Son archivos resultado del proceso de compilar el código fuente. El compilador es una aplicación, es decir, un programa ejecutable que toma como parámetro el código fuente y produce como salida un archivo o programa objeto, cuya extensión es *.OBJ*, el cual aún no puede ser ejecutado debido a que le faltan las librerías a que hace referencia. Cada lenguaje posee su propio compilador.

Ejecutables

Son archivos resultado del **enlace** con las librerías para que sean incorporadas al código máquina y pueda correr en forma autosuficiente. El *link* es una aplicación que toma como parámetro el código objeto y el resultado final es un archivo ejecutable *.EXE*.

Otros:

Documentos

Son archivos creados con un procesador de palabra, por ejemplo el Word.

Imágenes gráficas

Presentan diferentes formatos .GIF, .JPG, .BMP, etc. Son creados por software graficadores como el Paint, Corel, Harvard, PhotoShop, etc.

Audio

Archivos de sonido o música con extensiones como ser .MP3, .WAV, .MID, etc.

Miscelánea

Librerías dinámicas .DLL, Sistema .SYS, Dispositivos .DRV, etc..

Clasificación de lenguajes

Los lenguajes de computadoras pueden ser clasificados en cuanto al mayor acercamiento hacia la máquina o hacia el usuario en:

Bajo nivel i++;

Son lenguajes que se acercan más a la máquina que al usuario. Cada instrucción se traduce en una única instrucción de máquina, se dice entonces que la relación es 1 a 1; el lenguaje de bajo nivel es el assembler y la aplicación que lo convierte a código máquina se denomina ensamblador. La característica más emblemática es que los programas ejecutables son los más veloces. La desventaja es que es más compleja su programación. La extensión de los archivos de código fuente presentan la extensión *.ASM*. Por ejemplo sumar un valor a una variable, se escribiría:

Medio nivel i += 1;

Son lenguajes que se encuentran en un punto intermedio entre los de bajo y alto nivel. Lenguaje como C o Forth entran dentro de esta categoría. La forma en como se escribe el código en algunos casos puede llevarlo a un nivel más bajo o más alto, por ejemplo acumular un valor en una variable podría escribirse en lenguaje C de varias maneras diferentes, pero una de ellas generará un código de máquina más eficiente que las otras. El siguiente ejemplo muestra esto último: sum = sum + 1; sum+= 1 o ++sum o sum++; en los dos últimos casos generará un código más eficiente.

Alto nivel i = i + 1;

Los lenguajes de alto nivel se acercan más al usuario que a la máquina y los programas escritos en código fuente se asemejan al lenguaje natural. Estos programas corren más lentos que los de bajo nivel. Una sentencia suele ser convertida a varias instrucciones en código máquina. Lenguajes como Pascal, C, Cobol, Basic, Fortran, Modula, Ada, Prolog entre otros son de alto nivel. Por ejemplo, acumular un valor en una variable sería:

sum := sum + 1	Pascal
inc(sum)	Pascal
add 1 to sum	Cobol
sum = sum + 1	C
sum += 1	C
sum++	C

++sum C

Otras clasificaciones de los lenguajes podrían realizarse en cuanto al objetivo en que fueron concebidos, así existen lenguajes de propósito general, como ser el BASIC, PASCAL; otros destinados a la gestión y administración contable como el COBOL, RPG aún otros con fines científicos como el FORTRAN y otros para el desarrollo de soft de base como el C.

Traductores

Los traductores pueden ser de dos tipos diferentes:

1. **Intérpretes**
2. **Compiladores**

Intérprete

En el primer caso los lenguajes **intérpretes**, la ejecución se realiza dentro de un entorno de trabajo del lenguaje, y se ejecuta desde alli, o por medio de una aplicación, es decir un módulo de tiempo de ejecución que toma como parámetro el código fuente, cada sentencia a ejecutar primero debe ser interpretada a su equivalente en código máquina, generalmente una sentencia se divide en varias instrucciones de máquina, luego se ejecuta, esto se repite por cada sentencia que deba ser ejecutada, aún en los casos en que una misma sentencia se ejecute más de una vez deben de realizarse esos pasos; no se genera ningún código objeto en disco. Este tipo de lenguajes es oportuno cuando se está desarrollando la aplicación, en la cual tendremos que ejecutar el programa varias veces para refinarlo en detalles, hasta que quede el definitivo, entonces debido a que no hay tiempo de espera para la compilación total del programa; se hace conveniente en esos momentos; pero no cuando el programa haya quedado terminado de corregir detalles. **El tiempo de ejecución es mayor en un programa interpretado que si fuera compilado**. La ventaja es que entre cada ejecución del programa en la etapa de depuración no debemos esperar por el proceso de compilación, en la que muchas veces se demora bastante tiempo.

Compilador

Por otro lado un lenguaje **compilado** primero se compila todo el código fuente, creándose un código máquina y guardado en un archivo con extensión .OBJ, luego en un segundo proceso se le incorporan las librerías produciendo un código ejecutable y guardado en un archivo con extensión .EXE. En este momento podremos correr o ejecutar la aplicación o programa. El tiempo insumido será mucho menor a un programa interpretado, debido a que el código fuente fue traducido a código máquina con anterioridad y solamente el proceso se centra en ejecutarlo.

Existen lenguajes que son solamente interpretados y otros que son solamente compilados, pero también existen lenguajes que pueden correr con un intérprete y que además puedan ser compilados, una vez que se hayan depurados ciertos errores. Por ejemplo un programa realizado en lenguaje Basic -ciertas versiones- puede ser solo interpretado o si el programador lo desea compilado. Otros como la mayoría de los lenguajes sólo compilado, por ejemplo, COBOL, C, PASCAL, ALGOL, CLIPPER, etc.

Etapas de procesos de los archivos en el tiempo

A continuación se presenta otra clasificación de archivos de acuerdo a distintos procesos que podemos realizar:

- **Creación**
- **Actualización:**
  - Altas
  - Bajas
  - Modificaciones
- **Recuperación:**
  - Consultas
  - Informes
- **Mantenimiento:**
  - Estructuración
  - Organización

Modo de apertura de archivos

Entrada –Input-

Un archivo abierto como solo de entrada indica que solo se podrá **leer**, hacer un intento de escritura ocasionará en un error. En el diagrama de flujo el símbolo es el bloque del trapecio invertido, base menor hacia abajo.

Salida –Output-

Un archivo abierto como solo de salida indica que solo se podrá **grabar**, hacer un intento de lectura ocasionará en un error. En el diagrama de flujo el símbolo es el bloque del trapecio, base mayor hacia abajo.

Entrada / Salida -Input/Output-

Un archivo abierto en el modo de lectura-escritura indica que se podrán realizar ambas operaciones, es decir, leer y/o grabar indistintamente.

Modo de acceso a los registros

Secuencial

Se recorren los registros uno a continuación del otro, es decir, en forma adyacente o contigua. Para alcanzar el registro que ocupa la ubicación n debemos recorrer todos los registros que lo preceden. El tiempo empleado para accesar un registro n depende del lugar en que se encuentre ubicado el puntero al archivo.

Al azar

El tiempo empleado para accesar un registro **no** depende del lugar en que se encuentre ubicado el puntero al archivo, debido a que se accede al registro n directamente, vale decir que, el tiempo empleado para acceder a cualquier posición es el mismo desde el lugar en que se encuentre el puntero al archivo.

Por ejemplo, el control remoto de una TV vía satélite o por cable permite 2 tipos de accesos, uno secuencial, y otro al azar; en el primer caso, si se oprimen las teclas CH+ o CH- permite un acceso secuencial a los canales; por otro lado,

si tipeamos un número, -con las teclas numéricas-, cambiamos directamente a ese canal. ¡Imaginarse, por ejemplo, el tiempo empleado de acceder del canal 182 al canal 534 en forma secuencial!.

Organizaciones de archivos

Secuencial

La organización secuencial es aquella en la cual los registros solo pueden ser accedidos en forma secuencial. Además la apertura del archivo solo se realiza exclusivamente para entrada o exclusivamente para salida, en el primer caso solo se lo puede leer y en el segundo caso solo se lo puede grabar. Tanto la lectura o grabación se lleva a cabo hacia delante y no se permite retroceder a una posición previa. Los medios de almacenamiento por naturaleza son las cintas magnéticas pero también pueden ser los discos. En esta organización los registros suelen ser sometidos a un ordenamiento establecido. Además los registros podrán estar agrupados, contenidos en un bloque, esta técnica se conoce como *factor de bloqueo* en donde se determina un valor ***n***, siendo este valor n la cantidad de registros contenidos en un bloque, esta técnica es utilizada en archivos con acceso secuencial para lograr un menor tiempo de procesamiento cada vez que se acceda al dispositivo externo. Un bloque representa un *registro físico*, mientras que los ***n*** registros contenidos en él representan ***n*** *registros lógicos*. Este tipo de organización es oportuna cuando los registros deban ser leídos en el mismo orden en que fueron grabados y deban ser procesados la mayoría de ellos.

Indexada

La organización indexada presenta básicamente 2 archivos, uno de **datos** similar a la organización secuencial, es decir, los registros que se incorporan son agregados al final del archivo y normalmente no estarán ordenados físicamente. El otro archivo es el de **índice** y puede contener 2 ó 3 campos, a saber, un campo que contendrá la *clave* y que debe formar parte en el archivo de datos, pudiendo ser de cualquier tipo pero el tipo preferido es el de cadenas, debido a que en muchas oportunidades se suelen combinar los valores de dos o más campos por medio de la operación de concatenación. El segundo campo representa la *referencia* o *dirección*, y que indica en donde se encuentra el valor de esa clave en el archivo de datos. Por último en caso de existir el tercer campo establece un *estado* para informar si ese registro debe ser tenido en cuenta en los procesos ya que podría habérsele dado de baja, con un valor podría indicar que estará activo o caso contrario estará inactivo, on u off, verdadero o falso, 0 ó 1. En caso en que el registro no debe tomarse en cuenta es debido a que se ha realizado una baja lógica, es decir, el registro sigue ocupando un lugar físico en el medio externo. Las bajas físicas en estos casos se realiza cuando se vayan acumulando varias bajas lógicas, ya que este proceso requiere de un mayor tiempo de proceso.

A continuación se presentará un ejemplo con valores en el archivo de datos para construir el archivo de índices. El siguiente archivo de datos puede representar datos de Alumnos y a efectos de resumir, solo se mostrará por cada registro el campo NroLeg.

<u>Archivo de datos</u>: Alumnos.Dat

valores del campo Nro.Leg. x c/registro.

43	27	38	75	12	89	62	31	56	19	94	83	42	5	98
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

direcciones físicas de cada registro.

Archivo de índices

Alumnos.Idx

Para construir la tabla de índices se comenzará leyendo el primer valor del archivo, luego el siguiente y así sucesivamente, pero si vamos escribiendo en la tabla de más abajo notaremos que debemos insertar por cualquier lugar de la lista que vayamos armando y en el caso de escribirlo sobre papel es casi una tarea imposible; por esta razón vamos a implementar otro esquema gráfico que evitará la situación indicada anteriormente.

En el primer caso tendremos el siguiente problema:

CLV	REF.
~~43~~ 27	~~0~~ 1
43	0

Luego de haber copiado el valor 43 y su referencia 0, al leer el siguiente dato vemos que el valor clave 27 es menor al ingresado previamente, si queremos mantener un orden ascendente deberíamos escribir este valor en una línea previa, pero sin antes borrar el valor 43 y su referencia para luego escribirlos más abajo. Notamos luego que al continuar con los próximos valores esta tarea se vería más complicada, razón por la cual adoptaremos otro criterio gráfico.

Un nuevo nodo se inserta siempre como nodo hoja o terminal.

El gráfico representa un árbol binario, el nodo raíz es el punto de entrada para incorporar un nuevo valor, por lo tanto por cada **valor nuevo** que ingresa se compara si es **menor** al **valor del nodo actual**, si es así se dirige a la **izquierda**, **caso contrario** hacia la **derecha**, en caso de no existir un nodo se crea uno nuevo, siempre como nodo hoja. Si seguimos este criterio, los valores irán ubicándose de la forma en que quedaron arriba. Esta forma depende de cómo se vayan conociendo los valores claves, ya que si esos mismos valores claves estuvieran acomodados de una manera diferente, diferente sería entonces la estructura que adoptaría el árbol.

Ahora vamos a volcar estos valores a nuestra tabla original, para ello debemos saber como se procede a recorrer el árbol, la forma de hacerlo podría ser **in-orden IRD**[4], en **pre-orden RID**[1] o en **post-orden IDR**[1]. Vamos a recorrer in-orden. En un árbol binario

[4] IRD significa: Izquierda, Raíz, Derecha. RID significa: Raíz, Izquierda, Derecha. IDR significa: Izquierda, Derecha, raíz.

tenemos un sub-árbol izquierdo y un sub-árbol derecho, así el nodo con valor 27 es el nodo raíz del sub-árbol izquierdo, y el nodo con valor 75 es el nodo raíz del sub-árbol derecho, esta división en sub-árboles se puede continuar con los restantes nodos.

El recorrido se inicia por el nodo raíz, en este momento averiguamos si hay algún nodo hacia la izquierda, si es así descendemos un nivel, nuevamente hacemos lo mismo hasta llegar a un nivel en que no haya nodos a izquierda, entonces tomamos este valor en nodo raíz del sub-árbol y marcamos al nodo como visitado, en el ejemplo este valor es 5. Luego averiguamos si hay nodos a derecha, si es así descendemos un nivel y volvemos a aplicar el mismo criterio indicado anteriormente. En el ejemplo el siguiente valor a tomar es el 12 que se encuentra un nivel más arriba, debido a que no había nodos a derecha del nodo con valor 5. Luego el recorrido es hacia la derecha del nodo con valor 12, y así seguiremos avanzando y retrocediendo. Esta técnica es conocida como *back-tracking*.

A continuación se expondrá en el archivo de datos indicado anteriormente otro valor para cada registro, por ejemplo, el Apellido de cada alumno, para simplificar la escritura lo indicamos de la siguiente manera:

Nodos hojas son los nodos terminales. Un nuevo nodo se inserta siempre como nodo hoja o terminal.

Conocido el valor de una clave, ¿qué método de búsqueda aplicar?. Si el método de búsqueda fuera **secuencial** comenzaríamos desde el primer valor en la tabla si es el que buscamos detenemos la búsqueda e indicaremos que el valor se encontró, sino puede que sea mayor el valor a buscar en ese caso seguiremos buscando con los próximos valores de la tabla hasta posiblemente encontrarlo o bien detenernos cuando aparezca un valor mayor al que buscamos o bien termine la tabla, en estos casos se informará que el valor no se encontró en la tabla.

Tabla: NroLeg

CLV NroLeg	Ref.
5	13
12	4
19	9
27	1
31	7
38	2
42	14
43	0
56	8
62	6
75	3
83	11
89	5
94	10
98	12

Tabla: ApeNom

CLV ApeNom	Ref.
Álvarez	5
Benítez	13
Fernández	1
Fernández	10
García	9
González	2
Juárez	11
López	6
Martínez	0
Pérez	3
Rios	14
Sosa	7
Suárez	4
Torres	8
Vázquez	12

Si el valor clave está, entonces el siguiente paso será acceder con la referencia indicada en la tabla, al archivo de datos a esa misma posición, para obtener los datos requeridos.

Ahora bien, ¿será este método el secuencial el más adecuado cuando la tabla de índices se encuentra ordenada por la clave a buscar?. La respuesta es **NO**. Un mejor método es realizar una búsqueda **binaria**, aquella que parte en forma sucesiva por la mitad entre los valores *mínimo* **pri** y *máximo* **ult** de las direcciones de los registros extremos, es decir, en donde posiblemente se pueda encontrar el valor de la clave. Encontrado el punto medio entre los extremos *(pri + ult) / 2*, se compara el valor a buscar con el valor de la posición de este punto medio, si se encontró se abandona la búsqueda, informando que se encontró el valor, sino puede suceder que sea mayor o menor, en cualquiera de los casos se acorta la tabla por su mitad, es decir, cambia el valor extremo menor **pri** por el de **med** + 1 o bien cambia el valor extremo mayor **ult** por el de **med** – 1, esto se repite hasta encontrar el valor a buscar, si es éste el caso se accede al archivo de datos en la dirección indicada por el **campo ref**. de la clave encontrada en la tabla de índices, o bien si el valor extremo menor **pri** se hizo mayor al valor extremo mayor **ult**, se abandona la búsqueda e informa que el valor clave no se encontró en la tabla. Se establece que la cantidad máxima de comparaciones a realizar con N componentes, está dado por la siguiente expresión: $log_2 N$, y considerando a **N = 50.000**, la cantidad máxima de comparaciones será de **16**.

En el árbol si buscamos el valor 31, debemos pasar por los siguientes nodos: 43, 27, 38 y 31.

Un árbol **unialargado** es aquel árbol en que crece solamente por una de sus ramas, izquierda o derecha para todo nodo.

Ejercicios

1. ¿Cuál sería la estructura de un árbol si las claves vienen ordenadas en forma ascendente?.
2. Idem anterior pero con las claves ordenadas en forma descendentes.

Las claves en una organización indexada pueden ser *primarias* o *secundarias*. Si la clave es primaria identifica **unívocamente** a un registro y su valor **no puede repetirse** en la tabla. En cambio si la clave es secundaria, puede que se repita o no.

Ejemplos de claves primarias y secundarias:

Claves Primarias: *NroLeg* en el archivo maestro de Empleados, *CodArt* en el archivo maestro de Artículos, etc.

Claves Secundarias: *CodPos* en al archivo maestro de Empleados o Clientes o Proveedores, notamos en este caso que diferentes empleados o clientes o proveedores pueden tener su domicilio bajo un mismo código postal, por lo tanto esta clave secundaria será con duplicación o repetición.

Para un mismo archivo podrán presentarse varias claves candidatas a ser clave primaria, la elección queda más justificada cuanto más compacta sea la misma.

Las claves también pueden ser **simples** o **compuestas**. Una clave simple es la que se forma con el valor de un solo campo; mientras que una clave compuesta se forma con la concatenación de dos o más valores de campos.

Los valores de las claves deben formar parte en el archivo de datos. Si la clave es de tipo cadena es mejor, ya que en los casos de una clave compuesta tal vez se requiera concatenar dos valores de campos diferentes. Si uno de los campos no es de tipo cadena, no hay problema, ya que se lo podrá convertir a cadena.

Relativa

La organización relativa se denomina así porque las posiciones que ocupan los registros dentro del archivo son direcciones **relativas** y no **absolutas**. Debido a que las posiciones de los registros del archivo se comienzan a contar desde el punto de inicio del archivo y no desde el punto de inicio del disco. La organización relativa está considerada como la organización de archivos con mayor velocidad de acceso a los registros generalmente, aunque no siempre será así, dependiendo de los distintos casos que se presenten, como se verá más adelante. Esta organización de archivo junto con la organización secuencial han sido las primeras organizaciones en los inicios de la informática, en este caso precisamente, porque se requería un acceso a los registros que no fuera necesariamente secuencial. En cambio la organización indexada surgió con posterioridad, hoy ampliamente utilizada esta última en las bases de datos. Para poder acceder a un registro n en una organización relativa debemos conocer el valor de una clave debiendo ser de tipo numérico, ya que las direcciones en el archivo son valores numéricas. No existe un archivo de índices como en la organización indexada, solo un archivo de datos. Este tipo de organización requiere la habilidad del programador para manejarlo adecuadamente. Esta habilidad estará emparentada con la experiencia lograda con los años de trabajo en diferentes proyectos que haya realizado en su vida profesional. No obstante, algunos detalles podrán mostrarse.

Tipos de direccionamiento en una organización relativa

Existen dos tipos de direccionamiento en una organización relativa

Direccionamiento directo

Es cuando se accede una sola vez al archivo a un registro direccionado, siendo ese el registro requerido. En ciertos casos esa dirección será igual al valor de la clave, en otros casos, esa dirección se obtiene aplicando al valor de la clave una función denominada en forma genérica función **hashing**, que es una función de mapeo y, cuando aplicando esta función de mapeo o hasing obtiene para cualquier clave conocida direcciones de memoria válidas, se dice que es una función hashing perfecta, el cual no provoca *colisiones* o *sinónimos*.

Direccionamiento indirecto

Es cuando se accede más de una vez al archivo para localizar o establecer el valor de la clave en una ubicación en el archivo, debido a que ocurrieron *colisiones* o *sinónimos*.

A continuación se darán varios casos de estudio

Caso 1: En una entrevista con un cliente, nos informa llevar el proceso de un archivo de artículos, máximo 100 y la forma de identificar a cada artículo, se combino en numerarlos de 1 a 100. Notamos aquí una **relación 1 a 1** entre el código del artículo y la dirección que le debe corresponder en el archivo. Así conocida la clave de un artículo, digamos 23 le corresponde la dirección 23 en el archivo. Notamos entonces que el acceso a dicho registro tanto para leer como para grabar lo logramos con un solo acceso. Cuando se presenta esta situación el **direccionamiento** se denomina **directo**.

Caso 2: La situación con el cliente se presenta semejante al caso anterior pero los códigos de artículos deben ser entre 1001 y 1100. En este caso no podemos decir que conocida una clave con ese valor nos ubicamos en la dirección correspondiente en el archivo, ya que no existirán direcciones en el archivo con ninguno de esos valores claves. Pero no todo está perdido aún. Notamos enseguida que el intervalo de valores es igual a la cantidad de posibles artículos que podemos contar como máximo y que solo están desplazados 1000 posiciones, por lo tanto, si restamos ese valor a cualquier clave que conozcamos solucionamos el problema y obtendremos una dirección válida en el archivo, es decir, una dirección entre 1 y 100. Por ejemplo, si conocemos una clave cuyo valor fuera 1023, le restamos 1000, nos queda la dirección 23, siendo válida esta dirección para acceder en el archivo. Por lo tanto aplicando un cálculo a la clave hemos obtenido una dirección válida en el archivo. La relación 1 a 1 sigue existiendo para este caso, solo debemos realizar un cálculo para obtener una dirección válida en el archivo. El tipo de direccionamiento también es directo, ya que solo se requiere de un acceso para localizar un artículo.

Caso 3: La situación con el cliente se presenta semejante a los casos anteriores salvo que los códigos de artículos deben estar comprendidos entre 3427 y 7965. En este caso vemos que al igual del caso 2 con el valor de una clave no podemos ubicarnos en una dirección válida en el archivo, pero además notamos que el intervalo de valores de las claves es mucho mayor a la cantidad de artículos máximos que podemos tener. Por lo tanto esto trae aparejado un nuevo problema, el de las **colisiones** o **sinónimos**. Esto se produce cuando dos o más valores claves distintas generan una misma dirección en el archivo. Ahora bien, ¿cómo lograr obtener a partir de un valor clave una dirección válida en el archivo?. La técnica empleada se conoce como función de *mapeo*. Una función de mapeo ampliamente utilizada es la del **método del resto**, que consiste en tomar el valor clave k por un lado y el tamaño del archivo TA por otro, o

bien por un número primo más cercano al tamaño del archivo; se realiza la división en donde k es el dividendo y TA o el número primo el divisor; el cociente entero es la *dirección natural* $\boldsymbol{d_n}$ para acceder al archivo de datos. Si las direcciones en el archivo comenzaran desde uno en adelante, en estos casos se le suma 1 al resto.

Notamos que valores de claves distintas digamos $\boldsymbol{k_i \neq k_j}$ pueden ocasionar una misma dirección natural d_n, produciéndose entonces una colisión o sinónimo. Este problema es solucionable existiendo distintos métodos para resolverlo. Uno de ellos sería buscar en los registros adyacentes un lugar libre y una vez localizado ubicar los datos allí, en caso de estar dando un alta. Si llegamos al final del archivo, debemos siguir buscando desde el inicio del mismo. Si lo que deseamos es buscar el valor de una clave primero accedemos a su dirección natural si es el dato que hay allí lo encontramos y listo, sino seguiremos buscando con el método empleado para solucionar colisiones, por ejemplo buscar en los registros adyacentes, así seguiremos hasta encontrarlo o bien detenernos en un punto en donde se asegura que ese valor clave no se encuentra en el archivo. Una solución a esto último sería agregar un campo adicional al registro que indique un estado del registro con un valor de V indicando que el registro está vacante, nunca fue utilizado por alguna clave, O indicando que el registro está ocupado, S indicando que el registro fue suprimido, es decir, se realizó una baja lógica. Al dar de alta si su dirección natural estuviera ocupada, la clave nueva se ubicará en el primer registro que esté como vacante o suprimido y se marca el estado con O de ocupado, siempre y cuando esa clave no estuviera en el archivo, sino sería alta existente. Al dar de baja una clave, si no es la de su dirección natural se seguirá buscando hasta que aparezca en el archivo, en ese caso se marca el estado con S de suprimido, siendo una baja lógica, o bien seguir buscando hasta que se encuentre un registro como vacante o se haya completado la vuelta, en estos casos sería una baja inexistente.

En este último caso el direcionamiento se denomina **indirecto**, debido a que encontrar una clave podrá necesitar realizar más de un acceso.

Caso 4: La situación con el cliente se presenta semejante a los casos anteriores salvo que los códigos de artículos ahora son valores de tipo cadena de 5. En este caso vemos que con el valor de una clave no podemos ubicarnos en una dirección válida en el archivo, ya que la clave no es numérica, y como fuera dicho anteriormente, es de esperar que la clave sea numérica, ya que las direcciones en el archivo son posiciones numéricas, por lo que en estas situaciones, debemos convertirla en numérica; una forma sería tomar el cad [i] tomar su código ASCII y multiplicarlo por un múmero primo diferente, para cada cad [i], y acumular los resultados parciales. De esta manera ahora contamos con un número bastante grande, por lo que es de esperar, que este valor numérico no se corresponda con ninguna dirección válida en el archivo; pero además notamos que el intervalo de valores de las claves es mucho mayor a la cantidad de artículos máximos que podemos tener. Por lo tanto esto trae aparejado el mismo problema, que en el caso 3, el de las **colisiones** o **sinónimos**, por lo que aplicamos la misma técnica vista en el caso 3, para solucionar el problema de las colisiones o sinónimos, por lo que a partir de este instante, se soluciona como fuera indicado en el caso 3.

También en este caso el direccionamiento se denomina **indirecto**, debido a que encontrar una clave podrá necesitar realizar más de un acceso.

Para utilizar archivos con organización relativa se requiere amplia experiencia, debido a que es la organización de archivos más compleja de todas porque requiere una exigencia por parte del programador en el diseño del archivo, las funciones de mapeo y la resolución de las colisiones.

Estado del puntero al archivo en situación de leer o grabar

Si el puntero al archivo está ubicado en la marca de **eof()**, realizar una operación de lectura producirá un error en tiempo de ejecución activa el flag en un valor de -1, en cambio si movemos el puntero más allá de la marca de **eof**(), grabar esta bién y se expande el tamaño del archivo una componente más y la marca se reubica después del registro grabado, coincidiendo con el puntero al archivo.

Si el puntero al archivo está ubicado en una dirección más alejada digamos 15 y se graba un nuevo registro, también se graban los registros 12 a 14 pero su contenido será desconocido.

Archivo binario con 12 registros con direcciones de componentes [0, 11].

Si el puntero al archivo está ubicado en una dirección menor a 0 realizar una operación de lectura no producirá un error en tiempo de ejecución, no obstante se activa el flag al valor de -1. Para limpiar el flag en C++ se utiliza la función f.clear(), en donde, f es la variable stream.

La función f.eof() retornará true si el puntero al archivo se encuentra fuera de los límites del archivo, esto es, si está más allá de la marca eof o si se movió el puntero al archivo a una dirección menor a 0, caso contrario, retorna false.

Además existe la función f.good() que retorna true si se pudieron leer datos del archivo, pero, si no se pudieron leer datos, entonces retorna false.

Procesos clásicos con archivos

Corte de Control

El corte de control es un proceso en el cual los registros del archivo se encuentran ordenados por el valor de uno o más campos, denominados campos **clave** o **llave**, el ordenamiento puede ser ascendente –lo más común- o descendente. Este tipo de proceso generalmente es utilizado para realizar informes o reportes, en el cual se deba emitir de cada grupo –formado por el mismo valor del campo clave- totales, promedios, máximos o mínimos, etc..

En líneas generales este proceso posee una estructura que es bastante característica y que pasaremos a detallar a continuación.

Cada corte de control establece un **nivel**, así si existen ***n*** cortes de control, habrá n niveles, y cada uno de estos n cortes o niveles están contenidos dentro de un ciclo indefinido, a estos ciclos se le suma uno más, y representa el fin del proceso, siendo este ciclo el de mayor nivel y el más externo, luego en forma anidada se van desarrollando los ciclos internos en un orden que va de mayor nivel de corte hasta llegar al menor nivel de corte. El ciclo de mayor nivel establece el fin del proceso, siendo su condición,

el fin del archivo ***f.eof()*** que se está procesando. Las condiciones de los ciclos internos van **arrastrando** las condiciones de los ciclos anteriores o más externos a la que se le suma la propia condición del ciclo, es decir, la que producirá el corte de control de ese nivel, por lo tanto, cuanto más anidado sea el ciclo, más condiciones contendrá. El ciclo externo contiene una condición, y el ciclo más interno posee n + 1 condiciones y en cada uno estos ciclos las condiciones establecidad estarán relacionadas con el operador lógico "y", es decir, el operador && del lenguaje C/C++ o representado por el carácter acento circunflejo "^". Si un proceso requiere n cortes, entonces la cantidad de ciclos será de n + 1. Además **cada corte de control obliga a que los datos se encuentren ordenados por el valor del campo clave comenzando desde el de mayor nivel hasta el de menor nivel**, por ejemplo, si se requieren obtener totales de las ventas realizadas por diferentes vendedores, los datos deberían están organizados por el código de vendedor; un segundo ejemplo, si los vendedores realizan sus actuaciones en una zona y el proceso requiere informar además totales por zonas, en este caso los datos deben encontrarse ordenados primero por código de zona y luego por código de vendedor, debido a que es de esperar que primero cambien los vendedores y luego cambien las zonas. **El corte de control requiere de una lectura anticipada**. Las próximas lecturas del archivo se realizarán en el ciclo más interno y como última acción. Antes de ingresar a un ciclo denominamos a esa región del algoritmo **cabecera**. Al salir de un ciclo, denominamos a esa región del algoritmo **pié**. Dentro de un ciclo, denominamos a esa región del algoritmo **proceso**. En la cabecera generalmente realizamos las siguientes acciones: Inicializar, Emitir títulos y datos. En el pié generalmente realizamos las siguientes acciones: Cálculos, Emitir totales, promedios, máximos o mínimos, tomar alguna decisión. En el proceso generalmente realizamos las siguientes acciones: Cálculos, Emitir líneas de detalle, tomar alguna decisión. A continuación se presenta un modelo de algoritmo de Corte de Control en un formato general.

Preparar1 = Cabecera del Listado

- Abrir archivos
- Emitir títulos
- Inicializar variables
- **Leer(f, r)**

Leer 1er.item

~fda(f)

Preparar2 = Cabecera del Corte de Control

- Emitir títulos cabecera CC
- Inicializar variables
- Copiar r.cmpClv a variable ClvAnterior

~fda(f) ^ (r.cmpClv = ClvAnterior)

Proceso

- Cálculos
- Emitir líneas de detalle
- Selecciones
- **Leer(f, r)**

Leer próximos items

Terminar2 = Pié del Corte de Control

- Cálculos
- Emitir títulos y datos
- Seleccionar

Terminar1 = Pié del Listado

- Cálculos
- Emitir títulos y datos
- Cerrar archivos

Ejercicio de Corte de Control de Universidades y Facultados

```
/*
        Id.Programa: G2Ej09.cpp Corte de Control n = 2.
        Autor......: Lic. Hugo Cuello
        Fecha......: Mayo-2003
        Comentario.: Examen a alumnos de distintas Universidades y Facultades.
                                                        Tecnica de Corte de Control.
*/

#include<iomanip>
#include<iostream>
using namespace std;

typedef unsigned int  word;
typedef unsigned short byte;
typedef char str5[6];
typedef char str15[16];
typedef char str20[21];
struct sExa {
        str5  CodUni,
                        CodFacu;
        long  NroLeg;
        str20 ApeNom;
        byte  Nota;
};

// Prototipos -------------------------------------------------------
void Abrir(FILE **Examen);
void Inic(word &totInsGral, word &totAprGral);
void EmiteCabLst();
void IniCab(word &totIns, word &totApr, str5 Clave, str5 &ClaveAnt);
void EmiteCabCte(str15 titulo,str5 codigo);
void ProcAlum(sExa rExa, word &tInsFacu, word &tAprFacu);
void CalcPie(word totIns, word totApr, word &totInsM, word &totAprM);
void EmitePie(str15 titulo, word totInsFacu, word totAprFacu);
// Fin Prototipos ---------------------------------------------------

void main() {
        FILE *Examen = NULL;
        sExa rExamen;
        word totInsGral,
                totAprGral,
                totInsUni,
                totAprUni,
                totInsFacu,
                totAprFacu;
```

```
        str5    UniAnt,
                FacuAnt;

        Abrir(&Examen);
        freopen("Examen.Lst","w",stdout);
        Inic(totInsGral,totAprGral);
        EmiteCabLst();
        fread(&rExamen,sizeof(rExamen),1,Examen);
        while (!feof(Examen)) {
           IniCab(totInsUni,totAprUni,rExamen.CodUni,UniAnt);
           EmiteCabCte("*Cod.Univ.: ",rExamen.CodUni);
           while (!feof(Examen) && strcmp(rExamen.CodUni,UniAnt) == 0) {
              IniCab(totInsFacu,totAprFacu,rExamen.CodFacu,FacuAnt);
              EmiteCabCte("**Cod.Fac.: ",rExamen.CodFacu);
              while (!feof(Examen) && strcmp(rExamen.CodUni,UniAnt) == 0 &&
                        strcmp(rExamen.CodFacu,FacuAnt) == 0) {
                 ProcAlum(rExamen,totInsFacu,totAprFacu);
                 fread(&rExamen,sizeof(rExamen),1,Examen);
              }
              CalcPie(totInsFacu,totAprFacu,totInsUni,totAprUni);
              EmitePie("Facu.: ",totInsFacu,totAprFacu);
           }
           CalcPie(totInsUni,totAprUni,totInsGral,totAprGral);
           EmitePie("Univ.: ",totInsUni,totAprUni);
        }
        EmitePie("General: ",totInsGral,totAprGral);
        freopen("CON","w",stdout);
        fclose(Examen);
} //main

void Abrir(FILE **Exa) {

        *Exa = fopen("Examen.Dat","rb");
} //Abrir

void Inic(word &tInsG, word &tAprG) {

        tInsG = tAprG = 0;
} // Inic

void EmiteCabLst() {

        cout << "Listado examen a alumnos" << endl;
} //EmiteCab

void IniCab(word &totIns, word &totApr, str5 Clave, str5 &ClaveAnt) {

        totIns = totApr = 0;
        strcpy(ClaveAnt,Clave);
} //IniCab
```

```
void EmiteCabCte(str15 titulo, str5 codigo) {

        cout << endl << titulo << " " << codigo << endl;
        if (strstr(titulo,"Fac") != NULL)
                cout << endl << setw(5) << " " << "Nro.Leg.  Nota" << endl;
} //EmiteCabCte

void ProcAlum(sExa rExa, word &tInsFacu, word &tAprFacu) {

        ++tInsFacu;
        if (rExa.Nota >= 4) {
           ++tAprFacu;
           cout << setw(6) << "  " << setw(6) << rExa.NroLeg << "    " << setw(2) <<
           rExa.Nota << endl;
        }
} //ProcAlum

void CalcPie(word totIns, word totApr, word &totInsM, word &totAprM) {

        totInsM += totIns;
        totAprM += totApr;
} //CalcPie

string replicate(char car, unsigned n) {
  string cad = "";

  for(unsigned i = 1; i <= n; i++)
      cad += car;
  return cad;
} // replicate

void EmitePie(str15 titulo, word totIns, word totApr) {
  string ast;

        if (titulo[0] == 'F')
           ast = replicate('*',2);
        else
           if (titulo[0] == 'U')
              ast = replicate('*',1);
        cout << " " << ast << "Tot.Insc. " << titulo << " " << totIns << endl;
        cout << " " << ast << "Tot.Apr.. " << titulo << " " << totApr << endl;
} //EmitePie
```

Apareo de Archivos

El apareo de archivos es un proceso que dependiendo del tipo de organización que tengan estos archivos, el algoritmo adoptará una estructura particular. Este proceso es muy empleado para la **actualización del maestro** a través del archivo de novedades. Las novedades tendrán que ver con **altas, bajas o modificaciones** o algunas de ellas

solamente. Si ambos archivos poseen una organización secuencial, entonces ambos archivos deben encontrarse ordenados –ascendente o descendente- por medio del valor de una clave en común. El resultado de este proceso será un nuevo archivo de salida con la misma estructura que el maestro a actualizar, siendo este archivo el maestro actualizado. El archivo de novedades tiene la misma estructura que el maestro pero con un campo más, el cual indica el *código de movimiento*. Cada archivo trabajará con sus propios registros, es decir, un registro para el archivo maestro viejo, otro para el archivo de maestro nuevo y un registro para el archivo de novedades. Las situaciones de errores por alta existente o bajas o modificaciones inexistentes, se emitirán por medio del dispositivo de la impresora.

Este proceso que genera un nuevo archivo –el maestro actualizado- se denomina **proceso Padre – Hijo**.

El siguiente diagrama de sistema muestra el proceso Padre – Hijo

A continuación se presentará un modelo de apareo de archivos, con un archivo maestro desactualizado y un archivo de novedades, ambos con organización secuencial, con valores de campos claves sin repetición en cada uno de los archivos y ordenados ambos por el valor de una misma clave.

Al igual que con el Corte de Control, el proceso de **apareo requiere una lectura anticipada y debe ser una lectura especial** en el Lenguaje Pascal, como fue indicado en el tema de Corte de Control.

Un primer proceso se realizará mientras no haya finalizado ningún archivo y en un segundo proceso, se procesará el archivo que no haya finalizado, hasta agotarlo. Por lo tanto, habrá tres ciclos en secuencia.

Dentro del primer ciclo o proceso se compararán los valores de las claves de ambos archivos, por ejemplo, ¿la clave del maestro es igual a la clave de novedades? o ¿la clave del maestro es mayor a la clave de novedades? y por descarte será que ¿la clave de maestro es menor a la clave de novedades?.

En el primer caso si la **clave del registro maestro es igual a la clave del registro de novedades** se deberá comparar el código de movimiento, pudiendo ser igual a 'A' por alta, o a 'B' por baja o a 'M' por modificación.

Si el código de **movimiento es igual a 'A'** será un **error, por alta existente**, es decir, el valor de esa clave se encuentra ya en el archivo maestro y el **registro de maestro desactualizado deberá grabarse en el nuevo maestro** –para no perderlo-.

Si el código de **movimiento es igual a 'B'** no es error y no debe grabarse en el nuevo maestro, -de esta manera lo estamos dando de baja al no aparecer en el nuevo maestro-, por lo tanto, **no debemos realizar ninguna acción** por este caso.

Si el código de **movimiento es igual a 'M'** no es error y debemos **mover los campos a modificar indicados por el archivo de novedades al registro del maestro y luego grabarlo en el nuevo maestro**.

Por último debemos realizar la lectura especial por cada uno de los archivos, es decir, tanto del maestro desactualizado como de novedades.

A continuación se detalla el proceso a realizar si la **clave del registro maestro viejo es mayor a la clave del registro de novedades**.

En este caso resulta que existe un registro de novedades sin su registro correspondiente en el maestro. Por lo tanto se debe determinar el código de movimiento.

Si el código de **movimiento es igual a 'A'** es correcto ya que la clave no existe en el maestro viejo y debe realizarse el alta, para ello se debe a**signar los campos informados por la novedad al registro del maestro nuevo y grabar un nuevo registro en maestro nuevo**. No asignarlo al registro del maestro viejo, ¿por qué?.

Si el código de **movimiento es igual a 'B'** es **error, por baja inexistente**.

Si el código de **movimiento es igual a 'M'** es **error, por modificación inexistente**.

Por último debemos realizar la lectura especial solo de novedades, -se lee del archivo cuya clave resultó menor, de esta manera aseguramos alcanzar el valor de la otra clave-.

A continuación por decantación se detalla el proceso a realizar si la clave del registro maestro viejo es menor a la clave del registro de novedades.

En este caso resulta que existe un registro de maestro sin su registro correspondiente en el de novedades. Por lo tanto, aquí no se deberá analizar el código de movimiento, ya que no existe. El **registro de maestro viejo se debe grabar en el archivo maestro nuevo** y se debe realizar la lectura especial solo de maestro viejo, por haber resultado su clave menor a la de novedades-.

Este primer proceso finaliza cuando uno de los archivos finalice, podría darse el caso que finalicen ambos, si los valores de las claves del último registro de cada archivo son iguales.

Debido a que no se garantiza que finalicen ambos archivos en el proceso anterior, habrá que procesar los registros restantes del archivo que no finalizó.

Si es el caso del maestro viejo que no haya finalizado el proceso finalizará cuando se hayan procesados todos los registros restantes de éste archivo. El proceso consistirá en grabar todos los registros pendientes. Debido a que traemos un registro que hemos leído pero no hemos procesado, dentro del ciclo primero grabamos el registro y luego leemos con una lectura especial el próximo registro.

Por otro lado, si es el caso del archivo de novedades el que no haya finalizado el proceso finalizará cuando se hayan procesados todos los registros restantes de éste otro archivo. El proceso consistirá en realizar las mismas acciones vistas en el caso cuando la clave de maestro viejo es mayor a la clave de novedades, solo serán correctos los registros de novedades cuyos códigos de movimientos indiquen 'A'.A continuación se presenta un modelo de algoritmo de Apareo de archivos en un formato general.

C
Abrir(fMv,fN,fMn)
LeerM(fMv, rMv)
LeerN(fN, rN)
~ fdaf(fMv) ^ ~ fdaf(fN)
rMv.cmpClv ∴ rN.cmpClv
=
rN.CodMov
'A'
'B'
'M'
Alta existente
Grabar(fMn, rMv)
Mover campos de reg.de Mv a reg. de Mn, luego mover campos de reg. de Nov. a reg. de Mn.
Grabar(fMn, rMn)
LeerM(fMv, rMv)
LeerN(fN, rN)
>
rN.CodMov
'A'
'B'
'M'
Mover campos de reg.de Nov. a reg. de Mn
Grabar(fMn, rMn)
Baja inexistente
Modif. inexistente
LeerN(fN, rN)
α
<
LeerM
(fMv, rMv)
Grabar
(fMn, rMv)
β
A

Cerrar(fMv, fN, fMn)
F

Técnica de HIGH_VALUE o LOW_VALUE

Este modelo de apareo de archivos puede ser optimizado si se realizan algunos ligeros cambios. En principio vemos que las acciones **alfa** y **beta** se vuelven a reiterar cuando uno de los archivos haya finalizado.

La pregunta es ¿no podrían estas acciones realizarse solamente y en forma completa dentro del primer ciclo?. La respuesta es sí, pero realizando las siguientes modificaciones.

En principio se abandonará el ciclo si **ambos** archivos finalizaron. El secreto radica en asignar el valor más alto al campo clave del archivo que finalizó, si los archivos están ordenados en forma ascendente, caso contrario se asignará el valor más bajo. Esta técnica se conoce como **HIGH VALUE** o **LOW VALUE** en el segundo caso.

La lectura especial debe tener 2 parámetros, el archivo a leer y el registro a devolver.

En caso que haya finalizado el archivo se deberá mover al campo clave el valor más alto, -o el más bajo-, es decir, un valor al cual ningún dato leído podrá alcanzar. Por ejemplo, si el campo clave indicara un código de vendedor de 3 dígitos, el valor más alto sería 1000, ya que ningún dato que se lea alcanzaría a ese valor especial; un segundo ejemplo, podría ser tratándose de fechas con mes y día en un mismo campo clave, el valor más alto sería 1232 **¿por qué?.**

Las condiciones del ciclo, deberán compararse, los valores de los campos claves con el valor más alto, y para abandonar el ciclo las condiciones podrían ser las siguientes:

(rMv.cmpClv ≠ HIGH_VALUE) v (rN.cmpClv ≠ HIGH_VALUE)

HIGH_VALUE es una constante con nombre definida en la sección const, p.ej.:

```
const
  HIGH_VALUE = 1000;
```

De esta manera los ciclos posteriores dejarían de existir, ya que no se lo requieren. Con estos ligeros cambios el algoritmo solo quedaría con un solo ciclo.

La lectura especial podría presentar el siguiente aspecto:

Ejemplo de Apareo de archivos

```
/*
   Id.Programa: G2Ej10.cpp Apareo
   Autor......: Lic. Hugo Cuello
   Fecha......: ago-2014
   Comentario.: Apareo entre MaeVjo y Nov.
                Genera MaeNvo y LstErr.
*/

#include<iostream>
using namespace std;

typedef char str20[21];

struct sMae {
  long  cmpClv;
  int   cmp1,
        cmp2;
  str20 RazSoc,
        Domic,
        Local;
  long  NroCUIT;
};

struct sNov {
  sMae rMaeN;
  char codMov;
};

// Prototipos ----------------------------------------
```

```
void Abrir(FILE **, FILE **, FILE **);
void Apareo(FILE *, FILE *, FILE *);
void CerrarEliminarRenombrar(FILE *, FILE *, FILE *);
// Fin Prototipos -----------------------------------

void Abrir(FILE **MaeV, FILE **MaeN, FILE **Nov) {

  *MaeV = fopen("MaeVjo.Dat","rb");
  *MaeN = fopen("MaeNvo.Dat","wb");
  *Nov  = fopen("Noved.Dat","rb");
} //Abrir

void Apareo(FILE *MaeV, FILE *MaeN, FILE *Nov) {
  sMae rMaeV,
         rMaeN;
  sNov rNov;

  freopen("ErroresABM.Lst","w",stdout);
  fread(&rMaeV,sizeof(rMaeV),1,MaeV);
  fread(&rNov,sizeof(rNov),1,Nov);
  while (!feof(MaeV) && !feof(Nov))
    if (rMaeV.cmpClv == rNov.rMaeN.cmpClv) {
      switch (rNov.codMov) {
        case 'A': cout << "Error por Alta Existente, clave: " <<
                        rNov.rMaeN.cmpClv << "Ubicacion: " <<
                        (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
                  rMaeN = rMaeV;
                  fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                  break;
        case 'M': rMaeN = rMaeV;
                  rMaeN.cmp1 = rNov.rMaeN.cmp1;
                  strcpy(rMaeN.Domic,rNov.rMaeN.Domic);
                  fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                  break;
      }
      fread(&rMaeV,sizeof(rMaeV),1,MaeV);
      fread(&rNov,sizeof(rNov),1,Nov);
    }
    else
      if (rMaeV.cmpClv > rNov.rMaeN.cmpClv) {
        switch (rNov.codMov) {
          case 'A': rMaeN = rNov.rMaeN;
                    fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                    break;
          case 'B': cout << "Error por Baja inexistente, clave: " <<
                          rNov.rMaeN.cmpClv << "Ubicacion: " <<
                          (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
                    break;
          default: cout << "Error por Modificacion inexistente, clave: " <<
```

```
                    rNov.rMaeN.cmpClv << "Ubicacion: " <<
                    (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
      }
      fread(&rNov,sizeof(rNov),1,Nov);
    }
    else {
      rMaeN = rMaeV;
      fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
      fread(&rMaeV,sizeof(rMaeV),1,MaeV);
    }
  while (!feof(Nov)) {
    switch (rNov.codMov) {
      case 'A': rMaeN = rNov.rMaeN;
            fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
            break;
      case 'B': cout << "Error por Baja inexistente, clave: " <<
                    rNov.rMaeN.cmpClv << "Ubicacion: " <<
                    (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
            break;
      default: cout << "Error por Modificacion inexistente, clave: " <<
                    rNov.rMaeN.cmpClv << "Ubicacion: " <<
                    (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
    }
    fread(&rNov,sizeof(rNov),1,Nov);
  }
  while (!feof(MaeV)) {
    rMaeN = rMaeV;
    fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
    fread(&rMaeV,sizeof(rMaeV),1,MaeV);
  }
  freopen("CON","w",stdout);
}  //Apareo

void CerrarEliminarRenombrar(FILE *MaeV, FILE *MaeN, FILE *Nov) {

  fclose(MaeV);
  fclose(MaeN);
  fclose(Nov);
  remove("MaeVjo.Dat");
  rename("MaeNvo.Dat","MaeVjo.Dat");
} //CerrarEliminarRenombrar

int main() {
  FILE *MaeVjo,
       *MaeNvo,
       *Noved;

  Abrir(&MaeVjo,&MaeNvo,&Noved);
  Apareo(MaeVjo,MaeNvo,Noved);
  CerrarEliminarRenombrar(MaeVjo,MaeNvo,Noved);
```

```
  return 0;
}
```

Apareo de archivos con técnica de High Value

```
/*
   Id.Programa: G2Ej10HV.cpp Apareo
   Autor......: Lic. Hugo Cuello
   Fecha......: ago-2014
   Comentario.: Apareo entre MaeVjo y Nov.
                Genera MaeNvo y LstErr.
                Tecnica con HIGH_VALUE.
*/

#include<iostream>
using namespace std;

#define HIGH_VALUE 10000
typedef char str20[21];

struct sMae {
  long  cmpClv;
  int   cmp1,
        cmp2;
  str20 RazSoc,
        Domic,
        Local;
  long  NroCUIT;
};

struct sNov {
  sMae rMaeN;
  char codMov;
};

// Prototipos ------------------------------------------
void Abrir(FILE **, FILE **, FILE **);
void ApareoHV(FILE *, FILE *, FILE *);
void LecHV(FILE *, sMae &);
void LecHV(FILE *, sNov &);
void CerrarEliminarRenombrar(FILE *, FILE *, FILE *);
// Fin Prototipos --------------------------------------

void Abrir(FILE **MaeV, FILE **MaeN, FILE **Nov) {

  *MaeV = fopen("MaeVjo.Dat","rb");
  *MaeN = fopen("MaeNvo.Dat","wb");
```

```
  *Nov  = fopen("Noved.Dat","rb");
} //Abrir

void LecHV(FILE *MaeV, sMae &rMaeV) {

  fread(&rMaeV,sizeof(rMaeV),1,MaeV);
   if (feof(MaeV))
    rMaeV.cmpClv = HIGH_VALUE;
} //LecHV

void LecHV(FILE *Nov, sNov &rNov) {

  fread(&rNov,sizeof(rNov),1,Nov);
  if (feof(Nov))
    rNov.rMaeN.cmpClv = HIGH_VALUE;
} //LecHV

void ApareoHV(FILE *MaeV, FILE *MaeN, FILE *Nov) {
  sMae rMaeV,
       rMaeN;
  sNov rNov;

  freopen("ErroresABMHD.Lst","w",stdout);
  LecHV(MaeV,rMaeV);
  LecHV(Nov,rNov);
  while (rMaeV.cmpClv != HIGH_VALUE ||
           rNov.rMaeN.cmpClv != HIGH_VALUE)
    if (rMaeV.cmpClv == rNov.rMaeN.cmpClv) {
      switch (rNov.codMov) {
        case 'A': cout << "Error por Alta Existente, clave: " <<
                          rNov.rMaeN.cmpClv << "Ubicacion: " <<
                          (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
                  rMaeN = rMaeV;
                  fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                  break;
        case 'M': rMaeN = rMaeV;
                  rMaeN.cmp1 = rNov.rMaeN.cmp1;
                  strcpy(rMaeN.Domic,rNov.rMaeN.Domic);
                  fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                  break;
      }
      LecHV(MaeV,rMaeV);
      LecHV(Nov,rNov);
    }
    else
      if (rMaeV.cmpClv > rNov.rMaeN.cmpClv) {
        switch (rNov.codMov) {
          case 'A': rMaeN = rNov.rMaeN;
                    fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
                    break;
```

```
        case 'B': cout << "Error por Baja inexistente, clave: " <<
                    rNov.rMaeN.cmpClv << "Ubicacion: " <<
                    (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
                break;
        default: cout << "Error por Modificacion inexistente, clave: " <<
                    rNov.rMaeN.cmpClv << "Ubicacion: " <<
                    (ftell(Nov) - sizeof(rNov)) / sizeof(rNov) << endl;
      }
      LecHV(Nov,rNov);
    }
    else {
      rMaeN = rMaeV;
      fwrite(&rMaeN,sizeof(rMaeN),1,MaeN);
      LecHV(MaeV,rMaeV);
    }
  freopen("CON","w",stdout);
}  //ApareoHV

void CerrarEliminarRenombrar(FILE *MaeV, FILE *MaeN, FILE *Nov) {

  fclose(MaeV);
  fclose(MaeN);
  fclose(Nov);
  remove("MaeVjo.Dat");
  rename("MaeNvo.Dat","MaeVjo.Dat");
} //CerrarEliminarRenombrar

int main() {
  FILE *MaeVjo,
        *MaeNvo,
        *Noved;

  Abrir(&MaeVjo,&MaeNvo,&Noved);
  ApareoHV(MaeVjo,MaeNvo,Noved);
  CerrarEliminarRenombrar(MaeVjo,MaeNvo,Noved);

  return 0;
}
```

Búsqueda Binaria o Dicotómica

La búsqueda binaria es un proceso en el cual los datos deben encontrarse ordenados por el valor de la clave a buscar, generalmente en orden ascendente, pero puede ocurrir que se encuentren ordenados en forma descendente. Nuestro modelo de estudio se considera ordenado en forma ascendente. El proceso consiste en tomar el valor que se encuentre en una posición media entre las posiciones de los extremos, así tratándose de elementos que se encuentran en un archivo, esos valores extremos serían **cero** y **f.seekg(-sizeof r,ios::end)**. El punto medio se obtiene de aplicar la siguiente expresión:

med ← *(pri + ult) div 2*;

en donde: *pri* es el extremo inferior y *ult* es el extremo superior. Una vez obtenido el punto medio se accede a esa posición en el archivo y se lee el registro, luego se compara con el valor de la clave a buscar, pudiendo resultar alguna de las siguientes posibilidades:

1. Que el valor del registro leído sea igual al valor clave a buscar, en ese caso, el proceso de búsqueda finalizará saliendo del ciclo cambiando el estado a una variable boolean e informaremos el dato a retornar, por ejemplo, en qué posición se encontró, o un valor boolean, o el valor de un campo o todo el registro, según lo que convenga al proceso.
2. Que el valor del registro leído sea menor al valor clave a buscar, en ese caso deberá cambiarse el valor del extremo inferior, de la siguiente manera: *pri* ← *med + 1*.
3. Caso contrario, es decir, el valor del registro leído es mayor al valor clave a buscar, en ese caso deberá cambiarse el valor del extremo superior, de la siguiente manera: *ult* ← *med – 1*.

Este método va eliminando, por cada vez que obtenemos un punto medio, una mitad del conjunto de datos que nos va quedando, y habiendo leído el registro de esta posición, su valor no es el de la clave a buscar. Por lo tanto, aquí tenemos del porque este método recibe el nombre de búsqueda binaria o dicotómica.

Si la clave no se encuentra, el proceso finalizará cuando el valor del extremo inferior sea mayor al valor del extremo superior, es decir, cuando *pri* es mayor a *ult*.

La búsqueda binaria garantiza realizar pocos accesos al disco, si ***n*** es el tamaño del archivo, la cantidad de accesos recién se duplica con n^2. Por ejemplo, si n = 50000 la cantidad de accesos máximos será de 16. La expresión que determina esto está en función de $\log_2(n)$. Recordemos entonces que **para poder aplicar este método, el conjunto de datos, deberá encontrarse ordenado por el valor de la clave a buscar**.

A continuación se describe el algoritmo para un caso general de búsqueda binaria. El mismo será una función que recibe como parámetros el archivo y la clave y retorna la posición, si se encontró un valor ≥ 0, caso contrario –1.

De acuerdo a las necesidades de un proceso, el módulo de búsqueda binaria podrá ser una función si lo que se retorne es un valor simple como una posición del archivo o un estado de verdadero o falso por si se encontró o no la clave, una cadena, un real o cualquier otro valor simple o incluso un puntero a un tipo de dato simple o estructurado. Pero además podrá devolver un valor por medio de sus parámetros pasados por referencia.

Búsqueda Lineal o Secuencial

La búsqueda secuencial o lineal en archivos **no es aconsejable**, debido a que la cantidad de accesos a realizar para buscar el valor de una clave, en el peor de los casos será de n accesos, en el mejor de los casos será de 1 acceso y como promedio será de n/2 accesos, para un archivo con n registros.

Para aplicar la búsqueda binaria hemos visto en el tema anterior que el archivo debe encontrarse ordenado por el valor de un campo clave. El costo que tendremos en un archivo desordenado sería entonces tener que ordenarlo.

Podrían existir **casos excepcionales** en el cual podamos aplicar la búsqueda secuencial a un archivo desordenado, por lo tanto daremos el algoritmo correspondiente, para esta situación.

Buscar el valor de una clave en un archivo desordenado, primero debemos asegurar que el puntero al archivo esté ubicado al comienzo del mismo. Inicializamos el

estado de una variable de tipo boolean en falso, y al nombre de la función le asignamos –1 para indicar que aún el valor de la clave no se ha encontrado. El proceso se realizará mientras no se haya encontrado un registro que contenga el valor de la clave a buscar y mientras no sea fin de archivo. Dentro del proceso leemos un registro, luego lo comparamos con el valor de la clave, si es igual lo hemos encontrado, modificamos el estado de una variable de tipo boolean a verdadero y asignamos al nombre de la función la posición del archivo – 1. A continuación presentamos el algoritmo de búsqueda lineal.

Por último si un archivo se encuentra ordenado en concordancia con el valor de la clave a buscar, como se vio en párrafos anteriores, la búsqueda binaria es la técnica a emplear, pero, se podría emplear la búsqueda secuencial, el único impedimento es que serán necesarios más accesos al disco, ahora bien, si descartamos este hecho real, podríamos decir que el método de búsqueda secuencial visto anteriormente podría ser aplicado, pero sabiendo que el archivo ya se encuentra ordenado, existe una variante más óptima para estos casos. El proceso finalizaría ahora no solamente si se encuentra el valor de la clave a buscar, sino que también **nos detendríamos si el valor leído en una instancia se hiciera mayor al valor de la clave**, ya que no tendría razón de seguir avanzando en la búsqueda, debido a que todos los valores posteriores también serían mayores. La única razón de mostrar este método consistiría en que posteriormente se verán estructuras de datos en la cual el único acceso posible sería el secuencial y con esta estructura ordenada la búsqueda finalizaría al encontrarse el valor a buscar o bien cuando hayamos alcanzado un valor mayor al buscado. El algoritmo correspondiente de una búsqueda secuencial optimizada en archivos se detalla a continuación, -en realidad jamás utilizaremos este método- ya que la búsqueda binaria sería el método a emplear. Este método se muestra al solo efecto de que más adelante se verán estructuras de datos dinámicas lineales y ordenadas por el valor de una clave, y en estos casos buscar un elemento en estas listas será únicamente secuencial, no existiendo la posibilidad de realizar búsqueda binaria, por lo tanto, será una búsqueda secuencial optimizada, en el sentido que cuando nos topemos con un valor que se hizo mayor al buscado, en ese punto detendremos la búsqueda.

<u>Recordar</u>: Para archivos ordenados el método de búsqueda será el **dicotómico o binario**, en cambio si existe una relación 1:1, es decir el valor de la clave = dirección –*f. biyectiva*- o transformando el valor de la clave en una dirección válida en el archivo sin que se produzcan colisiones, el **direccionamiento directo** es el que debe emplearse. Si el archivo estuviera desordenado la creación de **archivos auxiliares** para lograr accesos más rápidos a los datos del archivo es otra de las posibilidades como técnicas a emplear.

El siguiente ejemplo ilustra una situación en la cual el archivo se encuentra desordenado. Suponiendo que este archivo fuera de vendedores conteniendo como campos el código del vendedor (3 dígitos), más otros campos de interés, y sabiendo que vamos a recurrir en forma reiterada a este archivo para buscar el valor de una clave por código de vendedor, los accesos secuenciales serían reiterados y por consecuencia la velocidad de ejecución de este proceso caería por debajo de lo óptimo o deseado. Aplicar el método de búsqueda binario no podríamos realizarlo debido a que el archivo no está ordenado y si bien existe la posibilidad de ordenarlo físicamente, aplicar ese método está fuera de las posibilidades conocidas hasta ahora, *es lo que se conoce como* ***método por copia*** *y se requieren un archivo del mismo tamaño que el original, -ahí va a quedar el archivo definitivo y ordenado-, más un espacio adicional para un archivo temporal de trabajo.* Pero como dijimos anteriormente este método no vamos a emplear. Otro método podría ser indexar el archivo a través del uso de un archivo de índices, pero acá también con los conocimientos logrados hasta ahora no contamos con la información necesaria para lograr este cometido. Otra posibilidad sería en crear un archivo en el cual pueda contener todas las claves posibles. La técnica a emplear en este último caso será de la creación de un archivo cascarón que contenga todas y solo todas

las posibles claves. Para el ejemplo previamente indicado este archivo constará de 1000 registros, cuyas direcciones irán de 0 a 999.

Una vez creado el archivo cascarón que reserva los espacios suficientes, se procederá a leer secuencialmente el archivo de datos, por cada registro leído se accede con direccionamiento directo a la posición en el archivo auxiliar indicada por el valor de la clave, y grabamos la referencia o dirección del valor de esa clave en el archivo de datos, y así sucesivamente hasta finalizar el recorrido en el archivo de datos.

Por ejemplo si leemos del archivo vendedores el valor clave 349, con este valor nos ubicamos en el archivo auxiliar y grabamos 0, debido a que el primer registro leído se encuentra en esa posición, el segundo registro leído el valor clave es 231, con este valor nos ubicamos en el archivo auxiliar y grabamos 1, debido a que el segundo registro leído se encuentra en esa posición, y así sucesivamente hasta finalizar el archivo cuya posición a grabar será la posición de vendedores menos uno.

A continuación se da un ejemplo de esta situación:

Notar en el auxiliar que la *dirección-clave* 349, dice que se encuentra en la referencia 0 en vendedores, que la *dirección-clave* 231, dice que se encuentra en la dirección 1 en vendedores y así sucesivamente. Luego en el proceso, al requerir un dato de un vendedor, accedemos con la clave conocida en el auxiliar, a la dirección que coincide con su clave y con el valor que obtenemos de leer ese registro accedemos directamente a vendedores. Por ejemplo si conocemos la clave del vendedor 185 accedemos a esa posición en el auxiliar, leemos el registro y con el valor 10 accedemos a esa posición en vendedores. ¡¡¡Esto es mucho más óptimo que acceder secuencialmente a un archivo desordenado!!! incluso más rápido que acceder en forma dicotómica o binaria. El costo de aplicar este método es que debemos de contar con un espacio adicional en disco para poder crear el archivo cascarón.

Ejemplos de la G2Ej4 Crear archivo Articulos

```
/*
  Id.Programa: G2Ej04.cpp
  Autor..........: Lic. Hugo Cuello
  Fecha..........: oct-2013
  Comentario.: Creacion archivo de Articulos.
*/

#include <conio.h>
#include <fstream>
using namespace std;
#define CENTINELA 0

typedef unsigned short ushort;
typedef char str20[21];
struct sArt {
        ushort CodArt;
```

```
        str20  Descrip;
        float  PreUni;
};

// Prototipos -----------
void Abrir(FILE **);
void ObtDatos(sArt &);
// Fin Prototipos -------

main() {
        FILE *Articulos;
        sArt rArticulo;

        Abrir(&Articulos);
        ObtDatos(rArticulo);
        while (rArticulo.CodArt) {
                fwrite(&rArticulo,sizeof(rArticulo),1,Articulos);
                ObtDatos(rArticulo);
        }
        fclose(Articulos);
        return 0;
} // main

void Abrir(FILE **Art) {
        *Art = fopen("Articulos.Dat","wb");
} // Abrir

void ObtDatos(sArt &rArt) {
   do {
     gotoxy(5,1);
     cout << "Alta de Articulos.Dat";
     gotoxy(10,5); clreol();
     cout << "Cod.Articulo Fin = " << CENTINELA << ": ";
     cin >> rArt.CodArt;
   }
   while (!(rArt.CodArt <= 100));
   if (rArt.CodArt) {
      do {
          gotoxy(10,7); clreol();
          cout << "Descripcion: ";
          gets(rArt.Descrip);
      }
      while (!strcmp(rArt.Descrip,""));
      do {
          gotoxy(10,9); clreol();
          cout << "Pre.Unitario: ";
          cin >> rArt.PreUni;
      }
      while (!rArt.PreUni);
```

```
    }
} // ObtDatos
```

Ejemplo de la G2Ej5 Actualizar Precio en Articulos

```
/*
        Id.Programa: G2Ej05.cpp
        Autor...........: Lic. Hugo Cuello
        Fecha...........: oct-2013
        Comentario.: Actualizacion de Precio en archivo de Articulos.
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef char str20[21];
typedef unsigned short ushort;
struct sArt {
   ushort CodArt;
   str20   Descrip;
   float    PreUni;
};

// Prototipos ------------------
void Abrir(FILE **);
void ObtDato(float &);
void ActReg(FILE *, sArt );
void Cerrar(FILE *, FILE *);
// Fin Prototipos --------------

void Abrir(FILE **Art) {
        *Art = fopen("Articulo.Dat","r+b");
} // Abrir

void ObtDato(float &porc) {
        clrscr();
        gotoxy(10,5); clreol();
        cout << "Porcentaje: ";
        cin >> porc;
} // ObtDato

void ActReg(FILE *Art, sArt rArt) {
        fseek(Art,ftell(Art) - sizeof(rArt),SEEK_SET);
        fwrite(&rArt,sizeof(rArt),1,Art);
        fseek(Art,ftell(Art),SEEK_SET); //Necesario hacer para Actualizar posicion del
                                        //puntero
} // ActReg

void main() {
        FILE  *Articulos;
```

```
        sArt  rArticulo;
        float porcje;

        Abrir(&Articulos);
        ObtDato(porcje);
        while (fread(&rArticulo,sizeof(rArticulo),1,Articulos)) {
                rArticulo.PreUni *= (1 + porcje / 100);
                ActReg(Articulos,rArticulo);
        }
        fclose(Articulos);
}
```

Ejemplo de la G2Ej06 Mayores Precios en Articulos grabar en archivo Mayores

```
/*
        Id.Programa: G2Ej06.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: oct-2013
        Comentario.: Genera archivo de Mayores precios de Articulos.
*/
#include <conio.h>
#include <fstream>
using namespace std;
typedef char str20[21];
typedef unsigned short ushort;
struct sArt {
        ushort CodArt;
        str20  Descrip;
        float  PreUni;
};

struct sMay {
  unsigned CodArt;
  float    PreUni;
};

// Prototipos -----------------
void Abrir(FILE **, FILE **);
void ObtDato(float &);
void GenReg(FILE *, sArt );
void Cerrar(FILE *, FILE *);
// Fin Prototipos -------------

void Abrir(FILE **Art, FILE **May) {
        *Art = fopen("Articulo.Dat","rb");
        *May = fopen("Mayores.Dat","wb");
} // Abrir

void ObtDato(float &impMax) {
        clrscr();
```

```
        gotoxy(10,5); clreol();
        cout << "Importe Maximo: ";
        cin >> impMax;
} // ObtDato

void GenReg(FILE *May, sArt rArt) {
        sMay rMay;
        rMay.CodArt = rArt.CodArt;
        rMay.PreUni = rArt.PreUni;
        fwrite(&rMay,sizeof(rMay),1,May);
} // GenReg

void Cerrar(FILE *Art, FILE *May) {
  fclose(Art);
  fclose(May);
} // Cerrar

void main() {
        FILE  *Articulos,
             *Mayores;
        sArt  rArticulo;
        float maxImp;

        Abrir(&Articulos,&Mayores);
        ObtDato(maxImp);
        while (fread(&rArticulo,sizeof(rArticulo),1,Articulos)) {
                if (rArticulo.PreUni > maxImp)
                  GenReg(Mayores,rArticulo);
        }
        Cerrar(Articulos,Mayores);
}
```

Relaciones entre archivos

Búsqueda Binaria, Secuencial, Direccionamiento Directo y Archivo Auxiliar

Ejemplo de la G2Ej11 de Facturas a Clientes en Cta./Cte.:

