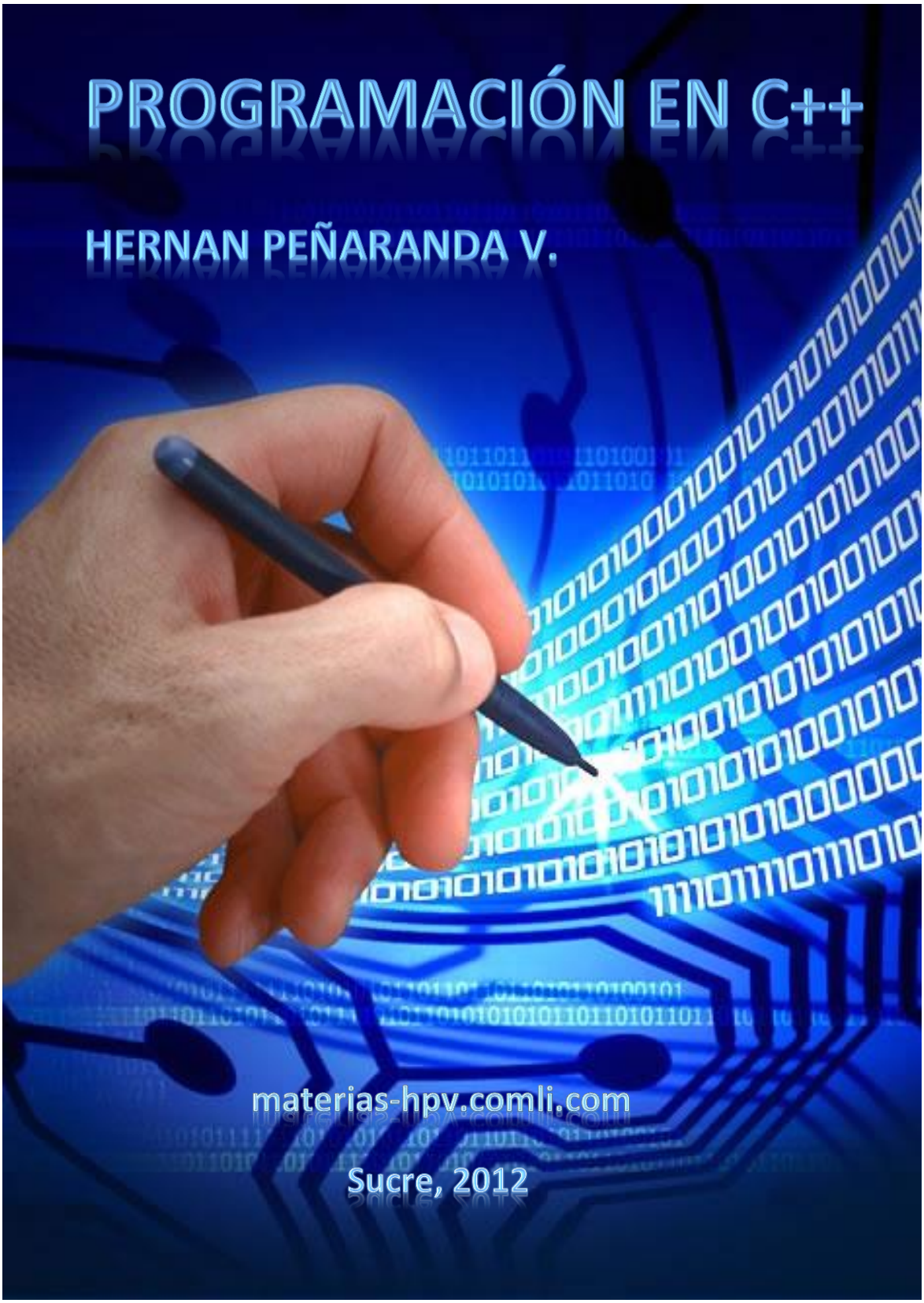
PROGRAMACIÓN EN C++

HERNAN PEÑARANDA V.



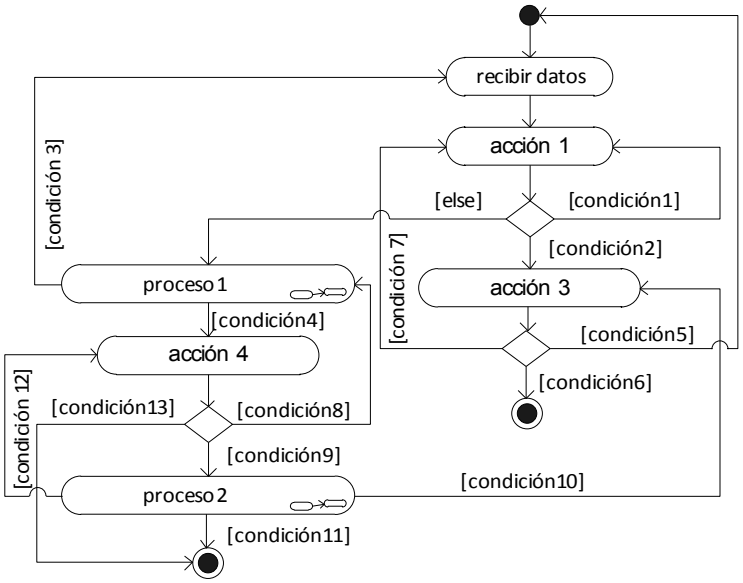
materias-hpv.comli.com

Sucre, 2012

1. INTRODUCCIÓN

El objetivo de la asignatura es que al concluir la misma el estudiante esté capacitado para resolver problemas, mediante la elaboración de programas desarrollados aplicando, de manera eficaz y eficiente, los fundamentos de la programación estructurada en el análisis, diseño e implementación de los mismos.

En los primeros días de la informática los programas eran elaborados directamente en lenguaje de máquina (o en ensamblador). El lenguaje de máquina incentiva y prácticamente obliga al uso de saltos dentro del programa (hacia adelante y hacia atrás). Esta práctica suele da lugar a una lógica confusa que al ser representada en un diagrama revela una gran cantidad de líneas que se cruzan y entrelazan por lo que se conoce como "lógica tipo espagueti":



Puesto que la mayoría de los programadores surgieron de ese ambiente, los primeros lenguajes de alto nivel (como Fortran, Basic y C) tenían (y aún tienen) un comando para realizar este tipo de saltos: el comando "goto".

El problema surge cuando es necesario corregir, modificar y/o ampliar un programa, actividades inevitables en la producción de software y que, si no han sido convenientemente planificadas, consumen más tiempo y más recursos que la elaboración del producto original.

Para un mantenimiento eficiente del software es imprescindible que la lógica de los diferentes programas que lo componen sea fácilmente comprensible, pues de lo contrario resulta más práctico elaborar un nuevo programa que tratar de entender y luego corregir el programa existente.

Es con este objetivo, el de facilitar el mantenimiento de software, que surge la Programación Estructurada (PE), la cual recoge las prácticas de programación que habían demostrado ser exitosas en la elaboración y mantenimiento de programas y que se resumen en los siguientes principios:

- ***Dividir un problema complejo en problemas más sencillos.***
- ***Emplear estructuras estándar para construir la totalidad del programa.***
- ***Emplear tipos de datos a la medida.***

Si bien hoy en día los ambientes de desarrollo son en su mayoría orientados a objetos, los principios de la programación estructurada siguen siendo válidos dentro de los nuevos paradigmas y por lo tanto siguen siendo de utilidad práctica.

Pero la importancia de la programación estructurada no sólo es histórica, sino que sigue siendo de utilidad práctica, tanto así que, el núcleo de todos los sistemas operativos actuales, como Windows, Linux y Androide, son programa estructurados (no orientados a objetos).

1.1. LA PROGRAMACIÓN MODULAR (DESCENDENTE)

El primer principio, de la programación estructurada es conocido también como programación modular o programación descendente. Su aplicación es universal y expresa la misma intención que la frase "divide y vencerás". Básicamente nos dice que se debe dividir un problema en problemas más pequeños y estos a su vez en otros, hasta que los mismos sean lo suficientemente sencillos como para ser resueltos independientemente.

Cada uno de los problemas en los que es dividido el problema principal se conoce como módulo y es la razón por la cual este principio es conocido también con el nombre de "programación modular".

Es importante aclarar que un módulo es considerado como tal sólo si resuelve el problema de manera independiente, es decir sin importar de donde vengan los datos ni donde o como vayan a ser empleados los resultados devueltos.

Este principio se conoce también como programación descendente (top-down) porque se parte de un problema complejo que se divide en problemas más sencillos, estos en otros más sencillos y así sucesivamente, descomponiendo el problema de arriba (del problema más complejo) hacia abajo (al problema más sencillo). No obstante, la solución por lo general es ascendente, pues se comienza por resolver los problemas más sencillos y se va subiendo en dirección a los problemas más complejos.

El dividir el problema en problemas más sencillos, facilita considerablemente el mantenimiento de los programas y e incentiva la reutilización de código. Un módulo que resuelve un problema específico, al ser independiente, puede ser empleado no sólo en el software para el cual fue elaborado, sino también en cualquier otro en el que se requiera resolver ese tipo de problema, de esa manera surgen las librerías que contienen módulos que pueden ser reutilizados en varios programas, reduciendo considerablemente el tiempo de desarrollo.

Además los módulos facilitan enormemente el mantenimiento y detección de errores, porque si en la elaboración de un programa se emplean módulos previamente probados y se produce un error, se sabe que el error no está en dichos módulos, sino en los módulos añadidos, e inclusive es relativamente sencillo identificar el módulo problemático, con lo que la corrección se reduce a la corrección de un sólo módulo.

1.2. LAS ESTRUCTURAS ESTÁNDAR

El objetivo del segundo principio es el de promover la elaboración de programas que sean fáciles de comprender. Si en la construcción de un programa se emplean sólo estructuras estándar, resulta más fácil de comprender (y en consecuencia mantener) que cuando se emplean estructuras arbitrarias.

Las tres estructuras que deberían emplearse para construir cualquier programa (de acuerdo al teorema de la programación estructurada) son: a) la

secuencia (instrucciones consecutivas), b) la selección (if-then-else) y c) la iteración (while).

En ocasiones, no obstante, el uso exclusivo de estas tres estructuras puede dar lugar a lógicas confusas, lo que va en contra del objetivo de las mismas. Por esta razón la mayoría de los lenguajes cuentan con instrucciones que permiten modificarlas.

Con el propósito de conservar la claridad, que es el objetivo principal de este principio, en esta materia se considerará que una estructura es estándar si sólo tiene una entrada y una salida.

1.1. TIPOS DE DATOS A LA MEDIDA

El propósito de este principio es el de reducir posibles fuentes de error cuando se reciben (o introducen) datos, así como cuando se devuelven (o muestran) resultados. En la práctica la aplicación más importante de este principio radica en la validación de datos, es decir en permitir que el usuario introduzca únicamente datos válidos para el programa o módulo.

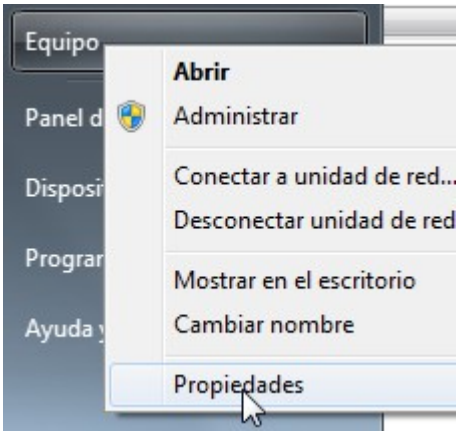
1.2. LENGUAJE DE PROGRAMACIÓN A EMPLEAR

Para que una metodología de programación sea de utilidad práctica tiene que ser implementada y para ello se debe emplear algún lenguaje de programación. El lenguaje de programación que se empleará en esta materia es C++.

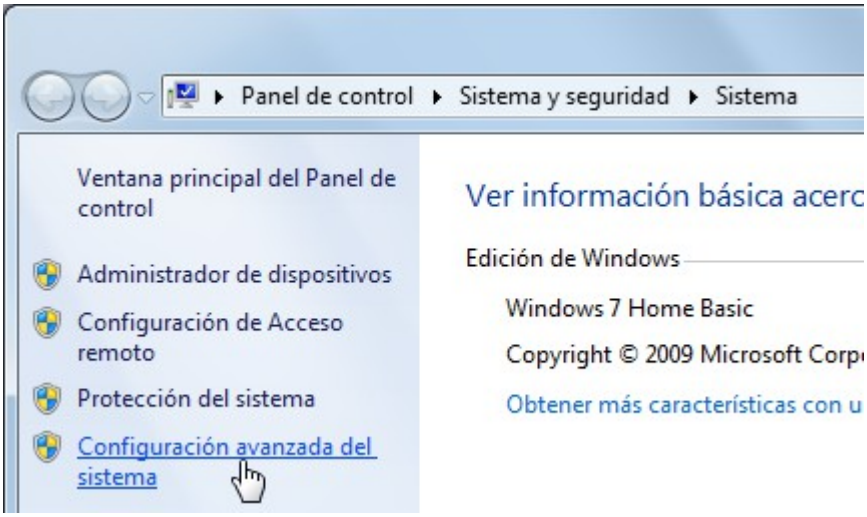
Como ambiente de desarrollo para C++, se empleará eclipse (www.eclipse.org). Eclipse trabaja por defecto con el lenguaje Java, pero existen distribuciones configuradas para otros lenguajes, incluida una distribución para C++, que es la que se empleará en la asignatura. Para compilar y hacer correr programas C++ en eclipse (o en cualquier otro IDE) se requiere de un compilador y el compilador que se empleará en la asignatura es MinGW (http://www.mingw.org/). Por comodidad el IDE, el compilador y los libros en formato digital han sido dejados en el centro de estudiantes de Ingeniería de Sistemas en la carpeta SIS101, pero los mismos pueden ser bajados también (gratuitamente) de Internet.

Para que Eclipse reconozca el compilador, y para que el mismo pueda ser empleado desde la ventada de comandos, es necesario modificar la variable "path" del sistema operativo, añadiendo al final del mismo, **teniendo mucho cuidado de no borrar nada**, el camino al directorio "bin", generalmente C:\MinGW\bin".

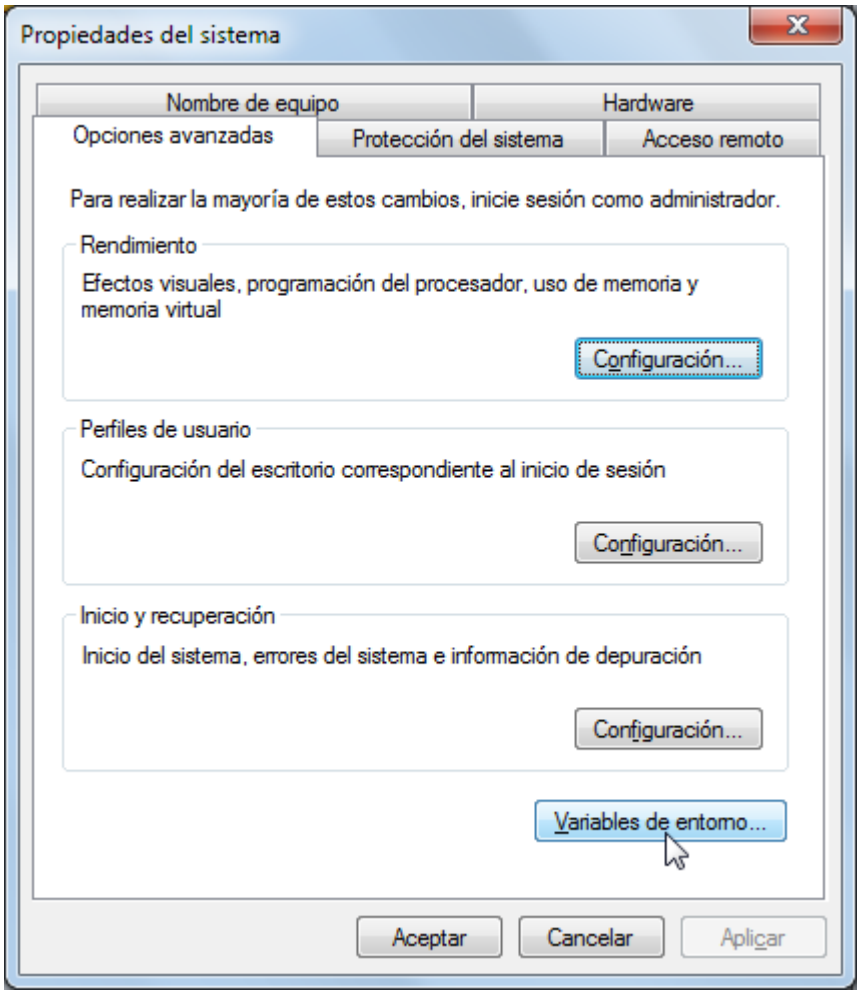
Con ese propósito, primero se hace click con el botón derecho del mouse en el botón (o icono) "Equipo" y se le elige la opción "Propiedades":



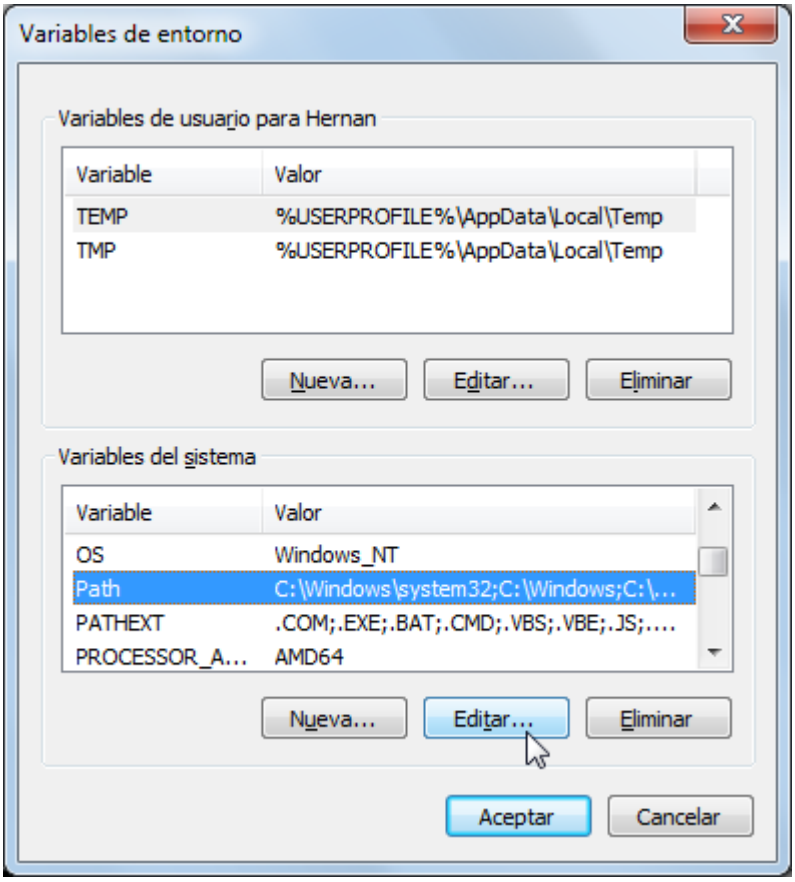
De esa manera se accede a la ventana del Sistema y en la misma se elige la opción "Configuración avanzada del sistema":



Con lo que aparece la ventana "Propiedades del sistema" y en la misma se hace clic en la opción "Variables de entorno":

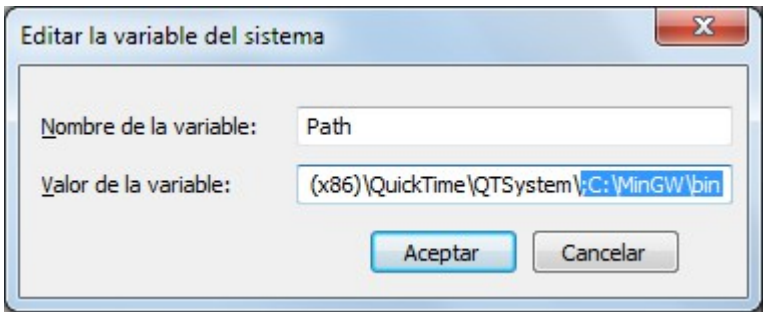


Entonces aparece la ventana "Variables de entorno" y en la misma, en el panel inferior, se busca la variable "Path" y se hace click en el botón "editar...":



Con lo que aparece la ventana "Editar la variable del sistema".

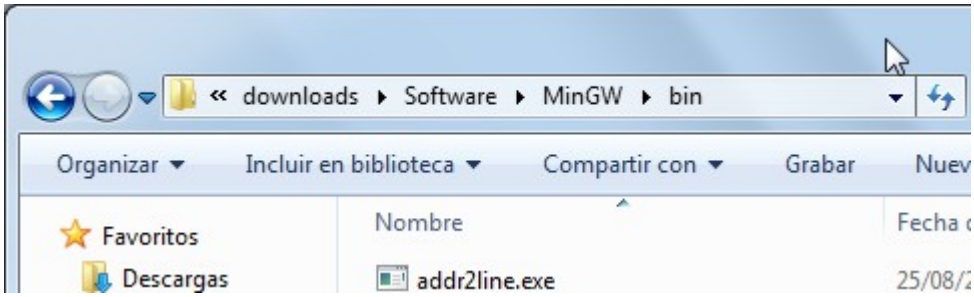
En el cuadro de edición inferior de dicha ventana se va al final del texto (tecla fin) y en esa parte se añade el camino para llegar al directorio "bin" del compilador "MinGW", precediendo el mismo con un punto y coma:



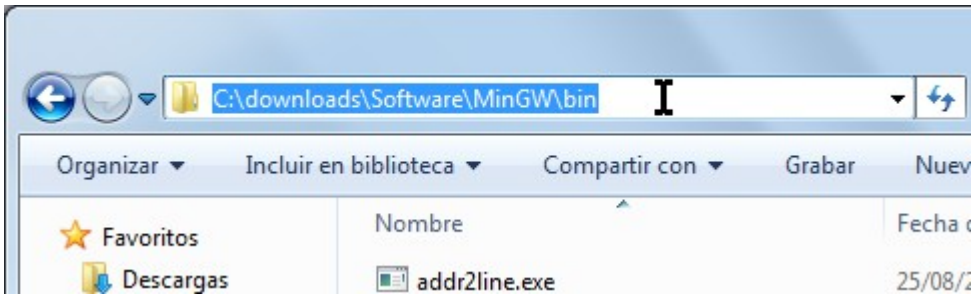
Se reitera que **se debe añadir el camino teniendo el cuidado de no borrar nada. Si se borra algo del camino varias aplicaciones de Windows (y Windows mismo) pueden quedar inservibles**.

Es importante tomar en cuenta también que se debe añadir el camino donde "realmente" se encuentra el directorio "bin" (dentro de la carpeta "MinGW"). Lo que se muestra en la anterior figura es sólo un ejemplo, y aunque es el caso más frecuente, no es una regla por lo que la carpeta "MinGW" puede encontrarse en cualquier otro disco y/o dentro de cualquier otra carpeta.

Por ejemplo si se ha descomprimido el compilador en el directorio "software" dentro de "downloads" en el disco "C", entonces el directorio "bin" se encontrará tal como se muestra en la siguiente figura:



El camino se obtiene haciendo click en la barra de navegación:



En este caso, este es el camino que se copiar (Ctrl+C) y pegar (Ctrl+V) al final de la variable "Path" (precediéndola con un punto y coma).

1.3. LOS PRIMEROS PROGRAMAS EN C++

C++ es un supraconjunto de C, es decir cuenta con todas las funciones y operadores de C, pero además cuenta con funciones y operadores propios de C++.

Desde el punto de vista conceptual, C++ está pensado en la programación orientada a objetos, mientras que C está pensado en la programación estructurada. Sin embargo, dado que C++ es un supraconjunto de C, permite también la programación estructurada.

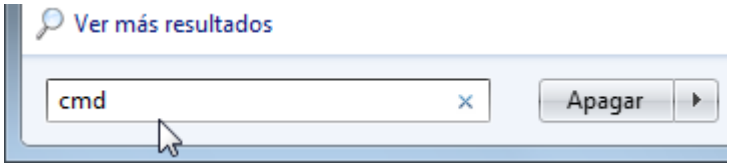
En este primer tema los programas serán elaborados empleando directamente el compilador (en la ventana de comandos) y un editor de textos básico (el bloc de notas de Windows). Esto con el propósito de comprender y aprender los pasos que realmente ocurren en la creación, compilación y ejecución de un programa, pasos que son realizados de forma transparente por los ambientes de desarrollo, por lo que en realidad no son aprendidos cuando los programas se elaboran exclusivamente en dichos ambientes.

El trabajar con estos recursos básicos no sólo permite comprender y aprender el proceso que realmente se lleva a cabo, sino que además, permite elaborar programas en dispositivos con limitados recursos de procesamiento y memoria. Sin embargo, no se pretende que todas las aplicaciones sean elaboradas exclusivamente con estos recursos. Los mismos son de utilidad práctica sólo en el desarrollo de aplicaciones pequeñas, como las que se elaboran en la presente materia. Para aplicaciones medianas y grandes es imprescindible el uso de ambientes de desarrollo y por esa razón, y como ya se dijo, en esta materia se empleará también el entorno de desarrollo Eclipse.