Diagrama de Sistema

```
/*
   Id.Programa: G2Ej11Facturas.cpp
   Autor...........: Lic. Hugo Cuello
   Fecha...........: ago-2014
   Comentario.: Emision de Facturas a Clientes en Cta.Cte.
                Relaciones entre archivos.
*/

#include<iomanip>
#include<iostream>
using namespace std;

#define IVA 0.21
#define not !
#define and &&

typedef char str5[6];
typedef char str10[11];
typedef char str15[16];
typedef char str20[21];
typedef unsigned short byte;
typedef unsigned word;
```

```
struct sPed {
  long codCli; //Ord. por codCli (con repeticion) + codArt (sin repeticion)
  byte codArt;
  word cant;
};

struct sCli {    //Ord. por codCli
  long  codCli;  //8 digitos.
  str20 razSoc,
        domic,
        local;
  word  codPos;
  char  codPcia;
  str15 nroCUIT;
  str20 formaPago;
  word  codVen;
  float saldo; //Actualizar el saldo
};

struct sArt {   //Relacion 1:1 clv = dir.
  byte  codArt; //1..100
  str20 marca,
        descrip;
  float precio;
  word  stockAct, //Actualizar el stockAct.
        stockMin,
        ptoRep;
  str10 uniMed;
  char  estado;
};

struct sVen {       //desordenado
  word  codVen;     //1..999
  str20 nomVen;
  float porcCom,
        impAcumCom; //Actualizar el impAcumCom.
};

struct sNro {       //desordenado
  str5 tipoDoc;
  long nroDoc;      //Actuaizar nroDoc.
};

struct sFecF {
  byte dia,
       mes;
  word year;
};

struct sFac {  //Ord.cronologico. El archivo existe y se deben agregar +registros.
```

```
  long  nroFac;
  sFecF fecFac;
  long  codCli;
  float impo;
  char  estado; //'A' = Adeuda, 'P' = Pagada.
};

struct sFec {
  int year,
      mes,
      dia,
      dsem;
};

struct sHor {
  int Hora,
      Min,
      Seg;
};

// Prototipos ------------------------------------------------------------
void Abrir(FILE **, FILE **, FILE **, FILE **,FILE **, FILE **, FILE **);
void ArmarAuxVen(FILE *, FILE *);
long GetDate(int &, int &, int &, int &);
void EmiteTapaObtFecha(sFec &);
void BusLinNrosInic(FILE *, str5 , sNro &);
void InicCabCli(byte &, float &, long &nDoc);
bool BusBinCli(FILE *, long , sCli &);
void BusDDVen(FILE *, word , sVen &, FILE *);
void EmiteCabCli(sCli , str20 , sFec , sHor, long );
void BusDDArt(FILE *, byte , sArt &);
void CalcDetFac(word , sArt &, byte &, float &,float &);
void ActArt(FILE *, sArt );
void EmiteDetFac(byte , sArt , word , float );
void CalcPieCli(float ,float &, float &, float &, sVen &);
void ActCli(FILE *, sCli );
void ActVen(FILE *, sVen );
void AgregarRegFac(FILE *, long , sFec , long , float );
void EmitePieFac(byte , float , float , float );
void ActNroI(FILE *, sNro );
void CerrarEliminar();
// Fin Prototipos -------------------------------------------------------

void Abrir(FILE **Ped, FILE **Cli, FILE **Art, FILE **Ven,FILE **nInic, FILE
                    **Fac, FILE **AuxVen) {

  *Ped    = fopen("Pedidos.Dat","rb");
  *Cli    = fopen("Clientes.Dat","r+b");
  *Art    = fopen("Articulos.Dat","r+b");
```

```
  *Ven    = fopen("Vendedores.Dat","r+b");
  *nInic  = fopen("NrosInic.Dat","r+b");
  *Fac    = fopen("Facturas.Dat","ab");
  *AuxVen = fopen("AuxVen.Tmp","w+b");
} //Abrir

void ArmarAuxVen(FILE *Ven, FILE *AuxVen) {
  int  refVen = -1;
  sVen rVen;

  for (int i = 0; i <= 999; i++)
    fwrite(&refVen,sizeof(int),1,AuxVen);
  while (fread(&rVen,sizeof(rVen),1,Ven)) {
    fseek(AuxVen,rVen.codVen * sizeof(rVen),SEEK_SET);
    refVen = (ftell(Ven) - sizeof(rVen)) / sizeof(rVen);
    fwrite(&refVen,sizeof(refVen),1,AuxVen);
  }
} //ArmarAuxVen

long GetDate(int &year, int &mes, int &dia, int &ds) {
  time_t     rawtime;
  tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  year = 1900 + timeinfo->tm_year;
  mes  = 1 + timeinfo->tm_mon;
  dia  = timeinfo->tm_mday;
  ds   = 1 + timeinfo->tm_wday;
  return (1900 + timeinfo->tm_year) * 10000 + (1 + timeinfo->tm_mon) * 100 +
            timeinfo->tm_mday;
} // GetDate

long GetTime(int &hh, int &mm, int &ss) {
  time_t     rawtime;
  tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  hh = timeinfo->tm_hour;
  mm = timeinfo->tm_min;
  ss = timeinfo->tm_sec;
  return timeinfo->tm_hour * 10000 + timeinfo->tm_min * 100 + timeinfo->tm_sec;
} // GetTime

void EmiteTapaObtFecha(sFec &rFec) {

  cout << "FACTURACION A CLIENTES" << endl;
  cout << " EN CUENTA CORRIENTE" << endl;
  GetDate(rFec.year,rFec.mes,rFec.dia,rFec.dsem);
```

```
} //EmiteTapaObtFecha

void BusLinNrosInic(FILE *nInic, str5 tDoc, sNro &rNroI) {
  bool sigo = true;

  while (sigo) {
    fread(&rNroI,sizeof(rNroI),1,nInic);
    if (strcmp(rNroI.tipoDoc,tDoc) == 0)
      sigo = false;
  }
} //BusLinNrosInic

void InicCabCli(byte &nItem, float &tFac, long &nDoc) {

  nItem = 0;
  tFac = 0.0;
  nDoc++;
} //InicCabCli

void ObtHora(sHor &rHor) {

  cout << "FACTURACION A CLIENTES" << endl;
  cout << " EN CUENTA CORRIENTE" << endl;
  GetTime(rHor.Hora,rHor.Min,rHor.Seg);
} //ObtHora

bool BusBinCli(FILE *Cli, long cCli, sCli &rCli) {
  int pri,
      ult,
      med;

  pri = 0;
  fseek(Cli,0L,SEEK_END);
  ult = ftell(Cli) / sizeof (rCli) - 1;
  while (pri <= ult) {
    med = (pri + ult) / 2;
    fseek(Cli,med * sizeof (rCli),SEEK_SET);
    fread(&rCli,sizeof (rCli),1,Cli);
    if (rCli.codCli == cCli)
      return true;
    else
      if (rCli.codCli < cCli)
        pri = med + 1;
      else
        ult = med - 1;
  }
  return false;
} // BusBinCli
```

```
void BusDDVen(FILE *Ven, word cVen, sVen &rVen, FILE *AuxiV) {
  word refVen;

  fseek(AuxiV,cVen * sizeof(rVen),SEEK_SET);
  fread(&refVen,sizeof(refVen),1,AuxiV);
  fseek(Ven,refVen * sizeof(rVen),SEEK_SET);
  fread(&rVen,sizeof(rVen),1,Ven);
} //BusDDVen

void EmiteCabCli(sCli rCli, str20 nVen, sFec rFec, sHor rHor, long nDoc) {
  cout << setw(38) << " " << "FACTURA: " << nDoc << endl;
  cout << setw(38) << " " << "FECHA..: " << setw(2) << rFec.dia << '-' <<
                                 setw(2) << rFec.mes << '-' <<
                                 setw(4) << rFec.year << endl;
  cout << setw(38) << " " << "HORA...: " << setw(2) << rHor.Hora << '-' <<
                                 setw(2) << rHor.Min << '-' <<
                                 setw(4) << rHor.Seg << endl;
  cout << "Cod. Cliente.: " << setw( 8) << rCli.codCli <<
         "Raz. Social..: " << setw(20) << rCli.razSoc <<
         "Domicilio....: " << setw(20) << rCli.domic <<
         "Forma de Pago: " << setw(20) << rCli.formaPago <<
         "Cod. Vendedor: " << setw( 3) << nVen << endl;
  cout << endl;
  cout << setw(40) << "-" << endl;
  cout << "Item  Cant.  Cod.Art. Descripcion       Pre.Uni.    T.Item" << endl;
  cout << setw(40) << "-" << endl;
} //EmiteCabCli

void BusDDArt(FILE *Art, byte cArt, sArt &rArt) {

  fseek(Art,cArt * sizeof(rArt),SEEK_SET);
  fread(&rArt,sizeof(rArt),1,Art);
} //BusDDArt

void CalcDetFac(word cant, sArt &rArt, byte &nItem, float &tItem,float &totFac) {

  nItem++;
  tItem = cant * rArt.precio;
  rArt.stockAct -= cant;
  totFac += tItem;
} //CalcDetFac

void ActArt(FILE *Art, sArt rArt) {

  fseek(Art,ftell(Art) - sizeof(rArt),SEEK_SET);
  fread(&rArt,sizeof(rArt),1,Art);
} //ActArt

void EmiteDetFac(byte nItem, sArt rArt, word cant, float tItem) {
```

```
  cout << setw(2) << nItem <<
        setw(5) << cant <<
        setw(3) << rArt.codArt <<
        setw(20) << rArt.descrip <<
        setw(8) << rArt.precio <<
        setw(8) << tItem << endl;
} //EmiteDetFac

void CalcPieCli(float tBruFac,float &impIVA, float &tNetoFac, float &saldo, sVen
                    &rVen) {

  impIVA = tBruFac * IVA;
  tNetoFac = tBruFac + impIVA;
  saldo += tNetoFac;
  rVen.impAcumCom += tBruFac * rVen.porcCom / 100;
} //CalcPieCli

void ActCli(FILE *Cli, sCli rCli) {

  fseek(Cli,ftell(Cli) - sizeof(rCli),SEEK_SET);
  fwrite(&rCli,sizeof(rCli),1,Cli);
} //ActCli

void ActVen(FILE *Ven, sVen rVen) {

  fseek(Ven,ftell(Ven) - sizeof(rVen),SEEK_SET);
  fwrite(&rVen,sizeof(rVen),1,Ven);
} //ActVen

void AgregarRegFac(FILE *Fac, long nFac, sFec rFec, long cCli, float tNetoFac) {
  sFac rFac;

  rFac.nroFac = nFac;
  rFac.fecFac.dia = rFec.dia;
  rFac.fecFac.mes = rFec.mes;
  rFac.fecFac.year = rFec.year;
  rFac.codCli = cCli;
  rFac.impo = tNetoFac;
  rFac.estado = 'A';
  fwrite(&rFac,sizeof(rFac),1,Fac);
} //AgregarRegFac

void EmitePieFac(byte nItem, float tBruFac, float impIVA, float tNetoFac) {
  byte i;

  for (i = 1; i <= 10 - nItem; i++)
    cout << endl;
  cout << setw(38) << " " << "Sub-Total.: " << setw(8) << tBruFac <<
        setw(38) << " " << "I.V.A.....: " << setw(8) << impIVA <<
```

```
        setw(38) << " " << "Total Neto: " << setw(8) << tNetoFac << endl;
} //EmitePieFac

void ActNroI(FILE *nInic, sNro rNroI) {

  fseek(nInic,ftell(nInic) - sizeof(rNroI),SEEK_SET);
  fwrite(&rNroI,sizeof(rNroI),1,nInic);
} //ActNroI

void CerrarEliminar() { //FILE *Ped, FILE *Cli, FILE *Art, FILE *Ven,FILE *nInic,
                        //FILE *Fac, FILE *AuxVen)

  fcloseall();
  remove("AuxVen.Tmp");
} //CerrarEliminar

int main() {
  FILE *Pedidos,
       *Clientes,
       *Articulos,
       *Vendedores,
       *NrosInic,
       *Facturas,
       *AuxVendedores;
  sFec rFecha;
  sHor rHora;
  sPed rPedido;
  sNro rNroInic;
  sCli rCliente;
  sVen rVendedor;
  sArt rArticulo;
  byte nroItem;
  float totItem,
        totBrutoFac,
        impIVA,
        totNetoFac;

Abrir(&Pedidos,&Clientes,&Articulos,&Vendedores,&NrosInic,&Facturas,
        &AuxVendedores);
  freopen("FacturasCtaCteCli.Lst","w",stdout);
  ArmarAuxVen(Vendedores,AuxVendedores);
  EmiteTapaObtFecha(rFecha);
  BusLinNrosInic(NrosInic,"FAC",rNroInic);
  fread(&rPedido,sizeof(rPedido),1,Pedidos);
  while (not feof(Pedidos)) {
    InicCabCli(nroItem,totBrutoFac,rNroInic.nroDoc);
    ObtHora(rHora);
    BusBinCli(Clientes,rPedido.codCli,rCliente);
    BusDDVen(Vendedores,rCliente.codVen,rVendedor,AuxVendedores);
    EmiteCabCli(rCliente,rVendedor.nomVen,rFecha,rHora,rNroInic.nroDoc);
```

```
    while (not feof(Pedidos) and rPedido.codCli == rCliente.codCli) {
      BusDDArt(Articulos,rPedido.codArt,rArticulo);
      CalcDetFac(rPedido.cant,rArticulo,nroItem,totItem,totBrutoFac);
      ActArt(Articulos,rArticulo);
      EmiteDetFac(nroItem,rArticulo,rPedido.cant,totItem);
      fread(&rPedido,sizeof(rPedido),1,Pedidos);
    }
    CalcPieCli(totBrutoFac,impIVA,totNetoFac,rCliente.saldo,rVendedor);
    ActCli(Clientes,rCliente);
    ActVen(Vendedores,rVendedor);
    AgregarRegFac(Facturas,rNroInic.nroDoc,rFecha,rCliente.codCli,totNetoFac);
    EmitePieFac(nroItem,totBrutoFac,impIVA,totNetoFac);
  }
  ActNroI(NrosInic,rNroInic);
  freopen("CON","w",stdout);
  CerrarEliminar();//usa fcloseall()  Pedidos, Clientes,Articulos, Vendedores, NrosInic,
                   //                  Facturas, AuxVendedores);
  return 0;
}
```

Ejemplo G2Ej13 Listado de Gastos ordenado Mes-Dia ImpAcum

```
/*
    Id.Programa: G2Ej13Gtos.cpp
    Autor......: Lic. Hugo Cuello
    Fecha......: ago-2014
    Comentario.: Listado de Gastos ord. x Mes y Dia Imp.Acum.
*/
#include<iomanip>
#include<iostream>
using namespace std;
  typedef unsigned short byte;
  struct sGto {
      byte  mes,
            dia;
      float impo;
};

// Prototipos -------------------------
void Abrir(FILE **, FILE **);
void ArmarAuxG(FILE *);
int CantDias(byte );
void AcumGto(FILE *, sGto );
string MesStr(unsigned );
void Listado(FILE *);
void CerrarEliminar(FILE *, FILE *);
// Fin Prototipos ---------------------

void Abrir(FILE **Gtos, FILE **AuxG) {
```

```
  *Gtos = fopen("Gastos.Dat","rb");
  *AuxG = fopen("AuxGtos.Tmp","w+b");
} //Abrir

void ArmarAuxG(FILE *AuxG) {
  float impAcum = 0;

  for (int i = 1; i <= 365; i++)
    fwrite(&impAcum,sizeof(impAcum),1,AuxG);
} //ArmarAuxG

int CantDias(byte mes) {
  int sumDias = 0;

  for (int mes_i = 1; mes_i < mes; mes_i++)
    switch (mes_i) {
      case  4:
      case  6:
      case  9:
      case 11: sumDias += 30;
               break;
      case  2: sumDias += 28;
               break;
      default: sumDias += 31;
    }
  return sumDias;
} //CantDias

void AcumGto(FILE *AuxG, sGto rGto) {
  float impAcum;
  int   pos;

  pos = CantDias(rGto.mes) + rGto.dia - 1;
  fseek(AuxG,pos * sizeof(float),SEEK_SET);
  fread(&impAcum,sizeof(impAcum),1,AuxG);
  impAcum += rGto.impo;
  fseek(AuxG,pos * sizeof(float),SEEK_SET);
  fwrite(&impAcum,sizeof(float),1,AuxG);
} //AcumGto

string MesStr(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
```

```
    case  8: return "Agosto";
    case  9: return "Septiembre";
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStr

void Listado(FILE *AuxG) {
  float impAcum,
        totGral = 0.0,
        totMes,
        impMenDia,
        impMayMes = 0.0;
  byte  nroDiaMen,
        nroMesMay;

  rewind(AuxG);
  freopen("GtosMesDiaAcum.Lst","w",stdout);
  cout.precision(2);
  cout.setf(ios::fixed);
  cout << "Listado de Gastos ord. por Mes y Dia por Imp.Acum." << endl;
  for (int mes = 1; mes <= 12; mes++) {
    totMes = 0;
    impMenDia = 1E6;
    cout << "Mes: " << MesStr(mes) << endl;
    cout << setw(7) << " " << "Dia    Imp.Acu." << endl;
    for (byte dia = 1; dia <= CantDias(mes + 1) - CantDias(mes); dia++) {
      fread(&impAcum,sizeof(impAcum),1,AuxG);
      if (impAcum) {
        if (impAcum < impMenDia) {
          impMenDia = impAcum;
          nroDiaMen = dia;
        }
        cout << setw(8) << " " << setw(2) << dia
             << setw(4) << " " << setw(8) << impAcum << endl;
        totMes += impAcum;
      }
    }
    totGral += totMes;
    if (impMenDia > impMayMes) {
      impMayMes = impMenDia;
      nroMesMay = mes;
    }
    cout << "Tot. Mes: " << totMes << endl;
    cout << "Nro. Dia < $: " << nroDiaMen << endl << endl;
  }
  cout << "Tot. Anual: " << totGral << endl;
```

```
  cout << "Nro. Mes de > $ de los < Dias: " << setw(2) <<
          nroMesMay << "(" << MesStr(nroMesMay) << ")" << endl;
  freopen("CON","w",stdout);
} //Listado

void CerrarEliminar(FILE *Gtos, FILE *AuxG) {

  fclose(Gtos);
  fclose(AuxG);
  remove("AuxGtos.Tmp");
} //CerrarEliminar

int main() {
  FILE *Gastos,
          *AuxGtos;
  sGto rGto;

  Abrir(&Gastos,&AuxGtos);
  ArmarAuxG(AuxGtos);
  while (fread(&rGto,sizeof(rGto),1,Gastos))
    AcumGto(AuxGtos,rGto);
  Listado(AuxGtos);
  CerrarEliminar(Gastos,AuxGtos);
  return 0;
}
```

Unidad IV Capítulo II

Unidad IV – Capítulo II - Archivos: FILE archivos en C/C++

Clasificación de archivos en el lenguaje C/C++

- ✓ **FILE* (variante para C / C++)**
- ✓ **STREAM (variante exclusiva para C++)**

En ambos casos se presentan las siguientes variantes:

1. **Texto**
2. **Binario**

Estilo FILE *

Archivo de texto

Un archivo de texto es un archivo con organización secuencial, solo se permite acceso secuencial a sus componentes. Las componentes de un archivo de texto se denominan líneas y cada línea finaliza con una marca, denominada fin de línea (**eoln**). El fin de un archivo está indicado por otra marca, denominada fin de archivo (**eof**). Cada una de las líneas de un archivo de texto es de longitud variable. Cada línea puede leerse en forma completa y almacenarse en una variable de tipo cadena. También es posible leer caracteres individuales de una línea, almacenada en una variable de tipo carácter. Otra alternativa de leer una línea es, si sabemos de antemano como está constituida cada una de las líneas, debe tener un formato fijo preestablecido, podremos leer cada uno de estos valores en una línea en correspondientes variables de acuerdo al tipo de valor a leer, p.e. si sabemos que por cada línea se registraron 4 valores y, siendo estos valores, un entero, un carácter, un valor real y una cadena de 20 caracteres como máximo, entonces para leer esta línea utilizamos cuatro variables según los tipos indicados recientemente, como ser **int** a, **char** b, **float** c, **char** d[20], luego por cada línea de este archivo de texto, presentan las mismas situaciones regulares como fue indicado. El archivo de texto debe su nombre porque al grabar datos en el archivo con los datos que están previamente en la memoria interna, **éstos datos están en una representación binaria, pero al grabarse en el archivo deben ser convertidos a texto** y **cuando se lean del archivo para volcar esos datos a la memoria interna también debe existir una conversión**. En el primer caso, esto es cuando grabamos la **conversión de datos es de binario a texto**, y en el segundo caso, esto es cuando leemos, la **conversión de datos es de texto a binario**.

Un archivo de texto es abierto solo para leer o solo para grabar.

Al abrir un archivo de texto de **solo lectura**, el archivo indicado debe existir y en este caso ubica el puntero al archivo al comienzo del mismo. En caso que no exista el archivo indicado ocasionará un error, por archivo inexistente.

Al abrir un archivo de texto **solo para grabar**, si no existe se crea desde cero o vacío. En caso que exista el archivo indicado se destruye todo su contenido y se lo crea desde cero o vacío. Por lo que se deberá tener sumo cuidado al abrir un archivo de texto en esta modalidad ya que podríamos perder todos los datos, salvo que estemos completamente seguros de esta situación.

Un tercer modo de apertura de archivos de texto es el modo **añadir** o **agregar**, esto hace que el puntero, si el archivo existe previamente, se sitúa al final del mismo, es decir, sobre la marca de fin de archivo. Esto permite agregar nuevas componentes al archivo de texto. Si no existe previamente el archivo de texto se lo crea desde cero o vacío y ubica el puntero al final, sobre la marca de fin de archivo.

Ejemplo

```
void main() {
  int a;
  FILE *ft;

  ft = fopen("Entero.Txt","w");
  a = 23;
  fprintf(ft,"%d\n",a);
  fclose(ft);
}
```

Representemos en un gráfico las acciones realizadas en la codificación previa:

Al abrir el archivo

Vemos que el puntero coincide con la marca de EOF (End Of File = Fin De Archivo), el tamaño del archivo es cero.

Luego de grabar el dato

Observamos que el puntero luego de grabar una componente avanza una posición y la marca de EOF es movida al final del archivo. La longitud del archivo es de uno. La posición inicial está indicada por el valor cero. El contenido de la primer componente línea son dos caracteres, el carácter 2 y el carácter 3. Se observa también que al finalizar la línea se incorpora una marca de fin de línea (End Of Line = Fin De Línea).

El ejemplo de **programa** indicado más arriba, nos muestra la declaración de la variable ft como un puntero a un tipo **FILE**. Esta variable representa el nombre lógico del archivo, y este nombre es el que se utilizará posteriormente con las sentencias referidas al archivo.

fopen()

FILE * fopen(const char *nombreArchivo, const char *modoApertura)

fopen retorna un puntero a un tipo FILE, generalmente esto se asigna a una variable de ese tipo FILE, p.e., la siguiente definición: **FILE** *f; hace que f sea una variable puntero a un tipo archivo. Luego al abrir un archivo realizamos la siguiente sentencia:

f = fopen("Ejemplo.Txt","w")

Observamos que el valor del primer argumento "Ejemplo.Txt", indica el nombre físico del archivo, al que se le podrá incluir una unidad de disco y su ruta, el valor de este primer argumento podrá ser un valor constante, como en el ejemplo, o una variable o una expresión de cadena. El valor del segundo argumento indica el modo de apertura del archivo, en este caso, el archivo se abre para grabar, y precisamente es un archivo de texto, también podríamos haber escrito en el modo de apertura equivalente al ejemplo como "wt", pero, generalmente usamos la forma indicada en el ejemplo, ya que es la opción por defecto. La función fopen asocia un flujo con un manejador de archivo y esta retorna un puntero que identifica el flujo con el archivo.

Antes de operar con el archivo debemos abrirlo, esto se logra por medio de la función **fopen**, que establece la apertura del archivo, el cual debemos indicar como primer argumento el nombre físico del archivo, el cual puede incluir la unidad de disco y la ruta y, como segundo argumento indicamos el **modo de apertura** del archivo, los cuales podrán ser;

Modo de apertura

"r", "rt" solo lectura,
"w", "wt" solo escritura,
"a", "at" agregar o añadir más componentes al final del archivo,
"r+", "r+t", "rt+" para lectura/escritura,
"w+", "w+t", "wt+" para crearlo y leer,
"a+", "a+t", "at+" para agregar nuevas componentes y leer.

Estos modos de apertura de archivos son para **archivos de texto**, ya que si no indicamos el formato de archivo, asume por defecto el de texto. Una alternativa para indicar que nos referimos a archivos de texto, es indicando la letra "t" por texto, como en "rt", "wt", "at", aunque no es necesario indicarlo de esta manera, en cambio si indicamos el modo de apertura como, "r+t", "w+t", "a+t", o sus equivalentes "rt+", "wt+", "at+", abre un archivo para leer y grabar, o abre un archivo para grabar y leer, o abre un archivo para agregar más componentes y leer. Esta función retorna un puntero a FILE el cual es asignado a la variable de tipo FILE.

freopen()

FILE *freopen(**const char** *nombreArchivo, **const char** *modo, **FILE** *flujo)

Asocia un nuevo archivo con un flujo ya abierto. Reemplaza un archivo por un flujo abierto, cerrando el flujo. Es sumamente poderoso freopen para cambiar los flujos stdin, stdout o strerr. El valor retornado por la función si la operación fue exitosa es el argumento al flujo, caso contrario, retorna NULL.

Ejemplo

FILE *ft;
ft = freopen("Salida.Txt","w",stdout)

ft = freopen("CON","w",staout)

El primer ejemplo reencauza la salida por defecto que es la pantalla a otro dispositivo, hacia el disco, como archivo de texto. Es decir las salidas producidas por ejemplo con fprintf o con cout, en lugar de ir a la pantalla, ahora van a un archivo de texto.

El segundo ejemplo reencauza la salida hacia el dispositivo estándar o por defecto, es decir, las salidas producidas por ejemplo con fprintf o cout, dejan de ir al arhivo en disco, y vuelven a salir hacia la pantalla o monitor.

Las **operaciones** de lectura y escritura, permiten leer datos o grabar datos al archivo respectivamente y para ello se pueden utilizar diferentes funciones. En el ejemplo, el archivo se abrió para crearlo como archivo nuevo, en el cual solo se graban registros. La función utilizada para grabar en el ejemplo es, **fprintf**, función semejante a printf, solo que en este caso se indica como primer argumento el dispositivo al que hace referencia, en este caso, la variable que representa el nombre lógico del archivo de texto ft. También observamos que esta función graba datos en el archivo con formato, indicado como segundo argumento, y finalmente el tercer argumento o siguientes indican los valores a grabar, que se corresponde c/u. con el segundo argumento que establecen los formatos de c/u. de las componentes a guardar en el archivo. Una vez finalizadas todas las operaciones de lectura/escritura con el archivo, debemos cerrarlo, esta acción se realiza por medio de la función **fclose**. No solo cierra el archivo indicado, sino, que además vacía el buffer del archivo, forzando a enviarlo al archivo lo que quedó allí, si obviamente el archivo en cuestión fuese de escritura, debido a que éste no se llenó completamente, razón por la cual fclose obliga a forzar el vaciado, para no perder esos datos, que en caso de no haberse grabado, lo hubiéramos perdido. La función utilizada para leer datos de un archivo de texto es **fscanf** que funciona de forma semejante a su equivalente para leer datos desde el dispositivo estándar que es el teclado, pero en este caso lee datos de un archivo de texto. Otras funciones para lectura en un archivo de texto son, **fgetc** que lee un carácter, **fgets** que lee una cantidad de caracteres de un string y para grabar las funciones adicionales son **fputc** el cual graba un carácter, **fputs** que graba una cantidad de caracteres de un string.

Las sintaxis de estas funciones se indican a continuación:

Funciones de lectura en archivo de texto

int fgetc(**FILE** *flujo)
char *fgets(**char** *cad, **int** n, **FILE** *flujo)
int fscanf(**FILE** *flujo, **const char** *formato [,dirección, …])

Funciones de escritura en archivo de texto

int fputc(**int** car, **FILE** *flujo)
int fputs(**const char** *cad, **FILE** *flujo)
int fprintf(**FILE** *flujo, **const char** *formato [,argumento, …])

Ejemplo de archivo de texto

Agrega nuevas componentes al archivo luego lo recorre para leer

```
/*
   Id.Programa: fileTexto01.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: jun-2016
   Comentario.: Operaciones de lectura/escritura
                en archivo de texto
*/

#include <fstream>
using namespace std;

main() {
  FILE *t;
  char car,
       cad20[21];

  t = fopen("CadCar.Txt","a+"); // Si no existe lo crea o
                                // Si existe ubica puntero al final.
  cout << "Ingresar 5 caracteres..." << endl;
  for (int i = 1; i <= 5; i++) {
    cout << "Car: ";
    cin >> car;
    fputc(car,t);
    fputc(' ',t);
    if (i == 3)
      fputc('\n',t);
  }
  fputc('\n',t);
  cout << "Ingresar 5 cadenas -no frases-" << endl;
  for (int i = 1; i <= 5; i++){
```

```
    cout << "Cad: ";
    cin >> cad20;
    fputs(cad20,t); // corta al leer un espacio en blanco
    fputc(' ',t);
    if (i == 3)
      fputc('\n',t);
  }
  fputc('\n',t);
  cout << "Ingresar 5 cadenas o frases" << endl;
  for (int i = 1; i <= 5; i++){
    cout << "*Cad: ";
    gets(cad20); // lee incluso espacios en blanco
    fprintf(t,"%s",cad20);
    fputc(' ',t);
    if (i == 3)
      fputc('\n',t);
  }
  fputc('\n',t);
  rewind(t);
  car = fgetc(t);
  while (!feof(t)){ // o (car != EOF)
    while (car != '\n') {
      cout << car;
      car = fgetc(t);
    }
    cout << endl;
    system("pause");
    car = fgetc(t);
  }
  fclose(t);
} //main
```

Lee archivo de texto y muestra en pantalla

```
/*
    Id.Programa: 1.LeeTxtMstPan.cpp
    Autor......: Lic. Hugo Cuello
    Fecha......: jun-2014
    Comentario.: Lee archivo de texto y muestra datos en Pantalla
*/

#include <iomanip.h>
#include <fstream>
```

```
using namespace std;
typedef char str20[21];
  struct sVen {
    int   codVen,
          cant;
    float preUni;
    str20 descrip; // eq. char descrip[21];
  };

int main () {
  FILE *VentasTxt; // es el nombre logico del archivo.
  sVen rVenta;
  char car;

  VentasTxt = fopen("Ventas.Txt","r");
  cout.precision(2);
  cout.setf(ios::fixed);
  cout << "           Listado de Ventas" << endl;
  cout << endl << "Cod.Ven. Cant.  Pre. Uni.  Descripcion" << endl;
  fscanf(VentasTxt,"%d%d%f%c",&rVenta.codVen,
                              &rVenta.cant,
                              &rVenta.preUni,
                              &car);
  fgets(rVenta.descrip,22,VentasTxt);
  while (!feof(VentasTxt)) {
    //cout.setf(ios::right);
    cout << "  " << setw(3) << rVenta.codVen << " ";
    cout << "    " << setw(4) << rVenta.cant << " ";
    cout << "   " << setw(7) << rVenta.preUni << " ";
    rVenta.descrip[strlen(rVenta.descrip) - 1] = '\0';
    cout << " " << rVenta.descrip << endl;
    fscanf(VentasTxt,"%d%d%f%c",&rVenta.codVen,
                              &rVenta.cant,
                              &rVenta.preUni,
                              &car);
    fgets(rVenta.descrip,22,VentasTxt);
  }
  fclose(VentasTxt);

  return 0;
}
```

Ingresa datos por teclado y graba en archivo de texto

```
/*
   Id.Programa: 2.IngTecGraTxt.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: jun-2014
   Comentario.: Ingresa datos por teclado y graba en archivo de texto
*/

#include <conio.h>
#include <iomanip.h>
#include <fstream>
using namespace std;
typedef char str20[21];

struct sVen {
   int   codVen,
         cant;
   float preUni;
   str20 descrip;
  };

// Prototipos
void Titulos();
void ObtDatos(sVen &);
// Fin Prototipos

void Titulos() {
  gotoxy(10,1);
  cout << "Ingreso de Ventas de Articulos";
  gotoxy(10,5);
  cout << "Cod.Art.Fin=0:";
  gotoxy(10,8);
  cout << "Cantidad.....:";
  gotoxy(10,10);
  cout << "Descripcion..:";
  gotoxy(10,12);
  cout << "Pre.Unitario.:";
} // Titulos

void ObtDatos(sVen &rVen) {

  Titulos();
  gotoxy(25,5); clreol();
```

```
  cin >> rVen.codVen;
  if(rVen.codVen) {
    gotoxy(25,8); clreol();
    cin >> rVen.cant;
    gotoxy(25,10); clreol();
    gets(rVen.descrip);
    gotoxy(25,12); clreol();
    cin >> rVen.preUni;
  }
} // ObtDatos

int main () {
  FILE *aVentasTxt;
  sVen rVenta;

  aVentasTxt = fopen("VentasTec.Txt","w");
  cout.precision(2);
  cout.setf(ios::fixed);

  ObtDatos(rVenta);
  while (rVenta.codVen) {
    fprintf(aVentasTxt,"%3d %4d %7.2f %-20s\n",rVenta.codVen,
                                    rVenta.cant,
                                    rVenta.preUni,
                                    rVenta.descrip);
    ObtDatos(rVenta);
  }
  fclose(aVentasTxt);

  return 0;
}
```

Lee archivo de texto y graba en archivo binario

```
/*
    Id.Programa: 3.LeeTxtGraBin.cpp
    Autor......: Lic. Hugo Cuello
    Fecha......: jun-2014
    Comentario.: Lee archivo de texto y graba en archivo binario
*/

#include <iomanip.h>
#include <fstream>
```

```
using namespace std;
typedef char str20[21];

struct sVen {
   int    codVen,
          cant;
   float  preUni;
   str20  descrip;
  };

int main () {
  FILE  *aVentasTxt,
        *aVentas;
  sVen  rVenta;
  char  car;

  cout.precision(2);
  cout.setf(ios::fixed);
  aVentasTxt = fopen("Ventas.Txt","r");
  aVentas    = fopen("Ventas.Dat","wb");
  fscanf(aVentasTxt,"%d%d%f%c",&rVenta.codVen,&rVenta.cant,&rVenta.preUni,
           &car);
  fgets(rVenta.descrip,22,aVentasTxt);
  while (!feof(aVentasTxt)) {
     fwrite(&rVenta, sizeof(rVenta),1,aVentas);
      fscanf(aVentasTxt,"%d%d%f%c",&rVenta.codVen,&rVenta.cant, &rVenta.preUni,
               &car);
     fgets(rVenta.descrip,22,aVentasTxt);
  }
  fclose(aVentasTxt);
  fclose(aVentas);
  return 0;
}
```

Lee archivo binario y muestra en pantalla

```
/*
   Id.Programa: 4.LeeBinMstPan.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: jun-2014
   Comentario.: Lee archivo binario y muestra datos en pantalla
*/

#include <iomanip.h>
```

```
#include <fstream>
using namespace std;
typedef char str20[21];
struct sVen {
    int   codVen,
          cant;
    float preUni;
    str20 descrip;
  };

int main () {
  FILE *aVentas;
  sVen rVenta;
  char car;

  aVentas = fopen("Ventas.Dat","rb");
  cout.precision(2);
  cout.setf(ios::fixed);
  fread(&rVenta,sizeof(rVenta),1,aVentas);
  while (!feof(aVentas)) {
    cout.setf(ios::right);
    cout << setw(3) << rVenta.codVen << " ";
    cout << setw(4) << rVenta.cant << " ";
    cout << setw(7) << rVenta.preUni << " ";
    rVenta.descrip[strlen(rVenta.descrip) - 1] = '\0';
    cout << rVenta.descrip << endl;
    fread(&rVenta,sizeof(rVenta),1,aVentas);
  }
  fclose(aVentas);

  return 0;
}
```

Lee archivo binario y graba en archivo de texto

```
/*
    Id.Programa: 5.LeeBinGraTxt.cpp
    Autor......: Lic. Hugo Cuello
    Fecha......: jun-2014
    Comentario.: Lee archivo binario graba en archivo de texto
*/

#include <iomanip.h>
```

```
#include <fstream>
using namespace std;

typedef char str20[21];
struct sVen {
   int   codVen,
         cant;
   float preUni;
   str20 descrip;
  };

int main () {
  FILE *aVentas,
         *aVentasTxt;
  sVen rVenta;
  char car;

  aVentas = fopen("Ventas.Dat","rb");
  aVentasTxt = fopen("VentasCpy.Txt","w");
  cout.precision(2);
  cout.setf(ios::fixed);
  fprintf(aVentasTxt,"%5sListado de Ventas de Articulos\n\n"," ");
  fprintf(aVentasTxt,"Cod.Art. Cant.  Pre.Uni.   Descripcion\n");
  fread(&rVenta,sizeof(rVenta),1,aVentas);
  while (!feof(aVentas)) {
    rVenta.descrip[strlen(rVenta.descrip) - 1] = '\0';
    fprintf(aVentasTxt,"   %3d   %4d   %7.2f    %-20s\n",rVenta.codVen,
                                          rVenta.cant,
                                          rVenta.preUni,
                                          rVenta.descrip);
    fread(&rVenta,sizeof(rVenta),1,aVentas);
  }
  fclose(aVentas);
  fclose(aVentasTxt);

  return 0;
}
```

Archivo binario

Un archivo binario es la representación o imagen de la memoria interna, es decir, no existe una conversión de formato de los datos, como si existe en los archivos de texto. Por lo que podríamos decir que el procesamiento de archivos binarios en más rápida que con los archivos de texto, debido a que no existen conversiones, tanto para

volcar los datos del archivo a la memoria interna o desde la memoria interna al disco. Tomemos el siguiente ejemplo:

| 0000 0000 | 0001 0111 |

La representación anterior está representada en el formato de dato binario entero con signo, con un tamaño de 2 bytes, el bit de mayor peso del byte izquierdo, representa el signo del número en este caso es cero indicando positivo y la representación del número está en forma directa y los restantes 15 bits son utilizados para representar el número; si tomamos aquellos bits 1's, tenemos los siguientes valores de izquierda a derecha; 16+4+2+1 = 23 en base 10, por lo que la interpretación de la representación binaria es el valor positivo entero 23. Ahora bien, al grabar este dato en un archivo binario, y si luego abrimos este archivo con un editor de texto, notaremos que la representación está formada por dos caracteres que representan, por un lado el carácter nulo, es decir sin representación gráfica, seguida por el símbolo ↕

| ↕ |

Los archivos binarios están estructurados por **componentes que denominamos registros en su sentido más amplio de la palabra**, aunque no necesariamente de tipo registro. Si por cada componente guardamos un solo dato, este dato puede que sea de tipo simple, como ser un entero con o sin signo, real, carácter, booleano o cadena. En cambio, si por cada componente guardamos más de un dato y, si cada uno de estos datos son de distinto tipo, en estos casos el tipo de componente lo definimos de tipo registro, en C/C++ tipo **struct** y, si por cada componente guardamos más de un dato y, si cada uno de estos datos son de igual tipo, en este otro caso el tipo de componente lo definimos de tipo arreglo, estructura de datos que se analizará más adelante. Un archivo binario no contiene marcas de fin de línea ya que sus componentes no se los denomina líneas, pero si contiene una marca al final del archivo, denominada marca de fin de archivo (eof). Las posiciones de cada componente se enumeran a partir de la **posición cero**, luego uno, dos y así sucesivamente. Podremos acceder a cada componente del archivo con acceso secuencial o bien con acceso al azar o aleatorio (random).

El **acceso secuencial** presenta la siguiente característica, se lee o se graba la componente en donde se encuentre localizado el puntero al archivo, luego de la operación de lectura o escritura el puntero del archivo avanza automáticamente a la próxima posición o componente. Este tipo de acceso es muy veloz cuando queremos acceder a las componentes en el mismo orden en que estas fueron creadas. Si por el contrario, queremos acceder a las componentes en distinto orden a como estos fueron grabados, este acceso es muy lento y no es aconsejable su implementación.

Por otro lado, para acceder a los registros en distinto orden en que fueron grabadas el acceso conveniente es el **acceso al azar**, este acceso permite acceder a una posición n, sin necesidad de pasar por aquellas componentes que le preceden, por lo que el tiempo de acceso es independiente del lugar en donde se encuentre el puntero del archivo para llegar a una componente cualquiera. Vale decir, si estamos al comienzo del

archivo, posición cero, y queremos ubicarnos en la posición 1563, el tiempo de acceso es igual, si el puntero del archivo se encuentra en la posición 1562 y queremos ubicarnos en la misma posición 1563. Cabe aclarar que el cambio de ubicación puede realizarse hacia adelante o hacia atrás, de donde nos encontramos para ir a la nueva posición. Además existe la posibilidad de desplazarnos a la nueva posición, tomando en cuenta desde la primer posición del archivo o desde la posición actual en donde está situado el puntero del archivo en ese instante.

Cuando leemos o grabamos en un archivo binario, se lee o se graba toda la componente completa, esto significa que si el tipo de componente fuese de tipo registro, entonces al leer o grabar esa componente, se leerán o grabarán todos los campos representados en el tipo registro.

Recordemos lo que fue indicado en párrafos previos en situaciones de lectura o escritura, si estamos fuera de los límites del archivo antes del inicio, después del inicio, o dentro de su tamaño actual.

Cuando recorramos el archivo en forma secuencial para leer sus componente, sabremos que hemos llegado al final, al detectar la marca de fin de archivo. Esta marca se detecta por medio de la función **feof(f)**, que presenta la siguiente forma:

Abrir un archivo

fopen()

FILE * fopen(**const char** *nombreArchivo, **const char** *modoApertura)

fopen retorna un puntero a un tipo FILE, generalmente esto se asigna a una variable de ese tipo FILE, p.e., la siguiente definición: **FILE** *fb; hace que fb sea una variable puntero a un tipo archivo. Luego al abrir un archivo realizamos la siguiente sentencia:

fb = fopen(“Ejemplo.Dat”,”wb”)

Observamos que el valor del primer argumento “Ejemplo.Dat”, que es una cadena de caracteres, indica el nombre físico del archivo, al que se le podrá incluir una unidad de disco y su ruta, el valor de este primer argumento podrá ser un valor constante, como en el ejemplo, o una variable o una expresión de cadena. El valor del segundo argumento indica el modo de apertura del archivo, que también es una cadena de caracteres, en este caso, el archivo se abre para grabar, y precisamente es un archivo

binario. La función fopen asocia un flujo con un manejador de archivo y esta retorna un puntero que identifica el flujo con el archivo.

Antes de operar con el archivo debemos abrirlo, esto se logra por medio de la función **fopen**, que establece la apertura del archivo, el cual debemos indicar como primer argumento el nombre físico del archivo, el cual puede incluir la unidad de disco y la ruta y, como segundo argumento indicamos el **modo de apertura** del archivo, los cuales podrán ser;

Modo de apertura

"rb" solo lectura,
"wb" solo escritura,
"rb+", "r+b" para lectura/escritura,
"wb+", "w+b" para crearlo y leer,
"ab" agrega, abre o crea solo grabar al final.
"ab+" agrega , abre o crea para actualizar en binario, graba al final del archivo.
Permite leer. Esta función retorna un puntero a FILE el cual es asignado a la variable de tipo FILE.

Si el archivo pudo ser abierto, entonces la función retorna un puntero genérico a FILE, caso contrario, la función retorna el valor **NULL**.

¿Qué sucede con fb luego de abrir el archivo del ejemplo anterior?

La función fopen retorna un puntero a la estructura FILE definido en <stdio.h> el cual contiene un descriptor, un entero, que indica un índice a la tabla de apertura de archivo. Al hacer una llamada de E/S como fprintf(fb,"%d",i), se localiza el **FCB** (*File Control Block*) en la tabla de apertura de archivo. Los datos localizados en este lugar se copian desde el FCB al disco, al abrir el archivo. El usuario no puede acceder a esta tabla, solo el S.O.

Detectar marca fin de archivo

feof()

int feof(**FILE** *flujo)

Macro que retorna un indicador de true, valor distinto de cero, si se alcanzó el fin de archivo de un flujo y este permanece hasta tanto no se realice un rewind, o sea, un

rebobinado o no se realice un cierre del archivo, caso contrario, retorna un valor igual a cero, indicando que no se alcanzó el fin de archivo o **EOF**. El uso típico de esta macro es cuando recorremos para leer en forma secuencial un archivo, y en la cabecera del ciclo establecemos la condición, si no se alcanzó la marca de fin de archivo, como p.e. !feof(f), recordando que la primer lectura la realizamos antes de ingresar al ciclo, esto es diferente al caso del lenguaje Pascal, en que realizamos las lecturas dentro del ciclo y como primera acción de esta estructura de control.

Ejemplo:

```
// Lectura secuencial de un archivo binario
…
FILE *fb;

fb = fopen("ArcBin.Dat","rb");
fread(&rComp,sizeof(rComp),1,fb); // Observar 1er. lectura antes de ingresar al ciclo.
while (!feof(fb)) { // compara con distinto de cero !=0
  ProcReg(rComp);
  fread(&rComp,sizeof(rComp),1,fb); // Observar próx. lecturas, últ. acción en el ciclo.
}
fclose(fb);
```

Determinar posición del puntero y el tamaño del archivo

ftell()

long ftell(**FILE** *flujo)

Existe la posibilidad que ante el requerimiento del tamaño del archivo, podamos conocer este valor, por medio de la función **ftell**, cuando el puntero se encuentre posicionado sobre la última componente grabada en el archivo, **retorna la cantidad de bytes** hasta esa posición con respecto al comienzo del archivo, en cambio sí ocurre un error la función retorna -1L y pone a **errno** en un valor positivo.

Para saber cuántas componentes contiene basta hacer una división, en donde el numerador es **ftell** y el denominador es el tamaño de cada componente **sizeof**. Por ejemplo, si contamos con un archivo con 10 componentes y por cada componente guardamos 3 valores como ser **short** a, **float** b, **char** c, el tamaño de cada componente se puede determinar con la función sizeof, y si las variables a, b y c son campos de rComp con estructura struct sComp, entonces el sizeof(rComp) retornará 7 bytes, por lo que si la función ftell nos regresa 70, ya que el puntero del archivo asumimos que se encuentra ubicado en la componente 10; entonces el tamaño del archivo en cantidad de componentes 10 en este caso, quedaría determinado por la siguiente expresión:

Calcular el tamaño en cantidad de componentes

long ftell(fb) / sizeof(rComp)

Suponiendo que rComp, su tamaño, sizeof(rComp) = 16, y si el puntero al archivo asociado con este tipo de registro estuviese ubicado en su primera componente, o sea, posición = cero, entonces ftell(fb) retornará cero, si el puntero estuviese ubicado en la segunda componente, posición = uno, ftell(fb) retornará 16, si estuviese ubicado en la tercer componente, posición = dos, entonces ftell(fb) retornará 32, y así sucesivamente. Por lo tanto, si quisiéramos averiguar en un instante dado en qué posición se encuentra el puntero del archivo, basta realizar una división entre el valor informado por ftell(f) dividido por el tamaño de la componente, es decir, sizeof(rComp), según el ejemplo que hemos tomado para este caso.

Ejemplos

considerando el tamaño de rComp = 7 bytes.

Si ftell(fb) informa cero, entonces, ftell(fb) / sizeof(rComp) = 0.

Si ftell(fb) informa 7, entonces, ftell(fb) / sizeof(rComp) = 1.

Si ftell(fb) informa 14, entonces, ftell(fb) / sizeof(rComp) = 2.

Mover el puntero del archivo a una nueva posición

fseek()

int fseek(**FILE** *flujo, **long** Ofs, **int** ptoRef)

También podremos averiguar en qué posición se encuentra el puntero del archivo en un instante dado, esto se logra a través de la función **ftell**, p.e., si estamos en la posición 4, establecido de la siguiente manera, fseek(fb,4 * sizeof(rComp),SEEK_SET); en el cual, la función fseek realiza la acción de mover el puntero del archivo indicado en la dirección establecida como segundo argumento en cantidad de bytes, considerando a partir de una ubicación absoluta en relación al comienzo del archivo, establecida por la constante **SEEK_SET (valor 0)**, si en cambio quisiéramos indicarlo con una ubicación relativa, es decir, desde la posición que se encuentre el puntero actualmente, lo indicamos con la constante, **SEEK_CUR (valor 1)**, y si quisiéramos ubicar el puntero del archivo al final del mismo, se lo indica con la constante, **SEEK_END (valor 2)**; ftell(fb) regresará el valor 28 en cantidad de bytes, ya que, 4 * 7 = 28, siendo 7 el tamaño de rComp, luego aplicando la expresión ftell(fb) / sizeof(rComp) el resultado será que el puntero del archivo fb se encuentra en la posición cuatro.

Si la acción de mover el puntero fue exitosa, entonces, fseek retorna cero, caso contrario, retorna un valor distinto de cero.

Para **determinar el tamaño del archivo en cantidad de componentes**, debemos mover el puntero al final del archivo y dividirlo por el tamaño de la componente, como se indica a continuación:

fseek(fb,0L,SEEK_END)

ftell(fb) / sizeof(rComp)

Si quisiéramos ubicarnos en una **posición anterior del lugar en que nos encontramos** en un momento dado, se establece de la siguiente manera:

fseek(fb,0L,SEEK_END)

fseek(fb,ftell(fb) - sizeof(rComp),SEEK_SET)

Mover el puntero del archivo al comienzo

rewind

rewind(fb)

Para ubicarnos al **comienzo del archivo**, podemos hacerlo de las siguientes formas:

fseek(fb,0L,SEEK_SET)
rewind(**FILE** *flujo)
rewind(fb)

En ambos casos ubica el puntero sobre el primer byte del archivo, o dicho de otra manera en su primer componente, no obstante observamos que la segunda forma es más compacta su escritura.

Ubicar el puntero a una posición n

Para mover el puntero a una posición n, podemos hacerlo de la siguiente forma:

fseek(fb,n * sizeof(rComp),SEEK_SET)

Para ubicarnos al final del archivo, la sentencia será

fseek(fb,0L,SEEK_END)

Ubicar el puntero sobre la última componente grabada

fseek(fb,0L,SEEK_END)

fseek(fb,ftell(fb) – sizeof(rComp),SEEK_SET)

Determinar el tamaño del archivo

Función de usuario determina el tamaño del archivo en cantidad de bytes:

```
// Función que retorna el tamaño del archivo en cantidad de bytes.
long fileSize(FILE *flujo) {
  long curpos,
       length;

  curpos = ftell(flujo);
  fseek(flujo,0L,SEEK_END);
  length = ftell(flujo);
  fseek(flujo,curpos,SEEK_SET);
  return length;
}
```

Utilizando esta función de usuario, para ubicarnos al final del archivo, haríamos ahora:

fseek(fb,fileSize(fb),SEEK_SET);

Y para ubicarnos en la última componente grabada en el archivo, utilizando la función de usuario fileSize, haríamos ahora:

fseek(fb,fileSize(fb) – sizeof(rComp),SEEK_SET)

Operación de lectura en un archivo binario

fread()

size_t fread(**void** *bufer, **size_t** tamaño, **size_t** cantidadComponentes, **FILE** *flujo)

La función fread, lee una componente completa de un archivo binario en donde se encuentra el puntero del archivo en un instante dado, luego de esta operación de lectura, el puntero avanza automáticamente a la próxima posición. Cabe aclarar que si queremos recorrer leyendo secuencialmente debemos, antes de ingresar al proceso cíclico realizar la primer lectura en forma directa, a continuación la condición del ciclo determinará que si no es fin de archivo ingresar al ciclo, para procesar el dato leído y como última acción de este ciclo, realizar las próxima lecturas, también en forma directa. Esta operación de lectura notar que es diferente en el caso del lenguaje Pascal, ya que los mecanismos de lectura funcionar en forma diferente, recordando que en Pascal, todas las lecturas se realizan dentro del ciclo como primer acción y luego se procesa el registro leído. Solo en los casos particulares que requieran lecturas anticipadas, como en los procesos de corte de control y apareo de archivos, que imperiosamente sí requieren lectura anticipada, en Pascal debemos realizar una lectura especial, es decir, un módulo que primero pregunte si no es fin de archivo, en estos casos sí leeremos un registro y a una variable booleana le asignaremos el valor de falso, caso contrario, no leeremos ningún registro más y a esa misma variable booleana le asignaremos el valor de verdadero, la condición del ciclo en vez de preguntar por la función eof() de Pascal, lo reemplazaremos por la variable booleana, como p.e. **not** fdaf. Luego las próximas lecturas se realizarán dentro del ciclo como última acción invocando nuevamente a este módulo de lectura especial. Queda claro entonces, que en el lenguaje C/C++ este módulo especial no es necesario implementarlo, ya que, el mecanismo de lectura funciona de manera diferente al Pascal, debido que naturalmente, leemos en forma directa antes de ingresar al ciclo siempre.

Cabe aclarar que la cantidad de bytes leídos está dada por la siguiente expresión cantidadComponentes * tamaño. Si la operación de lectura fue exitosa, entonces, retorna la cantidad de ítems leídos y no cantidad de bytes. En cambio, si la operación falló, ya sea por un error o por haber encontrado la marca de fin de archivo, entonces, retorna un valor pequeño, posiblemente cero.

Ejemplo

fread(&rComp, sizeof(rComp), 1,fb)

Operación de escritura en un archivo binario

fwrite()

size_t fwrite(**const void** *bufer, **size_t** tamaño, **size_t** cantidadComponentes, **FILE** *flujo)

La función fwrite, graba una componente completa de un archivo binario en donde se encuentra el puntero del archivo en un instante dado, luego de esta operación de escritura, el puntero avanza automáticamente a la próxima posición.

Cabe aclarar que la cantidad de bytes grabados está dada por la siguiente expresión cantidadComponentes * tamaño. Si la operación de lectura fue exitosa, entonces, retorna la cantidad de ítems leídos y no cantidad de bytes. En cambio, si la operación falló, ya sea por un error o por haber encontrado la marca de fin de archivo, entonces, retorna un valor pequeño, posiblemente cero.

Ejemplo

fwrite(&rComp, sizeof(rComp), 1,fb)

Cerrar el archivo

fclose()

int fclose(**FILE** *flujo)

La función fclose, antes de cerrar el archivo vacía el búfer forzando a enviar su contenido al archivo, luego lo cierra. Retorna cero, si la operación fue exitosa, caso contrario, retorna EOF.

Cerrar todos los archivos abiertos

fcloseall()

fcloseall()

Cierra todos los flujos abiertos y retorna un valor positivo que indica la cantidad de flujos cerrados, en cambio sí ocurrió un error retorna **EOF**.

Renombrar un archivo

rename()

int rename(**const char** *ViejoNombre, **const char** *NuevoNombre)

La función rename, renombra el archivo indicado como ViejoNombre, por otro nombre indicado como NuevoNombre, en donde ViejoNombre y NuevoNombre son cadena de caracteres. No se permite el uso de comodines. Si la operación fue correcta, entonces retorna cero, caso contrario, retorna -1L y pone a **errno** a uno de los siguientes valores:

ENOENT	Ningún archivo o directorio.
EACCES	Permiso negado.
ENOTSAM	Ningún dispositivo.

Eliminar un archivo

remove()

int remove(**const char** *NombreArchivo)

Esta macro, elimina el archivo indicado como NombreArchivo, antes de eliminar el archivo, se debe asegurar que el mismo esté cerrado. El NombreArchivo puede incluir la unidad de disco y la ruta en donde se encuentra el archivo. Si la operación fue exitosa, entonces, retorna cero, caso contrario, retorna -1L y pone a **errno** a uno de los siguientes valores:

ENOENT	Ningún archivo o directorio.
EACCES	Permiso negado.

Función de usuario equivalente a la sentencia de Pascal: truncate()

```
#include <dos.h>

struct REGPACK reg;

void truncate(FILE *flujo) {
  fseek(flujo,ftell(flujo),SEEK_SET); // Posición actual en cantidad de bytes.
  reg.r_ax = 0x4000; // Servicio grabar aleatorio.
  reg.r_bx = fileno(flujo); // Número de handle.
  reg.r_cx = 0; // Actualizar el tamaño del archivo.
  reg.r_ds = FP_SEG(fileno(flujo)); // Segmento del buffer de dato.
  reg.r_dx = FP_OFF(fileno(flujo)); // Desplazamiento del buffer de dato.
  intdos(&reg,&reg); // Invoca int 21h.
} // truncate
```

Definimos las siguientes variables puntero a FILE, local al módulo main:

```
FILE *f1 = NULL, *f2 = NULL, *f3 = NULL, *f4 = NULL, *f5 = NULL;
FILE *f6 0 NULL, *f7 = NULL, *f8 = NULL, *f9 = NULL, *f10 = NULL;
```

Invocamos al módulo desde la función main, para abrir estos archivos:

```
Abrir(&f1,&f2,&f3,&f4,&f5,&f6,&f7,&f8,&f9,&f10);
```

Módulo Abrir, cabecera y cuerpo de la función:

```
void Abrir(FILE **f1, FILE **f2, FILE **f3, FILE **f4, FILE **f5,
           FILE **f6, FILE **f7, FILE **f8, FILE **f9, FILE **f10) {
   *f1 = fopen("Arcf1.Dat","rb");   // Abre f1 binario para solo lectura, sino existe, es error.
   *f2 = fopen("Arcf2.Dat","wb"); // Abre f2 binario para solo escritura, si existe, lo borra.
```

```
   *f3 = fopen("Arcf3.Dat","rb+"); // Abre f3 binario para lectura/escritura, error si no existe.
   *f4 = fopen("Arcf4.Dat","wb+"); //Abre f4 binario para escritura/lectura, borra si existe.
   *f5 = fopen("Arcf5.Dat","r+b"); // Abre f5 binario para lectura/escritura, error si no existe.
   *f6 = fopen("Arcf6.Txt","r");    // Abre f6 texto para solo lectura, sino existe, es error.
   *f7 = fopen("Arcf7.Txt","w");  // Abre f7 texto para solo escritura, si existe, lo borra.
   *f8 = fopen("Arc8.Txt","a");    // Abre f8 texto para agregar al final más componentes.
   *f9 = fopen("Arc9.Txt","rt");   // Abre f9 texto para solo lectura, sino existe, es error.
   *f10 = fopen("Arc10.Txt","rt+"); //Abre f10 texto para lectura/escritura, error si no existe.
}
```

Ejemplo de posición del puntero:

```
#include <stdio.h>
#include <conio.h>

/*
  Id.Programa: PosFile.CPP
  Autor......: Lic. Hugo Cuello
  Fecha......: Junio-2013
  Comentario.: Grabar, Leer, Mover puntero de archivo al final, a
          una ubicación i, al inicio, en forma absoluta SEEK_SET,
          en forma relativa SEEK_CUR hacia adelante +n, 0 en la
          misma posición, o -1 una posición anterior; SEEK_END
          mueve al final del archivo. ftell(f) posición actual en
          cantidad de bytes en todos los casos.
          FILE *fopen(NombreArchivo,modoApertura) // Abrir archivo.
          FILE *f // Variable de tipo puntero a archivo.
          modo de apertura:
            r leer.
       w grabar.
       a agregar más componentes en archivo de texto.
            r+ abre para leer y grabar.
       w+ crea para grabar, luego poder leer en archivo de
         texto. También se y puede agregar una t. rt, wt, r+t,w+t.
            Si se agrega b entonces el archivo es binario:
            rb solo lee archivo binario.
            wb solo graba archivo binario.
       ab agrega registros al archivo binario.
       a+b agrega registros y leer archivo binario.
            r+b abre para leer o grabar archivo binario.
            w+b crea para grabar luego leer archivo binario.
            fclose(f) : Cierra un archivo.
            fread(&r,sizeof r,1,f)
            fwrite(&r,sizeof r,1,f)
            fseek(f,n * sizeof r,SEEK_SET | SEEK_CUR | SEEK_END)
            ftell(f)
            rewind(f)
*/
```

```
long fileSize(FILE *stream);

void main() {
  FILE *f;
  int  i;

  clrscr();
  f = fopen("Enteros.Dat","w+b");
  for (i = 0; i < 10; i++)
    fwrite(&i,sizeof i,1,f);
  printf("fileSize: %ld\n",fileSize(f)); //fileSize: 20
  rewind(f); //Mueve ptr.arch.a inicio, equivale a
  fseek(f,0L,SEEK_SET).
  printf("* \n"); //*
  for (i = 0; i < 3; i++) {
    fread(&i,sizeof i,1,f);
    printf("%d \n",i);  //0, 1, 2
  }
  fseek(f,0L,SEEK_SET); //Mueve ptr.arch.a inicio.
  printf("+ \n"); //+
  for (i = 0; i < 3; i++) {
    fread(&i,sizeof i,1,f);
    printf("%d \n",i); //0, 1, 2
  }
  fseek(f,0L,SEEK_SET);
  fread(&i,sizeof i,1,f);
  printf("->%d\n",i);
  fseek(f,0L,SEEK_END); //Mueve ptr.arch.final p/agregar+reg.x ej.
  printf("- \n");  //-
  for (i = 10; i < 15; i++)
    fwrite(&i,sizeof i,1,f);
  fseek(f,5*sizeof i,SEEK_SET);
  fread(&i,sizeof i,1,f);
  printf(":%d \n",i); //:5
  fseek(f,2*sizeof i,SEEK_CUR);
  fread(&i,sizeof i,1,f);
  printf(">%d \n",i); //>8
  fseek(f,ftell(f) - sizeof i,SEEK_SET); //Mueve ptr. pos.actual-1
  fread(&i,sizeof i,1,f);
  printf("<%d \n",i); //<8
  //fseek(f,0L,SEEK_END);
  fseek(f,fileSize(f) - sizeof i,SEEK_SET);
  fread(&i,sizeof i,1,f);
  printf(":) %d\n",i);   //:) 14
  printf(":( %ld \n",ftell(f) / sizeof i);  //:( 15
  printf("fileSize: %ld\n",fileSize(f)); //fileSize: 30
  fseek(f,0L,SEEK_SET);
  for (i = 0; i < 15; i++) {
    fread(&i,sizeof i,1,f);
```

```
    printf("%d ",i);
  }
  fclose(f);
}

long fileSize(FILE *stream) {
   long curpos,
        length;

   curpos = ftell(stream);
   fseek(stream, 0L, SEEK_END);
   length = ftell(stream);
   fseek(stream, curpos, SEEK_SET);
   return length;
}
```

Archivo de cabecera <stdio.h> o <cstdio>

Esta librería de C ejecuta operaciones de Entrada / Salida, las que tambi{en podrán ser ejecutadas en C++ utilizando la librería <cstdio>. Esta librería utiliza los llamados streams para operar con los dispositivos periféricos como ser teclados, impresoras, terminales o con cualquier otro tipo de archivos soportados por el sistema. Los streams son abstracciones para interactuar con ellos de un modo uniforme; todos los streams tienen propiedades similares independientemente de sus características individuales del medio físico con los que están asociados.

Los streams son manejados por la librería <cstdio> como punteros a objetos FILE. Un objeto a FILE identifica únivocamente a un stream, y es utilizado como un parámetro en las operaciones referente a los streams.

Existen tres streams estándar: stdin, stdout y stderr, los cuales son creados automáticamente y abiertod por todos los programas que utilizan la librería.

Propiedades de los stream

Los streams tienen ciertas propiedades que definen que funciones pueden ser utilizadas sobre ellos y como estos pueden tratar las entradas y salidas de datos. Algunas de estas propiedades están definidas en el momentos en que el stream es asociado a un archivo –abierto- utilizando la función fopen.

Acceso para Leer / Grabar -Read/Write-

Indica si el stream tiene acceso para leer o grabar –o ambos- al medio físico al cual está asociado.

Texto/ Binario -*Text / Binary*-

Un archivo de texto es un conjunto de líneas de texto, en donde, cada línea finaliza con un carácter de nueva línea.

Un archivo binario es una secuencia de caracteres grabados o leídos desde el medio físico sin ninguna conversión, teniendo una correspondencia uno a uno con los caracteres leídos o grabados al stream.

Buffer

Un buffer es un bloque de memoria en donde los datos son ubicados antes de iniciar una lectura o grabación con el arvhivo o dispositivo asociado.

Indicadores

Los stream tienen ciertos indicadores internos que indican su estado actual y que afectan a su comportamiento en las operaciones de entrada – salida.

Indicador de Error

Este indicador es puesto cuando un error ha ocurrido en una operación relacionada con el stream. El indicador puede ser verificado con la función ferror, y puede ser reseteado llamando a algunas de las siguientes funciones, **clearerr, freopen o rewind**.

Indicador Fin-De-Archivo

Este indicador cuando es puesto, indica después de la última operación de lectura o escritura encontró la marca de End of File o Fin de Archivo. Este indicador puede ser verificado con la función **feof**, y puede ser reseteado llamando a las siguientes funciones, **clearerr o freopen** o llamando a cualquier función de reposicionamiento, **rewind, fseek y fsetpos**.

Indicador de posición

Es un puntero interno para cada stream el cual apunta al próximo carácter a ser leído o grabado en la próxima operación de I/O. Su valor puede ser obtenido por las funciones **ftell y fgetpos**, y puede ser cambiado utilizando las funciones de reposicionamiento **rewind, fseek y fsetpos**.

Funciones streams

En esta tabla se han dejado los comentarios explicativos de los distintos elementos en su idioma original. Ud. podrá consultar en otros puntos de este mismo documento los comentarios en el idioma nativo de estos mismos elementos. Solo se ha dejado una breve traducción del significado del elemento.

Unidad IV Capítulo III

Unidad IV – Capítulo III - Archivos: STREAM en C++

Stream o flujo de datos externo en C++

Un **stream** o **flujo**, se puede definir como un dispositivo que produce información., el cual siempre está ligado a un dispositivo físico, como ser, pantalla, disco, impresora…, comportándose de manera análoga con total independencia del dispositivo en cuestión.

Todo programa que se ejecute, en forma automática estarán disponibles los siguientes flujos:

stream	comentario
cin	Entrada estándar por teclado, con buffer.
cout	Salida estándar por pantalla, con buffer.
cerr	Salida estándar de mensajes de error, sin buffer.
clog	Salida estándar de mensajes de error, con buffer.

Difference Table:

	cerr	clog
1.	Es un flujo de error estándar sin búfer.	Es un flujo de error estándar almacenado en buffer.
2.	Se utiliza para mostrar el error.	Se utiliza para iniciar sesión.
3.	Se utiliza para mostrar el mensaje inmediatamente.	No puede mostrar el mensaje inmediatamente.
4.	No puede almacenar el mensaje para mostrarlo más tarde.	Puede almacenar el mensaje en el búfer para mostrarlo más tarde.
5.	La "c" en cerr se refiere a "carácter" y "err" significa "error", por lo que cerr significa "error de carácter".	La "c" en clog se refiere a "carácter" y "log" se refiere a "registro", por lo tanto, clog significa "registro de caracteres".
6.	Es menos eficiente que clog porque es una salida sin búfer.	Es más eficiente que cerr porque es una salida almacenada en búfer.
7.	Se prefiere para errores críticos (errores que pueden provocar fallos del sistema).	No se prefiere para errores críticos (errores que pueden causar fallas del sistema).

Indicadores y Manipuladores

Manipuladores definidos en <iomanip.h>

	Manipuladores parametrizados de <iomanip.h>
1	*setbase(base)* base es un valor entre 1 y 36.
2	*setfill(car)*
3	*setprecision(cdig)* cdig es la cantidad de dígitos a mostrar en su parte decimal.
4	setw(ancho) ancho es la cantidad de lugares mínimo de formato.
5	setiosflag(flag) flag es una lista de 15 items a setear, ios::flag usa | para usar más de un flag. Ver lista de flags más abajo en Manipuladores_iomanip
6	resetiosflag(flag) Idem anterior.

Ejemplos de uso manipuladores iomanip

```
cout << setbase(16) << setfill('*') << setprecision(2);
cout << setiosflag(ios::rigth | ios::showpoint);
```

Manipuladores sin parámetros definidos en la clase ios

Los cambios son permanentes hasta que se produzca un nuevo cambio.

Ejemplos de uso de manipuladores sin parámetros

```
cout << hex << 123 << oct << 123 << endl;
cout.fill('*');
cout.width(20);
cout << right << "Hola" << endl;
```

		Manipuladores sin parámetros de la clase ios
1	*dec*	Cambia la base de numeración a decimal.
2	*hex*	Cambia la base de numeración a hexadecimal.
3	*oct*	Cambia la base de numeración a octal.
4	*ws*	Solo para streams de entrada.
5	*ends*	Agrega carácter nulo a una cadena.
6	*flush*	Vacía el buffer.
7	*endl*	Vacía el buffer y además cambia de línea.
8	*fixed*	Muestra reales en notación punto fijo.
9	*scientific*	Muestra reales en notación punto flotante o científica.
10	*boolalpha*	Emite false o true según corresponda en lugar de 0 o 1.
11	*left*	Alínea a izquierda.
12	*right*	Alínea a derecha.
13	*internal*	Inserta relleno internamente en un formato de salida.
14	*showbase*	Muestra 0 o 0x antes del valor en octal o si es hexadecimal.
15	*showpos*	Muestra el signo del valor siempre.
16	*showpoint*	Muestra punto decimal siempre.
17	*uppercase*	Convierte solo a mayúsculas la representación numérica.
18	*skipws*	Datos desde un stream espacios iniciales los ignora.
19	*unitbuf*	Vacía todos los buffers.
20	*nouppercase*	Desactiva uppercase.
21	*noboolalpha*	Desactiva boolalpha.
22	*noshowbase*	Desactiva showbase.
23	*noshowpoint*	Desactiva showpoint.
24	*noshowpos*	Desactiva showpos.
25	*noskipws*	Desactiva skipws.
26	*nounitbuf*	Desactiva unitbuf.

Ejemplos de manipuladores sin parámetros de la clase ios

Codificación de manipuladores sin parámetros de la clase ios

```
/*
   Id.Programa: Manipuladores_sin_parametros.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: julio-2016
   Comentario.: Manipuladores sin parametros
                1. dec
                2. hex
                3. oct
                4. ws
                5. ends
                6. endl
                7. flush
                8. boolalpha
                9. left
               10. right
               11. fixed
               12. internal
               13. scientific
               14. showbase
               15. showpoint
               16. showpos
               17. skipws
               18. unitbuf
               19. uppercase
               20. noboolalpha
               21. noshowbase
               22. noshowpoint
               23. noshowpos
               24. noskipws
               25. nounitbuf
               26. nouppercase
*/

#include <fstream>
using namespace std;

main() {
  bool existe = true;
  char a[21], b[21];

  cout << showbase << hex << 123 << endl;
```

```
  cout.width(10);
  cout << right << "Hola" << '*' << endl << boolalpha << existe << endl;
  cout.width(10);
  cout.fill('*');
  cout << showpos << dec << 123 << endl;
  cout << internal << -123 << endl;
  cout << uppercase << hex << 0x1ba9 << endl;
  cout << nouppercase << 0x1ba2 << endl;
  cin >> skipws >> a >> b;
  cout << a << '*' << endl << b << '*' << endl;
  return 0;
}

/* Salida
0x7b
     Hola*
true
******+123
-123
0X1BA9
0x1ba2
    Hola
        que tal?
Hola
que tal?
*/
```

Funciones miembro definidas en la clase ios

Su forma de uso presenta la siguiente notación

stream.función()

Máscaras de modificadores

Permiten trabajar con grupos de modificadores afines.

```
enum {
  basefield = dec+oct+hex,
  floatfield = scientific+fixed,
  adjustfield = left+right+internal
};
```

Ejemplos con funciones miembro de la clase ios
Codificación de funciones miembro de la clase ios

Indicadores de la clase ios

Se utilizan con las funciones miembro:

- **setf()**
- **unsetf()**
- **resetiosflag()**

y su formato es cout.setf(ios:: flag | ios::flag …)

Ord.	Indicadores (flag) de la clase ios	
1	*boolalpha*	Muestra false o true el reemplazo de 0 o 1.
2	*dec*	Muestra el valor entero en base decimal.
3	*oct*	Muestra el valor entero en base octal.
4	*hex*	Muestra el valor entero en base hexadecimal.
5	*internal*	Rellena con el carácter seleccionado internamente.
6	*left*	Alínea a la izquierda la próxima salida.
7	*right*	Alínea a la derecha la próxima salida.
8	*scientific*	Muestra el valor real en notación de punto flotante.
9	*fixed*	Muestra el valor real en notación de punto fijo.
10	*showbase*	Muestra la base numérica del valor entero.
11	*showpoint*	Muestra el punto decimal siempre.
12	*showpos*	Muestra el signo del número siempre.
13	*skipws*	Saltea o quita los espacios iniciales.
14	*unitbuf*	Vacía todos los buffers.
15	*uppercase*	Convierte a mayúscula base u valor entero.

Algunas funciones miembros <ios.h>

definidas en las clases istream, ostream, iostream

*cout.**width**(num)*
*cout.**width**()*
*cout.**precision**(num)*
*cout.**precision**()*
*cout.**fill**(car)*
*cout.**fill**()*

cout.width(n)

Obtiene (1) streamsize width() const;
Establece (2) streamsize width (streamsize wide);

La primer forma (1) retorna el valor actual del ancho del campo.

La segunda forma (2) también pone un nuevo ancho de campo al stream.

El ancho del campo determina el número mínimo de caracteres a ser escritos en una representación de salida. Si el ancho estándar de la representación es más corta que el ancho del campo, la representación es cubierto con caracteres de relleno en el lugar determinado por el formato de bandera adjustfield(uno de left, right o internal). El carácter de relleno puede ser establecido con la función miembro fill.

El ancho del campo también puede ser modificando utilizando el manipulador parametrizado *setw*().

Retorna: Valor del ancho del campo antes de la llamada.

```
// ancho del campo
#include <fstream>     // cout, left
using namespace std;

int main () {
  cout << 100 << '\n';
  cout.width(10);
  cout << 100 << '\n';
  cout.fill('x');
  cout.width(15);
  cout << left << 100 << '\n';
  return 0;
}
```

Emite

```
100
       100
100xxxxxxxxxxxx
```

cout.precision()

Obtener (1) streamsize precision() const;
Establecer (2) streamsize precision (streamsize prec);

La primer forma (1) retorna el valor actual del campo precisión punto flotante del stream.

La segunda forma (2) también establece el nuevo valor.

La precisión en punto flotante determina el número máximo de dígitos a ser escritos en operaciones de inserción.

La precisión decimal también puede ser modificada utilizando el manipulador parametrizado *setprecision*().

Retorna: La precisión indicada en el stream antes de la llamada.

```
// modify precision
#include <fstream>
using namespace std;

int main () {
  double f = 3.14159;
  cout.unsetf ( ios::floatfield );              // floatfield not set
  cout.precision(5);
  cout << f << '\n';
  cout.precision(10);
  cout << f << '\n';
  cout.setf( ios::fixed,  ios::floatfield ); // floatfield set to fixed
  cout << f << '\n';
  return 0;
}
```

Emite

```
3.1416
3.14159
3.1415900000
```

cout.fill()

Obtener (1)	`char fill() const;`
Establecer (2)	`char fill (char fillch);`

La primer forma (1) retorna el carácter de relleno.

La segunda forma (2) establece el carácter de relleno como el nuevo carácter de relleno y retorna el carácter de relleno usado antes de la llamada.

También puede ser utilizado el manipulador parametrizado *setfill*().

Retorna: El valor del carácter de relleno antes de la llamada.

```
// using the fill character
#include <fstream>
using namesapace std;

int main () {
  char prev;

  cout.width (10);
  cout << 40 << '\n';
  prev = cout.fill ('x');
  cout.width (10);
  cout << 40 << '\n';
  cout.fill(prev);
  return 0;
}
```

Emite

```
        40
```

xxxxxxxx40

Indicadores y Manipuladores de Entrada / Salida

Pueden ser empleados en operaciones de entrada como así también de salida. Los manipuladores pueden tener o no argumentos, los que no tienen están definidos en **<iostream.h>**, mientras que los que sí tienen están definidos en **<iomanip.h>**. La dificultad de los manipuladores contra los indicadores es que no permiten guardar las configuraciones anteriores. Un manipulador solo afecta al stream al que se aplica (cin, cout, cerr, clog). Además un manipulador permanece activo en el flujo correspondiente hasta que se cambia por otro manipulador, la excepción es setw().

Entrada/Salida con formato <iostream>

Se utilizan los siguientes métodos o funciones:

*cout.**setf**()*
*cout.**unsetf**()*
*cout.**flags**()*

Cada stream tienen asociados unos **indicadores** que son constantes de tipo **enum** que se almacena en un tipo long los cuales, los bits pueden activarse o desactivarse, a continuación se presenta dichos indicadores, utilizados con las funciones métodos indicados anteriormente (stream.setf(), stream.unsetf(), stream.flags(), como por ejemplo, cout.setf(ios::x), en donde x son algunos de los indicadores indicados en la siguiente tabla:

Tabla de Indicadores con cout.setf(), cout.unsetf(), cout.flags()

Ejemplos de indicadores

```
cout.setf(ios::left);
cout.setf(ios::fixed | ios::showpos | ios::showbase);
cout.setf(ios::skipws | ios::showpoint | ios::hex);
```

Además existen otros indicadores adicionales (adjustfield, basefield, floatfield) que actúan como combinaciones de las anteriores (máscaras), a saber:

Items	Indicadores compuestos	Combina excluyente (solo una en on)
1	adjustfield	left, rigth, internal
2	basefield	dec, oct, hex
3	floatfield	scientific, fixed

Por defecto todos los indicadores anteriores están desactivados, excepto skipws y dec.

```
Establecer (1)  fmtflags setf (fmtflags fmtfl);
  Máscara (2)  fmtflags setf (fmtflags fmtfl, fmtflags mask);
```

La primer forma de *setf*(1) es generalmente utilizado para poner banderas de formato independiente: *boolalpha*, *showbase*, *showpoint*, *showpos*, *skipws*, *unitbuf* and *uppercase*, which can also be unset directly with member *unsetf*().

La segunda forma de *setf*(2) es generalmente utilizada para poner un valor para una de las banderas indicadas, utilizando uno de los campos de máscara de bit.

Retorna: El formato de bandera selecionada en el stream antes de la llamada.

Ejemplo

```
// modifying flags with setf/unsetf
#include <fstream>
using namespace std;

int main () {
  cout.setf ( ios::hex, ios::basefield );  // set hex as the basefield
  cout.setf ( ios::showbase );             // activate showbase
  cout << 100 << '\n';
  cout.unsetf ( ios::showbase );           // deactivate showbase
  cout << 100 << '\n';
  return 0;
}
```

Emite

```
0x64
64
```

El formato para *flags*() es igual que para *setf*(). La diferencia es que *flags*() previamente inicializa en cero los bits, mientras que, *setf*(), solo modifica la configuración indicada, y mantiene todo lo que no se ha cambiado explícitamente, por lo que es una alternativa más segura.

Ejemplos adicionales

cout.setf(ios::showpos); // mostrar positivo, el signo +.
cout.setf(ios::showpoint | ios::fixed); // establece mostrar punto decimal y punto fijo.
cout << 100.0; // muestra +100.000000

Manipuladores sin argumentos

Se utilizan en stream como ser cout y como argumentos como si fueran expresiones, por ejemplo, cout << boolalpha << a << left << b << endl;

Ejemplos

cout << *right* << fill('*') << setw(20) << "Hola" << ' ' << 123 << endl;

Manipuladores parametrizados de <iomanip>

Tabla Manipuladores de <iomanip>

Ord.	Manipulador	Comentario

1	resetioflags(n)	Permite desactivar los flags (15) de formato de salida.
		enum {
		skipws
		left
		right
		internal
		dec
		oct
		hex
		showbase
		showpoint
		uppercase
		showpos
		scientific
		fixed
		unitbuf
		stdio
		}
		
		Veamos un ejemplo:
		
		`#include <fstream>`
		`#include <iomanip>`
		`using namespace std;`
		
		`main() {`
		`float x = 121.0/3;`
		`int y = 123;`
		
		`cout << "#" <<`

```
setiosflags(ios::left)
        << setw(12) <<
setprecision(4)
        << x << "#" << endl;
   cout << "#"
        << resetiosflags(ios::left
| ios::dec)
        << setiosflags(ios::hex |
            ios::showbase |
ios::right)
        << setw(8) << y << "#"
        << endl;
   return 0;
}
```

La salida tendrá este aspecto:

```
#40.33       #
#    0x7b#
```

2	setbase(n)	Permite cambiar la base de numeración que se usará para la salida. Sólo se admiten tres valores: 8, 10 y 16, es decir, octal, decimal y hexadecimal. Por ejemplo:

```
#include <fstream>
#include <iomanip>
using namespace std;

main() {
   int x = 123;

   cout << "#" << setbase(8) << x
        << "#" << setbase(10) << x
        << "#" << setbase(16) << x
        << "#" << endl;
   return 0;
}
```

La salida tendrá este aspecto:

```
#173#123#7b#
```

3	setfill(car)	Permite especificar el carácter de relleno cuando la anchura especificada sea mayor

		de la necesaria para mostrar la salida. Por ejemplo:
		`#include <fstream>`
		`#include <iomanip>`
		`using namespace std;`
		
		`main() {`
		`  int x = 123;`
		
		`  cout << "#" << setw(8) << setfill('0')`
		`  << x << "#" << endl;`
		`  cout << "#" << setw(8) << setfill('%')`
		`  << x << "#" << endl;`
		`  return 0;`
		`}`
		La salida tendrá este aspecto:
		`#00000123#`
		`#%%%%%123#`

4	setprecision(n)	Permite especificar el número de dígitos significativos que se muestran cuando se imprimen números en punto flotante: float o double. Por ejemplo:
		`#include <fstream>`
		`#include <iomanip>`
		`using namespace std;`
		
		`main() {`
		`  float x = 121.0/3;`
		
		`  cout << "#" << setprecision(3)`
		`  << x << "#" << endl;`
		`  cout << "#" << setprecision(1)`
		`  << x << "#" << endl;`
		`  return 0;`
		`}`
		La salida tendrá este aspecto:
		`#40.3#`

<table>
<tr><td></td><td></td><td>

```
#4e+01#
```

</td></tr>
<tr><td>**5**</td><td>setw(n)</td><td>

Permite cambiar la anchura en caracteres de la siguiente salida de cout. Por ejemplo:

```
#include <fstream>
#include <iomanip>
using namespace std;

main() {
   int x = 123, y = 432;

   cout << "#" << setw(6) << x <<
"#"
        << setw(12) << y << "#" <<
endl;
   return 0;
}
```

La salida tendrá este aspecto:

```
#   123#         432#
```

</td></tr>
<tr><td>**6**</td><td>setiosflag(n)</td><td>

Permiten activar o desactivar, respectivamente, los flags de formato de salida. Existen quince flags de formato a los que se puede acceder mediante un enum definido en la clase ios:

<table>
<tr><th>flag</th><th>Acción</th></tr>
<tr><td>skipws</td><td>ignora espacios en operaciones de lectura</td></tr>
<tr><td>left</td><td>ajusta la salida a la izquierda</td></tr>
<tr><td>right</td><td>ajusta la salida a la derecha</td></tr>
<tr><td>internal</td><td>deja hueco después del signo o el indicador de base</td></tr>
<tr><td>dec</td><td>conversión a decimal</td></tr>
<tr><td>oct</td><td>conversión a octal</td></tr>
<tr><td>hex</td><td>conversión a hexadecimal</td></tr>
<tr><td>showbase</td><td>muestra el indicador de base en la salida</td></tr>
<tr><td>showpoint</td><td>muestra el punto decimal en salidas en punto flotante</td></tr>
<tr><td>uppercase</td><td>muestra las salidas</td></tr>
</table>

</td></tr>
</table>

	hexadecimales en mayúsculas
showpos	muestra el signo '+' en enteros positivos
scientific	muestra los números en punto flotante en notación exponencial
fixed	usa el punto decimal fijo para números en punto flotante
unitbuf	vacía todos los buffers después de una inserción
stdio	vacía los buffers stdout y stderr después de una inserción

Veamos un ejemplo:

```
#include <fstream>
#include <iomanip>
using namespace std;

main() {
   float x = 121.0/3;
   int y = 123;

   cout << "#" <<
setiosflags(ios::left)
        << setw(12) <<
setprecision(4)
        << x << "#" << endl;
   cout << "#"
        << resetiosflags(ios::left
| ios::dec)
        << setiosflags(ios::hex |
            ios::showbase |
ios::right)
        << setw(8) << y << "#"
        << endl;
   return 0;
}
```

La salida tendrá este aspecto:

```
#40.33       #
#    0x7b#
```

Ejemplos

cout << setprecision(2);
cout.setf(ios::fixed); // **equivalente a** cout << fixed;
cout << setfill('*') << setw(10) << "Hola " << setw(12) << 123.3456 << endl;

Codificaciòn de Manipuladores en <iomanip.n>

```
/*
   Id.Programa: Manipuladores_iomanip.cpp
   Autor..........: Lic. Hugo Cuello
   Fecha…........: julio-2016
   Comentario.: Manipuladores
            1. setw          Permite cambiar la anchura en caracteres de la
                             siguiente salida de cout.
            2. setprecision  Permite especificar el número de dígitos
                             significativos que se muestran cuando se imprimen
                             números en punto flotante: float o double.
            3. setfill       Permite especificar el carácter de relleno cuando
                             la anchura especificada sea mayor de la necesaria
                             para mostrar la salida.
            4. setbase       Permite cambiar la base de numeración que se usará
                             para la salida. Sólo se admiten tres valores:
                             8, 10 y 16, es decir, octal, decimal y hexadecimal.
            5. setiosflags   Permiten activar o desactivar, respectivamente, los
                             flags de formato de salida. Existen quince flags
                             de formato a los que se puede acceder mediante
                             un enum definido en la clase ios.
            6. resetiosflags Idem anterior.
             Flags utilizados como argumentos en setiosflags y resetiosflags
   Flag          Acción
   skipws        ignora espacios en operaciones de lectura
   left          ajusta la salida a la izquierda
   right         ajusta la salida a la derecha
   internal      deja hueco después del signo o el indicador de base
   dec           conversión a decimal
   oct           conversión a octal
   hex           conversión a hexadecimal
   showbase      muestra el indicador de base en la salida
   showpoint     muestra el punto decimal en salidas en punto flotante
   uppercase     muestra las salidas hexadecimales en mayúsculas
   showpos       muestra el signo '+' en enteros positivos
   scientific    muestra los números en punto flotante en notación exponencial
   fixed         usa el punto decimal fijo para números en punto flotante
   unitbuf       vacía todos los buffers después de una inserción
```

```
   stdio          vacía los buffers stdout y stderr después de una inserción
*/

#include <fstream>
#include <iomanip>
using namespace std;

main() {
  float x = 121.0/3;
  int y = 123;

  cout << "#" << setiosflags(ios::left)
       << setw(12) << setprecision(4) << setfill('*') << setbase(16)
       << x << "#" << endl;
  cout << "#" << 123
       << resetiosflags(ios::left | ios::dec)
       << setiosflags(ios::hex | ios::showbase | ios::right)
       << setw(8) << y << "#"
       << endl;
   cout << y;
   return 0;
}

/* Salida
#40.33*******#
#7b****0x7b#
0x7b
*/
```

Funciones miembro booleanas

- ✓ **good()**
- ✓ **eof()**
- ✓ **fail()**
- ✓ **bad()**
- ✓ **rdstate()**

Tabla de estado de I-O

iostate value (member constants)	indicates	functions to check state flags				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	true	false	false	false	goodbit
eofbit	End-of-File reached on input operation	false	true	false	false	eofbit
failbit	Logical error on i/o operation	false	false	true	false	failbit
badbit	Read/writing error on i/o operation	false	false	true	true	badbit

Ejemplos

cout << hex << 100 << endl; // emite: 64 valor en base 16.
cout << setfill << '*' << setw(10) << 2343.0; // emite: ??????2343

Funciones útiles: good(), fail(), eof(), bad(), rdstate() o clear()

Hay varios flags de estado que podemos usar para comprobar el estado en que se encuentra un stream.

Concretamente nos puede interesar si hemos alcanzado el fin de fichero, o si el stream con el que estamos trabajando está en un estado de error.

La función principal para esto es *good()*, de la clase *ios*.

Después de ciertas operaciones con streams, a menudo no es mala idea comprobar el estado en que ha quedado el stream. Hay que tener en cuenta que ciertos estados de error impiden que se puedan seguir realizando operaciones de entrada y salida.

En el ejemplo de archivos de acceso aleatorio hemos usado *clear()* para eliminar el bit de estado *eofbit* del fichero de entrada, si no hacemos eso, las siguientes operaciones de lectura fallarían.

Otra condición que conviene verificar es la existencia de un fichero. En los ejemplos anteriores no ha sido necesario, aunque hubiera sido conveniente, verificar la existencia, ya que el propio ejemplo crea el fichero que después lee.

Cuando vayamos a leer un fichero que no podamos estar seguros de que existe, o que aunque exista pueda estar abierto por otro programa, debemos asegurarnos de que nuestro programa tiene acceso al stream. Por ejemplo:

```
#include <fstream>
using namespace std;

int main() {
   char mes[20];
   ifstream fich("meses1.dat", ios::in | ios::binary);

   // El fichero meses1.dat no existe, este programa es
   // una prueba de los bits de estado.

   if(fich.good()) {
      fich.read(mes, 20);
      cout << mes << endl;
   }
```

```
   else {
      cout << "Fichero no disponible" << endl;
      if(fich.fail()) cout << "Bit fail activo" << endl;
      if(fich.eof())  cout << "Bit eof activo" << endl;
      if(fich.bad())  cout << "Bit bad activo" << endl;
   }
   fich.close();

   return 0;
}
```

Ejemplo de fichero previamente abierto

```
#include <fstream>
using namespace std;

int main() {
   char mes[20];
   ofstream fich1("meses.dat", ios::out | ios::binary);
   ifstream fich("meses.dat", ios::in | ios::binary);

   // El fichero meses.dat existe, pero este programa
   // intenta abrir dos streams al mismo fichero, uno en
   // escritura y otro en lectura. Eso no es posible, se
   // trata de una prueba de los bits de estado.

   fich.read(mes, 20);
   if(fich.good())
      cout << mes << endl;
   else {
      cout << "Error al leer de Fichero" << endl;
      if(fich.fail()) cout << "Bit fail activo" << endl;
      if(fich.eof())  cout << "Bit eof activo" << endl;
      if(fich.bad())  cout << "Bit bad activo" << endl;
   }
   fich.close();
   fich1.close();

   return 0;
}
```

Entradas y Salidas de archivos stream

Las operaciones de acceso a archivos o ficheros requiere incluir el archivo de cabecera <fstream.hpp> o bien <fstream>. En este archivo están incluidas las siguientes librerías el cual permitirá la utilización de los archivos, a saber:

- ✓ **ifstream**
- ✓ **ofstream**
- ✓ **fstream**

las cuales derivan de la clase ios, que fuera representado en la tabla indicada más arriba. Los pasos a seguir para poder empezar a utilizar archivos se indican a continuación:

1. **Incluir el archivo de cabecera** <fstream> o <fstream.h>
2. **Abrir el o los archivos necesarios (**ifstream, ofstream, fstream**)**

3. **Operar con el o los archivos para leer o grabar (**read, write**)**
4. **Cerrar el o los archivos cuando ya no se requiera utiliazarlo(s) (**close**)**

Funciones miembro para abrir archivo

Nomenclatura utilizada

NF: Nombre Físico del archivo. Es el rótulo que aparece en el directorio del disco. Es una cadena de caracteres (char*)
NL: Nombre Lógico del archivo. Es un stream representado por una variable.
MA: Modo de Apertura del archivo. Hay 6 modos los cuales son:

- ✓ **ios::in**
- ✓ **ios::out**
- ✓ **ios::app**
- ✓ **ios::ate**
- ✓ **ios::trunc**
- ✓ **ios::binary**

Apertura del archivo

A diferencia de los stream estándar cin, cout, cerr, los cuales no requieren que se los abra en forma explícita, debido a que todos estos stream son abiertos en forma automática y disponibles al usuario para su utilización en la operaciones de lectura y/o escritura y tampoco es necesario cerrarlos ya que son cerrados automáticamente; los archivos en disco serán necesarios realizar en forma explícita estas acciones de apertura y cierre. Para abrir un stream existen las siguientes alternativas:

1. ifstream **strL(NF, MA) ;** ofstream **strG(NF, MA);** fstream **strLG(NF, MA);**
2. ifstream **stream(NF);** ofstream **strG(NF);** fstream **strLG(NF);**
3. ifstream **stream;** ofstream **strG;** fstream **strLG;**

Primero aclaremos que las diferencias de los nombres de los identificadores de clases corresponde a los siguientes casos:

- ✓ ifstream **Abre un stream para entrada (input), es decir, para leer.**
- ✓ ofstream **Abre un stream para salida (outup), es decir, para grabar.**
- ✓ fstream **Abre un stream para entrada/salida (i-o), es decir, para leer/grabar.**

El primer caso emplea dos argumentos, el primero indica el nombre físico del archivo, el cual es una cadena de caracteres (char*) el que podrá incluir además del nombre del archivo, su ruta completa, formada por la unidad de disco y sus directorios por los cuales atraviesa su camino. El segundo argumento indica el modo de apertura como fuera indicado anteriormente, el cual puede presentar más de un modo separado por la barra vertical (|) el cual indica el ‘o’ a nivel de bit.

El segundo caso emplea un único argumento que es el nombre físico del archivo, mientras que el modo de apertura se da por sobreentendido ya que la clase ifstream

establece que el modo de apertura es de entrada o input indicado por la primer letra del identificador ifstream.

El tercer caso no se emplea ningún argumento, en este caso se deberá abrir el archivo por medio de la función método 'open', aplicado al objeto stream, el cual presenta dos argumentos, el primero el nombre físico del archivo y el segundo el modo de apertura.

La sintaxis de apertura de archivo de Entrada es

Omisión (1)	**ifstream**();
Inicialización (2)	explicit **ifstream** (const char* filename, ios_base::openmode mode = ios_base::in);

Apertura de archivos en modo Entrada

Ejemplos

```
ifstream strL ("Alumnos.txt", ios::in);
ifstream strL2 ("Alumnos2.txt");
ifstream strL3;
   strL3.open("Alumnos3.Dat",ios::in | ios::binary);
```

Se espera que los archivos abiertos en modo de entrada existan, ya que de no existir el proceso no hará lo solicitado por no poder leer datos al no existir el archivo. En caso de existir el puntero se ubica en la primer componente del archivo, es decir la componente cero, aún así siempre el proceso interno se referirá a un byte y no a una componente, por lo que la posición del puntero se ubicará en el byte cero.

Los dos primeros ejemplos abren archivos de texto.

El tercer ejemplo abre un archivo binario. Además como al instante de crear el stream strL3 no se lo asoció a un nombre físico, se lo debe realizar como se establece en la siguiente línea el cual emplea el método open que va precedido por el stream y el punto que separa el objeto stream con el método.

```
// ifstream constructor.
#include <fstream>      // std::ifstream

main () {

  ifstream ifs ("test.txt", ifstream::in);

  char c = ifs.get();

  while (ifs.good()) {
    cout << c;
    c = ifs.get();
  }
```

```
 ifs.close();

 return 0;
}
```

Los mismos ejemplos pueden ser aplicados a las aperturas de stream en modo salidas y al modo combinado de entrada/salida.

La sintaxis de apertura de archivo de Salida es

Omisión (1)	`ofstream();`
Inicialización (2)	`explicit ofstream (const char* filename, ios_base::openmode mode = ios_base::out);`

Apertura de archivo en modo Salida

```
ofstream strL ("Alumnos.txt", ios::out);
ofstream strL2 ("Alumnos2.txt");
ofstream strL3;
  strL3.open("Alumnos3.txt",ios::out | ios::binary);
```

Si los archivos abiertos en modo de salida no existieran, no será error y el archivo es creado desde cero, sin embargo, en caso de que sí existieran, éstos se destruirán, perdiéndose todos los datos allí almacenados y a continuación los archivos se crearán desde cero. El puntero en estos casos se ubica sobre la marca de finde archivo, eof.

```
// ofstream constructor.
#include <fstream>      // std::ofstream

int main () {

  std::ofstream ofs ("test.txt", std::ofstream::out);

  ofs << "lorem ipsum";

  ofs.close();

  return 0;
}
```

La sintaxis de apertura de archivo de Entrada-Salida es

Omisión (1)	`fstream();`
Inicialización (2)	`explicit fstream (const char* filename,ios_base::openmode mode = ios_base::in | ios_base::out);`

Apertura de archivos en modo Entrada-Salida

```
fstream strL ("Alumnos.txt",  ios::in | ios::out);
```

```
fstream strL2 ("Alumnos2.txt");
fstream strL3;
   strL3.open("Alumnos3.txt", ios:: in | ios::out | ios::binary);
```

Los archivos deben existir al momento de su apertura, ya que si no existieran, ocasionaría un error. Si la apertura fue existosa entonces el puntero se ubica en su byte de posición cero, como fuera indicado anteriormente en los casos previos.

```
// fstream constructor.
#include <fstream>      // std::fstream

int main () {
```

```
  std::fstream fs ("test.txt", std::fstream::in |
std::fstream::out);

  // i/o operations here

  fs.close();

  return 0;
}
```

Operaciones de lectura en archivos de texto –incluye dispositivo por omisión-

Se trata de un objeto global definido en "iostream.h".

En ejemplos anteriores ya hemos usado el operador >>.

El operador >>

Ya conocemos el operador >> lo hemos usado para capturar variables.

```
istream &operator>>(int&)
```

Este operador está sobrecargado en cin para los tipos estándar: int&, short&, long&, double&, float&, char& y char*.

Además, el operador << devuelve una referencia objeto ostream, de modo que puede asociarse. Estas asociaciones se evalúan de izquierda a derecha, y permiten expresiones como:

```
cin >> var1 >> var2;
cin >> variable;
```

Cuando se usa el operador >> para leer cadenas, la lectura se interrumpe al encontrar un carácter '\0', ' ' o '\n'.

Hay que tener cuidado, ya que existe un problema cuando se usa el operador >> para leer cadenas: cin no comprueba el desbordamiento del espacio disponible para el

almacenamiento de la cadena, del mismo modo que la función gets tampoco lo hace. De modo que resulta poco seguro usar el operador >> para leer cadenas.

Por ejemplo, declaramos:

```
char cadena[10];
cin >> cadena;
```

Si el usuario introduce más de diez caracteres, los caracteres después del décimo se almacenarán en una zona de memoria reservada para otras variables o funciones.

Existe un mecanismo para evitar este problema, consiste en formatear la entrada para limitar el número de caracteres a leer:

```
char cadena[10];
cin.width(sizeof(cadena));
cin >> cadena;
```

De este modo, aunque el usuario introduzca una cadena de más de diez caracteres sólo se leerán diez.

Funciones interesantes de cin

Hay que tener en cuenta que cin es un objeto de la clase "istream", que a su vez está derivada de la clase "ios", así que heredará todas las funciones y operadores de ambas clases. Se mostrarán todas esas funciones con más detalle en la documentación de las bibliotecas, pero veremos ahora las que se usan más frecuentemente.

Formatear la entrada

El formato de las entradas de cin, al igual que sucede con cout, se puede modificar mediante flags. Estos flags pueden leerse o modificarse mediante las funciones flags, setf y unsetf.

Otro medio es usar manipuladores, que son funciones especiales que sirven para cambiar la apariencia de una operación de salida o entrada de un stream. Su efecto sólo es válido para una operación de entrada o salida. Además devuelven una referencia al stream, con lo que pueden ser insertados en una cadena entradas o salidas.

Por el contrario, modificar los flags tiene un efecto permanente, el formato de salida se modifica hasta que se restaure o se modifique el estado del flag.

Funciones manipuladoras con parámetros

Para usar estos manipuladores es necesario incluir el fichero de cabecera iomanip.

Existen cuatro de estas funciones manipuladoras aplicables a cin: setw, setbase, **setiosflags y resetiosflags.**

Todas trabajan del mismo modo, y afectan sólo a la siguiente entrada o salida.

En el caso de cin, no todas las funciones manipuladoras tienen sentido, y algunas trabajan de un modo algo diferentes que con streams de salida.

Manipulador setw

Permite establecer el número de caracteres que se leerán en la siguiente entrada desde cin. Por ejemplo:

```
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
   char cad[10];

   cout << "Cadena:"
   cin >> setw(10) >> cad;

   cout << cad << endl
   return 0;
}
```

La salida tendrá este aspecto.

```
Cadena: 1234567890123456
123456789
```

Hay que tener en cuenta que el resto de los caracteres no leídos por sobrepasar los diez caracteres, se quedan en el buffer de entrada de cin, y serán leídos en la siguiente operación de entrada que se haga. Ya veremos algo más abajo cómo evitar eso, cuando veamos la función "ignore". El manipulador setw no tiene efecto cuando se leen números.

```
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
   int x;

   cout << "Entero:"
   cin >> setw(3) >> x

   cout << x << endl
   return 0;
}
```

La salida tendrá este aspecto, por ejemplo

```
Entero: 1234567
```

```
1234567
```

Manipulador setbase

Permite cambiar la base de numeración que se usará para la entrada de números enteros. Sólo se admiten tres valores: 8, 10 y 16, es decir, octal, decimal y hexadecimal.

```
#include <fstream>
#include <iomanip>
using namespace std;
int main() {
   int x;
   cout << "Entero: ";
   cin >> setbase(16) >> x;
   cout << "Decimal: " << x << endl;
   return 0;
}
```

La salida tendrá este aspecto

```
Entero: fed4

Decimal: 65236
```

Manipuladores setiosflags y resetiosflags

Permiten activar o desactivar, respectivamente, los flags de formato de entrada. Existen quince flags de formato a los que se puede acceder mediante un enum definido en la clase ios:

	flag	Acción
1	skipws	ignora espacios en operaciones de lectura
2	left	ajusta la salida a la izquierda
3	right	ajusta la salida a la derecha
4	internal	deja hueco después del signo o el indicador de base
5	dec	conversión a decimal
6	oct	conversión a octal
7	hex	conversión a hexadecimal
8	showbase	muestra el indicador de base en la salida
9	showpoint	muestra el punto decimal en salidas en punto flotante
10	uppercase	muestra las salidas hexadecimales en mayúsculas
11	showpos	muestra el signo '+' en enteros positivos
12	scientific	muestra los números en punto flotante en notación exponencial
13	fixed	usa el punto decimal fijo para números en punto flotante
14	unitbuf	vacía todos los buffers después de una inserción
15	stdio	vacía los buffers stdout y stderr después de una inserción

De los flags de formato listados, sólo tienen sentido en cin los siguientes: skipws, dec, oct y hex.

```
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
   char cad[10];

   cout << "Cadena: ";
   cin >> setiosflags(ios::skipws) >> cad;
   cout << "Cadena: " << cad << endl;

   return 0;
}
```

La salida tendrá este aspecto

```
Cadena:         prueba
Cadena: prueba
```

Manipuladores sin parámetros

Existen otro tipo de manipuladores que no requieren parámetros, y que ofrecen prácticamente la misma funcionalidad que los anteriores. La diferencia es que los cambios son permanentes, es decir, no sólo afectan a la siguiente entrada, sino a todas las entradas hasta que se vuelva a modificar el formato afectado.

Manipuladores dec, hex y oct

```
inline ios& dec(ios& i)
inline ios& hex(ios& i)
inline ios& oct(ios& i)
```

Permite cambiar la base de numeración de las entradas de enteros:

Función	Acción
dec	Cambia la base de numeración a decimal
hex	Cambia la base de numeración a hexadecimal
oct	Cambia la base de numeración a octal

El cambio persiste hasta un nuevo cambio de base.

Ejemplo:

```
#include <fstream>
using namespace std;

int main() {
   int x, y, z;

   cout << "Entero decimal (x y z): ";
   cin >> dec >> x >> y >> z;
   cout << "Enteros: " << x << ", "
        << y << ", " << z << endl;
   cout << "Entero octal (x y z): ";
   cin >> oct >> x >> y >> z;
   cout << "Enteros: " << x << ", "
        << y << ", " << z << endl;
   cout << "Entero hexadecimal (x y z): ";
   cin >> hex >> x >> y >> z;
   cout << "Enteros: " << x << ", "
        << y << ", " << z << endl;

   return 0;
}
```

La salida tendrá este aspecto

```
Entero decimal (x y z): 10 45 25
Enteros: 10, 45, 25
Entero octal (x y z): 74 12 35
Enteros: 60, 10, 29
Entero hexadecimal (x y z): de f5 ff
Enteros: 222, 245, 255
```

Función ws

```
extern istream& ws(istream& ins);
```

Ignora los espacios iniciales en una entrada de cadena. Ejemplo

```
#include <fstream>
using namespace std;

int main() {
   char cad[10];

   cout << "Cadena: ";
   cin >> ws >> cad;
   cout << "Cadena: " << cad << endl;
```

```
   return 0;
}
```

La salida tendrá éste aspecto:

```
Cadena:      hola
Cadena: hola
```

Función width()

Cambia la anchura en caracteres de la siguiente entrada de stream:

```
int width();
int width(int);
```

La primera forma devuelve el valor de la anchura actual, la segunda permite cambiar la anchura para la siguiente entrada, y también devuelve el valor actual de la anchura. Esta función no tiene efecto con variables que no sean de tipo cadena.

```
  char cadena[10];

  cin.width(sizeof(cadena));
  cin >> cadena;
```

Función setf()

Permite modificar los flags de manipulación de formato:

```
long setf(long);
long setf(long valor, long mascara);
```

La primera forma activa los flags que estén activos tanto en el parámetro y deja sin cambios el resto.

La segunda forma activa los flags que estén activos tanto en valor como en máscara y desactiva los que estén activos en mask, pero no en valor. Podemos considerar que mask contiene activos los flags que queremos modificar y valor los flags que queremos activar.

Ambos devuelven el valor previo de los flags.

```
  int x;

  cin.setf(ios::oct, ios::dec | ios::oct | ios::hex);
  cin >> x;
```

Función unsetf()

Permite eliminar flags de manipulación de formato:

```
void unsetf(long mascara);
```

Desactiva los flags que estén activos en el parámetro.

Nota: en algunos compiladores he comprobado que esta función tiene como valor de retorno el valor previo de los flags.

```
int x;

cin.unsetf(ios::dec | ios::oct | ios::hex);
cin.setf(ios::hex);
  cin >> x;
```

Función flags()

Permite cambiar o leer los flags de manipulación de formato:

```
long flags () const;
long flags (long valor);
```

La primera forma devuelve el valor actual de los flags.

La segunda cambia el valor actual por valor, el valor de retorno es el valor previo de los flags.

```
int x;
long f;

f = flags();
f &= !(ios::hex | ios::oct | ios::dec);
f |= ios::dec;
cin.flags(f);
cin >> x;
```

Función get

La función get() tiene tres formatos:

```
int get();
istream& get(char& c);
istream& get(char* ptr, int len, char delim = '\n');
```

Sin parámetros, lee un carácter, y lo devuelve como valor de retorno:

Nota: Esta forma de la función get() se considera obsoleta.

Con un parámetro, lee un carácter:

En este formato, la función puede asociarse, ya que el valor de retorno es una referencia a un stream. Por ejemplo:

```
char a, b, c;

cin.get(a).get(b).get(c);
```

Con tres parámetros: lee una cadena de caracteres:

En este formato la función get lee caracteres hasta un máximo de 'len' caracteres o hasta que se encuentre el carácter delimitador.

```
char cadena[20];

cin.get(cadena, 20, '#');
```

Función getline

Funciona exactamente igual que la versión con tres parámetros de la función get(), salvo que el carácter delimitador también se lee, en la función get() no.

```
istream& getline(char* ptr, int len, char delim = '\n');
```

Función read

Lee n caracteres desde el cin y los almacena a partir de la dirección ptr.

```
istream& read(char* ptr, int n);
```

Función ignore

Ignora los caracteres que aún están pendientes de ser leídos:

```
istream& ignore(int n=1, int delim = EOF);
```

Esta función es útil para eliminar los caracteres sobrantes después de hacer una lectura con el operador >>, get o getline; cuando leemos con una achura determinada y no nos interesa el resto de los caracteres introducidos. Por ejemplo:

```
#include <fstream>
#include <iomanip>
using namespace std;

int main() {
  char cad[10];
  int i;

  cout << "Cadena: ";
  cin >> setw(10) >> cad;
  cout << "Entero: ";
```

```
  cin.ignore(100, '\n') >> i;
  cout << "Cadena: " << cad << endl;
  cout << "Entero: " << i << endl;

  cin.get();
  return 0;
}
```

La salida podría tener este aspecto:

```
Cadena: cadenademasiadolarga
Entero: 123
Cadena: cadenadem
Entero: 123
```

Función peek()

Esta función obtiene el siguiente carácter del buffer de entrada, pero no lo retira, lo deja donde está.

```
int peek();
```

Función putback()

Coloca un carácter en el buffer de entrada:

```
istream& putback(char);
```

Función get()

single character (1)	int get(); istream& get (char& c);
c-string (2)	istream& get (char* s, streamsize n); istream& get (char* s, streamsize n, char delim);
stream buffer (3)	istream& get (streambuf& sb); istream& get (streambuf& sb, char delim);

1) Un caracter

Extrae un carácter desde el stream.

El carácter es retornado primer caso, o puesto como valor en su argumento segundo caso.

(2) c-string

Extrae caracteres desde el stream y lo almacena en s como c-string, hasta n-1 caracteres que han sido extraidos o delimitado por el carácter encontrado por el carácter de nueva línea '\n' o limitado si se indico en su argumento.

El carácter null '\0' es automáticamente agregado para la secuencia si n es mayor a cero, aún si el string es vacio.

(3) stream buffer

Extrae caracteres desde el stream y lo inserta en la secuencia de salida controlado por el objeto sb buffer del stream, deteniéndose tan pronto como falle la inserción o tan pronto como el carácter delimitador es encontrado en la secuencia de entrada.

```
// istream::get ejemplo
#include <fstream>

main () {
  char str[256];

  cout << "Ingrese nombre de un archivo de texto existente: ";
  cin.get (str,256);    // obtiene c-string

  ifstream is(str);     // abre el archivo

  char c;
  while (is.get(c))
    cout << c;

  is.close();           // cierra el archivo

  return 0;
}
```

istream& **getline** (char* s, streamsize n);
istream& **getline** (char* s, streamsize n, char delim);

```
// istream::getline ejemplo
#include <fstream>     // std::cin, std::cout

int main () {
  char name[256], title[256];

  cout << "Por favor, ingrese su nombre: ";
  cin.getline (name,256);

  cout << "Por favor, ingrese su pelicula favorita: ";
  cin.getline (title,256);

  cout << name << "'su pelicula favorita es " << title;
  return 0;
}
```

Archivos Binarios en C++

#include <fstream>

Modos de Apertura de archivos binarios

```
ifstream f;   // Solo lectura.
f.open("nomFis.Dat",ios::binary);
o
ifstream f("nomFis.Dat",ios::binary);  // constructor
```

constructor si existe ubica el ptr al inicio de la componente; en caso de no existir el proceso finalizará en forma incorrecta.

```
ofstream g;  // Solo escritura.
g.open("nomFis.Dat",ios::binary);
o
ofstream g("nomFis.Dat", ios::binary); // constructor
```

si existe lo elimina se pierden todos los datos y lo crea desde 0, si no existe lo crea desde cero. **Existe una variante para no destruir el archivo previo, utilizando el modificador ios::in esto hará que al abrirlo NO lo elimine**, es decir lo mantiene y ubica el ptr al inicio, si grabamos sobre el pisaremos los datos previos por los nuevos datos, si no queremos pisar esa componente podemos mover el ptr sobre otra componente y grabar sobre esta otra componente, incluso mover el ptr al final del archivo. Otra variante es incluir los modificadores ios::in | ios::trunc su comportamiento es un tanto extraño, p.e., si el archivo tuviera 5 componentes de datos y movemos el ptr a la componente 2 (que en realidad es la componente 3) y grabamos nuevos datos, estos quedarán es esa posición, pero las componentes anteriores son inicializadas y las componentes posteriores serán eliminadas.

```
fstream h;   // Leer / Grabar.
h.open("nomFis.Dat", ios::binary | ios::in | ios::out);
o
fstream h("nomFis.Dat", ios::binary | ios::in | ios::out);
```

Ubica el puntero (^) al inicio, si no existe lo crea.
La barra vertical(|) se lee como o a nivel de bit y no como o lógico o boolean.

Modificadores de apertura de archivos binarios

ios::in	obligatorio utilizar con fstream, con ifstream es optativo.
ios::out	obligatorio utilizar con fstream, con ofstream es optativo.
ios::app	en **ofstream** conserva los datos grabados ubica el ptr al final. app es append
ios::ate	en ofstream conserva los datos grabados ubica el ptr al final. (*) ate es at end
ios::binary	en ifstream ofstream y fstream considera archivo binario.
ios::trunc	en fstream elimina todos los datos grabados. trunc es truncar

(*) La diferencia de ios::ate con ios::app es que si con ate movemos el puntero sobre una componente ya grabada, y queremos grabar, entonces, graba pisando la componente, perdiendo los datos previamente grabados en esa posición por los nuevos datos.
En cambio, si utilizamos ios::app si movemos el puntero sobre una componente previamante grabada y pretendemos grabar nuevos datos en esa componente, el proceso no nos va a dejar grabar ahi, sino que mueve el puntero nuevamente al final del archivo, para grabar los nuevos datos al final.

Operaciones de lectura y grabación

Leer en archivo binario

```
f.read((char *) &r, sizeof r);
```

lee donde está el puntero **toda la componente**, luego avanza el puntero a la próxima componente. La función *read* retorna un valor mayor a cero, en caso de haber leído datos, y ese valor indica la cantidad de bytes o caracteres leídos, por otro lado, si el puntero se encuentra sobre la marca de fin de archivo, al intentar leer, ningún dato es leído, la función *read* retorna cero.

Proceso de lectura secuencial a todas sus componentes

Primera forma, siempre es posible de utilizarla

```
f.read((char *) &r, sizeof r);
while(!f.eof()) {
   ProcReg(r);
   f.read((char *) &r, sizeof r);
}
```

Segunda forma, no siempre es posible utilizarla.

```
 while (f.read((char *) &r, sizeof r))
   ProcReg(r);
```

Cuando sea posible utilizar esta segunda forma, ya que, es más simple que la primera forma, en consecuencia la utilizaremos. La función *read* retorna un valor entero el cual indica la cantidad de bytes o caracteres leídos, generalmente un valor > 0, si no se lo compara con un valor, entonces el compilador asume que se lo compara con distinto de cero, en este caso, si es verdadero ingresa al ciclo, pero, en caso que la función *read* retorne cero, situación en que el puntero del archivo se encuentra sobre la marca de fin de archivo, al intentar leer en ese punto, la función es cuando retorna cero, que al compararse por distinto de cero es falso, abandonando el ciclo.

Considerando que el tamaño del buffer representado por r es de 30 bytes.

```
||_____a_____|_____b_____|_____c_____|_____d_____||eof()
0            29 30          59 60          89 90          119  nros. de byte
      0              1              2              3          nros. de componentes
^   el puntero al archivo está al inicio.
```

Al abrir el archivo con ifstream p.e., el puntero (^) se ubica en la posición (0), al leer la primer componente el (^) avanza al inicio de la segunda componente (1) byte 30, al leer la segunda componente el (^) avanza a la tercera componente (2) byte 60, luego al leer la tercera componente el (^) avanza a la cuarta componente (3) byte 90 y finalmente lee la cuarta componente el (^) avanza quedando sobre la marca de fin de archivo eof(), en todos estos casos la función retorna el valor 30, luego cuando se intente leer nuevamente y estando el (^) sobre la marca de fin de archivo, la función retorna cero.

Grabar en archivo binario

```
g.write((char *) &r, sizeof r);
```

Graba donde está el puntero **toda la componente**, luego avanza el puntero al inicio de la próxima componente.

Leer y / o grabar en el mismo archivo binario

Utilizado para actualizar los datos en el mismo archivo. Las operaciones que deben de realizarse con acceso secuencial son:

1. Leer los datos de la componente actual, luego el puntero avanza al inicio de la próxima componente.
2. Actualizar en la RAM algún dato leído guardado en el buffer, que es ni más ni menos que la variable que utilizamos en la función *read*, generalmente un registro.
3. Mover el puntero una componente hacia atrás para que el puntero se ubique nuevamente en el registro últimamente leído.
4. Finalmente grabar los nuevos datos, pisando la componente que antes habíamos leido, luego el puntero avanza automáticamente a la próxima componente, estando preparada para leer, esa otra componente, esto se repite hasta alcanzar el fin del archivo.

```
while (h.read((char *) &r, sizeof r)) {
   r.PreUni *= (1+ porcje / 100);
   h.seekg(-1 * sizeof r, ios::cur);          // nueve el ptr a la componente previa
   h.write((const char *) &r, sizeof r);  // graba sobre la componente antes leída.
}                                          // las 2 acciones últimas mejor en un módulo.
```

Cerrar archivo

```
f.close();
```

```
g.close();
h.close();
```

Funciones boolean

```
f.good()     retorna true si leyó datos, sino, false.
f.bad()      f.good() <> not f.bad()
f.fail()
f.eof()      retorna true si el puntero está sobre la marca de fin de archivo, sino, false.
f.is_open()  retorna true si el archivo fue exitosamente abierto, caso contrario false.
```

while(!f.eof())

Obtener la cantidad de bytes o caracteres leídos

```
f.gcount()
```

Retorna un valor entero que indica la cantidad de bytes o caracteres leídos.

Mover el ptr a otra posición con direccionamiento directo

```
h.seekg(dir,desdeDonde);
```

si desdeDonde no se indica se asume desde ios::beg.

h.seekg(pos);

mueve el ptr del archivo al nro. de byte indicado por pos, desde el origen del archivo.

```
h.seekg(dir,ios::beg);
```

mueve el ptr a la dirección dir en cant. de bytes desde el inicio del archivo.

```
h.seekg(dir,ios::cur);
```

mueve el ptr a la dirección dir en cant. de bytes desde la pos. actual.

```
h.seekg(dir,ios::end);
```

mueve el ptr a la dirección dir en cant. de bytes desde la pos. final.

```
h.seekp(dir,ios::beg);
```

idem a seekg pero para archivos de salida, o sea, para grabar.

```
h.seekp(dir,ios::cur);
```

seekg para archivos abiertos de solo lectura o lectura / escritura.

```
h.seekp(dir,ios::end);
```

seekp para archivos abiertos de solo escritura o lectura / escritura.

g indica get, p indica put.

```
f.seekg(pos);
f.seekg(pos,ios::beg);
g.seekp(pos);
g.seekp(pos,ios::cur);
```

El desplazamiento en cualquiera de los inicios indicados puede ser positivo en este caso se desplaza el (^) hacia la derecha o direcciones más altas; en cambio, si es negativo en este otro caso el (^) se desplaza hacia la izquierda o direcciones más bajas.
Situación de error puede darse si la dir a desplazar el (^) cae antes de la posición cero, o sea, posiciones negativas, en estos casos será un error. No obstante, si el (^) se desplaza más allá de la marca de fin de archivo, no será error y se crean todas las componentes desde el fin de archivo hasta la pos del desplazamiento.

Determinar la ubicación del puntero

```
h.tellg()
```

retorna un entero que indica el nro. de byte al que apunta.

```
h.tellp()
```

retorna un entero que indica el nro. de byte al que apunta.

```
f.tellg()
```

g se utiliza para archivos abierto de solo lectura o lectura / escritura.

```
g.tellp()
```

p se utiliza para archivos abierto de solo escritura o escritura / lectura.

Limpiar flag de error

```
f.clear()
```

cuando se lee secuencialmente y se llega al eof() el flag se pone en -1, para poder volver a utilizar el archivo se debe limpiar el flag con la función o método clear(), además se debe mover el (^) al inicio, al byte cero.

```
f.seekg(0);
f.clear();
```

Comandos del sistema para archivos binarios

```
rename(nomVjo,nomNvo);
```

nomVjo y nomNvo son cadenas del nomFis.

```
remove(nomFis);
```

nomFis es una cadena que representa el nombre físico del archivo.

Redirecciona la entrada a un archivo

```
freopen(nomFis,"r",stdin);
```

Redirecciona la salida a un archivo

```
freopen(nomFis,"w",stdout);
```

Regresa la entrada y la salida a los dispositivos por defecto al teclado y a la pantalla respectivamente.

```
freopen("CON","r",stdin);
freopen("CON","w",stdout);
```

Versión stream G4Ej04 Creación archivo de Artículos

```
/*
        Id.Programa: G4Ej04stream.cpp
        Autor..........: Lic. Hugo Cuello
        Fecha..........: jun-2016
        Comentario.: Creacion stream archivo de Articulos.
*/

#include <fstream>
#include <conio.h>
using namespace std;

#define CENTINELA 0

typedef char str20[21];
typedef unsigned short ushort;

struct sArt {
        ushort CodArt;
        str20  Descrip;
        float  PreUni;
};

void ObtDatos(sArt &rArt) {
  do {
    gotoxy(5,1);
    cout << "Alta de Articulos.Dat";
          gotoxy(10,5); clreol();
          cout << "Cod.Articulo FIN = " << CENTINELA << ": ";
```

```
            cin >> rArt.CodArt;
        }
        while (!(rArt.CodArt <= 100));
        if (rArt.CodArt) {
                do {
                        gotoxy(10,7); clreol();
                        cout << "Descripcion: ";
                        gets(rArt.Descrip);
                }
                while (!strcmp(rArt.Descrip,""));
                do {
                        gotoxy(10,9); clreol();
                        cout << "Pre.Unitario: ";
                        cin >> rArt.PreUni;
                }
                while (!rArt.PreUni);
  }
} // ObtDatos

main() {
   sArt rArticulo;
   ofstream Articulos("Articulos2.Dat",ios::binary);

        ObtDatos(rArticulo);
        while (rArticulo.CodArt) {
                Articulos.write((const char *) &rArticulo, sizeof rArticulo);
                ObtDatos(rArticulo);
        }
        Articulos.close();
        return 0;
}
```

Versión stream G4Ej05 Actualización Precio en Artículos

```
/*
        Id.Programa: G4Ej05stream.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: jun-2016
        Comentario.: Actualizacion de Precio en archivo de Articulos.
*/

#include <conio.h>
#include <iomanip.h>
#include <fstream>
using namespace std;

#define modeOpen ios::in | ios::out | ios::binary

typedef char str20[21];
typedef unsigned short ushort;
```

```
struct sArt {
        ushort CodArt;
        str20  Descrip;
        float  PreUni;
};

// Prototipos ------------------

void ObtDato(float &);
void ActReg(fstream &, sArt );
// Fin Prototipos --------------

void ObtDato(float &porc) {
   clrscr();
   gotoxy(10,5); clreol();
   cout << "Porcentaje: ";
   cin >> porc;
} // ObtDato

void ActReg(fstream &Art, sArt rArt) {
   long tam = Art.tellg();

   tam -= sizeof (rArt);
   Art.seekg(tam,ios::beg);
   Art.write((const char *) &rArt, sizeof rArt);
} // ActReg

main() {
  sArt  rArticulo;
  float porcje;
   long pos;

   cout.precision(2);
   cout.setf(ios::fixed);
   fstream Articulos;
   Articulos.open("Articulos.Dat", modeOpen);
   ObtDato(porcje);
   while (Articulos.read((char *) &rArticulo,sizeof(rArticulo))) {
      rArticulo.PreUni *= (1 + porcje / 100);
      ActReg(Articulos,rArticulo);
      cout << left << setw(20) << rArticulo.Descrip << " " << right << setw(8)
            << rArticulo.PreUni << endl;
   }
   Articulos.seekg(0L,ios::end);
   cout << Articulos.tellg() << endl;
   Articulos.close();
   return 0;
}
```

G4Ej06 Precios Mayores de Articulos grabar en Mayores

```
/*
        Id.Programa: G4Ej06stream.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: jun-2016
        Comentario.: Genera archivo de Mayores precios de Articulos.
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef unsigned short ushort;
typedef char str20[21];

struct sArt {
  ushort CodArt;
  str20  Descrip;
  float  PreUni;
};

struct sMay {
  ushort CodArt;
  float  PreUni;
};

// Prototipos -----------------
void Abrir(ifstream &, ofstream &);
void ObtDato(float &);
void GenReg(ofstream &, sArt );
void Cerrar(ifstream &, ofstream &);
// Fin Prototipos -------------

void Abrir(ifstream &Art, ofstream &May) {
  Art.open("Articulos.Dat",ios::in | ios::binary);
  May.open("Mayores.Dat",ios::out | ios::binary);
} // Abrir

void ObtDato(float &impMax) {
  clrscr();
  gotoxy(10,5); clreol();
  cout << "Importe Maximo: ";
  cin >> impMax;
} // ObtDato

void GenReg(ofstream &May, sArt rArt) {
  sMay rMay;

  rMay.CodArt = rArt.CodArt;
  rMay.PreUni = rArt.PreUni;
  May.write((const char*) (&rMay),sizeof rMay);
```

```
} // GenReg

void Cerrar(ifstream &Art, ofstream &May) {
  Art.close();
  May.close();
} // Cerrar

main() {
  ifstream Articulos;
  ofstream Mayores;
  sArt  rArticulo;
  float maxImp;

  Abrir(Articulos,Mayores);
  ObtDato(maxImp);
  while (Articulos.read((char*) (&rArticulo),sizeof(rArticulo)))
    if (rArticulo.PreUni > maxImp)
      GenReg(Mayores,rArticulo);
  Cerrar(Articulos,Mayores);
  return 0;
}
```

Versión Corte de Control con stream

```
/*
        Id.Programa: G4Ej09stream.cpp Corte de Control n = 2.
        Autor......: Lic. Hugo Cuello
        Fecha......: jun-2016
        Comentario.: Examen a alumnos de distintas Universidades y Facultades.
             Tecnica de Corte de Control.
*/

#include <iomanip>
#include <fstream>
using namespace std;

typedef unsigned int  word;
typedef unsigned short byte;
typedef char str5[6];
typedef char str15[16];
typedef char str20[21];
struct sExa {
        str5  CodUni,
              CodFacu;
        long  NroLeg;
        str20 ApeNom;
        byte  Nota;
};

// Prototipos -------------------------------------------------------
```

```
void Abrir(ifstream &Examen);
void Inic(word &totInsGral, word &totAprGral);
void EmiteCabLst();
void IniCab(word &totIns, word &totApr, str5 Clave, str5 &ClaveAnt);
void EmiteCabCte(str15 titulo,str5 codigo);
void ProcAlum(sExa rExa, word &tInsFacu, word &tAprFacu);
void CalcPie(word totIns, word totApr, word &totInsM, word &totAprM);
void EmitePie(str15 titulo, word totInsFacu, word totAprFacu);
// Fin Prototipos ----------------------------------------------------

main() {
        ifstream Examen;
        sExa rExamen;
        word totInsGral,
      totAprGral,
      totInsUni,
      totAprUni,
      totInsFacu,
      totAprFacu;
        str5 UniAnt,
      FacuAnt;

        Abrir(Examen);
        freopen("Examen.Lst","w",stdout);
        Inic(totInsGral,totAprGral);
        EmiteCabLst();
        Examen.read((char*) (&rExamen),sizeof(rExamen));
        while (!Examen.eof()) {
                IniCab(totInsUni,totAprUni,rExamen.CodUni,UniAnt);
                EmiteCabCte("*Cod.Univ.: ",rExamen.CodUni);
                while (!Examen.eof() && strcmp(rExamen.CodUni,UniAnt) == 0) {
                        IniCab(totInsFacu,totAprFacu,rExamen.CodFacu,FacuAnt);
                        EmiteCabCte("**Cod.Fac.: ",rExamen.CodFacu);
                        while (!Examen.eof() && strcmp(rExamen.CodUni,UniAnt) == 0
                                && strcmp(rExamen.CodFacu,FacuAnt) == 0) {
                           ProcAlum(rExamen,totInsFacu,totAprFacu);
                           Examen.read((char*) (&rExamen),sizeof(rExamen));
                        }
                        CalcPie(totInsFacu,totAprFacu,totInsUni,totAprUni);
                        EmitePie("Facu.: ",totInsFacu,totAprFacu);
                }
                CalcPie(totInsUni,totAprUni,totInsGral,totAprGral);
                EmitePie("Univ..: ",totInsUni,totAprUni);
        }
        EmitePie("General: ",totInsGral,totAprGral);
        freopen("CON","w",stdout);
        Examen.close();
     Return 0;
} //main
```

```
void Abrir(ifstream &Exa) {

        Exa.open("Examen.Dat",ios::binary);
} //Abrir

void Inic(word &tInsG, word &tAprG) {

        tInsG = tAprG = 0;
} // Inic

void EmiteCabLst() {

        cout << "Listado examen a alumnos" << endl;
} //EmiteCab

void IniCab(word &totIns, word &totApr, str5 Clave, str5 &ClaveAnt) {

        totIns = totApr = 0;
        strcpy(ClaveAnt,Clave);
} //IniCab

void EmiteCabCte(str15 titulo, str5 codigo) {

        cout << endl << titulo << " " << codigo;
        if (strstr(titulo,"Fac") != NULL)
                cout << endl << setw(5) << " " << "Nro.Leg.  Nota" << endl;
} //EmiteCabCte

void ProcAlum(sExa rExa, word &tInsFacu, word &tAprFacu) {

        ++tInsFacu;
        if (rExa.Nota >= 4) {
                ++tAprFacu;
                cout << setw(6) << "  " << setw(6) << rExa.NroLeg << "    " << setw(2)
                        << rExa.Nota << endl;
        }
} //ProcAlum

void CalcPie(word totIns, word totApr, word &totInsM, word &totAprM) {

        totInsM += totIns;
        totAprM += totApr;
} //CalcPie

string replicate(char car, unsigned n) {
  string cad = "";

  for(unsigned i = 1; i <= n; i++)
    cad += car;
```

```
  return cad;
} // replicate

void EmitePie(str15 titulo, word totIns, word totApr) {
  string ast;

    if (titulo[0] == 'F')
      ast = replicate('*',2);
    else
      if (titulo[0] == 'U')
        ast = replicate('*',1);
  cout << " " << ast << "Tot.Insc. " << titulo << " " << setw(6) << totIns << endl;
  cout << " " << ast << "Tot.Apr.. " << titulo << " " << setw(6) << totApr << endl;
} //EmitePie
```

Versión apareo de archivos técnica High Value y stream

```
/*
    Id.Programa: G4Ej10HVstream.cpp Apareo
    Autor..........: Lic. Hugo Cuello
    Fecha..........: jun-2016
    Comentario.: Apareo entre MaeVjo y Nov.
                 Genera MaeNvo y LstErr.
                 Tecnica con HIGH_VALUE.
*/

#include <fstream>
using namespace std;

#define HIGH_VALUE 10000
typedef char str20[21];

struct sMae {
  long  cmpClv;
  int   cmp1,
        cmp2;
  str20 RazSoc,
        Domic,
        Local;
  long  NroCUIT;
};

struct sNov {
  sMae rMaeN;
  char codMov;
};

// Prototipos ------------------------------------------
void Abrir(ifstream &, ofstream &, ifstream &);
```

```
void ApareoHV(ifstream &, ofstream &, ifstream &);
void LecHV(ifstream &, sMae &);
void LecHV(ifstream &, sNov &);
void CerrarEliminarRenombrar(ifstream &, ofstream &, ifstream &);
// Fin Prototipos ---------------------------------------

void Abrir(ifstream &MaeV, ofstream &MaeN, ifstream &Nov) {

  MaeV.open("MaeVjo.Dat",ios::binary);
  MaeN.open("MaeNvo.Dat",ios::binary);
  Nov.open("Noved.Dat",ios::binary);
} //Abrir

void LecHV(ifstream &MaeV, sMae &rMaeV) {

    MaeV.read((char *) &rMaeV,sizeof(rMaeV));
    if (MaeV.eof())
        rMaeV.cmpClv = HIGH_VALUE;
} //LecHV

void LecHV(ifstream &Nov, sNov &rNov) {

    Nov.read((char *) &rNov,sizeof(rNov));
    if (Nov.eof())
        rNov.rMaeN.cmpClv = HIGH_VALUE;
} //LecHV

void ApareoHV(ifstream &MaeV, ofstream &MaeN, ifstream &Nov) {
  sMae rMaeV,
       rMaeN;
  sNov rNov;

  freopen("ErroresABMHD.Lst","w",stdout);
  LecHV(MaeV,rMaeV);
  LecHV(Nov,rNov);
  while (rMaeV.cmpClv != HIGH_VALUE && rNov.rMaeN.cmpClv !=
                                                        HIGH_VALUE)
    if (rMaeV.cmpClv == rNov.rMaeN.cmpClv) {
      switch (rNov.codMov) {
        case 'A': cout << "Error por Alta Existente, clave: " <<
                        rNov.rMaeN.cmpClv << "Ubicacion: " <<
                        (Nov.tellg() - (long) sizeof(rNov)) / sizeof(rNov) << endl;
                rMaeN = rMaeV;
                MaeN.write((const char *) &rMaeN,sizeof(rMaeN));
                break;
        case 'M': rMaeN = rMaeV;
                rMaeN.cmp1 = rNov.rMaeN.cmp1;
                strcpy(rMaeN.Domic,rNov.rMaeN.Domic);
                MaeN.write((const char *) &rMaeN,sizeof(rMaeN));
```

```
                break;
      }
      LecHV(MaeV,rMaeV);
      LecHV(Nov,rNov);
    }
    else
      if (rMaeV.cmpClv > rNov.rMaeN.cmpClv) {
        switch (rNov.codMov) {
          case 'A': rMaeN = rNov.rMaeN;
                    MaeN.write((const char *) &rMaeN,sizeof(rMaeN));
                    break;
          case 'B': cout << "Error por Baja inexistente, clave: " <<
                          rNov.rMaeN.cmpClv << "Ubicacion: " <<
                          (Nov.tellg() - (long) sizeof(rNov)) / sizeof(rNov) << endl;
                    break;
          default: cout << "Error por Modificacion inexistente, clave: " <<
                        rNov.rMaeN.cmpClv << "Ubicacion: " <<
                        (Nov.tellg() - (long) sizeof(rNov)) / sizeof(rNov) << endl;
        }
        LecHV(Nov,rNov);
      }
      else {
        rMaeN = rMaeV;
        MaeN.write((const char *) &rMaeN,sizeof(rMaeN));
        LecHV(MaeV,rMaeV);
      }
  freopen("CON","w",stdout);
}  //ApareoHV

void CerrarEliminarRenombrar(ifstream &MaeV, ofstream &MaeN, ifstream &Nov)
{

  MaeV.close();
  MaeN.close();
  Nov.close();
  remove("MaeVjo.Dat");
  rename("MaeNvo.Dat","MaeVjo.Dat");
} //CerrarEliminarRenombrar

int main() {
  ifstream MaeVjo,
           Noved;
  ofstream MaeNvo;

  Abrir(MaeVjo,MaeNvo,Noved);
  ApareoHV(MaeVjo,MaeNvo,Noved);
  CerrarEliminarRenombrar(MaeVjo,MaeNvo,Noved);

  return 0;
}
```

Versión stream ejercicio Facturación a Clientes en Cta./Cte.

```
/*
   Id.Programa: G4Ej11FacturasStream.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: jun-2016
   Comentario.: Emision de Facturas a Clientes en Cta.Cte.
                Relaciones entre archivos.
*/

#include<iomanip>
#include <fstream>
using namespace std;

#define IVA 0.21

typedef char str5[6];
typedef char str10[11];
typedef char str15[16];
typedef char str20[21];
typedef unsigned short byte;
typedef unsigned word;

struct sPed {
  long codCli; //Ord. por codCli (con repeticion) + codArt (sin repeticion)
  byte codArt;
  word cant;
};

struct sCli {    //Ord. por codCli
  long  codCli;  //8 digitos.
  str20 razSoc,
        domic,
        local;
  word  codPos;
  char  codPcia;
  str15 nroCUIT;
  str20 formaPago;
  word  codVen;
  float saldo; //Actualizar el saldo
};
```

```
struct sArt {   //Relacion 1:1 clv = dir.
  byte  codArt; //1..100
  str20 marca,
        descrip;
  float precio;
  word  stockAct, //Actualizar el stockAct.
        stockMin,
        ptoRep;
  str10 uniMed;
  char  estado;
};

struct sVen {       //desordenado
  word  codVen;     //1..999
  str20 nomVen;
  float porcCom,
        impAcumCom; //Actualizar el impAcumCom.
};

struct sNro {       //desordenado
  str5 tipoDoc;
  long nroDoc;      //Actuaizar nroDoc.
};

struct sFecF {
  byte dia,
       mes;
  word year;
};

struct sFac {  //Ord.cronologico. El archivo existe y se deben agregar +registros.
  long  nroFac;
  sFecF fecFac;
  long  codCli;
  float impo;
  char  estado; //'A' = Adeuda, 'P' = Pagada.
};

struct sFec {

  int year,
      mes,
      dia,
      dsem;
```

```
};

struct sHor {
  int Hora,
      Min,
      Seg;
};

// Prototipos ------------------------------------------------------------
void Abrir(ifstream &, fstream &, fstream &, fstream &,fstream &, ofstream &, fstream
&);
void ArmarAuxVen(fstream &, fstream &);
long GetDate(int &, int &, int &, int &);
void EmiteTapaObtFecha(sFec &);
void BusLinNrosInic(fstream &, str5 , sNro &);
void InicCabCli(byte &, float &, long &nDoc);
bool BusBinCli(fstream &, long , sCli &);
void BusDDVen(fstream &, word , sVen &, fstream &);
void EmiteCabCli(sCli , str20 , sFec , long );
void BusDDArt(fstream &, byte , sArt &);
void CalcDetFac(word , sArt &, byte &, float &,float &);
void ActArt(fstream &, sArt );
void EmiteDetFac(byte , sArt , word , float );
void CalcPieCli(float ,float &, float &, float &, sVen &);
void ActCli(fstream &, sCli );
void ActVen(fstream &, sVen );
void AgregarRegFac(fstream &, long , sFec , long , float );
void EmitePieFac(byte , float , float , float );
void ActNroI(fstream &, sNro );
void CerrarEliminar();
// Fin Prototipos -------------------------------------------------------

void Abrir(ifstream &Ped, fstream &Cli, fstream &Art, fstream &Ven, fstream
&nInic, ofstream &Fac, fstream &AuxVen) {

  Ped.open("Pedidos.Dat",ios::binary);
  Cli.open("Clientes.Dat",ios::binary | ios::in | ios::out);
  Art.open("Articulos.Dat",ios::binary | ios::in | ios::out);
  Ven.open("Vendedores.Dat",ios::binary | ios::in | ios::out);
  nInic.open("NrosInic.Dat",ios::binary | ios::in | ios::out);
  Fac.open("Facturas.Dat",ios::app | ios::binary);
  AuxVen.open("AuxVen.Tmp",ios::binary | ios::in | ios::out);
} //Abrir
```

```
void ArmarAuxVen(fstream &Ven, fstream &AuxVen) {
  int  refVen = -1;
  sVen rVen;

  for (int i = 0; i <= 999; i++)
    AuxVen.write((const char *) &refVen,sizeof(int));
  while (Ven.read((char *) &rVen,sizeof(rVen))) {
    AuxVen.seekp(rVen.codVen * sizeof(rVen),ios::beg);
    refVen = (Ven.tellg() - (long) sizeof(rVen)) / sizeof(rVen);
    AuxVen.write((const char *) &refVen,sizeof(refVen));
  }
} //ArmarAuxVen

long GetDate(int &year, int &mes, int &dia, int &ds) {
  time_t     rawtime;
  struct tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  year = 1900 + timeinfo->tm_year;
  mes  = 1 + timeinfo->tm_mon;
  dia  = timeinfo->tm_mday;
  ds   = 1 + timeinfo->tm_wday;
  return (1900 + timeinfo->tm_year) * 10000 + (1 + timeinfo->tm_mon) * 100 +
timeinfo->tm_mday;
} // GetDate

long GetTime(int &hh, int &mm, int &ss) {
  time_t     rawtime;
  tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  hh = timeinfo->tm_hour;
  mm = timeinfo->tm_min;
  ss = timeinfo->tm_sec;
  return timeinfo->tm_hour * 10000 + timeinfo->tm_min * 100 + timeinfo->tm_sec;
} // GetTime

void EmiteTapaObtFecha(sFec &rFec) {

  cout << "FACTURACION A CLIENTES" << endl;
  cout << " EN CUENTA CORRIENTE" << endl;
```

```
  GetDate(rFec.year,rFec.mes,rFec.dia,rFec.dsem);
} //EmiteTapaObtFecha

void BusLinNrosInic(fstream &nInic, str5 tDoc, sNro &rNroI) {
  bool sigo = true;

  while (sigo) {
    nInic.read((char *) &rNroI,sizeof(rNroI));
    if (strcmp(rNroI.tipoDoc,tDoc) == 0)
      sigo = false;
  }
} //BusLinNrosInic

void InicCabCli(byte &nItem, float &tFac, long &nDoc) {

  nItem = 0;
  tFac = 0.0;
  nDoc++;
} //InicCabCli

void ObtHora(sHor &rHor) {

  cout << "FACTURACION A CLIENTES" << endl;
  cout << " EN CUENTA CORRIENTE" << endl;
  GetTime(rHor.Hora,rHor.Min,rHor.Seg);
} //ObtHora

bool BusBinCli(fstream &Cli, long cCli, sCli &rCli) {
  int pri,
      ult,
      med;

  pri = 0;
  Cli.seekg(0L,ios::end);
  ult = Cli.tellg() / sizeof (rCli) - 1;
  while (pri <= ult) {
    med = (pri + ult) / 2;
    Cli.seekg(med * sizeof (rCli),ios::beg);
    Cli.read((char *) &rCli,sizeof (rCli));
    if (rCli.codCli == cCli)
      return true;
    else
      if (rCli.codCli < cCli)
```

```
      pri = med + 1;
    else
      ult = med - 1;
  }
  return false;
} // BusBinCli

void BusDDVen(fstream &Ven, word cVen, sVen &rVen, fstream &AuxiV) {
  word refVen;

  AuxiV.seekg(cVen * sizeof(rVen),ios::beg);
  AuxiV.read((char *) &refVen,sizeof(refVen));
  Ven.seekg(refVen * sizeof(rVen),ios::beg);
  Ven.read((char *) &rVen,sizeof(rVen));
} //BusDDVen

void EmiteCabCli(sCli rCli, str20 nVen, sFec rFec, sHor rHor, long nDoc) {
  cout << setw(38) << " " << "FACTURA: " << nDoc << endl;
  cout << setw(38) << " " << "FECHA..: " << setw(2) << rFec.dia << '-' <<
                                 setw(2) << rFec.mes << '-' <<
                                 setw(4) << rFec.year << endl;
  cout << setw(38) << " " << "HORA...: " << setw(2) << rHor.Hora << '-' <<
                                 setw(2) << rHor.Min << '-' <<
                                 setw(4) << rHor.Seg << endl;
  cout << "Cod. Cliente.: " << setw(8) << rCli.codCli <<
         "Raz. Social..: " << setw(20) << rCli.razSoc <<
         "Domicilio....: " << setw(20) << rCli.domic <<
         "Forma de Pago: " << setw(20) << rCli.formaPago <<
         "Cod. Vendedor: " << setw(3) << nVen << endl;
  cout << endl;
  cout << setw(40) << "-" << endl;
  cout << "Item  Cant.  Cod.Art. Descripcion       Pre.Uni.    T.Item" << endl;
  cout << setw(40) << "-" << endl;
} //EmiteCabCli

void BusDDArt(fstream &Art, byte cArt, sArt &rArt) {

  Art.seekg(cArt * sizeof(rArt),ios::beg);
  Art.read((char *) &rArt,sizeof(rArt));
} //BusDDArt

void CalcDetFac(word cant, sArt &rArt, byte &nItem, float &tItem,float &totFac) {

  nItem++;
```

```
  tItem = cant * rArt.precio;
  rArt.stockAct -= cant;
  totFac += tItem;
} //CalcDetFac

void ActArt(fstream &Art, sArt rArt) {

  Art.seekg(Art.tellg() - (long) sizeof(rArt),ios::beg);
  Art.read((char *) &rArt,sizeof(rArt));
} //ActArt

void EmiteDetFac(byte nItem, sArt rArt, word cant, float tItem) {

  cout << setw(2) << nItem <<
        setw(5) << cant <<
        setw(3) << rArt.codArt <<
        setw(20) << rArt.descrip <<
        setw(8) << rArt.precio <<
        setw(8) << tItem << endl;
} //EmiteDetFac

void CalcPieCli(float tBruFac,float &impIVA, float &tNetoFac, float &saldo, sVen
&rVen) {

  impIVA = tBruFac * IVA;
  tNetoFac = tBruFac + impIVA;
  saldo += tNetoFac;
  rVen.impAcumCom += tBruFac * rVen.porcCom / 100;
} //CalcPieCli

void ActCli(fstream &Cli, sCli rCli) {

  Cli.seekp(Cli.tellp() - (long) sizeof(rCli),ios::beg);
  Cli.write((const char *) &rCli,sizeof(rCli));
} //ActCli

void ActVen(fstream &Ven, sVen rVen) {

  Ven.seekp(Ven.tellp() - (long) sizeof(rVen),ios::beg);
  Ven.write((const char *) &rVen,sizeof(rVen));
} //ActVen
```

```
void AgregarRegFac(ofstream &Fac, long nFac, sFec rFec, long cCli, float tNetoFac)
{
  sFac rFac;

  rFac.nroFac = nFac;
  rFac.fecFac.dia = rFec.dia;
  rFac.fecFac.mes = rFec.mes;
  rFac.fecFac.year = rFec.year;
  rFac.codCli = cCli;
  rFac.impo = tNetoFac;
  rFac.estado = 'A';
  Fac.write((const char *) &rFac,sizeof(rFac));
} //AgregarRegFac

void EmitePieFac(byte nItem, float tBruFac, float impIVA, float tNetoFac) {
  byte i;

  for (i = 1; i <= 10 - nItem; i++)
    cout << endl;
  cout << setw(38) << " " << "Sub-Total.: " << setw(8) << tBruFac <<
         setw(38) << " " << "I.V.A.....: " << setw(8) << impIVA <<
         setw(38) << " " << "Total Neto: " << setw(8) << tNetoFac << endl;
} //EmitePieFac

void ActNroI(fstream &nInic, sNro rNroI) {

  nInic.seekp(nInic.tellp() - (long) sizeof(rNroI),ios::beg);
  nInic.write((const char *) &rNroI,sizeof(rNroI));
} //ActNroI

void CerrarEliminar() { //FILE *Ped, FILE *Cli, FILE *Art, FILE *Ven,FILE *nInic,
                        //FILE *Fac, FILE *AuxVen)

  fcloseall();
  remove("AuxVen.Tmp");
} //CerrarEliminar

int main() {
  ifstream Pedidos;
  fstream  Clientes,
           Articulos,
           Vendedores,
           NrosInic,
           AuxVendedores;
```

```
ofstream Facturas;
sFec rFecha;
sHor rHora;
sPed rPedido;
sNro rNroInic;
sCli rCliente;
sVen rVendedor;
sArt  rArticulo;
byte  nroItem;
float totItem,
        totBrutoFac,
        impIVA,
        totNetoFac;

Abrir(Pedidos,Clientes,Articulos,Vendedores,NrosInic,Facturas,AuxVendedores);
freopen("FacturasCtaCteCli.Lst","w",stdout);
ArmarAuxVen(Vendedores,AuxVendedores);
EmiteTapaObtFecha(rFecha);
BusLinNrosInic(NrosInic,"FAC",rNroInic);
Pedidos.read((char *) &rPedido,sizeof(rPedido));
while (!Pedidos.eof()) {
  InicCabCli(nroItem,totBrutoFac,rNroInic.nroDoc);
  ObtHora(rHora);
  BusBinCli(Clientes,rPedido.codCli,rCliente);
  BusDDVen(Vendedores,rCliente.codVen,rVendedor,AuxVendedores);
  EmiteCabCli(rCliente,rVendedor.nomVen,rFecha,rHora,rNroInic.nroDoc);
  while (!Pedidos.eof() && rPedido.codCli == rCliente.codCli) {
    BusDDArt(Articulos,rPedido.codArt,rArticulo);
    CalcDetFac(rPedido.cant,rArticulo,nroItem,totItem,totBrutoFac);
    ActArt(Articulos,rArticulo);
    EmiteDetFac(nroItem,rArticulo,rPedido.cant,totItem);
    Pedidos.read((char *) &rPedido,sizeof(rPedido));
  }
  CalcPieCli(totBrutoFac,impIVA,totNetoFac,rCliente.saldo,rVendedor);
  ActCli(Clientes,rCliente);
  ActVen(Vendedores,rVendedor);
  AgregarRegFac(Facturas,rNroInic.nroDoc,rFecha,rCliente.codCli,totNetoFac);
  EmitePieFac(nroItem,totBrutoFac,impIVA,totNetoFac);
}
ActNroI(NrosInic,rNroInic);
freopen("CON","w",stdout);
CerrarEliminar(); //usa fcloseall() Pedidos, Clientes, Articulos, Vendedores, NrosInic,
                  //Facturas, AuxVendedores);
```

```
  return 0;
}
```

Versión stream de Gastos anuales

```
/*
    Id.Programa: G4Ej1e3GtosStream.cpp
    Autor......: Lic. Hugo Cuello
    Fecha......: jun-2016
    Comentario.: Listado de Gastos ord. x Mes y Dia Imp.Acum.
*/

#include<iomanip>
#include<fstream>
#include<iostream>
using namespace std;

typedef unsigned short byte;
struct sGto {
  byte  mes,
        dia;
  float impo;
};

// Prototipos -------------------------
void Abrir(ifstream &, fstream &);
void ArmarAuxG(fstream &);
int CantDias(byte );
void AcumGto(fstream &, sGto );
string MesStr(unsigned );
void Listado(fstream &);
void CerrarEliminar(ifstream &, fstream &);
// Fin Prototipos ---------------------

void Abrir(ifstream &Gtos, fstream &AuxG) {

  Gtos.open("Gastos.Dat",ios::in | ios::binary);
  AuxG.open("AuxGtos.Tmp",ios::in | ios::out | ios::binary);
} //Abrir

void ArmarAuxG(fstream &AuxG) {
  float impAcum = 0;

  for (int i = 1; i <= 365; i++)
    AuxG.write((const char *) &impAcum,sizeof(impAcum));
```

```
} //ArmarAuxG

int CantDias(byte mes) {
  int sumDias = 0;

  for (int mes_i = 1; mes_i < mes; mes_i++)
    switch (mes_i) {
      case  4:
      case  6:
      case  9:
      case 11: sumDias += 30;
               break;
      case  2: sumDias += 28;
               break;
      default: sumDias += 31;
    }
  return sumDias;
} //CantDias

void AcumGto(fstream &AuxG, sGto rGto) {
  float impAcum;
  int   pos;

  pos = CantDias(rGto.mes) + rGto.dia - 1;
  AuxG.seekg(pos * sizeof(float),ios::beg);
  AuxG.read((char *) &impAcum,sizeof(impAcum));
  impAcum += rGto.impo;
  AuxG.seekp(pos * sizeof(float),ios::beg);
  AuxG.write((const char *) &impAcum,sizeof(float));
} //AcumGto

string MesStr(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
    case  8: return "Agosto";
    case  9: return "Septiembre";
```

```
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStr

void Listado(fstream &AuxG) {
  float impAcum,
        totGral = 0.0,
        totMes,
        impMenDia,
        impMayMes = 0.0;
  byte  nroDiaMen,
        nroMesMay;

  AuxG.seekg(0);   // Equivalente a AuxG.seekg(0, ios::beg);
  freopen("GtosMesDiaAcum.Lst","w",stdout);
  cout.precision(2);
  cout.setf(ios::fixed);
  cout << "Listado de Gastos ord. por Mes y Dia por Imp.Acum." << endl;
  for (int mes = 1; mes <= 12; mes++) {
    totMes = 0;
    impMenDia = 1E6;
    cout << "Mes: " << MesStr(mes) << endl;
    cout << setw(7) << " " << "Dia    Imp.Acu." << endl;
    for (byte dia = 1; dia <= CantDias(mes + 1) - CantDias(mes); dia++) {
      AuxG.read((char *) &impAcum,sizeof(impAcum));
      if (impAcum) {
        if (impAcum < impMenDia) {
          impMenDia = impAcum;
          nroDiaMen = dia;
        }
        cout << setw(8) << " " << setw(2) << dia
             << setw(4) << " " << setw(8) << impAcum << endl;
        totMes += impAcum;
      }
    }
    totGral += totMes;
    if (impMenDia > impMayMes) {
      impMayMes = impMenDia;
      nroMesMay = mes;
    }
    cout << "Tot. Mes: " << totMes << endl;
```

```
    cout << "Nro. Dia < $: " << nroDiaMen << endl << endl;
  }
  cout << "Tot. Anual: " << totGral << endl;
  cout << "Nro. Mes de > $ de los < Dias: " << setw(2) <<
          nroMesMay << "(" << MesStr(nroMesMay) << ")" << endl;
  freopen("CON","w",stdout);
} //Listado

void CerrarEliminar(ifstream &Gtos, fstream &AuxG) {

  Gtos.close();
  AuxG.close();
  remove("AuxGtos.Tmp");
} //CerrarEliminar

main() {
  ifstream Gastos;
  fstream  AuxGtos;
  sGto rGto;

  Abrir(Gastos,AuxGtos);
  ArmarAuxG(AuxGtos);
  while (Gastos.read((char *) &rGto, sizeof(rGto)))
    AcumGto(AuxGtos,rGto);
  Listado(AuxGtos);
  CerrarEliminar(Gastos,AuxGtos);
  return 0;
}
```

U.T.N. F.R.B.A. Algoritmos y Estructuras de Datos Prof.: Hugo A. Cuello
Guía de ejercicios Nro. 4:
Estructuras de datos II: Archivos Binarios.

Aplicar en los siguientes ejercicios, comprensión del problema, diseño de la estrategia y desarrollar el algoritmo.