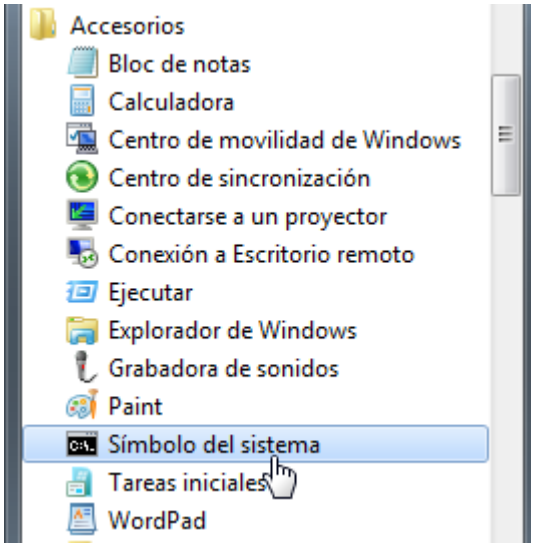
Desde un principio, aún en los programas más sencillos, se aplicarán los tres fundamentos de la programación estructurada, para que se adquiera la costumbre de emplearlos en la resolución de cualquier problema.

Como es usual se comenzará elaborando el programa "Hola Mundo", empleando primero las librerías y funciones ya conocidas de C, aunque compiladas con g++ (el compilador c++ de MinGW).

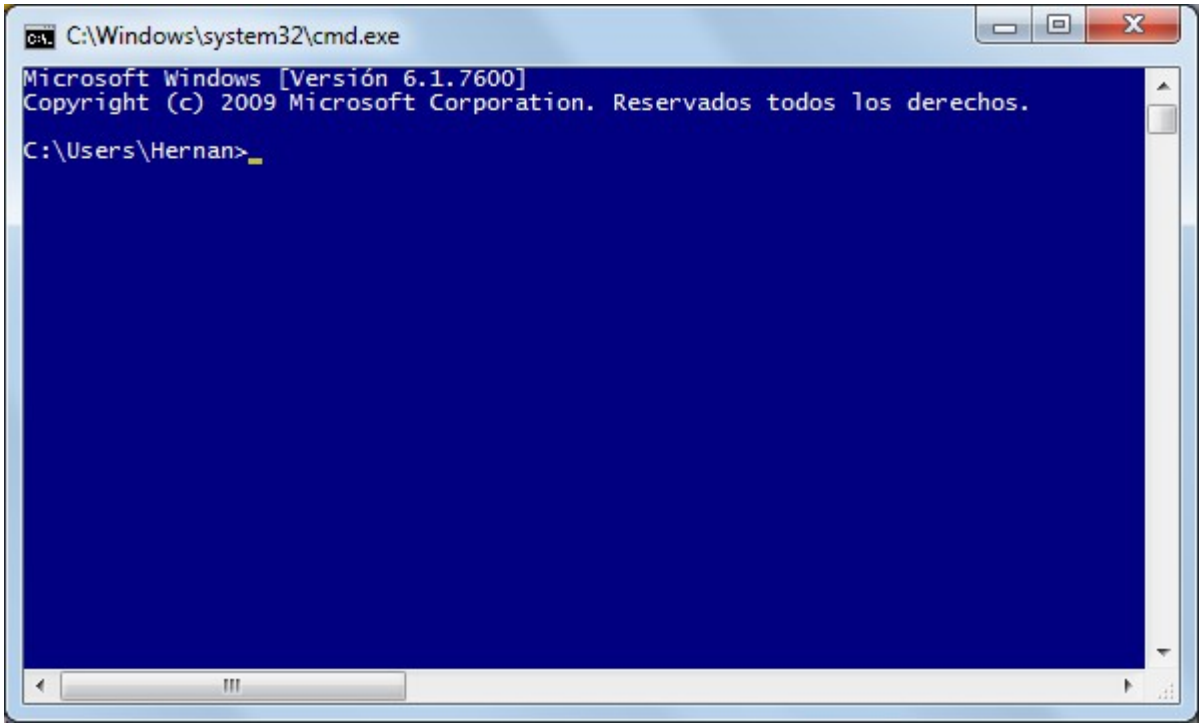
Para ello se abre una ventana de comandos escribiendo "cmd" en el cuadro de búsqueda de Windows:



O abriendo la aplicación "Símbolo del sistema" de la carpeta "Accesorios":



Con lo que aparece la ventana del símbolo del sistema:



El color, tamaño de la letra, tamaño de la pantalla, etc., son opciones que pueden ser modificadas en esta ventana. Lo primero que se recomienda hacer es ir al directorio en el cual se trabajará y en el mismo crear una

carpeta para los programas a elaborar, en este y posteriores ejemplos será "C:\HPV\programas\c++\pruebas":

```
C:\Users\Hernan>cd..

C:\Users>cd..

C:\>cd hpv

C:\HPV>cd programas

C:\HPV\programas>cd c++

C:\HPV\programas\c++>mkdir pruebas

C:\HPV\programas\c++>cd pruebas

C:\HPV\programas\c++\pruebas>
```

El comando "mkdir" (make directory) empleado en este ejemplo sirve para crear un directorio.

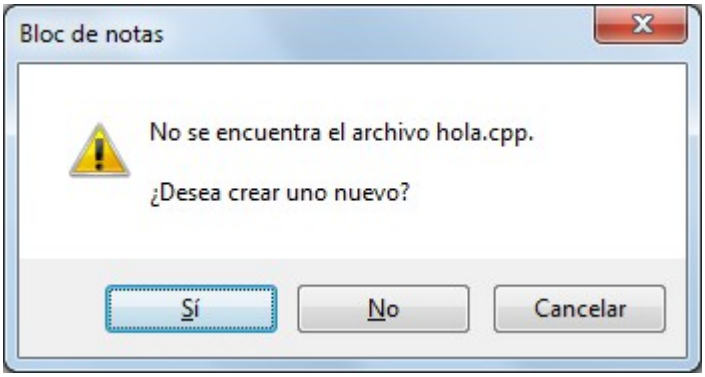
Se puede ir directamente a cualquier directorio escribiendo el camino completo después de "cd" (cambiar directorio), por ejemplo con las siguientes instrucciones se vuelve al directorio original "C:\Users\Hernan" y luego de vuelta a "C:\HPV\programas\c++\pruebas":

```
C:\HPV\programas\c++\pruebas>cd c:\Users\Hernan

c:\Users\Hernan>cd C:\HPV\programas\c++\pruebas

C:\HPV\programas\c++\pruebas>
```

Una vez en el directorio de trabajo se crea el programa fuente (hola.cpp) empleando el bloc de notas (notepad) de Windows:

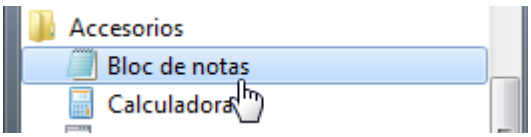
```
C:\HPV\programas\c++\pruebas>notepad hola.cpp_
```

Con lo que arranca "notepad" y pregunta si se quiere crear el archivo "hola.cpp":

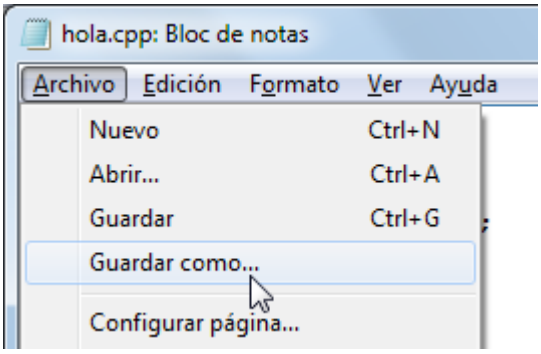


Se hace clic en "Si" y entonces se puede comenzar a escribir el programa en "notepad".

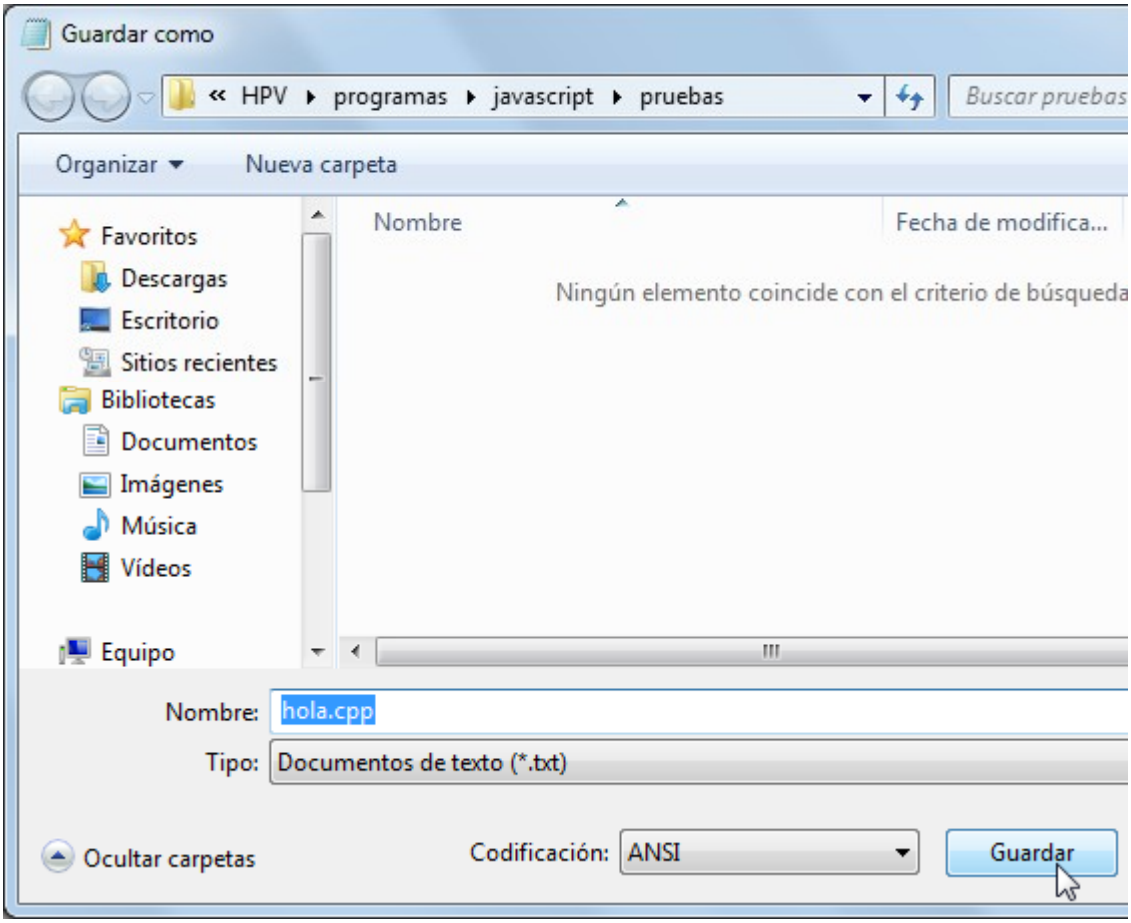
Alternativamente, se puede abrir el programa "notepad":



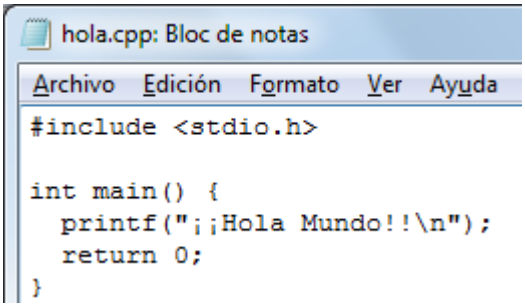
Y luego guardar el archivo:



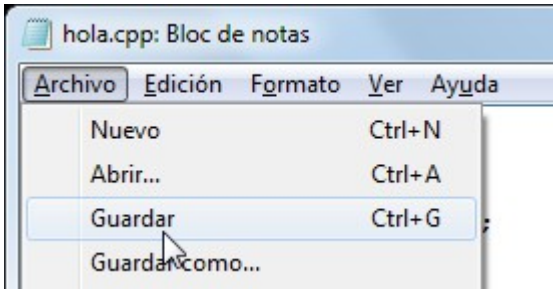
En el directorio creado previamente:



Una vez creado el archivo (con cualquiera de los métodos explicados), se escribe el programa:



Y se guarda:



Entonces puede ser compilado volviendo a la ventana de comandos y escribiendo la instrucción:

```
C:\HPV\programas\c++\pruebas>g++ -c hola.cpp
hola.cpp:6:2: warning: no newline at end of file
```

Como se puede observar, el compilador da una advertencia indicando que el archivo no tiene salto de línea al final del archivo (fila 6, columna 2), algo que requiere el compilador y que se puede arreglar simplemente añadiendo dicha línea (con la tecla Enter):

```
    printf("¡¡Hola Mundo!!\n");
    return 0;
}
```

Y guardando nuevamente el programa. Ahora al compilar ya no se produce ninguna advertencia o error (se aclara que una advertencia es sólo un posible error, pero no necesariamente es uno):

```
C:\HPV\programas\c++\pruebas>g++ -c hola.cpp
```

Si se ve la lista de archivos existentes en el directorio se obtiene:

```
C:\HPV\programas\c++\pruebas>dir
 El volumen de la unidad C es Acer
 El número de serie del volumen es: 9A51-192F

 Directorio de C:\HPV\programas\c++\pruebas

08/03/2012  09:09 p.m.    <DIR>          .
08/03/2012  09:09 p.m.    <DIR>          ..
08/03/2012  09:09 p.m.                83 hola.cpp
08/03/2012  09:09 p.m.               574 hola.o
               2 archivos            657 bytes
               2 dirs  334,358,310,912 bytes libres
```

Y, como se puede ver, ahora existe el archivo "hola.o", que es el archivo compilado y que se conoce como programa objeto.

Ahora se crea el ejecutable, que tendrá el nombre "hola.exe", escribiendo la instrucción:

```
C:\HPV\programas\c++\pruebas>g++ hola.o -o hola
```

Que no reporta ningún error. Si ahora se ve la lista de archivos:

```
C:\HPV\programas\c++\pruebas>dir
08/03/2012  09:14 p.m.    <DIR>          .
08/03/2012  09:14 p.m.    <DIR>          ..
08/03/2012  09:09 p.m.                83 hola.cpp
08/03/2012  09:14 p.m.            15,852 hola.exe
08/03/2012  09:09 p.m.               574 hola.o
               3 archivos         16,509 bytes
```

Se puede observar que se ha creado el archivo "hola.exe" que es el archivo ejecutable.

Finalmente se hace correr el programa escribiendo el nombre del archivo ejecutable:

```
C:\HPV\programas\c++\pruebas>hola
ííHola Mundo!!
```

Y como era de esperar, se obtiene el mensaje "¡¡Hola Mundo!!", con los dos primeros caracteres cambiados debido a una diferencia en los conjuntos de caracteres empleados por la ventana de comandos y el compilador.

De esa manera se ha escrito (programa fuente), compilado (programa objeto) y hecho correr (programa ejecutable) el primer programa en C++, pero en el mismo se han empleado únicamente librearías y comandos propios de C, lo que demuestra que C es un subconjunto de C++ y que en consecuencia es posible escribir, compilar y hacer correr programas C empleando un compilador C++.

Si bien en C++ es posible crear programas empleando librerías C, tal como se ha demostrado en el anterior ejemplo, se recomienda emplear las nuevas librerías "C" actualizadas para C++. Dichas librerías tienen los mismos nombres que las librerías originales, pero su nombre está precedido por una "c" y ya no tienen la extensión "h". Por ejemplo el anterior programa empleando estas nuevas librerías es:

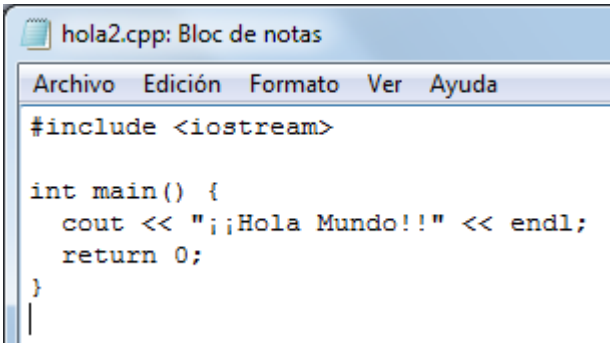
```
#include <cstdio>

int main() {
  printf("¡¡Hola Mundo!!\n");
  return 0;
}
```

Que se compila, enlaza (se crea el ejecutable) y se hace correr exactamente igual que con la librería original (<stdio.h>).

En lo futuro se emplearan estas nuevas librerías cuando se escriban programas que requieran librerías C.

Ahora se vuelve a escribir el programa pero empleando librerías C++ propiamente:

```
C:\HPV\programas\c++\pruebas>notepad hola2.cpp
```



Después de guardar el código y compilar el programa se obtiene:

```
C:\HPV\programas\c++\pruebas>g++ -c hola2.cpp
hola2.cpp: In function `int main()':
hola2.cpp:4: error: `cout' was not declared in this scope
hola2.cpp:4: error: `endl' was not declared in this scope
```

Donde el compilador informa que no conoce "cout" ni "endl", a pesar de que ambas instrucciones se encuentran en la librería <iostream>. Esto se debe a que en C++ los nombres de todas librerías estándar se encuentran en el espacio de nombres "std", por lo que es necesario incluir el mismo en el programa, lo que se hace con la instrucción "using namespace std":

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

int main() {
  cout << "¡¡Hola Mundo!!" << endl;
  return 0;
}
```

Ahora al compilar el programa no se produce ningún error, por lo que se puede crear (enlazar) y hacer correr el ejecutable:

```
C:\HPV\programas\c++\pruebas>g++ -c hola2.cpp

C:\HPV\programas\c++\pruebas>g++ hola2.o -o hola2

C:\HPV\programas\c++\pruebas>hola2
ííHola Mundo!!
```

Obteniéndose, como era de esperar, la misma salida que con el programa anterior.

En este programa, la instrucción "cout" sirve para mostrar en pantalla el texto (o dato) que se escribe a la derecha del símbolo "<<". Para cada nuevo dato, o comando, se debe emplear un nuevo "<<" y se pueden emplear tantos "<<" como se requiera.

La instrucción "endl", es un manipulador (un comando) que le indica a "cout" que inserte un salto de línea (es el equivalente a la instrucción "\n" de C, la cual por cierto puede ser empleada también con "cout").

1.4. EJEMPLOS

1. Elabore un programa que calcule y muestre el cubo de un número.

Como se sabe, para calcular el cubo de un número simplemente se multiplica el número 3 veces, por lo que el problema puede ser resuelto muy fácilmente. Sin embargo, y como se dijo, en la resolución de todos los problemas en esta materia, aún en uno tan sencillo como el presente, se aplicarán los tres fundamentos de la programación estructurada.

En ese sentido, primero se divide el problema en problemas más pequeños. Para ello se identifican las tareas (funciones) que se deben llevar a cabo para resolver el problema y que pueden ser realizadas independientemente: a) En primer lugar, para calcular el cubo de un número se debe leer dicho número y ese es un problema que puede ser resuelto independientemente, pues se puede leer un número sin importar en qué vaya a ser empleado. b) Una vez que se tiene el número el mismo debe ser multiplicado 3 veces para obtener el cubo, y esa es otra tarea (otra función) que puede ser resuelta independientemente: no importa de donde viene el número ni en qué vaya a ser empleado el resultado calculado. c) Finalmente el resultado debe ser mostrado en pantalla y esa es otra tarea que puede ser resuelta independientemente: no importa de donde vienen el número ni el cubo del mismo.

De esa manera, aplicando el primer principio de la programación estructurada, se ha dividido el problema en tres problemas más sencillos.

Para aplicar el segundo principio, se debe tener el cuidado de emplear sólo estructuras estándar al escribir el código. En un problema tan sencillo como el presente, sólo se requieren instrucciones consecutivas, por lo que la única estructura estándar requerida es la secuencia y es la que se empleará en la programación de todos los módulos.

A estas alturas, el tercer principio sólo puede ser aplicado parcialmente (pues para validar correctamente los datos se requieren las estructuras selectivas y/o iterativas). En este caso sólo se tiene un dato (el número) y un resultado (el cubo del número) y al no existir información adicional (no se sabe si el número es entero o real, ni si existe un valor máximo o mínimo), se opta por emplear un tipo real, pues los reales incluyen a los enteros y dentro de los reales se elige el tipo "double" porque es lo suficientemente grande como para manejar la mayoría de los números más frecuentes.

Ahora se crea el archivo ejem1.cpp:

```
C:\HPV\programas\c++\pruebas>notepad ejem1.cpp
```

En el mismo se escribe el programa fuente:

```
#include <cstdio>

double leerDato() {
  double x;
  printf("Escriba un número: ");
  scanf("%lf",&x);
  return x;
}

double cubo(double x) {
  return x*x*x;
}

void mostrarRes(double x, double r) {
  printf("El cubo de %g es %g\n",x,r);
}

int main() {
  double x,r;
  x = leerDato();
  r = cubo(x);
  mostrarRes(x,r);
  return 0;
}
```

Entonces se compila, enlaza y hace correr el programa:

```
C:\HPV\programas\c++\pruebas>g++ -c ejem1.cpp

C:\HPV\programas\c++\pruebas>g++ ejem1.o -o ejem1

C:\HPV\programas\c++\pruebas>ejem1
Escriba un n·mero: 3
El cubo de 3 es 27
```

Con lo que se obtiene el resultado esperado.

Ahora se vuelve a resolver el problema pero empleando las librerías e instrucciones propias de C++:

```
C:\HPV\programas\c++\pruebas>notepad ejem1b.cpp
```

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

double leerDato() {
  double x;
  cout << "Escriba un número: ";
  cin >> x;
  return x;
}

double cubo(double x) {
  return x*x*x;
}

void mostrarRes(double x, double r) {
  cout << "El cubo de "<< x << " es: " << r << endl;
}

int main() {
  double x,r;
  x = leerDato();
  r = cubo(x);
  mostrarRes(x,r);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem1b.cpp

C:\HPV\programas\c++\pruebas>g++ ejem1b.o -o ejem1b

C:\HPV\programas\c++\pruebas>ejem1b
Escriba un n·mero: 3
El cubo de 3 es: 27
```

Y como era de esperar se obtiene el mismo resultado.

2. Elabore un programa que lea un número y devuelva el logaritmo en base 7 del mismo.

Al igual que el anterior ejemplo, lo primero que se debe hacer es aplicar el primer principio de la programación estructurada. En este problema se pueden identificar tres módulos: a) Un módulo que lee el número, b) Un módulo que calcula el logaritmo en base 7 y c) Un módulo muestra el resultado.

En cuanto al segundo principio, sólo se emplea una estructura: la secuencia.

En lo relativo al tercer principio y por la misma razón que en el ejemplo anterior, una vez más se emplea el tipos de dato "double" para garantizar que el programa funcione tanto con números enteros como reales (relativamente grandes).

En cuanto al problema en sí, el logaritmo en base 7 se calcula dividiendo el logaritmo natural del número entre el logaritmo natural de 7.

La solución, empleando librerías C (con el nuevo formato), es:

```
C:\HPV\programas\c++\pruebas>notepad ejem2.cpp
```

```
#include <cstdio>
#include <cmath>

double leerNumero(){
  double n;
  printf("Numero? ");
  scanf("%lf",&n);
  return n;
}

double log7(double x){
  return log(x)/log(7);
}

void mostrarResultado(double x, double r){
  printf("log(%.16g) = %.16g\n",x,r);
}

int main(){
  double x,r;
  printf("***** Calculo del logaritmo en base 7 *****\n");
  x=leerNumero();
  r=log7(x);
  mostrarResultado(x,r);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem2.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\pruebas>ejem2
***** Calculo del logaritmo en base 7 *****
Numero? 6.7
log(6.7) = 0.9774899048244042
```

Y la solución empleando librerías C++ es:

```
C:\HPV\programas\c++\pruebas>notepad ejem2.cpp
```

```
#include <iostream>
#include <cmath>
using namespace std;

double leerNumero(){
  double n;
  cout<<"Numero? ";
  cin>>n;
  return n;
}

double log7(double x){
  return log(x)/log(7);
}

void mostrarResultado(double x, double r){
  cout.precision(16);
  cout<<"log("<<x<<") = "<<r<<endl;
```

```
}

int main(){
  double x,r;
  cout<<"***** Calculo del logaritmo en base 7 *****"<<endl;
  x=leerNumero();
  r=log7(x);
  mostrarResultado(x,r);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem2b.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem2b ejem2b.o

C:\HPV\programas\c++\pruebas>ejem2b
***** Calculo del logaritmo en base 7 *****
Numero? 6.7
log(6.7) = 0.9774899048244042
```

La instrucción cout.precision(16), de la función "mostrarResultados", le indica a C++ que muestre los resultados con hasta 16 dígitos de precisión, es el equivalente a la instrucción ".16g" de printf (que igualmente le indica a C que muestre los resultados con hasta 16 dígitos de precisión).

3. Elabore un programa que lea el nombre, celular y correo electrónico de una persona y muestre dicha información en pantalla.

De acuerdo al primer principio, se pueden identificar en este caso sólo dos módulos: Un módulo para leer los datos y Otro para mostrarlos. Una vez más sólo se requiere una estructura: la secuencia y para el tercer principio se debe tomar en cuenta que dos de los datos deben ser de tipo texto (char []) y uno entero sin signo (unsigned long int).