Guía de ejercicios #4

1. Crear el archivo **ARTICULOS.DAT**, grabando en forma secuencial, los registros ingresados por medio del teclado, cuyos datos son: Código de artículo (1..100), Descripción (cad20) y Precio unitario (real). Los datos se ingresan sin ningún orden. Validar cada dato ingresado: Cód.Art. 1..100, Descrip. <> "" y Prec.Unit.> 0.
2. Se requiere la actualización del archivo **ARTICULOS.DAT** aplicando un porcentaje único dado por teclado en el campo Precio, a todos los artículos, grabando en el mismo archivo.
3. Se requiere de un proceso tras conocer un valor único dado por teclado, genere un archivo **MAYORES.DAT**, a partir de **ARTICULOS.DAT**, con aquellos registros que cumplan con la siguiente condición; precio mayor al valor conocido; cuyo formato de registro consta de: Código de artículo y Precio.
4. Crear un archivo cascarón **ARTREL.DAT** conteniendo 100 artículos cuyos códigos están en el intervalo de 1 a 100. El valor de la clave está relacionado 1 a 1 con su dirección en el almacenamiento externo. Un valor de cero en el campo Código indicará que el registro no está activo. Los dos campos restantes indican Nombre y Precio de artículo. El registro con dirección cero, no se lo utilizará para el proceso, no obstante deberá crearse.

Proceso de actualización con Altas, Bajas y Modificaciones.

5. Se requiere de un proceso interactivo en el cual el usuario eligirá de un menú algunas de las siguientes opciones: **A: Alta, B: Baja, M: Modificación y L: Listar**. El programa deberá validar el valor de la clave, luego se buscará el registro, será error si existe sí se seleccionó un Alta, o si no existe y sí se seleccionó una Baja o una Modificación; caso contrario, se deberá proceder con el resto de las acciones. En el caso de listar se solicitará el valor de dos claves, para listar los artículos que se encuentren dentro del intervalo indicado (desde – hasta).

Proceso con la técnica de Corte de Control.

6. Se requiere de un proceso que informe los resultados obtenidos por cada estudiante de cada Facultad correspondiente a cada Universidad.

Para ello se cuenta con un archivo de datos **Exámenes.Dat** el cual se encuentra **ordenado por Universidad y Facultad**, conteniendo: Código de Universidad (str5), Código de Facultad (str5), Número de Legajo del alumno (7 díg.), Apellido Nombre (30), y Nota (2 díg.) obtenida en el examen.

Se solicita emitir lo siguiente:

Listado Examenes a alumnos aprobados en Universidades-Facultades
***Universidad**: X(5)

****Facultad**: X(5)

Nro.Legajo	**Nota**
9(7)	99

****Cantidad de inscriptos**: 9(4)
****Cantidad de aprobados**: 9(4)

***Cantidad de inscriptos**: 9(5)
***Cantidad de ap**robados: 9(5)

Cantidad de inscriptos: 9(6)
Cantidad de aprobados: 9(6)

Observación: Se aprueba con una nota mayor o igual a cuatro.

Proceso con la técnica de Apareo de Archivos.

7. Dados dos archivos **ordenados por el mismo valor clave** y ambos **sin repetición**, siendo uno de ellos el archivo **Maestro** y el otro de **Novedades**, se requiere actualizar el maestro en un nuevo archivo **Maestro actualizado**, sabiendo que el archivo de **Novedades** tiene los mismos campos que el maestro, más un campo de **Código de movimiento**, pudiendo contener una A: Alta, B: Baja o una M: Modificación. Además durante el proceso de actualización si se presenta un error de inconsistencia, se deberá informar en un **listado de errores**, el valor de la clave, tipo de error: **Alta existente, Baja o Modificación inexistente**, la clave y la posición del registro en el archivo de novedades. Batch = Lotes

Proceso con Relación entre archivos.

8. Se requiere de un proceso que emita las facturas a cada cliente de las operaciones realizadas en un período de tiempo. Para ello se cuentan con los siguientes archivos de datos:

   Articulos: Código de Artículo(1..100), Marca(str20), Descripción(str20), Precio(real), Stock Actual(entero), Stock Mínimo(entero), Punto de reposición(entero), Unidad de Medida(str10), Estado(car.). El valor del Código de Artículo se corresponde con su dirección en el almacenamiento externo, es decir, existe una relación 1 a 1.
   Clientes: Código de Cliente(8 díg.), Nombre(str20), Dirección(str20), Localidad(str20), Código Postal(1..9999), Código de Provincia('A'..'Z'), Nro. CUIT(str15), Forma de pago(str20), Código del vendedor(1..999), Saldo(real). Se encuentra ordenado por Código de Cliente.
   Pedidos: Código de Cliente, Código de Artículo, Cantidad(entero). Ordenado por Cliente con repetición y por Código de artículo s/repetición.
   Vendedores: Código de vendedor, Nombre(str20), Porcentaje de comisión(1..100), Importe acumulado de comisión por venta(real). El archivo está desordenado.
   Facturas: Numero de factura(8 díg.), Fecha de factura(ddmmaa), Código del cliente, Importe(real), Estado('A':adeuda, 'P':pagada). El archivo se encuentra ordenado cronológicamente. Este archivo existe antes de iniciar el proceso.
   NrosInic: Tipo de documento(str5), Número de documento(8 díg.). Se encuentra desordenado.

   Se pide:

   1: Emitir por cada cliente la factura. Según diseño a presentar al explicarse el ejercicio en clase.
   2: Actualizar los archivos: Articulo, Cliente, Vendedor, Facturas y NrosInic., según lo explicado en clase.

   Observación: Recursos de almacenamiento externo disponible para ser utilizado en forma racional.

Proceso con técnica de hashing

9. Se requiere de un proceso para obtener un listado de los gastos realizados en un año calendario, para ello se cuenta con el siguiente archivo:
   Gastos: sin orden, conteniendo: Mes(1..12), Dia(1..31), Importe del gasto(real). Se puede repetir otro gasto para un mismo día de un mismo mes.

   Se pide:

   Emitir ordenado por Mes y Día, por cada día de cada mes el importe total gastado, el importe total del mes y el importe total anual. El día en que se produjo el menor gasto por cada mes, el mes que se produjo el mayor gasto de los menores. Completar con títulos apropiados, el mes deberá emitirse el nombre (Enero, Febrero,...).

   Listado de Gastos ordenado por Mes y Dia acumulado

   Mes: X(10)

Dia	**Importe Acumulado**
99	9999,99

   Total del mes: 9(5),99
   Dia de menor gasto: 99

   Total anual: 9(6),99
   Nro. de mes de mayor gasto de los menores: 99

10. La Dirección de rentas de una provincia requiere poder realizar el cobro de cuotas adeudadas con su valor actualizado, para ello cuenta con los siguientes archivos:

    a. **Deuda**: por cada contribuyente habrá un registro cabecera, seguido de uno o varios campos o ningún registro de cuota adeudada, c/u. de ellos con el siguiente formato de registro: Nro.de Contribuyente(8 díg.), Apellido y Nombre(cad30), Mes/año de la cuota (2+4 díg.) –(00 / 00 para cabecera)-, Importe de origen de la cuota adeudada(real), Código de Estado (‘A’=adeuda, ‘P’=pagada) inicialmente está en ‘A’. Ordenado por **Nro.Cbte + año + mes**.
    b. **Indices**: conteniendo los índices de actualización de importes al mes de proceso, desde el mes 1 del año 1980 en adelante: Coeficiente de actualización(real).
    c. **Contribuyentes**: con los números de contribuyentes solicitantes de deuda, sin orden alguno, conteniendo: Número de Contribuyente.

    Se pide:

    1. El listado de deuda actualizada por contribuyente según se indica:
       Nro.Cbte.: 9(8) **Apellido, Nombre**: X(30)

AÑO	**MES**	**IMPORTE ORIGEN**	**IMPORTE ACTUAL**
9999	99	999999,99	9999999,99

    2. Al final del proceso, grabe un archivo con el mismo formato que el del punto a) con las cuotas adeudadas para usar al día siguiente.
    3. Grabe un archivo con la cobranza del día, donde por cada contribuyente que pagó su deuda se grabe un registro con: Número de contribuyente, Apellido y Nombre, Importe total pagado.

11. Una playa de estacionamiento por hora (con 1800 cocheras en 18 pisos), desea establecer ciertos controles. Para ello dispone de:
    Tarifas: con un único registro de las tarifas por hora de los coches pequeños, medianos y grandes.

Cocheras: conteniendo un registro por cada cochera, c/u. de ellos con: Patente del auto que ocupa la cochera(cad6), Hora de ocupación(dd.dd siendo la parte entera la hora y la parte decimal los minutos), Tipo de auto('P'=Pequeño, 'M'=Mediano, 'G'=Grande).
A continuación y durante todo el día se procesan registros con los siguientes datos por cada entrada/salida del estacionamiento: Código de entrada/salida (E/S o F para finalizar), Patente, Tipo de auto, Piso requerido(0..17), Hora de entrada o salida(dd.dd). El lote está ordenado naturalmente por hora.

Se pide:

1. Imprima un ticket por cada movimiento:

De SALIDA		**De ENTRADA**	
ENTRADA	: 99.99	**ENTRADA**	: 99.99
PISO	: 99	**PISO**	: 99
TIPO	: X(7)	**TIPO**	: X(7)
PATENTE	: XXX999	**PATENTE**	: XXX999
SALIDA	: 99.99		
IMPORTE	: 999,99		

2. Grabe en un archivo histórico un registro por cada ocupación con: Patente, Hora de entrada, Hora de salida, Piso, Importe.
3. Imprima al final del día un listado por cada piso con el formato para los autos que quedan dentro del estacionamiento:
   Piso: 99

Patente	**Tipo**	**Importe**
XXX999	X(7)	999.99

   Observación: Se cuenta con recursos en memoria externa a efectos de mejorar el proceso.

12. Un organismo provincial debe imprimir las patentes de su parque automotor, para ello cuenta con los siguientes archivos de registros:

Vehículos: Ordenado por código postal y patente, conteniendo: Modelo, Año de fabricación, Patente, Titular(cad40), Domicilio(cad40), Código postal(4 díg.).
Importes: ordenado por modelo y año de fabricación, donde cada registro contiene: Modelo (4 díg.), Año de Fabricación (4 díg.), Importe Anual(real).
Infracciones: Ordenado por patente, conteniendo: Nro.de patente(str6), Cantidad de infracciones(0 a 999). Hay un registro por cada vehículo.
Se ingresa por teclado solo una vez, la cantidad de cuotas y por cada cuota la fecha de vencimiento.
Desarrollar un programa que:

1. Pida la cantidad de cuotas -única para todos los vehículos- por año <= 12 y sus fechas de vencimiento(ddmmaaaa).
2. Imprima para cada vehículo ordenado por número de patente sus cuotas con el siguiente formato:

PATENTE : XXX999 **TITULAR** : X(40)
COD.POS. : 9999 **DOMICILIO** : X(40) **CANT.INFR**.: 999

CUOTA	**IMPORTE**	**FEC.VTO**
99	999.99	dd/mm/aaaa
..	..	..

13. Se tiene un archivo **Deportes.Dat**, sin orden, conteniendo el registro:

Código de Deporte(3 díg.), Nombre del Deporte(30 car.).

Se pide:

1. Emitir un listado que informe Código y Nombre de Deporte, ordenado por Código de Deporte. Optimizar el espacio en disco.

2. Idem anterior pero, el espacio disponible en disco es de 5000 bytes.

   Emitirlo en forma invertida :

   - Realizando lectura hacia atrás.
   - Realizando direccionamiento directo.
   - Realizando una copia en archivo auxiliar.

19. La D.G.I. requiere realizar un muestreo con las ventas de cierto tipo de empresas. Para ello se cuenta con:

    a) Un archivo de **Empresas** : **Nro.Empresa**(1..300000), Domicilio(20 car.), Razón Social(30 car.), Cód. Pos.(4 díg.), Localidad (30 car.). Existe una relación 1 a 1 de la **clave** Nro. Empresa con la dirección en disco.

    b) Un archivo de **Ventas.Dat** ordenado por Nro. Empresa y mes, conteniendo: Nro. Empresa, Nro. mes(1..12), Importe (real).

    Se pide :

    1) Imprima para las 100 empresas con mayor importe anual de ventas, **ordenado** en forma decreciente de importe:

Nro.Empresa	Domicilio	Localidad	Cod.Pos.	Importe anual
999999	X(20)	X(30)	9999	9(8).99

    2) Para las empresas que superen el millón de pesos generar un archivo **ordenado** por código postal con el siguiente registro : Nro.Empresa, Domicilio, Importe venta anual. No habrá más de 5000 empresas.

20. Una Municipalidad requiere un programa que permita realizar la cancelación de deuda de sus contribuyentes. Para ello se dispone de los siguientes archivos :

    a) **Deudores** : sin ningún orden donde cada registro contiene : Cód. de Contribuyente(5 car.), Año de la cuota (1998 a 2007), Mes de la cuota (1 a 12), Importe de la cuota, Estado (‘A’=Adeuda, ‘C’=Cancelada), Fecha de cobro (aaaammdd).
    b) **DatoPers** : <= 8000 registros ordenados por contribuyente, cada registro con : Cód. de Contribuyente., Apellido y Nombre (45 car.), Domicilio (45 car.), Cód. de zona Municipal (4 car.).
    c) **CobroMes** : donde cada registro contiene : Cód. de Contribuyente, Año de la cuota, Mes de la cuota, Fecha de cobro (aaaammdd), ordenado por fecha de cobro y código de contribuyente.

    Se pide :

    1. Actualice en el archivo Deudores las cuotas pagadas por cada contribuyente.
    2. Imprima un listado para control de cobranza como se indica :

Listado de control de cobranza

Contrib.	Ape. y Nom.	Año	Mes	Importe	Situac.
X(5)	X(45)	9999	99	99999.99	P
X(5)	X(45)	9999	99	99999.99	E

    en donde : **P** = Pago aceptado, **E** = error, cuota pagada anteriormente.

    3. Imprima el total de lo recaudado por cada zona Municipal ordenado por código de zona :

Cód. Zona	Importe Recaudado
XXXX	999999,99

    Nota : Se disponen de 4 Mbytes de memoria estática.

21. La Municipalidad desea realizar un control estadístico de las infracciones de los colectivos. Para ello dispone de :

    a) Un archivo de **Patentes.Dat** ordenado por Patente, donde cada registro contiene : Patente del colectivo (XXX999), Número de línea del colectivo (3 díg.).

    b) Un archivo de las infracciones de un mes, donde cada registro contiene : Día de la infracción (1 a 31), Patente del colectivo, Código de infracción (1 a 99).

    c) Un archivo de importes de infracciones, donde cada uno de los 99 registros contiene : Cód. de infracción, Importe de la misma, Código ('H'=habilitado, 'I'=Inhabilitado).

    Se pide :

    1. Grabe los registros con error de patente o código de infracción detectados en el proceso (imagen de b).
    2. Imprima la patente y cantidad de infracciones cometidas por colectivo (solo sí tienen > 10 infracciones), con el siguiente diseño :

       Listado de Infracciones de colectivos con más de 10 infracciones

Patente	**Infracciones**
XXX999	999

    3. Para cada línea que haya tenido colectivos con infracción, el siguiente listado, ordenado creciente por número de línea :

       Linea 999

Dia	**Importe**
99	9999.99

22. Una compañía de seguros requiere controlar los pagos mensuales de las pólizas de sus clientes. Para ello cuenta con :

    a) Un archivo de **Pólizas** (**ordenado** por póliza), donde c/registro contiene : Nro.de Póliza (8 díg.), Nro.de Cliente (8 díg.), Rubro asegurado (3 díg.), Importe de la cuota mensual.

    b) Un archivo de **Clientes** (no más de 7000), donde cada registro contiene : Nro.de Cliente, Apellido y Nombre (40 car.), Domicilio (50 car.).

    c) Un archivo de **Pagos** (sin orden), con los pagos mensuales hechos en el año, donde c/registro contiene : Nro.de Póliza, Nro.de mes pagado (1..12), Importe pagado.

    Se pide :

    1. Imprima el siguiente listado ordenado por cliente y póliza, de las pólizas que tengan una o más cuotas impagas, según se indica :

       Cliente : 99999999 **Apellido y Nombre: X(40)**

       ----------Meses Pagados----------

Póliza	**Rubro**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
99999999	999	P	P	A	P	P	P	P	A	A	P	P	A
99999999	999	A	P	P	A	P	P	A	A	P	P	A	P

    2. Imprima un listado ordenado por rubro, de la recaudación real y prevista por rubro :

       Recaudación por rubro

Rubro	**Real**	**Prevista**
999	999999.99	999999.99

    Nota : Se disponen de 128 Kbytes para estructuras estáticas. Espacio en disco: FileSize(Pólizas) * 16 bytes.

Unidad V — Capítulo I

Unidad V – Capítulo I - Punteros

Los punteros son objetos cuyos contenidos contienen una dirección en la memoria interna de la computadora. Si bien apuntan a una dirección de memoria, para que 2 punteros puedan ser comparados, o asignar la dirección que contiene una variable puntero a otra, **deben ser ambos del mismo tipo de dato al que apuntan**, para que no se genere un error de incompatibilidad de tipo. Por lo tanto, los punteros en C/C++ son **punteros a un tipo de dato** específico. No obstante, existe la posibilidad de poder definir punteros genéricos por medio del identificador ***void***.

MAPA DE MEMORIA

A continuación se pasará a detallar como divide el lenguaje C/C++ las regiones de memoria interna, al cargarse un programa.

Mapa de memoria C/C++

La memoria interna está dividida en segmentos, en donde **cada segmento tiene un tamaño de 64Kbytes**. Algunos de estos segmentos cumplen un rol específico a la hora

de correr o ejecutar un programa. Estos segmentos especiales son 3 más una región de memoria restante que representan lo siguiente:

1. **Segmento de DATOS –DATA-**: El lenguaje C/C++ utiliza este segmento para reservar espacio para las variables de ámbito o **alcance global**. Las variables definidas en este segmento son **variables estáticas** y deben ser definidas en **tiempo de compilación**, ocuparán el espacio reservado mientras la ejecución del programa está en marcha. Todas las variables del módulo principal más todas las variables de las unidades empleadas por el módulo principal ocuparán el único segmento de datos, que no podrá exceder los 64Kbytes. No obstante, existen otros modelos de memoria, el cual permiten ampliar el área destinada para el almacenamiento de variables estáticas. Los cuales se verán más adelante.
2. **Segmento de CÓDIGO -CODE-**: El lenguaje C/C++ utiliza este segmento para el código del programa (bloque principal + módulos definidos), pero el tamaño total no está limitado a un segmento de 64Kbytes. La manera de ampliar el tamaño de código por encima de los 64Kbytes es empleando los diferentes modelos de memoria, que se detallarán más adelante.
3. **Segmento de la PILA –STACK-**: El lenguaje C/C++ utiliza este segmento para guardar la dirección de retorno al invocar a un módulo, los parámetros y las variables de alcance o ámbito local. El stack crece desde dirección de memoria más alta hacia direcciones de memoria más bajas.
4. **Área de memoria del MONTÍCULO –HEAP-**: El lenguaje C/C++ utiliza esta región de memoria, en realidad son varios segmentos, para la creación de *variables dinámicas*, esto es, **variables que se crearán y se eliminarán cuando ya no se las necesite, en tiempo de ejecución**. Esto permite ampliar los 64Kbytes del segmento de datos; o bien aquellos otros modelos de memoria que permiten una mayor espacio para los datos, por lo tanto, está región de memoria estará disponible para poder extender la cantidad de variables en caso de requerir más nuestro proceso en ejecución. El heap crece desde una dirección de memoria baja hacia direcciones de memoria más altas.

Si se divide el bloque principal en módulos podremos ahorrar el espacio del segmento de datos, debido a que ciertas variables de trabajo temporal estarían definidas dentro de cada módulo ubicándose en el segmento de la pila o stack y dejarían de estar ubicadas en el segmento de datos. De esta manera se consigue aliviar para que no recaigan todas las variables al segmento de datos, sino que se distribuyan en el segmento de la pila. Esto es cada vez que se invoca a un módulo tanto los parámetros como las variables locales se guardan en la pila, es decir se crean en el momento de invocar al módulo y son destruidas al abandonarlo.

Segmento de Datos **Máximo 64Kbytes.** **Variables globales** Son variables estáticas, ya que se definen en tiempo de compilación Incluye las variables definidas en el bloque principal más las variables definidas en c/u. de las unidades mencionadas. Puede ser ampliado utilizando otros modelos de memoria que se verán más adelante.	**Segmento de la Pila –*Stack*-** **Máximo 64Kbytes.** **Turbo Pascal lo fija en 16Kbytes.** **Puede ampliarse con la directiva al compilador $M** **Parámetros formales y Variables locales** Se crean al invocar a un módulo y son destruidas al salir de él.
Segmento de Código **Máximo 64Kbytes.** Puede ampliarse utilizando unidades. Cada unidad emplea su propio segmento de código con un máximo de 64Kbytes. Puede ser ampliado utilizando otros modelos de memoria que se verán más adelante.	**Área del Montículo –Heap-** **Resto de la memoria disponible.** **Variables dinámicas** sin nombre propio Se accede en forma indirecta a través de variables de tipo puntero. Se crean y eliminan en tiempo de ejecución.

Punteros

Son variables cuyo contenido es una dirección de memoria. Puede estar constituido por dos partes, una indicando el segmento (Seg) y otra indicando el desplazamiento (Ofs), en un modelo de memoria Segmentada.

Puntero Cercano

(NEAR): Es aquel puntero que puede apuntar solamente al segmento al que pertenece, es decir, solo basta con indicar el desplazamiento dentro de un segmento. Por lo tanto, un puntero cercano solo pesa la mitad que un puntero lejano, esto es 2 bytes.

Puntero Lejano

(FAR): Es aquel puntero que puede apuntar no solo al segmento al que pertenece, sino, puede apuntar a otros segmentos. Un puntero lejano debe contener tanto el segmento como el desplazamiento, es decir, el doble de un puntero cercano, esto es 4 bytes.

Puntero Externo

Es aquel puntero que permite acceder a una estructura dinámica, fuera de ella.

Puntero Empotrado

Es aquel puntero que permite acceder a otro elemento de una estructura dinámica, dentro de ella.

Puntero Colgado o Desbocado

Es aquel puntero que no se sabe a ciencia cierta donde apunta, generalmente en estos casos puede provocar efectos colaterales adversos, como colgarse el proceso en ejecución, modificar el contenido de variables, entre otras cosas.

Puntero indefinido

Es aquel puntero que luego de una eliminación de una variable dinámica el puntero asociado a esa acción quedó indefinido (?).

Puntero Nulo

Es aquel puntero que no apunta a una posición concreta o a ningún objeto, en realidad es una dirección de memoria especial y esta posición es la posición cero. El valor puede ser asignado a cualquier tipo de puntero, ya que representa un puntero genérico. Se define la macro **NULL** de alguna de las siguientes formas:

```
#define NULL 0
#define NULL 0L
#define NULL ((void*) 0)
```

Puntero Genérico

Es aquel puntero que no está relacionado con ningún tipo en particular. Este tipo de punteros es más ampliamente utilizado para el soft de base o de sistema, no tanto para programas de aplicaciones.

Puntero a un tipo de dato

Es aquel puntero que apunta a un tipo particular de dato, se lo indica por medio del símbolo * (asterisco). Las variables definidas como puntero a un tipo de dato determinan la cantidad de bytes a que hace referencia el puntero.

Puntero constante

Es aquel puntero que una vez establecida la dirección a donde apunta, esta dirección no podrá ser modificada durante la ejecución del programa.

```
int a;
int *const pi = &a;
*pi = 5; // Modifica el contenido de la dirección apuntada por pi que es la dirección en donde se localiza la variable a.
```

pi = NULL; // Error, la dirección a la que apunta pi, no puede ser modificada o dicho de otra manera, no se puede cambiar el objeto apuntado por pi, por ser un puntero constante.

Puntero a constante

Es aquel puntero que apuntando a un objeto, no puede cambiar su contenido, si podemos cambiar al objeto a que apunta, o dicho de otra manera puede ser modificada la dirección a la que apunta.

int a;

const int *pi = &a;
pi = NULL; // Se modifica el objeto apuntado por pi, puede ser reasignado, en este caso no apunta a ningún objeto.
*pi = 5; // Error, no se permite modificar el contenido del objeto apuntado por pi. Si bien el contenido de la variable a no puede ser modificado a través del puntero pi, si podrá ser modificado por medio de otro identificador u otro medio que no sea pi.

Puntero constante a constante

Es aquel puntero que no puede modificar al objeto al que apunta, como así también no puede cambiar el contenido del objeto al que apunta.

```
int a;
const int *const pi = &a;
*pi = 5; // Error.
pi = NULL; // También es error.
```

Puntero a puntero

Es aquel puntero que apunta a un objeto que es a su vez un puntero a otro objeto. Puntero que apunta a un puntero el cual apunta a un objeto. La notación utilizada para estos casos es utilizar doble asterisco **. Esto puede ser confuso a primera vista, no obstante, se verá que su comprensión es bastante simple y no ofrece mayores confusiones.

```
int a;
int *pi = &a;
int **pe = &pi;
```

En el ejemplo pi es un puntero que apunta a un objeto de tipo entero, en este caso apunta al objeto variable a.

En cambio, pe es un puntero que apunta a un objeto de tipo puntero a entero, el que a su vez apunta a otro objeto de tipo entero la variable a.

Ejemplos:

int *pe;
char *pc;
float *pf;
sArt *ps;

& Operador de dirección

El ampersand <&> es un operador monario, que se aplica a un objeto como ser una variable estática que obtiene la dirección en donde está ubicada esa variable en la memoria RAM. La posición de la memoria será una región o área, como ser; de Datos, de Código, del Stack <Pila>, o del HEAP <Montículo>, que fuera analizadas anteriormente. De la misma manera que el operador unario new regresa la dirección de

memoria en el heap, en el instante cuando se le solicitó reservar una ubicación de memoria, el operador & también regresa una posición de memoria del objeto indicado a su derecha. La posición de memoria podemos considerarla como un número compuesto como lo es la hora habitual o tradicional, formado por dos componente, siendo estas, la hora y los minutos, si no tomamos en cuenta los segundos o milésimas de segundos, o considerada como un número simple, en este caso lo indicamos como la hora militar, esta se forma de la unión o yuxtapoción de la hora concatenada con los minutos.

Ejemplos:

20:30 es la hora tradicional y se lee como las veinte horas, treinta minutos.
2030 es la hora militar y se lee como las dos mil treinta.

De la misma manera podemos pensar que la memoria RAM cada byte de ella posee una dirección de memoria y podrá ser vista como un número compuesto o bien como un número simple. En el caso que lo veamos como un número compuesto las dos componentes que lo conforman sería el Segmento <Seg> y el Desplazamiento <Ofs> equivalente a la hora compuesta de hora y minuto. Si en cambio lo vemos como un número simple entonces concatenamos al Segmento el Desplazamiento a su derecha. La forma habitual de indicar las direcciones de memoria se utiliza el sistema Hexagecimal.

Ejemplo:

FA2C:3B5D
en donde la primer componente es el Segmento y la segunda el Desplazamiento.
FA2C3B5D es el modelo de memoria plano FLAT y, es equivalente a la hora militar.

Operador * desreferencia

El asterisco <*> es un operador monario e indica la desreferencia del objeto que se indica a su derecha, en otras palabras alude al contenido del objeto al que apunta.

Variables estáticas y variables dinámicas

```
short a,
        *p,
         *q;

   a = 5;
   p = &a;
   *p = *p + 3;
   cout << “a: “ << a << endl; // emite: 8
```

```
cout << "*p: " << *p << endl; // emite: 8
p = new short;
*p = a;
*p = *p – 3;
cout << "a: " << a << endl; // emite: 8
cout << "*p: " << *p << endl; // emite: 5
p = &a; // la variable puntero p deja de apuntar a la variable dinámica. Ops!
```

Antes de hacer esto debemos asegurarnos de no perder esa referencia a la variable dinámica, entonces lo que debemos realizar es hacer una copia de la dirección de donde apunta p en otra variable puntero, copiar entonces esta dirección a la variable dinámica.

```
q = p;
```

luego hacemos

```
p = &a;
```

ahora no hemos perdido la variable dinámica por que es apuntada por q, mientras p ahora está apuntado a la variable estática a.
si queremos eliminar la variable dinámica apuntada por q debemos hacer:

```
delete q;
```

luego de hacer esta acción q quedrá indefinido <?>, hacer una referencia al contenido de q será una acción incorrecta. Además q solo debe apuntar al HEAP, para que la acción del operador delete funcione correctamente, también delete devuelve la cantidad de bytes que new le restó al momento de solicitar memoria para una nueva variable dinámica.

Punto importante a tomar en cuenta

LAS VARIABLES DINÁMICAS NO TIENEN UN NOMBRE QUE LA IDENTIFIQUE, solo se puede acceder a través de una variable puntero que contenga la dirección de ubicación en la memoria RAM de esa variable dinámica. Una variable estática se puede acceder de dos maneras, una por medio de su propio nombre y otra por medio de una variable puntero que apunta a la dirección de memoria de esa variable estática.

Modelos de Memoria

Los distintos modelos de memoria establecen el alcance de los punteros, para acceder, tanto a los datos como así también al código. Si un modelo de memoria establece puntero cercano indica que ese puntero solo puede apuntar a posiciones de memoria que indican el desplazamiento dentro de un bloque o segmento de memoria, en

cambio, si el modelo de memoria establece puntero lejano, este establece que el puntero puede apuntar no solamente al bloque o segemento en que pertenece el puntero, sino, que además podrá apuntar a otros bloques o segmento de memoria al que no pertenece. Si consideramos que el espacio de almacenamiento para un puntero es de 4 bytes o 32 bits, se podrían direccionar hasta un máximo de 4294967296 bytes, con posiciones desde 0 hasta 4294967295, es decir, un alcance de 4Gb. En este caso decimos que un puntero que pesa en memoria 4 bytes puede direccionar 4Gbytes de memoria interna o de la RAM, siendo esta la capacidad máxima y un puntero que pueda direccionar a toda esta capacidad de memoria es un puntero lejano FAR. En cambio, si un puntero solo tiene un alcance dentro de su propio espacio en donde fue definido, el peso de este tipo de puntero será de 2 bytes o 16 bits, se podrían direccionar hasta un máximo de 65536 bytres, con posiciones desde 0 hasta 65535, es decir, un alcance de 64Kb. En este caso decimos que un puntero que pesa en memoria 2 bytes puede direcciona 4Kbytes de momoria interna o de la RAM, siendo esta la capacidad de un bloque de memoria o segmento, y un puntero que solo puede direccionar un bloque de memoria o segmento solamente es un puntero cercano NEAR.

TINY (diminuto)

Establece para el código, datos y pila todo en un solo segmento de 64Kb. Por lo tanto, emplea punteros cercanos, para código y para datos. Este tipo de programa puede ser convertido a .COM, el cual tiene la virtud de ser los programas ejecutables más rápidos, por el hecho de que utiliza punteros cercanos, siendo su aritmética de punteros más simple en los cálculos que los punteros lejanos, ya que, solo debe considerar el desplazamiento. La desventaja es que su tamaño tanto para código como para datos no puede superar los 64Kb. de espacio disponible para todo ello.

SMALL (pequeño)

Establece para el código un máximo de 64Kb, y para datos otros 64Kb. como máximo, por lo que el tamaño del programa puede alcanzar los 128 Kb. No obstante, en ambos casos utiliza punteros cercanos (código y datos), siendo tan veloces como en el modelo *tiny*.

COMPACT (compacto)

Establece para el código un máximo de 64Kb, por lo que utiliza puntero cercano, mientras que para datos puede utilizar varios segmentos c/u. de los cuales tendrá un máximo de 64 Kb, en este caso se utiliza punteros lejanos, por lo que, localizar una instrucción será muy rápido, pero para localizar un datos será un tanto más lento. La ventaja de este modelo es que permite disponer de muchos datos en memoria.

MEDIUM (medio)

Establece para el código varios segmentos c/u. de los cuales tendrá un máximo de 64Kb., mientras que para datos solamente un segmento máximo de 64Kb. Es a la inversa del modelo *compact*, por lo que utiliza punteros lejanos para el código, más lento para localizar una instrucción, y puntero cercano para datos, más rápido para localizar un dato. La ventaja de este modelo es que permite disponer de muchas instrucciones en memoria.

LARGE (grande)

Establece para el código varios segmentos c/u. de los cuales tendrá un máximo de 64Kb, como así también varios segmentos para datos, c/u. de los cuales tendrá un máximo de 64Kb. Por lo que, erá más lento para localizar tanto una instrucción como un dato. La ventaja es que da un tamaño mucho más extenso tanto para el código como para los datos. Aún así no se permiten estructuras de datos que superen los 64Kb.

HUGE (enorme)

Establece para el código varios segmentos c/u. de los cuales tendrá un máximo de 64Kb, como así también varios segmentos para datos, c/u. de los cuales tendrá un máximo de 64Kb. Por lo que, erá más lento para localizar tanto una instrucción como un dato. La ventaja es que da un tamaño mucho más extenso tanto para el código como para los datos. A diferencia del modelo *large*, sí se permiten estructuras de datos que superen los 64Kb.

FLAT (plano)

Este modelo de memoria permite adaptarse a la arquitectura de los modernas computadoras. A partir del 80x386 (1985) 32 bits para registros y direcciones hasta 4Gb modo segmentado.

NODO

El nodo es una entidad o unidad de información, constituido por al menos dos campos, uno de ellos establece la información y la otra componente indica un puntero empotrado, el cual apuntará a un nodo anterior o en tal caso a un nodo siguiente. Por lo tanto, un nodo presenta la forma de un registro o struct según la terminología empleada por C / C++. El campo de tipo puntero puede no apuntar a nada, en tal caso estará indicando que no hay nodo anterior o en tal caso no hay nodo siguiente. La representación gráfica de un nodo queda establecida de la siguiente forma:

Nodo que apunta a su predecesor, si no existe entonces Ant = NULL.

Info	Ant

Nodo que apunta al sucesor, si no existe entonces Sgte = NULL.

Info	Sgte

NULL

Es un valor especial de dirección de memoria, generalmente esa dirección es el primer byte de la memoria, la dirección cero. Podemos comparar un puntero para saber si no apunta a nada o por el contrario si apunta a alguna dirección distinta a NULL. Normalmente cuando inicializamos un puntero a una estructura dinámica de datos, le asignamos el valor NULL al puntero externo a dicha estructura. Si un puntero p, tiene

un valor NULL, será error querer determinar el contenido de la dirección apuntada por el puntero p. El valor NULL puede ser asignado a cualquier puntero que apunte a cualquier tipo de dato.

Ejemplos

```
int *pe;
tpPCL *pDin;

pe = NULL;
pDin = NULL;
pDin == NULL;
pDin != NULL;
```

Palabras reservadas que crean y eliminan variables dinámicas

Operador new

Crea una nueva variable dinámica en el heap. Busca una zona de memoria contigua de acuerdo al tamaño del tipo de dato indicado inmediatamente a su derecha del operador. Retorna un puntero al primer byte de ese conjunto de bytes, en caso de haber encontrado la cantidad indicada en forma contigua, retorna un puntero nulo. En la medida que se reserve memoria, la cantidad de memoria en el heap se irá reduciendo su capacidad. Para no perder el puntero retornado se deberá asignarlo a una variable de tipo puntero al tipo de dato que se corresponde al mismo tipo indicado a continuación del operador new. Si a un puntero se le modifica la dirección por otra diferente, no tendremos más el acceso a la variable a la que antes apuntaba, antes de hacer esto deberíamos realizar una copia de esa dirección en otra variable puntero. Las variables dinámicas creadas con new **no tienen nombre**, por lo que la única manera de poder acceder a su espacio de almacenamiento es a través de variables puntero.

La sintaxis es

Forma (1) : permite crear una variable simple o estructurada, pero no arreglos.

```
new tipo;
new tipo (inicializar);
new (tipo);
new (tipo) (inicializar);
```

Forma (2) : permite crear un arreglo o disposición u ocurrencia de elementos.

```
new tipo [n];
new (tipo) [n];
```

Ejemplo Forma (1)

```
int *p;

p = new int;
p = new (int);
p = new int (7); // asigna el valor 7 a la dirección apuntada por p.
p = new (int) (9);
```

Ejemplo Forma (2)

```
p = new int [10];
```

Crea un espacio para 10 elementos de tipo int cada uno, la dirección retornada por new es la dirección del primer byte del conjunto. Si el sizeof(int) = 4, entonces 4bytes * 10 componentes = 40 bytes, el byte de la extrema izquierda de ese conjunto es la dirección retornada por new.

Operador delete

Elimina una variable dinámica en la dirección indicada por el puntero indicado por delete, el cual debe apuntar al heap o montículo que es el área de la memoria RAM utilizada para las variables dinámicas. El espacio liberado es restituido al heap.

```
delete p;
delete [ ] p;
```

Uniones

Las uniones permiten que un área de memoria sea compartida por diferentes tipos de datos. Su forma es similar a la de un struct, aunque su comportamiento es diferente.

La sintaxis es:

```
union idTipo {
  idTipo nom;
  idTipo nom;

} idVar;
```

Ejemplos

```
union tipoU {
  char c;
  int    i;
  float f;
} varU;

varU.c = 'A';
varU.i;
varU.f;
```

son miembros de la unión tipoU. La asignación del carácter 'A' al miembro varU.c repercute en los otros miembros, por lo que no es posible asignar valores diferentes a sus miembros, como lo es en un struct.

```
union mix_t {
  int l;
  struct {
    short hi;
    short lo;
    } s;
  char c[4];
} mix;
```

Los miembros de la unión se solapan unos con otros, y el área utilizada es el de mayor de sus miembros.

Unidad V Capítulo II

Unidad V – Capítulo II - Estructuras Dinámicas de Datos

Estructuras Dinámicas de Datos

Las estructuras dinámicas de datos se las clasifica en dos grupos; lineales y no lineales. Dentro de las estructuras lineales hay con restricciones en cuanto a su manejo y sin restricciones en cuanto a su manejo.

1. Estructuras lineales
   1.1. Con restricciones
      1.1.1. Pilas (L.I.F.O.)
      1.1.2. Colas (F.I.F.O.)
      1.1.3. Colas circulares.
   1.2. Sin restricciones
      1.2.1. Listas (Freedom)
      1.2.2. Listas circulares.
      1.2.3. Listas doblemente enlazadas.
2. Estructuras no lineales
   2.1. Grafos
   2.2. Árboles
      2.2.1. Árboles binarios.
         2.2.1.1. Árboles binarios de búsqueda ABB.
         2.2.1.2. Árboles AVL.
      2.2.2. Árboles multicamino.
         2.2.2.1. Árboles B.
         2.2.2.2. Árboles B+.
         2.2.2.3. Árboles B*

Estr. Datos	Restricciones	Prohibido	Movimiento	Ptr. externo
Pila	SI	Recorrer los elementos	L.I.F.O.	1
Cola	SI		F.I.F.O.	2
Lista	NO		Freedom	1

Con restricciones

Pila (Stack) –L.I.F.O.-

Una pila es una estructura de datos en la cual los elementos se colocan por un punto llamado **tope** o **cima** y se quitan elementos por el mismo punto tope o cima. Por lo que decimos que una pila obedece al tratamiento de "último en entrar, es el primero en salir", esta frase se conoce con las siglas en inglés **L.I.F.O.**, es decir," Last Input First Output". Para poder acceder a los elementos en la Pila, se requiere un único puntero externo, el cual siempre apunta al tope o cina de la Pila. Antes de incorporar un elemento, debe inicializarse en NULL el puntero externo a la Pila. Cada elemento de la Pila, es decir, un nodo, posee un puntero empotrado, que apunta al elemento precedente, salvo el primer nodo incorporado a la Pila que apuntará a NULL. Las operaciones a realizar en la Pila entre otros son; **Push** (Meter), y **Pop** (Sacar). Si incorporamos los elementos, 5, 7 y 3; al quitarlos el orden sería, 3,7,5. Concluimos diciendo que una pila **invierte** el orden de ingreso de sus elementos. De lo observado concluimos diciendo que está prohibido recorrer los elementos internas de una pila.
Ejemplos de pilas de la vida diaria podrían ser: pila de platos, de ropa, de cartas.
Ejemplos de manejo de una pila en sistemas serían: cuando el **bloque principal** invoca a un módulo **Mod1**, y éste invoca a otro módulo digamos, **Mod2**, luego, cuando el módulo Mod2 finalice debe retornar al punto de su llamada o invocación, lo mismo cuando finalice el módulo Mod1, debe regresar al punto de su llamada o invocación, pues bien, ¿cómo se logra esto?, guardando en una pila la dirección del punto de la llamada del bloque principal, luego se guarda en la misma pila la dirección del punto de la llamada del módulo Mod1, luego al finalizar el módulo Mod2, se retira de la pila la dirección que está en su tope, sabiendo que debe regresar al módulo Mod1, y cuando finalice el módulo Mod1, retira de la pila la dirección que está en su tope, sabiendo que debe regresar al bloque principal. Otro ejemplo de manejo de pila es cuando se debe analizar una expresión aritmética, a efectos de transformar una expresión *infija* (los operadores aparecen entre los operandos) en notación *prefija* o *posfija* (los operadores aparecen antes de sus operandos o después de sus operandos respectivamente). Esta forma de notación es más adecuada para realizar las operaciones por parte de la computadora, ya que las operaciones se van realizando en el orden que aparecen, por lo que evita el uso de paréntesis. Estos tipos de notaciones se deben a los trabajos realizados por el matemático polaco *Lukasiewicsz*.

Ejemplos

a+b {notación *infija*}
+ab {notación *prefija*}
ab+ {notación *posfija*}

Funciones para la Pila

```
/*
        Id.Programa: Pila.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: Jul-2005 / Jun-2014
        Comentario.: Estructura dinamica de tipo Pila (=Stack) -L.I.F.O.-
            Modulos:
                Push - Meter
                Pop  - Sacar
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct sNodo *tPila;
struct sNodo {
  tInfo info;
  tPila ant;
};
```

```
void Push(tPila &Stack, tInfo valor) {
  tPila pNodo;

  pNodo = new sNodo;
  pNodo->info = valor;
  pNodo->ant = Stack;
  Stack = pNodo;
} //Push
```

```
void Pop(tPila &Stack, tInfo &valor) {
  tPila pElim;

  pElim = Stack;
  valor = Stack->info;
  Stack = Stack->ant;
  delete pElim;
} //Pop
```

```
int main () {
  tInfo   num;
```

```
 tPila Pila;

 clrscr();
 Pila = NULL;
 for(int i = 1; i <= 3; i++) {
   cout << "Ing.valor: ";
   cin >> num;
   Push(Pila, num);
 }
 cout << "-----------x----------" << endl;
 while(Pila) {
   Pop(Pila, num);
   cout << "valor: " << num << endl;
 }
   return 0;
} //main
```

Cola (Queue) – F.I.F.O.-

Una cola es una estructura de datos en la cual los elementos se colocan por un punto llamado fin y se quitan elementos por otro punto llamado frente. Por lo que decimos que una cola obedece al tratamiento de "primero en entrar, es el primero en salir", esta frase se conoce con las siglas en inglés **F.I.F.O.**, es decir, First Input First Output. Para poder acceder a los elemento en la Cola, se requieren dos punteros externos, uno que apunta al frente de la Cola y otro que apunta al final de la Cola. Cuando incorporemos un nuevo elemento, ingresamos a la Cola con el puntero externo que apunta al final, ya que un nuevo elemento será ubicado como último. En cambio, si debemos quitar un elemento en la Cola, será el que se ubica al frente de la misma, y en estos casos ingresaremos con el puntero externo que apunta al frente en la Cola. Antes de incorporar elementos en la Cola, ambos punteros externos deben ser inicializados en NULL. Cada nodo de la Cola contiene un puntero empotrado el cual apunta al próximo o siguiente nodo si existe, sino, apuntará a NULL. Las operaciones a realiar en la Cola entre otros son: **Enqueue** (Encolar o Agregar), y **Dequeue** (Desencolar o Suprimir). Si incorporamos los elemento, 5, 7 y 3; al quitarlos el orden sería, 5,7,3. Concluimos diciendo que una cola mantiene el orden de ingreso de sus elementos.
Ejemplos de manejo de una cola en la vida diaria se presentan en diversas situaciones, como ser: cola en un supermercado para abonar las compras, cola en un banco a ser atendido por ventanilla, cola en una parada de transporte público, cola en una sala de guardia en un hospital, cola para realizar trámites.
Ejemplo de manejo de una cola en sistemas sería cuando un proceso **Proc1** requiere el uso de un dispositivo, como es la impresora. Ante el requerimiento por parte del Proc1 se informa que el dispositivo está ocupada (Busy), debido a que un proceso previo Proc0 se le asignó dicho dispositivo. En lugar de permanecer ocioso el Proc1 en espera de que finalice el proceso Proc0, su salida es reencauzada o redireccionada a otro

dispositivo en disco. El proceso Proc1 considera que la salida está siendo enviada a la impresora, cuando en realidad se está generando un archivo en disco. El que se interpuso en esta acción es un servicio del Sistema Operativo, a través del S.P.O.O.L. (Simultaneous Peripherical Operation On Line - Operaciones Simultáneas de Periféricos En Linea), reencauzando la salida hacia el disco, si otras tareas o procesos requieren de la impresora, éstos otros procesos su salida también serían reencauzadas al disco, formando entonces una cola de espera. En la medida que la impresora quede disponible, el SPOOL tomará el primer archivo que se encuentre en la cola asignándole la impresora, y así sucesivamente. Cada proceso que finalice la impresión el propio SPOOL eliminará el archivo.

Funciones para la Cola

```
/*
        Id.Programa: Cola.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: Jul-2005 / Jun-2014
        Comentario.: Estructura dinamica de tipo Cola (=Queue) -F.I.F.O.-
            Modulos:
                Enqueue - Agregar
                Dequeue - Suprimir
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tCola;
struct sNodo {
  tInfo info;
  tCola sgte;
};
```

```
void Enqueue(tCola &qFte, tCola &qFin, tInfo valor) {
  tCola pNodo;

  pNodo = new sNodo;
  pNodo->info = valor;
  pNodo->sgte = NULL;
  if (!qFte)
    qFte = pNodo;
  else
    qFin->sgte = pNodo;
```

```
  qFin = pNodo;
} //Enqueue
```

```
void Dequeue(tCola &qFte, tCola &qFin, tInfo &valor) {
  tCola pElim;

  pElim = qFte;
  valor = qFte->info;
  qFte = qFte->sgte;
  if (!qFte)
    qFin = NULL;
  delete pElim;
} //Dequeue
```

```
int main () {
  tInfo   num;
  tCola  ColaFte,
         ColaFin;

  ColaFte = ColaFin = NULL;
  for(int i = 1; i <= 3; i++) {
     cout << "Ing.valor: ";
     cin >> num;
     Enqueue(ColaFte, ColaFin, num);
  }
  cout << "-----------x----------" << endl;
  while(ColaFte) {
     Dequeue(ColaFte, ColaFin, num);
     cout << "valor: " << num << endl;
  }
  return 0;
} //main
```

Sin restricciones

Lista (List) –Freedom-

Una lista esta compuesta por una serie de elementos, como por ejemplo una lista de personas, o animales o cosas. En general, trabajaremos con listas ordenadas por el valor de un campo clave o llave.

Las estructuras dinámicas de datos del tipo lista **no presentan** ningún tipo de **restricciones** en cuanto a su manejo. Esto es, cada nuevo elemento que se incorpore a la lista puede realizarse por cualquier parte, y cada elemento que se retire de la lista puede efectuarse tambien por cualquier parte. Un único puntero externo estará apuntando siempre al primer elemento que se encuentre en la lista en un momento determinado. La

primer acción que debemos realizar con una lista será asignar el valor **NULL** al puntero externo a la lista. El valor NULL es un valor genérico para que un puntero no apunte a nada, en realidad le asigna la dirección de segmento = 0 y desplazamiento = 0. Es decir, si Lista es el puntero externo a la lista, hacer Lista = NULL, es hacer que Lista no apunte a nada. El lenguaje lo indica de la siguiente manera: #define NULL 0. Por lo tanto, a una variable puntero se le puede asignar indistintamente el valor NULL o bien el valor cero.

Ejemplos:

```
short p = NULL,
      q = 0;
```

En general un elemento de la lista estará constituido por 2 componentes. Una componente indicará la información de ese elemento y la otra componente un puntero **empotrado** el cual apuntará al elemento siguiente ubicado en la lista. Por lo tanto el puntero externo a la lista y cada uno de los punteros empotrados apuntan a una misma estructura de datos del tipo Lista.

Las operaciones a realizar en una Lista son más que en las listas con restricciones de Pila y Cola; entre ellos encontramos los siguientes procesos: InsertaNodo, InsertaInicio, InsertaEnMedio, SuprimeNodo, SacarPrimerNodo, ExisteNodo, CrearNodo.

La estructura idónea para representar un elemento de la lista es el tipo registro que tendrá básicamente dos campos, un campo **INFO** que contiene un valor de información y un campo **SGTE** que contendrá la dirección del elemento siguiente en la lista, salvo que sea el último en cuyo caso contendrá el valor NULL.

Funciones para la Lista

```
/*
        Id.Programa: Lista.cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: Jul-2005 / Jun-2014
        Comentario.: Estructura dinamica de tipo Lista (=List) ordenada -Freedom-.
          Modulos:
            InsertaNodo    "Modulo despachador"
            InsertaInicio  "Si Lista es Vacia Inserta primero sino por el frente o inicio"
            InsertaEnMedio "Inserta despues del primero, no antes, incluye por final"
            SuprimeNodo    "Elimina un nodo, en cualquier punto de la Lista, sino enc.
no."
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
```

```
typedef struct tNodo *tLista;
struct sNodo {
  tInfo info;
  tLista sgte;
};
```

```
void InsertaInicio(tLista &List, tInfo valor) {
  tLista pNvo;

  pNvo = new sNodo;
  pNvo->info = valor;
  pNvo->sgte = List;
  List = pNodo;
} //InsertaInicio
```

```
void InsertaEnMedio(tLista &List, tInfo valor) {
  tLista   pNodo,
           pAct;

  pAct = List;
  while (pAct->sgte && valor > pAct->sgte->Info)
     pAct = pAct->sgte;
  pNvo = new sNodo;
  pNvo->info = valor;
  pNvo->sgte = pAct->sgte;
  pAct->sgte = pNvo;
} //InsertaEnMedio
```

```
void InsertaNodo(tLista &List, tInfo valor) {
  if (!List || valor < List->info)
     InsertaInicio(List, valor);
  else
     InsertaEnMedio(List, valor);
} //InsertaNodo
```

```
void SuprimeNodo(tLista &List, tInfo valor) {
  tLista   pAct,
           pAnt;

  pAct = List;
  pAnt = NULL;
  while (pAct && valor > pAct->info) {
     pAnt = pAct;
```

```
    pAct = pAct->sgte;
  }
  if (pAct && valor == pAct->info) {
    if (pAnt)
      pAnt->sgte = pAct->sgte;
    else
      List = pAct->sgte;
    delete pAct;
  }
} //SuprimeNodo
```

```
int main () {
  tInfo   num;
  tLista  Lista;

  clrscr();
  Lista = NULL;
  for(int i = 1; i <= 7; i++) {
    cout << "Ing. valor: ";
    cin >> num;
    InsertaNodo(Lista, num);
  }
  cout << "-----------x----------" << endl;
  while (Lista) {
    cout << “Ing. valor entero: “;
    cin >> num;
    SuprimeNodo(Lista, num);
    cout << "valor: " << num << endl;
  }
  return 0;
} //main
```

Los procesos básicos para el manejo de la estructura del tipo lista son:

1. void CrearLista(tLista &Lista); o en su lugar Lista:= NULL
2. void InsertaNodo(tLista &Lista, tInfo valor);
3. void InsertaPrimero(tLista &Lista, tInfo valor);
4. void InsertaDelante(tLista &Lista, tInfo valor);
5. void InsertaInicio(tLista &Lista, tInfo valor); // reemplaza Inserta Primero Delante.
6. void InsertaEnMedio(tLista &Lista, tInfo valor);
7. void SuprimeNodo(tLista &Lista, tInfo &valor);
8. bool ListaVacia(tLista &Lista); o en su lugar preguntar si Lista == NULL
9. bool ListaLlena(uint MagnitudNodo);

El módulo *InsertaPrimero* es invocado cuando el puntero a la Lista es NULL y esto sucede una vez, en cambio el módulo *InsertaDelante* es invocado cuando un nuevo nodo se incorpora a la lista al inicio, debido a que el valor del campo clave es menor al valor del primer nodo en la lista. Solo una sentencia difiere en ambos módulos, esto es, cuando se asigna al campo sgte del nuevo nodo el valor NULL o el valor del puntero externo de Lista, pero tanto en un caso como en el otro se podría asignar el valor del puntero externo de Lista, debido a que cuando la lista está vacía se asignará el valor NULL y cuando ya exista algún nodo en la lista se le asignará la dirección del nodo que está siendo apuntado por el puntero externo Lista. Por lo tanto, se reemplazarán los módulos *InsertaPrimero* e *InsertaDelante* por el módulo **InsertaInicio(tLista &Lista, tInfo valor);** en el cual la sentencia ptr->sgte = NULL o ptr->sgte.Lista establecida en los módulos InsertaPrimero e InsertaDelante respectivamente se reemplará por ptr->sgte.Lista.

El módulo **InsertaNodo(Lista, valor);** también se reemplazará por otras sentencias. En lugar de realizar tres decisiones solo serán necesarias tomar dos decisiones, en la cual, la primera decisión tendrá una condición compuesta, siendo ellas (Lista == NULL) || (valor < info->Lista), en los casos en que sea verdadero se invocará al módulo InsertaInicio y en caso contrario, se invocará a InsertaEnMedio.

A continuación se presenta un proceso que ingresa cinco elementos a la lista manteniendo siempre un orden por el valor del campo info en forma ascendente y posteriormente los retira.

```
tInfo x;
tLista Lista = NULL;
for (ushort i = 1, i <= 5, i++) {
   cout << "Ing. valor: ";
   cin >> x; //20, 50, 10, 30, 70
   InsertaNodo(Lista,x)
};
while (Lista != NULL) {
   cout << "Ing. valor: ";
   cin >> x; //50, 10, 70, 20, 30
   SuprimeNodo(Lista,x);
  cout << x << endl;
};
```

Los módulos presentados anteriormente son solo un modelo que resuelven algunas de las situaciones que podrán presentarse en diferentes procesos. Por ejemplo, los procesos anteriormente indicados ordenan una lista en forma creciente, siempre genera una nueva variable dinámica al invocarlos, y en casos de repetición del campo clave, se inserta como segundo nodo si el valor de la clave que se repite es el valor del primer nodo de la lista, pero se incorporará delante de los valores de las otras claves correspondientes. Tal vez este comportamiento sea el esperado para ciertos procesos pero no para otros. En ciertos casos algunos ligeros cambios modificarán ese

comportamiento y en otros casos debamos realizar modificaciones más drásticas. Hay situaciones en la cual no queremos repetir el valor de un campo clave, lo que hacemos es buscar si existe ese valor en la lista, en caso de no encontrarse creamos el nodo, pero si existe no. En el primer caso al no existir el nodo, si invocamos al módulo InsertaNodo, se volverá a recorrer nuevamente la lista hasta el punto de inserción, pero si ya habíamos recorrido la lista, porque no optimizar el proceso enlazando el nuevo nodo con aquellos nodos dentro de su entorno localizados en la búsqueda previa. Si queremos enlazar los nodos en una lista pero ordenada en forma decreciente, ¿qué debemos modificar de los módulos mencionados más arriba?. Si el campo info debemos almacenar más de un valor, entonces deberá ser de un tipo estructurado de datos, como ser del tipo registro y en estos casos también vale la siguiente pregunta ¿cuáles son los cambios a realizar en los módulos indicados precedentemente? Estas y otras cuestiones más podrían presentarse en los diferentes procesos y que deberíamos replantearnos acerca de utilizar o no esos módulos o reemplazarlos por otros.

Para los procesos que requieran no repetir nodos para el valor de una misma clave, a continuación se presentan dos módulos adicionales a los establecidos previamente:

1. tLista ExisteNodo(tLista Lista, tInfo valor);

El módulo *ExisteNodo* busca un nodo con un valor en cuyo campo clave coincida con el valor pasado, en ese caso informa la dirección de memoria del nodo con valor igual a valor, caso contrario retorna la dirección del nodo previo o si no existe un valor NULL.

```
tPila ExisteNodo(tLista &Lista, tLista &pAnt, tclv clv) {
   tLista pAct;

   pAnt = NULL;
   pAct = Lista;
   while (pAct && clv > pAct->info) {
      pAnt = pAct;
      pAct = pAct->sgte;
   }
   return iif(pAct && clv == pAct->info, pAct, NULL);
} // ExisteNodo
```

2. **void CrearNodo(tLista &Lista, tLista &ptrNodoAnt, tInfo tInfo valor);**
   2.1.

El módulo *CrearNodo* crea un nuevo nodo enlazado a continuación del nodo apuntado por pNodoAnt si es distinto de NULL, caso contrario lo inserta por el inicio de la lista.

```
void CrearNodo(tLista &Lista, tLista &pNodoAnt, tInfo valor) {
   tLista pNodo;

   if (!pNodoAnt) // si pNodoAnt = NULL
      InsertaInicio(Lista,valor);
   else
      InsertaEnLugar(pNodoAnt, valor);
} // CrearNodo
```

El módulo *InsertaEnLugar* enlaza al nuevo nodo en el entorno donde se detuvo el recorrido de la Lista con el mógulo ExisteNodo, para ello requiere el puntero al nodo previo en el lugar en donde se insertará el nuevo nodo, esto lo logramos hacer con el puntero al nodo previo que es pNodoAnt.

```
void InsertaEnLugar(tLista &pNodoAnt, tInfo valor) {
  tLista pNodo = new sNodo;

  pNodo->info = valor;
  pNodo->sgte = pNodoAnt->sgte;
  pNodoAnt->sgte = pNodo;
} // InsertaEnLugar
```

En los casos de requerir la localización ordinal de un nodo en la lista, se presenta el siguiente módulo *BuscarPosNodo.*

```
tLista BuscarPosNodo(tLista &Lista, int pos) {
        tLista pAct = Lista;

        for (int i = 1; i < pos; i++)
                pAct = pAct->sgte;
        return pAct;
} // BuscarPosNodo
```

Ejemplo

Se tiene una lista, en donde cada nodo representa un día de un mes. Buscar el nodo que se corresponde con un día informado se procederá a recorrer la lista tantos nodos como lo indique el número del día conocido, es decir, si el día dado es el primero la función *BuscarPosNodo* retorna el puntero al primer nodo de la lista, si el día dado es el segundo entonces la función retorna el puntero al segundo nodo y así sucesivamente.

La diferencia con el módulo *ExisteNodo* está dada por el hecho de que en la lista mencionada en el ejemplo anterior no necesitamos guardar el dato del número del día, ya que en la lista existe un nodo por cada día del mes, en cambio cuando no aseguramos encontrar un nodo por cada día de un mes, -según este ejemplo- en estos casos debemos guardar en el nodo el número del día, la función *ExisteNodo* buscará si el nodo visitado en un momento determinado su campo clave coincide con el valor buscado.

A continuación se presentará una estructura dinámica combinada con otra estructura dinámica. En estos casos podemos encontrarnos con una lista de listas o una lista de pilas o una lista de colas. Por un lado tenemos una lista principal y por otro una lista secundaria o sublista o subpila o subcola por cada uno de los nodos de la lista principal. El campo INFO del nodo en la lista principal deberá contar con un sub-campo de tipo puntero el cual apuntará a la sub-estructura de datos dinámica.

Buscar un sub-nodo requerirá primero buscar un nodo en la lista principal y una vez localizado ese nodo, buscar el sub-nodo en la sub-estructura dinámica que cuelga de ese nodo en la lista principal. Gráficamente lo podemos representar de la siguiente manera:

El siguiente ejemplo representa una estructura estática de punteros a estructuras dinámicas de datos y de esta una sub-estructura también dinámica de datos.

Un nodo sin el campo sgte. ¿Es un nodo?.
Un nodo en que el campo Info es de tipo registro.
Un nodo en que el campo Info es un arreglo.
Un nodo con el campo Info que es un puntero a un tipo registro. ¿Cuál sería la utilidad?

Estructuras dinámicas lineales

Cola circular

```
/*
 *****************
 *   Id.Programa: ColaCircular.cpp
 *   Autor......: Lic.Hugo Cuello
 *   Fecha......: sep-2014
 *   Comentario.: Cola Circular
 * ***************
                                *--------------*
   *--------*    *--------*    | tInfo  |tCola|       *---*
   |    | *-|--->|    | *-|--->|nroVuelo|sgte |<------|-* |
   *--^-----*    *--------*    |  4 b.  |4 b. |       *---*
      |                        *-----------|--*  tCola cFin
      *-----------------------------------*
*/
// ----------------- include ---------------
#include <time.h>
#include <conio.h>
#include <fstream>
using namespace std;
// ------------ Fin include ----------------

// ------------ typedef y struct -----------
typedef int tInfo;
typedef struct tNodo *tCola;
struct sNodo {
  tInfo info;
  tCola sgte;
};
// ------------ Fin typedef y struct -------

// ------------ prototipos -----------------
void Enqueue(tCola &, tInfo );
void Dequeue(tCola &, tInfo & );
// ------------ Fin prototipos -------------
```

```
// ------------ funciones ------------------
main() {
  tCola colaFin = NULL;
  int valor;

  cout << 5 << endl;
  Enqueue(colaFin,5);
  cout << 3 << endl;
  Enqueue(colaFin,3);
  cout << 7 << endl;
  Enqueue(colaFin,7);
  cout << "------------------" << endl;
  while (colaFin) {
    Dequeue(colaFin,valor);
    cout << valor << endl;
  }
  getch();
  return 0;
} // main

void Enqueue(tCola &qFin, tInfo valor) {
  tCola pNvo  = new sNodo;

  pNvo->info = valor;
  if (!qFin)
    pNodo->sgte = pNco;
  else {
    pNvo->sgte = qFin->sgte;
    qFin->sgte = pNvo;
  }
  qFin = pNvo;
} //Enqueue

void Dequeue(tCola &qFin, tInfo &valor) {
  tCola pElim;

  pElim = qFin->sgte;
  valor = qFin->sgte->Info;
  if (qFin != qFin->sgte)
    qFin->sgte = qFin->sgte->sgte;
  else
    qFin = NULL;
  delete pElim;
```

```
} //Dequeue
// ------------- Fin funciones -------------
```

Listas doblemente enlazadas

```
/**************
*        Id.Programa: G5Ej00cListaSgteAnt.Cpp
*        Autor......: Lic. Hugo Cuello
*        Fecha......: Jul-2005 / Jun-2014
*        Comentario.: Estructura dinamica de tipo Lista (=List) ordenada -Free-
*            Modulos:
*              InsertaNodoSgte, InsertaNodoAnt
*                "Modulo despachador"
*              InsertaInicioSgte, InsertaInicioAnt
*                "Si ListaFte es Vacia Inserta primero
*                  sino por el frente o inicio" o
*                "Si ListaFin es Vacia Inserta ultimo
*                  sino por el fondo o final
*              InsertaEnMedioSgte, InsertaEnMedioAnt
*                "Inserta despues del primero, no antes,
*                  incluye por final" o
*                "Inserta antes del ultimo, no despues,
*                  incluye por inicio"
*              SuprimeNodoSgte, SuprimeNodoAnt
*                "Elimina un nodo, en cualquier punto de la Lista,
*                  sino enc. no."
***************/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct sNodo *tLista;
struct sNodo {
  tLista ant;
  tInfo  Info;
  tLista sgte;
};

void InsertaInicioFte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNvo = new sNodo;
```

```
  pNvo->info = valor;
  pNvo->sgte = ListFte;
  pNvo->ant = NULL;
  if (!ListFte)
    ListFin = pNvo;
  else
    ListFte->ant = pNvo;
  ListFte = pNvo;
} //InsertaInicioFte

void InsertaInicioFin(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNvo = new sNodo;

  pNvo->info = valor;
  pNvo->ant = ListFin;
  pNvo->sgte = NULL;
  if (!ListFin)
    ListFte = pNvo;
  else
    ListFin->sgte = pNvo;
  ListFin = pNvo;
} //InsertaInicioFin

void InsertaEnMedioSgte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNvo,
         pAct;

  pNvo = new sNodo;
  pNvo->Info = valor;
  pAct = ListFte;
  while (pAct->sgte && valor > pAct->sgte->info)
          pAct = pAct->sgte;
  pNvo->sgte = pAct->sgte;
  pAct->sgte = pNvo;
  pNvo->ant = pAct;
  if (!pNvo->sgte)
    ListFin = pNvo;
  else
    pNvo->sgte->ant = pNvo;
} //InsertaEnMedioSgte

void InsertaEnMedioAnt(tLista &ListFte, tLista &ListFin, tInfo valor) {
```

```
  tLista pNvo,
          pAct;

  pNvo = new sNodo;
  pNvo->info = valor;
  pAct = ListFin;
  while (pAct->ant && valor < pAct->ant->info)
            pAct = pAct->ant;
  pNvo->ant = pAct->ant;
  pAct->ant = pNvo;
  pNvo->sgte = pAct;
  if (!pNvo->ant)
    ListFte = pNvo;
  else
    pNvo->ant->sgte = pNvo;
} //InsertaEnMedioAnt

void InsertaNodoFte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  if (!ListFte || valor < ListFte->info)
            InsertaInicioFte(ListFte,ListFin, valor);
  else
            InsertaEnMedioSgte(ListFte,ListFin,valor);
} //InsertaNodoFte

void InsertaNodoFin(tLista &ListFte, tLista &ListFin, tInfo valor) {
  if (!ListFin || valor > ListFin->Info)
            InsertaInicioFin(ListFte,ListFin,valor);
  else
            InsertaEnMedioAnt(ListFte,ListFin,valor);
} //InsertaNodoFin

void SuprimeNodoSgte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pAct,
          pAnt;

  pAct = ListFte;
  pAnt = NULL;
  while (pAct && valor > pAct->info) {
    pAnt = pAct;
            pAct = pAct->sgte;
  }
  if (pAct && valor == pAct->info) {
            if (!pAnt)
      ListFte = pAct->sgte;
```

```
    else
      pAnt->sgte = pAct->sgte;
    if (!pAct->sgte)
      ListFin = pAnt;
    else
      pAct->Sgte->Ant = pAnt;
    if (!ListFte)
      ListFin = NULL;
    delete pAct;
  }
} //SuprimeNodoSgte

void SuprimeNodoAnt(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pAct,
         pAnt;

  pAct = ListFin;
  pAnt = NULL;
  while (pAct && valor < pAct->info) {
    pAnt = pAct;
         pAct = pAct->ant;
  }
  if (pAct && valor == pAct->info) {
         if (!pAnt)
      ListFin = pAct->ant;
    else
      pAnt->ant = pAct->ant;
    if (!pAct->ant)
      ListFte = pAnt;
    else
      pAct->ant->sgte = pAnt;
    if (!ListFin)
      ListFte = NULL;
    delete pAct;
  }
} //SuprimeNodoAnt

void EmiteListaSgte(tLista List) {
  tLista pAct = List;

  while (pAct) {
    cout << pAct->info << " ";
    pAct = pAct->sgte;
```

```
  }
  cout << endl;
} // EmiteListaSgte

void EmiteListaAnt(tLista List) {
  tLista pAct = List;

  while (pAct) {
    cout << pAct->info << " ";
    pAct = pAct->ant;
  }
  cout << endl;
} // EmiteListaAnt

main () {
  tInfo    num;
  tLista  ListaFte = NULL,
          ListaFin = NULL;

  clrscr();
  cout << "Ing. valor FIN = 0: ";
  cin >> num;
  while ( num > 0) {
    InsertaNodoFte(ListaFte, ListaFin, num);
    cout << "Ing. valor: ";
    cin >> num;
    InsertaNodoFin(ListaFte, ListaFin, num);
    cout << "* Ing. valor FIN = 0: ";
    cin >> num;
  }
  cout << "-----------x----------" << endl;

  while (ListaFte) {
    EmiteListaSgte(ListaFte);
          cout << "Ing. valor a eliminar: ";
          cin >> num;
          SuprimeNodoSgte(ListaFte, ListaFin, num);
  }
  cout << "***********X***********" << endl;
  cout << "Ing. valor FIN = 0: ";
  cin >> num;
  while ( num > 0) {
    InsertaNodoFte(ListaFte, ListaFin, num);
    cout << "Ing. valor: ";
```

```
    cin >> num;
    InsertaNodoFin(ListaFte, ListaFin, num);
    cout << "+ Ing. valor FIN = 0: ";
    cin >> num;
  }
  cout << "-----------x----------" << endl;

  while (ListaFte) {
    EmiteListaAnt(ListaFin);
            cout << "Ing. valor a eliminar: ";
            cin >> num;
            SuprimeNodoAnt(ListaFte, ListaFin, num);
  }
  return 0;
}
```

Estructuras dinámicas no lineales:

Árbol binario

```
/*
   Id.Programa: ArbolBinario.cpp
   Autor......: Lic. Hugo Cuello
   Fecha......: junio-2015 Fecha original en Pascal junio-2002.
   Comentario.: Genera, Emite (In-Orden, Pre-Orden, Pos-Orden), Busca
                y Elimina, de Estructura Dinamica Arbol Binario.
*/

typedef unsigned short word;
typedef word tInfo;
typedef struct tNodo *tTree;
struct tNodo {
  tTree pIzq;
  tInfo Info;
  tTree pDer;
};

#include <conio.h>
#include <iomanip.h>
#include <fstream>
using namespace std;

void CrearArbol(tTree &Tree) {
  Tree = NULL;
```

```
} //CrearAbrol

bool ArbolVacio(tTree &Tree) {
  return Tree == NULL;
} //ArbolVacio

bool ArbolLleno(word MagnitudNodo) {
  return MagnitudNodo > 10000;
} //ArbolLleno

void InsertarHoja(tTree &Tree, tTree pNvo) {
  if (ArbolVacio(Tree))
    Tree = pNvo;
  else
    if (pNvo->Info < Tree->Info)
      InsertarHoja(Tree->pIzq,pNvo);
    else
      InsertarHoja(Tree->pDer,pNvo);
} //InsertaNodoHoja

void ListarInOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    ListarInOrden(Tree->pIzq);
    cout << setw(3) << Tree->Info;
    ListarInOrden(Tree->pDer);
  }
} //ListarInOrden

void ListarPreOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    cout << setw(3) << Tree->Info;
    ListarPreOrden(Tree->pIzq);
    ListarPreOrden(Tree->pDer);
  }
} //ListarPreOrden

void ListarPosOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    ListarPosOrden(Tree->pIzq);
    ListarPosOrden(Tree->pDer);
    cout << setw(3) << Tree->Info;
  }
} //ListarPosOrden
```

```
tTree BuscarNodo(tTree &Tree, tInfo valor) {
  if (ArbolVacio(Tree))
    return NULL;
  else
    if (valor == Tree->Info)
      return Tree;
    else
      if (valor < Tree->Info)
        return BuscarNodo(Tree->pIzq,valor);
      else
        return BuscarNodo(Tree->pDer,valor);
} //BuscarNodo

void EliminarNodo(tTree &Tree, tInfo valor) {
  tTree pAct,
        pAnt,
        pElim;

  if (!ArbolVacio(Tree))
    if (valor < Tree->Info)
      EliminarNodo(Tree->pIzq,valor);
    else
      if (valor > Tree->Info)
        EliminarNodo(Tree->pDer,valor);
      else {
        pElim = Tree;
        if (!pElim->pDer)
          Tree = pElim->pIzq;
        else
          if (!pElim->pIzq)
            Tree = pElim->pDer;
          else {
            pAct = pElim->pIzq;
            pAnt = pAct;
            while (pAct->pDer) {
              pAnt = pAct;
              pAct = pAct->pDer;
            }
            if (pAct == pAnt)
              pElim->pIzq = pAct->pIzq;
            pElim->Info = pAct->Info;
            pElim = pAct;
            pAnt->pDer = pAct->pIzq;
```

```
        }
      delete pElim;
      gotoxy(1,24); clreol();
      cout << "El elemento " << setw(3) << valor << " fue eliminado... ";
    }
  else {
    gotoxy(1,24); clreol();
    cout << "El elemento " << valor << " no existe... ";
  }
} //EliminarNodo

word IngElem(string mens) {
  word elem;

  gotoxy(20,8);
  cout << mens;
  gotoxy(10,25);
  cout << "Para finalizar ingresar un valor = 0...";
  gotoxy(10,9); clreol();
  cout << "Ing. valor: ";
  cin >> elem;
  return elem;
} //IngElem

main() {
  tTree pRoot,
        pNvo,
        pNodo;
  word  elem;
  char  opc;

  clrscr();
  CrearArbol(pRoot);
  do {
    clrscr();
    cout << "1: Insertar" << endl;
    cout << "2: Quitar" << endl;
    cout << "3: Listar In-Orden" << endl;
    cout << "4: Listar Pre-Orden" << endl;
    cout << "5: Listar Pos-Orden" << endl;
    cout << "6: Buscar elemento" << endl;
    cout << "0: Finalizar" << endl;
    cout << "Su opcion: ";
    opc = getche();
```

```
  switch(opc) {
    case '1':
      elem = IngElem("Insertar");
      while (elem) {
        pNvo = new tNodo;
        pNvo->Info = elem;
        pNvo->pIzq = NULL;
        pNvo->pDer = NULL;
        InsertarHoja(pRoot,pNvo);
        elem = IngElem("Insertar");
      }
      break;
    case '2':
      elem = IngElem("Quitar");
      while (elem) {
        EliminarNodo(pRoot,elem);
        cout << "Oprima una tecla para continuar...";
        getch();
        gotoxy(1,24); clreol();
        elem = IngElem("Quitar");
      }
      break;
    case '3':
      gotoxy(20,8); clreol();
      cout << "Listar In-Orden...";
      gotoxy(20,12);
      ListarInOrden(pRoot);
      gotoxy(1,24); clreol();
      cout << "Oprima una tecla para continuar...";
      getch();
      break;
    case '4':
      gotoxy(20,8); clreol();
      cout << "Listar Pre-Orden...";
      gotoxy(20,12);
      ListarPreOrden(pRoot);
      gotoxy(1,24); clreol();
      cout << "Oprima una tecla para continuar...";
      getch();
      break;
    case '5':
      gotoxy(20,8); clreol();
      cout << "Listar Pos-Orden...";
```

```
      gotoxy(20,12);
      ListarPosOrden(pRoot);
      gotoxy(1,24); clreol();
      cout << "Oprima una tecla para continuar...";
      getch();
      break;
    case '6':
      elem = IngElem("Buscar");
      while (elem) {
        pNodo = BuscarNodo(pRoot,elem);
        gotoxy(1,24); clreol();
        if (pNodo)
          cout << "El elemento " << pNodo->Info << " se encontro...";
        else
          cout << "El elemento " << elem << " no existe... ";
        cout << "Oprima una tecla para continuar...";
        getch();
        gotoxy(1,24); clreol();
        elem = IngElem("Buscar");
      }
    }
  }
  while (opc != '0');
}
```

Unidad V Capítulo III

Unidad V – Capítulo III - Recursividad

La recursividad es una herramienta de programación.