La solución, empleando las librearías C, es:

```
C:\HPV\programas\c++\pruebas>notepad ejem3.cpp
```

```
#include <cstdio>

void leerDatos(char nombre[], unsigned long int &celular,
  char correo[]){
  printf("%15s ","Nombre?");
  scanf("%s",nombre);
  printf("%15s ","Celular?");
  scanf("%lu",&celular);
  printf("%15s ","Correo?");
  scanf("%s",correo);
}

void mostrarDatos(char nombre[], unsigned long int celular,
  char correo[]){
  printf("Datos leidos: \n");
  printf("%13s : %s\n","Nombre",nombre);
  printf("%13s : %lu\n","Celular",&celular);
  printf("%13s : %s\n","Correo",correo);
}

int main(){
  char nombre[30], correo[30];
```

```
    unsigned long int celular;
    leerDatos(nombre,celular,correo);
    mostrarDatos(nombre,celular,correo);
    return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem3.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem3 ejem3.o

C:\HPV\programas\c++\pruebas>ejem3
        Nombre? Carlos
       Celular? 7654321
        Correo? carlitos_bolivia@gmail.com
Datos leidos:
       Nombre : Carlos
      Celular : 2686660
       Correo : carlitos_bolivia@gmail.com
```

Observe que en la función leerDatos se ha escrito un "&" delante del parámetro "celular", se procede así para declarar dicho parámetro por referencia. Un parámetro por referencia reciben la variable en sí (no sólo su valor), por lo que cualquier modificación al valor de dicha variable (dentro de la función) es una modificación a la variable original.

Se emplean parámetros por referencia cuando, como en este caso, se devuelven resultados en dichos parámetros. Así, en este ejemplo se devuelve el número de celular leído.

Sin embargo, los parámetros "nombre" y "correo", donde se devuelven el nombre y correo leídos, no están como parámetros por referencia (no tienen el "&" delante de sus nombres). Se procede así porque dichos parámetros son vectores (por eso tienen corchetes después del nombre) y los vectores (arrays) en C y C++ son punteros.

Ahora bien, un puntero es una variable que almacena una dirección de memoria (donde existe algún tipo de información) es decir que un puntero "apunta" (tiene la dirección) donde se encuentra la información, pero no la información en sí. Por lo tanto, cuando se modifica la información a la que apunta un puntero, se está modificando la información original. En este sentido, los punteros se comportan de forma similar a los parámetros por referencia y cuando se trabaja con cadenas de caracteres (char variable[] o char *variable) se puede pensar en los mismos como si se tratara de parámetros por referencia.

Algunas funciones C, como "scanf", reciben sus parámetros como punteros. Esa es la razón por la que se precede con un "&" las variables que se mandan a "scanf" (como ocurre con la variable "celular" en la función "leerDatos"). El operador "&" sirve para obtener la dirección de memoria de la variable a la que precede (valor que puede ser asignado a un puntero). En este ejemplo no se obtienen las direcciones de memoria de "nombre" y "celular" (no tienen un "&" delante) porque (como se ha explicado) dichas variables son de hecho direcciones de memoria (punteros).

Los punteros serán estudiados con más detalle en temas posteriores, por el momento, es suficiente recordar que un puntero es una dirección de memoria (donde se encuentra algún tipo de información), que se obtiene la dirección de memoria de una variable con el operador "&" y que los vectores, en C y C++, son también punteros.

Por otra parte, en "printf" de las funciones "leerDatos" y "mostrarDatos", se han empleado los especificadores de formato "%15s", y "%13s". Como se recordará, dichos formatos le indica a C que muestre el texto ("s") en un ancho de 15 y 13 caracteres alineados a la derecha (re-

cuerde que para alinear a la izquierda el número debe ser negativo). Es esta orden la que permite leer y mostrar los valores alineados.

La solución empleando librerías C++ es:

```
C:\HPV\programas\c++\pruebas>notepad ejem3b.cpp
```

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

void leerDatos(string &nombre, unsigned long int &celular,
  string &correo){
  cout.width(15);
  cout<<"Nombre? ";
  cin>>nombre;
  cout.width(15);
  cout<<"Celular? ";
  cin>>celular;
  cout.width(15);
  cout<<"Correo? ";
  cin>>correo;
}

void mostrarDatos(string nombre, unsigned long int celular,
  string correo){
  cout<<"Datos leidos: "<<endl;
  cout.width(15);
  cout<<"Nombre : "<<nombre<<endl;
  cout.width(15);
  cout<<"Celular : "<<celular<<endl;
  cout.width(15);
  cout<<"Correo : "<<correo<<endl;
}

int main(){
  string nombre,correo;
  unsigned long int celular;
  leerDatos(nombre,celular,correo);
  mostrarDatos(nombre,celular,correo);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem3b.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem3b ejem3b.o

C:\HPV\programas\c++\pruebas>ejem3b
       Nombre? Carlos
      Celular? 7654321
       Correo? carlitos_bolivia@gmail.com
Datos leidos:
      Nombre : Carlos
     Celular : 7654321
      Correo : carlitos_bolivia@gmail.com
```

Como se puede observar, en C++, se puede emplear el tipo "string" cuando se trabaja con texto (en lugar de "char *" o "char []"). Este tipo de dato permite tratar las cadenas de caracteres como datos simples y es por ello que en la función "leerDatos" se reciben todos los datos por referencia.

Para dar formato tanto a la introducción como a la presentación de resul-

tados se emplea "cout.with()" que viene a ser el equivalente a "%ancho" de la instrucción "printf" de C.

4. Elabore un programa que reciba el radio de un círculo y devuelva su perímetro.

En este caso, al aplicar el primer principio, se identifican 3 módulos: a) Uno para leer el dato (el radio); b) Otro para calcular el perímetro y c) Otro para mostrar el valor calculado.

Una vez más, en lo referente al segundo principio, sólo existe una estructura estándar: la secuencia.

Para el tercer principio, se emplea el tipo de dato "double" pues tiene la precisión suficiente como para dar resultados aceptables en la mayoría de los casos.

En cuanto al problema en si, el perímetro de un círculo se calcula con:

$$perímetro = 2 \cdot pi \cdot r$$

Por lo que la solución del problema consiste simplemente en calcular y devolver el valor de esta expresión.

La solución, empleando librerías C, es:

```
C:\HPV\programas\c++\pruebas>notepad ejem4.cpp
```

```
#include <cstdio>
#include <cmath>

double leerRadio(){
  double r;
  printf("%20s ","Radio?");
  scanf("%lf",&r);
  return r;
}

double perimetro(double r){
  return 2*M_PI*r;
}

void mostrarPerimetro(double p){
  printf("%20s %.5g\n","El perimetro es:",p);
}

int main(){
  double r,p;
  printf("***** Perimetro de un circulo *****\n");
  r=leerRadio();
  p=perimetro(r);
  mostrarPerimetro(p);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem4.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem4 ejem4.o

C:\HPV\programas\c++\pruebas>ejem4
***** Perimetro de un circulo *****
              Radio? 3.25
    El perimetro es: 20.42
```

Y la solución, empleando librerías C++, es:

```
C:\HPV\programas\c++\pruebas>notepad ejem4b.cpp
```

```
#include <iostream>
#include <cmath>
using namespace std;

double leerRadio(){
  double r;
  cout.width(19);
  cout<<"Radio? ";
  cin>>r;
  return r;
}

double perimetro(double r){
  return 2*M_PI*r;
}


void mostrarPerimetro(double p){
  cout.width(19);
  cout.precision(5);
  cout<<"El perimetro es: "<<p<<endl;
}

int main(){
  double r,p;
  cout<<"***** Perimetro de un circulo *****"<<endl;
  r=leerRadio();
  p=perimetro(r);
  mostrarPerimetro(p);
  return 0;
}
```

```
C:\HPV\programas\c++\pruebas>g++ -c ejem4b.cpp

C:\HPV\programas\c++\pruebas>g++ -o ejem4b ejem4.o

C:\HPV\programas\c++\pruebas>ejem4b
***** Perimetro de un circulo *****
              Radio? 3.25
    El perimetro es: 20.42
```

1.5. EJERCICIOS

Los ejemplos y ejercicios correspondientes al tema 1 deben ser presentados en la carpeta tp_tema1 y cada ejemplo y ejercicio debe ser nombrado claramente (ejem1, ejem2, …, ejer1, ejer1, …). En este tema, todos los ejemplos y ejercicios deben ser creados desde la línea de comando, el programa debe ser escrito en "notepad", compilado y enlazado con g++ y probado en la línea de comando.

1. Elabore un programa que empleando librerías C muestre el mensaje "Hola Mundo" en la primera línea y el mensaje "¡Bienvenido a la programación estructurada!" en la segunda.
2. Elabore un programa que empleando librerías tipo C++ muestre su nombre en la primera línea y sus apellidos en la segunda.

3. Elabore un programa que empleando librerías tipo C muestre en una línea el nombre de la facultad y en la segunda el nombre de la carrera.
4. Elabore un programa que empleando librerías tipo C++ muestre en pantalla la figura de un árbol de navidad, dibujado con el carácter asterisco "*", empleando 5 filas para el follaje, 2 para el tronco y uno para el pedestal.
5. Elabore un programa que empleando librerías C reciba un número y muestre el cuadrado del mismo.
6. Elabore un programa que empleando librerías tipo C++ reciba un número y muestre el logaritmo en base 5 del mismo.
7. Elabore un programa que empleando librerías C reciba el nombre, apellido y dirección de una persona y muestre dicha información en pantalla.
8. Elabore un programa que empleando librerías tipo C++ reciba el nombre, sexo y edad de una persona y muestre dicha información en pantalla.
9. Elabore un programa que empleado librerías tipo C reciba el radio de un círculo y muestre (con 12 dígitos de precisión) el área del mismo.
10. Elabore un programa que empleando librerías tipo C++ reciba el radio y altura de un cilindro y muestre (con 9 dígitos de precisión) el volumen del mismo.

2. SECUENCIA

En este tema se introduce formalmente la primera estructura estándar: "la secuencia" y se adquiere más práctica en su aplicación.

2.1. DIAGRAMAS DE ACTIVIDADES

Antes de comenzar el estudio de la secuencia se introduce la simbología que se empleará para expresar la lógica que resuelve un problema (los algoritmos): los *diagramas de actividades.*

Los algoritmos describen la secuencia lógica de acciones que deben llevarse a cabo para resolver un determinado problema. Existen varias formas en las que se pueden expresar los algoritmos, de ellas emplearemos los *diagramas de actividades*, el lenguaje hablado y el código documentado.

Los *diagramas de actividades* constituyen una de las varias clases de diagramas que forman parte de *UML*, el *Lenguaje de Modelado Unificado*, que actualmente es el lenguaje estándar para el modelado de sistemas de software. Se ha elegido este tipo de diagramas no sólo por ser un estándar, sino porque permiten expresar los algoritmos de manera clara, ordenada, sencilla y porque además son muy versátiles, permitiendo representar diferentes tipos de estructuras.

El inicio de un diagrama de actividades se representa con un círculo relleno:

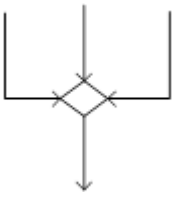
El final del diagrama de actividades se representa con un círculo que contiene un círculo relleno en su interior:

Una acción (o sentencia) se representa con un rectángulo con sus bordes redondeados:

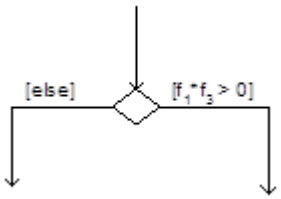
Un flujo, que es el símbolo empleado para unir los elementos del diagrama y señalar la dirección en que se deben seguir las acciones, se representa por una flecha continua abierta:

Una unión o bifurcación se representa por un pequeño rombo:

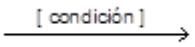
Cuando se emplea como unión llegan dos o más flujos y sale uno.



Cuando se emplea como bifurcación ingresa un flujo y salen dos o más acompañados de alguna condición.



Una condición, tal como se puede observar en la anterior figura, se representa como texto encerrado entre corchetes y siempre está relacionado a algún flujo:



El texto de la condición puede ser una expresión matemática, una expresión relacional, una condición textual, etc. Para la condición por defecto se puede emplear *[else]* (caso contrario).

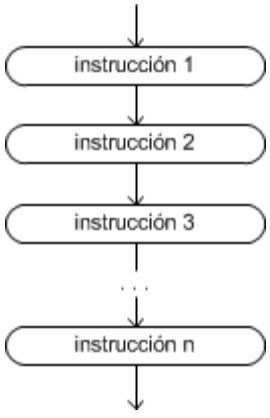
Una actividad representa un conjunto de acciones (una subrutina) que se detalla luego por separado empleando otro diagrama de actividades. Se representa como una acción con un icono de una acción llamando a otra en la parte inferior derecha:

Las notas se emplean para comentar y documentar los diagramas de actividades. Se representan como una hoja con una esquina doblada y asociada, mediante una línea discontinua, a cualquiera de los elementos de un diagrama de actividades:

Al elaborar un diagrama de actividades se debe tener presente que es un lenguaje y como tal es necesario respetar su sintaxis (es decir los símbolos) pues cada uno de ellos tiene su propio significado y si se cambia, cambia también la lógica del algoritmo. Por ejemplo, para representar un flujo siempre se debe emplear la flecha continua abierta, no una flecha cerrada y rellena: ⟶ , una flecha cerrada no rellena: ⟶ , o una flecha discontinua abierta: ⇢ , pues cada una de ellas tienen un significado muy diferente (mensaje, herencia y dependencia respectivamente).

2.2. DIAGRAMA DE ACTIVIDADES DE UNA SECUENCIA

El diagrama de actividades de una secuencia tiene la siguiente forma:



Como se ve corresponde a la ejecución de instrucciones consecutivas. En C/C++, para que una secuencia corresponda a una estructura y se ejecute como un bloque, debe estar encerrada entre llaves:

```
{
  instrucción 1;
  instrucción 2;
  instrucción 3;
  . . .
  instrucción n;
}
```

2.3. TIPOS DE DATOS

Puesto que son importantes en la aplicación del tercer principio de la programación estructurada (emplear tipos de datos a la medida), en este tema se realiza un breve repaso de los tipos de datos básicos disponibles en C/C++.

Cinco son los tipos de datos básicos disponibles en C/C++: chr, int, float, double y void. Estos cinco tipos conjuntamente los modificadores "signed", "unsigned", "long" y "short" dan lugar al siguiente conjunto de tipos de datos:

Tipo	Tamaño (bits)	Rango
char	8	-127 - 127
unsigned char	8	0 - 255
signed char	8	-127 - 127
int	32	-2147483647 - 2147483647
unsigned int	32	0 - 4294967295
signed int	32	-2147483647 - 2147483647
short int	16	-32767 - 32767
unsigned short int	16	0 - 65535
signed short int	16	-32767 -32767
long int	32	-2147483647 - 2147483647
signed long int	32	-2147483647 - 2147483647
unsigned long int	32	0 - 4294967295
float	32	6 dígitos de precisión
double	64	10 dígitos de precisión
long double	80	10 dígitos de precisión

Los tamaños y rangos dados son sólo aproximados, pues dependen del procesador y compilador que se esté empleando.

Además de estos tipos básicos, en C++ se disponen de los tipos "bool" para valores lógicos (con los valores true=verdadero y false=falso) y "string" para cadenas de caracteres.

2.4. OPERADORES

Puesto que también son de uso frecuente se hace un repaso de los operadores básicos disponibles en C/C++, los cuales son:

Operador	Operación
+	Suma
-	Resta, menos unario
*	Multiplicación
/	División
%	Residuo
++	Incremento

--	Decremento

Se recuerda que si los operadores de incremento preceden a la variable, primero incrementan (o disminuyen) su valor y luego se emplea el nuevo valor en las operaciones, mientras que si están después de la variable, primero se emplea su valor original en las operaciones y luego recién se incrementa (o disminuye) su valor.

A más de los operadores básicos, se pueden emplear los operadores de asignación compuestos (donde "x" es 9 y "y" es 3):

Operador	Ejemplo	Equivalente a	Resultado (x)
+=	x+=y	x=x+y	12
-=	x-=y	x=x-y	6
*=	x*=y	x=x*y	27
/=	x/=y	x=x/y	3
%=	x%=y	x=x%y	0

Dado que la lógica involucrada en la solución de problemas secuenciales es muy simple, no se elaborarán diagramas de flujo para representar la lógica.

2.5. EJEMPLOS

1. Elabore un programa que reciba los lados de un triángulo y devuelva el perímetro y área del mismo. Los resultados deben ser mostrados con 10 dígitos de precisión.

Las fórmulas para calcular el perímetro y área de un triángulo son:

$$perímetro = a + b + c$$

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

$$s = \frac{1}{2} perímetro$$

A pesar de que los problemas secuenciales son básicos, se pueden cometer errores al evaluar las instrucciones en el orden equivocado. Así en este ejemplo para calcular el área primero se debe calcular el parámetro "s" y luego el área, no al revés, porque entonces "s" tendría un valor arbitrario con el que se obtendría un resultado erróneo.

En este problema se pueden identificar cinco módulos: a) Un módulo que lee tres números (los lados del triángulo), b) Un módulo que calcula el perímetro del triángulo, c) Un módulo que calcula el área del triángulo, d) un módulo que muestra el perímetro del triángulo y e) Un módulo que muestra el área del triángulo.

En cuanto al tercer principio se emplearán tipos de datos "double", para garantizar que el programa funcione tanto con números enteros como reales.

La solución, empleando librerías C, es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem1a.cpp
```

```
#include <cstdio>
#include <cmath>

void leerLados(double &a, double &b, double &c){
  printf("%25s? ","Lado1");
  scanf("%lf",&a);
  printf("%25s? ","Lado2");
```

```
  scanf("%lf",&b);
  printf("%25s? ","Lado3");
  scanf("%lf",&c);
}

double perimetro(double a, double b, double c){
  return a+b+c;
}

double area(double a, double b, double c){
  double s,r;
  s=perimetro(a,b,c)/2;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}

void mostrarPerimetro(double p){
  printf("%25s: %.10g\n","Perimetro",p);
}

void mostrarArea(double a){
  printf("%25s: %.10g\n","Area",a);
}

int main(){
  printf("*****Perimetro y area de un triangulo*****\n");
  double a,b,c,p,s;
  leerLados(a,b,c);
  p=perimetro(a,b,c);
  s=area(a,b,c);
  mostrarPerimetro(p);
  mostrarArea(s);
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem1a.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem1a ejem1a.o

c:\HPV\programas\c++\tema2_2_2012>ejem1a
*****Perimetro y area de un triangulo*****
                  Lado1? 2
                  Lado2? 2
                  Lado3? 3
               Perimetro: 7
                    Area: 1.984313483
```

Y la solución empleando librerías C++ es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem1b.cpp
```

```
#include <iostream>
#include <cmath>
using namespace std;

void leerLados(double &a, double &b, double &c){
  cout.width(25);
  cout<<"Lado1? ";
  cin>>a;
```

```
  cout.width(25);
  cout<<"Lado2? ";
  cin>>b;
  cout.width(25);
  cout<<"Lado3? ";
  cin>>c;
}

double perimetro(double a, double b, double c){
  return a+b+c;
}

double area(double a, double b, double c){
  double s,r;
  s=perimetro(a,b,c)/2;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}

void mostrarPerimetro(double p){
  cout.width(25);
  cout.precision(10);
  cout<<"Perimetro: "<<p<<endl;
}

void mostrarArea(double a){
  cout.width(25);
  cout.precision(10);
  cout<<"Area: "<<a<<endl;
}

int main(){
  cout<<"*****Perimetro y area de un triangulo*****"<<endl;
  double a,b,c,p,s;
  leerLados(a,b,c);
  p=perimetro(a,b,c);
  s=area(a,b,c);
  mostrarPerimetro(p);
  mostrarArea(s);
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem1b.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem1b ejem1b.o

c:\HPV\programas\c++\tema2_2_2012>ejem1b
*****Perimetro y area de un triangulo*****
                  Lado1? 2
                  Lado2? 2
                  Lado3? 3
               Perimetro: 7
                    Area: 1.984313483
```

2. Elabore un programa que reciba el radio y altura de un cono circular recto y devuelva su área total y volumen. Los resultados deben ser mostrados con 12 dígitos de precisión.

Las fórmulas para calcular el área y volumen son:

$$area\ lateral = \pi r \sqrt{r^2 + h^2}$$
$$area\ total = area\ lateral + \pi r^2$$
$$volumen = \frac{1}{3}\pi r^2 h$$

En este caso se identifican 5 módulos: a) Uno para leer los datos (el radio y la altura); b) Otro para calcular el área total; c) Otro para calcular el volumen; d) Otro para mostrar el área y e) Otro para mostrar el volumen.

Una vez más, al tratarse de un problema numérico general, se emplea el tipo de dato "double.

La solución empleando librerías C es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem2a.cpp
```

```
#include <cstdio>
#include <cmath>

void leerDatos(double &r, double &h){
  printf("%15s? ","Radio");
  scanf("%lf",&r);
  printf("%15s? ","Altura");
  scanf("%lf",&h);
}

double area(double r, double h){
  double al;
  al=M_PI*r*sqrt(r*r+h*h);
  return al+M_PI*r*r;
}

double volumen(double r, double h){
  return M_PI*r*r*h/3;
}

void mostrarArea(double a){
  printf("%15s: %.12g\n","Area",a);
}

void mostrarVolumen(double v){
  printf("%15s: %.12g\n","Volumen",v);
}

int main(){
  printf("*****Area y volumen de un cono*****\n");
  double r,h,a,v;
  leerDatos(r,h);
  a=area(r,h);
  v=volumen(r,h);
  mostrarArea(a);
  mostrarVolumen(v);
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem2a.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem2a ejem2a.o

c:\HPV\programas\c++\tema2_2_2012>ejem2a
*****Area y volumen de un cono*****
          Radio? 2
         Altura? 5
           Area: 46.4023590073
        Volumen: 20.9439510239
```

La solución empleando librerías C++ es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem2b.cpp
```

```
#include <iostream>
#include <cmath>
using namespace std;

void leerDatos(double &r, double &h){
  cout.width(15);
  cout<<"Radio? ";
  cin>>r;
  cout.width(15);
  cout<<"Altura? ";
  cin>>h;
}

double area(double r, double h){
  double al;
  al=M_PI*r*sqrt(r*r+h*h);
  return al+M_PI*r*r;
}

double volumen(double r, double h){
  return M_PI*r*r*h/3;
}

void mostrarArea(double a){
  cout.width(15);
  cout.precision(12);
  cout<<"Area: "<<a<<endl;
}

void mostrarVolumen(double v){
  cout.width(15);
  cout.precision(12);
  cout<<"Volumen: "<<v<<endl;
}

int main(){
  cout<<"*****Area y volumen de un cono*****"<<endl;
  double r,h,a,v;
  leerDatos(r,h);
  a=area(r,h);
  v=volumen(r,h);
  mostrarArea(a);
  mostrarVolumen(v);
```

```
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem2b.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem2b ejem2b.o

c:\HPV\programas\c++\tema2_2_2012>ejem2b
*****Area y volumen de un cono*****
        Radio? 2
       Altura? 5
         Area: 46.4023590073
      Volumen: 20.9439510239
```

3. Elabore un programa que calcule el valor de la siguiente función. El resultado debe ser mostrado con 10 dígitos de precisión.

$$f(p)=\cos(k\cdot l)\cdot\cosh(k\cdot l)+1$$

$$k^2=\frac{p}{a}$$

$$a^2=\frac{E\cdot I\cdot g}{A\cdot y}$$

$$l=120$$

$$I=170.6$$

$$E=3\text{x}10^6$$

$$y=0.066$$

$$A=32$$

$$g=386$$

Como este problema tiene varias sentencias de se debe pensar en el orden de evaluación: En primer lugar se deben asignar los valores de las constantes, luego se debe calcular el valor de "a" (pues se su valor se requiere para calcular "k"), luego el de "k" y finalmente el valor de la función.

Como la función depende de una sola variable, sólo se requieren tres módulos: a) Uno para leer el valor de "p"; b) Otro para calcular el valor de la función y c) Otro para mostrar el valor calculado.

Una vez más, al tratarse de un problema numérico general, se empleará el tipo de dato "double".

La solución, empleando librerías C, es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem3a.cpp
```

```
#include <cstdio>
#include <cmath>

double leerP(){
  double p;
  printf("%21s? ","Valor de p");
  scanf("%lf",&p);
  return p;
}

double fp(double p){
  const double l=120, I=170.6, E=3e6, y=0.066, A=32, g=386;
  double a,k;
  a=sqrt(E*I*g/(A*y));
  k=sqrt(p/a);
```

```
  return cos(k*l)*cosh(k*l)+1;
}

void mostrarResultado(double f){
  printf("%21s: %.10g\n","Valor de la funcion",f);
}

int main(){
  double p,f;
  p=leerP();
  f=fp(p);
  mostrarResultado(f);
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem3a.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem3a ejem3a.o

c:\HPV\programas\c++\tema2_2_2012>ejem3a
          Valor de p? 50
  Valor de la funcion: 1.088488666
```

La solución, empleando librerías C++, es:

```
c:\HPV\programas\c++\tema2_2_2012>notepad ejem3b.cpp
```

```
#include <iostream>
#include <cmath>
using namespace std;

double leerP(){
  double p;
  cout.width(21);
  cout<<"Valor de p? ";
  cin>>p;
  return p;
}

double fp(double p){
  const double l=120, I=170.6, E=3e6, y=0.066, A=32, g=386;
  double a,k;
  a=sqrt(E*I*g/(A*y));
  k=sqrt(p/a);
  return cos(k*l)*cosh(k*l)+1;
}

void mostrarResultado(double f){
  cout.width(21);
  cout.precision(10);
  cout<<"Valor de la funcion: "<<f<<endl;
}

int main(){
  double p,f;
  p=leerP();
  f=fp(p);
  mostrarResultado(f);
```

```
  return 0;
}
```

```
c:\HPV\programas\c++\tema2_2_2012>g++ -c ejem3b.cpp

c:\HPV\programas\c++\tema2_2_2012>g++ -o ejem3b ejem3b.o

c:\HPV\programas\c++\tema2_2_2012>ejem3b
        Valor de p? 50
Valor de la funcion: 1.088488666
```

2.6. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema2" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa, que empleando librerías C, calcule el valor de la siguiente función. Los resultados deben ser mostrados con 12 dígitos de precisión.

$$f(x)=x+y^2+z^2-14$$
$$y=\frac{3x^{2.1}-7.0}{5}$$
$$z=\frac{14.3-y^{1.2}}{4}$$

2. Resuelva el ejercicio anterior empleando librerías C++, muestre los resultados con 9 dígitos de precisión.
3. Elabore un programa, que empleando librerías C, calcule el valor de la siguiente función. Los resultados deben ser mostrados con 7 dígitos de precisión (debe despejar los valores de "z", "y" y "w").

$$f(x)=x+w+y-39$$
$$\sqrt{(x)}+z^2=35$$
$$y+e^{z/2}-8/z=30$$
$$w+z^2-2z=31$$

4. Resuelva el ejercicio anterior empleando librerías C++, muestre los resultados con 14 dígitos de precisión.
5. Elabore un programa, que empleando librerías C, calcule el valor de la siguiente función. Muestre los resultados con 12 dígitos de precisión.

$$f(x_1)=L_1\cdot x_1+V_1\cdot y_1-C_0$$
$$L_1=\frac{L_c}{1-x_1}$$
$$V_1=M-L_1$$
$$y_1=1420\,x_1$$
$$L_c=L_0(1-x_0)$$
$$M=L_0+V_0$$
$$C_0=L_0\cdot x_0+V_0\cdot y_0$$
$$L_0=300$$
$$x_0=0$$
$$V_0=100$$
$$y_0=0.2$$

6. Resuelva el ejercicio anterior empleando librerías C++, muestre los resultados con 8 dígitos de precisión.

3. SELECCION-1

En este tema continua el repaso de las estructuras estándar, las mismas que deben ser empleadas para aplicar correctamente el segundo principio de la programación estructurada: "construir programas empleado estructuras estándar".

De acuerdo con el teorema de la programación estructurada sólo se requiere una estructura selectiva: la estructura **if-then-else**. Sin embargo, muchos lenguajes (incluido C++) cuentan con algunas otras estructuras selectivas, las cuales, en ocasiones, permiten resolver los problemas más eficientemente y sobre todo con mayor claridad, por lo que es conveniente estudiarlas y emplearlas en dichos casos.

En ese sentido, en la asignatura se amplía el concepto de "estructura estándar" para incluir dentro de ellas todas las estructuras que tengan un sólo flujo de entrada y uno de salida, de manera que puedan encajar en un programa (como bloque), sin introducir desorden en la lógica.

Antes de comenzar el repaso de estas estructuras es necesario recordar algunos conceptos.

3.1. EXPRESIONES LÓGICAS

Una *expresión lógica* o *expresión condicional* es aquella que devuelve un valor lógico. Un valor lógico es un booleano que sólo puede tener uno de dos valores: falso *(false)* o verdadero *(true)*.

En C el número cero es interpretado como falso y cualquier valor diferente de cero es interpretado como verdadero. Como ya se mencionó en el tema anterior, C++ cuenta con el tipo booleano "bool" el cual puede tomar los valores "true" (verdadero) o "false" (falso), sin embargo, se puede seguir empleando el número cero para falso y el uno (o cualquier valor diferente de cero) para verdadero.

Cuando se convierte un valor booleano a un número el valor "false" es convertido a cero y el valor "true" en uno.

Para construir una expresión lógica se emplean operadores relacionales y operadores lógicos.

3.1.1. Operadores relacionales

Los operadores relacionales permiten comparar dos valores, devolviendo verdadero cuando se cumple la relación y falso en caso contrario. En C++ se cuenta con los siguientes operadores relacionales:

==	Igual a
!=	Diferente a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

3.1.2. Operadores lógicos

Los operadores lógicos, como su nombre sugiere, sólo trabajan con valores de tipo lógico (true o false). Puesto que los operadores relacionales siem-

pre devuelven un resultado de tipo lógico, pueden emplearse en combinación con estos operadores para construir expresiones lógicas más ricas.

En C++ se cuenta con los siguientes operadores lógicos:

!	Negación lógica
&&	Y lógico
||	O lógico

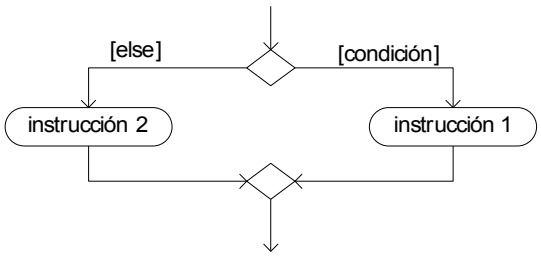
El operador "**!**" cambia un valor lógico de falso *(false)* a verdadero *(true)* y viceversa.

El operador "**&&**" devuelve verdadero si los dos valores que compara son verdaderos y falso en cualquier otro caso.

El operador "**||**" devuelve falso si los dos valores que compara son falsos y verdadero en cualquier otro caso.

3.2. ESTRUCTURA IF-ELSE

La estructura *IF-ELSE* tiene la siguiente lógica:



Como se puede ver si la *condición* es verdadera se ejecuta la "instrucción 1", caso contrario se ejecutan la "instrucción 2".

El código respectivo en C/C++ es:

```
if {condición}
  instrucción_1;
else
  instrucción_2;
```

Si en lugar de una instrucción se tiene una secuencia de instrucciones, las mismas deben estar encerradas entre llaves (para que conformen una estructura propiamente y sean ejecutadas como un bloque).

El caso contrario (else) de esta estructura es opcional, es decir que puede no existir. Cuando eso sucede, si la condición es falsa, el programa pasa a ejectuar la instrucción que se encuentra después de la estructura "if".

La "instrucción" pueden ser también cualquier otra estructura estándar, incluida otra estructura "if-else". En este último caso (cuando una estructura "if-else" está dentro de otra estructura "if-else") se dice que las estructuras están anidadas. En C/C++ se pueden anidar tantas estructuras como se quiera, sin embargo, se recomienda anidar como máximo 3 estructuras, porque luego la lógica se hace muy compleja, dificultando el mantenimiento del programa (algo que va contra los principios de la programación estructurada).

3.3. OPERADORES DE INCREMENTO Y DECREMENTO

Como ya se repasó en el anterior tema, los operadores de incremento y de-

cremento son:

Operador	Operación equivalente
x++	x=x+1
x--	x=x-1
++x	x=x+1
--x	x=x-1

Es conveniente profundizar un poco más en la forma en que operan cuando se encuentran como prefijo o posfijo. Cuando el operador se encuentra como prefijo, primero se realiza el incremento y luego se emplea la variable, por el contrario cuando el operador se encuentra como posfijo primero se emplea la variable y luego se incrementa su valor.

Por ejemplo las siguientes instrucciones asignan el número 10 a la variable "x" (quedando la variable "y" con el número 5):

```
y = 4; x = ++y * 2;
```

Mientras que las siguientes instrucciones asignan el número 8 a la variable "x" (quedando igualmente la variable "y" con el número 5):

```
y = 4; x = y++ * 2;
```

3.4. LA FUNCIÓN EXIT()

Al igual que "return" termina la ejecución de un módulo (de una función), la función *exit(número)*, termina la ejecución de un programa, devolviendo el "número" (que es un código de error) al proceso que le llamó (normalmente el sistema operativo).

La razón por la que se escribe la instrucción "return 0" al final de la función principal (main) es para informar al sistema operativo, o al proceso que llamó al programa, de una ejecución exitosa, cualquier valor diferente indica que el programa ha concluido debido a algún error. Lo mismo es válido para la función "exit".

En lugar de 0 y valores diferentes de 0, se pueden emplear también las constantes (macros) EXIT_SUCCESS para informar que el programa ha sido ejecutado con éxito y EXIT_FAILURE, para informar que el programa ha concluido debido a algún error (estas constantes están definidas en la librería <cstdlib>).

3.5. EJEMPLOS

1. Elabore un programa que, empleando librerías C, determine si un número dado es par, impar o cero. El programa debe contar con un módulo que devuelva 0 si el número es cero, 1 si es impar y 2 si es par.

En este problema se identifican los siguientes módulos (primer principio de la programación estructurada): a) Un módulo para leer el número; b) Un módulo para determinar si el número es par, impar o cero y c) Un módulo para mostrar el resultado.

En cuanto a la lógica, para determinar si un número es par, simplemente se divide entre 2 y si el residuo es cero, entonces el número es par, caso contrario (si es uno) es impar.

El problema surge cuando el numero es cero (pues el residuo de 0 entre 2 es también 0), por ello, para determinar si un número es par, impar o cero, primero se debe averiguar si el número es cero y si no determinar si es par

o impar.

En lo referente al tercer principio de la programación estructurada (emplear tipos de datos a la medida), ahora que se cuenta con una estructura lógica (la estructura if-else) se pueden validar los datos, es decir verificar que los datos sean del tipo correcto.

Como en este problema se quiere averiguar si un número es par, impar o cero y sólo los números enteros pueden serlo, el programa debe permitir introducir únicamente datos de tipo entero. Para ello, una vez introducido el número se debe verificar que no tenga parte fraccionaria y si la tiene mostrar un mensaje de error y terminar el programa.

Con estas consideraciones, la solución es:

```
C:\HPV\programas\c++\tema3_2_2012>notepad ejem1.cpp
```

```
#include <cstdio>
#include <cstdlib>
#include <cmath>

int leerNumero(){
  double x;
  printf("%15s? ","Numero");
  scanf("%lf",&x);
  if (fmod(x,1)) {
    printf("%s\n","Error: El numero debe ser entero");
    exit(EXIT_FAILURE);
  }
  return (int)x;
}

//Retorna 0, si "n" es cero, 1 si es impar y 2 si es par
int parImparCero(int n){
  if (!n) return 0;
  if (n%2)
    return 1;
  else
    return 2;
}

void mostrarSiPar(int r){
  printf("%15s: ","El numero es");
  if (!r)
    printf("%s\n","cero");
  else
    if (r==1)
      printf("%s\n","impar");
    else
      printf("%s\n","par");
}

int main(){
  int n,r;
  n=leerNumero();
  r=parImparCero(n);
  mostrarSiPar(r);
  return EXIT_SUCCESS;
```

```
}
```

Observe que en el módulo "leerNumero" el número se lee en una variable de tipo real (double), a pesar de que el módulo devuelve un resultado de tipo entero. Se procede así porque de otra forma no sería posible comprobar si el número introducido es o no real. Cuando se declara una variable de tipo entero y se introduce un número real, el compilador trunca automáticamente la parte fraccionaria, así por ejemplo, si la variable es de tipo entero y se introduce un número real, como 53.324, no se genera un error, sino que se guarda solo la parte entera (53), en consecuencia, el programa informaría erróneamente que el número es impar.

Para comprobar si un número real tiene o no parte fraccionaria se ha empleado la función "fmod(n,d)", que calcula el residuo de la división de "n" entre "d" (igual que el operador "%", sólo que con números reales). Si al dividir un número real entre 1 el residuo es diferente de 0, se sabe que el número tiene parte fraccionaria. En consecuencia la expresión relacional que permite comprobar si el número tiene o no parte fraccionaria debería ser "if (fmod(x,1)!=0)", sin embargo, en la función "leerNumero" sólo se ha escrito la expresión "if (fmod(x,1))" (¡sin compararla con nada!). Se ha procedido así, porque, como se dijo, en C/C++ todo valor diferente de 0 es un valor verdadero, por lo tanto, si el residuo es diferente de 0 (si el número tiene parte fraccionaria) la condición es de hecho verdadera y si no (si el residuo es 0) es falsa.

Observe también que antes de devolver el resultado se convierte el número real (que se sabe no tiene parte fraccionaria) en un número entero (int). Para ello simplemente se escribe delante del valor, variable o expresión a convertir, el tipo de dato entre paréntesis (en el ejemplo "(int)"). A esta técnica se conoce como "moldeo de tipos" y la forma empleada en el ejemplo corresponde al estilo C. En C++, el moldeo de tipos puede hacerse también escribiendo el tipo de dato y encerrando entre paréntesis el valor, variable o expresión a convertir (como si se tratara de una función), así, en C++, la conversión puede hacerse con "int(x)" (aunque por supuesto es válido también el estilo C).

Observe igualmente que en las funciones "parImparCero" y "mostrarSiPar", se tienen condiciones como "!n" y "!x". Estas expresiones constituyen valores lógicos por la misma razón que "float(n,1)", es decir porque en C/C++ el 0 es falso y todo valor diferente de 0 es verdadero. En consecuencia, si "n" es cero la expresión es falsa y al negarla (con el operador "!") se convierte en verdadera. Estas expresiones pueden ser escritas igualmente en la forma convencional, es decir "n!=0" y "x!=0", sin embargo, la mayoría de los programadores prefieren la forma no convencional, por lo que es más frecuente encontrarla en los programas profesionales.

Lo mismo es válido en la expresión "n%2": si el número es impar el residuo es 1, por lo tanto es verdadero, caso contrario es falso (cero). Observe, sin embargo que este tipo de expresiones no se pueden emplear en lugar de "if (r==1)", porque si se escribiera "if (r)", sería verdadero tanto cuando "r" es 1 (impar), como cuando "r" es 2 (par).

Compilando y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema3_2_2012>g++ -c ejem1.cpp

C:\HPV\programas\c++\tema3_2_2012>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\tema3_2_2012>ejem1
        Numero? 123
   El numero es: impar
```

```
C:\HPV\programas\c++\tema3_2_2012>ejem1
        Numero? 428
   El numero es: par

C:\HPV\programas\c++\tema3_2_2012>ejem1
        Numero? 0
   El numero es: cero

C:\HPV\programas\c++\tema3_2_2012>ejem1
        Numero? 5423.2323
Error: El numero debe ser entero
```

2. Elabore un programa que, empleando librerías tipo C++, determine si un año dado es o no bisiesto. Como un año "es" o "no es" bisiesto (no existen años más o menos bisiestos) el programa debe contar con una función que devuelva verdadero si el año es bisiesto y falso en caso contrario.

Un año es bisiesto si es divisible entre cuatro, con excepción de los años que terminan en dos ceros (como 1900), a los cuales se les debe quitar los dos ceros antes de verificar si son divisibles entre cuatro.

Analizando el problema (primer principio) se identifican los siguientes módulos: a) Un módulo para leer el año; b) Un módulo para determinar si el año es o no es bisiesto y c) Un módulo para mostrar el resultado.

En cuanto al tercer principio (tipos de datos a la medida), se debe validar que los años sean enteros y positivos.

Tomando en cuenta estas consideraciones, una de las posibles forma de elaborar el programa es:

```
C:\HPV\programas\c++\tema3_2_2012>g++ -c ejem2.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

unsigned leerAnio(){
  double a;
  cout.width(15);
  cout<<"Año? ";
  cin>>a;
  if (fmod(a,1)){
    cout<<"Error: El año debe ser entero"<<endl;
    exit(EXIT_FAILURE);
  }
  if (a<=0){
    cout<<"Error: El año debe ser positivo"<<endl;
    exit(EXIT_FAILURE);
  }
  return unsigned(a);
}

bool bisiesto(unsigned a){
  if (!(a%100)) a/=100;
  return !(a%4);
}

void mostrarSiBisiesto(unsigned a,bool r){
```

```
  cout.width(15);
  cout<<"El año "<<a<<" ";
  if (r)
    cout<<"es bisiesto"<<endl;
  else
    cout<<"no es bisiesto"<<endl;
}

int main(){
  unsigned a;
  bool r;
  a=leerAnio();
  r=bisiesto(a);
  mostrarSiBisiesto(a,r);
  return EXIT_SUCCESS;
}
```

Como se puede ver, en el módulo "leerAnio" no sólo se comprueba que el número sea entero, sino además que sea positivo. El tipo "unsigned" empleado es la forma corta de escribir "unsigned int".

En el módulo que resuelve el problema "bisiesto", primero se comprueba si el año tiene dos ceros al final del mismo, para ello se calcula el residuo de su división entre 100: si el residuo es cero tiene dos ceros, caso contrario no. Como cero es equivalente a falso, se niega el resultado para convertirlo en verdadero. Por supuesto, la instrucción puede ser escrita también en la forma normal, es decir: "if (a%100==0) a=a/100;".

Como se puede ver también (en el módulo "bisiesto"), una vez que se ha comprobado que el año no tiene dos ceros al final (o dichos ceros han sido eliminados), se devuelve directamente el resultado de "!(a%4)". Se procede así, porque se sabe que cuando el año es divisible entre 4 es bisiesto, caso contrario no, pero cuando es divisible el residuo es cero (falso), por lo que debe ser negado (con el operador "!") para convertirlo en verdadero. Como ocurre siempre, la expresión puede ser escrita también en la forma normal: "a%4==0".

Compilando, enlazando y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPV\programas\c++\tema3_2_2012>g++ -c ejem2.cpp

C:\HPV\programas\c++\tema3_2_2012>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? 2000
        El año 2000 es bisiesto

C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? 1900
        El año 1900 no es bisiesto

C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? 2012
        El año 2012 es bisiesto

C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? 2006
        El año 2006 no es bisiesto

C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? 2012.2238842
Error: El año debe ser entero
```

```
C:\HPV\programas\c++\tema3_2_2012>ejem2
          Año? -1990
Error: El año debe ser positivo
```

3. Elabore un programa que, empleando librerías tipo C, lea tres números reales, encuentre el mayor de ellos y muestre dicho valor.

Los módulos que se pueden identificar en este problema (primer principio) son: a) Un módulo para leer los tres números; b) Un módulo para encontrar el mayor de los tres números y c) Un módulo para mostrar dicho número.

En lo referente a la lógica que resuelve el problema (encontrar el mayor de tres números) primero se encuentra el mayor de los dos primeros números, entonces se encuentra el mayor entre ese número y el tercero.

En cuanto al tercer principio, como se trata de un número real, se permite introducir cualquier número, por lo tanto la única validación posible consiste en emplear el tipo de dato adecuado y el más adecuado para un caso general (como el presente) es el tipo "double".

Como de costumbre, el segundo principio (emplear estructuras estándar) se aplica directamente en la construcción del programa y como hasta ahora sólo se cuenta con dos estructuras (la secuencia y la estructura "if-else") se emplean ambas estructuras en la elaboración del programa.

Tomando en cuenta las anteriores consideraciones, una de las posibles formas de implementar la solución es:

```
C:\HPV\programas\c++\tema3_2_2012>notepad ejem3.cpp
```

```
#include <cstdio>
#include <cstdlib>
using namespace std;

void leerNumeros(double &a, double &b, double &c){
  printf("%20s:\n","Números a ordenar");
  printf("%20s? ","Primer numero");
  scanf("%lf",&a);
  printf("%20s? ","Segundo numero");
  scanf("%lf",&b);
  printf("%20s? ","Tercer numero");
  scanf("%lf",&c);
}

double mayor(double a, double b, double c){
  if (a>b) b=a;
  if (b>c) c=b;
  return c;
}

void mostrarMayor(double m){
  printf("%20s: %.9g\n","El mayor valor es",m);
}

int main(){
  double a,b,c,m;
  leerNumeros(a,b,c);
  m=mayor(a,b,c);
  mostrarMayor(m);
  return EXIT_SUCCESS;
```

```
}
```

En el módulo "mayor" el mayor de los dos primeros valores se guarda en la variable "b", mientras que el mayor de los tres en la variable "c". Por esa razón (porque se modifican sus valores) los parámetros deben ser recibidos por valor y no por referencia.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema3_2_2012>g++ -c ejem3.cpp

C:\HPV\programas\c++\tema3_2_2012>g++ -o ejem3 ejem3.o

C:\HPV\programas\c++\tema3_2_2012>ejem3
  Introduzca tres numeros:
            Primer numero? 4.5
           Segundo numero? 1.2
            Tercer numero? 3.9

         El mayor valor es: 4.5

C:\HPV\programas\c++\tema3_2_2012>ejem3
  Introduzca tres numeros:
            Primer numero? 1.2
           Segundo numero? 4.5
            Tercer numero? 3.9

         El mayor valor es: 4.5

C:\HPV\programas\c++\tema3_2_2012>ejem3
  Introduzca tres numeros:
            Primer numero? 3.9
           Segundo numero? 1.2
            Tercer numero? 4.5

         El mayor valor es: 4.5
```

4. Elabore un programa que, empleando librerías tipo C++, lea tres números enteros, los ordene ascendentemente y muestre los números ordenados.

Los módulos que se pueden identificar son: a) Un módulo para leer los números; b) Un módulo para ordenarlos y c) Un módulo para mostrarlos. Sin embargo, como en este caso se deben validar 3 números (para verificar que sean enteros), es conveniente crear un cuarto módulo para llevar a cabo esta tarea, evitando así repetir el código 3 veces.

En lo referente a la lógica, para ordenar tres números primero se lleva el mayor a la tercera posición, luego el mayor de los dos restantes a la segunda, con lo que los tres números quedan ordenados.

Tomando en cuenta las anterior consideraciones una posible solución es:

```
C:\HPV\programas\c++\tema3_2_2012>notepad ejem4.cpp
```

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

int validar(double x){
  if (fmod(x,1)) {
    cout<<"Error: El numero debe ser entero."<<endl;
    exit(EXIT_FAILURE);
  }
  return int(x);
```

```
}

void leerNumeros(int &a, int &b, int &c){
  double x,y,z;
  cout.width(20);
  cout<<"Numeros a ordenar:"<<endl;
  cout.width(20);
  cout<<"Primer numero? ";
  cin>>x;
  a=validar(x);
  cout.width(20);
  cout<<"Segundo numero? ";
  cin>>y;
  b=validar(y);
  cout.width(20);
  cout<<"Tercer numero? ";
  cin>>z;
  c=validar(z);
}

void ordenar(int &a, int &b, int &c){
  int aux;
  if (a>b) {aux=a; a=b; b=aux;}
  if (b>c) {aux=b; b=c; c=aux;}
  if (a>b) {aux=a; a=b; b=aux;}
}

void mostrarNumeros(int a, int b, int c){
  cout.width(20);
  cout<<"Numeros ordenados: "<<a<<endl;
  cout.width(20);
  cout<<" "<<b<<endl;
  cout.width(20);
  cout<<" "<<c<<endl;
}


int main(){
  int a,b,c;
  leerNumeros(a,b,c);
  ordenar(a,b,c);
  mostrarNumeros(a,b,c);
  return EXIT_SUCCESS;
}
```

Como se puede ver, en el módulo "ordenar" se comprueba si "a" es mayor que "b" y de ser así se intercambian sus valores (para que la variable "b" quede con el mayor valor), luego se comprueba si "b" (que ya tiene el mayor valor de los dos primeros números) es mayor que "c" y de ser así se intercambian sus valores. De esa manera, después de estas dos condiciones, se garantiza que la variable "c" tenga el mayor de los tres valores.

Luego se vuelve a comparar "a" con "b" (porque sus valores pueden haber cambiado con los intercambios) garantizando así que "b" se quede con el mayor de los dos primeros valores (y en consecuencia "a" con el menor).