La recursión permite definir un objeto en "términos de sí mismo".

Encontramos ejemplos recursivos de la vida diaria; cuando enfrentamos dos espejos y vemos que las imágenes que muestran se replican indefinidamente.

El Genio y su amo, es otro ejemplo de recursividad; veamos como: Aladino frota la lámpara y aparece el Genio, éste le dice, Amo selecciona tres deseos, Aladino le contesta cuáles son sus dos primeros deseos y el Genio se lo concede, y cuando debe decidir por el tercer deseo notando que es el último, piensa un poco y luego le dice al Genio, concédeme tres deseos más, esto se repite hasta el infinito cada vez que debe decidir por el tercer deseo, cual pide tres deseos más.

En la programación de computadoras un módulo o sub-programa que se invoca a sí mismo se dice que es recursivo.

La recursión puede establecerse de dos formas a saber:

Directa o simple

Cuando un sub-programa se invoca a sí mismo este tipo de recursividad se dice que es directa.

El sub-programa M1 se invoca a sí mismo.

La observación que podemos indicar es ¿cómo salimos de esta trampa?, ya que una de las premisas para considerar un algoritmo es que debe ser finito, y si quedamos atrapados en la recursividad, jamás finalizará este proceso, razón por la cuál debemos poder evadir la recursividad en algún instante del proceso. Generalmente en estos casos se establece al menos una pregunta, por lo que en una de las ramas debemos establecer la situación recursiva y, por la otra rama de la pregunta debemos establecer la situación no recursiva, es decir una sentencia que no vuelva a invocar al sub-programa.

Indirecta

Cuando un sub-programa digamos M1 invoca a otro sub-programa M2, el cuál a su vez invoca al sub-programa M1, en estos casos estamos en la recursividad indirecta, como lo muestra el siguiente gráfico:

De la misma manera que en la recursividad directa para poder salir de la recursión debemos establecer al menos una pregunta que nos permita realizar una sentencia no recursiva.

En ambos casos, para abandonar la recursividad debemos encontrar algún caso básico o **base**, para que de esta manera podamos encontrar una salida directa a la recursividad. Por otro lado, la entrada de datos debe alcanzar el o los casos básicos o ir acercándonos a estos casos básicos.

Existen casos específicos de recursividad, casos que son naturalmente recursivos, en estos casos, es ventajoso dar una solución recursiva al problema a tratar. Las listas dinámicas y los árboles son procesos naturalmente recursivos y aprovechando esta herramienta de programación queda justificada su utilización para la solución de problemas recursivos.

No todos los lenguajes permiten emplear técnicas recursivas, solo los lenguajes que permitan **asignación dinámica de memoria** podrán contar con esta herramienta informática. Los lenguajes Pascal, C / C++ entre otros ofrecen esta posibilidad. Lenguajes como Cobol, Basic no permiten esta posibilidad.

Todo proceso recursivo puede realizarse como proceso iterativo.

Ventajas y desventajas en la recursividad

Los procesos recursivos otorgan la **ventaja** cuando estos son naturalmente recursivos de ser **más concisos** en la codificación y por ende más claros en su comprensión.

Por otro lado, los procesos recursivos tienen la **desventaja** de ser **más lentos** en resolver un problema y además **requieren mayor cantidad de memoria** para llevar a cabo el proceso.

El funcionamiento interno es manejado por una **Pila** o **Stack**, el que almacenará los valores tanto de los parámetros como de las variables locales que utilice el módulo o sub-programa.

El proceso presenta dos recorridos en la Pila, uno para meter elementos para recordar a futuro o tomarlos más adelante y en segundo lugar extraer los elementos guardados en la Pila para completar los cálculos pendientes que no pudieron ser realizados en su momento por las llamadas recursivas. Este proceso finalizará cuando la Pila haya quedado vacía.

La recursividad es una herramienta o técnica matemática que parte de una instancia compleja y las define en función de instancias más simples del mismo problema, hasta el punto en donde esas instancias se definan en forma explícita.

Llamadas recursivas Tn = f(Tn-1, Tn-2,…, To)

Términos particulares o base, que permita finalizar la recurrencia, Ti = g(i)

Condiciones particulares, situaciones no recursivas, caso base.

Condiciones generales, situaciones recursivas.

La recurrencia se puede comparar con el método inductivo, en la cual para probar la veracidad de un hecho se basa en instancias anteriores al hecho, por la cual su veracidad fue previamente demostrado.

La recursividad utiliza Estructuras de Selección simple o múltiple, a fin de separar los casos base de los casos recurrentes.

Recursión lineal o recursión a la cola o recursivo simple es, cuando cada llamada recursiva genera a lo sumo otra mueva llamada recurrente. Cuando existe solo una llamada de invocación al módulo. Pueden presentarse **dos casos** 1) la llamada posee dos componentes para efectuar un cálculo pero debe postergarse por la llamada recursiva o, 2) simplemente debe invocarse al módulo (recursividad a la cola).

Recursión no lineal o recursivo múltiple es, cuando cada llamada genera dos o más llamadas recurrentes. Recursión binaria es, cuando utiliza por cada llamada recurrente solo dos llamadas recurrentes. (por ej. Sucesión de Fibonacci).

Recurrente: proceso que se repite.

Recurrencia: propiedad de secuencias en la que cualquier término puede calcularse conociendo los precedentes. En aquellos casos en que el objeto en estudio sea naturalmente recursivo, se lograra el beneficio indicado anteriormente.

En otros casos en que no se note diferencia entre un proceso recursivo de uno iterativo, la elección sin lugar a dudas será el caso iterativo, ya que logra mayor rapidez y menor consumo de memoria.

La razón por la cual se requiere mayor tiempo y memoria, está dado por el hecho en que la recursividad utiliza una estructura de tipo Pila.

La Pila o Stack es una zona de la memoria interna para la asignación dinámica de memoria conocida como Heap o montículo.

El manejo de la Pila requiere dos momentos, uno para ir metiendo valores y de otro momento para ir sacando esos valores, siempre que se incorporen o se quiten elementos se hace por el tope de la Pila (L.I.F.O.), vale decir, Último en Entrar es el Primero en Salir.

El uso de la **Pila** es el verdadero motivo por el cual los métodos recurrentes requieren más memoria y tiempo que los métodos iterativos.

Problemas recursivos o iterativos

- Factorial
- Cadena para invertirla
- Búsqueda Binaria
- Fibonacci
- Listas ligadas
- Árboles
- Método de Ordenamiento por Quick Sort
- Ordenamiento Externo por Mezcla.
- Torres de Hanoi

Mostrar diferentes ejemplos recursivos para

- Recursión directa o simple
- Recursión indirecta
- Recursión a la cola
- Recursión múltiple
- Recursión binaria
- Recursión indirecta (Casos específicos del compilador parse)

Ejemplos

Ejemplo axb

```
/*
  Id.Programa: axb.cpp
  Fecha......: junio-2015
  Autor......: Lic. Hugo Cuello
```

```
  Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

long axb(int x, int y) {
 if (y == 1)
   return x;
 else
   return axb(x,y - 1) + x;
} //axb


main() {
 int a,
     b;

 cout << "Ing. valor a: ";
 cin >> a;
 cout << "Ing. valor b: ";
 cin >> b;
 cout << "a * b= " << axb(a,b);
 return 0;
}
```

Ejemplo BusBin

```
/*
  Id.Programa: BusBin.cpp
  Fecha......: junio-2015
  Autor......: Lic. Hugo Cuello
  Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef int tv[10001];

long BusBin(tv v, int clv, int pri, int ult) {
 int med;
```

```
  if (pri > ult)
    return 0;
  else {
    med = (pri + ult) / 2;
    if (v[med] == clv)
      return med;
    else
      if (v[med] < clv)
        return BusBin(v,clv,med + 1,ult);
      else
        return BusBin(v,clv,pri,med - 1);
  }
} //BusBin

main() {
  tv  vec;
  int elem;
 for (register i = 1; i <= 10000; i++)
    vec[i] = i * 2;
  cout << "Ing. elemento: ";
  cin >> elem;
  cout << "El elemento " << elem;
  if (BusBin(vec,elem,1,10000))
    cout << " se encontro en el vector...";
  else
    cout << " no se encontro en el vector...";
  return 0;
}
```

Ejemplo Crazy

```
/*
   Id.Programa: Crazy.cpp
   Fecha...........: junio-2015
   Autor...........: Lic. Hugo Cuello
   Comentar....: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

long Crazy(int x, int y) {
  if (x == y)
    return 1;
```

```
  else
    if (x > y)
      return 0;
    else
      return x + Crazy(x + 1, y);
} //Crazy

main() {
  int a,
      b;

  cout << "Ing. entero: ";
  cin >> a;
  cout << "Ing. entero: ";
  cin >> b;
  cout << "Crazy(" << a << ", " << b << ")= " << Crazy(a,b);
  return 0;
}
```

Ejemplo DolorCabeza

```
/*
   Id.Programa: DolorCabeza.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tLista;
struct tNodo {
  tInfo  Info;
  tLista Sgte;
};

void InsertaInicio(tLista &Lista, tInfo valor) {
  tLista pNvo;

  pNvo = new tNodo;
  pNvo->Info = valor;
```

```
  pNvo->Sgte = Lista;
  Lista = pNvo;
} //InsertaInicio

long DolorCabeza(tLista &Lista) {
  if (Lista)
    return 1 + DolorCabeza(Lista->Sgte);
  else
    return DolorCabeza(Lista);
} //DolorCabeza

main() {
  tLista Lista;

  for (register i = 1; i <=30; i++)
    InsertaInicio(Lista,random(100));
  cout << DolorCabeza(Lista); // jamás regresa de la recursividad, no hay casos base
  return 0;
}
```

Ejemplo DolorCabeza2

```
/*
   Id.Programa: DolorCabeza2.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tLista;
struct tNodo {
  tInfo  Info;
  tLista Sgte;
};

void InsertaInicio(tLista &Lista, tInfo valor) {
  tLista pNvo;

  pNvo = new tNodo;
  pNvo->Info = valor;
```

```
  pNvo->Sgte = Lista;
  Lista = pNvo;
} //InsertaInicio

long DolorCabeza(tLista &Lista) {
  if (Lista)
    return 1 + DolorCabeza(Lista->Sgte);
  else
    return DolorCabeza(Lista);
} //DolorCabeza

void DolorCabeza2(tLista &Lista) {
  tLista pElim;

  if (Lista) {
    pElim = Lista;
    DolorCabeza2(Lista->Sgte);
    delete pElim;
  }
} //DolorCabeza2

main() {
  tLista Lista;

  for (register i = 1; i <=30; i++)
    InsertaInicio(Lista,random(100));
  cout << DolorCabeza(Lista); // jamás regresa de la recursividad, no hay casos base
  DolorCabeza2(Lista);
  return 0;
}
```

Ejemplo Factorial

```
/*
   Id.Programa: factorial.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;
```

```
long factorial(int x) {
  if (x == 0 || x == 1)
    return 1;
  else
    return x * factorial(x - 1);
} //factorial

main() {
  int n;

  cout << "Ing. valor n para calcular su factorial: ";
  cin >> n;
  cout << n << "!= " << factorial(n);
  return 0;
}
```

Ejemplo Fibonacci

```
/*
   Id.Programa: fibonacci.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

long fibonacci(int x) {
  if (x == 0)
    return 0;
  else
    if (x == 1)
      return 1;
    else
      return fibonacci(x - 1) + fibonacci(x - 2);
} //factorial

main() {
  int n;

  cout << "Ing. valor n para calcular los n + 1 terminos de la suc. de Fibonacci: ";
  cin >> n;
  cout << "Sucesion de Fibonacci hasta " << n << "= " << fibonacci(n);
```

```
  return 0;
}
```

Ejemplo Indirecta

```
/*
   Id.Programa: Indirecta.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef unsigned char byte;
typedef char str20[21];

void Basc(char car);

void Aasc(char car) {
  if (car > 'A')
    Basc(car);
  cout << car << ' ';
} //Aasc

void Basc(char car) {
  Aasc(--car);
} //Basc

main() {
  char car;

  cout << "Ing. caracter: ";
  car = getche();
  Aasc(car);
  return 0;
}
```

Ejemplo Insensata

```
/*
   Id.Programa: Insensata.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tLista;
struct tNodo {
  tInfo  Info;
  tLista Sgte;
};

void InsertaInicio(tLista &Lista, tInfo valor) {
  tLista pNvo;

  pNvo = new tNodo;
  pNvo->Info = valor;
  pNvo->Sgte = Lista;
  Lista = pNvo;
} //InsertaInicio

bool Insensata(tLista &Lista) {
  if (!Lista)
    return false;
  else
    return (1 + 1 == 2) && Insensata(Lista->Sgte);
} //Insensata

void Insensata2(tLista &Lista) {
  tLista pElim;

  if (Lista) {
    pElim = Lista;
    Insensata2(Lista->Sgte);
    delete(pElim);
  }
} //Insensata2

main() {
```

```
  tLista Lista;

  for (register i = 1; i <=30; i++)
    InsertaInicio(Lista,random(100));
  cout << boolalpha << Insensata(Lista);
  Insensata2(Lista);
  return 0;
}
```

Ejemplo maxComDiv

```
/*
   Id.Programa: maxComDiv.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

long mcd(int x, int y) {
  if (y == 0)
    return x;
  else
    return mcd(y, x % y);
} //mcd

main() {
  int a,
      b;

  cout << "Ing. dividendo: ";
  cin >> a;
  cout << "Ing. divisor: ";
  cin >> b;
  cout << "mcd(" << a << ", " << b << ")= " << mcd(a,b);
  return 0;
}
```

Ejemplo Mistery

```
/*
   Id.Programa: Mistery.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef unsigned short word;
typedef int tInfo;
typedef struct tNodo *tLista;
struct tNodo {
  tInfo  Info;
  tLista Sgte;
};

void InsertaInicio(tLista &Lista, tInfo valor) {
  tLista pNvo;

  pNvo = new tNodo;
  pNvo->Info = valor;
  pNvo->Sgte = Lista;
  Lista = pNvo;
} //InsertaInicio

word Mistery(tLista &Lista) {
  if (!Lista)
    return 0;
  else
    return Lista->Info + Mistery(Lista->Sgte);
} //Mistery

word Mistery2(tLista &Lista, tLista &pTemp) {
  tLista pElim;

  pTemp = Lista;
  if (!pTemp)
    return 0;
  else
    return pTemp->Info + Mistery2(Lista,pTemp->Sgte);
} //Mistery2
```

```
void Mistery3(tLista &Lista) {
  tLista pElim;

  if (Lista) {
    pElim = Lista;
    Mistery3(Lista->Sgte);
    delete(pElim);
  }
} //Mistery3

main() {
  tLista Lista,
         pCopia;

  for (register i = 1; i <=30; i++)
    InsertaInicio(Lista,random(100));
  cout << Mistery(Lista);
  cout << Mistery2(Lista,pCopia);
  Mistery3(Lista);
  return 0;
}
```

Ejemplo Palíndromo

```
/*
   Id.Programa: Palindromo.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef unsigned char byte;
typedef char str20[21];

void Palindromo(str20 sarta, byte n) {
  if (n == 0)
    cout << sarta[0];
  else {
    cout << sarta[n];
    Palindromo(sarta, n - 1);
```

```
  }
} //Palindromo

main() {
  str20 cad;

  cout << "Ing. cadena: ";
  gets(cad);
  Palindromo(cad,strlen(cad) - 1);
  return 0;
}
```

Ejemplo Pot2

```
/*
   Id.Programa: Pot2.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

long Pot(int x) {
  if (x == 0)
    return 1;
  else
    return Pot(x - 1) * 2;
} //Pot

bool EsPot(int x) {
  int y = 0;
  while (Pot(y) < x)
    y++;
  if (Pot(y) <= x)
    return true;
  else
    return false;
} //EsPot

main() {
  int a,
      b;
```

```
  cout << "Ing. valor entero: ";
  cin >> a;
  cout << "Ing. valor entero: ";
  cin >> b;
  cout << boolalpha;
  cout << "EsPot(" << a << "): " << EsPot(a) << endl;
  cout << "EsPot(" << b << "): " << EsPot(b) << endl;
  return 0;
}
```

Ejemplo Puzzle

```
/*
   Id.Programa: Puzzle.cpp
   Fecha......: junio-2015
   Autor......: Lic. Hugo Cuello
   Comentar...: Calculo factorial recursivo
*/

#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tLista;
struct tNodo {
  tInfo  Info;
  tLista Sgte;
};

void InsertaInicio(tLista &Lista, tInfo valor) {
  tLista pNvo;

  pNvo = new tNodo;
  pNvo->Info = valor;
  pNvo->Sgte = Lista;
  Lista = pNvo;
} //InsertaInicio

long Puzzle(tLista &Lista) {
  if (!Lista)
    return 1;
  else
```

```
    if (Lista->Info > 0)
      return Lista->Info * Puzzle(Lista->Sgte);
    else
      return Puzzle(Lista->Sgte);
} //Puzzle

void Puzzle2(tLista &Lista) {
  tLista pElim;

  if (Lista) {
    pElim = Lista;
    Puzzle2(Lista->Sgte);
    delete(pElim);
  }
} //Puzzle2

main() {
  tLista Lista;

  for (register i = 1; i <=30; i++)
    InsertaInicio(Lista,random(100));
  cout << Puzzle(Lista);
  Puzzle2(Lista);
  return 0;
}
```

U.T.N. F.R.B.A. Algoritmos y Estructuras de Datos Prof.: Hugo A. Cuello
Guía de ejercicios Nro. 5
Estructuras dinámicas y Recursividad.

Guía de ejercicios #5

1. Se tiene un archivo de Gastos producidos en un año calendario, conteniendo: Importe (real) y Mes (1..12), sin orden. Se requiere de un proceso que : Emita un listado de los gastos producidos en c/u. de los meses del año en orden inverso con respecto a como se conocieron c/u. de los datos. Además informar total por c/u. de los meses y el total anual.
2. Idem anterior, pero el listado deberá emitirse en el mismo orden con respecto a como se conocieron c/u. de los datos.
3. Idem al punto 1, pero el listado deberá emitirse ordenado por Importe en forma creciente.
4. Se tiene un archivo de Gastos producidos en un año calendario, conteniendo: Importe (real), Mes (1..12) y Día (1..31), sin orden, el cual podrá repetirse un gasto para un mismo mes de un mismo día. Se requiere de un proceso que : Emita un listado de los gastos producidos en c/u. de los meses del año y ordenado por Día (acumulado). Además informar total por c/u. de los meses y el total anual.
5. Crear un programa el cual solicite por medio de un menú apropiado, agregar, suprimir o listar los elementos de una lista.
6. Idem anterior, pero elementos de una pila.
7. Idem al punto 5., pero elementos de una cola.
8. Por medio de un menú apropiado que solicite Término del Polinomio 1, Término del Polinomio 2, Sumar Polinomios, Listar Polinomios; realizar las acciones correspondientes en un programa.
9. Dada una pila con elementos enteros generada en forma al azar, ordenarla en forma creciente y luego emitir los elementos. Usando otra pila
10. Dada una cola con elementos enteros generada en forma al azar, ordenarla en forma creciente y luego emitir los elementos. Usando otra cola.
11. Dada una expresión en notación infija, convertirla en notación prefija y luego emitirla.
12. Dada una expresión en notación infija, convertirla en notación postfija y luego emitirla.
13. Dado un archivo de datos desordenado, se solicita ordenarlo por el valor de un campo clave.
14. La D.G.I. requiere poder realizar el cobro sobre los Bienes Personales, para ello se cuenta con :
    a) Un archivo de **Declaración Jurada** de las personas ordenado creciente por Apellido-Nombre, con el siguiente formato de registro: Apellido y Nombre (car.30), Dirección (car.30), Localidad (car.20), Código Postal (4 díg.), Teléfono (car.15), Tipo y Nro. de documento (car.12), Importe Total de Bienes (real), originalmente en cero.
    b) Un archivo de **Bienes Personales**, donde por cada persona habrá un registro de cabecera, seguido de uno o varios o ningún registro de bienes, cuyo formato de registro es : Tipo y Nro. de documento, Tipo de Bien (car.20), Importe (real),

Descripción (car.50). El archivo se encuentra ordenado creciente por Tipo y Nro. de documento y Tipo de Bien. En el registro cabecera el Tipo de Bien contiene un asterisco '*'. Cada registro del punto a) se corresponde con cada registro cabecera del punto b).

c) Un archivo de Impuestos, no más de 10 registros, cuyo formato de registro contiene : Monto (real), Porcentaje (2 díg.), ordenado por Monto en forma creciente.

Se pide :

1. Un listado de c/u. de los bienes por cada persona, si el total de los valores es mayor o igual al monto mínimo, de acuerdo al siguiente diseño :

Listado de Bienes Personales que pagan Impuesto

Apellido, Nombres : x----------x
T-Nro.Documento : x----------x

T.Bien	**Descripción**	**Valor**	**%**
Impuesto			
x----------x	x----------x	99999,99	
.	.	.	
	**** Total** :	99999,99	99
99999,99			

2. Generar un archivo de Grandes Bienes de aquellas personas cuyo total de valores sea mayor al mayor monto establecido en Impuestos, con el siguiente diseño de registro : Apellido-Nombre, T-Nro.Doc., Importe Total.
3. Actualizar el campo Importe Total de Bienes en el archivo del punto a).

15. Se requiere de un proceso que simule las siguientes acciones : Un ama de casa (doña Rosa), va al supermercado a realizar las distintas compras. Como suele olvidarse de las cosas que necesita, decide anotar cada requerimiento de productos en el orden en que surgen las necesidades. Al llegar al supermercado, comienza a realizar las compras, pasando por las góndolas, momento en que se detiene y observa si necesita algo de lo anotado; si es así, va tachando de su lista cuando se carga en el chango el producto encontrado. Esto continúa por cada góndola visitada. Para llevar a cabo estas acciones se cuenta con :

a) un archivo de **Artículos** sin orden alguno cuyo formato de registro es : Código de Artículo (<=60000), Descripción (30 car.), Stock Actual (entero), Stock Mínimo (entero), Precio (real), Unidad de Medida (10 car.).

b) Una **lista de requerimientos de Compra** que deberá ser ingresada por el comprador antes de comenzar las compras, conteniendo como información : Descripción del Artículo (30 car), cantidad (entero).

Se pide :

1. Generar una lista de productos comprados (puede diferir de la lista de requerimientos), conteniendo como información lo siguiente : Cantidad (entero), dirección del Artículo en el archivo Artículos (entero largo).
2. Emitir el Ticket de la compra realizada, según el siguiente diseño :

Código de artículo,		
Descripción,	.	
Cantidad,	.	Se repite por cada compra
Precio,	.	
Total por artículo		

Cantidad de artículos : 999
Total a Pagar : $ 9.999,99

3. Emitir el Ranking de las 10 compras de artículos, de mayor importe total, ordenado decreciente por Importe, con el siguiente formato :

Ranking de los x artículos de mayor importe

Código de Artículo	**Importe**
99999	99.999,99

4. Actualizar el stock del archivo artículos.
5. Generar un archivo de Reposiciones, para aquellos artículos cuyo stock actual esté por debajo del stock mínimo, ordenado decreciente por la diferencia entre el stock mínimo y el stock actual, de acuerdo al siguiente formato de registro : Código de artículo, Stock Actual, Stock Mínimo.

16. Un Aeropuerto Internacional requiere las actualizaciones en el Tablero de Vuelos, para ello se cuenta con los siguientes archivos de datos :

a) **Empresas Aeronáuticas** : ordenado por Código de Empresa, máximo 20, conteniendo : Código de Empresa (5 car.), Nombre (30 car.), Domicilio (20 car.), Teléfono (15 car.), Otros datos (70 car.).

b) **Vuelos** : (de un día, de 6:00 a 24:00 hs.), en donde el Nro. de vuelo se corresponde unívocamente con la dirección del registro en el archivo, no más de 800 vuelos, conteniendo : Código de la Empresa, Código de Movimiento (´L' = Llega, 'S'= Sale), Lugar de Origen / Destino (30 car.), Hora (hhmm, de Llegada / Salida), Cantidad de Pasajeros (3 díg.), Nro. de Plazas (3 díg.), Escalas (lógico). El campo Cantidad de Pasajeros viene originalmente con valor cero, el proceso deberá actualizar la Cantidad de Pasajeros.

c) **Pasajeros** : (Ordenado por Nro. de vuelo con repetición), conteniendo : Nro. de vuelo, Fecha (ddmmaa), Apellido y Nombre (20 car.), Edad (2 díg.), Nacionalidad (20 car.), Sexo ('F', 'M'), Tipo y Nro. de Documento (15 car.).

Se pide :

1. Visualizar el Tablero de Vuelos Programados para el día, de la siguiente forma:

Tablero de Vuelos del día : 99-99-99

Empresa **Empresa**: x-----x
Llegada Salida **Llegada Salida**
- - - - - - - - - - -
 # Vuelo # Vuelo
HH:MM HH:MM
Origen Destino

Cada vez que un vuelo llega o sale se deberá actualizar el tablero, el cuál puede contener un máximo de 10 vuelos a la vez.

2. Mantener los vuelos (Llegadas, Salidas) en una estructura dinámica. (¿cuál será la más apropiada?).
3. Emitir la Lista de Pasajeros del día de la fecha para todos los vuelos programados, según el siguiente diseño :

Lista de Pasajeros

 # Vuelo : 999
Empresa : X(5) **¿Con escalas?** : XX
Llegada HH : MM **Salida** : HH : MM

Apellido, Nombre	**Nacionalidad**	**T.Nro.Doc.**	
X(20)	X(20)	X(15)	Idem

Cantidad de Menores : 999
Cantidad de Mujeres : 999
Cantidad Total : 999

Nota : HH:MM del nro.vuelo(1) < HH:MM del nro.vuelo(2) < < HH:MM nro.vuelo(n).
El proceso controlará la hora del Sistema.
Si la edad es < 21 se considera menor de edad.

17. Un supermercado requiere un proceso que informe a diferentes consultas, contando para ello con :

    1. Un archivo de **Artículos** desordenado, conteniendo: Código de Artículo (7 car.), Descripción del Artículo (30 car.), Stock Actual (entero), Stock Mínimo (entero), Código de Rubro (5 car.), Unidad de Medida (10 car.), Precio Unitario (real).
    2. Un archivo de **Rubros**, ordenado por Código de Rubro, conteniendo : Código de Rubro, Descripción (30 car.).

    Se pide :

    1. Generar una "Lisa de Rubros" ordenada, y por cada rubro generar "Listas de Artículos", ordenada, que conforman dicho rubro, de acuerdo al siguiente diseño :
       Por cada rubro : El código de rubro y dos punteros, uno al inicio a la lista de artículos y el otro al próximo rubro.
       Por cada artículo : El código de artículo, la dirección del artículo y un puntero al próximo artículo.
    2. Una vez hecho el punto 1. Informe :
       2.1. ¿Qué artículos pertenecen a cada rubro?, según el siguiente diseño:

       Fecha : 99-99-99 **Hoja** : 999

       Listado por Rubros y Artículos

       Cód.Rubro : X(5) **Nombre** : X(30)

Cód.Art.	**Nombre**	**Precio**
X(7)	X(30)	$ 9.999,99

       Cantidad de ítem : 9999

       2.2. ¿Cuál es el Rubro que contiene la mayor cantidad de items?
       2.3. Los valores porcentuales por Rubro :

       Listado de valores porcentuales por Rubro

Cód.Rubro	**Valor %**
X(5)	999,99

    3. Genere un archivo de índice Articulo.Idx, en el cual ordene lógicamente al archivo artículo, por Código de Rubro y dentro de éste por Código de Artículo, conteniendo : Clave y referencia en artículo.

18. Se requiere de un proceso que simule las siguientes acciones :

    Se realiza una competencia automovilística por clasificación de distintas series. Por cada vuelta establecida deberá actualizarse las posiciones. Al término de la carrera, el ganador pasa a una "Lista de Clasificados". Esto se repite por cada una de las series.

Al finalizar las series, se disputa la "Gran Final". También en este caso por cada vuelta establecida debe actualizarse las posiciones. Al finalizar la competencia los tres primeros puestos suben al podio.

Para llevar a cabo estas acciones se cuenta con los siguientes datos :

a) Un archivo de **Series**, con repetición ordenado creciente por Nro. de Serie, conteniendo : Nro. de Serie (entero), Nombre del Piloto (30 car.), Marca del auto (20 car.), Nro. del auto (byte).

b) Un archivo de **Pilotos**, ordenado por Nombre del Piloto, conteniendo : Nombre del Piloto (30 car.), Lugar de Nacimiento (30 car.), Provincia (20 car.), Fecha de Nacimiento (ddmmaa), Puntaje en el Campeonato (entero), Cantidad de Carreras Ganadas en el Campeonato (entero).

c) Se ingresa por teclado : Cantidad de vueltas en las "Series", en la "Gran Final" y por cada vuelta y Piloto : Nombre del Piloto, Posición, Tiempo empleado, En el caso de producierse abandono, Nro. del auto.

Se pide :

1. Generar una "Lista de Carreras" –Series-, actualizando por cada vuelta las posiciones, en donde cada nodo debe contener como información : Nombre del Piloto, Nro. del auto, Tiempo de la vuelta más rápida. Indica la posición de largada el orden de aparición de los registros.
2. Generar una "Lista de Clasificados" -Gran Final-, de los ganadores de cada una de las series por orden de Tiempo de la vuelta más rápida.
3. Una vez finalizadas las series se disputa la Gran Final. Procesando la "Lista de Clasificados", se debe actualizar por cada vuelta las posiciones, en donde cada nodo contiene la misma información que la "Lista de Carreras" del punto anterior.
4. Informar :

   4.1 Por cada serie : Nro. de Serie, Nombre del ganador, Nro. del auto, Mejor tiempo, la cantidad de participantes antes y después de la carrera.

   4.2 Los ganadores del punto 2. Para ocupar el podio (1er, 2do y 3er. puesto), con mensajes alcaratoiros y cantidad de abandonos.

   4.3 La lista de llegada del punto 2. :

   Posición Nombre del Piloto Nro.auto Mejor tiempo

5. Actualizar el archivo Pilotos, para aquellos corredores que obtuvieron puntaje. El primer puesto también actualiza carreras ganadas.

Nota : Debe tomarse en cuenta los abandonos que se produzcan en cada competencia, eliminándolos de la lista.

El puntaje se establece de la siguiente manera : 1er. puesto: 40 puntos, 2do.puesto: 30 puntos, 3er.puesto: 15 puntos, 4to. puesto: 10 puntos y 5to.puesto: 5 puntos.

19. Dado un archivo de **Vacantes**, con la cantidad de vacantes de c/aula de la facultad, ordenado por Especialidad y División, donde c/registro contiene :

    Nro. de Especialidad (1 a 9), Nro. de División (1 <= 40, 41 <= 60, 61 <= 99), Cantidad de Vacantes (<=99), Cantidad de Inscriptos (<= 99 originalmente en cero).
    Existe al menos una división por cada turno, en todas las especialidades.

    Se dispone además de un archivo de **Solicitudes**, en orden cronológico de los alumnos de primer año que solicitan inscribirse, donde c/registro contiene : Nro. de Legajo (6

díg.), Apellido y Nombres (30 car.), Nro. de Especialidad, Código de Turno ('M', 'T', 'N').

Se pide :

1. Asigne división a cada solicitante.
2. Grabe un registro, con el formato del 2do. Archivo más un campo de Nro. de División, por c/u. de los registros leídos del mismo archivo, para aquellos solicitantes que no encontraron vacante, este campo debe contener cero.
3. Haga un listado de los alumnos solicitantes en c/división de c/especialidad, ordenado por Especialidad, Turno y Cronológico, según se indica :

Especialidad : 9 **Turno** : x----x

Orden Original	**Legajo**	**Apellido, Nombre**
99	999999	x----------x

Nota : Se dispone de 216 bytes en memoria estática. Tamaño de nodos 8 bytes, con un nodo por cada solicitante.

20. Una municipalidad requiere un programa que permita realizar la concelación de deuda de sus contribuyentes. Para ello se dispone de los siguientes archivos :

a) **Deudores** sin ningún orden, donde c/registro contiene : Código de Contribuyente (5 car.), Año de la cuota (88 a 97), Mes de la Cuota (1 a 12), Importe de la Cuota, Estado ('A' = Adeuda, 'C' = Cancelada), Fecha de Cobro (aammdd).

b) **Datos Personales** (<= 8000 registros ordenados por Contribuyente), c/registro con : Código de Contribuyente, Apellido y Nombre (45 car.), Domicilio (45 car.), Código de Zona Municipal (4 car.).

c) **Cobro del Mes** ordenado por Fecha de Cobro y Código de Contribuyente, donde c/registro contiene : Código de Contribuyente, Año de la Cuota, Mes de la Cuota, Fecha de Cobro (aammdd).

Se pide :

1. Actualice en el archivo Deudores las cuotas pagadas por cada contribuyente.
2. Imprima un Listado para control de cobranza como se indica :

Listado de Control de Cobranza

Contrib.	**Apellido, Nombre**	**Año**	**Mes**	**Importe**	**Situac.**
X(5)	X(45)	99	99	99999,99	P
		99	99	99999,99	E

En Situación se lo indica con **P = Pago** aceptado o **E = error** por cuota pagada anteriormente.

3. Imprima el total de lo recaudado por c/zona municipal ordenado por código de zona:

Código de Zona	**Importe Recaudado**
xxxx	999999,99

Nota : Se dispone de 4 Mbytes de memoria.

21. La Capital Federal requiere un estudio estadístico de infracciones realizadas por todo tipo de vehículo en sus calles. Para ello cuenta con 2 archivos c/u de ellos con infracciones de los 12 meses del año 1997, correspondientes a las 2 zonas en que se encuentra dividida la capital, para los que c/registro contiene: Cód. de patente (6 car), Nro. registro Conductor (8 díg.), Fecha de la infracción (aammdd), Zona (6 car.), Cód. de la infracción (5 dig), Nro. placa policial (6 car.). Ambos ordenados por fecha creciente.

Se pide :

1. Genere un único archivo: **CAPITOT.DAT** a partir de los 2 mencionados, ordenado por código de patente (creciente) y fecha (decreciente).

2. Para los vehículos con infracciones en todos los meses del año, imprima el siguiente listado :

Total de Infracciones por mes

COD/PATENTE	Mes 1	Mes 2	Mes 3	Mes 11	Mes 12	Total
XXXXXX	9999	9999	9999	9999	9999	99999

NOTA: Tamaño máximo de nodo 15 bytes. Memoria estática 0,1 Kb.

22. En una Universidad se requiere consolidar la información de aprobación de trabajos prácticos de una de sus especialidades. Para ello se cuenta con los siguientes archivos :

a) Un archivo de **Asistencias regulares** generado por bedelía, ordenado por Materia, donde cada registro contiene : Nro. de Legajo (6 díg.), Código de Materia (3 díg.). Solo están los alumnos que terminaron de cursar c/regulares.

b) Un archivo de **Notas de parciales** generado por los docentes, ordenado por número de curso, donde c/registro contiene : Nro. de Curso (2 díg.), Código de Materia, Nro. de Legajo, Nota primer parcial (0 a 10), nota segundo parcial (0 a 10), Fecha de firma T.P. (ddmmaa). Nota : Están todos los alumnos inscriptos originalmente, una fecha = 0 indica que no firmó.

c) Un archivo de **Trabajos Prácticos**, previamente aprobados, ordenado por fecha de firma, donde cada registro contiene : Nro. de Legajo, Código de Materia, Fecha de firma, Nro. de Curso.

Se pide :

1. Detecte e imprima los 3 posibles errores del archivo de Notas (algún parcial desaprobado, o sin fecha de firma o sin regularidad).

2. Genere un archivo de Trabajos Prácticos aprobados, ordenado por legajo y código de materia, a partir del archivo del punto c) y de los registros correctos del archivo del punto b).

Nota : Tamaño máximo del nodo 13 bytes; máximo espacio utilizable en disco = filesize(Notas) + filesize (TrabPrac).

23. La empresa MoviTel necesita de un proceso para facturar a sus clientes las comunicaciones realizadas en un mes. Para ello cuenta con los siguientes archivos de datos:

a) **COMUNICACIONES** (ordenado por Cliente), conteniendo el registro: Número de Cliente (9 díg.); Cantidad de pulsos (3 díg.); Día del mes (1 a 31).

b) **CLIENTES** (ordenado por Cliente), conteniendo: Número de Cliente; Razón Social (35 car.); Domicilio-Localidad (50 car.); Categoría (2 díg.).

c) **CATEGORIAS** (sin orden), conteniendo: Código de Categoría; Importe de abono; Importe por pulso adicional; Pulsos sin cargo (3 díg.).

Se pide:

1. Imprima la factura de cada cliente (ordenado por Cliente), según el siguiente formato:

Nro.Cliente: 9(9) **Razón Social**: X(35)
Domicilio: X(50)

ABONO	(9999 pulsos sin cargo)		999.99
ADICIONAL	(9999 x imp.pulso-adic.)	+	999.99
	T O T A L	9999.99	

2. Imprima al final una lista (ordenada por **categoría y cliente**) de los clientes que tuvieron menos de 50 pulsos en alguno de los días, con el siguiente formato:

LISTADO DE COMUNICACIONES DIARIAS

Categoría: 99

Nro.Cliente	**1**	**2**	**3**	**< < > >**	**29**	**30**	**31**	**Cant.Dias Pul.< 50**
999999999	999	999	999		999	999	999	99

NOTA: La memoria disponible es suficiente para estructuras dinámicas si se utiliza un nodo por cada usuario que cumpla la condición del punto 2. No olvidar de liberar la memoria cuando ya no se necesite un nodo.
Realizar gráfico de estructuras aclarar que tipo de estructura lineal se emplea, estructura del nodo, cantidad de memoria a emplear, definir y declarar la sección type y var, desarrollar estrategia, algoritmo y codificación en Pascal. Minimizar el tamaño de los nodos.

24. "**AEROPUERTOS XXI**" requiere de un proceso de los vuelos de un día desde y hacia diferentes destinos, contando con los siguientes archivos :

a) **PASAJEROS**, ord. x Nro.Vuelo del día., conteniendo : Nro.Vuelo (5 díg.), Apel.Nom.(20 car.), Nro.DNI (8 díg.), Nacional.(20 car.), Fec.Nac.(8 díg. aaaammdd), Clase (0='Especial',1='Primera',2='Común'), Sexo ('F', 'M').

b) **VUELOS**, con 100.000 registros, en donde: Nro.de Vuelo = dir.en el archivo, conteniendo : Nro.de Vuelo, Cód.Aerop.Origen, Cód.Aerop.Destino, Hora Salida (4 díg. = hhmm), Tiempo (4 díg. = hhmm).

Nota: Nro.Vuelo = 0, significa que no es un vuelo válido para el día. El intervalo de tiempo entre un vuelo y otro no es menor a 60 segundos y no existe posibilidad de solaparse. El primer vuelo llega/sale a las 00:00 y el último vuelo llega/sale a las 23:59 de existir un vuelo en esos horarios por cada aeropuerto.

c) **AEROPUERTOS**, con 100 regs. Relac. 1:1 Cód.Aerop. = dir.arch., conteniendo: Nom.Aerop. (20 car.).

d) **TARIFAS,** de cada clase y destinos (real x 3), sin repetir el mismo par de aeropuertos, ordenado de 0-1 a 0-99, 1-2 a 1-99, ... , 98-99.

Se pide :

1. Emitir un listado de las **Salidas** del día, de todos los vuelos de "AEROPUERTOS XXI", ord. x Cód.Aerop. creciente y Nro.Vuelo decreciente de las Salidas de todos los aeropuertos:

SALIDAS DE VUELOS
CÓD.AEROP.: 99 AEROP. : X(20) (Origen)
**** CÓD.AEROP.: 99 AEROP.** : X(20) (Destino)
**** NRO.DE VUELO**: 99999 **HORA**: hh:mm (de salida)

ORDEN	**APELLIDO, NOMBRE**	**Nro. Doc.**	**Fec. Nac.**
999	XXXXXXXXXXXXXXXX XX	99.999.999	99/99/9999

2. Grabar en un archivo de **Llegadas** del día, de todos los vuelos de cada aeropuerto con igual orden que el punto 1., con el siguiente formato de registro: Cód.Aero.Orig, CódAero.Dest., Nro.Vuelo, Hora de llegada (hhmm), Cant.Hombres, Cant.Mujeres, Importe de cada clase.

RESTRICCIONES:
Memoria estática: 40.000 bytes. **Memoria en disco**: Solo para el punto 2. **Memoria dinámica**: 1 nodo por cada Nro.Vuelo sin repetir. **Tamaño del nodo**: 8 bytes.

25. Una empresa de turismo representante de una compañía aeronáutica, para gestionar sus viajes en el mes de enero, reserva con anticipación una cierta cantidad de pasajes aéreos a diferentes ciudades del mundo (máximo un vuelo por ciudad/día). Se cuenta con los siguientes archivos :

a) **RESERVAS** (sin orden), donde cada registro contiene: Código de Ciudad (3 díg.), Día de Salida (2 díg.), Hora de Salida (6 díg.), Total de Pasajes Reservados (3 díg.), Nro. de Vuelo (4 díg.), Precio de cada Pasaje (real).

b) **PEDIDOS** (sin orden), con los pasajes solicitados por sus clientes, conteniendo : Código de Ciudad, Cantidad Solicitada (3 díg.), Día de Salida, Nro. de Documento (8 díg.).

Desarrollar la estrategia, algoritmo y codificación del programa que :

1. Imprima para el día en que se rechazaron más pedidos, el siguiente listado de rechazos (ordenado por Código de Ciudad) :

DIA DE MAXIMO RECHAZO 99

NRO. DOCUMENTO	COD. CIUDAD	CANT. SOLICITADA
99999999	9999	999

2. Grabe un archivo **VENTAS.DAT** con los pedidos aceptados (registro de igual tipo que el de PEDIDOS más un campo Importe -resultado de Cant. Solicitada * Precio-, ordenado por día y Cód. de Ciudad).

NOTA :
Memoria estática: disponible 309 Kb.
Memoria dinámica: fileSize(Pedidos) * 8. Tamaño del nodo 8 bytes.
Espacio en disco : utilizar menos que el utilizado por PEDIDOS.DAT * 1,6.

26. Una empresa de ventas domiciliaria de electrodomésticos que requiere conocer el desempeño de sus vendedores y clientes, cuenta con :

a) **VENTAS** (ordenado por mes/día de venta), con las ventas del último año donde c/registro contiene:

Nro.de Cliente (7 díg.), Mes/día de venta (4 díg.), Código de Vendedor (6 car.), Importe de la Venta.

b) **CLIENTES** (ordenado por Nro. de Cliente), donde c/registro contiene :

Nro. de Cliente, Domicilio (40 car.), Apellido y Nombre (30 car.), Nro. de CUIT (11 díg.).

Desarrollar la estrategia, algoritmo y codificación del programa que :

1. Imprima un listado de ventas mensuales de cada vendedor ordenado por código de vendedor, mes y día (pero sólo para los meses en los cuales el importe total del mes supera al promedio general del mes).

VENDEDOR : X(6) **MES** : X(10)

DIA	APELLIDO, NOMBRE -del cliente-	IMPORTE
99	X(30)	999999.99

IMPORTE TOTAL : 9999999.99

2. Un listado de los clientes que le compraron a dos o más vendedores según se indica :

APELLIDO, NOMBRE –del cliente-	IMPORTE TOTAL
X(30)	9999999.99

NOTA : Los vendedores son 200 como máximo.
Se dispone de 98 kb. De memoria. Tamaño máximo de nodo = 8 bytes.
Máximo espacio disponible en disco filesize(Clientes) * 14 bytes.

27. Una empresa prestadora de servicios de internet, requiere facturar a sus usuarios y para ello dispone de los siguientes archivos :

a) **CONEXIONES** (orden cronológico), en el que cada registro contiene : Día del mes (1 a 31), Número de usuario (8 díg.), Código de destino (16 car.), Cantidad de minutos (3 díg.).

b) **CATEGORIA** (sin orden), en el que cada registro contiene : Categoría, Minutos sin cargo (4 díg.), Importe de abono, Importe por minuto adicional.

c) **USUARIOS** (ordenado por usuario), en el que cada registro contiene : Número de usuario, Apellido y Nombre (34 car.), Domicilio (49 car.), Categoría (2 díg.).

Desarrollar la estrategia, algoritmo y codificación del programa que :

1. Imprima la factura de cada usuario (ordenado por su número), según se indica:
   USUARIO NRO.: 99999999 **APELLIDO Y NOMBRE** : X(34)
   DOMICILIO : X(49)

ABONO (999 minutos sin cargo)		999.99
ADICIONAL 999 x imp.adic.	+	999.99
TOTAL		9999.99

2. Imprima ordenado por código de destino y día todas las conexiones individuales de usuarios que duraron más de 360 minutos, según se indica:
   CODIGO DE DESTINO: X(16)

DIA	**APELLIDO Y NOMBRE USUARIO**	**MINUTOS**
99	X(34)	9999

NOTA : Hay 130 Kb. de memoria estática. Nodo máximo = 8 bytes. No hay espacio en disco. Cantidad máxima de destinos 5000.

Ejercicios con Recursividad

En todos los casos se solicita crear una función para:

1. Hallar el factorial de un número n.
2. Obtener el M.C.D.
3. Encontrar el valor del término n de la sucesión de Fibonacci.
4. Emitir los elementos de una lista.
5. Determinar si una cadena dada es o no un palíndromo.
6. Comprobar si un valor clave se encuentra o no en un array ordenado por el valor de esa clave, retornando la posición en donde se encontró, caso contrario, retornar cero.
7. Calcular la potencia de a elevado a la n.

Estructuras Dinámicas Lineales

Funciones para la Pila –Stack-

```
/**************
*         Id.Programa: G5Ej00aPila.Cpp
*         Autor......: Lic. Hugo Cuello
*         Fecha......: Jul-2005 / Jun-2014
*         Comentario.: Estructura dinamica de tipo Pila (=Stack) -L.I.F.O.-
*             Modulos:
*               Push - Meter
*               Pop  - Sacar
**************/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct sNodo *tPila;
struct sNodo {
  tInfo Info;
  tPila Ant;
};

void Push(tPila &Stack, tInfo valor) {
  tPila pNodo;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pNodo->Ant = Stack;
  Stack = pNodo;
} //Push

void Pop(tPila &Stack, tInfo &valor) {
  tPila pElim;

  pElim = Stack;
  valor = Stack->Info;
  Stack = Stack->Ant;
  delete pElim;
} //Pop
```

Funciones para la Cola –Queue-

```
/**************
*        Id.Programa: G5Ej00bCola.Cpp
*        Autor......: Lic. Hugo Cuello
*        Fecha......: Jul-2005 / Jun-2014
*        Comentario.: Estructura dinamica de tipo Cola (=Queue) -F.I.F.O.-
*            Modulos:
*              Enqueue - Agregar
*              Dequeue - Suprimir
**************/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct sNodo *tCola;
struct sNodo {
  tInfo Info;
  tCola Sgte;
};

void Enqueue(tCola &qFte, tCola &qFin, tInfo valor) {
  tCola pNodo;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pNodo->Sgte = NULL;
  if (!qFte)
    qFte = pNodo;
  else
    qFin->Sgte = pNodo;
  qFin = pNodo;
} //Enqueue - Agregar

void Dequeue(tCola &qFte, tCola &qFin, tInfo &valor) {
  tCola pElim;

  pElim = qFte;
  valor = qFte->Info;
  qFte = qFte->Sgte;
  if (!qFte)
    qFin = NULL;
  delete pElim;
```

```
} //Dequeue - Suprimir
```

Funciones para la Lista –List- (primera parte)

```
/*
        Id.Programa: Lista.Cpp
        Autor......: Lic. Hugo Cuello
        Fecha......: Jul-2005 / Jun-2014
        Comentario.: Estructura dinamica de tipo Lista (=List) ordenada -Freedom-.
          Modulos:
            InsertaNodo    "Modulo despachador"
            InsertaInicio  "Si Lista es Vacia Inserta primero sino por el frente o inicio"
            InsertaEnMedio "Inserta despues del primero, no antes, incluye por final"
            SuprimeNodo    "Elimina un nodo, en cualquier punto de la Lista, sino enc.
no."
*/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct tNodo *tpPCL;
struct tNodo {
  tInfo Info;
  tpPCL Sgte;
};
```

```
void InsertaInicio(tpPCL &List, tInfo valor) {
  tpPCL pNodo;

  pNodo = new tNodo;
  pNodo->Info = valor;
  pNodo->Sgte = List;
  List = pNodo;
} //InsertaInicio
```

```
void InsertaEnMedio(tpPCL &List, tInfo valor) {
  tpPCL pNodo,
        pAct;
```

```
  pNodo = new tNodo;
  pNodo->Info = valor;
  pAct = List;
  while (pAct->Sgte && valor > pAct->Sgte->Info)
      pAct = pAct->Sgte;
  pNodo->Sgte = pAct->Sgte;
  pAct->Sgte = pNodo;
} //InsertaEnMedio
```

```
void InsertaNodo(tpPCL &List, tInfo valor) {
  if (!List || valor < List->Info)
      InsertaInicio(List, valor);
  else
      InsertaEnMedio(List, valor);
} //InsertaNodo
```

```
void SuprimeNodo(tpPCL &List, tInfo valor) {
  tpPCL pAct,
            pAnt;

  pAct = List;
  pAnt = NULL;
  while (pAct && valor > pAct->Info) {
      pAnt = pAct;
      pAct = pAct->Sgte;
  }
  if (pAct && valor == pAct->Info) {
      if (pAnt)
          pAnt->Sgte = pAct->Sgte;
      else
          List = pAct->Sgte;
      delete pAct;
  }
} //SuprimeNodo
```

```
int main () {
  int   num;
        tpPCL Lista;

  clrscr();
  Lista = NULL;
  for(int i = 1; i <= 7; i++) {
      cout << "Ing. valor: ";
```

```
    cin >> num;
    InsertaNodo(Lista, num);
  }
  cout << "-----------x----------" << endl;
  while (Lista) {
    num = Lista->Info;
    SuprimeNodo(Lista, num);
    cout << "valor: " << num << endl;
  }
  return 0;
} //main
```

Módulos para Listas (segunda parte)

```
tpPCL iif(bool exprLog, tpPCL ptr1, tpPCL ptr2) {
   if (exprLog)
      return ptr1;
   else
      return ptr2;
} // iif
```

```
tpPCL ExisteNodo(tpPCL &Lista, tpPCL &pAnt, tclv clv) {
   tpPCL pAct = Lista;

   pAnt = NULL;
   while (pAct && clv > pAct->Info.clv) {
      pAnt = pAct;
      pAct = pAct->Sgte;
   }
   return iif(pAct && clv == pAct->Info.clv, pAct, NULL);
} // ExisteNodo
```

```
void CrearNodo(tpPCL &Lista, tpPCL &pNodoAnt, tInfo valor) {
   tpPCL pNodo;

   if (!pNodoAnt)
      InsertaInicio(Lista,valor);
   else {
      pNodo = new tNodo;
      pNodo->Info = valor;
      pNodo->Sgte = pNodoAnt->Sgte;
      pNodoAnt->Sgte = pNodo;
   }
```

```
} // CrearNodo
```

```
tpPCL BuscarNodo(tpPCL &Lista, tClv clv) {
   tpPCL pAct = Lista;

   while (pAct != NULL && clv != pAct->Info.clv)
      pAct = pAct->Sgte;
   return pAct;
} // BuscarNodo
```

```
void InsertaUltimoNodo(tpPCL &Lista, tInfo valor) {
   tpPCL pNodo,
         pAct;

   pNodo = new tNodo;
   pNodo->Info = valor;
   pNodo->Sgte = NULL;
   if (Lista == NULL)
      Lista = pNodo;
   else {
      pAct = Lista;
      while (pAct->Sgte != NULL)
         pAct = pAct->Sgte;
      pAct->Sgte = pNodo;
   }
} // InsertaUltimoNodo
```

```
tpPCL BuscarPosNodo(tpPCL &Lista, word pos) {
   tpPCL pAct = Lista;
   int     i;

   for (i = 1; i < pos; i++)
      pAct = pAct->Sgte;
   return pAct;
} // BuscarPosNodo
```

```
void ReInsertaNodo(tpPCL &Lista, tpPCL &pNodo) {
   tpPCL pAct;

   if (Lista == NULL || pNodo->Info.clv < Lista->Info.clv) {
      pNodo->Sgte = Lista;
      Lista = pNodo;
   }
```

```
   else {
      pAct = Lista;
      while (pAct->Sgte != NULL && pNodo->Info.clv > pAct->Sgte->Info.clv)
         pAct = pAct->Sgte;
      pNodo->Sgte = pAct->Sgte;
      pAct->Sgte = pNodo;
   }
} // ReInsertaNodo
```

```
void ReOrdenarLst(tpPCL &Lista) {
   tpPCL ListaAux = NULL,
         pNodo;
   while (Lista != NULL) {
      pNodo = Lista;
      Lista = Lista->Sgte;
      ReInsertaNodo(ListaAux,pNodo);
   }
   Lista = ListaAux;
} // ReOrdenarLst
```

```
void SacarPrimerNodo(tpPCL &Lista, tInfo &valor) {
   tpPCL pElim;

   pElim = Lista;
   valor = Lista->Info;
   Lista = Lista->Sgte;
   delete pElim;
} // SacarPrimerNodo
```

Vector de punteros a Listas de días con importes acumulados

```
/**************
*        Id. Programa: G5Ej04mVectorAuxMes&12Lst.Cpp
*        Autor.......: Lic. Hugo Cuello
*        Fecha.......: octubre-2014
*        Comentario..: Archivo de Gastos con:
*              Mes, Dia, Importe  (desordenado).
*            Se pide:
*              Emitir un listado ord. x Mes y Dia  con
*              importes acumulados por dia de cada
*              mes.
```

```
*             Estructuras de Datos:
*               Vector auxiliar con 12 componentes, una
*               por cada mes del año, de punteros a listas
*               de Días (total de nodos 365, uno por cada
*               día del año).
***************/

#include <iomanip>
#include <fstream>
#include "h:\tc\prgsFtes\Include\DG5E04GtArcAuxMeslstDias.hpp"
using namespace std;

const
  MESES = 12,
  DIAS  = 31;

// Se define la macro readf con 2 argumentos-parametros
// f : Nombre Logico del archivo
// r : Buffer de almacenamiento
#define readf(f,r)  fread (&(r),sizeof((r)),1,(f))
#define writef(f,r) fwrite(&(r),sizeof((r)),1,(f))

typedef unsigned char byte;
typedef unsigned short word;
typedef char *pChar;
typedef tListaDia tvMes[DIAS + 1];

struct sGasto{
  float Impo;
  byte  Mes,
        Dia;
};

void Abrir(FILE **Gtos) {
          *Gtos  = fopen("GastosMD.Dat","rb");
} // Abrir

void ArmarED(tvMes vM) {
  for (register i = 1; i <= MESES; i++)
    vM[i] = NULL;
} // ArmarED

byte CantDiasMes(byte mes) {
  switch(mes) {
```

```
    case  4:
    case  6:
    case  9:
    case 11:
      return 30;
    case  2:
      return 28;
    default:
      return 31;
  }
} // CantDiasMes

word CantDias(byte mes) {
  word sumDias = 0;

  for (register i = 1; i < mes; i++)
    switch (i) {
      case  4:
      case  6:
      case  9:
      case 11:
        sumDias += 30;
        break;
      case  2:
        sumDias += 28;
        break;
      default:
        sumDias += 31;
    }
  return sumDias;
} //CantDias

void AcumGto(tvMes vM, sGasto rGto) {
  tListaDia pNodoDia,
            pAntDia;
  sInfoDia  rInfoD;

  pNodoDia = ExisteNodo(vM[rGto.Mes],pAntDia,rGto.Dia);
  if (!pNodoDia) {
    rInfoD.Dia = rGto.Dia;
    rInfoD.ImpAcum = rGto.Impo;
    CrearNodo(vM[rGto.Mes],pAntDia,rInfoD);
  }
```

```
 else
   pNodoDia->Info.ImpAcum += rGto.Impo;
}  // AcumGto

void Listado(tvMes vM) {
          float totAnu = 0.0,
       totMes;
       pChar mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL","MAYO",
                         "JUNIO","JULIO","AGOSTO","SEPTIEMBRE",
                         "OCTUBRE","NOVIEMBRE","DICIEMBRE"
                       };
       tListaDia pLstDia;
       sInfoDia  rInfoD;

          freopen("LstGtosMDm.Lst","w",stdout);
          cout << "(m) Listado de Gastos por Mes y Dia Acumulado (m)" << endl;
          cout.setf(ios::fixed);
          cout.precision(2);
          for (register i = 1; i <= MESES; i++) {
   totMes = 0.0;
   cout << "Mes: " << setw(2) << i << " ";
   cout.setf(ios::left);
   cout << mesCad[i] << endl;
   cout.unsetf(ios::left);
   cout << "        Dia          Imp.Acum." << endl;
   while (vM[i]) {
                       SacarPrimerNodo(vM[i],rInfoD);
                       if (rInfoD.ImpAcum > 0.0) {
                         totMes += rInfoD.ImpAcum;
                         cout << setw(9) << "";
                         cout << setw(2) << (short) rInfoD.Dia;
                         cout << setw(9) << "";
                         cout << "$ ";
                         cout << setw(10) << rInfoD.ImpAcum << endl;
                       }
   }
   totAnu += totMes;
   cout << endl << "Total del Mes:      $";
   cout.fill('*');
   cout << setw(11) << totMes << endl << endl;
   cout.fill(' ');
 }
          cout << "Total Anual:        $";
          cout.fill('*');
```

```
        cout << setw(11) << totAnu << endl;
        freopen("CON","w",stdout);
} // Listado

int main() {
        FILE   *Gastos;
        sGasto rGasto;
        tvMes  vMes;

        Abrir(&Gastos);
        ArmarED(vMes);
        while (readf(Gastos,rGasto))
   AcumGto(vMes,rGasto);
  Listado(vMes);
        fclose(Gastos);

        return 0;
} // main
```

DG5E04GtArcAuxMesLstDias.hpp

```
/**************
*       Id.Programa: DG5E04GtArcAuxMesLstDias.hpp
*       Autor......: Lic. Hugo Cuello
*       Fecha......: octubre-2013
*       Comentario.: Modulos de Listas dinamicas, con:
*                                                       Archivo Auxiliar de Meses de
punteros a
*           Lista de días sin pre-Armar.
*                                                       ListaDia INFO:
*             byte Dia,
*             float ImpAcum.
*           Programas: --> G5Ej04lArcAuxMes&12Lst.Cpp
*                   --> G5Ej04mVectorAuxMes&12lst.Cpp
***************/

typedef unsigned char byte;
typedef struct sNodoDia *tListaDia;

struct sInfoDia {
        byte  Dia;
        float ImpAcum;
```

```
};

struct sNodoDia {
	sInfoDia  Info;
	tListaDia Sgte;
};

tListaDia iif(bool exprLog, tListaDia ptr1, tListaDia ptr2) {
	if (exprLog)
		return ptr1;
	else
		return ptr2;
} // iif

tListaDia ExisteNodo(tListaDia &Lista,tListaDia &pAnt,byte dia) {
	tListaDia pAct;

	pAnt = NULL;
	pAct = Lista;
	while (pAct != NULL && dia > pAct->Info.Dia) {
		pAnt = pAct;
		pAct = pAct->Sgte;
	}
	return iif(pAct && dia == pAct->Info.Dia,pAct,NULL);
} // ExisteNodo

void InsertaInicio(tListaDia &Lista, sInfoDia valor) {
	tListaDia pNodo;

	pNodo = new sNodoDia;
	pNodo->Info = valor;
	pNodo->Sgte = Lista;
	Lista = pNodo;
} // InsertaInicio

void CrearNodo(tListaDia &Lista, tListaDia &pAnt, sInfoDia valor) {
	tListaDia pNodo;

	if (!pAnt)
		InsertaInicio(Lista,valor);
	else {
		pNodo = new sNodoDia;
		pNodo->Info = valor;
		pNodo->Sgte = pAnt->Sgte;
```

```
                pAnt->Sgte = pNodo;
        }
} // CrearNodo

void SacarPrimerNodo(tListaDia &Lista, sInfoDia &valor) {
        tListaDia pElim;

        pElim = Lista;
        valor = Lista->Info;
        Lista = Lista->Sgte;
        delete pElim;
} // SacarPrimerNodo
```

Lista única de meses y dias

```
/**************
*       Id. Programa: G5Ej04gListaMesDia.Cpp
*       Autor.......: Lic. Hugo Cuello
*       Fecha.......: octubre-2014
*       Comentario..: Archivo de Gastos con:
*             Mes, Dia, Importe  (desordenado).
*           Se pide:
*             Emitir un listado ord. x Mes y Dia  con
*             importes acumulados por dia de cada mes.
*           Estructuras de Datos:
*             Lista unica pre-Armada con 1 nodo por
*             cada dia anual.
***************/

#include <iomanip>
#include <fstream>
#include "c:\tc\prgsFtes\Include\DG5E04GtLstU.hpp"
using namespace std;

const
  MESES = 12;

// Se define la macro readf con 2 argumentos-parametros
// f : Nombre Logico del archivo
// r : Buffer de almacenamiento
#define readf(f,r) fread(&(r),sizeof((r)),1,(f))
```

```
typedef unsigned char byte;
typedef char *pChar;

struct sGasto{
  float Impo;
  byte  Mes,
        Dia;
};

void Abrir(FILE **Gtos) {
          *Gtos = fopen("GastosMD.Dat","rb");
} // Abrir

void ArmarED(tListaMD &LstM) {
  for (register i = 1; i <= 365; i++)
    InsertaInicio(LstM,0.0);
} // ArmarEd

byte CantDiasMes(byte mes) {
  switch(mes) {
    case  4:
    case  6:
    case  9:
    case 11:
      return 30;
    case  2:
      return 28;
    default:
      return 31;
  }
} // CantDiasMes

word CantDias(byte mes) {
  word sumDias = 0;

  for (register i = 1; i < mes; i++)
    switch (i) {
      case  4:
      case  6:
      case  9:
      case 11:
        sumDias += 30;
        break;
      case  2:
```

```
      sumDias += 28;
      break;
    default:
      sumDias += 31;
   }
  return sumDias;
} //CantDias

void AcumGto(tListaMD &LstM, sGasto rGto) {
  tListaMD pNodoMD;

  pNodoMD = BuscarPosNodo(LstM, CantDias(rGto.Mes) + rGto.Dia);
  pNodoMD->ImpAcum += rGto.Impo;
}  // AcumGto

void Listado(tListaMD &LstMD) {
          float totAnu = 0.0,
      totMes,
      ImpAcum;
      pChar mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL","MAYO",
                        "JUNIO","JULIO","AGOSTO","SEPTIEMBRE",
                        "OCTUBRE","NOVIEMBRE","DICIEMBRE"
                      };

          freopen("LstGtosMDg.Lst","w",stdout);
          cout << "(g) Listado de Gastos por Mes y Dia Acumulado (g)" << endl << endl;
          cout.setf(ios::fixed);
          cout.precision(2);
          for (register i = 1; i <= MESES; i++) {
   totMes = 0.0;
   cout << "Mes: " << setw(2) << i << " ";
   cout.setf(ios::left);
   cout << mesCad[i] << endl;
   cout.unsetf(ios::left);
   cout << "        Dia          Imp.Acum." << endl;
   for (register j = 1; j <= CantDiasMes(i); j++) {
                      SacarPrimerNodo(LstMD,ImpAcum);
                      if (ImpAcum > 0.0) {
                        totMes += ImpAcum;
                        cout << setw(9) << "";
                        cout << setw(2) << j;
                        cout << setw(9) << "";
                        cout << "$ ";
```

```
                    cout << setw(10) << ImpAcum << endl;
                  }
   }
   totAnu += totMes;
   cout << endl << "Total del Mes:      $";
   cout.fill('*');
   cout << setw(11) << totMes << endl << endl;
   cout.fill(' ');
  }
        cout << "Total Anual:       $";
        cout.fill('*');
        cout << setw(11) << totAnu << endl;
        freopen("CON","w",stdout);
} // Listado

int main() {
        FILE     *Gastos;
        sGasto   rGasto;
        tListaMD ListaMD = NULL;

        Abrir(&Gastos);
        ArmarED(ListaMD);
        while (readf(Gastos,rGasto))
      AcumGto(ListaMD,rGasto);
   Listado(ListaMD);
        fclose(Gastos);

        return 0;
} // main
```

DG5E04GtLstU.hpp

```
/*************
*       Id.Programa: DG5E04GtLstU.hpp
*       Autor......: Lic. Hugo Cuello
*       Fecha......: octubre-2014
*       Comentario.: Modulos de Listas dinamicas, con:
*                        ListaMes pre-Armada.
*                        INFO:
*                          tListaDia pSlstDia.
*                          ListaDia sin pre-Armar.
*                        INFO:
*                          byte  Dia
*                          float ImpAcum.
*           Polimorfismo en SobreCarga Funcional,
```

```
*           Programa: --> G5Ej04gListaMesDia.Cpp
**************/

typedef unsigned short word;
typedef float tInfoMD;
typedef struct sNodoMD *tListaMD;
struct sNodoMD {
        tInfoMD  ImpAcum;
        tListaMD Sgte;
};

tListaMD BuscarPosNodo(tListaMD &LstMD, word n) {
  tListaMD pAct;

  pAct = LstMD;
  for (register i = 1; i < n; i++)
    pAct = pAct->Sgte;
  return pAct;
} // BuscarPosNodo

void InsertaInicio(tListaMD &LstMD, tInfoMD valor) {
        tListaMD pNodo;

        pNodo = new sNodoMD;
        pNodo->ImpAcum = valor;
        pNodo->Sgte = LstMD;
        LstMD = pNodo;
} // InsertaInicio

void SacarPrimerNodo(tListaMD &LstMD, tInfoMD &valor) {
        tListaMD pElim;

        pElim = LstMD;
        valor = LstMD->ImpAcum;
        LstMD = LstMD->Sgte;
        delete pElim;
} // SacarPrimerNodo
```

Lista y SubListas Opc. 1

```
/**************
*       Id. Programa: G5Ej04aListaSubLstOpc1.Cpp
*       Autor.......: Lic. Hugo Cuello
*       Fecha.......: octubre-2014
```

```
*         Comentario..: Archivo de Gastos con:
*               Mes, Dia, Importe  (desordenado).
*             Se pide:
*               Emitir un listado ord. x Mes y Dia  con
*               importes acumulados por dia de cada mes.
*             Estructuras de Datos:
*                Lista y SubLista, ambas pre-armadas:
*                 Info de la Lista con:
*                   tListaDia pSlstDia
*                 Info de la SubLista con:
*                   float ImpAcum.
***************/

#include <iomanip>
#include <fstream>
#include "c:\tc\prgsFtes\Include\DG5E04GtOpc1.hpp"
using namespace std;

// Se define la macro readf con 2 argumentos-parametros
// f : Nombre Logico del archivo
// r : Buffer de almacenamiento
#define readf(f,r) fread(&(r),sizeof((r)),1,(f))

const
  MESES = 12;

typedef char *pChar;

struct sGasto{
  float Impo;
  byte  Mes,
        Dia;
};

void Abrir(FILE **Gtos) {
          *Gtos = fopen("GastosMD.Dat","rb");
} // Abrir

byte CantDiasMes(byte mes) {
  switch(mes) {
    case  4:
    case  6:
    case  9:
    case 11:
```

```
      return 30;
    case  2:
      return 28;
    default:
      return 31;
  }
} // CantDiasMes

void ArmarED(tListaMes &LstM) {
  for (register i = MESES; i >= 1; i--) {
    InsertaInicio(LstM,NULL);
    for (register j = 1; j <= CantDiasMes(i); j++)
      InsertaInicio(LstM->pSlstDia,0.0);
  }
} // ArmarEd

void AcumGto(tListaMes &LstM, sGasto rGto) {
  tListaMes pNodoMes;
  tListaDia pNodoDia;

  pNodoMes = BuscarPosNodo(LstM, rGto.Mes);
  pNodoDia = BuscarPosNodo(pNodoMes->pSlstDia,rGto.Dia);
  pNodoDia->ImpAcum += rGto.Impo;
}  // AcumGto

void Listado(tListaMes &LstM) {
          float totAnu = 0.0,
       totMes,
       ImpAcum;
       tListaDia pSlstDia;
       pChar mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL","MAYO",
                         "JUNIO","JULIO","AGOSTO","SEPTIEMBRE",
                         "OCTUBRE","NOVIEMBRE","DICIEMBRE"
                       };

          freopen("LstGtosMDa.Lst","w",stdout);
          cout << "(a) Listado de Gastos por Mes y Dia Acumulado (a)" << endl << endl;
          cout.setf(ios::fixed);
          cout.precision(2);
          for (register i = 1; i <= MESES; i++) {
                    SacarPrimerNodo(LstM,pSlstDia);
                    totMes = 0.0;
                    cout << "Mes: " << setw(2) << i << " ";
```

```
                cout.setf(ios::left);
                cout << mesCad[i] << endl;
                cout.unsetf(ios::left);
                cout << "        Dia          Imp.Acum." << endl;
                for (register j = 1; j <= CantDiasMes(i); j++) {
                        SacarPrimerNodo(pSlstDia,ImpAcum);
                        if (ImpAcum > 0.0) {
                                totMes += ImpAcum;
                                cout << setw(9) << "";
                                 cout << setw(2) << j;
                                cout << setw(9) << "";
                                cout << "$ ";
                                 cout << setw(10) << ImpAcum << endl;
                         }
                }
                totAnu += totMes;
                cout << endl << "Total del Mes:      $";
                cout.fill('*');
                cout << setw(11) << totMes << endl << endl;
                cout.fill(' ');
        }
        cout << "Total Anual:       $";
        cout.fill('*');
        cout << setw(11) << totAnu << endl;
        freopen("CON","w",stdout);
} // Listado

int main() {
        FILE      *Gastos;
        sGasto    rGasto;
        tListaMes ListaMes = NULL;

        Abrir(&Gastos);
        ArmarED(ListaMes);
        while (readf(Gastos,rGasto))
        AcumGto(ListaMes,rGasto);
         Listado(ListaMes);
        fclose(Gastos);

        return 0;
} // main
```

DG5E04GtOpc1.hpp

```
/*************
```

```
*        Id.Programa: DG5E04GtOpc1.hpp
*        Autor......: Lic. Hugo Cuello
*        Fecha......: octubre-2014
*        Comentario.: Modulos de Listas dinamicas, con:
*                ListaMes INFO:
*                 tListaDia pSlstDia.
*                ListaDia INFO:
*                float ImpAcum.
*                Ambas estructuras Pre-Armadas.
*                Polimorfismo en SobreCarga Funcional,
*                Programa: --> G5Ej04aListaSubLstOpc1.Cpp
**************/

typedef unsigned char byte;
typedef struct       sNodoMes *tListaMes;
typedef struct       sNodoDia *tListaDia;
typedef float        tInfoDia;
typedef tListaDia     tInfoMes;

struct sNodoMes {
        tInfoMes  pSlstDia;
        tListaMes Sgte;
};

struct sNodoDia {
        tInfoDia  ImpAcum;
        tListaDia Sgte;
};

template <class T>
T BuscarPosNodo(T &LstX, byte mes) {
  T pAct;

  pAct = LstX;
  for (register i = 1; i < mes; i++)
    pAct = pAct->Sgte;
  return pAct;
} // BuscarPosNodo

/*
tListaDia BuscarPosNodo(tListaDia &LstD, byte dia) {
  tListaDia pAct;
```

```
  pAct = LstD;
  for (register i = 1; i < dia; i++)
    pAct = pAct->Sgte;
  return pAct;
} // BuscarPosNodo
*/

void InsertaInicio(tListaMes &LstM, tListaDia valor) {
        tListaMes pNodo;

        pNodo = new sNodoMes;
        pNodo->pSlstDia = valor;
        pNodo->Sgte = LstM;
        LstM = pNodo;
} // InsertaInicio

void InsertaInicio(tListaDia &LstD, tInfoDia valor) {
        tListaDia pNodo;

        pNodo = new sNodoDia;
        pNodo->ImpAcum = valor;
        pNodo->Sgte = LstD;
        LstD = pNodo;
} // InsertaInicio

void SacarPrimerNodo(tListaMes &LstM, tInfoMes &valor) {
        tListaMes pElim;

        pElim = LstM;
        valor = LstM->pSlstDia;
        LstM = LstM->Sgte;
        delete pElim;
} // SacarPrimerNodo

void SacarPrimerNodo(tListaDia &LstD, tInfoDia &valor) {
        tListaDia pElim;

        pElim = LstD;
        valor = LstD->ImpAcum;
        LstD = LstD->Sgte;
        delete pElim;
} // SacarPrimerNodo
```

Lista y SubListas Opc. 2

```
/***************
*        Id. Programa: G5Ej04bListaSubLstOpc2.Cpp
*        Autor.......: Lic. Hugo Cuello
*        Fecha.......: octubre-2014
*        Comentario..: Archivo de Gastos con:
*              Mes, Dia, Importe  (desordenado).
*            Se pide:
*              Emitir un listado ord. x Mes y Dia  con
*              importes acumulados por dia de cada
*              mes.
*            Estructuras de Datos: Lista pre-Armada
*              y SubLista, sin pre-armar:
*                Info de la Lista con:
*                  tListaDia pSlstDia
*                Info de la SubLista con:
*                  byte  Dia
*                  float ImpAcum.
****************/

#include <iomanip>
#include <fstream>
#include "h:\tc\prgsFtes\Include\DG5E04GtOpc2.hpp"
using namespace std;

const
  MESES = 12;

// Se define la macro readf con 2 argumentos-parametros
// f : Nombre Logico del archivo
// r : Buffer de almacenamiento
#define readf(f,r) fread(&(r),sizeof((r)),1,(f))

typedef char *pChar;

struct sGasto{
  float Impo;
  byte  Mes,
        Dia;
};

void Abrir(FILE **Gtos) {
          *Gtos = fopen("GastosMD.Dat","rb");
} // Abrir
```

```
byte CantDiasMes(byte mes) {
  switch(mes) {
    case  4:
    case  6:
    case  9:
    case 11:
      return 30;
    case  2:
      return 28;
    default:
      return 31;
  }
} // CantDiasMes

void ArmarED(tListaMes &LstM) {
  for (register i = 1; i <= MESES; i++)
    InsertaInicio(LstM,NULL);
} // ArmarEd

void AcumGto(tListaMes &LstM, sGasto rGto) {
  tListaMes pNodoMes;
  tListaDia pNodoDia,
            pAntDia;
  tInfoDia  rInfoD;

  pNodoMes = BuscarPosNodo(LstM, rGto.Mes);
  pNodoDia = ExisteNodo(pNodoMes->pSlstDia,pAntDia,rGto.Dia);
  if (!pNodoDia) {
    rInfoD.Dia = rGto.Dia;
                    rInfoD.ImpAcum = rGto.Impo;
                    CrearNodo(pNodoMes->pSlstDia,pAntDia,rInfoD);
  }
  else
    pNodoDia->Info.ImpAcum += rGto.Impo;
}  // AcumGto

void Listado(tListaMes &LstM) {
            float totAnu = 0.0,
        totMes,
        ImpAcum;
        tListaDia pSlstDia;
        tInfoDia  rInfoD;
        pChar mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL","MAYO",
```

```
                    "JUNIO","JULIO","AGOSTO","SEPTIEMBRE",
                    "OCTUBRE","NOVIEMBRE","DICIEMBRE"
                  };

        freopen("LstGtosMDb.Lst","w",stdout);
        cout << "(b) Listado de Gastos por Mes y Dia Acumulado (b)" << endl << endl;
        cout.setf(ios::fixed);
        cout.precision(2);
        for (register i = 1; i <= MESES; i++) {
  SacarPrimerNodo(LstM,pSlstDia);
  totMes = 0.0;
  cout << "Mes: " << setw(2) << i << " ";
  cout.setf(ios::left);
  cout << mesCad[i] << endl;
  cout.unsetf(ios::left);
  cout << "        Dia          Imp.Acum." << endl;
  while (pSlstDia) {
                  SacarPrimerNodo(pSlstDia,rInfoD);
                  if (rInfoD.ImpAcum > 0.0) {
                    totMes += rInfoD.ImpAcum;
                    cout << setw(9) << "";
                    cout << setw(2) << (short) rInfoD.Dia;
                    cout << setw(9) << "";
                    cout << "$ ";
                    cout << setw(10) << rInfoD.ImpAcum << endl;
                  }
  }
  totAnu += totMes;
  cout << endl << "Total del Mes:      $";
  cout.fill('*');
  cout << setw(11) << totMes << endl << endl;
  cout.fill(' ');
 }
        cout << "Total Anual:       $";
        cout.fill('*');
        cout << setw(11) << totAnu << endl;
        freopen("CON","w",stdout);
} // Listado

int main() {
        FILE      *Gastos;
        sGasto    rGasto;
        tListaMes ListaMes = NULL;
```

```
        Abrir(&Gastos);
        ArmarED(ListaMes);
        while (readf(Gastos,rGasto))
        AcumGto(ListaMes,rGasto);
        Listado(ListaMes);
        fclose(Gastos);

        return 0;
} // main
```

DG5E04GtOpc2.hpp

```
/*************
*       Id.Programa: DG5E04GtOpc2.hpp
*       Autor......: Lic. Hugo Cuello
*       Fecha......: octubre-2014
*       Comentario.: Modulos de Listas dinamicas, con:
*                       ListaMes pre-Armada.
*                       INFO:
*                               tListaDia pSlstDia.
*                               ListaDia sin pre-Armar.
*                       INFO:
*                               byte  Dia
*                               float ImpAcum.
*                       Polimorfismo en SobreCarga Funcional,
*                       Programa: --> G5Ej04bListaSubLstOpc2.Cpp
**************/

typedef unsigned char byte;
typedef struct      sNodoMes *tListaMes;
typedef struct      sNodoDia *tListaDia;
typedef tListaDia     tInfoMes;

struct tInfoDia {
  byte  Dia;
  float ImpAcum;
};

struct sNodoMes {
        tInfoMes  pSlstDia;
        tListaMes Sgte;
};
```

```
struct sNodoDia {
        tInfoDia  Info;
        tListaDia Sgte;
};

tListaMes BuscarPosNodo(tListaMes &LstM, byte mes) {
  tListaMes pAct;

  pAct = LstM;
  for (register i = 1; i < mes; i++)
    pAct = pAct->Sgte;
  return pAct;
} // BuscarPosNodo

void InsertaInicio(tListaMes &LstM, tListaDia valor) {
        tListaMes pNodo;

        pNodo = new sNodoMes;
        pNodo->pSlstDia = valor;
        pNodo->Sgte = LstM;
        LstM = pNodo;
} // InsertaInicio

void InsertaInicio(tListaDia &LstD, tInfoDia valor) {
        tListaDia pNodo;

        pNodo = new sNodoDia;
        pNodo->Info = valor;
        pNodo->Sgte = LstD;
        LstD = pNodo;
} // InsertaInicio

tListaDia iif(bool exprLog, tListaDia ptr1, tListaDia ptr2) {
        if (exprLog)
                return ptr1;
        else
                return ptr2;
} // iif

tListaDia ExisteNodo(tListaDia &LstD,tListaDia &pAnt,byte dia) {
        tListaDia pAct = LstD;

        pAnt = NULL;
```

```
        while (pAct && dia > pAct->Info.Dia) {
                pAnt = pAct;
                pAct = pAct->Sgte;
        }
        return iif(pAct && dia == pAct->Info.Dia,pAct,NULL);
} // ExisteNodo

void InsertaSR(tListaDia pAnt, tInfoDia valor) {
  tListaDia pNodo;

  pNodo = new sNodoDia;
  pNodo->Info = valor;
  pNodo->Sgte = pAnt->Sgte;
  pAnt->Sgte = pNodo;
} // InsertaSR

void CrearNodo(tListaDia &LstD, tListaDia &pAnt, tInfoDia valor) {
        tListaDia pNodo;

        if (!pAnt)
                InsertaInicio(LstD,valor);
        else {
      InsertaSR(pAnt,valor);
        }
} // CrearNodo

void SacarPrimerNodo(tListaMes &LstM, tInfoMes &valor) {
        tListaMes pElim;

        pElim = LstM;
        valor = LstM->pSlstDia;
        LstM = LstM->Sgte;
        delete pElim;
} // SacarPrimerNodo

void SacarPrimerNodo(tListaDia &LstD, tInfoDia &valor) {
        tListaDia pElim;

        pElim = LstD;
        valor = LstD->Info;
        LstD = LstD->Sgte;
        delete pElim;
} // SacarPrimerNodo
```

Lista y Sublistas Opc. 3

```
/**************
*        Id. Programa: G5Ej04cListaSubLstOpc3.Cpp
*        Autor.......: Lic. Hugo Cuello
*        Fecha.......: octubre-2014
*        Comentario..: Archivo de Gastos con:
*              Mes, Dia, Importe  (desordenado).
*            Se pide:
*              Emitir un listado ord. x Mes y Dia  con
*              importes acumulados por dia de cada
*              mes. Estructuras de Datos: Lista y
*              SubLista, ambas sin pre-armar:
*                Info de la Lista con:
*                  byte      Mes
*                  tListaDia pSlstDia
*                Info de la SubLista con:
*                  byte  Dia,
*                  float ImpAcum
***************/

#include <iomanip>
#include <fstream>
#include "h:\tc\prgsFtes\Include\DG5E04GtOpc3.hpp"
using namespace std;

// Se define la macro readf con 2 argumentos-parametros
// f : Nombre Logico del archivo
// r : Buffer de almacenamiento
#define readf(f,r) fread(&(r),sizeof((r)),1,(f))

typedef char *pChar;

struct sGasto{
  float Impo;
  byte  Mes,
        Dia;
};

void Abrir(FILE **Gtos) {
          *Gtos = fopen("GastosMD.Dat","rb");
} // Abrir
```

```
void AcumGto(tListaMes &LstM, sGasto rGto) {
        tListaMes pMes,
        pAntMes;
        tListaDia pDia,
        pAntDia;
        sInfoMes  rInfoM;
        sInfoDia  rInfoD;

        pMes = ExisteNodo(LstM,pAntMes,rGto.Mes);
        if (!pMes) {
   rInfoD.Dia = rGto.Dia;
   rInfoD.ImpAcu = rGto.Impo;
   rInfoM.Mes = rGto.Mes;
   rInfoM.pSlstDia = NULL;
   InsertaInicio(rInfoM.pSlstDia,rInfoD);
   CrearNodo(LstM,pAntMes,rInfoM);
        }
        else {
   pDia = ExisteNodo(pMes->Info.pSlstDia,pAntDia,rGto.Dia);
   if (!pDia) {
                rInfoD.Dia = rGto.Dia;
                rInfoD.ImpAcu = rGto.Impo;
                CrearNodo(pMes->Info.pSlstDia,pAntDia,rInfoD);
   }
   else
                pDia->Info.ImpAcu += rGto.Impo;
        }
}  // AcumGto

void Listado(tListaMes &LstM) {
        float    totAnu = 0.0,
       totMes;
        sInfoMes rInfoM;
        sInfoDia rInfoD;
 pChar    mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL",
                "MAYO","JUNIO","JULIO","AGOSTO",
                "SEPTIEMBRE","OCTUBRE","NOVIEMBRE","DICIEMBRE"
              };

        freopen("LstGtosMDc.Lst","w",stdout);
        cout << "(c) Listado de Gastos por Mes y Dia Acumulado (c)" << endl << endl;
        cout.setf(ios::fixed);
        cout.precision(2);
        while (LstM) {
```

```
   SacarPrimerNodo(LstM,rInfoM);
   totMes = 0.0;
   cout << "Mes: " << setw(2) << (short) rInfoM.Mes << " ";
   cout.setf(ios::left);
   cout << mesCad[rInfoM.Mes] << endl;
   cout.unsetf(ios::left);
   cout << "        Dia          Imp.Acum." << endl;
   while (rInfoM.pSlstDia) {
                    SacarPrimerNodo(rInfoM.pSlstDia,rInfoD);
                    totMes += rInfoD.ImpAcu;
                    cout << setw(9) << "";
                    cout << setw(2) << (short) rInfoD.Dia;
                    cout << setw(9) << "";
                    cout << "$ ";
                    cout << setw(10) << rInfoD.ImpAcu << endl;
   }
   totAnu += totMes;
   cout << endl << "Total del Mes:      $";
   cout.fill('*');
   cout << setw(11) << totMes << endl << endl;
   cout.fill(' ');
  }
         cout << "Total Anual:       $";
         cout.fill('*');
         cout << setw(11) << totAnu << endl;
         freopen("CON","w",stdout);
} // Listado

main() {
         FILE      *Gastos;
         sGasto    rGasto;
         tListaMes ListaMes = NULL;

         Abrir(&Gastos);
         while (readf(Gastos,rGasto))
         AcumGto(ListaMes,rGasto);
         Listado(ListaMes);
         fclose(Gastos);

         return 0;
} // main
```

DG5E04GtOpc3.hpp

```
/**************
*       Id.Programa: DG5E04GtOpc3.hpp
*       Autor......: Lic. Hugo Cuello
*       Fecha......: octubre-2013
*       Comentario.: Modulos de Listas dinamicas, con:
*                                ListaMes INFO: byte Mes, tListaDia pSlstDia.
*                                 ListaDia INFO: byte Dia, float ImpAcum.
*                                Ambas estructuras sin Pre-Armar.
*                                Polimorfismo en SobreCarga Funcional,
*           Programa: --> G5Ej04cListaSubLstOpc3.Cpp
***************/

typedef unsigned char byte;
typedef struct sNodoMes *tListaMes;
typedef struct sNodoDia *tListaDia;

struct sInfoMes {
        byte      Mes;
        tListaDia pSlstDia;
};

struct sNodoMes {
        sInfoMes  Info;
        tListaMes Sgte;
};

struct sInfoDia {
        byte  Dia;
        float ImpAcu;
};

struct sNodoDia {
        sInfoDia  Info;
        tListaDia Sgte;
};

template <class T>
T iif(bool exprLog, T ptr1, T ptr2) {
        if (exprLog)
                return ptr1;
        else
                return ptr2;
```

```
} // iif

/*
tListaDia iif(bool exprLog, tListaDia ptr1, tListaDia ptr2) {
	if (exprLog)
		return ptr1;
	else
		return ptr2;
} // iif
*/
template <class T>
T ExisteNodo(T &Lista,T &pAnt,byte mes) {
	T pAct;

	pAnt = NULL;
	pAct = Lista;
	while (pAct && mes > pAct->Info.Mes) {
		pAnt = pAct;
		pAct = pAct->Sgte;
	}
	return iif(pAct && mes == pAct->Info.Mes,pAct,NULL);
} // ExisteNodo
/*
tListaDia ExisteNodo(tListaDia &Lista,tListaDia &pAnt,byte dia) {
	tListaDia pAct;

	pAnt = NULL;
	pAct = Lista;
	while (pAct != NULL && dia > pAct->Info.Dia) {
		pAnt = pAct;
		pAct = pAct->Sgte;
	}
	return iif(pAct && dia == pAct->Info.Dia,pAct,NULL);
} // ExisteNodo
*/
void InsertaInicio(tListaMes &Lista, sInfoMes valor) {
	tListaMes pNodo;

	pNodo = new sNodoMes;
	pNodo->Info = valor;
	pNodo->Sgte = Lista;
	Lista = pNodo;
} // InsertaInicio
```

```
void InsertaInicio(tListaDia &Lista, sInfoDia valor) {
	tListaDia pNodo;

	pNodo = new sNodoDia;
	pNodo->Info = valor;
	pNodo->Sgte = Lista;
	Lista = pNodo;
} // InsertaInicio

void CrearNodo(tListaMes &Lista, tListaMes &pAnt, sInfoMes valor) {
	tListaMes pNodo;

	if (!pAnt)
		InsertaInicio(Lista,valor);
	else {
		pNodo = new sNodoMes;
		pNodo->Info = valor;
		pNodo->Sgte = pAnt->Sgte;
		pAnt->Sgte = pNodo;
	}
} // CrearNodo

void CrearNodo(tListaDia &Lista, tListaDia &pAnt, sInfoDia valor) {
	tListaDia pNodo;

	if (!pAnt)
		InsertaInicio(Lista,valor);
	else {
		pNodo = new sNodoDia;
		pNodo->Info = valor;
		pNodo->Sgte = pAnt->Sgte;
		pAnt->Sgte = pNodo;
	}
} // CrearNodo

void SacarPrimerNodo(tListaMes &Lista, sInfoMes &valor) {
	tListaMes pElim;

	pElim = Lista;
	valor = Lista->Info;
	Lista = Lista->Sgte;
	delete pElim;
} // SacarPrimerNodo
```

```
void SacarPrimerNodo(tListaDia &Lista, sInfoDia &valor) {
        tListaDia pElim;

        pElim = Lista;
        valor = Lista->Info;
        Lista = Lista->Sgte;
        delete pElim;
} // SacarPrimerNodo
```

Cola Circular

```
/*
 *****************
 *    Id.Programa: ColaCircular.cpp
 *    Autor......: Lic.Hugo Cuello
 *    Fecha......: sep-2014
 *    Comentario.: Cola Circular
 * ****************
                                        *--------------------*
            *--------*     *--------*    |  int        |tpPCL|        *---*
            |      | *-|---> |      | *-|--->|nroVuelo|Sgte   |<------|-*  |
            *--^-----*     *--------*    |  4 b.       |4 b.   |        *---*
                |                          *---------------|---*        cFin
                *-----------------------------------------*
*/
// ----------------- include ---------------
#include <time.h>
#include <conio.h>
#include <fstream>
using namespace std;
// ------------ Fin include ----------------

// ------------ typedef y struct -----------
typedef int tInfo;
typedef struct tNodo *tpPCL;
struct tNodo {
  tInfo Info;
  tpPCL Sgte;
};
// ------------ Fin typedef y struct -------

// ------------ prototipos -----------------
```

```
void Enqueue(tpPCL &, tInfo );
void Dequeue(tpPCL &, tInfo & );
// ------------ Fin prototipos -------------

// ------------ funciones ------------------
int main() {
  tpPCL colaFin = NULL;
  int valor;

  cout << 5 << endl;
  Enqueue(colaFin,5);
  cout << 3 << endl;
  Enqueue(colaFin,3);
  cout << 7 << endl;
  Enqueue(colaFin,7);
  cout << "------------------" << endl;
  while (colaFin) {
    Dequeue(colaFin,valor);
    cout << valor << endl;
  }
  getch();
} // main

void Enqueue(tpPCL &qFin, tInfo valor) {
  tpPCL pNodo;

  pNodo = new tNodo;
  pNodo->Info = valor;
  if (!qFin)
    pNodo->Sgte = pNodo; // primer nodo en la cola circular
  else {
    pNodo->Sgte = qFin->Sgte;
    qFin->Sgte = pNodo;
  }
  qFin = pNodo;
} //Enqueue

void Dequeue(tpPCL &qFin, tInfo &valor) {
  tpPCL pElim;

  pElim = qFin->Sgte;
  valor = qFin->Sgte->Info;
  if (qFin != qFin->Sgte)
    qFin->Sgte = qFin->Sgte->Sgte;
```

```
  else
    qFin = NULL;
  delete pElim;
} //Dequeue
// ------------- Fin funciones -------------
```

Lista doblemente enlazada

```
/**************
*        Id.Programa: G5Ej00cListaSgteAnt.Cpp
*        Autor......: Lic. Hugo Cuello
*        Fecha......: Jul-2005 / Jun-2014
*        Comentario.: Estructura dinamica de tipo Lista (=List) ordenada -Free-
*            Modulos:
*              InsertaNodoSgte, InsertaNodoAnt
*                "Modulo despachador"
*              InsertaInicioSgte, InsertaInicioAnt
*                "Si ListaFte es Vacia Inserta primero
*                  sino por el frente o inicio" o
*                "Si ListaFin es Vacia Inserta ultimo
*                  sino por el fondo o final
*              InsertaEnMedioSgte, InsertaEnMedioAnt
*                "Inserta despues del primero, no antes,
*                  incluye por final" o
*                "Inserta antes del ultimo, no despues,
*                  incluye por inicio"
*              SuprimeNodoSgte, SuprimeNodoAnt
*                "Elimina un nodo, en cualquier punto de la Lista,
*                  sino enc. no."
***************/

#include <conio.h>
#include <fstream>
using namespace std;

typedef int tInfo;
typedef struct sNodo *tLista;
struct sNodo {
  tLista Ant;
  tInfo  Info;
  tLista Sgte;
};
```

```
void InsertaInicioFte (tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNodo;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pNodo->Sgte = ListFte;
  pNodo->Ant = NULL;
  if (!ListFte)
    ListFin = pNodo;
  else
    ListFte->Ant = pNodo;
  ListFte = pNodo;
} //InsertaInicioFte

void InsertaInicioFin(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNodo;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pNodo->Ant = ListFin;
  pNodo->Sgte = NULL;
  if (!ListFin)
    ListFte = pNodo;
  else
    ListFin->Sgte = pNodo;
  ListFin = pNodo;
} //InsertaInicioFin

void InsertaEnMedioSgte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNodo,
         pAct;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pAct = ListFte;
  while (pAct->Sgte && valor > pAct->Sgte->Info)
          pAct = pAct->Sgte;
  pNodo->Sgte = pAct->Sgte;
  pAct->Sgte = pNodo;
  pNodo->Ant = pAct;
  if (!pNodo->Sgte)
    ListFin = pNodo;
  else
```

```
   pNodo->Sgte->Ant = pNodo;
} //InsertaEnMedioSgte

void InsertaEnMedioAnt(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pNodo,
         pAct;

  pNodo = new sNodo;
  pNodo->Info = valor;
  pAct = ListFin;
  while (pAct->Ant && valor < pAct->Ant->Info)
          pAct = pAct->Ant;
  pNodo->Ant = pAct->Ant;
  pAct->Ant = pNodo;
  pNodo->Sgte = pAct;
  if (!pNodo->Ant)
    ListFte = pNodo;
  else
    pNodo->Ant->Sgte = pNodo;
} //InsertaEnMedioAnt

void InsertaNodoFte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  if (!ListFte || valor < ListFte->Info)
          InsertaInicioFte(ListFte,ListFin, valor);
  else
          InsertaEnMedioSgte(ListFte,ListFin,valor);
} //InsertaNodoFte

void InsertaNodoFin(tLista &ListFte, tLista &ListFin, tInfo valor) {
  if (!ListFin || valor > ListFin->Info)
          InsertaInicioFin(ListFte,ListFin,valor);
  else
          InsertaEnMedioAnt(ListFte,ListFin,valor);
} //InsertaNodoFin

void SuprimeNodoSgte(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pAct,
        pAnt;

  pAct = ListFte;
  pAnt = NULL;
  while (pAct && valor > pAct->Info) {
    pAnt = pAct;
```

```
            pAct = pAct->Sgte;
  }
  if (pAct && valor == pAct->Info) {
            if (!pAnt)
      ListFte = pAct->Sgte;
    else
      pAnt->Sgte = pAct->Sgte;
    if (!pAct->Sgte)
      ListFin = pAnt;
    else
      pAct->Sgte->Ant = pAnt;
    if (!ListFte)
      ListFin = NULL;
    delete pAct;
  }
} //SuprimeNodoSgte

void SuprimeNodoAnt(tLista &ListFte, tLista &ListFin, tInfo valor) {
  tLista pAct,
         pAnt;

  pAct = ListFin;
  pAnt = NULL;
  while (pAct && valor < pAct->Info) {
    pAnt = pAct;
            pAct = pAct->Ant;
  }
  if (pAct && valor == pAct->Info) {
            if (!pAnt)
      ListFin = pAct->Ant;
    else
      pAnt->Ant = pAct->Ant;
    if (!pAct->Ant)
      ListFte = pAnt;
    else
      pAct->Ant->Sgte = pAnt;
    if (!ListFin)
      ListFte = NULL;
    delete pAct;
  }
} //SuprimeNodoAnt

void EmiteListaSgte(tLista List) {
  tLista pAct = List;
```

```
  while (pAct) {
    cout << pAct->Info << " ";
    pAct = pAct->Sgte;
  }
  cout << endl;
} // EmiteListaSgte

void EmiteListaAnt(tLista List) {
  tLista pAct = List;

  while (pAct) {
    cout << pAct->Info << " ";
    pAct = pAct->Ant;
  }
  cout << endl;
} // EmiteListaAnt

int main () {
  int    num;
  tLista ListaFte = NULL,
         ListaFin = NULL;
  clrscr();
  cout << "Ing. valor: ";
  cin >> num;
  while ( num > 0) {
    InsertaNodoFte(ListaFte,ListaFin,num);
    cout << "Ing. valor: ";
    cin >> num;
    InsertaNodoFin(ListaFte,ListaFin,num);
    cout << "* Ing. valor: ";
    cin >> num;
  }
  cout << "-----------x----------" << endl;

  while (ListaFte) {
      EmiteListaSgte(ListaFte);
            cout << "Ing. valor: ";
            cin >> num;
            SuprimeNodoSgte(ListaFte,ListaFin,num);
  }
  cout << "***********X***********" << endl;
  cout << "Ing. valor: ";
```

```
  cin >> num;
  while ( num > 0) {
    InsertaNodoFte(ListaFte,ListaFin,num);
    cout << "Ing. valor: ";
    cin >> num;
    InsertaNodoFin(ListaFte,ListaFin,num);
    cout << "+ Ing. valor: ";
    cin >> num;
  }
  cout << "-----------x----------" << endl;

  while (ListaFte) {
     EmiteListaAnt(ListaFin);
            cout << "Ing. valor: ";
            cin >> num;
            SuprimeNodoAnt(ListaFte,ListaFin,num);
  }

  return 0;
}
```

Codificación G5Ej Suma de Polinomios

```
/*
          Id.Programa: SumaPoli.cpp
          Autor......: Prof. Hugo Cuello
          Fecha......: nov-2013
          Comentario.: Genera en forma arbitraria y al azar terminos de
                       polinomios P y Q, ord. x expo descendente.
                  Se pide:
                       Realizar la suma de P + Q --> R
*/
#include <stdio.h>
#include <conio.h>
#include <fstream.h>
#include "dSumPoli.h"
#define LOW_VALUE -1

void ObTerPoli(tInfo &valor, char NomPoli) {
          clrscr();
          gotoxy(10,1);
          cout << "Datos de un termino del Polinomio " << NomPoli;
          gotoxy(15,5);
          cout << "expo: ";
```

```
        cin >> valor.expo;
        gotoxy(15,7);
        cout << "coef: ";
        cin >> valor.coef;
} // ObTerPoli

char Menu() {
        char opc;

        clrscr();
        gotoxy(10,1),
        cout << "Suma de Polinomios";
        gotoxy(15,5);
        cout << "1: Termino P";
        gotoxy(15,7);
        cout << "2: Termino Q";
        gotoxy(15,9);
        cout << "3: Salir";
        gotoxy(20,12);
        cout << "Su Opcion: ";
        do
                opc = getche();
        while (!(opc>='1' && opc<='3'));
        return opc;
} // Menu

void LecEspLst(tLista &Px, tInfo &valor) {
        if (Px != NULL)
                SacarPrimerNodo(Px,valor);
        else
                valor.expo = LOW_VALUE;
} // LecEspLst

void SumarPoli(tLista &P, tLista &Q, tLista &R) {
        tLista RFin;
        tInfo rInfoP,
              rInfoQ,
              rInfoR;

        RFin = NULL;
        LecEspLst(P,rInfoP);
        LecEspLst(Q,rInfoQ);
        while (rInfoP.expo != LOW_VALUE || rInfoQ.expo != LOW_VALUE)
```

```
                if (rInfoP.expo == rInfoQ.expo) {
                        rInfoR.expo = rInfoP.expo;
                        rInfoR.coef = rInfoP.coef + rInfoQ.coef;
                        AgregarNodo(R,RFin,rInfoR);
                        LecEspLst(P,rInfoP);
                        LecEspLst(Q,rInfoQ);
                }
                else
                        if (rInfoP.expo > rInfoQ.expo) {
                                AgregarNodo(R,RFin,rInfoP);
                                LecEspLst(P,rInfoP);
                        }
                        else {
                                AgregarNodo(R,RFin,rInfoQ);
                                LecEspLst(Q,rInfoQ);
                        }
} // SumarPoli

void EmitirPoli(tLista &Px, char NomPoli) {
        tLista PxAux;

        cout << endl << NomPoli << " = ";
        PxAux = Px;
        while (PxAux != NULL) {
                cout << PxAux->Info.coef << " ";
                if (PxAux->Info.expo > 0)
                        cout << "x^" << PxAux->Info.expo << " +";
                PxAux = PxAux->Sgte;
        }
        cout << '\b' << " Enter para continuar...";
        getch();
} // EmitirPoli

void LiberarNodos(tLista &Px) {
        tInfo valor;

        while (Px != NULL)
                SacarPrimerNodo(Px,valor);
} // LiberarNodos

main() {
        tLista P,Q,R;
        tInfo  valor;
        char   opc;
```

```
        P = Q = R = NULL;
        do {
                opc = Menu();
                switch (opc) {
                        case '1':
                                ObTerPoli(valor,'P');
                                InsertaNodo(P,valor);
                                break;
                        case '2':
                                ObTerPoli(valor,'Q');
                                InsertaNodo(Q,valor);
                                break;
                }
        } while (opc != '3');
        EmitirPoli(P,'P');
        EmitirPoli(Q,'Q');
        SumarPoli(P,Q,R);
        EmitirPoli(R,'R');
        LiberarNodos(R);
        Return 0;
} // main
```

Árbol Binario

ANEXOS - Árbol Binario.

```
/*
  Id.Programa: ArbolBinarioIdx.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: junio-2015 Fecha original en Pascal junio-2002.
  Comentario.: Genera, Emite (In-Orden, Pre-Orden, Pos-Orden), Busca
               y Elimina, de Estructura Dinamica Arbol Binario.
*/
#include <conio.h>
#include <iomanip.h>
#include <fstream>
using namespace std;
typedef unsigned short word;
typedef word tInfo;
typedef struct tNodo *tTree;
struct tNodo {
 tTree pIzq;
 tInfo Info;
 tTree pDer;
};
const
 MAX_SIZE_NODO = 10000;

void CrearArbol(tTree &Tree) {
 Tree = NULL;
} //CrearAbrol

bool ArbolVacio(tTree &Tree) {
 return Tree == NULL;
} //ArbolVacio

bool ArbolLleno(word MagnitudNodo) {
 return MagnitudNodo > MAX_SIZE_NODO;
} //ArbolLleno

void InsertarHoja(tTree &Tree, tTree pNvo) {
 if (ArbolVacio(Tree))
   Tree = pNvo;
 else
   if (pNvo->Info < Tree->Info)
     InsertarHoja(Tree->pIzq,pNvo);
```

```
    else
      InsertarHoja(Tree->pDer,pNvo);
} //InsertaNodoHoja

void ListarInOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    ListarInOrden(Tree->pIzq);
    cout << setw(3) << Tree->Info;
    ListarInOrden(Tree->pDer);
  }
} //ListarInOrden

void ListarPreOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    cout << setw(3) << Tree->Info;
    ListarPreOrden(Tree->pIzq);
    ListarPreOrden(Tree->pDer);
  }
} //ListarPreOrden

void ListarPosOrden(tTree &Tree) {
  if (!ArbolVacio(Tree)) {
    ListarPosOrden(Tree->pIzq);
    ListarPosOrden(Tree->pDer);
    cout << setw(3) << Tree->Info;
  }
} //ListarPosOrden

tTree BuscarNodo(tTree &Tree, tInfo valor) {
  if (ArbolVacio(Tree))
    return NULL;
  else
    if (valor == Tree->Info)
      return Tree;
    else
      if (valor < Tree->Info)
        return BuscarNodo(Tree->pIzq,valor);
      else
        return BuscarNodo(Tree->pDer,valor);
} //BuscarNodo

void EliminarNodo(tTree &Tree, tInfo valor) {
  tTree pAct,
        pAnt,
        pElim;

  if (!ArbolVacio(Tree))
    if (valor < Tree->Info)
```

```
      EliminarNodo(Tree->pIzq,valor);
    else
      if (valor > Tree->Info)
        EliminarNodo(Tree->pDer,valor);
      else {
        pElim = Tree;
        if (!pElim->pDer)
          Tree = pElim->pIzq;
        else
          if (!pElim->pIzq)
            Tree = pElim->pDer;
          else {
            pAct = pElim->pIzq;
            pAnt = pAct;
            while (pAct->pDer) {
              pAnt = pAct;
              pAct = pAct->pDer;
            }
            if (pAct == pAnt)
              pElim->pIzq = pAct->pIzq;
            pElim->Info = pAct->Info;
            pElim = pAct;
            pAnt->pDer = pAct->pIzq;
          }
        delete pElim;
        gotoxy(1,24); clreol();
        cout << "El elemento " << setw(3) << valor << " fue eliminado... ";
      }
  else {
    gotoxy(1,24); clreol();
    cout << "El elemento " << valor << " no existe... ";
  }
} //EliminarNodo

word IngElem(string mens) {
  word elem;

  gotoxy(20,8);
  cout << mens;
  gotoxy(10,25);
  cout << "Para finalizar ingresar un valor = 0...";
  gotoxy(10,9); clreol();
  cout << "Ing. valor: ";
  cin >> elem;
  return elem;
} //IngElem
```

```
void MenuOps(tTree pRoot) {
 tTree pNvo,
       pNodo;
  word elem;
  char opc;

 do {
   clrscr();
   cout << "1: Insertar" << endl;
   cout << "2: Quitar" << endl;
   cout << "3: Listar In-Orden" << endl;
   cout << "4: Listar Pre-Orden" << endl;
   cout << "5: Listar Pos-Orden" << endl;
   cout << "6: Buscar elemento" << endl;
   cout << "0: Finalizar" << endl;
   cout << "Su opcion: ";
   opc = getche();
   switch(opc) {
     case '1':
       elem = IngElem("Insertar");
       while (elem) {
         pNvo = new tNodo;
         pNvo->Info = elem;
         pNvo->pIzq = NULL;
         pNvo->pDer = NULL;
         InsertarHoja(pRoot,pNvo);
         elem = IngElem("Insertar");
       }
       break;
     case '2':
       elem = IngElem("Quitar");
       while (elem) {
         EliminarNodo(pRoot,elem);
         cout << "Oprima una tecla para continuar...";
         getch();
         gotoxy(1,24); clreol();
         elem = IngElem("Quitar");
       }
       break;
     case '3':
       gotoxy(20,8); clreol();
       cout << "Listar In-Orden...";
       gotoxy(20,12);
       ListarInOrden(pRoot);
       gotoxy(1,24); clreol();
       cout << "Oprima una tecla para continuar...";
       getch();
       break;
```

```
      case '4':
        gotoxy(20,8); clreol();
        cout << "Listar Pre-Orden...";
        gotoxy(20,12);
        ListarPreOrden(pRoot);
        gotoxy(1,24); clreol();
        cout << "Oprima una tecla para continuar...";
        getch();
        break;
      case '5':
        gotoxy(20,8); clreol();
        cout << "Listar Pos-Orden...";
        gotoxy(20,12);
        ListarPosOrden(pRoot);
        gotoxy(1,24); clreol();
        cout << "Oprima una tecla para continuar...";
        getch();
        break;
      case '6':
        elem = IngElem("Buscar");
        while (elem) {
          pNodo = BuscarNodo(pRoot,elem);
          gotoxy(1,24); clreol();
          if (pNodo)
            cout << "El elemento " << pNodo->Info << " se encontro...";
          else
            cout << "El elemento " << elem << " no existe... ";
          cout << "Oprima una tecla para continuar...";
          getch();
          gotoxy(1,24); clreol();
          elem = IngElem("Buscar");
        }
    }
  }
  while (opc != '0');
} //MenuOps

int main() {
  tTree pRaiz;

  CrearArbol(pRaiz);
  MenuOps(pRaiz);
}
```

Codificación Ventas Indexado con Árbol Binario Búsqueda (ABB)

```
/*
*  Id.Programa: VentasIndex.cpp
*  Autor......: Lic. Hugo Cuello
*  Fecha......: octubre-2014
*  Comentario.: Archivo de Ventas Indexado con Arbol Binario
****************
*/

#include <iomanip.h>
#include <fstream>
using namespace std;

struct tInfo {
  int codArt,
      refArt;
};

typedef struct tNodo *tTree;
struct tNodo {
  tTree pIzq;
  tInfo Info;
  tTree pDer;
};

typedef char str20[21];
struct tsVta {
  int codArt,
      cant;
  float preUni;
  str20 descrip;
};

bool ArbolVacio(tTree &Tree) {
  return Tree == NULL;
} //ArbolVacio

void InsertarNodoHoja(tTree &Tree, tTree pNvo) {
  if (ArbolVacio(Tree))
    Tree = pNvo;
  else
    if (pNvo->Info.codArt < Tree->Info.codArt)
      InsertarNodoHoja(Tree->pIzq,pNvo);
    else
      InsertarNodoHoja(Tree->pDer,pNvo);
} //InsertarNodoHoja
```

```
void ListarInOrden(FILE **Vtas, tTree &Tree) {
  tsVta rVta;

  if (!ArbolVacio(Tree)) {
    ListarInOrden(Vtas,Tree->pIzq);
    fseek(*Vtas,Tree->Info.refArt * sizeof(rVta),SEEK_SET);
    fread(&rVta,sizeof rVta,1,*Vtas);
    cout << setw(3) << rVta.codArt << " " << rVta.cant << " " << rVta.preUni << " " <<
rVta.descrip << endl;
    ListarInOrden(Vtas,Tree->pDer);
  }
} //ListarInOrden

void Abrir(FILE **Vtas) {
  *Vtas = fopen("Ventas.Dat","rb");
} //Abrir

main() {
  FILE *Ventas;
  tsVta rVenta;
  tInfo rInfo;
  tTree ArbolBin = NULL,
          pNodoNew;

  Abrir(&Ventas);
  while (fread(&rVenta,sizeof rVenta,1,Ventas)) {
    rInfo.codArt = rVenta.codArt;
    rInfo.refArt = ftell(Ventas) / sizeof (rVenta) - 1;
    pNodoNew = new tNodo;
    pNodoNew-> Info = rInfo;
    pNodoNew-> pIzq = pNodoNew->pDer = NULL;
    InsertarNodoHoja(ArbolBin,pNodoNew);
  }
  ListarInOrden(&Ventas,ArbolBin);
  fclose(Ventas);
  return 0;
} //main()
```

Templates

TEMPLATES - Templates o Plantillas

El lenguaje C++ ofrece la posibilidad de sobrecargar nombres en las funciones, esto se conoce como "**sobrecarga funcional**", al invocar a una función que contiene el mismo nombre de otra función, el sistema reconocerá a qué función cederle el control del proceso, mediante el análisis en su lista de parámetros. Esto se logra determinando en la llamada y comparándo sus parámetros con ambas funciones, y en la que haya concordancia es la función que tomará el control de ejecución. Las situaciones que pueden presentarse entonces dependerá de la cantidad de parámetros o el tipo de los parámetros o el orden de los mismos.

El inconveniente de este método es que debemos duplicar el código en ambas funciones aunque este código sea exactamente el mismo.

La ventaja es que no tenemos que dar nombres distintos a ambas funciones.

Otra característica es que el código de ambas funciones puede diferir entre ellas.

El lenguaje C++ además de permitir la sobrecarga funcional ofrece otra alternativa muy interesante y es el uso de **templates**, es decir, de plantillas.

La idea central de utilizar templates es la de no duplicar el código, por lo que solo se realizará una sola función, permitiendo diferentes tipos de datos para su proceso.

Por lo tanto, el código no podrá ser diferente para los distintos tipos de datos, si esto fuera necesario, entonces deberá realizarse otra función con otro código diferente.

En definitiva un template permite emplear una función con un mismo código para distintos tipos de datos.

A continuación se presentan diferentes ejemplos del uso de templates para las estructuras dinámicas de datos, pero esto no está limitado para estos casos, sino, que puede extenderse para otras estructuras de datos, a saber, archivos, arreglos.

Versiones de Pila, Cola y Lista con templates

Pila con template

```
/*************
*       Id.Programa: G5Anx01PILA-Tmpl.cpp
*       Autor      : Lic. Hugo Cuello
*       Fecha......: oct-2013
*       Comentario.: Plantillas Template.
*           Se definen 4 Template
*            1. de tipo int.
*            2. de tipo float.
*             3. de tipo string.
*             4. de tipo sArt.
```

```
*           Ventaja del uso de Templates:
*                 Permite declarar tipos genericos de datos
*           Aplicación: Se crean 4 Pilas;
*             1. de sNodo<int>*      Pila = NULL;
*             2. de sNodo<float>*   Pila2 = NULL;
*             3. de sNodo<string>* Pila3 = NULL;
*             4. de sNodo<sArt>*   Pila4 = NULL;
**************/

#include <fstream>
using namespace std;
#define sNodoT            sNodo<T>
#define psNodoT           sNodoT*
#define psNodoINT         sNodo<int>*
#define psNodoFLOAT       sNodo<float>*
#define psNodoSTRING sNodo<string>*
#define psNodoSArt        sNodo<sArt>*

template <typename T> // o  <class T> preferiblemente en lugar de typename si es P.O.O.
struct sNodo {
   T           info;
   psNodoT ant;
};

template <typename T>
 void Push(psNodoT &Pila, T valor) {
   psNodoT pNodo;

   pNodo = new sNodoT;
   pNodo->info = valor;
   pNodo->ant = Pila;
   Pila = pNodo;
} // Push - Meter

template <typename T>
 void Pop(psNodoT &Pila, T &valor) {
   psNodoT pElim;

   pElim = Pila;
   valor = Pila->info;
   Pila = Pila->ant;
   delete pElim;
} // Pop - Sacar
```

```
typedef char str20[21];
struct sArt {
   unsigned codArt,
                 cant;
   str20       descrip;
   float       preUni;
};

void PilaINT( ) {
  psNodoINT Pila  = NULL;
  int   a;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. int: ";
    cin >> a;
    Push<int>(Pila, a);  // o Push (Pila, a);
  }
  cout << "-----------x-----------" << endl;
  while (Pila) {
    Pop< >(Pila, a);
    cout << a << endl;
  }
  cout << "-----------x-----------" << endl;
} // PilaINT

void PilaFLOAT( ) {
  psNodoFLOAT  Pila = NULL;
  float  r;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. float: ";
    cin >> r;
    Push<float>(Pila, r);
  }
  cout << "----------x-----------" << endl;
  while (Pila) {
    Pop<float>(Pila, r);
    cout << r << endl;
  }
  cout << endl << "-------+-------" << endl << endl;
} // PilaFLOAT
```

```
void PilaSTRING( ) {
  psNodoSTRING Pila = NULL;
  string s;
  char   cad[81];

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. cadena: ";
    gets(cad);
    s = cad;
    Push<string>(Pila, s);
  }
  cout << "----------x-----------" << endl;
  while (Pila) {
    Pop<string>(Pila,s);
    cout << s << endl;
  }
  cout << "----------x-----------" << endl;
} //PilaSTRING

void PilaSTRUCT( ) {
  psNodoSArt Pila = NULL;
  sArt            rArt;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. Cod.Art: ";
    cin >> rArt.codArt;
    cout << endl << "     Cant....: ";
    cin >> rArt.cant;
    cout << endl << "     Descrip.: ";
    gets(rArt.descrip);
    cin.clear();
    cout << endl << "     Pre.Uni.: ";
    cin >> rArt.preUni;
    Push<sArt>(Pila,rArt);
  }
  cout << "-----------x-----------" << endl;
  while (Pila) {
    Pop<sArt>(Pila,rArt);
    cout << rArt.codArt << " "
         << rArt.cant << " "
         << rArt.descrip << " "
         << rArt.preUni << endl;
  }
} // PilaSTRUCT
```

```
main() {
  PilaINT( );
  PilaFLOAT( );
  PilaSTRING( );
  PilaSTRUCT( );
  return 0;
}
```

Cola con template

```
/*************
*        Id.Programa: G5Anx02COLA-Tmpl.cpp
*        Autor      : Lic. Hugo Cuello
*        Fecha......: octubre-2013
*        Comentario.: Plantillas Template.
*            Se definen 4 Template:
*             1. de tipo int.
*             2. de tipo float.
*              3. de tipo string.
*              4. de tipo sArt.
*            Ventaja del uso de Templates:
*                  Permite declarar tipos genericos de datos
*            Aplicacion: Se crean 4 Colas;
*              1. de sNodo<int>*    ColaFte1 = NULL, *ColaFin1 = NULL;
*              2. de sNodo<float>*  ColaFte2 = NULL, *ColaFin2 = NULL;
*              3. de sNodo<string>* ColaFte3 = NULL, *ColaFin3 = NULL;
*              4. de sNodo<sArt>*   ColaFte4 = NULL, *ColaFin4 = NULL;
**************/
#include <fstream>
using namespace std;
#define sNodoT              sNodo<T>
#define psNodoT             sNodoT*
#define psNodoINT           sNodo<int>*
#define psNodoFLOAT         sNodo<float>*
#define psNodoSTRING        sNodo<string>*
#define psNodoSTRUCT        sNodo<sArt>*

template <typename T> o  <class T> preferiblemente en lugar de typename si es P.O.O.
struct sNodo {
   T            info;
   psNodoT sgte;
  };
```

```
template <typename T>
void Enqueue(psNodoT &cFte, psNodoT &cFin, T valor) {
  psNodoT pNodo;

  pNodo = new sNodoT;
  pNodo->info = valor;
  pNodo->sgte = NULL;
  if (cFte = = NULL)
     cFte = pNodo;
  else
     cFin->sgte = pNodo;
  cFin = pNodo;
} // Enqueue - Agregar

template <typename T>
void Dequeue(psNodoT &cFte, psNodoT &cFin, T &valor) {
  psNodoT pElim;

  pElim = cFte;
  valor = cFte->info;
  cFte = cFte->sgte;
  if (cFte == NULL)
     cFin = NULL;
  delete pElim;
} // Dequeue - Suprimir

typedef char str20[21];
struct sArt {
   unsigned codArt,
                cant;
   str20        descrip;
   float        preUni;
};

void ColaINT( ) {
  psNodoINT ColaFte  = NULL;
  psNodoINT ColaFin  = NULL;
  int   a;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. int: ";
    cin >> a;
    Enqueue<int>(ColaFte, ColaFin, a);
  }
```

```
  cout << "-----------x-----------" << endl;
  while (ColaFte) {
    Dequeue<int>(ColaFte, ColaFin, a);
    cout << a << endl;
  }
  cout << "-----------x-----------" << endl;
} // ColaINT

void ColaFLOAT( ) {
  psNodoFLOAT ColaFte = NULL;
  psNodoFLOAT ColaFin = NULL;
  float  r;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. float: ";
    cin >> r;
    Enqueue<float>(ColaFte, ColaFin, r);
  }
  cout << "----------x-----------" << endl;
  while (ColaFte) {
    Dequeue<float>(ColaFte, ColaFin, r);
    cout << r << endl;
  }
  cout << endl << "-------+-------" << endl << endl;
} // ColaFLOAT

void ColaSTRING( ) {
  psNodoSTRING ColaFte = NULL;
  psNodoSTRING ColaFin = NULL;
  string s;
  char   cad[81];

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. cadena: ";
    gets(cad);
    s = cad;
    Enqueue<string>(ColaFte, ColaFin, s);
  }
  cout << "----------x-----------" << endl;
  while (ColaFte) {
    Dequeue<string>(ColaFte, ColaFin, s);
    cout << s << endl;
  }
```

```
  cout << "----------x-----------" << endl;
} // ColaSTRING

void ColaSTRUCT( ) {
  psNodoSTRUCT ColaFte = NULL;
  psNodoSTRUCT ColaFin = NULL;
  sArt   rArt;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. Cod.Art: ";
    cin >> rArt.codArt;
    cout << endl << "     Cant....: ";
    cin >> rArt.cant;
    cout << endl << "     Descrip.: ";
    gets(rArt.descrip);
    cin.clear();
    cout << endl << "     Pre.Uni.: ";
    cin >> rArt.preUni;
    Enqueue<sArt>(ColaFte, ColaFin, rArt);
  }
  cout << "-----------x-----------" << endl;
  while (ColaFte) {
    Dequeue<sArt>(ColaFte, ColaFin, rArt);
    cout << rArt.codArt << " "
         << rArt.cant << " "
         << rArt.descrip << " "
         << rArt.preUni << endl;
  }
} // ColaSTRUCT

main ( ) {
  ColaINT();
  ColaFLOAT();
  ColaSTRING();
  ColaSTRUCT();
  return 0;
}
```

Lista con template

```
/*************
```

```
*        Id.Programa: G5Anx03LISTA-Tmpl.cpp
*        Autor      : Lic. Hugo Cuello
*        Fecha......: octubre-2013
*        Comentario.: Plantillas Template.
*           Se definen 4 Template:
*             1. de tipo int.
*             2. de tipo float.
*             3. de tipo string.
*             4. de tipo sArt.
*           Ventaja del uso de Templates:
*                                          Permite declarar tipos genericos de datos
*           Aplicacion: Se crean 4 Listas;
*             1. de sNodo<int>*    Lista = NULL;
*             2. de sNodo<float>*  Lista2 = NULL;
*             3. de sNodo<string>* Lista3 = NULL;
*             4. de sNodo<sArt>*   Lista4 = NULL;
**************/

#include <fstream>
using namespace std;
#define sNodoT          sNodo<T>
#define psNodoT         sNodoT*
#define psNodoINT       sNodo<int>*
#define psNodoFLOAT     sNodo<float>*
#define psNodoSTRING    sNodo<string>*
#define psNodoSTRUCT sNodo<sArt>*

template <typename T> o  <class T> preferiblemente en lugar de typename si es P.O.O.
struct sNodo {
   T          info;
   psNodoT sgte;
};

template <typename T>
  void InsertaInicio(psNodoT &Lista, T valor) {
   psNodoT pNodo;

   pNodo = new sNodoT;
   pNodo->info = valor;
   pNodo->sgte = Lista;
   Lista = pNodo;
} // InsertaInicio
```

```
template <typename T>
void InsertaEnMedio(psNodoT &Lista, T valor) {
  psNodoT pAct;
  psNodoT pNodo;

  pAct = Lista;
  while (pAct->sgte  && valor > pAct->sgte->info)
      pAct = pAct->sgte;
  pNodo = new sNodoT;
  pNodo->info = valor;
  pNodo->sgte = pAct->sgte;
  pAct->sgte = pNodo;
} // InsertaEnMedio

template <typename T>
void InsertaNodo(psNodoT &Lista, T valor) {
  if (!Lista || valor < Lista->Info)
      InsertaInicio(Lista, valor);
  else
       InsertaEnMedio(Lista, valor);
} // InsertaNodo

//-------------------------------------

template <typename T>
void InsertaEnMedioClv(psNodoT &Lista, T valor) {
  psNodoT pAct;
  psNodoT pNodo;

  pAct = Lista;
  while (pAct->sgte  && valor.codArt > pAct->sgte->info.codArt)
      pAct = pAct->sgte;
  pNodo = new sNodoT;
  pNodo->info = valor;
  pNodo->sgte = pAct->sgte;
  pAct->sgte = pNodo;
} // InsertaEnMedio

template <typename T>
void InsertaNodoClv(PsNodoT &Lista, T valor) {
  if (!Lista || valor.codArt  <  Lista->Info.codArt)
      InsertaInicio(Lista,valor);
  else
      InsertaEnMedioClv(Lista,valor);
```

```
} // InsertaNodo


//-----------------------------------

template <typename T>
void SacarPrimerNodo(psNodoT &Lista, T &valor) {
   psNodoT pElim;

   pElim = Lista;
   valor = Lista->info;
   Lista = Lista->sgte;
   delete pElim;
} // SacarPrimerNodo

typedef char str20[21];
struct sArt {
   unsigned codArt,
                cant;
   str20       descrip;
   float       preUni;
};

void ListaINT() {
  psNodoINT  Lista  = NULL;
  int    a;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. int: ";
    cin >> a;
    InsertaNodo<int>(Lista, a);
  }
  cout << "-----------x-----------" << endl;
  while (Lista) {
    SacarPrimerNodo<int>(Lista, a);
    cout << a << endl;
  }
  cout << "-----------x-----------" << endl;
} // ListaINT

void ListaFLOAT( ) {
  psNodoFLOAT Lista = NULL;
  float    r;
```

```
  for (register i = 1; i <= 3; i++) {
    cout << "Ing. float: ";
    cin >> r;
    InsertaNodo<float>(Lista, r);
  }
  cout << "----------x-----------" << endl;
  while (Lista) {
    SacarPrimerNodo<float>(Lista, r);
    cout << r << endl;
  }
  cout << endl << "-------+-------" << endl << endl;
} // ListaFLOAT

void ListaSTRING( ) {
  psNodoSTRING Lista3 = NULL;
  string s;
  char   cad[81];

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. cadena: ";
    gets(cad);
    s = cad;
    InsertaNodo<string>(Lista3, s);
  }
  cout << "----------x-----------" << endl;
  while (Lista3) {
    SacarPrimerNodo<string>(Lista3, s);
    cout << s << endl;
  }
  cout << "----------x-----------" << endl;
} // ListaSTRING

void ListaSTRUCT() {
  psNodoSTRUCT Lista4 = NULL;
  sArt   rArt;

  for (register i = 1; i <= 3; i++) {
    cout << "Ing. Cod.Art: ";
    cin >> rArt.codArt;
    cout << endl << "     Cant....: ";
    cin >> rArt.cant;
    cout << endl << "     Descrip.: ";
    gets(rArt.descrip);
```

```
    cin.clear();
    cout << endl << "     Pre.Uni.: ";
    cin >> rArt.preUni;
    InsertaNodoClv<sArt>(Lista4, rArt); // Ord.x cmp.reg. +adelante se usará *func.
  }
  cout << "-----------x-----------" << endl;
  while (Lista4) {
    SacarPrimerNodo<sArt>(Lista4, rArt);
    cout << rArt.codArt << " "
         << rArt.cant << " "
         << rArt.descrip << " "
         << rArt.preUni << endl;
  }
} // ListaSTRUCT

main( ) {
  ListaINT( );
  ListaFLOAT( );
  ListaSTRING( );
  ListaSTRUCT( );
  return 0;
}
```

Punteros a funciones

Una función es código y al igual que las variables estas están ubicadas en direcciones de la memoria interna RAM. De la misma manera que un objeto variable podemos obtener su dirección en la memoria interna, también podemos obtener la dirección de inicio de una función, para llevar a cabo esta acción necesitaremos una variable puntero al que le vamos asignar la dirección de memoria interna de una función; por lo que estaremos en condición de afirmar que un puntero a una función es técnicamente una variable que almacena la dirección de comienzo de la función. El puntero que recibe la dirección de memoria de una función es una especie de alias a esa función. Además podremos pasar como argumento una función por medio de un puntero a esa función. El uso de los punteros a funciones deben tener una sintaxis específicas, deben encerrarse entre paréntesis, se podrán indicar los tipos de cada parámetro y el valor de retorno. El operador de llamada a función ( ) tiene una prioridad más alta que el operador de anulación de referencia *. Sin ellos, el compilador interpreta la sentencia como una función que devuelve un puntero a un tipo de retorno especificado.

Ejemplo de uso de puntero a función

```
#include <fstream.h>
using namespace std;
void imprime() {
  cout << "Emitiendo un título" << endl;
}

main() {
  void (*ptr_funct) (void) = imprime; //Observar que no utiliza & para asignar la dirección.
  ptr_funct(); //Invocación a la función imprime.
  return 0;
}
```

Ejemplo de Operaciones aritméticas con punteros a funciones

```
/*
  Id.Programa: PunteroFunciónOperArtiméticas.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: noviembre-2023
  Comentario.: Operaciones aritméticas
               Todas las funciones tienen el mismo formato float (float, float)
               Se establecen 4 punteros a función por medio de un vector.
               A cada componente se le asigna el nombre de una función
                 sumar, restar, multiplicar, dividir.
*/

#include <fstream>
using namespace std;

float sumar(float x, float y) {
  return x + y;
} // sumar

float restar(float x, float y) {
  return x - y;
} // restar

float multiplicar(float x, float y) {
  return x * y;
} // multiplicar
```

```
float dividir(float x, float y) {
  return x / y;
} // dividir

main() {
  float a = 12,
        b = 2;

  float (*pfuncOpArit[4]) (float, float) = {sumar, restar, multiplicar, dividir};
  char operador[4] = {'+','-','*','/'};

  cout << "Operaciones aritméticas" << endl;
  for (short i = 0; i < 4; i++)
    cout << a << ' '
         << operador[i] << ' '
         << b << ' '
         << '=' << ' '
         << pfuncOpArit[i](a,b) << endl;
  return 0;
}
```

Ejemplo de punteros a función como parámetros de otra función

```
/*
  Id.Programa: PunteroFunciónOperArtiméticas2.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: noviembre-2023
  Comentario.: Operaciones aritméticas
               Todas las funciones tienen
             el mismo formato float (float, float)
               Se establece una función que recibe
             dos operandos y el puntero a la función
             de la operación aritmética a calcular.
*/

#include <fstream>
using namespace std;



float sumar(float x, float y) {
  return x + y;
} // sumar
```

```
float restar(float x, float y) {
  return x - y;
} // restar

float multiplicar(float x, float y) {
  return x * y;
} // multiplicar

float dividir(float x, float y) {
  return x / y;
} // dividir

float fOperArit (float x, float y, float (*fCalcular) (float, float)) {
  return fCalcular (x, y);
} // fOperArit

main() {
  float a = 12,
        b = 2;
  cout << "Operaciones Aritméticas" << endl;
  cout << a << " + " << b << " = " << fOperArit(a, b, sumar) << endl;
  cout << a << " - " << b << " = " << fOperArit(a, b, restar) << endl;
  cout << a << " * " << b << " = " << fOperArit(a, b, multiplicar) << endl;
  cout << a << " / " << b << " = " << fOperArit(a, b, dividir) << endl;
  return 0;
}
```

Aplicaciones de puntero a función para estructuras dinámicas de datos

Funciones para criterios

En aquellos casos en que una función requiera realizar alguna comparación, en las cuales puedan intervenir los distintos operadores de (>, <, =, <=, >=, <>), podremos pasarle un puntero a una función como parámetro, para llevar a cabo la comparación que requiera. Cada una de estas funciones deberán ser realizadas. Por otro lado, se aplicarán templates a las funciones para generalizarlas y que acepten distintos tipos de datos para su manipulación.
Para aplicar las funciones criterios tendremos tres instancias, en donde, en cada una de ellas deberemos respetar los formatos para el uso de ellas.

Instancias

Desde la Invocación

Agregamos un argumento cuya sintaxis es:
f (arg1,arg2 , (bool (*) (void *, void *)) (**funcCriterio**));
en donde:
f representa la función que se invoca.

arg1, arg2 son los argumentos que debemos pasar a la función invocada, la cantidad de argumentos estará indicada por la propia función a invocar.

El tercer argumento indicada de color rojo es el formato que debemos emplear tal cual se lo indica, salvo, el último par de paréntesis representado por funcCriterio, en donde debemos indicarle el nombre de la función que se aplicará, dentro de la función f.

Desde la Cabecera

tipo f (tipo &par1, tipo par2 , bool (***criterio**) (void *, void *)) {

en donde:

tipo es el tipo de la función representada por f.

f representa la función invocada.

par1, par2 son los parámetros formales que hemos pasado a la función invocada, la cantidad de parámetros estará indicada por la propia función invocada.

El tercer argumento indicada en color rojo es el formato que debemos emplear tal cual se lo indica.

*criterio representa el parámetro puntero a la función recibida por funcCriterio y que se aplicará en el cuerpo de la función f, como criterio de comparación entre dos elementos. Este nombre es arbitrario, lo que significa que se le podrá dar un nombre representativo del criterio a emplear.

Desde el cuerpo

if (cond1 && criterio(&var1, &var2))

en donde:

criterio es el puntero a la función enviada desde la invocación, represenada por funcCriterio.

var1 representa el primer elemento a comparar

var2 representa el segundo elemento a comparar.

Los nombres var1 y var2 son arbitrarios, y se cambiarán por los nombres de las variables a comparar.

A continuación se presentará el código de un proceso a realizar con las diferentes estructuras dinámicas de datos lineales (Pila, Cola, Lista), utilizando la herramienta **template** y funciones *criterios*.

Templates y Criterios con Pilas, Colas y Listas

```
/*
  Id.Programa: Template a EDD.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: octubre-2023
  Comentario.: Template y funciones Criterios aplicadas
               a Estructuras Dinamicas de Datos Lineales.
*/

#include <iomanip>
#include <fstream>
using namespace std;
```

```
const MAX_REGS = 12;

typedef char str20[21];

struct sArt {
  short codArt;
  str20 descrip;
  float preUni;
};

typedef sArt  tInfo;
typedef short tInfo2;

#define sNodoT        sNodo<T>
#define psNodoT       sNodo<T>*
#define psNodoInfo    sNodo<tInfo>*
#define psNodoInfo2 sNodo<tInfo2>*

template <typename T>
struct sNodo {
  T      info;
  psNodoT proxNodo;  // pNodo
};

// ----< Funciones para Criterios >----

bool fCodArtHigh(tInfo &elem1, tInfo &elem2) {
  return elem1.codArt > elem2.codArt;
}

bool fCodArtEqual(tInfo &elem1, tInfo &elem2) {
  return elem1.codArt == elem2.codArt;
}

bool fCodArtLess(tInfo &elem1, tInfo &elem2) {
  return elem1.codArt < elem2.codArt;
}

bool fDescripHigh(tInfo &elem1, tInfo &elem2) {
  return strcmp(elem1.descrip,elem2.descrip) > 0;
}

bool fDescripLess(tInfo &elem1, tInfo &elem2) {
  return strcmp(elem1.descrip,elem2.descrip) < 0;
}

bool fDescripNotEq(tInfo &elem1, tInfo &elem2) {
  return strcmp(elem1.descrip,elem2.descrip) != 0;
```

```
}

bool fDescripEqual(tInfo &elem1, tInfo &elem2) {
  return strcmp(elem1.descrip,elem2.descrip) == 0;
}

bool fPreUniHigh(tInfo &elem1, tInfo &elem2) {
  return elem1.preUni > elem2.preUni;
}

bool fPreUniEqual(tInfo &elem1, tInfo &elem2) {
  return elem1.preUni == elem2.preUni;
}

bool fPreUniLess(tInfo &elem1, tInfo &elem2) {
  return elem1.preUni < elem2.preUni;
}

//----< Modulos para Pila >----

template <typename T>
void Push (psNodoT &Stk, T valor) {
  psNodoT pNvo = new sNodoT;

  pNvo->info = valor;
  pNvo->proxNodo = Stk;
  Stk = pNvo;
} // Push

template <typename T>
void Pop (psNodoT &Stk, T &valor) {
  psNodoT pElim = Stk;

  valor = Stk->info;
  Stk = Stk->proxNodo;
  delete pElim;
} // Pop

//----< Modulos para Cola >----

template <typename T>
void Enqueue (psNodoT &cFte, psNodoT &cFin, T valor) {
  psNodoT pNvo = new sNodoT;

  pNvo->info = valor;
  pNvo->proxNodo = NULL;
  if (!cFte)
```

```
    cFte = pNvo;
  else
    cFin->proxNodo = pNvo;
  cFin = pNvo;
} // Enqueue

template <typename T>
void Dequeue (psNodoT &cFte, psNodoT &cFin, T &valor) {
  psNodoT pElim = cFte;

  valor = cFte->info;
  cFte = cFte->proxNodo;
  if (!cFte)
    cFin = NULL;
  delete pElim;
} // Dequeue

//----< Modulos para Lista >----

template <typename T>
void InsertaInicio (psNodoT &List, T valor) {
  psNodoT pNvo = new sNodoT;

  pNvo->info = valor;
  pNvo->proxNodo = List;
  List = pNvo;
} // InsertaInicio

template <typename T>
void InsertaEnMedio (psNodoT &List, T &valor,
                                bool (* criterio)  (void *, void *) ) {
  psNodoT pNvo,
            *pAct = List;

  while (pAct->proxNodo && criterio(&valor,&pAct->proxNodo->info) )
    pAct = pAct->proxNodo;
  pNvo = new sNodoT;
  pNvo->info = valor;
  pNvo->proxNodo = pAct->proxNodo;
  pAct->proxNodo = pNvo;
} // InsertaEnMedio



template <typename T>
void InsertaNodo (psNodoT &List, T valor,
              bool (* criterio)  (void *, void *), void * fCrit ) {
  if (!List || criterio(&valor,&List->info) )
```

```
    InsertaInicio (List, valor);
  else
    InsertaEnMedio (List, valor, (bool (*) (void *, void * ) ) (fCrit) );
} // InsertaNodo

template <typename T>
void SacarPrimerNodo(psNodoT &List, T &valor) {
  psNodoT pElim = List;

  valor = List->info;
  List = List->proxNodo;
  delete pElim;
} // SacarPrimerNodo

template <typename T>
void SuprimeNodo (psNodoT &List, T &valor,
                                bool (* criterio)  (void *, void *),
                                bool (* criterio2) (void *, void *) ) {
  psNodoT pAct = List;
  psNodoT pAnt = NULL;

  while (pAct && criterio(&valor, &pAct->info) ) {
    pAnt = pAct;
    pAct = pAct->proxNodo;
  }
  if (pAct && criterio2(&valor, &pAct->info) ) {
    if (!pAnt)
      List = List->proxNodo;
    else
      pAnt->proxNodo = pAct->proxNodo;
    delete pAct;
  }
} // SuprimeNodo

template <typename T>
void EmiteLst (psNodoT &List) {
  psNodoT pAct = List;

  cout << "  Cod.Art Descripcion            Pre.Uni." << endl;
  while (pAct) {
    cout << "--> " << setw(5) << pAct->info.codArt << ' '
          << left << setw(20) << pAct->info.descrip << ' '
           << right << setw(10) << pAct->info.preUni << endl;
    pAct = pAct->proxNodo;
  }
  cout << endl;
} // EmiteLst
```

```
template <typename T>
void EnlazaInicio (psNodoT &Lista, psNodoT &pNodo) {
   pNodo->proxNodo = Lista;
   Lista = pNodo;
} // EnlazaInicio

template <typename T>
void EnlazaEnMedio (psNodoT &Lista, psNodoT &pNodo,
                                    bool (* criterio) (void *, void *) ) {
  psNodoT pAct = Lista;
  while (pAct->proxNodo && criterio(&pNodo->info,&pAct->proxNodo->info) )
    pAct = pAct->proxNodo;
  pNodo->proxNodo = pAct->proxNodo;
  pAct->proxNodo = pNodo;
}  // EnlazaEnMedio

template <typename T>
void ReInsertaNodo (psNodoT &ListaAux, psNodoT &pNodo,
                bool (* criterio) (void *, void *) , void * fCritHigh ) {

  if (!ListaAux || criterio(&pNodo->info,&ListaAux->info) )
    EnlazaInicio (ListaAux,pNodo);
  else
    EnlazaEnMedio (ListaAux,pNodo,
                            (bool (*) (void *, void * ) ) (fCritHigh) );
} // ReInsertaNodo

template <typename T>
void ReOrdenarLst (psNodoT &Lista, void * fCritLess, void * fCritHigh ) {
  psNodoT ListaAux = NULL;
  psNodoT pNodo;

  while (Lista) {
    pNodo = Lista;
    Lista = Lista->proxNodo;
    ReInsertaNodo (ListaAux,pNodo,
                            (bool (*) (void *, void * )) (fCritLess), fCritHigh );
  }
  Lista = ListaAux;
} // ReOrdenarLst

//----< Operaciones con Pila >----

template <typename T>
void OperarPila (psNodoT &Pila)  {
  tInfo2 x2;
```

```
  for (short i = 1; i <=3; i++) {
    cout << "Ing. valor: ";
    cin >> x2;
    Push (Pila,x2);
  }
  while (Pila) {
    Pop (Pila,x2);
    cout << x2 << ' ';
  }
  cout << endl;
} // OperarPila

//----< Operaciones con Cola >----

template <typename T>
void OperarCola (psNodoT &ColaFte, psNodoT &ColaFin) {
  tInfo2 x2;

  for (short i = 1; i <=3; i++) {
    cout << "Ing. valor: ";
    cin >> x2;
    Enqueue (ColaFte,ColaFin,x2);
  }
  while (ColaFte) {
    Dequeue (ColaFte,ColaFin,x2);
    cout << x2 << ' ';
  }
  cout << endl;
} // OperarCola

//----< Operaciones con Lista >----

template <typename T>
void OperarLista(psNodoT &Lista) {
  tInfo x;

  for (short i = 1; i <= MAX_REGS; i++) {
    cout << "Ing. CodArt.: ";
    cin >> x.codArt;
    cout << "Ing. Descrip: ";
    cin >> x.descrip;
    cout << "Ing. PreUni.: ";
    cin >> x.preUni;
    InsertaNodo (Lista,x, (bool (*) (void *, void * ) ) (fPreUniLess), &fPreUniHigh );
  }
  while (Lista) {
    SacarPrimerNodo (Lista,x);
```

```
  cout << setw( 5) << x.codArt << ' '
        << setw(20) << x.descrip << ' '
        << setw(10) << x.preUni << endl;
}
cout << endl;

// Genera nuevamente Lista para reOrdenar segun criterio
for (short i = 1; i <= MAX_REGS; i++) {
  cout << "*Ing. CodArt: ";
  cin >> x.codArt;
  cout << "Ing. Descrip: ";
  cin >> x.descrip;
  cout << "Ing. PreUni.: ";
  cin >> x.preUni;
  InsertaNodo (Lista,x, (bool (*) (void *, void * ) ) (fCodArtLess), &fCodArtHigh);
}
EmiteLst (Lista);
ReOrdenarLst (Lista,&fDescripLess, &fDescripHigh);
while (Lista) {
  EmiteLst (Lista);
  cout << "+Ing. Descrip: ";
  cin >> x.descrip;
  SuprimeNodo (Lista,x, (bool (*) (void *, void * ) ) (fDescripHigh),
                         (bool (*) (void *, void * ) ) (fDescripEqual));
}

for (short i = 1; i <= MAX_REGS; i++) {
  cout << "*Ing. CodArt: ";
  cin >> x.codArt;
  cout << "Ing. Descrip: ";
  cin >> x.descrip;
  cout << "Ing. PreUni.: ";
  cin >> x.preUni;
  InsertaNodo (Lista,x, (bool (*) (void *, void * ) ) (fPreUniLess), &fPreUniHigh );
}
EmiteLst (Lista);
ReOrdenarLs (Lista, &fCodArtLess, &fCodArtHigh);
while (Lista) {
  EmiteLst (Lista);
  cout << "+Ing. CodArt: ";
  cin >> x.codArt;
  SuprimeNodo (Lista,x, (bool (*) (void *, void * ) ) (fCodArtHigh),
                         (bool (*) (void *, void * ) ) (fCodArtEqual));
}

for (short i = 1; i <= MAX_REGS; i++) {
  cout << "*Ing. CodArt: ";
  cin >> x.codArt;
```

```
    cout << "Ing. Descrip: ";
    cin >> x.descrip;
    cout << "Ing. PreUni.: ";
    cin >> x.preUni;
    InsertaNodo (Lista,x, (bool (*) (void *, void * )) (fCodArtLess), &fCodArtHigh);
  }
  EmiteLst (Lista);
  ReOrdenarLst (Lista, &fPreUniLess, &fPreUniHigh );
  while (Lista) {
    EmiteLst (Lista);
    cout << "+Ing. PreUni: ";
    cin >> x.preUni;
    SuprimeNodo (Lista,x, (bool (*) (void *, void * ) ) (fPreUniHigh),
                                  (bool (*) (void *, void * ) ) (fPreUniEqual));
  }

  for (short i = 1; i <= MAX_REGS; i++) {
    cout << "*Ing. CodArt: ";
    cin >> x.codArt;
    cout << "Ing. Descrip: ";
    cin >> x.descrip;
    cout << "Ing. PreUni.: ";
    cin >> x.preUni;
    InsertaNodo (Lista,x, (bool (*) (void *, void * ) ) (fCodArtLess), &fCodArtHigh);
  }
  while (Lista) {
    EmiteLst (Lista);
    cout << "+Ing. Descripcion: ";
    cin >> x.descrip;
    SuprimeNodo (Lista,x, (bool (*) (void *, void * ))  (fDescripNotEq),
                                  (bool (*) (void *, void * ) ) (fDescripEqual));
  }
} // OperarLista

main() {
  psNodoInfo2 Pila      = NULL;
  psNodoInfo2 ColaFte = NULL;
  psNodoInfo2 ColaFin = NULL;
  psNodoInfo  Lista      = NULL;

  //---------< PILA >----------------
  OperarPila(Pila);
  //---------< COLA >----------------
  OperarCola(ColaFte,ColaFin);
  //---------< LISTA >---------------
  OperarLista(Lista);
  return 0;
```

```
}
```

"CADENAS DE CARACTERES I - estilo clásico de C-"

TABLAS – CADENAS ESTILO C - Tabla Cadenas de Caracteres estilo C

#Ord.	Función	Significado	Ejemplos	ANSI
cad1 y **cad2** son de tipo char *; **cad3** y **cad4** son de tipo const char *; **n** es de tipo size_t; y **car** es de tipo int convertido a char. **pcad** es char *.				
1	char ***strcat**(cad1,cad4)			**SI**
	Concatena la cad4 al final de cad1. El primer carácter de cad4 se superpone sobre el carácter '\0'de cad1 y agrega el carácter '\0' después del último carácter de cad4. El espacio físico de cad1 debe soportar la nueva longitud. **Retorna** cad1.		**strcpy**(cad1,"Super"); cout << **strcat**(cad1,"man"); **Emite**: Superman.	
2	char ***strchr**(cad3,car)			SI
	Retorna un puntero a la primer aparición de car en cad3 o NULL si no está.		cout << boolalpha << (**strchr**("abcdabcdabcdefg",'d')!= NULL); **Emite**: true Cout << **strchr**("abcdabcdabcdefg", 'd'); **Emite**: dabcdabcdefg.	
3	int **strcmp**(cad3,cad4) / int **strcoll**(cad1,cad2)			SI
	Compara la cad3 con cad4. **Retorna** < 0 si cad3 < cad4, 0 si cad3 = cad4 o > 0 si cad3 > cad4. cad3[i] cmp cad4[i]		cout << boolalpha << **strcmp** ("abcd","abxdef") != 0; **Emite**: true La diferencia del código ASCII entre caracteres correspondientes cad3[i] – cad4[i].	
4	char ***strcpy**(cad1,cad4)			SI
	Copia la cad4 en cad1, incluye '\0'. **Retorna** cad1.		cout << **strcpy**(cad1,"Hola"); cout << boolalpha << **strcmp**(strcpy(cad1,"Hola"), "Hola"== 0; **Emite**: true	
5	size_t **strcspn**(cad3, cad4)			SI
	Retorna el índice del primer carácter en cad1 que se corresponde con algún carácter de cad2.		cout << **strspn**("cadena o sarta", "edst"); **Emite**: 2, ya que el carácter 'd' esta en cad3.	
6	int **stricmp**(cad3,cad4)			NO
	Compara la cad3 con cad4. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4. **Ignora** minúsculas de mayúsculas.		cout << boolalpha << **stricmp** ("abcd","aBCd") == 0; **Emite**: true La diferencia del código ASCII entre caracteres correspondientes, salvo en los alfabéticos que considera el mismo código ASCII sin importar si es mayúscula y/o minúscula.	
7	size_t **strlen**(cad3)			SI

#Ord.	Función	Significado	Ejemplos	ANSI
	Retorna la longitud de la cad3.		cout << strlen("abcde"); **Emite**: 5.	
8	char ***strlwr**(cad1)			NO
	Convierte la cadena cad1 a minúsculas. **Retorna** la cadena cad1 convertida a minúsculas.		strcpy(cad1,"aBCde34+2xZYt"); cout << **strlwr**(cad1); **Emite**: "abcde34+2xzyt"	
9	char***strncat**(cad1,cad4,n)			**SI**
	Concatena la cad4 al final de cad1, pero hasta n caracteres, agrega '\n', **Retorna** cad1.		**strcpy**(cad1,"Super"); cout << **strncat**(cad1,"mancito",3); **Emite**: Superman.	
10	int **strncmp**(cad3,cad4,n)			SI
	Compara la cad3 con cad4. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4, pero hasta n caracteres.		cout << boolalpha << **strncmp** ("abcdefghij","abcdexyzhij",4); **Emite**: false.	
11	char ***strncpy**(cad1,cad4,n)			SI
	Copia hasta n caracters de cad4 a cad1, si cad4 tiene menos de n caracteres rellena con '\0'. **Retorna** cad1.		cout << **strncpy**(cad1,"Funciones",7); **Emite**: Funcion. cout << **strncpy**(cad1,"Func",7); **Emite**: Func\0\0\0. pero emite Func.	
12	int **strnicmp**(cad3,cad4,n)			NO
	Compara la cad3 con cad4, sin importar las minúsculas con respecto a las mayúsculas. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4.		cout << boolalpha << **strnicmp** ("abcdefghij","abcDExyZhij",4) != 0; **Emite**: false.	
13	char ***strnset**(cad1,car,n)			SI

	Pone el valor car en los primeros n caracteres de cad1, sobreescribiendo los n primeros caracteres. Si cad1 tiene menos caracteres que n, entonces solo sobre escribe sobre esos caracteres. **Retorna** cad1.		strcpy(cad1,"abcdefghij");
			cout << **strnset**(cad1,'*',5);
			Emite: *****fghij
			strcpy(cad1,"abc");
			cout << **strnset**(cad1,'*',5);
			Emite: ***	

14	char ***strpbrk**(cad3,cad4)			SI
	Retorna un puntero al primer carácter de la cadena apuntada por cad1, que se corresponde con algún carácter en la cadena cad2. Si no hay correspondencia retorna NULL.		cout << strpbrk("esto es una prueba","absj"); **Emite**: sto es una prueba	
15	char***strrchr**(cad3,car)			SI
	Retorna un puntero a la última ocurrencia en la cad3. Si no encuentra **retorna** NULL.		cout << **strrchr**("esto es una prueba','s'); **Emite**: s una prueba	
16	char ***strrev**(cad1)			NO
	Invierte la cadena cad1. **Retorna** cad1.		strcpy(cad1,"abcde"); cout << **strrev**(cad1); **Emite**: edcba	
17	char ***strset**(cad1,car)			NO
	Pone car en todos los caracteres de cad1, sobreescribiendolos. **Retorna** cad1.		strcpy(cad1,"abcde"); cout << **strset**(cad1,'*'); **Emite**: *****	
18	size_t **strspn**(cad1, cad2)			SI
	Retorna el índice del primer carácter en cad1 que no se corresponde con nungún carácter de cad2.		cout << **strspn**("cadena o sarta", "eacdst"); **Emite**: 4, ya que el carácter n no esta en cad2.	
19	char ***strstr**(cad3,cad4)			SI

#Ord.	Función	Significado	Ejemplos	ANSI

	Retorna un puntero a la primer aparición de cad4 en cad3 o NULL si no está.		cout<<**strstr**("abcdefghijklmnfghop" , "fgh"); // fghijklmnfghop.
			pcad=**strstr**("abcdefghijabdefgxyz", "defg");
			cout << pcad << endl;
			Emite: defghijabdefgxyz;	

20	char ***strtok**(cad1,cad2)			SI

	Retorna un puntero a la siguiente cadena de cad1, los caracteres de cad2 representan separadores o delimitadores entre las cadenas. La primer invocación el primer argumento es cad1, en próximas invocaciones se indica con NULL o '\0'.		char *token;
			
			token = **strtok**("Esto.es una cadena,de caracteres"," ,.");
			
			while (token) {
			cout << *token << endl;
			token = **strtok**(NULL," ,.");
			}	

21	char ***strupr**(cad1)			NO
	Convierte la cadena cad1 a mayúsculas. **Retorna** cad1.		strcpy(cad1,"aBCde34+2xYz"; cout << **strupr**(cad1); **Emite**: ABCDE34+2XYZ	

```
/* qsort example */

#include <fstream>
#include <stdlib.h>     /* qsort */
using namespace std;

int values[] = { 40, 10, 100, 90, 20, 25 };

int compare (const void * a, const void * b) {
  return ( *(int *) a - *(int *) b );
}

main () {
  int n;
  qsort (values, 6, sizeof(int), compare);
  for (n=0; n<6; n++)
     cout << values[n];
  return 0;
}
```

“CADENAS DE CARACTERES II -Clase string-”

Resumen de funciones de string C++

+= (3)	p.591
append() (6)	p.591
assign() (6)	p.592
at() (2)	p.591
begin()	p.589
c_str()	p.594
capacity()	p.590
clear()	p.591
compare() (4)	p.597
Constructor (7)	p.589
copy()	p.594
empty()	p.591
end()	p.589
erase() (3)	p.593
find() (4)	p.594
find_first_not_of() (4)	p.596
find_first_of() (4)	p.595
find_last_not_of() (4)	p.596
find_last_of() (4)	p.595
getline() (2)	p.599
insert() (7)	p.593
iterator	p.589
length()	p.590
max_size()	p.590
Operador [ ] (2)	p.591
Operador +	p.597
Operador = (3)	p.589
Operador extractor >>	p.599
Operador insertor <<	p.599
Operadores relacionales ==,!=,<,>,>=,<=	p.598
push_back()	p.592
rbegin()	p.590
rend()	p.590
replace() (6)	p.593
reserve()	p.590
resize() (2)	p.590
reverse_iterator	p.590
rfind() (4)	p.595
size()	p.590

static const size_t npos = -1	p.597
substr()	p.596
swap()	p.594

TABLAS – CADENAS CLASE STRING - Tabla de Cadenas estilo C++

Función	Significado	Ejemplos

La clase string permite tratar cadenas de caracteres al estilo de otros lenguajes de programación como el Pascal o Basic, otorgando las facilidades para la manipulación de cadenas de caracteres. Al definir una variable se le asigna de un espacio dinámico variable, es decir, al asignar un nuevo contenido dinámicamente se le asigna el espacio para poder almacenar el contenido asignado, pudiendo ampliar o reducer el espacio de acuerdo a la necesidad del momento. El programador se olvida de solicitor el espacio requerido, ya que ese mecanismo es automático.
La clase string ofrece un conjunto de herramientas que se verán a continuación.