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPU\programas\c++\tema3_2_2012>g++ -c ejem4.cpp

C:\HPU\programas\c++\tema3_2_2012>g++ -o ejem4 ejem4.o

C:\HPU\programas\c++\tema3_2_2012>ejem4
  Numeros a ordenar:
     Primer numero? 23
    Segundo numero? 12
     Tercer numero? 17

 Numeros ordenados: 12
                    17
                    23

C:\HPU\programas\c++\tema3_2_2012>ejem4
  Numeros a ordenar:
     Primer numero? 17
    Segundo numero? 23
     Tercer numero? 12

 Numeros ordenados: 12
                    17
                    23

C:\HPU\programas\c++\tema3_2_2012>ejem4
  Numeros a ordenar:
     Primer numero? 23
    Segundo numero? 12
     Tercer numero? 17

 Numeros ordenados: 12
                    17
                    23

C:\HPU\programas\c++\tema3_2_2012>ejem4
  Numeros a ordenar:
     Primer numero? 23
    Segundo numero? 12.43
Error: El numero debe ser entero.
```

5. Elabore un programa que, empleando librerías tipo C, encuentre y muestre (con 9 dígitos de precisión) las soluciones (raíces) reales y complejas de una ecuación cuadrática.

La forma general de la ecuación cuadrática es:

$$a \cdot x^2 + b \cdot x + c = 0$$

Donde las dos soluciones se calculan con la ecuación general:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

El tipo de solución resultante depende del valor de $b^2 - 4 \cdot a \cdot c$, denominado discriminante y existen tres posibles casos: a) Si es positivo, la raíz cuadrada es real siendo las dos soluciones reales y diferentes; b) Si es cero, la raíz cuadrada es cero siendo las dos soluciones reales e iguales y c) Si es negativo la raíz cuadrada es imaginaria y en consecuencia las dos soluciones son complejas (un par conjugado).

En este problema se pueden identificar los siguientes módulos: a) Un módulo para leer los coeficientes de la ecuación cuadrática; b) Un módulo para calcular y devolver las soluciones y c) Un módulo para mostrar las soluciones encontradas.

En cuanto a los tipos de datos (tercer principio), los coeficientes de la ecuación cuadrática pueden ser valores tanto enteros como reales, por lo que la única validación posible es la de emplear un tipo de dato real, siendo el real más genérico el tipo "double".

Tomando en cuenta las anteriores consideraciones, una posible solución es:

```
C:\HPV\programas\c++\tema3_2_2012>notepad ejem5.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

void leerCoeficientes(double &a, double &b, double &c){
  printf("\n%18s: %s\n","Coeficientes de","a*x^2+b*x+c=0");
  printf("%18s? ","a");
  scanf("%lf",&a);
  printf("%18s? ","b");
  scanf("%lf",&b);
  printf("%18s? ","c");
  scanf("%lf",&c);
}

void cuadratica(double a, double b, double c, double &r1,
                double &r2, double &im) {
  double r,d=b*b-4*a*c;
  if (d>0)
    {r=sqrt(d);
     r1=(-b-r)/(2*a);
     r2=(-b+r)/(2*a);
     im=0;}
  else
    if (d==0)
      {r1=r2=-b/(2*a);
       im=0;}
    else
      {r1=r2=-b/(2*a);
       im=sqrt(-d)/(2*a);}
}

void mostrarRaices(double r1, double r2, double im) {
  printf("\n%18s:\n\n","Raices");
  if (im)
    {printf("%18s: %.9g + %.9gi\n","r1",r1,im);
     printf("%18s: %.9g - %.9gi\n","r2",r2,im);}
  else
   {printf("%18s: %.9g\n","r1",r1);
    printf("%18s: %.9g\n","r2",r2);}
}

int main(){
  double a,b,c,r1,r2,im;
  leerCoeficientes(a,b,c);
  cuadratica(a,b,c,r1,r2,im);
  mostrarRaices(r1,r2,im);
```

```
  return 0;
}
```

Observe que en el módulo "cuadratica", en el caso "d>0", el valor de la raíz cuadrada se guarda en una variable auxiliar "r". Se procede así para evitar calcular dos veces la raíz cuadrada del mismo número.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPV\programas\c++\tema3_2_2012>g++ -c ejem5.cpp

C:\HPV\programas\c++\tema3_2_2012>g++ -o ejem5 ejem5.o

C:\HPV\programas\c++\tema3_2_2012>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? 2
                 c? 3

            Raices:

                r1: -1 + 1.414213561
                r2: -1 - 1.414213561

C:\HPV\programas\c++\tema3_2_2012>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? -10
                 c? 21

            Raices:

                r1: 3
                r2: 7

C:\HPV\programas\c++\tema3_2_2012>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? -14
                 c? 49

            Raices:

                r1: 7
                r2: 7
```

3.6. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema3" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa que, empleando librerías tipo C, permita determinar si un número dado es positivo, negativo o cero. El programa debe contar con un módulo que devuelva 0 si el número es cero, -1 si es negativo y 1 si es positivo.

2. Elabore un programa que, empleando librerías tipo C++, lea los tres lados de un triángulo y determine si los mismos conforman o no un triángulo. Tres lados conforman un triángulo si, **en todos los casos**, la suma de

dos de ellos es mayor al tercero. El programa debe contar con un módulo que devuelva verdadero si los lados conforman un triángulo y falso en caso contrario.

3. Elabore un programa que, empleando librerías tipo C, lea cuatro números reales y muestre el menor de ellos.

4. Elabore un programa que, empleando librerías tipo C++, lea cuatro números enteros, los ordene descendentemente y los muestre en pantalla.

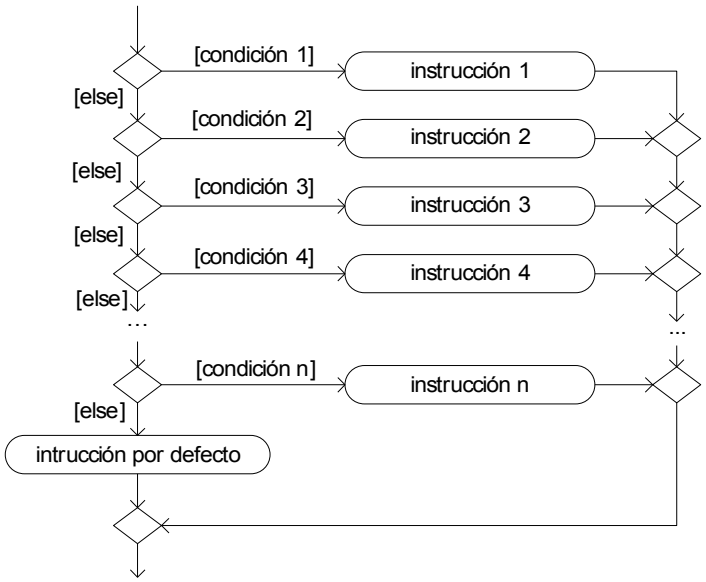
5. Elabore un programa que, empleando librerías tipo C, lea los coeficientes de la ecuación cuadrática "$a+bx+cx^2$" y muestre las soluciones reales e imaginarias de la misma. En el módulo que resuelve el problema, debe tomar en cuenta que si las soluciones no son reales y diferentes, la parte real de las soluciones (para el casos reales e iguales como para el caso complejos) es la misma "-b/(2*a)"), por lo que debe ser calculada una sola vez, en el caso contrario de la estructura "if", donde también se debe calcular la parte compleja "si" el resultado es de ese tipo.

4. SELECCION-2

En este tema continua el repaso de las estructuras selectivas, con una variante de la estructura "if-else", la estructura "if-else if" y con otra estructura selectiva, la estructura switch.

4.1. LA ESTRUCTURA SWITCH E IF – ELSE IF

Cuando la lógica que resuelve el problema implica dos o más condiciones consecutivas, donde el caso contrario es siempre otra condición, tal como se muestra en el siguiente diagrama:



La forma más clara de codificarla es mediante la estructura "if – else if", que simplemente es la estructura "if", pero ordenada de manera que se puedan ver claramente las diferentes condiciones:

```
if (condición_1)
    instrucción_1;
  else if (condición_2)
    instrucción_2;
  else if (condición_3)
    instrucción_3;
  else if (condición_4)
    instrucción_4;
  ...
  else if (condición_n)
    instrucción_n;

 else

   instrucciones_por_defecto;
```

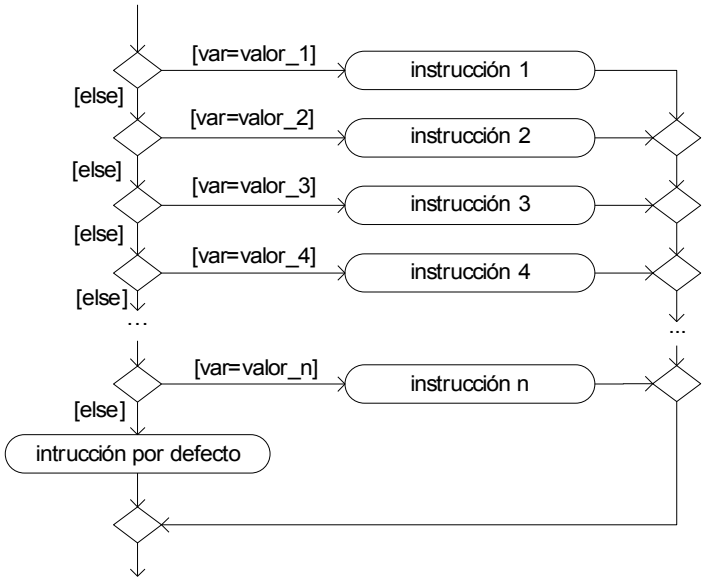
Como se puede ver en el diagrama, si una de las condiciones es verdadera se ejecuta la instrucción respectiva y el programa continua con la siguiente instrucción del programa. Si ninguna de las instrucciones es verdadera se ejecutan las instrucciones por defecto (las del último "else").

Al igual que en la estructura "if-else" estándar, la última instrucción "else" (la de las instrucciones por defecto) es opcional, en cuyo caso la

estructura simplemente termina y continua con la siguiente instrucción del programa.

Como es usual, si en lugar de una instrucción se tiene una secuencia de instrucciones, las mismas deben ser encerradas entre llaves (formando así una secuencia, es decir un bloque que se ejecuta como una instrucción).

Cuando en todos los casos la condición es de igualdad y se compara el valor de una variable (o el de una expresión) con diferentes valores ordinales (es decir enteros, enumerados, caracteres, etc.), como ocurre en el siguiente diagrama:



Una forma más clara y eficiente de programarla es con estructura "switch":

```
switch (var) {
  case valor_1: instrucción_1; break;
  case valor_2: instrucción_2; break;
  case valor_3: instrucción_3; break;
  case valor_4: instrucción_4; break;
  ...
  case valor_n: instrucción_n; break;
  default: instrucciones_por_defecto;
}
```

En esta estructura el comando "break", es el que termina la instrucción. Si se omite y por ejemplo es verdadero el caso 2, se ejecutan también las instrucciones de todos los otros casos (caso 2, 3, etc.). Esto no siempre constituye un error, en ocasiones se omite "break", a propósito, para que en un conjunto de casos se ejecuten las mismas instrucciones.

Por ejemplo en el siguiente código:

```
switch(mes){
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: strcpy(s,"31 días"); break;
```

```
    case 4:
    case 6:
    case 9:
    case 11: strcpy(s,"30 días"); break;
    default: strcpy(s,"28 días");
  }
  return s;
}
```

Cuando la variable "mes" es igual a 1, 3, 5, 7, 8, 10 o 12 la variable "s" toma el valor "31 días", mientras que si la variable "mes" es igual a 4, 6, 9, 11 la variable "s" toma el valor "30 días" y cuando "mes" es 2 (el caso por defecto) la variable "s" toma el valor "28 días".

Note, sin embargo, que en el caso 12, se emplea un "break" después de la instrucción, porque de no emplearse, la ejecución continuaría con los siguientes casos. Lo mismo ocurre con el caso 11.

4.2. EXPRESIONES ARITMÉTICO RELACIONALES

Una expresión relacional puede ser transformada en un resultado numérico recordando que un resultado verdadero (true) es equivalente al número 1 y un resultado falso (false) es equivalente a 0. Esto permite, por ejemplo, resolver problemas empleando la estructura "switch" cuando los valores que se comparan no son ordinales y/o no son igualdades.

Así en la solución de la ecuación cuadrática se tienen tres casos que dependen del valor del discriminante "d": a) d>0; b) d<0; y c) d==0. Obviamente la forma más sencilla (y recomendable) de resolver el problema es con la estructura "if-else if", pero puede ser resuelta también con la estructura "switch" empleando expresiones aritmético relacionales, tal como se muestra en el siguiente código:

```
5 void cuad(double a, double b, double c,
6              double &r1, double &r2, double &im){
7   double r,d=b*b-4*a*c;
8   switch ((d>0)+(d<0)*2) {
9     case 1:
10      r=sqrt(d);r1=(-b-r)/(2*a);r2=(-b+r)/(2*a);im=0;
11      break;
12    case 2:
13      r1=r2=-b/(2*a);im=sqrt(-d)/(2*a);
14      break;
15    default:
16      r1=r2=-b/(2*a);im=0;
17      break;
18  }
19 }
```

La lógica es la siguiente: a) Si el discriminante es mayor a cero, la expresión "d>0" es verdadera (es decir es 1) y "d<0" falsa (es decir es 0), por lo tanto el resultado de (d>0)+(d<0)*2 (= 1+0*2 = **1) es 1**; b) Si el discriminante es menor a cero, la expresión "d>0" es falsa (es decir 0) y "d<0" es verdadera (es decir 1), por lo tanto el resultado de: (d>0)+(d<0)*2 (= 0+1*2 = 2) es 2; c) Finalmente, si el discriminante es cero tanto "d>0" como "d<0" son falsas (0), por lo tanto el resultado es: (d>0)+(d<0)*2 (= 0+0*2 = **0) es 0**.

Por supuesto, esta no es la única forma en que puede ser escrita la expresión aritmético relacional, por ejemplo puede ser también:

```
(d>0)+(d==0)*2
```

En este caso si "d>0" el resultado es 1, si "d==0" el resultado es 2 y si "d<0" el resultado es 0. O también:

```
(d<0)+(d==0)*2
```

En este caso si "d<0" el resultado es 1, si "d==0" el resultado es 2 y si "d>0" el resultado es 0. Se puede pensar en otras formas más de escribir la expresión aritmético relacional, sin embargo, lo importante es que sea fácil de entender y que los resultados sean únicos para cada posible caso, es decir no se debe permitir que dos o más casos devuelvan el mismo resultado.

4.3. EJEMPLOS

1. Elabore un programa que, empleando librerías C++, determine si un número dado es par, impar o cero. El programa debe contar con un módulo que devuelva 0 si el número es cero, 1 si es impar y 2 si es par.

Este problema ya fue resuelto, en el tema anterior con le estructura "if-else", ahora se resuelve con la estructura "switch". Lo único que realmente cambia con relación a dicha solución es el módulo "parImparCero", donde, dado que las condiciones no son de igualdad, se emplea la siguiente expresión aritmético-relacional:

```
(n==0)+(n%2==1)*2
```

Como se puede ver, esta expresión devuelve 1 si "n" es cero, 2 si "n" es impar y 0 si es par, siendo esos los tres casos de la estructura "switch". Empleando esta expresión, una posible solución es:

```
C:\HPV\programas\c++\tema4_2_2012>notepad ejem1.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

int leerNumero(){
  double x;
  cout.width(20);
  cout<<endl<<"Numero? ";
  cin>>x;
  if (fmod(x,1)){
    cout<<"Error: El numero debe ser entero"<<endl;
    exit(EXIT_FAILURE);
  }
  return int(x);
}

int parImparCero(int n){
  switch ((n==0)+(n%2==1)*2) {
    case 1: return 0;
    case 2: return 1;
    default: return 2;
  }
}

void mostrarSiPar(int r){
```

```
  cout.width(20);
  cout<<"El numero es: ";
  switch(r){
    case 2: cout<<"par"<<endl; break;
    case 1: cout<<"impar"<<endl; break;
    default: cout<<"cero"<<endl;
  }
}

int main(){
  int n,r;
  n=leerNumero();
  r=parImparCero(n);
  mostrarSiPar(r);
  return EXIT_SUCCESS;
}
```

Como se puede ver, también se ha empleado la estructura "switch" para mostrar los resultados, en este caso, sin embargo, no es necesario una expresión aritmético relacional, pues todos los casos son condiciones de igualdad.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema4_2_2012>g++ -c ejem1.cpp

C:\HPV\programas\c++\tema4_2_2012>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\tema4_2_2012>ejem1

            Numero? 145
      El numero es: impar

C:\HPV\programas\c++\tema4_2_2012>ejem1

            Numero? 784
      El numero es: par

C:\HPV\programas\c++\tema4_2_2012>ejem1

            Numero? 0
      El numero es: cero

C:\HPV\programas\c++\tema4_2_2012>ejem1

            Numero? 342.32
Error: El numero debe ser entero
```

2. Elabore un programa que, empleando librerías tipo C, determine si un año dado es o no bisiesto. Como un año "es" o "no es" bisiesto (no existen años más o menos bisiestos) el programa debe contar con una función que devuelva verdadero si el año es bisiesto y falso en caso contrario.

Este problema fue resuelto en el anterior tema con la estructura "if-else". Como se recordará, un año es bisiesto si es divisible entre cuatro, con excepción de los años que terminan en dos ceros, a los cuales se les debe quitar los dos ceros antes de verificar si son divisibles entre cuatro.

Una vez más, el único módulo que realmente cambia es el que determina si el año es o no bisiesto. Como en este caso se emplea la estructura "switch" (y las condiciones no son de igualdad), se debe emplear una expresión aritmético relacional:

```
(a%100==0 && a%400==0)+(a%100!=0 && a%4==0)*2
```

Como se puede ver, esta expresión devuelve 1 si el año termina en 2 ceros y es divisible entre 400 (es decir si es bisiesto); devuelve 2 si el año no es divisible entre 100 y es divisible entre 4 (es decir si es bisiesto) y devuelve 0 en caso contrario (es decir si no es bisiesto).

Con esta expresión una posible solución es:

```
C:\HPV\programas\c++\tema4_2_2012>notepad ejem2.cpp
```

```
#include <cstdio>
#include <cstdlib>
#include <cmath>

unsigned leerAnio(){
  double a;
  printf("\n%15s? ","Año");
  scanf("%lf",&a);
  if (fmod(a,1)){
    printf("Error: El año debe ser entero\n");
    exit(EXIT_FAILURE);
  }
  if (a<=0) {
    printf("Error: El año debe ser positivo\n");
    exit(EXIT_FAILURE);
  }
  return (unsigned)a;
}

int bisiesto(unsigned a){
  switch((a%100==0 && a%400==0)+(a%100!=0 && a%4==0)){
    case 1:
    case 2: return 1;
    default: return 0;
  }
}

void mostrarSiBisiesto(unsigned a, int r){
  printf("%16s %u ","El año",a);
  switch(r){
    case 1: printf("es bisiesto\n"); break;
    default: printf("no es bisiesto\n");
  }
}


int main(){
  unsigned a;
  int r;
  a=leerAnio();
  r=bisiesto(a);
  mostrarSiBisiesto(a,r);
  return EXIT_SUCCESS;
}
```

En la estructura "switch" de la función "bisiesto" no se emplea el coman-

do "break" porque se devuelve directamente el resultado con la instrucción "return" (que como se sabe devuelve el resultado y sale de la función).

Note también que el resultado de la función "bisiesto" es devuelto como un entero (no como un booleano "bool"). Se ha procedido así simplemente para demostrar que en C y en C++ el número 1 es equivalente al valor lógico verdadero (true) y 0 a falso (false).

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPV\programas\c++\tema4_2_2012>g++ -c ejem2.cpp

C:\HPV\programas\c++\tema4_2_2012>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? 2000
        El a±o 2000 es bisiesto

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? 1900
        El a±o 1900 no es bisiesto

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? 2012
        El a±o 2012 es bisiesto

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? 2006
        El a±o 2006 no es bisiesto

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? 198.32
Error: El a±o debe ser entero

C:\HPV\programas\c++\tema4_2_2012>ejem2

          A±o? -2012
Error: El a±o debe ser positivo
```

3. Elabore un programa que, empleando librerías tipo C++, lea tres números reales, encuentre el mayor de ellos y muestre el mismo en pantalla.

Este problema ha sido resuelto también con la estructura "if-else" (y es la forma más eficiente y clara de hacerlo), en este tema (y sólo a manera de práctica) es vuelto a resolver con la estructura "switch".

Con la estructura "switch" es necesario modificar la lógica empleando la siguiente expresión aritmético lógica:

```
(a>=b && a>=c)+(b>=a && b>=c)*2+(c>=a %% c>=b)*3
```

Esta expresión devuelve 1 si "a" es mayor que "b" y "c", 2 si "b" es mayor que "a" y "c" y 3 si "c" es mayor que "a" y "b". Con esta modificación en la lógica, una posible solución es:

```
C:\Users\Hernan>notepad ejem3.cpp
```

```
#include <iostream>
#include <cstdlib>
using namespace std;

void leerNumeros(double &a, double &b, double &c){
```

```
  cout.width(25);
  cout<<endl<<"Introduzca 3 numeros: "<<endl;
  cout.width(25);
  cout<<"Primer numero? ";
  cin>>a;
  cout.width(25);
  cout<<"Segundo numero? ";
  cin>>b;
  cout.width(25);
  cout<<"Tercer numero? ";
  cin>>c;
}

double mayor(double a, double b, double c){
  switch((a>=b && a>=c)+(b>=a && b>=c)*2+(c>=a && c>=b)*3){
    case 1: return a;
    case 2: return b;
    default: return c;
  }
}

void mostrarMayor(double m){
  cout.width(25);
  cout<<endl<<"El mayor valor es: "<<m<<endl;
}

int main(){
  double a,b,c,m;
  leerNumeros(a,b,c);
  m=mayor(a,b,c);
  mostrarMayor(m);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\Users\Hernan>g++ -c ejem3.cpp

C:\Users\Hernan>g++ -o ejem3 ejem3.o

C:\Users\Hernan>ejem3

   Introduzca 3 numeros:
         Primer numero? 4.5
        Segundo numero? 1.2
         Tercer numero? 3.9

      El mayor valor es: 4.5

C:\Users\Hernan>ejem3

   Introduzca 3 numeros:
         Primer numero? 1.2
        Segundo numero? 4.5
         Tercer numero? 3.9

      El mayor valor es: 4.5

C:\Users\Hernan>ejem3

   Introduzca 3 numeros:
```

```
          Primer numero? 3.9
         Segundo numero? 1.2
          Tercer numero? 4.5

       El mayor valor es: 4.5
```

4. Elabore un programa que, empleando librerías tipo C, lea tres números enteros, los ordene ascendentemente y muestre los números ordenados.

Una vez más, este problema ya fue resuelto con la estructura "if-else". Para emplear la estructura "switch", la lógica debe se modificada de manera que la expresión lógica devuelva los seis posibles casos:

```
(a<=b && b<=c)+(a<=c && c<=b)*2+(b<=a && a<=c)*3+(b<=c && c<=a)*4+(c<=a
&& a<=b)*5
```

Esta expresión devuelve 1 si el orden es "a, b, c"; devuelve 2 si el orden es "a, c, b", devuelve 3 si el orden es "b, a, c"; devuelve 4 si el orden es "b, c, a"; devuelve 5 si el orden es "c, a, b" y devuelve 0 si el orden es "c, b, a" (el caso por defecto).

Con esta modificación a la lógica una posible solución al problema es:

```
C:\HPV\programas\c++\tema4_2_2012>notepad ejem4.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

int validar(double x){
  if (fmod(x,1)){
    printf("Error: El numero debe ser entero.\n");
    exit(EXIT_FAILURE);
  }
  return (int)x;
}

void leerNumeros(int &a, int &b, int &c){
  double x,y,z;
  printf("\n%20s: \n","Numeros a ordenar");
  printf("%20s? ","Primer numero");
  scanf("%lf",&x);
  a=validar(x);
  printf("%20s? ","Segundo numero");
  scanf("%lf",&y);
  b=validar(y);
  printf("%20s? ","Tercer numero");
  scanf("%lf",&z);
  c=validar(z);
}

void ordenar(int &x, int &y, int &z){
  int a=x, b=y, c=z;
  switch((a<=b && b<=c)+(a<=c && c<=b)*2+(b<=a && a<=c)*3+
         (b<=c && c<=a)*4+(c<=a && a<=b)*5){
    case 1: break;
    case 2: y=c; z=b; break;
    case 3: x=b; y=a; break;
    case 4: x=b; y=c; z=a; break;
    case 5: x=c; y=a; z=b; break;
```

```
    default: x=c; z=a; break;
  }
}

void mostrarNumeros(int a, int b, int c){
  printf("\n%20s: %d\n","Numeros ordenados",a);
  printf("%20s  %d\n"," ",b);
  printf("%20s  %d\n"," ",c);
}

int main(){
  int a,b,c;
  leerNumeros(a,b,c);
  ordenar(a,b,c);
  mostrarNumeros(a,b,c);
  return EXIT_SUCCESS;
}
```

Observe que en el módulo "ordenar" sólo se asignan valores a las variables que cambian de posición. Si bien con ello el código se hace más corto y eficiente, se hace también más confuso, por lo que (a pesar de ser una solución más larga y menos eficiente), una solución más fácil de mantener es:

```
void ordenar(int &x, int &y, int &z){
  int a=x, b=y, c=z;
  switch((a<=b && b<=c)+(a<=c && c<=b)*2+(b<=a && a<=c)*3+
         (b<=c && c<=a)*4+(c<=a && a<=b)*5){
    case 1: x=a; y=b; z=c; break;
    case 2: x=a; y=c; z=b; break;
    case 3: x=b; y=a; z=c; break;
    case 4: x=b; y=c; z=a; break;
    case 5: x=c; y=a; z=b; break;
    default: x=c; y=b; z=a; break;
  }
}
```

Donde se identifican claramente los 6 posibles casos, facilitando así el mantenimiento y corrección de errores. Una alternativa quizá aún mas confusa se logra intercambiando los valores según el caso que corresponda:

```
void ordenar(int &a, int &b, int &c){
  int aux;
  switch((a<=b && b<=c)+(a<=c && c<=b)*2+(b<=a && a<=c)*3+
         (b<=c && c<=a)*4+(c<=a && a<=b)*5){
    case 1: break;
    case 2: aux=c; c=b; b=aux; break;
    case 3: aux=b; b=a; a=aux; break;
    case 4: aux=c; c=a; a=aux; aux=b; b=a; a=aux; break;
    case 5: aux=c; c=a; a=aux; aux=c; c=b; b=aux; break;
    default: aux=c; c=a; a=aux; break;
  }
}
```

Por supuesto, con las tres alternativas se obtienen los mismos resultados, el elegir una u otra depende, sobre todo, de cuán fácil resulte de entender la lógica que resuelve el problema.