1.Constructor de objeto string.

default (1)	string();
copy (2)	string (const string& str);
substring (3)	string (const string& str, size_t pos, size_t len = npos);
from c-string (4)	string (const char* s);
from sequence (5)	string (const char* s, size_t n);
fill (6)	string (size_t n, char c);
range (7)	template <class InputIterator> string (InputIterator first, InputIterator last);

string s0 (“Initial string”);	Define s0 e inicializa con la constante literal “Initial string”.
string s1;	Define s1 cuyo contenido es la cadena vacía, de longitud 0.
string s2 (s0);	Define s2 copiando el contenido de la variable s0.
string s3 (s0,8,3);	Define s3 copiando de s0 los caracteres “str”.
string s4 (“Another character sequence”);	Define s4 copiando la cadena “A character sequence”
string s5 (“A character sequence”,6);	Define s5 copiando la cadena “A char”.
string s6 (10,’x’);	Define s6 copiando 10 caracteres x’s.
string s7 (s0.begin(),s0.begin()+7);	Define s7 copiando la cadena “Initial”.

2.Operador =

string (1)	string& operator= (const string& str);
c-string (2)	string& operator= (const char* s);
character (3)	string& operator= (char c);

```
string str1, str2, str3;
char cad[21];

str1 = "Test string: ";   // c-string
str2 = 'x';               // single character
strcpy(cad,”C-string”);
str3 = cad; //copia en str3, la cadena “C-string”.
str3 = str1 + str2; //str3 contiene la cadena
                                “Test string: x”.
```

3.Iteradores

begin() / end() **iterator begin();** **const_iterator begin() const;**	string str ("Test string"); for (string::iterator it = str.begin(); it != str.end(); ++it) cout << *it; cout << '\n';

	Emite: Test string
rbegin() / rend() **reverse_iterator rbegin();** **const_reverse_iterator rbegin() const;**	string str ("now step live..."); for(std::string::reverse_iterator rit=str.rbegin();rit!= str.rend(); ++rit) cout << *rit; **Emite**: …evil pets won
4.Capacity	
size() / length() **size_t size() const;**	string str ("Test string"); cout << "El tamaño de str es " << str.**size**() << " bytes.\n"; cout << "La longitud de str es " << str.**length**() << " bytes.\n"; **Emite**: El tamaño de str es 11 bytes **Emite**: La longitud de str es 11 bytes
max_size() **size_t max_size() const;**	string str ("Test string"); cout << "size: " << str.size() << "\n"; cout << "length: " << str.length() << "\n"; cout << "capacity: " << str.capacity() << "\n"; cout << "max_size: " << str.**max_size**() << "\n"; **Emite**: size: 11 length: 11 capacity: 15 max_size: 4294967291
resize() **void resize (size_t n);** **void resize (size_t n, char c);**	string str ("I like to code in C"); cout << str << '\n'; unsigned sz = str.size(); str.resize (sz+2,'+'); cout << str << '\n'; str.**resize** (14); cout << str << '\n'; **Emite**: I like to code in C I like to code in C++ I like to code
capacity() **size_t capacity() const;**	string str ("Test string"); cout << "size: " << str.size() << "\n"; cout << "length: " << str.length() << "\n"; cout << "capacity: " << str.**capacity**() << "\n"; cout << "max_size: " << str.max_size() << "\n"; **Emite**: size: 11 length: 11 capacity: 15 max_size: 429496729

reserve()
void reserve (size_t n = 0);
	string str;
	
	ifstream file ("test.txt",std::ios::in|std::ios::ate);
	if (file) {
	ifstream::streampos filesize = file.tellg();
	str.**reserve**(filesize);
	
	file.seekg(0);
	while (!file.eof())
	{
	str += file.get();

	`}`
	`cout << str;`
	`}`

clear()
void clear();
	`char c;`
	`string str;`
	`cout << "Escriba algunas lineas de texto. Ingrese un punto para finalizar:\n";`
	`do {`
	`c = cin.get();`
	`str += c;`
	`if (c=='\n') {`
	`cout << str;`
	`str.clear();`
	`}`
	`} while (c!='.');`

empty()
bool empty() const;
	`string content;`
	`string line;`
	`cout << "Ingrese un texto. Para terminar Enter: \n";`
	`do {`
	`getline(cin,line);`
	`content += line + '\n';`
	`line = line.substr(0,line.size() - 1); // agregue esta linea.`
	`} while (!line.empty());`
	`cout << "El texto ingresado fue:\n" << content;`

5.Acceso a elementos	

Operador [ ]
char& operator[] (size_t pos);
const char& operator[] (size_t pos) const;
	`string str ("Test string");`
	`for (int i=0; i<str.length(); ++i)`
	`cout << str[i];`
	Emite: Test string

at()
char& at (size_t pos);
const char& at (size_t pos) const;
	`string str ("Test string");`
	`for (unsigned i = 0; i < str.length(); ++i)`
	`cout << str.at(i);`
	Emite: Test string

6.Modificadores	
Operador += **string (1)** **c-string (2)** **character (3)**	 **string& operator+= (const string& str);** **string& operator+= (const char* s);** **string& operator+= (char c);**

`string name ("John");`
`string family ("Smith");`
`name += " K. ";  // c-string`
`name += family;  // string`
`name += '\n';  // character`
 	
`cout << name;`
Emite: John K. Smith	

append()
string (1)
substring (2)
c-string (3)
buffer (4)
fill (5)
	`string& append (const string& str);`
	`string& append (const string& str, size_t subpos, size_t sublen);`
	`string& append (const char* s);`
	`string& append (const char* s, size_t n);`
	`string& append (size_t n, char c);`
	`template <class InputIterator>`

range (6)	string& append (InputIterator first, InputIterator last);

```
string str;
string str2="Writing ";
string str3="print 10 and then 5 more";

// used in the same order as described above:
str.append(str2);                    // "Writing "
str.append(str3,6,3);                // "10 "
str.append("dots are cool",5);       // "dots "
str.append("here: ");                // "here: "
str.append(10u,'.');                 // "........."
str.append(str3.begin()+8,str3.end()); // " and then 5 more"
str.append<int>(5,0x2E);             // "....."

cout << str << '\n';
```

Emite: Writing 10 dots here: ..... and then 5 more.....

push_back()
void push_back (char c);
	string str;
	ifstream file ("test.txt",ios::in);
	if (file) {
	while (!file.eof()) str.**push_back**(file.get());
	}
	cout << str << '\n';
	Emite: el contenido del archivo test.txt

assign()	
string (1)	string& assign (const string& str);
substring (2)	string& assign (const string& str, size_t subpos, size_t sublen);
c-string (3)	string& assign (const char* s);
buffer (4)	string& assign (const char* s, size_t n);
fill (5)	string& assign (size_t n, char c);
range (6)	template <class InputIterator> string& assign (InputIterator first, InputIterator last);

```
string str;
string base="The quick brown fox jumps over a lazy dog.";

// used in the same order as described above:

str.assign(base);
cout << str << '\n';

str.assign(base,10,9);
cout << str << '\n';      // "brown fox"

str.assign("pangrams are cool",7);
cout << str << '\n';      // "pangram"

str.assign("c-string");
cout << str << '\n';      // "c-string"

str.assign(10,'*');
std::cout << str << '\n';      // "**********"

str.assign<int>(10,0x2D);
```

```
cout << str << '\n';        // "----------"

str.assign(base.begin()+16,base.end()-12);
cout << str << '\n';        // "fox jumps over"
```

insert()	
string (1)	string& **insert** (size_t pos, const string& str);
substring (2)	string& **insert** (size_t pos, const string& str, size_t subpos, size_t sublen);
c-string (3)	string& **insert** (size_t pos, const char* s);
buffer (4)	string& **insert** (size_t pos, const char* s, size_t n);
fill (5)	string& **insert** (size_t pos, size_t n, char c);
	void ins**e**rt (iterator p, size_t n, char c);
single character (6)	iterator **insert** (iterator p, char c);
range (7)	template <class InputIterator>
	void **insert** (iterator p, InputIterator first, InputIterator last);

erase()	
sequence (1)	string& erase (size_t pos = 0, size_t len = npos);
character (2)	iterator erase (iterator p);
range (3)	iterator erase (iterator first, iterator last);

```
string str ("This is an example sentence.");
cout << str << '\n';
                                // "This is an example sentence."
str.erase (10,8);                    //          ^^^^^^^^
cout << str << '\n';
                                // "This is an sentence."
str.erase (str.begin()+9);           //         ^
cout << str << '\n';
                                // "This is a sentence."
str.erase (str.begin()+5, str.end()-9);  //      ^^^^^
cout << str << '\n';
                                // "This sentence."
```

replace()	
string (1)	string& replace (size_t pos, size_t len, const string& str);
	string& replace (iterator i1, iterator i2, const string& str);
substring (2)	string& replace (size_t pos, size_t len, const string& str,
	size_t subpos, size_t sublen);
c-string (3)	string& replace (size_t pos, size_t len, const char* s);
	string& replace (iterator i1, iterator i2, const char* s);
buffer (4)	string& replace (size_t pos, size_t len, const char* s, size_t n);
	string& replace (iterator i1, iterator i2, const char* s, size_t n);
fill (5)	string& replace (size_t pos, size_t len, size_t n, char c);
	string& replace (iterator i1, iterator i2, size_t n, char c);
range (6)	template <class InputIterator>
	string& replace (iterator i1, iterator i2, InputIterator first, InputIterator last);

```
string base="this is a test string.";
string str2="n example";
string str3="sample phrase";
string str4="useful.";

// replace signatures used in the same order as described above:

// Using positions:            0123456789*123456789*12345
string str=base;          // "this is a test string."
```

```
str.replace(9,5,str2);          // "this is an example string." (1)
str.replace(19,6,str3,7,6);     // "this is an example phrase." (2)
str.replace(8,10,"just a");     // "this is just a phrase."     (3)
str.replace(8,6,"a shorty",7);  // "this is a short phrase."    (4)
str.replace(22,1,3,'!');        // "this is a short phrase!!!"  (5)

// Using iterators:                                  0123456789*123456789*
str.replace(str.begin(),str.end()-3,str3);           // "sample phrase!!!"      (1)
str.replace(str.begin(),str.begin()+6,"replace");    // "replace phrase!!!"     (3)
str.replace(str.begin()+8,str.begin()+14,"is coolness",7);   // "replace is cool!!!"    (4)
str.replace(str.begin()+12,str.end()-4,4,'o');       // "replace is cooool!!!"  (5)
str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());// "replace is useful."    (6)
cout << str << '\n';
```

swap()
void swap (string& str);

```
string buyer ("money");
string seller ("goods");

cout << "Before the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';

seller.swap (buyer);

cout << " After the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';
```

7.Operaciones con string

c_str()
const char* c_str() const;

```
string str ("Please split this sentence into tokens");

char * cstr = new char [str.length()+1];
strcpy (cstr, str.c_str());

// cstr now contains a c-string copy of str

char * p = strtok (cstr," ");
while (p!=0)
{
  cout << p << '\n';
  p = strtok(NULL," ");
}

delete[] cstr;
```

copy()
size_t copy (char* s, size_t len, size_t pos = 0) const;

```
char buffer[20];
string str ("Test string...");
size_t length = str.copy(buffer,6,5);
buffer[length]='\0';
cout << "buffer contains: " << buffer << '\n';
```

Emite: buffer contains: string

find()

string (1)	size_t find (const string& str, size_t pos = 0) const;
c-string (2)	size_t find (const char* s, size_t pos = 0) const;
buffer (3)	size_t find (const char* s, size_t pos, size_t n) const;
character (4)	size_t find (char c, size_t pos = 0) const;

```
string str ("There are two needles in this haystack with needles.");
string str2 ("needle");

// different member versions of find in the same order as above:
```

```
size_t found = str.find(str2);
if (found != string::npos)
   cout << "first 'needle' found at: " << found << '\n';

found=str.find("needles are small",found+1,6);
if (found != string::npos)
   cout << "second 'needle' found at: " << found << '\n';

found=str.find("haystack");
if (found != string::npos)
   cout << "'haystack' also found at: " << found << '\n';

found=str.find('.');
if (found != string::npos)
   cout << "Period found at: " << found << '\n';

// let's replace the first needle:
str.replace(str.find(str2),str2.length(),"preposition");
cout << str << '\n';
```

rfind()	
string (1)	size_t rfind (const string& str, size_t pos = npos) const;
c-string (2)	size_t rfind (const char* s, size_t pos = npos) const;
buffer (3)	size_t rfind (const char* s, size_t pos, size_t n) const;
character (4)	size_t rfind (char c, size_t pos = npos) const;

```
string str ("The sixth sick sheik's sixth sheep's sick.");
string key ("sixth");

size_t found = str.rfind(key);
if (found != string::npos)
  str.replace (found,key.length(),"seventh");

cout << str << '\n';
```

Emite: The sixth sick sheik's seventh sheep's sick.

find_first_of()	
string (1)	size_t find_first_of (const string& str, size_t pos = 0) const;
c-string (2)	size_t find_first_of (const char* s, size_t pos = 0) const;
buffer (3)	size_t find_first_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_first_of (char c, size_t pos = 0) const;

```
string str ("Please, replace the vowels in this sentence by asterisks.");
size_t found = str.find_first_of("aeiou");
while (found != string::npos)
{
  str[found] = '*';
  found = str.find_first_of("aeiou",found+1);
}
cout << str << '\n';
```

Emite: Pl**s*, r*pl*c* th* v*w*ls *n th*s s*nt*nc* by *st*r*sks.

find_last_of()	
string (1)	size_t find_last_of (const string& str, size_t pos = npos) const;
c-string (2)	size_t find_last_of (const char* s, size_t pos = npos) const;
buffer (3)	size_t find_last_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_last_of (char c, size_t pos = npos) const;

```
cout << "Splitting: " << str << '\n';
```

```
  size_t found = str.find_last_of("/\\");
  cout << " path: " << str.substr(0,found) << '\n';
  cout << " file: " << str.substr(found+1) << '\n';
}

int main () {
  string str1 ("/usr/bin/man");
  string str2 ("c:\\windows\\winhelp.exe");

  SplitFilename (str1);
  SplitFilename (str2);
```

Emite:

```
        Splitting: /usr/bin/man
        path: /usr/bin
        file: man
        Splitting: c:\windows\winhelp.exe
        path: c:\windows
        file: winhelp.exe
```

find_first_not_of()	
string (1)	size_t find_first_not_of (const string& str, size_t pos = 0) const;
c-string (2)	size_t find_first_not_of (const char* s, size_t pos = 0) const;
buffer (3)	size_t find_first_not_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_first_not_of (char c, size_t pos = 0) const;

```
  string str ("look for non-alphabetic characters...");

  size_t found = str.find_first_not_of("abcdefghijklmnopqrstuvwxyz ");

  if (found != string::npos)
  {
    cout << "The first non-alphabetic character is " << str[found];
    cout << " at position " << found << '\n';
  }
```

Emite: The first non-alphabetic character is - at position 12

find_last_not_of()	
string (1)	size_t find_last_not_of (const string& str, size_t pos = npos) const;
c-string (2)	size_t find_last_not_of (const char* s, size_t pos = npos) const;
buffer (3)	size_t find_last_not_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_last_not_of (char c, size_t pos = npos) const;

```
  string str ("Please, erase trailing white-spaces   \n");
  string whitespaces (" \t\f\v\n\r");

  size_t found = str.find_last_not_of(whitespaces);
  if (found!=std::string::npos)
    str.erase(found+1);
  else
    str.clear();            // str is all whitespace

  cout << '[' << str << "]\n";
```

Emite: [Please, erase trailing white-spaces]

substr()	

string substr (size_t pos = 0, size_t len = npos) const;	string str="We think in generalities, but we live in details."; // (quoting Alfred N. Whitehead)
	
	string str2 = str.substr (3,5); // "think"

```
size_t pos = str.find("live");      // position of "live" in str

string str3 = str.substr (pos);     // get from "live" to the end

cout << str2 << ' ' << str3 << '\n';
```

Emite: think live in details.

compare()

string (1)	int compare (const string& str) const;
substrings (2)	int compare (size_t pos, size_t len, const string& str) const;
	int compare (size_t pos, size_t len, const string& str, size_t subpos, size_t sublen) const;
c-string (3)	int compare (const char* s) const;
	int compare (size_t pos, size_t len, const char* s) const;
buffer (4)	int compare (size_t pos, size_t len, const char* s, size_t n) const;

```
string str1 ("green apple");
string str2 ("red apple");

if (str1.compare(str2) != 0)
  cout << str1 << " is not " << str2 << '\n';

if (str1.compare(6,5,"apple") == 0)
  cout << "still, " << str1 << " is an apple\n";

if (str2.compare(str2.size()-5,5,"apple") == 0)
  cout << "and " << str2 << " is also an apple\n";

if (str1.compare(6,5,str2,4,5) == 0)
  cout << "therefore, both are apples\n";
```

Emite:

```
green apple is not red apple
still, green apple is an apple
and red apple is also an apple
therefore, both are apples
```

8.Constante miembro

static const size_t npos = -1;	Es el máximo valor de tipo size_t Al asignar el valor -1 su complemento representa el valor más grande possible, ya que, size_t es un tipo entero sin signo. En las cadenas de tipo string es utilizado para alcanzar el fi n de la cadena.

9.Sobrecarga de funciones no miembros

Operador +

string (1)	string operator+ (const string& lhs, const string& rhs);
c-string (2)	string operator+ (const string& lhs, const char* rhs);
	string operator+ (const char* lhs, const string& rhs);
character (3)	string operator+ (const string& lhs, char rhs);
	string operator+ (char lhs, const string& rhs);

```
string firstlevel ("com");
string secondlevel ("cplusplus");
string scheme ("http://");
string hostname;
string url;
```

```
hostname = "www." + secondlevel + '.' + firstlevel;
url = scheme + hostname;

cout << url << '\n';
```
Emite: http://www.cplusplus.com

Operadores relacionales

(1)

bool operator== (const string& lhs, const string& rhs);

```
bool operator== (const char*   lhs, const string& rhs);
bool operator== (const string& lhs, const char*   rhs);
```

(2)

```
bool operator!= (const string& lhs, const string& rhs);
bool operator!= (const char*   lhs, const string& rhs);
bool operator!= (const string& lhs, const char*   rhs);
```

(3)

```
bool operator<  (const string& lhs, const string& rhs);
bool operator<  (const char*   lhs, const string& rhs);
bool operator<  (const string& lhs, const char*   rhs);
```

(4)

```
bool operator<= (const string& lhs, const string& rhs);
bool operator<= (const char*   lhs, const string& rhs);
bool operator<= (const string& lhs, const char*   rhs);
```

(5)

```
bool operator>  (const string& lhs, const string& rhs);
bool operator>  (const char*   lhs, const string& rhs);
bool operator>  (const string& lhs, const char*   rhs);
```

(6)

```
bool operator>= (const string& lhs, const string& rhs);
bool operator>= (const char*   lhs, const string& rhs);
bool operator>= (const string& lhs, const char*   rhs);
```

```
string foo = "alpha";
string bar = "beta";

if (foo==bar) cout << "foo and bar are equal\n";
if (foo!=bar)  cout << "foo and bar are not equal\n";
if (foo< bar)  cout << "foo is less than bar\n";
if (foo> bar)  cout << "foo is greater than bar\n";
if (foo<=bar) cout << "foo is less than or equal to bar\n";
if (foo>=bar) cout << "foo is greater than or equal to bar\n";
```
Emite: foo and bar are not equal
foo is less than bar
foo is less than or equal to bar

swap()

void swap (string& x, string& y);

```
string buyer ("money");
string seller ("goods");

cout << "Before the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';

swap (buyer,seller);

cout << " After the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';
```

	Emite: Before the swap, buyer has money and seller has goods After the swap, buyer has goods and seller has money
Operador >> -extractor- **istream& operator>> (istream& is, string& str);**	string name; cout << "Please, enter your name: "; cin >> name; cout << "Hello, " << name << "!\n"; **Emite**: Please, enter your name: Juan Hello, Juan
Operador << -insertor- **ostream& operator<< (ostream& os, const string& str);**	string str = "Hello world!"; cout << str << '\n'; **Emite**: Hello world!
getline() (1) **istream& getline (istream& is, string& str, char delim);** **(2)** **istream& getline (istream& is, string& str);**	Extracts characters from is and stores them into str until the delimitation character delim is found (or the newline character, '\n', for (2)). The extraction also stops if the end of file is reached in is or if some other error occurs during the input operation. If the delimiter is found, it is extracted and discarded (i.e. it is not stored and the next input operation will begin after it). Note that any content in str before the call is replaced by the newly extracted sequence. Each extracted character is appended to the string as if its member push_back was called. std::string name; cout << "Please, enter your full name: "; getline (std::cin,name); cout << "Hello, " << name << "!\n"; **Emite**: Please, enter your full name: Juan García Hello, Juan García!

“CARACTERES <ctype.h>”

TABLAS – CARACTERES - Tabla Funciones de Caracteres

#Ord.	Función	Significado	Ejemplos
El header es <ctype.h>. car es tipo int o unsigned char. Retorna un int. Las funciones retorna un valor distinto de cero, si el parámetro satisfice la condición, sino cero, falso.			
1	int isdigit(car)	Retorna verdadero si car es un dígito 0-9, sino, retorna 0, falso.	int num = 0; **for**(unsigned i = 0; i < strlen(cad); i++) **if**(isdigit(cad[i]) num = num * 10 + cad[i] – ‘0’;
2	int isalpha(car)	Retorna verdadero si car es alfabético, sino, retorna 0, falso.	
3	int isalnum(car)	Retorna verdadero si car es letra o dígito (A-Z, a-z, 0-9), sino, retorna 0, falso.	
4	int isxdigit(car)	Retorna verdadero si car es dígito 0-0, A-Z,a-z, sino, retorna 0, falso.	
5	int islower(car)	Retorna verdadero si car es letra minúscula, sino, retorna 0, falso.	char car = ‘b’; if (islower(car)) car = car – (‘a’ – ‘A’);
6	int isupper(car)	Retorna verdadero si car es letra mayúscula, sino, retorna 0, falso.	char car = ‘A’; if (isupper(car)) car = car + (‘a’ – ‘A’);
7	int isspace(car)	Retorna verdadero si car es espacio en blanco ‘ ‘,o nueva línea ‘\n’, o avance de página ‘\f’, o retorno de carro ‘\r’, o tabulador horizontal ‘\t’ o tabulador vertical ‘\v’, sino, retorna 0, falso.	
8	int iscntrl(car)	Retorna verdadero si car es un caracter de control, sino, retorna 0, falso.	
9	int ispunct(car)	Retorna verdadero si car es un carácter de impresión diferente de espacio, dígito o letra, sino, retorna 0, falso.	
10	int isprint(car)	Retorna verdadero si car	

		es un caracter de impresion, incluido el espacio, sino, retorna 0, falso.	
11	int isgraph(icart)	Retorna verdadero si car es un carácter de impresión diferente del espacio, sino, retorna 0, falso.	

12	int tolower(car)	Retorna un character convertido a minúscula si car es letra mayúscula, sino retorna el mismo car.	char car = 'A';
			if (isupper(car))
			car = tolower(car);

13	int toupper(car)	Retorna el carácter convertido a mayúscula, si car es letra minúscula, sino, retorna 0, falso.	char car = 'b';
			if (islower(car))
			car = toupper(car);

OBTENER LA FECHA Y HORA DEL SISTEMA

Obtener la Fecha y Hora del Sistema

```
/*
   Id.Programa: ObtFechaHora.cpp
   Autor......: Lic.Hugo Cuello
   Fecha......: jul-2014
   Comentario.: Obtiene la fecha y hora del Sistema.
*/

#include <time.h>
#include <fstream>
using namespace std;

long GetTime(int &hh, int &mm, int &ss) {
  time_t     rawtime;
  struct tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  hh = timeinfo->tm_hour;
  mm = timeinfo->tm_min;
  ss = timeinfo->tm_sec;
  return timeinfo->tm_hour * 10000 + timeinfo->tm_min * 100 + timeinfo->tm_sec;
} // GetTime

long GetDate(int &year, int &mes, int &dia, int &ds) {
  time_t     rawtime;
  struct tm *timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  year = 1900 + timeinfo->tm_year;
  mes  = 1 + timeinfo->tm_mon;
  dia  = timeinfo->tm_mday;
  ds   = 1 + timeinfo->tm_wday;
```

```
 return (1900 + timeinfo->tm_year) * 10000 + (1 + timeinfo->tm_mon) * 100 +
timeinfo->tm_mday;
} // GetDate

main() {
 int  h,m,s;
 long horaL;
 int  year,mes,dia,diaSem;
 long fechaL;

 horaL = GetTime(h,m,s);
 cout << "La hora es: " << h << ":" << m << ":" << s << endl;
 cout << "La hora larga es: " << horaL << endl;
 fechaL = GetDate(year,mes,dia,diaSem);
 cout << "La fecha es: " << dia << "/" << mes << "/" << year << endl;
 cout << "Dia de la semana: " << diaSem << endl;
 cout << "La fecha larga es: " << fechaL << endl;
 return 0;
}
```

FUNCIONES PARA ARCHIVOS DE TEXTO Y BINARIO –estilo FILE*-

TABLAS – ARCHIVOS FILE* - Funciones para archivos de Texto y Binario

ARCHIVOS
NL indica Nombre Lógico, variable de tipo archivo. **NF** indica Nombre Físico. **NR** indica Nombre de Registro. **Dir** indica dirección o referencia o posición. **LUGAR** indica **SEEK_SET** considera que el puntero es desde el inicio, **SEEK_CUR** considera que el puntero es desde el lugar actual, **SEEK_END** considera que el puntero es desde el final. **IMPORTANTE**: Si se actualiza un archivo, en la modalidad “r+b” o “w+b”, se debe indicar después de grabar el registro previamente leído, la próxima posición con fseek(NL,ftell(NL),SEEK_SET), ya que no avanza el puntero automáticamente, a efectos de continuar con la lectura secuencial de los próximas componentes. Pág.266 Apéndice B “El Lenguaje de Programación C” 2da.Ed. – Kernighan & Ritchie. La ruta del nombre físico del archivo se utiliza ‘/’ para cambiar de niveles entre las carpetas, y NO ‘\’ o en su lugar “\\” doble barra invertida. Ej.: “C:/Borland/BCC55/prgsFtes/Articulos.Dat”. Ej.: “C:\\Borland\\BCC55\\prgsFtes\\Articulos.Dat”.

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++
Tipo	Tipo *text* Un archivo de texto se compone de líneas y cada línea finaliza con una marca de fin de línea **eoln**. Función **eoln**(NL). Retorna true si el puntero está sobre la marca eoln, sino, false. El fin del archivo está indicado por otra marca, en este caso la marca de fin de archivo **eof**. Función **eof**(NL). Retorna true si el puntero está sobre la marca de eof, sino, false.	En el modo de apertura "t".	Tipo **file of** *tipo* *tipo* puede ser cualquier tipo simple de dato primitivo o cadena de caracteres, o puntero; como cualquier tipo estructurado de dato, registro, conjunto o arreglo o combinadas arreglo de registros, arreglo de conjuntos. La única excepción es el tipo archivo, sea de text o file of.	En el modo de apertura "b".
Asociar NL, NF	assign(NL,NF)		Assign(NL,NF)	
Apertura	**reset**(NL) Debe existir, sino es error. Ubica el puntero al archivo en la primer componente o primera línea. **rewrite**(NL) Si existe lo elimina y lo crea desde cero. Si no existe no es error y lo crea desde cero. Ubica el puntero sobre la marca eof. **append**(NL) Si existe ubica el puntero al final sobre la marca eof, permitiendo agregar nuevas componentes. Un archivo de texto abierto con reset, solo puede leerse. Un archivo de texto abierto con rewrite, solo puede grabarse. Un archivo de texto abierto con append, solo puede agregar componentes al final del archivo.	aTxt=**fopen**(NF,"r"), o fopen(NL,"w"), o fopen(NL,"a"), o fopen(NL,"rt"), o fopen(NL,"wt"), o fopen(NL,"at"), o fopen(NL,"r+"), o fopen(NL,"w+"), o fopen(NL,"a+"), o fopen(NL,"r+t"), o fopen(NL,"w+t"), o fopen(NL,"a+t");, o fopen(NL,"rt+"), o fopen(NL,"wt+), o fopen(NL,"at+"). **freopen**(NF,modo,flujo) flujo puede ser stdin, stdout, stderr. Asocia un nuevo archivo con un flujo ya abierto, al cual cerrará. Retorna NULL si no pudo llevarse a cabo la reapertura. Para cerrar un flujo estandar redireccionado a otro flujo, hacer, **fclose**(stdout) esto restituye la salida por defecto que es la pantalla. También se puede usar	**reset**(NL) **rewrite**(NL) *filemode* es una variable primitiva cuyo valor por omisión es 2, indicando que puede leer o grabar si se abre con reset. Si se asigna un valor de 0 solo lectura, un valor de 1 solo grabar. Si se abre con rewrite siempre se considera el valor 2, o dicho de otra manera, no se considera el valor de filemode, por lo que se puede leer o grabar.	aBin=**fopen**(NF,"rb"), o fopen(NF,"wb"), o fopen(NL,"r+b"), o fopen(NL,"w+b"), o fopen(NL,"rb+"), o fopen(NL,"wb+") o fopen(NL,"ab") o fopen (NL,"ab+") o fopen(NL,"a+b").

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++
		freopen("CON","w",std out).		

Operación	**<u>Leer</u>**:
	read(NL,var,…)
	readln(NL,var,…)
	
	<u>Grabar</u>:
	write(NL,exp,…)
	writeln(NL,exp,…)
		<u>Leer</u>:
		fscanf(NL,*formato*,var, …)
		int fgetc(NL) o retorna EOF si es fin de archive o error.
		char* fgets (cad, cant, NL) lee hasta n-1 cars, si encuentra nueva linea se detiene, incluyéndose en cad terminado por '\0', retorna cad o NULL si es fin de archivo o error.
		Int getc(FILE *f) equivalente a fgetc.
		getchar() // teclado+ENTER
		getch() // sin eco.
		getche() // con eco.
		Getw()
		scanf(form,&var) // teclado
		cin >> var // teclado C++
		
		<u>Grabar</u>:
		Int fprintf (NL, *formato*, exp,…)
		int fputc(car,NL)
		int fputs(cad3,NL)
		int putc(car,FILE *f) equivalente a fputc.
		int putchar(car) // pantalla.Equivalente a putc(car, stdout)
		putw(int,f) // 2 bytes.
		printf(form,exp) //
			<u>Leer</u>:
			read(NL,NR)
			El registro a leer es donde está ubicado el puntero del archivo, luego de esta operación el puntero avanza automáticamente a la próxima componente. Leer fuera de los límites del archivo siempre es error.
			<u>Grabar</u>:
			write(NL,NR)
			El registro a grabar es en donde se encuentra ubicado el puntero del archivo, luego de esta operación el puntero avanza automáticamente a la próxima componente. Grabar antes de la posición cero es error, pero grabar más allá de la marca de eof no es error y además de crear la
				<u>Leer</u>:
				fread(&NR,tamaño ,cant,NL)
				
				<u>Grabar</u>:
				fwrite(&NR,tamañ o,cant,NL)

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++
		pantalla **cout** << exp // pantalla C++	componente se generar todas aquellas componentes entre la marca eof hasta la componente recién creada.	
Cierre	**close**(NL) Cerrar un archivo de texto implica vaciar el buffer y luego cierra el archivo. Si un archivo de texto no es cerrado y si el buffer no se había completado implica que se perderán esos datos, ya que no se grabarán en el archivo. En un archivo binario esta situación no sucede. De todas maneras es conveniente cerrar siempre un archivo abierto.	**fclose**(NL) **fcloseall**()	**close**(NL)	**fclose**(NL) **fcloseall**()
Estado de fin de archivo	**eof**(NL) Retorna true si el puntero está sobre la marca eof, sino, false. Es útil utilizar esta función cuando recorremos secuencialmente el archivo para leer.	**feof**(NL) **ferror**(FILE *f)	**eof**(NL)	**feof**(NL) **ferror**(FILE *f)
Estado posición del puntero			**filepos**(NL) Retorna la posición del puntero en cantidad de componentes.	**ftell**(NL) Retorna la posición del puntero en cantidad de bytes. Para convertir a cantidad de componentes, se debe realizar el siguiente cálculo: **ftell**(NL) / **sizeof**(NR)
Estado tamaño del archivo			**filesize**(NL) Retorna el tamaño del archivo en cantidad de componentes.	No existe un comando específico para esta acción. No obstante, puede realizarse con otros comandos, **fseek** y ftell.
Mover el puntero a otra posición			**seek**(NL,DIR) Mueve el puntero del archivo indicado a la	**fseek**(NL,DIR,ORIGEN) ORIGEN es **SEEK_SET** la

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++
			posición indicada por DIR	cantidad de componentes a desplazar se considera desde el incio del archivo; si es **SEEK_CUR** se considera desde la posición actual del puntero; si es **SEEK_END** se considera desde el final del archivo. <u>Ejemplos</u>: Suponiendo que el archivo contiene 15 componentes posiciones de 0 a 14, y que el puntero se encuentra en la posición 7, que es la componente 8, luego de hacer: fseek(NL,5,SEEK_SET) el puntero se ubica en la posición 5 que es la componente 6. fseek(NL,5,SEEK_CUR) el puntero se ubica en la posición 7 + 5 = 12 o componente 13. fseek(NL,5,SEEK_END) el puntero se ubica en la posición 15 + 5 = 20 o componente 21. **rewind**(NL); Mueve el puntero al inicio.
Cortar el archivo en una posición seleccionada			**truncate**(NL)	No existe una función equivalente al procedimiento truncate de Turbo Pascal. En cambio, si se accede a los servicios brindados por el D.O.S. interrupción 21h mediante la

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++

				siguiente función de usuario se obtiene su equivalente.
				
				#include <dos.h>
				
				struct REGPACK rf;
				
				void truncate(**FILE** *f) {
				
				fseek(f,ftell(f),SEEK_SET);
				// Posición actual en
				// cantidad de bytes.
				rf.r_ax = 0x4000;
				// Servicio
				// grabar aleatorio.
				Rf.r_bx = fileno(f);
				// Número de hadle.
				Rf.r_cx = 0;
				// Actualizar el tamño
				// del archivo.
				Rf.r_ds = FP_SEG(fileno(f));
				// Segmento del buffer
				// de dato.
				Rf.r_dx = FP_OFF(fileno(f));
				// Desplazamiento del
				// buffer de dato.
				Intdos(&rf,&rf);
				// Invoca int 21h.
				} // truncate

Cambiar el nombre físico a un archivo.	**rename**(NL,NvoNF);	**rename**(VjoNF,NvoNF);	**rename**(NL,NvoNF)	**rename**(VjoNom,NvoNom); VjoNom y NvoNom son cadenas, no se permite uso de comodines, retorna cero si la operación fue correcta, sino -1L y errno se pone

Motivo	Texto en Pascal	Texto en C/C++	Binario en Pascal	Binario en C/C++
				a **ENOENT** indica, Ningún archivo o directorio. **EACCES** indica, Permiso negado. **ENOTSAM** indica, Ningún dispositivo.
Eliminar físicamente un archivo.	**erase**(NL);	**remove**(NF);	**erase**(NL);	**remove**(NF); El archivo debe estar cerrado, se permite indicar la ruta de acceso, retorna cero sil a operación fue correcta, sino, -1L y errno se pone a : ENOENT indica, Ningún archivo o directorio. EACCES indica, Permiso negado.

FUNCIONES PARA ARCHIVOS DE TEXTO Y BINARIO
-estilo C++ stream-

TABLAS ARCHIVOS ESTILO C++ STREAMS – Funciones Texto y Binario

NL indica Nombre Lógico, variable de tipo archivo. NF indica Nombre Físico. NR indica Nombre de Registro. Dir indica dirección o referencia o posición. LUGAR indica **ios::beg** considera que el puntero es desde el inicio, **ios::cur** considera que el puntero es desde el lugar actual, **ios::end** considera que el puntero es desde el final.

IMPORTANTE: Si se recorre el archivo secuencialmente hasta encontrar el fin de archivo, a continuación se debe limpiar el estado del flag que indica que se llegó al final del archivo por medio de la siguiente acción **f.clear()**, ya que al no realizar esta acción no se podrá utilizar el archivo, en donde f indica el stream.

La ruta del nombre físico del archivo se utiliza '/' para cambiar de niveles entre las carpetas, y NO '\' o en su lugar "\\" doble barra invertida.

Ej.: "C:/Borland/BCC55/prgsFtes/Articulos.Dat".

Ej.: "C:\\Borland\\BCC55\\prgsFtes\\Articulos.Dat".

Input/Output library

click on an element for detailed information

Tabla funciones miembro para archivos de Texto y Binario estilo C++ stream

Acción	FILE *	STREAM	Comentario
ABREVIATURAS NL o f : Nombre Lógico NF : Nombre Físico NR o buf : Nombre de Registro MA : Modo de Apertura NFV : Nombre Físico Viejo NFN : Nombre Físico Nuevo DIR : Dirección			
Definir variable (NL)	FILE *f;	ifstream f; ofstream f; fstream f;	f para leer f para grabar f para leer/grabar
	Ejemplos: ifstream Articulos(“Articulos.Dat”,ios::out | ios::app); Se definió un stream y se lo asoció al nombre físico. ifstream Articulos; Articulos.open(“Articulos.Dat”,ios::binary | ios::in); Primero se definió un stream pero sin asociarlo a un nombre físico, por lo que será necesario utilizar posteriormente la función miembro open.		
Abrir	f = fopen(NF, MA);	f.open(NF, MA);	MA: ios::in | ios::out ios::binary | ios::trunc ios::app | ios::ate
Redirccionar Salida	freopen(“NF”,MA,dispositivo)		freopen(“Salida.Txt”,”w”,stdout) Las salidas de printf o cout van a un archivo de texto “Salida.Txt”.
Leer	fread(NL, NR);	f.read((char*) &buf, size); f.read(**reinterpret_cast** <char*> (&block), sizeof block);	Leer desde el stream una cantidad de size bytes o caracteres.
Grabar	fwrite(NL, NR);	f.write((const char*) &buf, size); f.write(**reinterpret_cast** <const char*> (&block), sizeof block);	Graba al stream una cantidad de size bytes o caracteres.
Cerrar	fclose(NL);	f.close();	Cierra un stream
Cerrar todos los archivos	fcloseall()		Cierra todos los archivos abiertos.
Fin de archivo	feof(NL)	f.eof()	Retorna true si el puntero al archivo está ubicado sobre la marca de fin de archivo, sino, false.
Decir posición	ftell(NL)	f.tellg() f.tellp()	Abierto para leer Abierto para grabar

Mover posición Acceso al azar	fseek(NL, DIR)	f.seekg(pos, whereas);
		
		f.seekp(pos, whereas);
			Abierto para leer
			whereas:
			ios::beg, cur, end
			Abierto para grabar
			whereas:
			ios::beg, cur, end

Mover puntero al inicio del archivo	rewind(NL)	f.seekg(0,ios::beg) f.seekp(0,ios::beg)	Regresa el puntero del archivo indicado como NL al inicio del archivo, es decir, rebobina la cinta del carrete.
Detectar abierto		is_open()	false o true dependiendo de si no pudo o pudo abrir el archivo respectivamente.
Leer carácter		f.get(car);	Lee un carácter desde el stream.
Grabar carácter		f.put(car);	Graba un carácter al stream.
Estado apertura		f.bad();	true si ha ocurrido un error de apertura.
		f.good();	true si no ha ocurrido un error en la apertura.
		f.fail();	true si ha ocurrido un error de apertura.
Limpiar error.		f.clear()	Establece un nuevo valor para las banderas del estado de error internos de la secuencia. Un tipo de error al intentar leer al final del stream provoca error -1.
Leer línea car's.		f.getline(char*, streamsize n); f.getline(char* ,streamzise n, char delim);	Lee una línea desde el stream o n caracteres.
Renombrar	rename(NFV,NFN)	rename(NFV,NFN)	Renombra el archivo NFV con un nuevo nombre indicado en NFN.
Eliminar	remove(NF)	remove(NF)	Elimina físicamente el archivo indicado como NF.

TIPO ENUM - Tipo "enum" o enumerado

```
enum [<identificador_de_enum>] {
   <nombre> [= <valor>], ...}
<identificador>[,<identificador2>[,<identificador3>]...];
enum <identificador_de_enum> {
   <nombre> [= <valor>], ...}
[<identificador>[,<identificador2>[,<identificador3>]...]];
```

Se trata de una sintaxis más elaborada que las que hemos visto hasta ahora.

Este tipo nos permite definir conjuntos de constantes enteras, llamados datos de tipo enumerado. Las variables declaradas de este tipo sólo podrán tomar valores dentro del dominio definido en la declaración.

Vemos que hay dos sintaxis. En la primera, el identificador de tipo es opcional, y si lo usamos podremos declarar más variables del tipo enumerado en otras partes del programa:

```
[enum] <identificador_de_enum>
<identificador>[,<identificador2>[,<identificador3>]...];
```

La segunda sintaxis nos permite añadir una lista de variables, también opcional.

De este modo podemos separar la definición del tipo enumerado de la declaración de variables de ese tipo:

```
enum orden {primero=1, segundo, tercero};
...
enum orden  id1, id2, id3;
```

O podemos hacer ambas cosas en la misma sentencia: definición y declaración:

```
enum orden {primero=1, segundo, tercero} id1, id2, id3;
```

Si decidimos no usar un identificador para el enumerado sólo podremos declarar variables en ese momento, y no en otros lugares del programa, ya que no será posible referenciarlo:

```
enum {primero=1, segundo, tercero} uno, dos;
```

Varios identificadores pueden tomar el mismo valor, pero cada identificador sólo puede usarse en un tipo enumerado. Por ejemplo:

```
enum tipohoras { una=1, dos, tres, cuatro, cinco,
  seis, siete, ocho, nueve, diez, once,
  doce, trece=1, catorce, quince,
  dieciseis, diecisiete, dieciocho,
  diecinueve, veinte, ventiuna,
  ventidos, ventitres, venticuatro = 0};
```

En este caso, una y trece valen 1, dos y catorce valen 2, etc. Y veinticuatro vale 0. Como se ve en el ejemplo, una vez se asigna un valor a un elemento de la lista, los siguientes toman valores correlativos. Si no se asigna ningún valor, el primer elemento tomará el valor 0.

Los nombres de las constantes pueden utilizarse en el programa, pero no pueden ser leídos ni escritos. Por ejemplo, si el programa en un momento determinado nos pregunta la hora, no podremos responder doce y esperar que se almacene su valor correspondiente. Del mismo modo, si tenemos una variable enumerada con el valor doce y la mostramos por pantalla, se mostrará 12, no doce. Deben considerarse como "etiquetas" que sustituyen a enteros, y que hacen más comprensibles los programas. Insisto en que internamente, para el compilador, sólo son enteros, en el rango de valores válidos definidos en cada enum.

La lista de valores entre las llaves definen lo que se denomina el "dominio" del tipo enumerado. Un dominio es un conjunto de valores posibles para un dato. Una variable del tipo enumerado no podrá tomar jamás un valor fuera del dominio.

CONCLUSIÓN

Conclusión

Las variantes que podemos formar con las **ESTRUCTURAS DE DATOS** aprendidas en la cursada de "Algoritmos y Estructuras de Datos", solo dependerán de nuestra *imaginación*, *creatividad*, o el *ingenio*, para representar al **mundo real**.

El uso de una **metodología** para resolver problemas, como fue analizada en las primeras clases, nos brindará de un cimiento sólido para encarar la solución. Es imprescindible primero **comprender** el problema, luego de lo cual, elaboraremos una **estrategia**, en la cual destacaremos las *partes relevantes*, y por *refinamiento sucesivo* iremos ampliando nuestro grado de detalles; por lo tanto, siempre iniciaremos nuestra solución **desde lo general hacia lo particular**. Luego construiremos nuestro **algoritmo** el cual dará la solución a nuestro problema. Por último, **codificaremos** el algoritmo con un lenguaje de computadora, para que pueda ser ejecutado por la máquina.

Es importante elaborar una **muestra de datos** bien pensada para la situación, que contemple todas las posibilidades, incluso las más extremas y de los **resultados que esperaríamos obtener** para luego compararlo con los resultados conseguidos con la ejecución del programa por la computadora.

Definimos Algoritmo como la concatenación de sentencias de asignaciones internas y/o externas que son regidas por **ESTRUCTURAS DE CONTROL DE PROGRAMA** – *concatenación*, *selección* y *repetición*-, que determinan el orden de ejecución de esas sentencias. Por lo tanto, la solución no solo correcta, sino, óptima dependerá mucho de las estructuras de control seleccionadas, es decir, debemos plantearnos que tipo de estructura de control debemos utilizar, en el caso de repetición, un ciclo mientras o hasta o exacto; en una selección una selección simple o múltiple, ciclos anidadas o en secuencia, decisiones anidadas o en secuencia.

El uso adecuado de las estructuras elegidas, tanto para los datos como para el control de programa harán que nuestro algoritmo sea más sencillo, simple, compacto, más rápido para ejecutar, y esto no es poca cosa.

Encontrar alternativas de solución para un mismo problema nos va a permitir seleccionar la más adecuada, oportuna, en otras palabras, la solución óptima. El que encontremos diferentes alternativas, no implica que no existan otras. Lo importante es obtener un abanico de posibilidades y decidir cual elegir y no quedarnos con la primera solución encontrada.

Sitios web de obtención de información utilizadas en esta documentación

Sitios web de información empleada en el documento

http://c.conclase.net/curso/
Teoría con ejemplos de los temas explicados de C / C++ en castellano.

http://www.cplusplus.com/
Teoría con ejemplos de los temas explicados de C / C++ en inglés.

http://pseint.sourceforge.net/
Software que permite codificar en lenguaje natural y generación de los gráficos algorítmicos en programación lineal o la forma Nassi – Shneiderman.

TABLAS PARA ARCHIVOS DEL LENGUAJE C

	Funciones Miembro de la clase ios	
1	*stream.bad()*	Devuelve un valor distinto de cero si ha ocurrido un error. Sólo se comprueba el bit de estado ios::badbit, de modo que esta función no equivale a !good().
2	*stream.clear(estado=0)*	Sirve para modificar los bits de estado de un stream, normalmente para eliminar un estado de error. Se suelen usar constantes definidas en el enum io_state definido en ios, usando el operador de bits OR para modificar varios bits a la vez.
3	*filebuf* stream.close()*	Cierra el filebuf y el fichero asociados.
4	*stream.eof()*	Devuelve un valor distinto de cero si se ha alcanzado el fin de fichero. Esta función únicamente comprueba el bit de estado ios::eofbit.
5	*stream.fail()*	Devuelve un valor distinto de cero si una operación sobre el stream ha fallado. Comprueba los bits de estado ios::badbit y ios::failbit.
6	*stream.fill()* *stream.fill(car)*	Cambia el carácter de relleno que se usa cuando la salida es más ancha de la necesaria para el dato actual.
7	*stream.flags()* *stream.flags(long)*	Permite cambiar o leer los flags de manipulación de formato. La primera forma devuelve el valor actual de los flags. La segunda cambia el valor actual por valor, el valor de retorno es el valor previo de los flags.
8	*stream.flush()*	Vacía el buffer asociado al stream. Procesa todas las salidas pendientes.
9	*int stream.gcount()*	Devuelve el número de caracteres sin formato de la última lectura. Las lecturas sin formato son las realizadas mediante las funciones get, getline y read.
10	*int stream.get()* *stream.get(char *,int len,char = '\n')* *stream.get(char &)* *stream.get(streambuf &,char = 'n')*	La primera forma extrae el siguiente carácter o EOF si no hay disponible ninguno. La segunda forma extrae caracteres en la dirección proporcionada en el parámetro char* hasta que se recibe el delimitador del tercer parámetro, el fin de fichero o

		hasta que se leen len-1 bytes. Siempre se añade un carácter nulo de terminación en la cadena de salida. El delimitador no se extrae desde el stream de entrada. La función sólo falla si no se extraen caracteres. La tercera forma extrae un único carácter en la referencia a char proporcionada. La cuarta forma extrae caracteres en el streambuf especificado hasta que se encuentra el delimitador.
11	*stream.getline(char *,int,char = '\n')*	Extrae caracteres hasta que se encuentra el delimitador y los coloca en el buffer, elimina el delimitador del stream de entrada y no lo añade al buffer.
12	*stream.good()*	Devuelve un valor distinto de cero si no ha ocurrido ningún error, es decir, si ninguno de los bits de estado está activo. Aunque pudiera parecerlo, (good significa bueno y bad malo, en inglés) esta función no es exactamente equivalente a !bad(). En realidad es equivalente a rdstate() == 0.
13	*stream.ignore(int n=1,INT delim=EOF)*	Hace que los siguientes n caracteres en el stream de entrada sean ignorados; la extracción se detiene antes si se encuentra el delimitador delim. El delimitador también es extraído del stream.
14	*filebuf* stream.open(nomFis,modoAper)*	Abre un fichero para el objeto especificado. Para el parámetro mode se pueden usar los valores del enum open_mode definidos en la clase ios. open_mode –modo de apertura- **enum** open_mode { in, out, ate, app, trunc, nocreate, noreplace, binary };
15	*int stream.peek()*	Devuelve el siguiente carácter sin extraerlo del stream.
16	*stream.precision()* *stream.precision(car)*	Permite cambiar el número de caracteres significativos que se mostrarán cuando trabajemos con números en coma flotante: float o double. La primera forma devuelve el valor actual de la precisión, la segunda permite modificar la precisión para las siguientes salidas, y también devuelve el valor actual.
17	*stream.put(car)*	Inserta un carácter en el stream de salida.

18	*stream.putback(car)*	Devuelve un carácter al stream.
19	*stream.rdstate()*	Devuelve el estado del stream. Este estado puede ser una combinación de cualquiera de los bits de estado definidos en el enum ios::io_state, es decir ios::badbit, ios::eofbit, ios::failbit e ios::goodbit. El goodbit no es en realidad un bit, sino la ausencia de todos los demás. De modo que para verificar que el valor obtenido por rdstate es ios::goodbit tan sólo hay que comparar. En cualquier caso es mejor usar la función good(). En cuanto a los restantes bits de estado se puede usar el operador & para verificar la presencia de cada uno de los bits. Aunque de nuevo, es preferible usar las funciones bad(), eof() o fail().
20	*stream.read(char *,int)*	Extrae el número indicado de caracteres en el array char*. Se puede usar la función gcount() para saber el número de caracteres extraídos si ocurre algún error.
21	*stream.seekg(pos)* *stream.seekg(offset,dir)*	La primera forma se mueve a posición absoluta, tal como la proporciona la función tellg. La segunda forma se mueve un número offset de bytes la posición del cursor del stream relativa a dir. Este parámetro puede tomar los valores definidos en el enum seek_dir: {beg, cur, end}; Para streams de salida usar ostream::seekp. Usar seekpos o seekoff para moverse en un buffer de un stream.
22	*stream.seekp(dir)* *stream.seekp(dir,donde)*	La primera forma mueve el cursor del stream a una posición absoluta, tal como la devuelve la función tellp. La segunda forma mueve el cursor a una posición relativa desde el punto indicado mediante el parámetro donde -seek_dir-, que puede tomar los valores del enum seek_dir: beg, cur, end.
23	*stream.setf(long)* *stream.setf(valor,máscara)*	Permite modificar los flags de manipulación de formato. La primera forma activa los flags que estén activos tanto en el parámetro y deja sin cambios el resto. La segunda forma activa los flags que estén activos tanto en valor como en

		máscara y desactiva los que estén activos en mask, pero no en valor. Podemos considerar que mask contiene activos los flags que queremos modificar y valor los flags que queremos activar.
		Ambos devuelven el valor previo de los flags.
		Máscara
		enum {
		basefield = dec+oct+hex,
		floatfield = scientific+fixed,
		adjustfield = left+right+internal
		};

24	*stream.tellg()*	Devuelve la posición actual del stream.
25	*stream.tellp()*	Devuelve la posición absoluta del cursor del stream.
26	*stream.unsetf(máscara)*	Permite eliminar flags de manipulación de formato. Desactiva los flags que estén activos en el parámetro.
27	*stream.width()* *stream.width(entero)*	Cambia la anchura en caracteres de la siguiente salida de stream. La primera forma devuelve el valor de la anchura actual, la segunda permite cambiar la anchura para las siguientes salidas, y también devuelve el valor actual de la anchura.
28	*stream.write(const char *, int n)*	Inserta n caracteres (aunque sean nulos) en el stream de salida.

Operaciones con archivos

Operaciones con Archivos	
1	***int remove****(const char * nomFis)* Elimina físicamente un archivo. Deletes the file whose name is specified in *filename*. This is an operation performed directly on a file identified by its *filename*; No streams are involved in the operation. Proper file access shall be available. **Parameters** `filename` C string containing the name of the file to be deleted. Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system). **Return value** If the file is successfully deleted, a zero value is returned.

On failure, a nonzero value is returned.
On most library implementations, the errno variable is also set to a system-specific error code on failure.

Example

```
/* remove example: remove myfile.txt */
#include <stdio.h>

int main ()
{
  if( remove( "myfile.txt" ) != 0 )
    perror( "Error deleting file" );
  else
    puts( "File successfully deleted" );
  return 0;
}
```

If the file `myfile.txt` exists before the execution and the program has write access to it, the file would be deleted and this message would be written to `stdout`:

```
File successfully deleted
```

Otherwise, a message similar to this would be written to `stderr`:

```
Error deleting file: No such file or directory
```

2 ***int rename****(const char * nomVjo, const char * nomNvo)*
Renombra físicamente un archivo.

Changes the name of the file or directory specified by *oldname* to *newname*.

This is an operation performed directly on a file; No streams are involved in the operation.

If *oldname* and *newname* specify different paths and this is supported by the system, the file is moved to the new location.

If *newname* names an existing file, the function may either fail or override the existing file, depending on the specific system and library implementation.

Proper file access shall be available.

Parameters

`oldname`

C string containing the name of an existing file to be renamed and/or moved.
Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).

`newname`

C string containing the new name for the file.
Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).

Return value

If the file is successfully renamed, a zero value is returned.
On failure, a nonzero value is returned.
On most library implementations, the errno variable is also set to a system-specific error code on failure.

Example

```
/* rename example */
#include <stdio.h>

int main ()
{
  int result;
  char oldname[] ="oldname.txt";
  char newname[] ="newname.txt";
  result= rename( oldname , newname );
  if ( result == 0 )
    puts ( "File successfully renamed" );
  else
    perror( "Error renaming file" );
  return 0;
}
```

If the file `oldname.txt` could be successfully renamed to `newname.txt` the following message would be written to `stdout`:

```
File successfully renamed
```

Otherwise, a message similar to this will be written to `stderr`:

```
Error renaming file: Permission denied
```

3 ***int fclose(FILE * stream)***

Cierra un archivo y quita la asociación. Stream es un puntero a FILE. Retorna cero si el cierre fue correcto, sino EOF.

Closes the file associated with the *stream* and disassociates it.

All internal buffers associated with the stream are disassociated from it and flushed: the content of any unwritten output buffer is written and the content of any unread input buffer is discarded.

Even if the call fails, the stream passed as parameter will no longer be associated with the file nor its buffers.

Parameters

`stream`

Pointer to a FILE object that specifies the stream to be closed.

Return Value

If the stream is successfully closed, a zero value is returned. On failure, EOF is returned.

```
/* fclose example */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  pFile = fopen ("myfile.txt","wt");
  fprintf (pFile, "fclose example");
  fclose (pFile);
  return 0;
}
```

4 ***fcloseall()***

Cierra todos los archivos abiertos por el usuario. Función propia de Borland.

5 ***int fflush (FILE * stream)***

Vacía un stream.

If the given *stream* was open for writing (or if it was open for updating and the last i/o operation was an output operation) any unwritten data in its output buffer is written to the file.

If *stream* is a null pointer, all such streams are flushed.

In all other cases, the behavior depends on the specific library implementation. In some implementations, flushing a stream open for reading causes its input buffer to be cleared (but this is not portable expected behavior).

The stream remains open after this call.

When a file is closed, either because of a call to fclose or because the program terminates, all the buffers associated with it are automatically flushed.

Parameters

`stream`

Pointer to a FILE object that specifies a buffered stream.

Return Value

A zero value indicates success.
If an error occurs, EOF is returned and the error indicator is set (see ferror).

Example

In files open for update (i.e., open for both reading and writing), the stream shall be flushed after an output operation before performing an input operation. This can be done either by repositioning (fseek, fsetpos, rewind) or by calling explicitly `fflush`, like in this example:

```
/* fflush example */
#include <stdio.h>
char mybuffer[80];
int main()
{
   FILE * pFile;
   pFile = fopen ("example.txt","r+");
   if (pFile == NULL) perror ("Error opening file");
   else {
     fputs ("test",pFile);
     fflush (pFile);    // flushing or repositioning
                                          required
     fgets (mybuffer,80,pFile);
     puts (mybuffer);
     fclose (pFile);
     return 0;
  }
}
```

6

FILE *fopen(const char * nomArc, const char * modoAper)

Abre un archivo. Retorna un puntero, sino NULL si el archivo no pudo ser abierto.

Opens the file whose name is specified in the parameter *filename* and associates it with a stream that can be identified in future operations by the FILE pointer returned.

The operations that are allowed on the stream and how these are performed are defined by the *mode* parameter.

The returned stream is *fully buffered* by default if it is known to not refer to an interactive device (see setbuf).

The returned pointer can be disassociated from the file by calling fclose or freopen. All opened files are automatically closed on normal program termination.

The running environment supports at least FOPEN_MAX files open simultaneously.

Parameters

`filename`

C string containing the name of the file to be opened.
Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).

`mode`

C string containing a file access mode. It can be:

"r"	**read:** Open file for input operations. The file must exist.
"w"	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist.

"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
"a+"	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist.

With the *mode* specifiers above the file is open as a *text file*. In order to open a file as a *binary file*, a "b" character has to be included in the *mode* string. This additional "b" character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the "+" sign for the mixed modes ("rb+", "wb+", "ab+").

The new C standard (C2011, which is not part of C++) adds a new standard subspecifier ("x"), that can be appended to any "w" specifier (to form "wx", "wbx", "w+x" or "w+bx"/"wb+x"). This subspecifier forces the function to fail if the file exists, instead of overwriting it.
If additional characters follow the sequence, the behavior depends on the library implementation: some implementations may ignore additional characters so that for example an additional "t" (sometimes used to explicitly state a *text file*) is accepted.
On some library implementations, opening or creating a text file with update mode may treat the stream instead as a binary file.
Text files are files containing sequences of lines of text. Depending on the environment where the application runs, some special character conversion may occur in input/output operations in *text mode* to adapt them to a system-specific text file format. Although on some environments no conversions occur and both *text files* and *binary files* are treated the same way, using the appropriate mode improves portability.
For files open for update (those which include a "+" sign), on which both input and output operations are allowed, the stream shall be flushed (fflush) or repositioned (fseek, fsetpos, rewind) before a reading operation that follows a writing operation. The stream shall be repositioned (fseek, fsetpos, rewind) before a writing operation that follows a reading operation (whenever that operation did not reach the end-of-file).

Return Value

If the file is successfully opened, the function returns a pointer to a FILE object that can be used to identify the stream on future operations.
Otherwise, a null pointer is returned.
On most library implementations, the errno variable is also set to a system-specific error code on failure.

Example

```
/* fopen example */
#include <stdio.h>
int main ()
{
```

```
  FILE * pFile;
  pFile = fopen ("myfile.txt","w");
  if (pFile!=NULL)
  {
    fputs ("fopen example",pFile);
    fclose (pFile);
  }
  return 0;
}
```

7 ***FILE *freopen(const char * filename, const char * mode, FILE * stream)***

Reabre un stream con diferente archivo o modo.

Reuses *stream* to either open the file specified by *filename* or to change its access *mode*.

If a new *filename* is specified, the function first attempts to close any file already associated with *stream* (third parameter) and disassociates it. Then, independently of whether that stream was successfuly closed or not, `freopen`opens the file specified by *filename* and associates it with the *stream* just as fopen would do using the specified *mode*.

If *filename* is a null pointer, the function attempts to change the *mode* of the stream. Although a particular library implementation is allowed to restrict the changes permitted, and under which circumstances.

The *error indicator* and *eof indicator* are automatically cleared (as if clearerr was called).

This function is especially useful for redirecting predefined streams like `stdin`, `stdout` and `stderr` to specific files (see the example below).

Parameters

`filename`

C string containing the name of the file to be opened.
Its value shall follow the file name specifications of the running environment and can include a path (if supported by the system).
If this parameter is a null pointer, the function attempts to change the mode of the *stream*, as if the file name currently associated with that stream had been used.

`mode`

C string containing a file access mode. It can be:

"r"	**read:** Open file for input operations. The file must exist.
"w"	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist.
"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
"a+"	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but

	output operations move the position back to the end of file. The file is created if it does not exist.

With the *mode* specifiers above the file is open as a *text file*. In order to open a file as a *binary file*, a `"b"` character has to be included in the *mode* string. This additional `"b"` character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the `"+"` sign for the mixed modes ("rb+", "wb+", "ab+").

The new C standard (C2011, which is not part of C++) adds a new standard subspecifier (`"x"`), that can be appended to any `"w"` specifier (to form `"wx"`, `"wbx"`, `"w+x"` or `"w+bx"/"wb+x"`). This subspecifier forces the function to fail if the file exists, instead of overwriting it.
If additional characters follow the sequence, the behavior depends on the library implementation: some implementations may ignore additional characters so that for example an additional `"t"` (sometimes used to explicitly state a *text file*) is accepted.
On some library implementations, opening or creating a text file with update mode may treat the stream instead as a binary file.

`stream`

pointer to a FILE object that identifies the stream to be reopened.

Return value

If the file is successfully reopened, the function returns the pointer passed as parameter *stream*, which can be used to identify the reopened stream.
Otherwise, a null pointer is returned.
On most library implementations, the errno variable is also set to a system-specific error code on failure.

Example

```
/* freopen example: redirecting stdout */
#include <stdio.h>

int main ()
{
  freopen ("myfile.txt","w",stdout);
  printf ("This sentence is redirected to a file.");
  fclose (stdout);
  return 0;
}
```

This sample code redirects the output that would normally go to the standard output to a file called `myfile.txt`, that after this program is executed contains:

```
This sentence is redirected to a file.
```

8

int fprintf (FILE * stream, const char * formato, ...)

Graba datos formateados al stream.

Writes the C string pointed by *format* to the *stream*. If *format* includes *format specifiers* (subsequences beginning with `%`), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

After the *format* parameter, the function expects at least as many additional arguments as specified by *format*.

Parameters

`stream`

Pointer to a FILE object that identifies an output stream.

`format`

C string that contains the text to be written to the stream.

It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype:

`%[flags][width][.precision][length]specifier`

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding

specifier	**Output**	**Example**
`d` *or* `i`	Signed decimal integer	392
`u`	Unsigned decimal integer	7235
`o`	Unsigned octal	610
`x`	Unsigned hexadecimal integer	7fa
`X`	Unsigned hexadecimal integer (uppercase)	7FA
`f`	Decimal floating point, lowercase	392.65
`F`	Decimal floating point, uppercase	392.65
`e`	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
`E`	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
`g`	Use the shortest representation: `%e` or `%f`	392.65
`G`	Use the shortest representation: `%E` or `%F`	392.65
`a`	Hexadecimal floating point, lowercase	-0xc.90fep-2
`A`	Hexadecimal floating point, uppercase	-0XC.90FEP-2
`c`	Character	a
`s`	String of characters	sample
`p`	Pointer address	b8000000
`n`	Nothing printed. The corresponding argument must be a pointer to a `signed int`. The number of characters written so far is stored in the pointed location.	
`%`	A `%` followed by another `%` character will write a single `%` to the stream.	`%`

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

flags	**description**
–	Left-justify within the given field width; Right justification is the default (see *width* sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or –) even for positive numbers. By default, only negative numbers are preceded with a – sign.

(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with `o`, `x` or `X` specifiers the value is preceeded with `0`, `0x` or `0X` respectively for values different than zero. Used with `a`, `A`, `e`, `E`, `f`, `F`, `g` or `G` it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (`0`) instead of spaces when padding is specified (see *width* sub-specifier).

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The *width* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	**description**
.number	For integer specifiers (`d`, `i`, `o`, `u`, `x`, `X`): *precision* specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A *precision* of `0` means that no character is written for the value `0`. For `a`, `A`, `e`, `E`, `f` and `F` specifiers: this is the number of digits to be printed **after** the decimal point (by default, this is 6). For `g` and `G` specifiers: This is the maximum number of significant digits to be printed. For `s`: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for *precision*, `0` is assumed.
.*	The *precision* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

The *length* sub-specifier modifies the length of the data type. This is a chart showing the types used to interpret the corresponding arguments with and without *length* specifier (if a different type is used, the proper type promotion or conversion is performed, if allowed):

	specifiers						
length	**d i**	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*

ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Note that the `c` specifier takes an `int` (or wint_t) as argument, but performs the proper conversion to a `char` value (or a `wchar_t`) before formatting it for output.

Note: Yellow rows indicate specifiers and sub-specifiers introduced by C99. See `<cinttypes>` for the specifiers for extended types.

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for `n`).
There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.

Return Value

On success, the total number of characters written is returned.

If a writing error occurs, the *error indicator* (ferror) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, errno is set to `EILSEQ` and a negative number is returned.

Example

Edit & Run

```
/* fprintf example */
#include <stdio.h>

int main ()
{
   FILE * pFile;
   int n;
   char name [100];

   pFile = fopen ("myfile.txt","w");
   for (n=0 ; n<3 ; n++)
   {
     puts ("please, enter a name: ");
     gets (name);
     fprintf (pFile, "Name %d [%-10.10s]\n",n+1,name);
   }
   fclose (pFile);

   return 0;
```

```
}
```

This example prompts 3 times the user for a name and then writes them to `myfile.txt` each one in a line with a fixed length (a total of 19 characters + newline).
Two format tags are used:
`%d` : Signed decimal integer
`%-10.10s` : left-justified (`-`), minimum of ten characters (`10`), maximum of ten characters (`.10`), string (`s`).
Assuming that we have entered `John`, `Jean-Francois` and `Yoko` as the 3 names, `myfile.txt` would contain:

```
Name 1 [John      ]
Name 2 [Jean-Franc]
Name 3 [Yoko      ]
```

9 ***int fscanf(FILE * stream, const char * formato, ...)***
Lee datos formateados desde el stream.

```
int fscanf ( FILE * stream, const char * format, ... );
```

Read formatted data from stream
Reads data from the *stream* and stores them according to the parameter *format* into the locations pointed by the additional arguments.

The additional arguments should point to already allocated objects of the type specified by their corresponding format specifier within the *format* string.

Parameters

`stream`
Pointer to a FILE object that identifies the input stream to read data from.

`format`
C string that contains a sequence of characters that control how characters extracted from the stream are treated:

- **Whitespace character:** the function will read and ignore any whitespace characters encountered before the next non-whitespace character (whitespace characters include spaces, newline and tab characters -- seeisspace). A single whitespace in the *format* string validates any quantity of whitespace characters extracted from the *stream* (including none).
- **Non-whitespace character, except format specifier (%):** Any character that is not either a whitespace character (blank, newline or tab) or part of a *format specifier* (which begin with a `%` character) causes the function to read the next character from the stream, compare it to this non-whitespace character and if it matches, it is discarded and the function continues with the next character of *format*. If the character does not match, the function fails, returning and leaving subsequent characters of the stream unread.
- **Format specifiers:** A sequence formed by an initial percentage sign (`%`) indicates a format specifier, which is used to specify the

type and format of the data to be retrieved from the *stream* and stored into the locations pointed by the additional arguments.

A *format specifier* for `fscanf` follows this prototype:

```
%[*][width][length]specifier
```

Where the *specifier* character at the end is the most significant component, since it defines which characters are extracted, their interpretation and the type of its corresponding argument:

specifier	**Description**	**Characters extracted**
`i`, `u`	Integer	Any number of digits, optionally preceded by a sign (`+` or `-`). Decimal digits assumed by default (`0-9`), but a `0` prefix introduces octal digits (`0-7`), and`0x` hexadecimal digits (`0-f`).
`d`	Decimal integer	Any number of decimal digits (`0-9`), optionally preceded by a sign (`+` or `-`).
`o`	Octal integer	Any number of octal digits (`0-7`), optionally preceded by a sign (`+` or `-`).
`x`	Hexadecimal integer	Any number of hexadecimal digits (`0-9`, `a-f`, `A-F`), optionally preceded by `0x` or `0X`, and all optionally preceded by a sign (`+` or `-`).
`f`, `e`, `g` `a`	Floating point number	A series of decimal digits, optionally containing a decimal point, optionally preceeded by a sign (`+` or `-`) and optionally followed by the `e` or `E` character and a decimal integer (or some of the other sequences supported by `strtod`). Implementations complying with C99 also support hexadecimal floating-point format when preceded by `0x` or `0X`.
`c`	Character	The next character. If a *width* other than `1` is specified, the function reads exactly*width* characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.
`s`	String of characters	Any number of non-whitespace characters, stopping at the first whitespace character found. A terminating null character is automatically added at the end of the stored sequence.
`p`	Pointer address	A sequence of characters representing a pointer. The particular format used depends on the system and library implementation, but it is the same as the one used to format `%p` in fprintf.
`[`*characters*`]`	Scanset	Any number of the characters specified between the brackets. A dash (`-`) that is not the first character may produce non-portable behavior in some library implementations.
`[^`*characters*`]`	Negated scanset	Any number of characters none of them specified as *characters* between the brackets.
`n`	Count	No input is consumed.

		The number of characters read so far from *stream* is stored in the pointed location.
%	%	A % followed by another % matches a single %.

Except for n, at least one character shall be consumed by any specifier. Otherwise the match fails, and the scan ends there.

The *format specifier* can also contain sub-specifiers: *asterisk* (*), *width* and *length* (in that order), which are optional and follow these specifications:

sub-specifier	**description**
*	An optional starting asterisk indicates that the data is to be read from the stream but ignored (i.e. it is not stored in the location pointed by an argument).
width	Specifies the maximum number of characters to be read in the current reading operation (optional).
length	One of hh, h, l, ll, j, z, t, L (optional). This alters the expected type of the storage pointed by the corresponding argument (see below).

This is a chart showing the types expected for the corresponding arguments where input is stored (both with and without a *length* sub-specifier):

	specifiers					
length	**d i**	**u o x**	**f e g a**	**c s [] [^]**	**p**	**n**
(none)	int*	unsigned int*	float*	char*	void**	int*
hh	signed char*	unsigned char*				signed char*
h	short int*	unsigned short int*				short int*
l	long int*	unsigned long int*	double*	wchar_t*		long int*
ll	long long int*	unsigned long long int*				long long int*
j	intmax_t*	uintmax_t*				intmax_t*
z	size_t*	size_t*				size_t*
t	ptrdiff_t*	ptrdiff_t*				ptrdiff_t*
L			long double*			

Note: Yellow rows indicate specifiers and sub-specifiers introduced by C99.

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a pointer to allocated storage where the interpretation of the extracted characters is stored with the appropriate type.
There should be at least as many of these arguments as the number of values stored by the *format specifiers*. Additional arguments are ignored by the function.
These arguments are expected to be pointers: to store the result of

a `fscanf` operation on a regular variable, its name should be preceded by the *reference operator* (`&`) (see example).

Return Value

On success, the function returns the number of items of the argument list successfully filled. This count can match the expected number of items or be less (even zero) due to a matching failure, a reading error, or the reach of the *end-of-file*.

If a reading error happens or the *end-of-file* is reached while reading, the proper indicator is set (feof or ferror). And, if either happens before any data could be successfully read, EOF is returned.

If an encoding error happens interpreting wide characters, the function sets errno to `EILSEQ`.

```
/* fscanf example */
#include <stdio.h>

int main () {
  char str [80];
  float f;
  FILE * pFile;

  pFile = fopen ("myfile.txt","w+");
  fprintf (pFile, "%f %s", 3.1416, "PI");
  rewind (pFile);
  fscanf (pFile, "%f", &f);
  fscanf (pFile, "%s", str);
  fclose (pFile);
  printf ("I have read: %f and %s \n",f,str);
  return 0;
}
```

10 ***int printf(const char * formato, ...)***

Graba datos formateados al stdout.

Writes the C string pointed by *format* to the standard output (stdout).
If *format* includes *format specifiers* (subsequences beginning with `%`), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

Parameters

`format`

C string that contains the text to be written to stdout.
It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype: [see compatibility note below]
`%[flags][width][.precision][length]specifier`

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	**Output**	**Example**
`d` *or* `i`	Signed decimal integer	`392`
`u`	Unsigned decimal integer	`7235`
`o`	Unsigned octal	`610`
`x`	Unsigned hexadecimal integer	`7fa`
`X`	Unsigned hexadecimal integer (uppercase)	`7FA`
`f`	Decimal floating point, lowercase	`392.65`
`F`	Decimal floating point, uppercase	`392.65`
`e`	Scientific notation (mantissa/exponent), lowercase	`3.9265e+2`
`E`	Scientific notation (mantissa/exponent), uppercase	`3.9265E+2`
`g`	Use the shortest representation: `%e` or `%f`	`392.65`
`G`	Use the shortest representation: `%E` or `%F`	`392.65`
`a`	Hexadecimal floating point, lowercase	`-0xc.90fep-2`
`A`	Hexadecimal floating point, uppercase	`-0XC.90FEP-2`
`c`	Character	`a`
`s`	String of characters	`sample`
`p`	Pointer address	`b8000000`
`n`	Nothing printed. The corresponding argument must be a pointer to a `signed int`. The number of characters written so far is stored in the pointed location.	
`%`	A `%` followed by another `%` character will write a single `%` to the stream.	`%`

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

flags	**description**
`-`	Left-justify within the given field width; Right justification is the default (see *width* sub-specifier).
`+`	Forces to preceed the result with a plus or minus sign (`+` or `-`) even for positive numbers. By default, only negative numbers are preceded with a `-` sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
`#`	Used with `o`, `x` or `X` specifiers the value is preceeded with `0`, `0x` or `0X` respectively for values different than zero. Used with `a`, `A`, `e`, `E`, `f`, `F`, `g` or `G` it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.

0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see *width* sub-specifier).

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The *width* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	**description**
.number	For integer specifiers (d, i, o, u, x, X): *precision* specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A *precision* of 0 means that no character is written for the value 0. For a, A, e, E, f and F specifiers: this is the number of digits to be printed **after** the decimal point (by default, this is 6). For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for *precision*, 0 is assumed.
.*	The *precision* is not specified in the *format* string, but as an additional integer value argument preceding the argument that has to be formatted.

The *length* sub-specifier modifies the length of the data type. This is a chart showing the types used to interpret the corresponding arguments with and without *length* specifier (if a different type is used, the proper type promotion or conversion is performed, if allowed):

	specifiers						
length	**d i**	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Note regarding the `c` specifier: it takes an `int` (or wint_t) as argument, but performs the proper conversion to a `char` value (or a `wchar_t`) before formatting it for output.

Note: Yellow rows indicate specifiers and sub-specifiers introduced by C99. See `<cinttypes>` for the specifiers for extended types.

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for `n`).
There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.

On success, the total number of characters written is returned.

Return value

If a writing error occurs, the *error indicator* (ferror) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, errno is set to `EILSEQ` and a negative number is returned.

Example

```
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100,
100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416,
3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Salida:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
```

```
Width trick:    10
A string
```

11 ***int scanf(const char * formato, ...)***

Lee datos formateados desde el stdin.

Reads data from stdin and stores them according to the parameter *format* into the locations pointed by the additional arguments.

The additional arguments should point to already allocated objects of the type specified by their corresponding format specifier within the *format* string.

Parameters

`format`

C string that contains a sequence of characters that control how characters extracted from the stream are treated:

- **Whitespace character:** the function will read and ignore any whitespace characters encountered before the next non-whitespace character (whitespace characters include spaces, newline and tab characters – see isspace). A single whitespace in the *format* string validates any quantity of whitespace characters extracted from the *stream* (including none).
- **Non-whitespace character, except format specifier (%):** Any character that is not either a whitespace character (blank, newline or tab) or part of a *format specifier* (which begin with a `%` character) causes the function to read the next character from the stream, compare it to this non-whitespace character and if it matches, it is discarded and the function continues with the next character of *format*. If the character does not match, the function fails, returning and leaving subsequent characters of the stream unread.
- **Format specifiers:** A sequence formed by an initial percentage sign (`%`) indicates a format specifier, which is used to specify the type and format of the data to be retrieved from the *stream* and stored into the locations pointed by the additional arguments.

A *format specifier* for `scanf` follows this prototype:

`%[*][width][length]specifier`

Where the *specifier* character at the end is the most significant component, since it defines which characters are extracted, their interpretation and the type of its corresponding argument:

specifier	**Description**	**Characters extracted**
`i`	Integer	Any number of digits, optionally preceded by a sign (`+` or `-`). Decimal digits assumed by default (`0-9`), but a `0` prefix introduces octal digits (`0-7`), and`0x` hexadecimal digits (`0-f`). *Signed* argument.

d *or* u	Decimal integer	Any number of decimal digits (0-9), optionally preceded by a sign (+ or -). d is for a *signed* argument, and u for an *unsigned*.
o	Octal integer	Any number of octal digits (0-7), optionally preceded by a sign (+ or -). *Unsigned* argument.
x	Hexadecimal integer	Any number of hexadecimal digits (0-9, a-f, A-F), optionally preceded by 0x or 0X, and all optionally preceded by a sign (+ or -). *Unsigned* argument.
f, e, g a	Floating point number	A series of decimal digits, optionally containing a decimal point, optionally preceeded by a sign (+ or -) and optionally followed by the e or E character and a decimal integer (or some of the other sequences supported by strtod). Implementations complying with C99 also support hexadecimal floating-point format when preceded by 0x or 0X.
c	Character	The next character. If a *width* other than 1 is specified, the function reads exactly*width* characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.
s	String of characters	Any number of non-whitespace characters, stopping at the first whitespace character found. A terminating null character is automatically added at the end of the stored sequence.
p	Pointer address	A sequence of characters representing a pointer. The particular format used depends on the system and library implementation, but it is the same as the one used to format %p in fprintf.
[*characters*]	Scanset	Any number of the characters specified between the brackets. A dash (-) that is not the first character may produce non-portable behavior in some library implementations.
[^*characters*]	Negated scanset	Any number of characters none of them specified as *characters* between the brackets.
n	Count	No input is consumed. The number of characters read so far from stdin is stored in the pointed location.
%	%	A % followed by another % matches a single %.

Except for n, at least one character shall be consumed by any specifier. Otherwise the match fails, and the scan ends there.

The *format specifier* can also contain sub-specifiers: *asterisk* (*), *width* and *length* (in that order), which are optional and follow these specifications:

sub-specifier	**description**
*	An optional starting asterisk indicates that the data is to be read from the stream but ignored (i.e. it is not stored in the location pointed by an argument).
width	Specifies the maximum number of characters to be read in the

	current reading operation (optional).
length	One of `hh`, `h`, `l`, `ll`, `j`, `z`, `t`, `L` (optional). This alters the expected type of the storage pointed by the corresponding argument (see below).

This is a chart showing the types expected for the corresponding arguments where input is stored (both with and without a *length* sub-specifier):

	specifiers					
length	**d i**	**u o x**	**f e g a**	**c s [] [^]**	**p**	**n**
(none)	int*	unsigned int*	float*	char*	void**	int*
hh	signed char*	unsigned char*				signed char*
h	short int*	unsigned short int*				short int*
l	long int*	unsigned long int*	double*	wchar_t*		long int*
ll	long long int*	unsigned long long int*				long long int*
j	intmax_t*	uintmax_t*				intmax_t*
z	size_t*	size_t*				size_t*
t	ptrdiff_t*	ptrdiff_t*				ptrdiff_t*
L			long double*			

Note: Yellow rows indicate specifiers and sub-specifiers introduced by C99.

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a pointer to allocated storage where the interpretation of the extracted characters is stored with the appropriate type.
There should be at least as many of these arguments as the number of values stored by the *format specifiers*. Additional arguments are ignored by the function.
These arguments are expected to be pointers: to store the result of a `scanf` operation on a regular variable, its name should be preceded by the *reference operator* (`&`) (see example).

Return Value

On success, the function returns the number of items of the argument list successfully filled. This count can match the expected number of items or be less (even zero) due to a matching failure, a reading error, or the reach of the *end-of-file*.

If a reading error happens or the *end-of-file* is reached while reading, the proper indicator is set (feof or ferror). And, if either happens before any data could be successfully read, EOF is returned.

If an encoding error happens interpreting wide characters, the function

sets errno to EILSEQ.

Example

```
/* scanf example */
#include <stdio.h>

int main ()
{
  char str [80];
  int i;

  printf ("Enter your family name: ");
  scanf ("%79s",str);
  printf ("Enter your age: ");
  scanf ("%d",&i);
  printf ("Mr. %s , %d years old.\n",str,i);
  printf ("Enter a hexadecimal number: ");
  scanf ("%x",&i);
  printf ("You have entered %#x (%d).\n",i,i);

  return 0;
}
```

This example demonstrates some of the types that can be read with scanf:

```
Enter your family name: Soulie
Enter your age: 29
Mr. Soulie , 29 years old.
Enter a hexadecimal number: ff
You have entered 0xff (255).
```

12 ***int sprintf (char * str, const char * formato, ...)***

Graba datos formateados a string.

Composes a string with the same text that would be printed if *format* was used on printf, but instead of being printed, the content is stored as a *C string* in the buffer pointed by *str*.

The size of the buffer should be large enough to contain the entire resulting string (see snprintf for a safer version).

A terminating null character is automatically appended after the content.

After the *format* parameter, the function expects at least as many additional arguments as needed for *format*.

Parameters

`str`

Pointer to a buffer where the resulting C-string is stored.
The buffer should be large enough to contain the resulting string.

`format`

C string that contains a format string that follows the same specifications as *format* in printf (see printf for details).

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for `n`).
There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.

Return Value

On success, the total number of characters written is returned. This count does not include the additional null-character automatically appended at the end of the string.
On failure, a negative number is returned.

Example

```
/* sprintf example */
#include <stdio.h>

int main ()
{
  char buffer [50];
  int n, a=5, b=3;
  n = sprintf (buffer, "%d plus %d is %d", a, b, a+b);
  printf ("[%s] is a string %d chars long\n",buffer,n);
  return 0;
}
```

Salida:

```
[5 plus 3 is 8] is a string 13 chars long
```

13 ***int sscanf (const char * s, const char * formato, ...)***

Lee datos formateados desde una cadena.

Reads data from *s* and stores them according to parameter *format* into the locations given by the additional arguments, as if scanf was used, but reading from *s* instead of the standard input (stdin).

The additional arguments should point to already allocated objects of the type specified by their corresponding format specifier within the *format* string.

Parameters

`s`

C string that the function processes as its source to retrieve the data.

`format`

C string that contains a format string that follows the same specifications as *format* in scanf (see scanf for details).

`... (additional arguments)`

Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a pointer to allocated storage where the interpretation of the extracted characters is stored with the appropriate

type.
There should be at least as many of these arguments as the number of values stored by the *format specifiers*. Additional arguments are ignored by the function.

Return Value

On success, the function returns the number of items in the argument list successfully filled. This count can match the expected number of items or be less (even zero) in the case of a matching failure.
In the case of an input failure before any data could be successfully interpreted, EOF is returned.

Example

```
/* sscanf example */
#include <stdio.h>

int main ()
{
  char sentence []="Rudolph is 12 years old";
  char str [20];
  int i;

  sscanf (sentence,"%s %*s %d",str,&i);
  printf ("%s -> %d\n",str,i);

  return 0;
}
```

Salida:

```
Rudolph -> 12
```

14 ***int vfprintf (FILE * stream, const char * formato, va_list arg)***

Graba datos formateados desde el stream en una lista de argumentos variable.

Writes the C string pointed by *format* to the *stream*, replacing any *format specifier* in the same way as printf does, but using the elements in the variable argument list identified by *arg* instead of additional function arguments.

Internally, the function retrieves arguments from the list identified by *arg* as if va_arg was used on it, and thus the state of *arg* is likely altered by the call.

In any case, *arg* should have been initialized by va_start at some point before the call, and it is expected to be released byva_end at some point after the call.

Parameters

`stream`

Pointer to a FILE object that identifies an output stream.

`format`

C string that contains a format string that follows the same specifications as *format* in printf (see printf for details).

`arg`

A value identifying a variable arguments list initialized with va_start. va_list is a special type defined in <cstdarg>.

Return Value

On success, the total number of characters written is returned.

If a writing error occurs, the *error indicator* (ferror) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, errno is set to `EILSEQ` and a negative number is returned.

Example

```
/* vfprintf example */
#include <stdio.h>
#include <stdarg.h>

void WriteFormatted (FILE * stream, const char * format, ...)
{
  va_list args;
  va_start (args, format);
  vfprintf (stream, format, args);
  va_end (args);
}

int main ()
{
   FILE * pFile;

   pFile = fopen ("myfile.txt","w");

   WriteFormatted (pFile,"Call with %d variable
argument.\n",1);
   WriteFormatted (pFile,"Call with %d variable
%s.\n",2,"arguments");

   fclose (pFile);

   return 0;
}
```

The example demonstrates how the `WriteFormatted` can be called with a different number of arguments, which are on their turn passed to the `vfprintf` function.
`myfile.txt` would contain:

myfile.txt
`Call with 1 variable argument.` `Call with 2 variable arguments.`

15 ***int vprintf (const char * formato, va_list arg)***

Graba datos formateados desde una lista de argumentos variable a stdout.

Writes the C string pointed by *format* to the standard output (stdout), replacing

any *format specifier* in the same way asprintf does, but using the elements in the variable argument list identified by *arg* instead of additional function arguments.

Internally, the function retrieves arguments from the list identified by *arg* as if va_arg was used on it, and thus the state of *arg* is likely altered by the call.

In any case, *arg* should have been initialized by va_start at some point before the call, and it is expected to be released byva_end at some point after the call.

Parameters

`format`

C string that contains a format string that follows the same specifications as *format* in printf (see printf for details).

`arg`

A value identifying a variable arguments list initialized with va_start. va_list is a special type defined in <cstdarg>.

Return Value

On success, the total number of characters written is returned.

If a writing error occurs, the *error indicator* (ferror) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, errno is set to `EILSEQ` and a negative number is returned.

Example

```
/* vprintf example */
#include <stdio.h>
#include <stdarg.h>

void WriteFormatted ( const char * format, ... )
{
  va_list args;
  va_start (args, format);
  vprintf (format, args);
  va_end (args);
}

int main ()
{
   WriteFormatted ("Call with %d variable argument.\n",1);
   WriteFormatted ("Call with %d variable
%s.\n",2,"arguments");

   return 0;
}
```

The example illustrates how the `WriteFormatted` can be called with a different number of arguments, which are on their turn passed to the `vprintf` function,

showing the following output:

```
Call with 1 variable argument.
Call with 2 variable arguments.
```

16 ***int vsprintf (char * s, const char * formato, va_list arg)***

Graba datos formateados de una lista de argumentos variable de cadena.

Composes a string with the same text that would be printed if *format* was used on printf, but using the elements in the variable argument list identified by *arg* instead of additional function arguments and storing the resulting content as a *C string* in the buffer pointed by *s*.

Internally, the function retrieves arguments from the list identified by *arg* as if va_arg was used on it, and thus the state of *arg* is likely to be altered by the call.

In any case, *arg* should have been initialized by va_start at some point before the call, and it is expected to be released byva_end at some point after the call.

Parameters

`s`

Pointer to a buffer where the resulting C-string is stored.
The buffer should be large enough to contain the resulting string.

`format`

C string that contains a format string that follows the same specifications as *format* in printf (see printf for details).

`arg`

A value identifying a variable arguments list initialized with va_start.
va_list is a special type defined in <cstdarg>.

Return Value

On success, the total number of characters written is returned.
On failure, a negative number is returned.

Example

```
1  /* vsprintf example */
2  #include <stdio.h>
3  #include <stdarg.h>
4
5  void PrintFError ( const char * format, ... )
6  {
7    char buffer[256];
8    va_list args;
9    va_start (args, format);
10   vsprintf (buffer,format, args);
11   perror (buffer);
12   va_end (args);
13 }
14
15 int main ()
16 {
17   FILE * pFile;
18   char szFileName[]="myfile.txt";
```

```

  pFile = fopen (szFileName,"r");
  if (pFile == NULL)
    PrintFError ("Error opening '%s'",szFileName);
  else
  {
    // file successfully open
    fclose (pFile);
  }
  return 0;
}
```

In this example, if the file `myfile.txt` does not exist, perror is called to show an error message similar to:

```
Error opening file 'myfile.txt': No such file or directory
```

17 ***int fgetc (FILE * stream)***

Obtiene carácter desde el stream.

Returns the character currently pointed by the internal file position indicator of the specified *stream*. The internal file position indicator is then advanced to the next character.

If the stream is at the end-of-file when called, the function returns EOF and sets the *end-of-file indicator* for the stream (feof).

If a read error occurs, the function returns EOF and sets the *error indicator* for the stream (ferror).

`fgetc` and getc are equivalent, except that getc may be implemented as a macro in some libraries.

Parameters

`stream`

Pointer to a FILE object that identifies an input stream.

Return Value

On success, the character read is returned (promoted to an `int` value).
The return type is `int` to accommodate for the special value EOF, which indicates failure:
If the position indicator was at the *end-of-file*, the function returns EOF and sets the *eof indicator* (feof) of *stream*.
If some other reading error happens, the function also returns EOF, but sets its *error indicator* (ferror) instead.

Example

```
/* fgetc example: money counter */
#include <stdio.h>
int main () {
  FILE * pFile;
```

```
  int c;
  int n = 0;

  pFile = fopen ("myfile.txt","r");
  if (pFile==NULL) perror ("Error opening file");
  else
  {
    do {
      c = fgetc (pFile);
      if (c == '$') n++;
    } while (c != EOF);
    fclose (pFile);
    printf ("The file contains %d dollar sign characters
($).\n",n);
  }
  return 0;
}
```

This program reads an existing file called `myfile.txt` character by character and uses the `n` variable to count how many dollar characters (`$`) the file contains.

18 ***char * fgets (char * str, int num, FILE * stream)***

Obtiene una cadena desde el stream.

Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

A newline character makes `fgets` stop reading, but it is considered a valid character by the function and included in the string copied to *str*.

A terminating null character is automatically appended after the characters copied to *str*.

Notice that `fgets` is quite different from gets: not only `fgets` accepts a *stream* argument, but also allows to specify the maximum size of *str* and includes in the string any ending newline character.

Parameters

`str`

Pointer to an array of `char`s where the string read is copied.

`num`

Maximum number of characters to be copied into *str* (including the terminating null-character).

`stream`

Pointer to a FILE object that identifies an input stream.
stdin can be used as argument to read from the *standard input*.

Return Value

On success, the function returns *str*.
If the *end-of-file* is encountered while attempting to read a character, the *eof*

indicator is set (feof). If this happens before any characters could be read, the pointer returned is a null pointer (and the contents of *str* remain unchanged).
If a read error occurs, the *error indicator* (ferror) is set and a null pointer is also returned (but the contents pointed by *str*may have changed).

Example

```
/* fgets example */
#include <stdio.h>

int main()
{
   FILE * pFile;
   char mystring [100];

   pFile = fopen ("myfile.txt" , "r");
   if (pFile == NULL) perror ("Error opening file");
   else {
     if ( fgets (mystring , 100 , pFile) != NULL )
       puts (mystring);
     fclose (pFile);
   }
   return 0;
}
```

This example reads the first line of `myfile.txt` or the first 99 characters, whichever comes first, and prints them on the screen.

19 ***int fputc (int car, FILE * stream)***
Graba un carácter al stream.

Writes a *character* to the *stream* and advances the position indicator.

The character is written at the position indicated by the *internal position indicator* of the *stream*, which is then automatically advanced by one.

Parameters

`character`
The `int` promotion of the character to be written.
The value is internally converted to an `unsigned char` when written.

`stream`
Pointer to a FILE object that identifies an output stream.

Return Value

On success, the *character* written is returned.
If a writing error occurs, EOF is returned and the *error indicator* (ferror) is set.

Example

```
/* fputc example: alphabet writer */
#include <stdio.h>

int main ()
```

```
{
  FILE * pFile;
  char c;

  pFile = fopen ("alphabet.txt","w");
  if (pFile!=NULL) {

    for (c = 'A' ; c <= 'Z' ; c++)
      fputc ( c , pFile );

    fclose (pFile);
  }
  return 0;
}
```

This program creates a file called `alphabet.txt` and writes `ABCDEFGHIJKLMNOPQRSTUVWXYZ` to it.

20 ***int fputs (const char * str, FILE * stream)***

Graba una cadena al stream.

Writes the *C string* pointed by *str* to the *stream*.

The function begins copying from the address specified (*str*) until it reaches the terminating null character (`'\0'`). This terminating null-character is not copied to the stream.

Notice that `fputs` not only differs from puts in that the destination *stream* can be specified, but also `fputs` does not write additional characters, while puts appends a newline character at the end automatically.

Parameters

`str`

C string with the content to be written to *stream*.

`stream`

Pointer to a FILE object that identifies an output stream.

Return Value

On success, a non-negative value is returned.
On error, the function returns EOF and sets the *error indicator* (ferror).

Example

```
/* fputs example */
#include <stdio.h>

int main () {
   FILE * pFile;
   char sentence [256];

   printf ("Enter sentence to append: ");
   fgets (sentence,256,stdin);
   pFile = fopen ("mylog.txt","a");
```

```
  fputs (sentence,pFile);
  fclose (pFile);
  return 0;
}

```

This program allows to append a line to a file called mylog.txt each time it is run.

21 ***int getc (FILE * stream)***

Obtiene un carácter desde el stream.

Returns the character currently pointed by the internal file position indicator of the specified *stream*. The internal file position indicator is then advanced to the next character.

If the stream is at the end-of-file when called, the function returns EOF and sets the *end-of-file indicator* for the stream (feof).

If a read error occurs, the function returns EOF and sets the *error indicator* for the stream (ferror).

`getc` and fgetc are equivalent, except that `getc` may be implemented as a macro in some libraries. See getchar for a similar function that reads directly from stdin.

Parameters

`stream`

Pointer to a FILE object that identifies an input stream.
Because some libraries may implement this function as a macro, and this may evaluate the *stream* expression more than once, this should be an expression without side effects.

Return Value

On success, the character read is returned (promoted to an `int` value).
The return type is `int` to accommodate for the special value EOF, which indicates failure:
If the position indicator was at the *end-of-file*, the function returns EOF and sets the *eof indicator* (feof) of *stream*.
If some other reading error happens, the function also returns EOF, but sets its *error indicator* (ferror) instead.

Example

```
/* getc example: money counter */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  int c;
  int n = 0;
  pFile=fopen ("myfile.txt","r");
  if (pFile==NULL) perror ("Error opening file");
```

```
  else
  {
    do {
      c = getc (pFile);
      if (c == '$') n++;
    } while (c != EOF);
    fclose (pFile);
    printf ("File contains %d$.\n",n);
  }
  return 0;
}
```

This program reads an existing file called `myfile.txt` character by character and uses the `n` variable to count how many dollar characters (`$`) does the file contain.

22 ***int getchar (void)***
Obtiene un carácter desde stdin.

Returns the next character from the standard input (stdin).

It is equivalent to calling getc with stdin as argument.

Parameters
(none)

Return Value
On success, the character read is returned (promoted to an `int` value).
The return type is `int` to accommodate for the special value EOF, which indicates failure:
If the standard input was at the *end-of-file*, the function returns EOF and sets the *eof indicator* (feof) of stdin.
If some other reading error happens, the function also returns EOF, but sets its *error indicator* (ferror) instead.

Example

```
/* getchar example : typewriter */
#include <stdio.h>

int main () {
  int c;
  puts ("Enter text. Include a dot ('.') in a sentence to
exit:");
  do {
    c = getchar();
    putchar (c);
  } while (c != '.');
  return 0;
}
```

A simple typewriter. Every sentence is echoed once ENTER has been pressed until a dot (.) is included in the text.

23 ***char * gets (char * str)***
Obtiene una cadena desde stdin.

Reads characters from the *standard input* (stdin) and stores them as a C string into *str* until a newline character or the*end-of-file* is reached.

The newline character, if found, is not copied into *str*.

A terminating null character is automatically appended after the characters copied to *str*.

Notice that `gets` is quite different from fgets: not only `gets` uses stdin as source, but it does not include the ending newline character in the resulting string and does not allow to specify a maximum size for *str* (which can lead to buffer overflows).

Parameters

`str`

Pointer to a block of memory (array of `char`) where the string read is copied as a *C string*.

Return Value

On success, the function returns *str*.
If the *end-of-file* is encountered while attempting to read a character, the *eof indicator* is set (feof). If this happens before any characters could be read, the pointer returned is a null pointer (and the contents of *str* remain unchanged).
If a read error occurs, the *error indicator* (ferror) is set and a null pointer is also returned (but the contents pointed by *str*may have changed).

Compatibility

The most recent revision of the C standard (2011) has definitively removed this function from its specification.
The function is deprecated in C++ (as of 2011 standard, which follows C99+TC3).

Example

```
/* gets example */
#include <stdio.h>

int main() {
  char string [256];
  printf ("Insert your full address: ");
  gets (string);     // warning: unsafe (see fgets instead)
  printf ("Your address is: %s\n",string);
  return 0;
}

```

24 ***int putc (int car, FILE * stream)***
Graba un carácter al stream.

The character is written at the position indicated by the *internal position indicator* of the *stream*, which is then automatically advanced by one.

`putc` and fputc are equivalent, except that `putc` may be implemented as a macro in

some libraries. See putchar for a similar function that writes directly to stdout.

Parameters

`character`

The `int` promotion of the character to be written.
The value is internally converted to an `unsigned char` when written.
Because some libraries may implement this function as a macro, and this may evaluate the *stream* expression more than once, this should be an expression without side effects.

`stream`

Pointer to a FILE object that identifies an output stream.

Return Value

On success, the *character* written is returned.
If a writing error occurs, EOF is returned and the *error indicator* (ferror) is set.

Example

```
/* putc example: alphabet writer */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  char c;

  pFile=fopen("alphabet.txt","wt");
  for (c = 'A' ; c <= 'Z' ; c++) {
    putc (c , pFile);
    }
  fclose (pFile);
  return 0;
}
```

This example program creates a file called `alphabet.txt` and writes ABCDEFGHIJKLMNOPQRSTUVWXYZ to it.

25 ***int putchar (int character)***

Graba un carácter al stdout.

Writes a *character* to the *standard output* (stdout).

It is equivalent to calling putc with stdout as second argument.

Parameters

`character`

The `int` promotion of the character to be written.
The value is internally converted to an `unsigned char` when written.

Return Value

On success, the *character* written is returned.
If a writing error occurs, EOF is returned and the *error indicator* (ferror) is set.

Example

```
/* putchar example: printing the alphabet */
#include <stdio.h>

int main () {
  char c;
  for (c = 'A' ; c <= 'Z' ; c++)
    putchar (c);

  return 0;
}
```

This program writes `ABCDEFGHIJKLMNOPQRSTUVWXYZ` to the standard output.

26 ***int puts (const char * str)***

Graba una cadena al stdout.

The function begins copying from the address specified (*str*) until it reaches the terminating null character (`'\0'`). This terminating null-character is not copied to the stream.

Notice that `puts` not only differs from fputs in that it uses stdout as destination, but it also appends a newline character at the end automatically (which fputs does not).

Parameters

`str`

C string to be printed.

Return Value

On success, a non-negative value is returned.
On error, the function returns EOF and sets the *error indicator* (ferror).

Example

```
/* puts example : hello world! */
#include <stdio.h>

int main ()
{
  char string [] = "Hello world!";
  puts (string);
}
```

27 ***int ungetc (int car, FILE * stream)***

Devuelve un carácter desde el stream.

A *character* is virtually put back into an input *stream*, decreasing its *internal file position* as if a previous getc operation was undone.

This *character* may or may not be the one read from the *stream* in the preceding input operation. In any case, the next character retrieved from *stream* is the *character* passed to this function, independently of the original one.

Notice though, that this only affects further input operations on that *stream*, and not the content of the physical file associated with it, which is not modified by any calls to this function.

Some library implementations may support this function to be called multiple times, making the characters available in the reverse order in which they were *put back*. Although this behavior has no standard portability guarantees, and further calls may simply fail after any number of calls beyond the first.

If successful, the function clears the *end-of-file indicator* of *stream* (if it was currently set), and decrements its internal*file position indicator* if it operates in binary mode; In text mode, the *position indicator* has unspecified value until all characters put back with `ungetc` have been read or discarded.

A call to fseek, fsetpos or rewind on *stream* will discard any characters previously put back into it with this function.

If the argument passed for the *character* parameter is EOF, the operation fails and the input *stream* remains unchanged.

Parameters

`character`

The `int` promotion of the character to be put back.
The value is internally converted to an `unsigned char` when put back.

`stream`

Pointer to a FILE object that identifies an input stream.

Return Value

On success, the *character* put back is returned.
If the operation fails, EOF is returned.

Example

```
/* ungetc example */
#include <stdio.h>

int main () {
  FILE * pFile;
  int c;
  char buffer [256];

  pFile = fopen ("myfile.txt","rt");
  if (pFile==NULL)
    perror ("Error opening file");
  else
    while (!feof (pFile)) {
      c=getc (pFile);
      if (c == EOF)
        break;
      if (c == '#')
        ungetc ('@',pFile);
      else
        ungetc (c,pFile);
```

```
      if (fgets (buffer,255,pFile) != NULL)
        fputs (buffer,stdout);
      else
        break;
  }
  return 0;
}
```

This example opens an existing file called `myfile.txt` for reading and prints its lines, but first gets the first character of every line and puts it back into the stream replacing any starting `#` by an `@`.

28 ***size_t fread (void * ptr, size_t size, size_t count, FILE * stream)***

Lee un bloque de datos desde el stream.

Reads an array of *count* elements, each one with a size of *size* bytes, from the *stream* and stores them in the block of memory specified by *ptr*.

The position indicator of the stream is advanced by the total amount of bytes read.

The total amount of bytes read if successful is `(size*count)`.

Parameters

`ptr`
: Pointer to a block of memory with a size of at least `(size*count)` bytes, converted to a `void*`.

`size`
: Size, in bytes, of each element to be read.
size_t is an unsigned integral type.

`count`
: Number of elements, each one with a size of *size* bytes.
size_t is an unsigned integral type.

`stream`
: Pointer to a FILE object that specifies an input stream.

Return Value

The total number of elements successfully read is returned.
If this number differs from the *count* parameter, either a reading error occurred or the *end-of-file* was reached while reading. In both cases, the proper indicator is set, which can be checked with ferror and feof, respectively.
If either *size* or *count* is zero, the function returns zero and both the stream state and the content pointed by *ptr* remain unchanged.
size_t is an unsigned integral type.

Example

```
/* fread example: read an entire file */
#include <stdio.h>
#include <stdlib.h>
```

```
 4
 5 int main () {
 6   FILE * pFile;
 7   long lSize;
 8   char * buffer;
 9   size_t result;
10
11   pFile = fopen ( "myfile.bin" , "rb" );
12   if (pFile==NULL) {
13     fputs ("File error",stderr); exit (1);
14   }
15   // obtain file size:
16   fseek (pFile , 0 , SEEK_END);
17   lSize = ftell (pFile);
18   rewind (pFile);
19   // allocate memory to contain the whole file:
20   buffer = (char*) malloc (sizeof(char)*lSize);
21   if (buffer == NULL) {
22     fputs ("Memory error",stderr); exit (2);
23   }
24   // copy the file into the buffer:
25   result = fread (buffer,1,lSize,pFile);
26   if (result != lSize) {
27     fputs ("Reading error",stderr); exit (3);
28   }
29 /* the whole file is now loaded in the memory buffer. */
30   // terminate
31   fclose (pFile);
32   free (buffer);
33   return 0;
   }
```

This code loads `myfile.bin` into a dynamically allocated memory buffer, which can be used to manipulate the content of a file as an array.

29 ***size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream)***
Graba un bloque de datos al stream.

Writes an array of *count* elements, each one with a size of *size* bytes, from the block of memory pointed by *ptr* to the current position in the *stream*.

The *position indicator* of the stream is advanced by the total number of bytes written.

Internally, the function interprets the block pointed by *ptr* as if it was an array of `(size*count)` elements of type `unsigned char`, and writes them sequentially to *stream* as if fputc was called for each byte.

Parameters

`ptr`
Pointer to the array of elements to be written, converted to a `const void*`.

`size`
Size in bytes of each element to be written.
size_t is an unsigned integral type.

`count`

Number of elements, each one with a size of *size* bytes.
size_t is an unsigned integral type.

`stream`

Pointer to a FILE object that specifies an output stream.

Return Value

The total number of elements successfully written is returned.
If this number differs from the *count* parameter, a writing error prevented the function from completing. In this case, the*error indicator* (ferror) will be set for the *stream*.
If either *size* or *count* is zero, the function returns zero and the *error indicator* remains unchanged.
size_t is an unsigned integral type.

Example

```
/* fwrite example : write buffer */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  char buffer[] = { 'x' , 'y' , 'z' };
  pFile = fopen ("myfile.bin", "wb");
  fwrite (buffer , sizeof(char), sizeof(buffer), pFile);
  fclose (pFile);
  return 0;
}
```

A file called `myfile.bin` is created and the content of the buffer is stored into it. For simplicity, the buffer contains `char`elements but it can contain any other type. `sizeof(buffer)` is the length of the array in bytes (in this case it is three, because the array has three elements of one byte each).

30 ***int fgetpos (FILE * stream, fpos_t * pos)***

Obtiene la posición actual del stream.

Retrieves the current position in the *stream*.

The function fills the fpos_t object pointed by *pos* with the information needed from the *stream*'s *position indicator* to restore the *stream* to its current position (and multibyte state, if *wide-oriented*) with a call to fsetpos.

The ftell function can be used to retrieve the current position in the *stream* as an integer value.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

`pos`

Pointer to a fpos_t object.
This should point to an object already allocated.

Return Value

On success, the function returns zero.
In case of error, errno is set to a platform-specific positive value and the function returns a non-zero value.

Example

```
/* fgetpos example */
#include <stdio.h>
int main () {
   FILE * pFile;
   int c;
   int n;
   fpos_t pos;

   pFile = fopen ("myfile.txt","r");
   if (pFile==NULL) perror ("Error opening file");
   else {
     c = fgetc (pFile);
     printf ("1st character is %c\n",c);
     fgetpos (pFile,&pos);
     for (n=0;n<3;n++) {
        fsetpos (pFile,&pos);
        c = fgetc (pFile);
        printf ("2nd character is %c\n",c);
     }
     fclose (pFile);
   }
   return 0;
}
```

Possible output (with `myfile.txt` containing `ABC`):

```
1st character is A
2nd character is B
2nd character is B
2nd character is B
```

The example opens `myfile.txt`, then reads the first character once, and then reads 3 times the same second character.

31 ***int fseek (FILE * stream, long int offset, int origen)***

Reposiciona el indicador de posición al stream.

For streams open in binary mode, the new position is defined by adding *offset* to a reference position specified by *origin*.

For streams open in text mode, *offset* shall either be zero or a value returned by a previous call to ftell, and *origin* shall necessarily be `SEEK_SET`.

If the function is called with other values for these arguments, support depends on the particular system and library implementation (non-portable).

The *end-of-file internal indicator* of the *stream* is cleared after a successful call to this function, and all effects from previous calls to ungetc on this *stream* are dropped.

On streams open for update (read+write), a call to `fseek` allows to switch between reading and writing.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

`offset`

Binary files: Number of bytes to offset from *origin*.
Text files: Either zero, or a value returned by ftell.

`origin`

Position used as reference for the *offset*. It is specified by one of the following constants defined in <cstdio>exclusively to be used as arguments for this function:

Constant	Reference position
SEEK_SET	Beginning of file
SEEK_CUR	Current position of the file pointer
SEEK_END	End of file *

* Library implementations are allowed to not meaningfully support `SEEK_END` (therefore, code using it has no real standard portability).

Return Value

If successful, the function returns zero.
Otherwise, it returns non-zero value.
If a read or write error occurs, the *error indicator* (ferror) is set.

Example

```
/* fseek example */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  pFile = fopen ( "example.txt" , "wb" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
  return 0;
}
```

After this code is successfully executed, the file `example.txt` contains:

```
This is a sample
```

32 ***int fsetpos (FILE * stream, const fpos_t * pos)***

Pone el indicador de posición al stream.

Restores the current position in the *stream* to *pos*.

The *internal file position indicator* associated with *stream* is set to the position represented by *pos*, which is a pointer to an fpos_t object whose value shall have been previously obtained by a call to fgetpos.

The *end-of-file internal indicator* of the *stream* is cleared after a successful call to this function, and all effects from previous calls to ungetc on this *stream* are dropped.

On streams open for update (read+write), a call to `fsetpos` allows to switch between reading and writing.

A similar function, fseek, can be used to set arbitrary positions on streams open in binary mode.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

`position`

Pointer to a fpos_t object containing a position previously obtained with fgetpos.

Return Value

If successful, the function returns zero.
On failure, a non-zero value is returned and errno is set to a system-specific positive value.

Example

```
1  /* fsetpos example */
2  #include <stdio.h>
3
4  int main ()
5  {
6    FILE * pFile;
7    fpos_t position;
8
9    pFile = fopen ("myfile.txt","w");
10   fgetpos (pFile, &position);
11   fputs ("That is a sample",pFile);
12   fsetpos (pFile, &position);
13   fputs ("This",pFile);
14   fclose (pFile);
15   return 0;
16 }
```

After this code is successfully executed, a file called `myfile.txt` will contain:

```
This is a sample
```

33 ***long int ftell (FILE * stream)***

Obtiene la posición actual del stream.

Returns the current value of the position indicator of the *stream*.

For binary streams, this is the number of bytes from the beginning of the file.

For text streams, the numerical value may not be meaningful but can still be used to restore the position to the same position later using fseek (if there are characters put back using ungetc still pending of being read, the behavior is undefined).

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

Return Value

On success, the current value of the position indicator is returned.
On failure, `-1L` is returned, and errno is set to a system-specific positive value.

Example

```
/* ftell example : getting size of a file */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  long size;

  pFile = fopen ("myfile.txt","rb");
  if (pFile==NULL) perror ("Error opening file");
  else
  {
    fseek (pFile, 0, SEEK_END);   // non-portable
    size = ftell (pFile);
    fclose (pFile);
    printf ("Size of myfile.txt: %ld bytes.\n",size);
  }
  return 0;
}
```

This program prints out the size of `myfile.txt` in bytes (where supported).

34 ***void rewind (FILE * stream)***

Pone la posición del stream al inicio.

Sets the position indicator associated with *stream* to the beginning of the file.

The *end-of-file* and *error* internal indicators associated to the *stream* are cleared after a successful call to this function, and all effects from previous calls to ungetc on this *stream* are dropped.

On streams open for update (read+write), a call to `rewind` allows to switch between reading and writing.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

Return Value

none

Example

```
/* rewind example */
#include <stdio.h>

int main ()
{
  int n;
  FILE * pFile;
  char buffer [27];

  pFile = fopen ("myfile.txt","w+");
  for ( n='A' ; n<='Z' ; n++)
    fputc ( n, pFile);
  rewind (pFile);
  fread (buffer,1,26,pFile);
  fclose (pFile);
  buffer[26]='\0';
  puts (buffer);
  return 0;
}
```

A file called `myfile.txt` is created for reading and writing and filled with the alphabet. The file is then rewinded, read and its content is stored in a buffer, that then is written to the standard output:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

35 ***void clearerr (FILE * stream)***

Limpia los indicadores de error.

Resets both the *error* and the *eof* indicators of the *stream*.

When a i/o function fails either because of an error or because the end of the file has been reached, one of these internal indicators may be set for the *stream*. The state of these indicators is cleared by a call to this function, or by a call to any of: rewind, fseek, fsetpos and freopen.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

Return Value

None

Example

```
/* writing errors */
#include <stdio.h>
int main () {
  FILE * pFile;

  pFile = fopen("myfile.txt","r");
  if (pFile==NULL)
    perror ("Error opening file");
  else {
    fputc ('x',pFile);
    if (ferror (pFile)) {
      printf ("Error Writing to myfile.txt\n");
      clearerr (pFile);
    }
    fgetc (pFile);
    if (!ferror (pFile))
      printf ("No errors reading myfile.txt\n");
    fclose (pFile);
  }
  return 0;
}
```

This program opens an existing file called `myfile.txt` for reading and causes an I/O error trying to write on it. That error is cleared using `clearerr`, so a second error checking returns false.

Salida:

```
Error Writing to myfile.txt
No errors reading myfile.txt
```

36 *int feof (FILE * stream)*
Verifica el final del archivo.

Checks whether the *end-of-File indicator* associated with *stream* is set, returning a value different from zero if it is.

This indicator is generally set by a previous operation on the *stream* that attempted to read at or past the end-of-file.

Notice that *stream*'s internal position indicator may point to the *end-of-file* for the next operation, but still, the *end-of-file*indicator may not be set until an operation attempts to read at that point.

This indicator is cleared by a call to clearerr, rewind, fseek, fsetpos or freopen. Although if the *position indicator* is not repositioned by such a call, the next i/o operation is likely to set the indicator again.

Parameters

`stream`
Pointer to a FILE object that identifies the stream.

Return Value

A non-zero value is returned in the case that the *end-of-file indicator* associated with the stream is set.
Otherwise, zero is returned.

Example

```
/* feof example: byte counter */
#include <stdio.h>

int main () {
  FILE * pFile;
  int n = 0;

  pFile = fopen ("myfile.txt","rb");
  if (pFile==NULL)
    perror ("Error opening file");
  else {
    while (fgetc(pFile) != EOF) {
      ++n;
    }
    if (feof(pFile)) {
      puts ("End-of-File reached.");
      printf ("Total number of bytes read: %d\n", n);
    }
    else
      puts ("End-of-File was not reached.");
    fclose (pFile);
  }
  return 0;
}
```

This code opens the file called `myfile.txt`, and counts the number of characters that it contains by reading all of them one by one. The program checks whether the end-of-file was reached, and if so, prints the total number of bytes read.

37 ***int ferror (FILE * stream)***

Verifica el indicador de error.

Checks if the *error indicator* associated with *stream* is set, returning a value different from zero if it is.

This indicator is generally set by a previous operation on the *stream* that failed, and is cleared by a call to clearerr, rewindor freopen.

Parameters

`stream`

Pointer to a FILE object that identifies the stream.

Return Value

A non-zero value is returned in the case that the *error indicator* associated with the stream is set.
Otherwise, zero is returned.

Example

```
/* ferror example: writing error */
#include <stdio.h>
int main ()
{
  FILE * pFile;
  pFile=fopen("myfile.txt","r");
  if (pFile==NULL) perror ("Error opening file");
  else {
    fputc ('x',pFile);
    if (ferror (pFile))
      printf ("Error Writing to myfile.txt\n");
    fclose (pFile);
  }
  return 0;
}
```

This program opens an existing file called `myfile.txt` in read-only mode but tries to write a character to it, generating an error that is detected by `ferror`.

Salida:

```
Error Writing to myfile.txt
```

38 ***void perror (const char * str)***

Imprime un mensaje de error.

Interprets the value of errno as an error message, and prints it to stderr (the standard error output stream, usually the console), optionally preceding it with the custom message specified in *str*.

errno is an integral variable whose value describes the error condition or diagnostic information produced by a call to a library function (any function of the C standard library may set a value for errno, even if not explicitly specified in this reference, and even if no error happened), see errno for more info.

The error message produced by `perror` is platform-depend.

If the parameter *str* is not a null pointer, *str* is printed followed by a colon (:) and a space. Then, whether *str* was a null pointer or not, the generated error description is printed followed by a newline character (`'\n'`).

`perror` should be called right after the error was produced, otherwise it can be overwritten by calls to other functions.

Parameters.

`str`

C string containing a custom message to be printed before the error message itself.
If it is a null pointer, no preceding custom message is printed, but the error message is still printed.
By convention, the name of the application itself is generally used as

parameter.

Return Value
none

Example

```
/* perror example */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  pFile=fopen ("unexist.ent","rb");
  if (pFile==NULL)
    perror ("The following error occurred");
  else
    fclose (pFile);
  return 0;
}
```

If the file `unexist.ent` does not exist, something similar to this may be expected as program output:

```
The following error occurred: No such file or directory
```

39	UFSIZ Tamaño del buffer.
40	EOF Macro que indica Fin del Archivo, generalmente representado por el valor -1. It is a macro definition of type `int` that expands into a negative integral constant expression (generally, `-1`). It is used as the value returned by several functions in header <cstdio> to indicate that the End-of-File has been reached or to signal some other failure conditions. It is also used as the value to represent an invalid character. In C++, this macro corresponds to the value of `char_traits<char>::eof()`.
41	FILENE_MAX Máxima longitud del nombre de archivo.
42	FOPEN_MAX Cantidad máxima de archivos abiertos simultáneamente.
43	NULL Macro que indica Puntero nulo, establecido por un valor de cero.
44	SEEK_SET A partir del comienzo del archivo mueve el puntero.
45	SEEK_CUR A partir de la posición actual del puntero mueve el puntero hacia adelante si es positivo o hacia atrás si es negativo.
46	SEEK_END A partir del final del archivo mueve el puntero hacia adelante si es positivo o hacia atrás si es negativo.
47	FILE Objeto que contiene información para controlar un stream.

48	fpos_t Objeto que contiene información para indicar la posición dentro del archivo. This type of object is used to specify a position within a file. An object of this type is capable of specifying uniquely any position within a file. The information in `fpos_t` objects is usually filled by a call to fgetpos, which takes a pointer to an object of this type as argument. The content of an `fpos_t` object is not meant to be read directly, but only to be used as an argument in a call to fsetpos.
49	size_t Tipo entero sin signo. Alias of one of the fundamental unsigned integer types. It is a type able to represent the size of any object in bytes: `size_t` is the type returned by the `sizeof` operator and is widely used in the standard library to represent sizes and counts. In <cstdio>, it is used as the type of some parameters in the functions fread, fwrite and setvbuf, and in the case of freadand fwrite also as its returning type.

TABLAS PARA ARCHIVOS DEL LENGUAJE C++

Items	Indicadores valor		Comentario
	Indicadores: cout.**setf**(ios::indic$_1$ | ios::indic$_2$…) cout.**unsetf**(ios::indic$_1$ | ios::indic$_2$…) cout.**flags**(ios::indic$_1$ | ios::indic$_2$…) El símbolo (|) barra vertical, lease o lógico, el cual permite indicar más de una bandera en una misma función miembro de la clase ios.		**Permite**: Activar banderas indicadas sin modificar las otras. Desactivar banderas indicadas sin modificar las otras. Resetear todo y activa banderas indicadas.
	Ejemplos de formas de uso cout.**setf**(ios::boolalpha | ios::fixed); cout.**unsetf**(ios::boolalpha); cout.**flags**(ios::left)		
1	skipws	0x0001	Se descartan blancos iniciales a la entrada.
2	left	0x0002	La salida se alinea a la izquierda.
3	right	0x0004	La salida se alinea a la derecha.
4	internal	0x0008	Se alinea el signo y los caracteres indicativos por la izquierda y los dígitos por la derecha.
5	dec	0x0010	Salida decimal para enteros por omisión.
6	oct	0x0020	Salida octal para enteros.
7	hex	0x0040	Salida hexadecimal para enteros.
8	showbase	0x0080	Se muestra la base de los valores numéricos.
9	showpoint	0x0100	Se muestra el punto decimal.
10	uppercase	0x0200	Los caracteres de formato aparecen en mayúsculas.
11	showpos	0x0400	Se muestra el signo + en los valores positivos.
12	scientific	0x0800	Notación científica para coma flotante.
13	fixed	0x1000	Notación normal para coma flotante.
14	unitbuf	0x2000	Salida sin buffer (se vuelca cada operación).
15	stdio	0x4000	Vacía los buffers de stdout y stderr luego de una inserción.

Funciones miembro de la clase ios

Funciones miembro de <ios.h>
Ejemplos de formas de uso

stream.indicador()
cout.indicador();
cout.bad();
cout.eof();
cout.fill();

1 **bad()**
Clase ios

```
int bad();
```

Retorna un valor distinto de cero si ha ocurrido un error.

Sólo se comprueba el bit de estado *ios::badbit*, de modo que esta función no equivale a !good().

```
1  // error state flags
2  #include <fstream>     // cout, ios
3  #include <sstream>      // stringstream
4  using namespace std;
5
6  void print_state (const std::ios& stream) {
7    cout << " good()=" << stream.good();
8    cout << " eof()=" << stream.eof();
9    cout << " fail()=" << stream.fail();
10   cout << " bad()=" << stream.bad();
11 }
12
13 int main () {
14   stringstream stream;
15
16   stream.clear (stream.goodbit);
17   cout << "goodbit:"; print_state(stream);cout << '\n';
18   stream.clear (stream.eofbit);
19   cout << " eofbit:"; print_state(stream);cout << '\n';
20   stream.clear (stream.failbit);
21   cout << "failbit:"; print_state(stream);cout << '\n';
22   stream.clear (stream.badbit);
23   cout << " badbit:"; print_state(stream);cout << '\n';
24   return 0;
25 }
26
27
28
```

2 **clear()**
Clase ios

```
void clear(iostate state=0);
```

Sirve para modificar los bits de estado de un *stream*, normalmente para eliminar un estado de error. Se

suelen usar constantes definidas en el enum *io_state*definido en *ios*, usando el operador de bits OR para modificar varios bits a la vez.

```
1  // clearing errors
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    char buffer [80];
7    fstream myfile;
8
9    myfile.open ("test.txt",fstream::in);
10   myfile << "test";
11   if (myfile.fail()){
12     cout << "Error writing to
13                          test.txt\n";
14     myfile.clear();
15   }
16   myfile.getline (buffer,80);
17   cout << buffer << " successfully
18                 read from file.\n";
19
20   return 0;
21 }
22
```

3 **close()**
Clase fstream

```
filebuf* close();
```

Actualiza la información actualmente en el buffer y cierra el fichero. En caso de error, retorna el valor 0.

```
1  // print the content of a text file.
2  #include <fstream>
3
4  int main () {
5    ifstream ifs;
6
7    ifs.open ("test.txt");
8    char c = ifs.get();
9    while (ifs.good()) {
10     cout << c;
11     c = ifs.get();
12   }
13   ifs.close();
14   return 0;
15 }
16
17
18
19
20
```

4	**eof**()
	Clase ios
		`int eof();`
		
		Retorna un valor distinto de cero si se ha alcanzado el fin de fichero.
		
		Esta función únicamente comprueba el bit de estado *ios::eofbit*.
		
		`//eof example`
		`#include <fstream>`
		
		`int main () {`
		
		`  ifstream  is("example.txt");  //`
		`open file`
		
		`  char c;`
		`  while (is.get(c)) // loop getting`
		`  single characters`
		`  cout << c;`
		`  if (is.eof())  // check for EOF`
		`  cout << "[EoF reached]\n";`
		`  else`
		`  cout << "[error reading]\n";`
		`  is.close();  // close file`
		`  return 0;`
		`}`

5	**fail**() Clase ios	`int fail();` Retorna un valor distinto de cero si una operación sobre el *stream* ha fallado. Comprueba los bits de estado *ios::badbit* y *ios::failbit*.

6	**fill**()
	Clase ios
		Cambia el carácter de relleno que se usa cuando la salida es más ancha de la necesaria para el dato actual.
		
		`int fill();`
		`int fill(char);`
		
		La primera forma devuelve el valor actual del carácter de relleno, la segunda permite cambiar el carácter de relleno para las siguientes salidas, y también devuelve el valor actual.
		
		Ejemplo:

```
int x = 23;
cout << "|";
cout.width(10);
cout.fill('%');
cout << x << "|" << x << "|" << endl;
```

7 **flags()**
Clase ios_base

Permite cambiar o leer los flags de manipulación de formato.

```
long flags () const;
long flags (long valor);
```

La primera forma devuelve el valor actual de los flags.

La segunda cambia el valor actual por valor, el valor de retorno es el valor previo de los flags.

Ejemplo:

```
int x = 235;
long f;

cout << "|";
f = flags();
f &= !(ios::adjustfield);
f |= ios::left;
cout.flags(f);
cout.width(10);
cout << x << "|" << endl;
```

8 **flush()**
Clase ostream

```
ostream& flush();
```

Vacía el buffer asociado al stream. Procesa todas las salidas pendientes.

```
1  // istream::peek example
2  #include <fstream>
3  #include <string>       //string
4  #include <cctype>       //isdigit
5  using namespace std;
6
7  int main () {
8
9    stcout << "Please, enter a number or
10 a word: ";
11   cout.flush();     // ensure output is
12 written
13
14   cin  >>ws;    //  eat  up  any  leading
15 white spaces
16   int   c   =cin.peek();      //   peek
17 character
```

```
18
19  if ( c == EOF )
20     return 1;
21  if (isdigit(c) ) {
22    int n;
23    cin >> n;
24    cout << "You entered the number: "
25                  << n << '\n';
26  }
27  else {
28    string str;
29    cin >> str;
      cout << "You entered the word: "
           << str << '\n';
    }
    return 0;
}
```

9 **gcount()**
Clase istream

```
int gcount();
```

Retorna el número de caracteres sin formato de la última lectura. Las lecturas sin formato son las realizadas mediante las funciones get, getline y read.

```
1  // cin.gcount example
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    char str[20];
7
8    cout << "Please, enter a word: ";
9    cin.getline(str,20);
10   cout <<cin.gcount() << " characters
11 read: " << str << '\n';
12
     return 0;
   }
```

10 **get()**
Clase istream

```
int get();
istream& get(char*, int len, char = '\n');
istream& get(char&);
istream& get(streambuf&, char = '\n');
```

La primera forma extrae el siguiente carácter o EOF si

no hay disponible ninguno.

La segunda forma extrae caracteres en la dirección proporcionada en el parámetro char* hasta que se recibe el delimitador del tercer parámetro, el fin de fichero o hasta que se leen len-1 bytes. Siempre se añade un carácter nulo de terminación en la cadena de salida. El delimitador no se extrae desde el stream de entrada. La función sólo falla si no se extraen caracteres.

La tercera forma extrae un único carácter en la referencia a char proporcionada.

La cuarta forma extrae caracteres en el streambuf especificado hasta que se encuentra el delimitador.

```
1  // print the content of a text file.
2  #include <fstream>
3  #include <fstream>      //ifstream
4  using namespace std;
5
6  int main () {
7    ifstream ifs;
8
9    ifs.open ("test.txt",ifstream::in);
10
11   char c = ifs.get();
12
13   while (ifs.good()) {
14     cout << c;
15     c = ifs.get();
16   }
17
18   ifs.close();
19
20   return 0;
     }
```

11 **getline()**
Clase istream

```
istream& getline(char*, int, char = '\n');
```

Extrae caracteres hasta que se encuentra el delimitador y los coloca en el buffer, elimina el delimitador del stream de entrada y no lo añade al buffer.

```
1  // cin.gcount example
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    char str[20];
```

```
7
8   cout << "Please, enter a word: ";
9   cin.getline(str,20);
10  cout <<cin.gcount() << " characters
11 read: " << str << '\n';
12
    return 0;
}
```

12 **good**()
Clase ios

Retorna un valor distinto de cero si no ha ocurrido ningún error, es decir, si ninguno de los bits de estado está activo.

Aunque pudiera parecerlo, (*good* significa bueno y *bad* malo, en inglés) esta función no es exactamente equivalente a !*bad*().
En realidad es equivalente a *rdstate*() == 0.

```
1  // print the content of a text file.
2  #include <fstream>
3  using namespace std;
4
5  main () {
6    ifstream ifs;
7
8    ifs.open ("test.txt",ifstream::in);
9
10   char c = ifs.get();
11
12   while (ifs.good()) {
13     cout << c;
14     c = ifs.get();
15   }
16
17   ifs.close();
18
19   return 0;
20 }
```

13 **ignore**()
Clase istream

```
istream& ignore(int n = 1, int delim = EOF);
```

Hace que los siguientes n caracteres en el stream de entrada sean ignorados; la extracción se detiene antes si se encuentra el delimitador delim.

El delimitador también es extraído del stream.

```
1  // istream::ignore example
2  #include <fstream>
3  using namespace std;
```

```
4
5  int main () {
6    char first, last;
7
8    cout << "Please, enter your first
9  name followed by your surname: ";
10
11   first =cin.get();        // get one
12 character
13   cin.ignore(256,'  ');      // ignore
14 until space
15
16   last =cin.get();         // get one
17 character

     cout << "Your initials are " <<
   first << last << '\n';

     return 0;
   }
```

14 **open()**
Clase fstream

```
filebuf* open(const char *name, int mode,
   int prot = filebuf::openprot);
```

Abre un fichero para un objeto de una clase específica, con el nombre *name*. Para el parámetro *mode* se puede usar el enum open_mode definido en la clase ios.

Parámetro mode	Efecto
ios::app	(append) Se coloca al final del fichero antes de cada operación de escritura.
ios::ate	(at end) Se coloca al final del stream al abrir el fichero.
ios::binary	Trata el stream como binario, no como texto.
ios::in	Permite operaciones de entrada en un stream.
ios::out	Permite operaciones de salida en un stream.
ios::trunc	(truncate) Trunca el fichero a cero al abrirlo.

El parámetro *prot* se corresponde con el permiso de acceso DOS y es usado siempre que no se indique el modo *ios::nocreate*. Por defecto se usa el permiso para leer y escribir. En algunas versiones de las bibliotecas de streams no existe este parámetro, ya que está íntimamente asociado al sistema operativo.

```
enum open_mode { in, out, ate, app, trunc,
nocreate, noreplace, binary };
```

```
1  // print the content of a text file.
2  #include <fstream>
3  using namespace std;
4
5  main () {
6    ifstream ifs;
7
8    ifs.open ("test.txt",ifstream::in);
9
10   char c = ifs.get();
11
12   while (ifs.good()) {
13     cout << c;
14     c = ifs.get();
15   }
16
17   ifs.close();
18
19   return 0;
20 }
```

15 **peek()**
Clase istream

```
int peek();
```

Retorna el siguiente carácter sin extraerlo del stream.

```
1  // istream::peek example
2  #include <fstream>
3  #include <string>       //string
4  #include <cctype>       //isdigit
5  using namespace std;
6
7  int main () {
8
9    cout << "Please, enter a number or a
10 word: ";
11   cout.flush();     // ensure output is
12 written
13
14   cin  >>  std::ws;    //  eat  up  any
15 leading white spaces
16   int  c  =  std::cin.peek();   //  peek
17 character
```

```
18
19  if ( c == EOF )
20     return 1;
21  if (isdigit(c) ) {
22    int n;
23    cin >> n;
24    cout << "You entered the number: "
25 << n << '\n';
26  }
27  else {
28    string str;
29    cin >> str;
      cout << "You entered the word: "
   << str << '\n';
    }

    return 0;
   }
```

16	**precisión()** Clase ios_base	Permite cambiar el número de caracteres significativos que se mostrarán cuando trabajemos con números en coma flotante: float o double.

```
int precision();
int precision(char);
```

La primera forma retorna el valor actual de la precisión, la segunda permite modificar la precisión para las siguientes salidas, y también devuelve el valor actual.

```
1  // modify precision
2  #include <fstream>
3
4  int main () {
5    double f = 3.14159;
6    cout.unsetf    (ios::floatfield    );
7  // floatfield not set
8    cout.precision(5);
9    cout << f << '\n';
10   cout.precision(10);
11   cout << f << '\n';
12   cout.setf(ios::fixed,ios::floatfield
13 ); // floatfield set to fixed
14   cout << f << '\n';
     return 0;
   }
```

17 put()
Clase ostream

```
ostream& put(char ch);
```

Inserta un carácter en el stream de salida.

```
   char l = 'l';
   unsigned char a = 'a';

   cout.put('H').put('o').put(l).put(a) <<
endl;
```

```
// typewriter
#include <fstream>      // std::ofstream

int main () {
  ofstream outfile ("test.txt");
  char ch;

  cout << "Type some text (type a dot to
finish):\n";
  do {
    ch = std::cin.get();
    outfile.put(ch);
  } while (ch!='.');

  return 0;
}
```

18 **putback**()
Clase istream

```
istream& putback(char);
```

Retorna un carácter al stream.

```
1  // istream::putback example
2  #include <string>       //string
3
4  int main () {
5    cout << "Please, enter a number or a
6  word: ";
7    char c =cin.get();
8
9    if ( (c >= '0') && (c <= '9') ) {
10     int n;
11     cin.putback (c);
12     cin >> n;
13     cout << "You entered a number: "
14 << n << '\n';
15   }
16   else {
17     string str;
18     cin.putback (c);
```

```
19     getline (std::cin,str);
20     cout << "You entered a word: " <<
21 str << '\n';
22   }
23   return 0;
24 }
```

19 **rdstate()**
Clase ios

```
int rdstate();
```

Retorna el estado del *stream*. Este estado puede ser una combinación de cualquiera de los bits de estado definidos en el enum *ios::io_state*, es decir*ios::badbit*, *ios::eofbit*, *ios::failbit* e *ios::goodbit*. El goodbit no es en realidad un bit, sino la ausencia de todos los demás. De modo que para verificar que el valor obtenido por rdstate es *ios::goodbit* tan sólo hay que comparar. En cualquier caso es mejor usar la función good().

En cuanto a los restantes bits de estado se puede usar el operador & para verificar la presencia de cada uno de los bits. Aunque de nuevo, es preferible usar las funciones bad(), eof() o fail().

20 **read()**
Clase istream

```
istream& read(char*, int);
```

Extrae el número indicado de caracteres en el array char*. Se puede usar la función gcount() para saber el número de caracteres extraídos si ocurre algún error.

```
1  // read a file into memory
2  #include <fstream>      //ifstream
3
4  int main () {
5
6    ifstream is
7  ("test.txt",ifstream::binary);
8    if (is) {
9      // get length of file:
10     is.seekg (0, is.end);
11     int length = is.tellg();
12     is.seekg (0, is.beg);
13
14     char * buffer = new char [length];
15
16     cout << "Reading " << length << "
17 characters... ";
18     // read data as a block:
19     is.read (buffer,length);
```

```
20
21     if (is)
22       cout << "all characters read
23 successfully.";
24     else
25       cout << "error: only " <<
26 is.gcount() << " could be read";
27     is.close();
28
29     // ...buffer contains the entire
30 file...
31
       delete[] buffer;
     }
     return 0;
   }
```

21 **seekg()**
Clase istream

```
istream& seekg(streampos pos);
istream& seekg(streamoff offset, seek_dir
dir);
```

La primera forma se mueve a posición absoluta, tal como la proporciona la función tellg.

La segunda forma se mueve un número *offset* de bytes la posición del cursor del stream relativa a dir. Este parámetro puede tomar los valores definidos en el enum *seek_dir*: {*beg*, *cur*, *end*};

Para streams de salida usar ostream::seekp.

Usar *seekpos* o *seekoff* para moverse en un buffer de un stream.

```
1  // read a file into memory
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    ifstream is("test.txt",ifstream ::
7                                binary);
8    if (is) {
9      // get length of file:
10     is.seekg (0, is.end);
11     int length = is.tellg();
12     is.seekg (0, is.beg);
13
14     // allocate memory:
15     char * buffer = new char [length];
16
17     // read data as a block:
18     is.read (buffer,length);
```

```
19
20      is.close();
21
22      // print content:
23      cout.write (buffer,length);
24
25      delete[] buffer;
26    }
27
28    return 0;
  }
```

22 **seekp()**
Clase ostream

```
ostream& seekp(streampos);
ostream& seekp(streamoff, seek_dir);
```

La primera forma mueve el cursor del stream a una posición absoluta, tal como la devuelve la función tellp.

La segunda forma mueve el cursor a una posición relativa desde el punto indicado mediante el parámetro *seek_dir*, que puede tomar los valores del enum*seek_dir*: *beg*, *cur*, *end*.

```
1  // position in output stream
2  #include <fstream>      //ofstream
3  using namespace std;
4
5  int main () {
6
7    ofstream outfile;
8    outfile.open ("test.txt");
9
10   outfile.write("This is an apple",16);
11   long pos = outfile.tellp();
12   outfile.seekp (pos-7);
13   outfile.write (" sam",4);
14
15   outfile.close();
16
17   return 0;
   }
```

23 **setf()**
Clase ios_base

Permite modificar los flags de manipulación de formato.

```
long setf(long);
long setf(long valor, long mascara);
```

La primera forma activa los flags que estén activos

tanto en el parámetro y deja sin cambios el resto.

La segunda forma activa los flags que estén activos tanto en valor como en máscara y desactiva los que estén activos en *mask*, pero no en valor. Podemos considerar que *mask* contiene activos los flags que queremos modificar y *valor* los flags que queremos activar.

Ambos devuelven el valor previo de los flags.

```
int x = 235;

cout << "|";
cout.setf(ios::left, ios::left |
   ios::right | ios::internal);
cout.width(10);
cout << x << "|" << endl;
```

24 **tellg()**
Clase istream

```
long tellg();
```

Retorna la posición actual del stream.

```
1  // read a file into memory
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6    ifstream        is        ("test.txt",
7  ifstream::binary);
8    if (is) {
9      // get length of file:
10     is.seekg (0, is.end);
11     int length = is.tellg();
12     is.seekg (0, is.beg);
13
14     // allocate memory:
15     char * buffer = new char [length];
16
17     // read data as a block:
18     is.read (buffer,length);
19
20     is.close();
21
22     // print content:
23     cout.write (buffer,length);
24
25     delete[] buffer;
26   }
27
28   return 0;
}
```

25	**tellp**()
	Clase ostream
		`streampos tellp();`
		Retorna la posición absoluta del cursor del stream.
		t `// position in output stream`
		`#include <fstream>  //ofstream`
		`using namespace std;`
		
		`int main () {`
		
		`  ofstream outfile;`
		`  outfile.open ("test.txt");`
		
		`  outfile.write("This is an apple",16);`
		`  long pos = outfile.tellp();`
		`  outfile.seekp (pos-7);`
		`  outfile.write (" sam",4);`
		
		`  outfile.close();`
		
		`  return 0;`
		`}`

26	**unsetf**()
	Clase ios_base
		Permite eliminar flags de manipulación de formato:
		`void unsetf(long mascara);`
		Desactiva los flags que estén activos en el parámetro.
		1 `// modifying flags with setf/unsetf`
		2 `#include <fstream>  //cout,ios`
		3 `using namespace std;`
		4
		5 `int main () {`
		6 `  cout.setf(ios::hex,ios::basefield );`
		7 `// set hex as the basefield`
		8 `  cout.setf ( std::ios::showbase );`
		9 `// activate showbase`
		10 `  cout << 100 << '\n';`
		11 `  cout.unsetf ( std::ios::showbase );`
		`// deactivate showbase`
		`  cout << 100 << '\n';`
		`  return 0;`
		`}`

27	**width**() Clase ios_base	Cambia la anchura en caracteres de la siguiente salida de *stream*: `int width();`

<table>
<tr><td></td><td></td><td>

```
int width(int);
```

La primera forma retorna el valor de la anchura actual, la segunda permite cambiar la anchura para las siguientes salidas, y también devuelve el valor actual de la anchura.

```
int x = 23;

cout << "#";
cout.width(10);
cout << x << "#" << x << "#" << endl;
```

</td></tr>
<tr><td>28</td><td>write()
Clase ostream</td><td>

```
ostream& write(const char*, int n);
```

Inserta *n* caracteres (aunque sean nulos) en el stream de salida.

```
// position in output stream
#include <fstream>      //ofstream
using namespace std;

main () {

  ofstream outfile;
  outfile.open ("test.txt");

  outfile.write("This is an apple",16);
  long pos = outfile.tellp();
  outfile.seekp (pos-7);
  outfile.write (" sam",4);

  outfile.close();

  return 0;
}
```

</td></tr>
</table>

AGREGADOS ADICIONALES - Función fatoi convierte cadena a entero

```
/*
  Id.Programa: fatoi.cpp
  Autor......: Lic. Hugo Cuello
  Fecha......: julio-2021
  Comentario.: fatoi(const char cad[], int &cod) a diferencia de la función
                 atoi(const char cad[]) devuelve en su segundo parametro formal el
                 estado de la conversion. Si fue exitosa devuelve cero en cod, sino, > 0.
*/

#include <ctype.h>
#include <fstream>
using namespace std;
/* atoi: convierte s a entero; versión 2 */
int fatoi(char s[], int &cod) {
  int i = 0, n, sign;
  for (; isspace(s[i]); i++ ); // ignora espacio en blanco
  sign = (s[i] == '-') ? -1 : 1;
  if(s[i] == '+' || s[i] == '-') // ignora el signo
    i++;
  for (n = 0 ; isdigit(s[i]); i++ )
    n = 10 * n + (s[i] - '0');
  if(s[i] == '\0')
     cod = 0;
  else
    cod = i+1;
  return sign * n;
} // fatoi

main() {
  char cad[15];
  int  nro,
       cErr;
  strcpy(cad,"1234");
  nro = fatoi(cad,cErr);
  if (cErr == 0)
    cout << nro;
  else
    cout << "Error al convertir cad. a nro.";
  return 0;
}
```

Enlaces a archivos

Diagramas de Estructuras en Visio:

Guía #1 Algoritmos.

Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto.

Guía #4 Estr. Datos: Arhivos Binarios.

Guía #5 Estr. Datos: Dinámicas Lineales y No Lineales.

Código Fuente en C++ Borland / cygwin GCC v 20

Guía #1 Algoritmos.

Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto.

Guía #3 Conjuntos.

Guía #4 Archivos Binarios.

Guía #5 Estr. Datos: Dinámicas Lineales y No Lineales.

Código Fuente en Python V 12.

Guía #1 Algoritmos.

Guía #2 Estr. Datos: Registro, Cadenas, Arreglos con Archivo de Texto.

Guía # 3: Conjuntos.

Guía # 4: Archivos Binarios.

Guía # 5: Estr. Datos: Dinámicas Lineales y No Lineales.

Código Fuente en Embarcadfero C++ Builder 12.1

Guía #1 Algoritmos **VCL Proyectos.**

Guía #1 Algoritmos **CONSOLA Proyectos.**

Guía #2 Estr. Datos Registro + Arreglos + Archivo de Texto **VCL Proyectos.**

Guía #2 Estr. Datos Registro + Arreglos + Archivo de Texto **CONSOLA Proyectos**