Compilando el programa (con cualquiera de las alternativas), creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPU\programas\c++\tema4_2_2012>g++ -c ejem4.cpp

C:\HPU\programas\c++\tema4_2_2012>g++ -o ejem4 ejem4.o

C:\HPU\programas\c++\tema4_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 23
      Segundo numero? 12
       Tercer numero? 17

   Numeros ordenados: 12
                      17
                      23

C:\HPU\programas\c++\tema4_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 17
      Segundo numero? 23
       Tercer numero? 12

   Numeros ordenados: 12
                      17
                      23

C:\HPU\programas\c++\tema4_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 17
      Segundo numero? 12
       Tercer numero? 23

   Numeros ordenados: 12
                      17
                      23

C:\HPU\programas\c++\tema4_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 23
      Segundo numero? 12.45
Error: El numero debe ser entero.
```

5. Elabore un programa que, empleando librerías tipo C++, encuentre y muestre (con 9 dígitos de precisión) las soluciones (raíces) reales y complejas de una ecuación cuadrática.

Una vez más, este problema ha sido resuelto con la estructura "if-else", para resolverla con la estructura "switch" se crea una expresión aritmético relacional similar a la del primer ejemplo:

$$(d>0)+(d<0)*2$$

Esta expresión devuelve 1 si el discriminante es mayor a cero (raíces reales y diferentes), devuelve 2 si el discriminante es negativo (raices complejas) y devuelve 0 el discriminante es 0 (raíces reales e iguales).

Con esta modificación, una posible solución del problema es:

```
C:\HPU\programas\c++\tema4_2_2012>notepad ejem5.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
```

```
using namespace std;

void leerCoeficientes(double &a, double &b, double &c){
  cout.width(18);
  cout<<endl<<"Coeficientes de: "<<"a*x^2+b*x+c=0"<<endl;
  cout.width(18);
  cout<<"a? ";
  cin>>a;
  cout.width(18);
  cout<<"b? ";
  cin>>b;
  cout.width(18);
  cout<<"c? ";
  cin>>c;
}

void cuadratica(double a, double b, double c,
                double &r1, double &r2, double &im){
  double r, d=b*b-4*a*c;
  switch((d>0)+(d<0)*2){
    case 1: r=sqrt(d);
            r1=(-b-r)/(2*a);
            r2=(-b+r)/(2*a);
            im=0;
            break;
    case 2: r1=r2=-b/(2*a);
            im=sqrt(-d)/(2*a);
            break;
    default: r1=r2=-b/(2*a);
             im=0;
  }
}

void mostrarRaices(double r1, double r2, double im){
  cout.width(18);
  cout<<endl<<"Raices: "<<endl;
  switch(im==0){
    case 1: cout.width(18); cout.precision(9);
            cout<<"r1: "<<r1<<endl;
            cout.width(18);
            cout<<"r2: "<<r2<<endl;
            break;
    default: cout.width(18); cout.precision(9);
             cout<<"r1: "<<r1<<" + "<<im<<"i"<<endl;
             cout.width(18);
             cout<<"r2: "<<r2<<" - "<<im<<"i"<<endl;
  }
}

int main(){
  double a,b,c,r1,r2,im;
  leerCoeficientes(a,b,c);
  cuadratica(a,b,c,r1,r2,im);
  mostrarRaices(r1,r2,im);
  return EXIT_SUCCESS;
```

```
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema4_2_2012>g++ -c ejem5.cpp

C:\HPV\programas\c++\tema4_2_2012>g++ -o ejem5 ejem5.o

C:\HPV\programas\c++\tema4_2_2012>ejem5

 Coeficientes de: a*x^2+b*x+c=0
                  a? 1
                  b? 2
                  c? 3

          Raices:
              r1: -1 + 1.41421356i
              r2: -1 - 1.41421356i

C:\HPV\programas\c++\tema4_2_2012>ejem5

 Coeficientes de: a*x^2+b*x+c=0
                  a? 1
                  b? -10
                  c? 21

          Raices:
              r1: 3
              r2: 7

C:\HPV\programas\c++\tema4_2_2012>ejem5

 Coeficientes de: a*x^2+b*x+c=0
                  a? 1
                  b? -14
                  c? 49

          Raices:
              r1: 7
              r2: 7
```

4.4. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema4" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa que, empleando la estructura "switch" y librerías tipo C++, permita determinar si un número dado es positivo, negativo o cero. El programa debe contar con un módulo que devuelva 0 si el número es cero, -1 si es negativo y 1 si es positivo.

2. Elabore un programa que, empleando la estructura "switch" y librerías tipo C, lea los tres lados de un triángulo y determine si los mismos conforman o no un triángulo. Tres lados conforman un triángulo si, **en todos los casos**, la suma de dos de ellos es mayor al tercero. El programa debe contar con un módulo que devuelva verdadero si los lados conforman un triángulo y falso en caso contrario.

3. Elabore un programa que, empleando la estructura "switch" y librerías tipo C++, lea cuatro números reales y muestre el menor de ellos.

4. Elabore un programa que, empleando la estructura “switch” y librerías tipo C, lea tres números enteros, los ordene descendentemente y los muestre en pantalla.

5. Elabore un programa que, empleando la estructura “switch” y librerías tipo C, lea los coeficientes de la ecuación cuadrática “$a+bx+cx^2$” y muestre las soluciones reales e imaginarias de la misma. En el módulo que resuelve el problema, la parte real para los casos reales e iguales y complejos, debe ser calculado una sola vez.

5. SELECCION-3

En este tema continua el repaso de las estructuras selectivas, estudiando en este caso el operador "?".

5.1. EL OPERADOR CONDICIONAL "?"

C/C++ cuentan con el operador condicional "?" que puede ser empleado en lugar de la instrucción "if" en aquellos casos donde la lógica involucrada es relativamente simple.

El operador condicional "?" tiene la misma lógica que la estructura "if" (y por lo tanto tiene la misma representación gráfica) siendo su sintaxis:

condición **?** instrucción_1 : instrucción_2;

Sin embargo, a diferencia de "if-else", este operador devuelve un resultado, que es el valor devuelto por la "instrucción_1" (cuando la "condición" es verdadera) o por la "instrucción_2" (cando la condición es falsa).

El que el operador "?" devuelva un resultado lo hace particularmente adecuado para ser empleado en sentencias de asignación:

variable = condición **?** instrucción_1 : instrucción_2;

O para devolver directamente el resultado de una función:

return condición **?** instrucción_1; instrucción_2;

Si bien este operador se emplea sobre todo con expresiones lógicas simples, particularmente sentencias de asignación y/o retorno de resultados, es posible emplearla también con lógicas un tanto más complejas, las que pueden constar de dos o más instrucciones consecutivas. En tales casos, dichas instrucciones deben estar separadas con comas (no puntos y comas como ocurre en una secuencia normal).

La coma, no sólo funciona con este operador, sino también con cualquier otra estructura C/C++ y cuando es empleada de esta manera (como un separador secuencial) las instrucciones se ejecutan una a continuación de otra, pero no devuelven ningún resultado (se evalúan como "void"), de manera que el valor de la última expresión (la que se encuentra más a la derecha) es el valor de la instrucción en su conjunto.

Así, al ejecutar las siguientes instrucciones:

```
x = 2;
v = (y=2*x,z=3*y,4*z);
```

La variable "y" queda con el número 4, la variable "z" con 12 y la variable "v" con 48.

En conjunción con el operador "?", permite evaluar expresiones compuestas como la siguiente:

```
return a>b ? (b=2*a,c=3*b+log(a),exp(b+c)) : (b=3*a,c=b+sqrt(a),b+c);
```

En este tema (y sólo para adquirir práctica) se empleará la coma y el operador "?" para resolver todos los problemas, sin embargo, en situaciones reales, solo deben ser empleados para resolver problemas simples.

5.2. EJEMPLOS

1. Elabore un programa que, empleando librerías C y el operador "?", determine si un número dado es par, impar o cero. El programa debe contar con

un módulo que devuelva 0 si el número es cero, 1 si es impar y 2 si es par.

Este problema ya fue resuelto, en los anteriores temas, siendo la lógica que resuelve el problema la misma. Lo único que cambia es que ahora se implementa con el operador "?", siendo una posible solución:

```
C:\HPV\programas\c++\tema_5_2_2012>notepad ejem1.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

int leerNumero(){
  double x;
  printf("\n%20s? ","Numero");
  scanf("%lf",&x);
  if (fmod(x,1)){
    printf("Error: El numero debe ser entero.\n");
    exit(EXIT_FAILURE);
  }
  return (int)x;
}


int parImparCero(int n){
  return n==0 ? 0 : (n%2 ? 1: 2);
}

void mostrarSiPar(int r){
  printf("%20s: %s\n","El numero es",
    r==0 ? "cero" : (r==1 ? "impar": "par"));
}


int main(){
  int n,r;
  n=leerNumero();
  r=parImparCero(n);
  mostrarSiPar(r);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema_5_2_2012>g++ -c ejem1.cpp

C:\HPV\programas\c++\tema_5_2_2012>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\tema_5_2_2012>ejem1

              Numero? 56
       El numero es: par

C:\HPV\programas\c++\tema_5_2_2012>ejem1

              Numero? 179
       El numero es: impar

C:\HPV\programas\c++\tema_5_2_2012>ejem1
```

```
            Numero? 0
       El numero es: cero

C:\HPU\programas\c++\tema_5_2_2012>ejem1

            Numero? 389.234
Error: El numero debe ser entero.
```

Como de costumbre, esta no es la única posible solución, se puede (sin cambiar la lógica) pensar en otras formas más de implementar el código, por ejemplo la impresión de resultados se puede lograr también con:

```
r==1 ? "impar" : (r ? "par" : "cero")
```

O también con:

```
r ? (r==1 ? "impar" : "par") : "cero"
```

O con:

```
!r ? "cero" : (r==2 ? "par" : "impar")
```

Y de esa forma se pueden plantear otras alternativas más.

2. Elabore un programa que, empleando librerías tipo C++ y el operador "?", determine si un año dado es o no bisiesto. Como un año "es" o "no es" bisiesto (no existen años más o menos bisiestos) el programa debe contar con una función que devuelva verdadero si el año es bisiesto y falso en caso contrario.

Como se recordará, un año es bisiesto si es divisible entre cuatro, con excepción de los años que terminan en dos ceros, a los cuales se les debe quitar los dos ceros antes de verificar si son divisibles entre cuatro (o lo que es lo mismo comprobar si son divisibles entre 400).

Una posible solución, empleando el operador "?" es:

```
C:\HPU\programas\c++\tema_5_2_2012>notepad ejem2.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

unsigned leerAnio(){
  double a;
  cout.width(15);
  cout<<endl<<"Año? ";
  cin>>a;
  if (fmod(a,1)){
    cout<<"Error: El año debe ser entero."<<endl;
    exit(EXIT_FAILURE);
  }
  if (a<0) {
    cout<<"Error: El año debe ser positivo."<<endl;
    exit(EXIT_FAILURE);
  }
  return unsigned(a);
}

bool bisiesto(unsigned a){
  a = a%100 ? a : a/100;
```

```
  return a%4 ? false : true;
}


void mostrarSiBisiesto(unsigned a, bool r){
  cout.width(15);
  cout<<"El año: "<<a<<" "<<(r ? "es bisiesto" : "no es bisiesto")<<endl;
}

int main(){
  unsigned a;
  bool r;
  a=leerAnio();
  r=bisiesto(a);
  mostrarSiBisiesto(a,r);
  return EXIT_SUCCESS;
}
```

Como de costumbre el ejemplo constituye sólo una de las muchas posibles soluciones. Así se puede averiguar si un año es o no bisiesto también con (módulo "bisiesto"):

```
a = a%100==0 ? a/100 : a;
return !(a%4) ? true : false;
```

O también con:

```
a = a%100!=0 ? a : a/100;
return !(a%4);
```

O con:

```
return a%100 ? !(a%4) : !(a%400);
```

Y muchas otras posibles soluciones más. Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPV\programas\c++\tema_5_2_2012>g++ -c ejem2.cpp

C:\HPV\programas\c++\tema_5_2_2012>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\tema_5_2_2012>ejem2

          A±o? 2000
       El a±o: 2000 es bisiesto

C:\HPV\programas\c++\tema_5_2_2012>ejem2

          A±o? 1900
       El a±o: 1900 no es bisiesto

C:\HPV\programas\c++\tema_5_2_2012>ejem2

          A±o? 2006
       El a±o: 2006 no es bisiesto

C:\HPV\programas\c++\tema_5_2_2012>ejem2

          A±o? 1994.323
Error: El a±o debe ser entero.

C:\HPV\programas\c++\tema_5_2_2012>ejem2

          A±o? -2012
Error: El a±o debe ser positivo.
```

3. Elabore un programa que, empleando librerías tipo C y el operador "?", lea tres números reales, encuentre el mayor de ellos y muestre el mismo en pantalla.

Este problema (al igual que los anteriores) ya ha sido resuelto con la estructuras "if" y "switch", ahora simplemente se la codifica con el operador "?", siendo una posible solución la siguiente:

```
C:\HPU\programas\c++\tema_5_2_2012>notepad ejem3.cpp
```

```
#include <cstdio>
#include <cstdlib>

void leerNumeros(double &a, double &b, double &c){
  printf("\n%25s: \n","Introduzca 3 numeros");
  printf("%25s? ","Primer numero");
  scanf("%lf",&a);
  printf("%25s? ","Segundo numero");
  scanf("%lf",&b);
  printf("%25s? ","Tercer numero");
  scanf("%lf",&c);
}

double mayor(double a, double b, double c){
  b = a>b ? a : b;
  return b>c ? b : c;
}

void mostrarMayor(double m){
  printf("\n%25s: %.12g\n","El mayor valor es",m);
}

int main(){
  double a,b,c,m;
  leerNumeros(a,b,c);
  m=mayor(a,b,c);
  mostrarMayor(m);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPU\programas\c++\tema_5_2_2012>g++ -c ejem3.cpp

C:\HPU\programas\c++\tema_5_2_2012>g++ -o ejem3 ejem3.o

C:\HPU\programas\c++\tema_5_2_2012>ejem3

     Introduzca 3 numeros:
           Primer numero? 4.5
         Segundo numero? 1.2
          Tercer numero? 3.9

        El mayor valor es: 4.5

C:\HPU\programas\c++\tema_5_2_2012>ejem3

     Introduzca 3 numeros:
           Primer numero? 1.2
         Segundo numero? 3.9
          Tercer numero? 4.5
```

```
        El mayor valor es: 4.5

C:\HPV\programas\c++\tema_5_2_2012>ejem3

     Introduzca 3 numeros:
            Primer numero? 1.2
           Segundo numero? 3.9
            Tercer numero? 4.5

        El mayor valor es: 4.5
```

4. Elabore un programa que, empleando librerías tipo C++ y el operador "?", lea tres números enteros, los ordene ascendentemente y muestre los números ordenados.

Una vez más, este problema ha sido resuelto con las estructuras "if-else" y "switch". Como ya se ha visto en los anteriores ejemplos, la solución con el operador "?" es parecida a la de la estructura "if-else", siendo una posible solución:

```
C:\HPV\programas\c++\tema_5_2_2012>notepad ejem4.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

int validar(double x){
  if (fmod(x,1)){
    cout<<"Error: El numero debe ser entero."<<endl;
    exit(EXIT_FAILURE);
  }
  return int(x);
}

void leerNumeros(int &a, int &b, int &c){
  double x,y,z;
  cout.width(20);
  cout<<endl<<"Numeros a ordenar"<<": "<<endl;
  cout.width(20);
  cout<<"Primer numero"<<"? ";
  cin>>x;
  a=validar(x);
  cout.width(20);
  cout<<"Primer numero"<<"? ";
  cin>>y;
  b=validar(y);
  cout.width(20);
  cout<<"Primer numero"<<"? ";
  cin>>z;
  c=validar(z);
}

void ordenar(int &a, int &b, int &c){
  int aux;
  b = a>b ? aux=a, a=b, aux : b;
  c = b>c ? aux=b, b=c, aux : c;
  b = a>b ? aux=a, a=b, aux : b;
}
```

```
void mostrarNumeros(int a, int b, int c){
  cout.width(20);
  cout<<endl<<"Numeros ordenados"<<": "<<a<<endl;
  cout.width(20);
  cout<<""<<"  "<<b<<endl;
  cout.width(20);
  cout<<""<<"  "<<c<<endl;

}

int main(){
  int a,b,c;
  leerNumeros(a,b,c);
  ordenar(a,b,c);
  mostrarNumeros(a,b,c);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\tema_5_2_2012>g++ -c ejem4.cpp

C:\HPV\programas\c++\tema_5_2_2012>g++ -o ejem4 ejem4.o

C:\HPV\programas\c++\tema_5_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 23
       Primer numero? 12
       Primer numero? 17

   Numeros ordenados: 12
                      17
                      23

C:\HPV\programas\c++\tema_5_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 17
       Primer numero? 23
       Primer numero? 12

   Numeros ordenados: 12
                      17
                      23

C:\HPV\programas\c++\tema_5_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 17
       Primer numero? 12
       Primer numero? 23

   Numeros ordenados: 12
                      17
                      23

C:\HPV\programas\c++\tema_5_2_2012>ejem4

   Numeros a ordenar:
       Primer numero? 23
       Primer numero? 12.45
Error: El n·mero debe ser entero.
```

5. Elabore un programa que, empleando librerías tipo C y el operador "?", encuentre y muestre (con 9 dígitos de precisión) las soluciones (raíces) reales y complejas de una ecuación cuadrática.

Este problema fue resuelto también en los temas anteriores. Una posible solución, empleando el operador "?", es:

```
C:\HPV\programas\c++\2-2012\tema5>notepad ejem5.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

void leerCoeficientes(double &a, double &b, double &c){
  printf("\n%18s: %s\n","Coeficientes de","a*x^2+b*x+c=0");
  printf("%18s? ","a");
  scanf("%lf",&a);
  printf("%18s? ","b");
  scanf("%lf",&b);
  printf("%18s? ","c");
  scanf("%lf",&c);
}

void cuadratica(double a, double b, double c,
                double &r1, double &r2, double &im){
  double d=b*b-4*a*c;
  r1 = d>0 ? (-b-sqrt(d))/(2*a) : -b/(2*a);
  r2 = d>0 ? (-b+sqrt(d))/(2*a) : r1;
  im = d<0 ? sqrt(-d)/(2*a) : 0;
}

void mostrarRaices(double r1, double r2, double im){
  printf("\n%18s:\n","Raices");
  im==0 ?
    (printf("%18s: %.9g\n","r1",r1),
     printf("%18s: %.9g\n","r2",r2)) :
    (printf("%18s: %.9g + %.9gi\n","r1",r1,im),
     printf("%18s: %.9g - %.9gi\n","r2",r2,im));
}

int main(){
  double a,b,c,r1,r2,im;
  leerCoeficientes(a,b,c);
  cuadratica(a,b,c,r1,r2,im);
  mostrarRaices(r1,r2,im);
  return EXIT_SUCCESS;
}
```

Observe que en el módulo "mostrarRaices" no se emplea el resultado devuelto por el operador "?" (pues en realidad dicho resultado no existe). En este sentido se reitera que la solución propuesta es sólo una de las muchas posibles formas en que se puede resolver el problema.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\2-2012\tema5>g++ -c ejem5.cpp
```

```
C:\HPV\programas\c++\2-2012\tema5>g++ -o ejem5 ejem5.o

C:\HPV\programas\c++\2-2012\tema5>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? 2
                 c? 3

            Raices:
                r1: -1 + 1.41421356i
                r2: -1 - 1.41421356i

C:\HPV\programas\c++\2-2012\tema5>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? -14
                 c? 49

            Raices:
                r1: 7
                r2: 7

C:\HPV\programas\c++\2-2012\tema5>ejem5

   Coeficientes de: a*x^2+b*x+c=0
                 a? 1
                 b? -10
                 c? 21

            Raices:
                r1: 3
                r2: 7
```

5.3. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema5" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa que, empleando el operador "?" y librerías tipo C, permita determinar si un número dado es positivo, negativo o cero. El programa debe contar con un módulo que devuelva 0 si el número es cero, -1 si es negativo y 1 si es positivo.

2. Elabore un programa que, empleando el operador "?" y librerías tipo C++, lea los tres lados de un triángulo y determine si los mismos conforman o no un triángulo. Tres lados conforman un triángulo si, **en todos los casos**, la suma de dos de ellos es mayor al tercero. El programa debe contar con un módulo que devuelva verdadero si los lados conforman un triángulo y falso en caso contrario.

3. Elabore un programa que, empleando el operador "?" y librerías tipo C, lea cuatro números reales y muestre el menor de ellos.

4. Elabore un programa que, empleando el operador "?" y librerías tipo C++, lea tres números enteros, los ordene descendentemente y los muestre en pantalla.

5. Elabore un programa que, empleando el operador "?" y librerías tipo C++, lea los coeficientes de la ecuación cuadrática "$a+bx+cx^2$" y muestre las

soluciones reales e imaginarias de la misma. En el módulo que resuelve el problema, la parte real para los casos reales e iguales y complejos, debe ser calculado una sola vez.

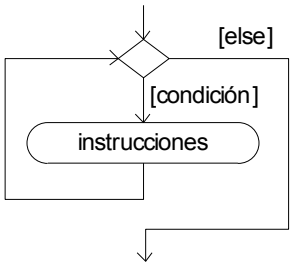
6. ITERACIÓN 1

En este tema se estudia la primera estructura iterativa, la estructura "while". Las estructuras iterativas son aquellas que permiten repetir una o más instrucciones 2 o más veces.

De acuerdo al teorema de la programación estructurada, la única estructura necesaria para resolver cualquier problema iterativo es la estructura "while", sin embargo, en ocasiones se puede resolver un problema de manera más clara y eficiente empleando otras estructuras iterativas, por lo que serán estudiadas en los siguientes temas.

6.1. LA ESTRUCTURA "WHILE"

La lógica de la estructura "while" es la siguiente:



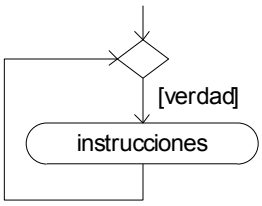
Es decir que en esta estructura las "instrucciones" se repite mientras la "condición" es verdadera. Cuando la "condición" es falsa el proceso concluye y el programa continua con la instrucción que le sigue a la estructura "while".

La forma de codificar esta instrucción es:

```
while (condición) {
    instrucciones;
}
```

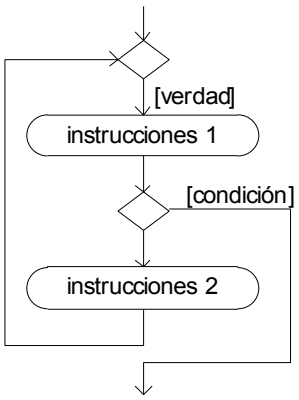
Si en lugar de varias "instrucciones" se tiene una sola, no son necesarias las llaves, aunque el emplearlas no constituye un error.

Cuando la "condición" es verdadera (true o 1) el proceso iterativo no concluye nunca, dando lugar a un ciclo infinito:



Por supuesto, un ciclo infinito como tal, es algo que debe evitarse en un programa pues constituye un error, sin embargo, resulta de utilidad práctica cuando la lógica que resuelve el problema sale del ciclo en algún punto intermedio del mismo, tal como se puede ver en la lógica representada en el diagrama de actividades de la siguiente página.

Para codificar ese tipo de lógica se crea un ciclo infinito y se sale del mismo con el modificador "**break**". Este modificador termina inmediatamente la ejecución del ciclo y el programa continua con la instrucción que sigue a la estructura iterativa (el modificador "break" no sólo es válido en la estructura "while", sino en todas las estructuras iterativas).



Si el módulo termina después del ciclo, se puede emplear la instrucción "return" en lugar de la instrucción "break". Como se recordará, la instrucción "return" termina inmediatamente la ejecución del módulo (de la función) devolviendo un resultado.

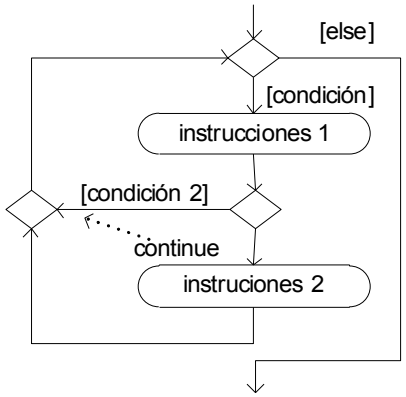
Entonces, si existen otras instrucciones después del ciclo (de la estructura "while") se la codifica de la siguiente forma:

```
while (true) {
  instrucciones 1;
  if (condición) break;
  instrucciones 2;
}
```

Pero si la función termina inmediatamente después del ciclo, puede ser codificada también con:

```
while (true) {
  instrucciones 1;
  if (condición) return resultado;
  instrucciones 2;
}
```

Aunque menos útil (porque generalmente se puede emplear una estructura "if-else" en su lugar) se cuenta también con el modificador "**continue**". Este modificador interrumpe la ejecución del ciclo actual y pasa a ejecutar el siguiente ciclo:



Esta lógica se codifica de la siguiente forma:

```
while (condición) {
  instrucciones 1;
  if (condición 2) continue;
  instrucción 2;
```

```
}
```

Las instrucciones (en todos los casos) hacen referencia o a una instrucción o a una secuencia de instrucción separadas con puntos y comas.

Los modificadores "break", "continue" y el comando "return" pueden ser empleadas con todas las estructuras iterativas (no sólo con la estructura "while"), tal como se verá en los siguientes temas.

6.2. EJEMPLOS

1. Elabore un programa que, empleando librerías tipo C y la estructura "while", invierta los dígitos de un número entero (positivo o negativo). Por ejemplo si el número es 12345 el resultado debe ser 54321.

Este problema puede ser resuelto de la siguiente forma: se extraen los dígitos del número, desde el último hasta el primero y se los va almacenando, en el orden inverso, en otra variable.

Los números pueden ser extraídos (desde el último hasta el primero) calculando el residuo de su división entre 10, luego ese número puede ser descartado del número original calculando su cociente entre 10.

Para almacenar los números extraídos, en el orden inverso, se inicia una variable en cero, se la multiplica por 10 y se le suma el número extraído (se multiplica la variable por 10 para dejar un dígito libre a la derecha).

Por ejemplo si el número a invertir es "n=12345" y la variable donde se almacena el resultado es "ni=0", las operaciones para invertir el número son:

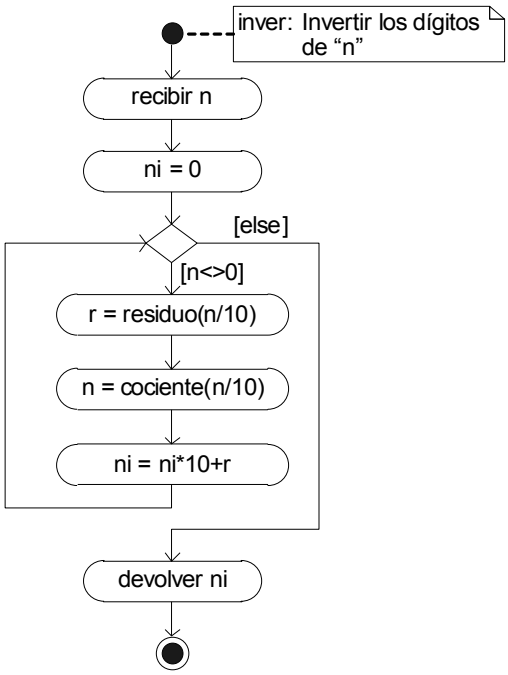
```
r = residuo(n/10) = residuo(12345/10) = 5;
n = cociente(n/10) = cociente(12345/10) = 1234;
ni = ni*10+r = 0*10+5 = 0+5 = 5;

r = residuo(n/10) = residuo(1234/10) = 4;
n = cociente(n/10) = cociente(1234/10) = 123;
ni = ni*10+r = 5*10+4 = 50+4 = 54;

r = residuo(n/10) = residuo(123/10) = 3;
n = cociente(n/10) = cociente(123/10) = 12;
ni = ni*10+r = 54*10+3 = 540+3 = 543;

r = residuo(n/10) = residuo(12/10) = 2;
n = cociente(n/10) = cociente(12/10) = 1;
ni = ni*10+r = 543*10+2 = 5430+2 = 5432;

r = residuo(n/10) = residuo(1/10) = 1;
n = cociente(n/10) = cociente(1/10) = 0;
ni = ni*10+r = 5432*10+1 = 54320+1 = 54321;
```

Como se puede ver, el proceso se repite "mientras" el valor de la variable a invertir es diferente de 0, siendo esta la condición del ciclo.

El algoritmo, en forma de diagrama de actividades, se presenta en la siguiente página y el código respectivo es el siguiente:

```
C:\HPV\programas\c++\2-2012\tema6>notepad ejem1.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>
```




```
int leerNumero(){
  double n;
  printf("\n%20s? ","Numero a invertir");
  scanf("%lf",&n);
  if (fmod(n,1)){
    printf("Error: El numero debe ser entero.\n");
    exit(EXIT_FAILURE);
  }
  return (int)n;
}

int invertir(int n){
  int r,ni=0;
  while(n!=0){
    r=n%10;
    n=n/10;
    ni=ni*10+r;
  }
  return ni;
}

void mostrarNumero(int ni){
  printf("%20s: %d\n","Numero invertido",ni);
}


int main(){
  int n,ni;
  n=leerNumero();
  ni=invertir(n);
  mostrarNumero(ni);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\2-2012\tema6>g++ -c ejem1.cpp

C:\HPV\programas\c++\2-2012\tema6>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\2-2012\tema6>ejem1

   Numero a invertir? 12345
    Numero invertido: 54321

C:\HPV\programas\c++\2-2012\tema6>ejem1

   Numero a invertir? -372872
    Numero invertido: -278273

C:\HPV\programas\c++\2-2012\tema6>ejem1

   Numero a invertir? 234.534
Error: El numero debe ser entero.
```

2. Elabore un programa que, empleando librerías tipo C++ y la estructura "while", calcule la raíz cuadrada de un número real.

La raíz cuadrada de un número "n" puede ser calculada con la ecuación deducida aplicando el método de Newton:

$$x_2 = \frac{1}{2}\left(x_1 + \frac{n}{x_1}\right)$$

Esta ecuación es por naturaleza iterativa (repetitiva), se comienza asumiendo un valor para la raíz "x_1", con ese valor se calcula "x_2" y se compara con el valor asumido: si son iguales el proceso concluye (siendo la solución x_2), caso contrario el proceso se repite haciendo que "x_1" tome el valor de "x_2".

Por ejemplo, para calcular la raíz cuadrada de 2 (n=2), con 9 dígitos de precisión, se puede asumir (suponer) que la solución es 1 (x1=1), y con este valor se calcula "x2".

x2=(x1+n/x1)=(1+2/1)/2=1.5

Como "x1" y "x2" no son iguales, el procesos e repite, pero empleando ahora como valor asumido el valor de "x2":

x1=x2=1.5
x2=(x1+n/x1)=(1.5+2/1.5)/2=1.5=1.41666667

Una vez más "x1" y "x2" no son iguales, por lo que el proceso se repite:

x1=x2=1.41666667
x2=(x1+n/x1)=(1.41666667+2/1.41666667)/2=1.41421569

Y como todavía no son iguales, el proceso se repite, continuando así hasta que se cumpla la igualdad.

x1=x2=1.41421569
x2=(x1+n/x1)=(1.41421569+2/1.41421569)/2=1.41421356

x1=x2=1.41421356
x2=(x1+n/x1)=(1.41421356+2/1.41421356)/2=1.41421356

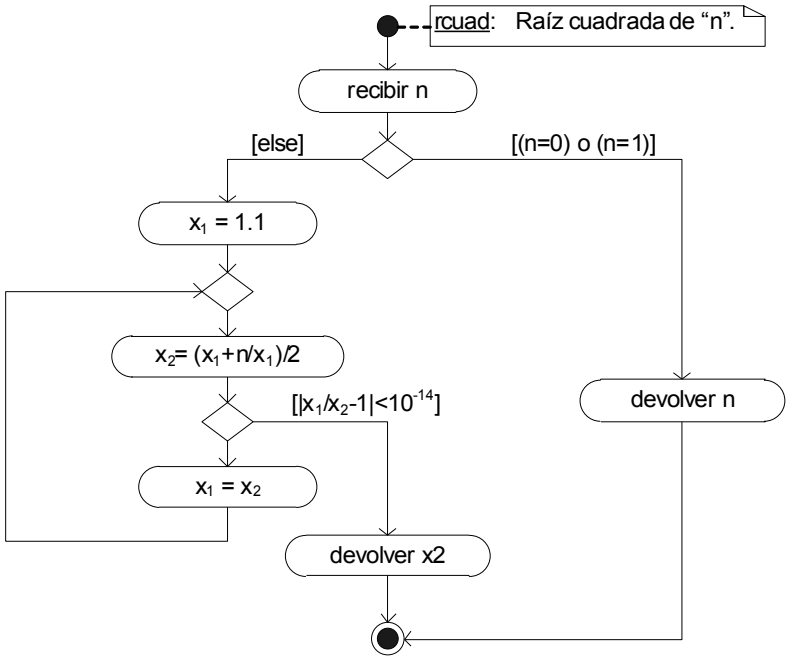
Ahora "x1" y "x2" son iguales (en los 9 dígitos), por lo que el proceso concluye, siendo la raíz cuadrada de 2: 1.41421356.

Para averiguar si dos valores "x_1" y "x_2" son iguales en por lo menos "d" dígitos se emplea la expresión:

$$\left|\frac{x_1}{x_2}-1\right|<10^{-d}$$

Que es verdadera cuando dichos valores son iguales en "d" o más dígitos. El valor inicial asumido "x_1" puede ser cualquiera, sin embargo, el número de repeticiones es menor cuanto más cercano se encuentra dicho valor de la solución. A pesar de ello y por simplicidad, en este ejemplo se empleará un valor sumido igual a 1.1, siendo la precisión de 14 dígitos.

El algoritmo en forma de diagrama de actividades es:



Como se puede ver, antes de entrar al ciclo, se descartan los casos 0 y 1, porque se sabe que la raíz cuadrada de 0 es 0 y la de 1 es 1, además la lógica de este problema se ajusta mejor a un ciclo infinito, pues el proceso concluye inmediatamente los valores asumido y calculado son iguales en el número de dígitos especificado (en este caso 14).

El código respectivo es:

```
C:\HPV\programas\c++\2-2012\tema6>notepad ejem2.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

double leerNumero(){
  double n;
  cout.width(20);
  cout<<endl<<"Numero"<<"? ";
  cin>>n;
  if (n<0){
    cout<<"Error: El numero debe ser positivo."<<endl;
    exit(EXIT_FAILURE);
```

```
  }
  return n;
}

double rcuad(double n){
  if (n==0 || n==1) return n;
  double x1=1.1,x2;
  while(true){
    x2=(x1+n/x1)/2;
    if (abs(x1/x2-1)<1e-14) return x2;
    x1=x2;
  }
}

void mostrarRaiz(double r){
  cout.width(20);
  cout.precision(14);
  cout<<"Raiz cuadrada"<<": "<<r<<endl;
}


int main(){
  double n,r;
  n=leerNumero();
  r=rcuad(n);
  mostrarRaiz(r);
  return EXIT_SUCCESS;
}
```

Observe que en este ejemplo se sale del ciclo infinito empleando el comando "return" (pues no existe ninguna otra instrucción después del ciclo). Se reitera que la solución presentada es sólo una de las posibles soluciones al problema.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPU\programas\c++\2-2012\tema6>g++ -c ejem2.cpp

C:\HPU\programas\c++\2-2012\tema6>g++ -o ejem2 ejem2.o

C:\HPU\programas\c++\2-2012\tema6>ejem2

               Numero? 2
       Raiz cuadrada: 1.4142135623731

C:\HPU\programas\c++\2-2012\tema6>ejem2

               Numero? 1
       Raiz cuadrada: 1

C:\HPU\programas\c++\2-2012\tema6>ejem2

               Numero? 0
       Raiz cuadrada: 0

C:\HPU\programas\c++\2-2012\tema6>ejem2

               Numero? -32
Error: El numero debe ser positivo.
```

6.3. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema6" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa que, empleando librerías tipo C++ y la estructura "while", reciba un número entero positivo o negativo y devuelva su número de dígitos.

2. Elabore un programa que, empleando librerías tipo C y la estructura "while", reciba un número entero positivo o negativo y devuelva su primer dígito.

3. Elabore u programa que, empleando librerías tipo C++ y la estructura "while", reciba un número entero positivo o negativo y devuelva dicho número sin su su primer dígito.

4. Elabore un programa que, empleando librerías tipo C y la estructura "while", calcule la raíz cúbica de un número real con la ecuación:

$$x_2 = \frac{1}{3} \cdot \left(2x_1 + \frac{n}{x_1^2} \right)$$

5. Elabore un programa que, empleando librerías tipo C++ y la estructura "while", encuentre una de las soluciones de la ecuación cúbica: $a \cdot x^3 + b \cdot x^2 + c \cdot x + d = 0$ con 12 dígitos de precisión. El módulo que resuelve el problema debe recibir los coeficientes de la ecuación ("a", "b", "c" y "d"), así como el valor inicial asumido "x1". Además, puesto que no siempre es posible encontrar una solución, se debe incluir un contador de repeticiones que comience en 1 y que al llegar a 30 salga del ciclo mostrando un mensaje de error (la lógica es similar a la de las raíces cuadrada y cúbica).

$$x_2 = \frac{2 \cdot a \cdot x_1^3 + b \cdot x_1^2 - d}{3 \cdot a \cdot x_1^2 + 2 \cdot b \cdot x_1 + c}$$

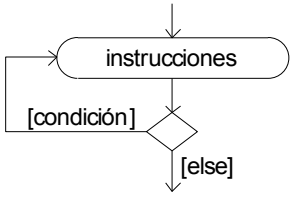
Ecuación de prueba: $x^3 + 2x^2 + 3x + 4 = 0$ ($sol = -1.65062919144$), valor asumido 1.1.

7. ITERACIÓN 2

En este tema continúa el repaso de las estructuras iterativas, con una variante de la estructura "while", la estructura "do-while".

7.1. LA ESTRUCTURA "DO-WHILE"

La lógica de la estructura "do-while" (hacer-mientras) es la siguiente:



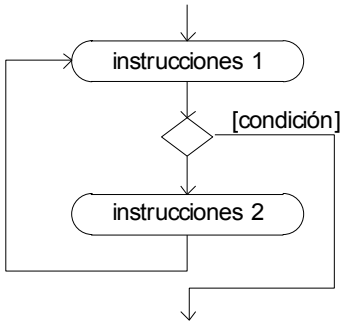
La lógica es esencialmente la misma que en la estructura "while", sólo que ahora la condición está al final del ciclo y no al principio y en consecuencia las instrucciones se repiten por lo menos una vez, mientras que en la estructura "while" las instrucciones pueden no repetirse ni una vez (dependiendo del valor inicial de la condición).

La forma de codificar esta instrucción es:

```
do {
  instrucciones;
} while (condición);
```

Al igual que con "while", si en lugar de varias "instrucciones" se tiene una sola, no es necesario encerrarla entre llaves (aunque el hacerlo tampoco constituye un error).

Con "do-while" (al igual que con "while") se pude crear también un ciclo infinito, para ello simplemente se coloca como condición el valor "true" (verdadero). Como ya se ha visto en el tema anterior, los ciclos infinitos permiten implementar lógicas como la siguiente:



Que se codifica con el comando "break", cuando existen instrucciones después del ciclo:

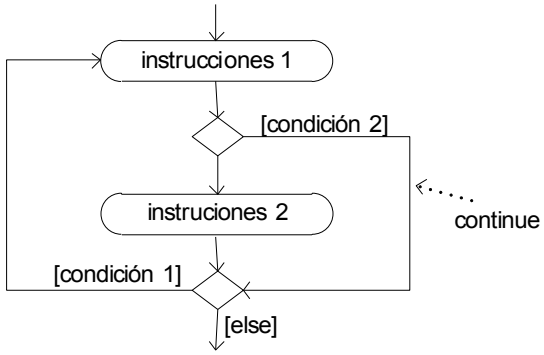
```
do {
  instrucciones 1;
  if (condición) break;
  instrucciones 2;
} while (true);
```

O con "return" cuando se devuelven directamente resultados:

```
do {
  instrucciones 1;
```

```
  if (condición) return resultado;
  instrucciones 2;
} while (true);
```

Por supuesto, para codificar lógicas como la siguiente:



Es decir, cuando se quiere saltar directamente al siguiente ciclo, se puede emplear el comando "continue":

```
do {
  instrucciones 1;
  if (condición 2) continue;
  instrucciones 2;
} while (condición 1);
```

Como el propósito del tema es el estudio de la estructura "do-while", en todos los ejemplos y ejercicios de este tema se emplea dicha estructura, sin embargo, en la práctica se debe emplear esta estructura cuando la lógica indica que las instrucciones deben repetirse por lo menos una vez, por el contrario se debe emplear la estructura "while" cuando la lógica implica que las instrucciones pueden no repetirse ni una vez.

En los ciclos infinitos es indistinto emplear cualquier estructura iterativa.

7.2. EJEMPLOS

1. Elabore un programa que, empleando librerías tipo C y la estructura "do-while", calcule la raíz cuadrada de un número real.

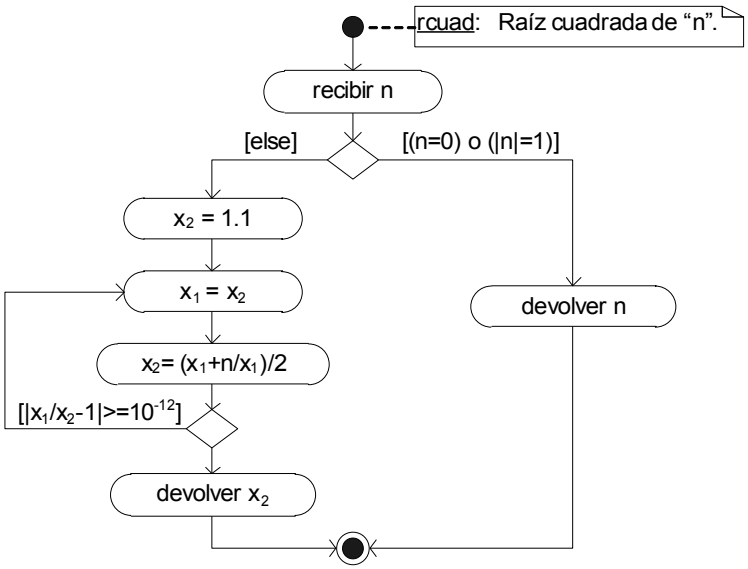
Como ya se vio en el tema anterior, la raíz cuadrada de un número "n" puede ser calculada con la ecuación:

$$x_2 = \frac{1}{2}\left(x_1 + \frac{n}{x_1}\right)$$

La solución propuesta en dicho tema (la que emplea un ciclo infinito) puede ser codificada igualmente (sin ninguna modificación) con la estructura "do-while", sin embargo, simplemente para mostrar otra de las muchas posibles soluciones y en la cual además se emplee propiamente la estructura "do-while" (no como un ciclo infinito) en este ejemplo se resuelve el problema siguiendo la lógica que se muestra en el diagrama de actividades de la siguiente página:

Observe que en este algoritmo el valor inicial (1.1) se asigna a la variable "x2" (no a "x1"). Se procede así porque la primera instrucción dentro del ciclo asigna el valor de "x2" a "x1", por lo que, al ejecutarse el primer ciclo "x1" toma efectivamente el valor inicial asumido (1.1).

El código elaborado, de acuerdo a este algoritmo, es:




```
c:\HPV\programas\c++\2-2012\tema7>notepad ejem1.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

double leerNumero(){
  double n;
  printf("\n%20s? ","Numero");
  scanf("%lf",&n);
  if (n<0){
    printf("Error: El numero debe ser positivo.\n");
    exit(EXIT_FAILURE);
  }
  return n;
}

double rcuad(double n){
  if (n==0 || n==1) return n;
  double x1,x2=1.1;
  do{
    x1=x2;
    x2=(x1+n/x1)/2;
  }while(fabs(x1/x2-1)>=1e-12);
  return x2;
}

void mostrarRaiz(double r){
  printf("%20s: %.12g\n","La raiz cuadrada es",r);
}

int main(){
  double n,r;
  n=leerNumero();
  r=rcuad(n);
  mostrarRaiz(r);
  return EXIT_SUCCESS;
```

```
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
c:\HPV\programas\c++\2-2012\tema7>g++ -c ejem1.cpp

c:\HPV\programas\c++\2-2012\tema7>g++ -o ejem1 ejem1.o

c:\HPV\programas\c++\2-2012\tema7>ejem1

              Numero? 2
 La raiz cuadrada es: 1.41421356237

c:\HPV\programas\c++\2-2012\tema7>ejem1

              Numero? 64
 La raiz cuadrada es: 8

c:\HPV\programas\c++\2-2012\tema7>ejem1

              Numero? -32
Error: El numero debe ser positivo.
```

2. Elabore un programa que, empleando librerías tipo C++ y la estructura "do-while", calcule el exponente de un número real "x" aplicando la serie de Taylor:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \infty$$

Como esta serie va hasta el infinito no es posible en la práctica calcular la sumatoria (pues se requeriría un tiempo infinito), entonces, lo que se hace es ir sumando los términos de la serie hasta que los resultados de dos sumas consecutivas sean aproximadamente iguales, es decir hasta que el valor de la suma se mantenga prácticamente constante.

Los términos de una serie se calculan en base al término anterior mediante la razón (o regla) que rige la serie. Por ejemplo, en esta serie, el nuevo término se calcula multiplicando el término anterior por "x" y dividiendo entre el número de término respectivo (siendo el primer término el número 0).

Así para calcular el cuarto término (en base al tercero) las operaciones necesarias son:

$$\frac{x^3}{3!} \cdot \frac{x}{4} = \frac{x^4}{4!}$$

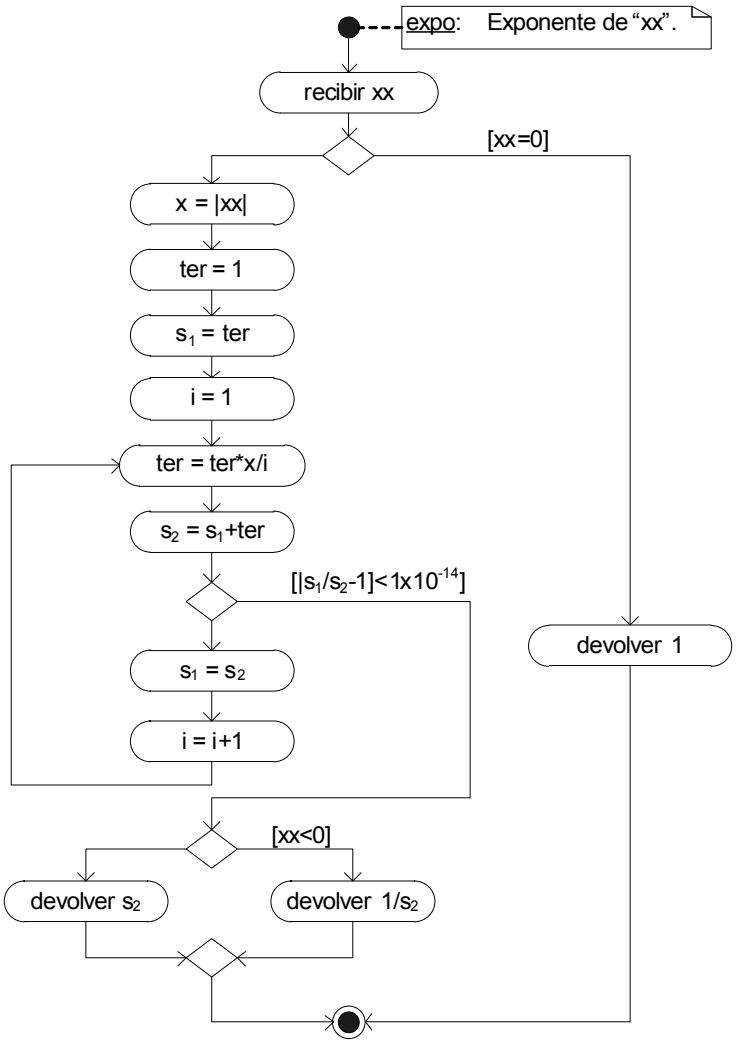
Por lo tanto la razón a emplear en esta serie es:

$$\frac{x}{n}$$

Donde "n" es el número de término que se quiere calcular.

Casi siempre lo más difícil al resolver este tipo de problemas es el deducir la razón o regla que rige la misma. Luego prácticamente todas las series siguen la misma lógica: a) se inicializan variables; b) se calcula el nuevo término (aplicando la razón o regla); c) se compara el nuevo término con el anterior: si son aproximadamente iguales el proceso concluye, siendo la solución la última sumatoria; d) se actualizan e intercambian variables y se repite el proceso desde el paso "b".

El algoritmo en forma de diagrama de actividades es:



Como se puede ver, además de los pasos antes indicados, en este problema se verifica primero si el número es cero, porque el exponente de "0" es "1". Además, para evitar restas, se trabaja con el valor absoluto del número, **Las restas de números grandes generan grandes errores de redondeo por lo que siempre que sea posible deben ser evitadas**.

En este ejemplo es posible evitar las restas porque el resultado negativo puede ser calculado en base al resultado positivo aplicando la relación matemática:

$$e^{-x}=\frac{1}{e^{x}}$$

Es por eso que antes de devolver el resultado se pregunta si el número era negativo y de ser así, se devuelve la inversa del valor calculado.

Es necesario aclarar que no siempre es posible evitar el trabajar con valores negativos, en este ejemplo ha sido posible gracias a que existe la relación matemática apropiada. Si no se puede trabajar con valores positivos se debe procurar trabajar con valores pequeños.

Con las anteriores consideraciones, el programa es:

```
c:\HPV\programas\c++\2-2012\tema7>notepad ejem2.cpp
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

double leerNumero(){
  double x;
  cout.width(20);
  cout<<endl<<"Numero"<<"? ";
  cin>>x;
  return x;
}

double expo(double xx){
  if (xx==0) return 1;
  double s1,s2,ter,x;
  int i;
  x=abs(xx);
  ter=1;
  s1=ter;
  i=1;
  do{
    ter*=x/i;
    s2=s1+ter;
    if (abs(s1/s2-1)<1e-14) break;
    s1=s2;
    i++;
  } while(true);
  return xx>0 ? s1 : 1/s2;
}

void mostrarResultado(double r){
  cout.width(20);
  cout.precision(14);
  cout<<"Exponente"<<": "<<r<<endl;
}

int main(){
  double x,r;
  x=leerNumero();
  r=expo(x);
  mostrarResultado(r);
  return EXIT_SUCCESS;
}
```

Observe que en este caso se sale del ciclo infinito empleando el comando "break", porque existe una instrucción después del ciclo. Note también que, al igual que en el anterior ejemplo, los resultados son mostrados con 14 dígitos de precisión, pues esa es la precisión con la que se calcula el resultado.

Compilando y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
c:\HPV\programas\c++\2-2012\tema7>g++ -c ejem2.cpp

c:\HPV\programas\c++\2-2012\tema7>g++ -o ejem2 ejem2.o
```

```
c:\HPU\programas\c++\2-2012\tema7>ejem2

            Numero? 2.34
         Exponente: 10.381236562732

c:\HPU\programas\c++\2-2012\tema7>ejem2

            Numero? -21.45
         Exponente: 4.8348539902099e-10
```

3. Elabore un programa que, empleando librerías tipo C y la estructura "do-while", calcule el seno de un ángulo en radianes aplicando la serie de taylor:

$$\text{sen}(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots\infty$$

En este caso el nuevo término se calcula multiplicando el anterior por $-x^2$ y dividiendo entre un contador y su valor siguiente. Dicho contador comienza en 0 e incrementa de 2 en 2, por ejemplo para calcular el cuarto término a partir del tercero las operaciones son:

$$\frac{x^5}{5!}\cdot\frac{-x^2}{6\cdot 7}=-\frac{x^7}{7!}$$

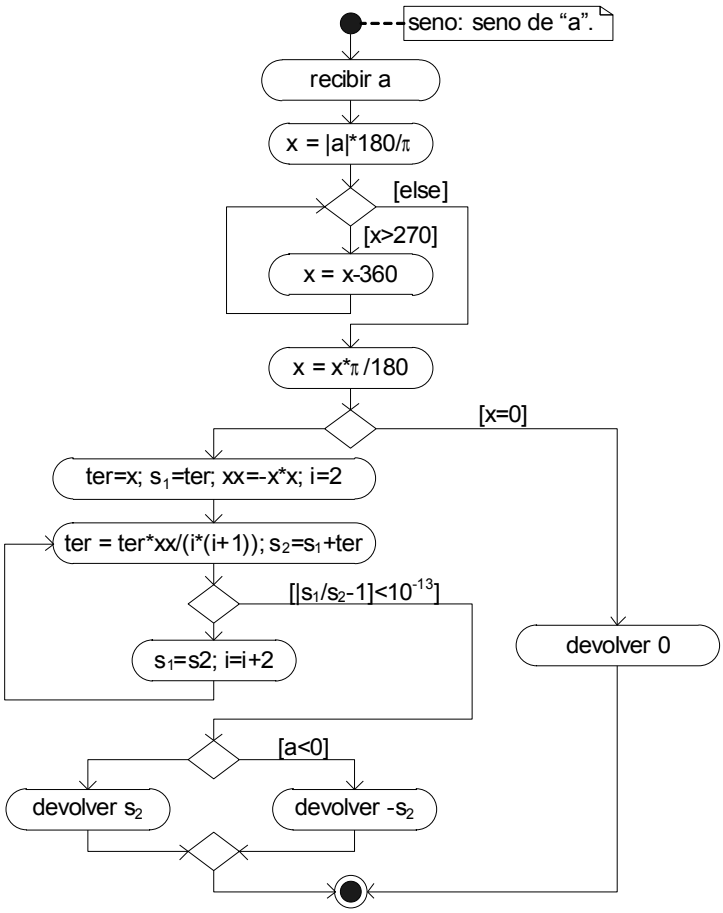
En este problema las restas están en la serie, por lo que no pueden ser evitadas, como ya se dijo, cuando eso sucede se debe buscar la forma trabajar con números pequeños. En el seno (y otras funciones trigonométricas) es posible evitar números grandes, pues se sabe que todo ángulo superior a 3*π/2 puede ser reducido a su equivalente entre -3*π/2 y 3*π/2. Para ello simplemente se resta al ángulo 2*π hasta que su valor sea menor o igual a 3*π/2.

Para disminuir los errores de redondeo, la reducción del ángulo se la hace en grados, además, como se cumple que sin(-α)=-sin(α), se puede trabajar con el valor absoluto del ángulo y cambiar el mismo al momento de devolver el resultado.

El algoritmo, en forma de diagrama de actividades, se presenta en la siguiente página y el código respectivo es:

```
c:\HPU\programas\c++\2-2012\tema7>notepad ejem3.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

double leerAngulo(){
  double x;
  printf("\n%20s? ","Angulo");
  scanf("%lf",&x);
  return x;
}

double seno(double a){
  double x,s1,s2,ter,xx;
  int i;
  x=fabs(a)*180/M_PI;
  while(x>270) x-=360;
  x*=M_PI/180;
  if(x==0) return 0;
```




```
  ter=x;
  s1=ter;
  xx=-x*x;
  i=2;
  do{
    ter*=xx/(i*(i+1));
    s2=s1+ter;
    if (fabs(s1/s2-1)<1e-13) break;
    s1=s2;
    i+=2;
  }while(true);
  return a<0 ? -s2 : s2;
}

void mostrarSeno(double s){
  printf("%20s: %.13g\n","Seno",s);
}

int main(){
  double a,s;
  a=leerAngulo();
  s=seno(a);
  mostrarSeno(s);
  return EXIT_SUCCESS;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algu-

nos valores de prueba, se obtiene:

```
c:\HPU\programas\c++\2-2012\tema7>g++ -c ejem3.cpp

c:\HPU\programas\c++\2-2012\tema7>g++ -o ejem3 ejem3.o

c:\HPU\programas\c++\2-2012\tema7>ejem3

              Angulo? 5.67
                Seno: -0.5754753801952

c:\HPU\programas\c++\2-2012\tema7>ejem3

              Angulo? 123.45
                Seno: -0.8003546353267

c:\HPU\programas\c++\2-2012\tema7>ejem3

              Angulo? -345.3
                Seno: 0.2717316164205
```

7.3. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema7" (tanto para los ejemplos como para los ejercicios).

1. Elabore un programa que, empleando librerías tipo C++ y la estructura "do-while", calcule la raíz cúbica de un número real con la ecuación:

$$x_2 = \frac{1}{3} \cdot \left(2x_1 + \frac{n}{x_1^2} \right)$$

2. Elabore un programa que, empleando librerías C y la estructura "do-while", calcule el seno hiperbólico de un número, con 11 dígitos de precisión, aplicando la serie de Taylor:

$$\operatorname{senh}(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots + \infty$$

3. Elabore un programa que, empleando librerías C++ y la estructura "do-while", calcule el coseno de un ángulo en radianes, con 12 dígitos de precisión, aplicando la serie de Taylor:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \infty$$

4. Elabore un programa que, empleando librerías C y la estructura "do-while", calcule el arco tangente hiperbólico de un número, con 14 dígitos de precisión, aplicando la serie de Taylor:

$$\tanh^{-1}(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \ldots \infty \; ; \qquad |x| < 1$$

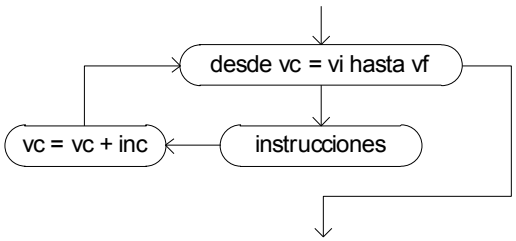
8. ITERACIÓN 3

En este tema se estudia otra estructura iterativa, la estructura "for".

8.1. LA ESTRUCTURA "FOR"

La estructura "for" es la estructura iterativa menos estándar de todas. Su implementación suele diferir entre los diferentes lenguajes.

En general, sin embargo, la lógica de la estructura "for" es la que se muestra en la figura:



En esta estructura, las "instrucciones" se repiten "desde" que una variable de control (o contador) "vc" toma un valor inicial "vi" hasta que llega a un valor final "vf", incrementando en cada repetición del ciclo el valor de la variable de control en "inc".

En C++ la estructura "for" tiene la siguiente forma:

```
for (inicialización; condición; incremento) {
  instrucciones;
}
```

En realidad, la estructura "for" en C/C++ tiene una lógica más parecida a la estructura "while" que a una estructura "for" propiamente: Las "instrucciones" se repiten mientras la "condición" es verdadera, sólo que con "for", antes comenzar el ciclo se ejecutan las instrucciones escritas en el sector de "inicialización", además, antes de cada repetición se ejecutan las instrucciones del sector de "incremento".

En consecuencia en C/C++, la estructura "for" es más bien una estructura "while" a la cual se le ha añadido un sector de inicialización (que se ejecuta una sola vez, antes de comenzar el ciclo) y un sector de incremento (que se ejecuta cada vez, antes de repetir el ciclo).

Para implementar con esta estructura la lógica "for" tradicional (la lógica mostrada en el diagrama), se escriben:

```
for (vc=vi; vc<=vf; vc=vc+inc) {
  instrucciones;
}
```

Como de costumbre, si la instrucción es una sola, no es necesario emplear las llaves, aunque el emplearlas no constituye un error.

Par implementar un ciclo infinito con esta estructura no es necesario escribir (como podría pensarse) la condición verdadera (true), sino simplemente dejar en blanco el contenido entre paréntesis:

```
for (;;) instrucción;
```

Como se recordará, en C/C++, se puede emplear la coma como operador de secuencia, ello permite que cada uno de los bloques de esta estructura se pueda escribir más de una instrucción, lo que hace de esta estructura una

de las más flexibles del lenguaje.

En general, no obstante, se recomienda emplearla cuando la lógica que resuelve el problema corresponde a la lógica tradicional de la estructura "for", es decir cuando se sabe el número de veces que se debe repetir el proceso.

8.2. IDENTACIÓN DEL CÓDIGO

A la práctica de escribir el código que pertenece a una estructura alineado unos espacios más a la derecha se conoce como identación. **A partir del presente tema, todo el código que se elabore deberá estar identado, de lo contrario será considerado incorrecto**.

El que el código esté o no identado no afecta ni al rendimiento ni a la eficiencia de un programa. Para el compilador es lo mismo que el programa esté en una sola línea y sólo con los espacios imprescindibles o que esté en varias líneas con todos los espacios extras que se quiera, pero para las personas, que son quienes escriben los programas, es mucho más fácil comprender la lógica de un programa y detectar (y/o evitar errores) si el código está escrito de forma ordenada.

Por ejemplo el siguiente programa, que muestra el mes equivalente a un número, está identado:

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

int main() {
  int n;
  cout<<"introduzca un número entero: ";
  cin>>n;
  cout<<"El mes equivalente es ";
  switch (n) {
    case 1: cout<<"Enero\n"<<endl; break;
    case 2: cout<<"Febrero\n"<<endl; break;
    case 3: cout<<"Marzo\n"<<endl; break;
    case 4: cout<<"Abril\n"<<endl; break;
    case 5: cout<<"Mayo\n"<<endl; break;
    case 6: cout<<"Junio\n"<<endl; break;
    case 7: cout<<"Julio\n"<<endl; break;
    case 8: cout<<"Agosto\n"<<endl; break;
    case 9: cout<<"Septiembre\n"<<endl; break;
    case 10: cout<<"Octubre\n"<<endl; break;
    case 11: cout<<"Noviembre\n"<<endl; break;
    case 12: cout<<"Diciembre\n"<<endl; break;
    default: cout<<"No es equivalente a un mes\n"<<endl;
  }
  return 0;
}
```

Como se puede ver, el código que pertenece a la función "main" está más a la derecha, igualmente el código que pertenece a "switch" está más a la derecha, de esa manera se pueden identificar fácilmente las diferentes estructuras, así como identificar errores como llaves no cerradas y/o estructuras superpuestas.

Sin embargo, **el identar no consiste, como algunos creen, en escribir cada nueva instrucción más y más a la derecha**, sino en escribir de 2 a 4 columnas más a la derecha sólo el código que pertenece a una estructura. Todo el código que se encuentra dentro de una estructura (y que no pertenece a

otra) debe ser escrito al mismo nivel (tal como se muestra en este y en todos los ejemplos que se han escrito desde el primer tema).

Desde el punto de vista del compilador, el anterior código y el siguiente son iguales:

```
#include <iostream>
using namespace std;int main(){int n;cout<<"introduzca un número entero: "
cin>>n;cout<<"El mes equivalente es ";switch(n){case 1:cout<<"Enero\n"<<
endl;break;case 2:cout<<"Febrero\n"<<endl;break;case 3:cout<<"Marzo\n"<<
endl;break;case 4:cout<<"Abril\n"<<endl;break;case 5:cout<<"Mayo\n"<<endl;
break;case 6:cout<<"Junio\n"<<endl; break;case 7:cout<<"Julio\n"<<endl;
break;case 8:cout<<"Agosto\n"<<endl; break;case 9:cout<<"Septiembre\n"<<
endl;break;case 10:cout<<"Octubre\n"<<endl;break;case 11:cout<<
"Noviembre\n"<<endl; break;case 12:cout<<"Diciembre\n"<<endl;break;
default:cout<<"No es equivalente a un mes\n"<<endl;}return 0;}
```

Pero es claro que desde el punto de vista humano no, este código (a pesar de que está resaltado) resulta mucho más difícil de comprender que el anterior.

8.3. EJEMPLOS

1. Elabore un programa que, empleando librerías tipo C++ y la estructura "for", calcule el factorial de un número entero.

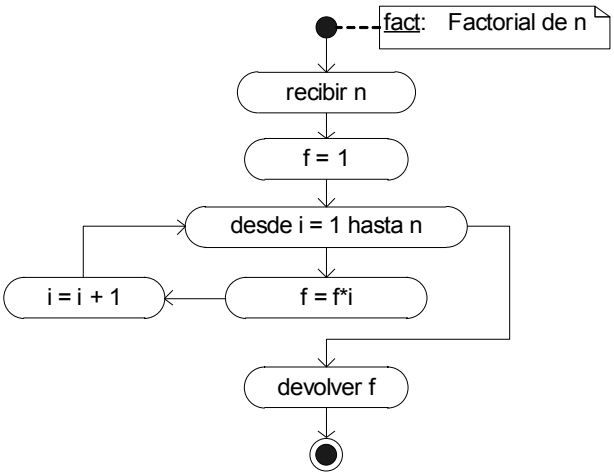
El factorial de un número entero "n" es el resultado de multiplicar los números enteros existentes entre 1 y dicho número, es decir:

$$n!=\prod_{i=1}^{n} i=1*2*3*4*\cdots*n;\quad 0!=1$$

Cuando se tienen productorias $\left(\prod\right)$ o sumatorias $\left(\sum\right)$ con límites definidos (o cuando se conoce el número de veces que debe repetirse el proceso) la estructura más adecuada para resolver el problema es "for".

En la mayoría de los casos la productoria (o sumatoria) se traduce directamente a un ciclo "for". Por ejemplo, para el factorial, la productoria, donde el contador "i" va desde 1 hasta "n", se traduce en un ciclo "for" donde un contador "i" va desde 1 hasta "n" y al ser una productoria, en cada repetición del ciclo se multiplica por "i" la productoria anterior.

El algoritmo que resuelve el problema, siguiendo el anterior razonamiento es el siguiente:



Como es una productoria, la variable "f", que es donde se almacena el resultado, comienza en 1.

En la estructura "for" si el valor inicial es de entrada superior al límite, las instrucciones del ciclo no se ejecutan ni una vez y el programa pasa directamente a la instrucción que se encuentra después del ciclo. Por lo tanto, en este caso, el ciclo puede comenzar directamente en 2, entonces cuando "n" sea 0 o 1, el ciclo no se ejecutará y se devolverá directamente el valor de "f", que como es 1, corresponde al resultado correcto (pñues el factorial de 0 y 1 es 1).

El programa, tomando en cuenta las anteriores consideraciones, es:

```
C:\HPV\programas\c++\2-2012\tema8>notepad ejem1.cpp
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
using namespace std;

unsigned leerNumero(){
  double n;
  do{
    cout<<setw(20)<<"Numero"<<"? ";
    cin>>n;
    if (fmod(n,1)) {
      cout<<setw(20)<<"Error"<<": El numero debe ser entero."<<endl;
      continue;
    }
    if (n<0) {
      cout<<setw(20)<<"Error"<<": El numero debe ser >=0."<<endl;
      continue;
    }
    break;
  }while(true);
  return (unsigned)n;
}

double factorial(unsigned n){
  double f=1;
  for (unsigned i=2;i<=n;i++) f*=i;
  return f;
}

void mostrarFactorial(double r){
  cout<<setw(20)<<"El factorial es"<<": "<<setprecision(12)<<r<<endl;
}

int main(){
  unsigned n;
  double r;
  n=leerNumero();
  r=factorial(n);
  mostrarFactorial(r);
  return EXIT_SUCCESS;
}
```

Hay algunos aspectos que se deben comentar con relación al programa.

En primer lugar, se hace notar que la función "factorial" devuelve un resultado de tipo real (double) a pesar de que el factorial de un número entero es otro entero. Se procede así (aparentemente violando el tercer principio de la programación estructurada) porque los valores de los factoriales son números muy grandes y el entero más grande disponible en C++ (unsigned long int) tiene apenas 10 dígitos, de manera que si se empleara este tipo la función sólo podría calcular como máximo el factorial de 12.

En realidad los valores son tan grandes que sería conveniente emplear inclusive el tipo "long double", lamentablemente en C/C++ este tipo no es estándar y en el caso del compilador MinGW, no está aún correctamente implementado.

En la validación del dato (en la función "leerNumero") se ha empleado un ciclo infinito (implementado con la estructura "do-while"). En este ciclo se lee el número hasta que sea un valor válido, es decir hasta que sea un entero positivo. Con ese fin se pregunta si el número es real (si "fmod(n,1)" es diferente de 0) y de ser así se muestra un mensaje de error y se pasa al siguiente ciclo (donde se vuelve a pedir el número) con el comando "continue". Lo mismo ocurre si el número es entero pero negativo. Como se puede ver, para salir del ciclo infinito (cuando el número es entero y positivo) se emplea el comando "break".

Por otra parte, para pedir y mostrar los resultados con formato, se han empleado las instrucciones "setw" y "setprecision", que son los equivalentes a las intrucciones "cout.width" y "cout.precision", sólo que son más cortas y pueden ser empleadas directamente con la instrucción "cout". Estas instrucciones se conocen como "manipuladores" y para poder emplearlas se debe incluir la librería "iomanip" (tal como se ha hecho en el ejemplo).

Compilando y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\2-2012\tema8>g++ -c ejem1.cpp

C:\HPV\programas\c++\2-2012\tema8>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\2-2012\tema8>ejem1
          Numero? 5
     El factorial es: 120

C:\HPV\programas\c++\2-2012\tema8>ejem1
          Numero? 7
     El factorial es: 5040

C:\HPV\programas\c++\2-2012\tema8>ejem1
          Numero? 12
     El factorial es: 479001600

C:\HPV\programas\c++\2-2012\tema8>ejem1
          Numero? 30
     El factorial es: 2.65252859812e+32

C:\HPV\programas\c++\2-2012\tema8>ejem1
          Numero? 120
     El factorial es: 6.68950291345e+198
```

Observe que el factorial de 120 tiene 198 dígitos (muy por encima de los 10 dígitos permitidos por el número entero más grande disponible).

2. Elabore un programa que, empleando librerías tipo C y la estructura "for", calcule el Chebyshev enésimo de un número real "x".

La ecuación de definición del Chebyshev es:

$$Ch_{n-1}(x)=2\cdot x\cdot Ch_n(x)-Ch_{n-1}(x)$$
$$Ch_0(x)=1$$
$$Ch_1(x)=x$$

Como se puede ver, por definición, el Chebyshev 0 de "x" es 1 y el Chebyshev 1 de "x" es "x". Comenzando con estos valores se puede calcular el Chebyshev 2 de "x", luego con este valor y el anterior se calcula el Chebyshev de 3 de "x" y así sucesivamente hasta llegar al Chebyshev enésimo (n) de "x".

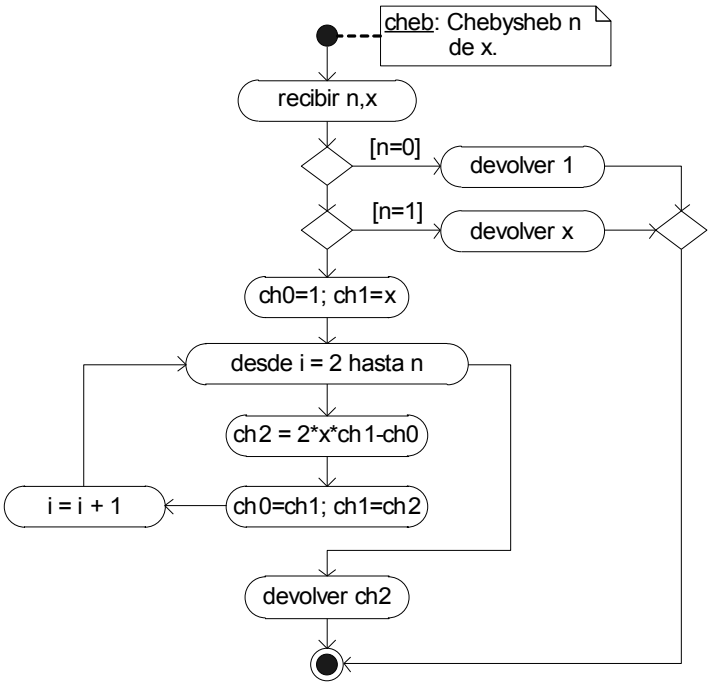
Por ejemplo para calcular el Chebyshev 7 (n=7) de "x" las operaciones a realizar son:

$$Ch_2(x)=2\cdot x\cdot Ch_1(x)-Ch_0(x)$$
$$Ch_3(x)=2\cdot x\cdot Ch_2(x)-Ch_1(x)$$
$$Ch_4(x)=2\cdot x\cdot Ch_3(x)-Ch_2(x)$$
$$Ch_5(x)=2\cdot x\cdot Ch_4(x)-Ch_3(x)$$
$$Ch_6(x)=2\cdot x\cdot Ch_5(x)-Ch_4(x)$$
$$Ch_7(x)=2\cdot x\cdot Ch_6(x)-Ch_5(x)$$

Lo que implica el cálculo de los Chebyshev desde 2 hasta 7. De aquí, se deduce que para calcular el Chebyshev "n" de "x" el proceso se repite desde 2 hasta "n", por lo tanto, como se tiene un número definido de iteraciones, la estructura más adecuada para resolver el problema es la estructura "for".

Como en este problema sólo interesa el valor del Chebyshev enésimo (n) de "x", no es necesario guardar los valores intermedios, por lo que sólo se guardan los dos últimos (para calcular con ellos el siguiente).

El algoritmo que resuelve el problema es el siguiente:



El programa elaborado, siguiendo esta lógica, es:

```
C:\HPU\programas\c++\2-2012\tema8>notepad ejem2.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>

unsigned leerOrden(){
  double n;
  while(true){
    printf("%20s? ","Orden");
    scanf("%lf",&n);
    if (fmod(n,1)) {
      printf("%20s: %s\n","Error","El orden debe ser entero.");
      continue;
    }
    if (n<0) {
      printf("%20s: %s\n","Error","El orden debe ser positivo.");
      continue;
    }
    break;
  }
  return (unsigned)n;
}

double leerNumero(){
  double x;
  printf("%20s? ","Numero");
  scanf("%lf",&x);
  return x;
}

double chebyshev(unsigned n, double x){
  if (n==0) return 1;
  if (n==1) return x;
  double ch0=1,ch1=x,ch2;
  for(unsigned i=2;i<=n;i++){
    ch2=2*x*ch1-ch0;
    ch0=ch1;
    ch1=ch2;
  }
  return ch2;
}

void mostrarChebyshev(double r){
  printf("%20s: %.12g\n","El Chebyshev es",r);
}

int main(){
  unsigned n;
  double x,r;
  n=leerOrden();
  x=leerNumero();
  r=chebyshev(n,x);
  mostrarChebyshev(r);
  return EXIT_SUCCESS;
}
```

Observe que (como ocurrió en el anterior ejemplo) el contador de la estructura "for" (la variable "i") se declara directamente dentro del ciclo (de tipo "unsigned"). Esta es una práctica frecuente cuando se programa en C++, sin embargo, es posible (y en ocasiones recomendable) declarar el contador antes de emplearlo en la estructura.

En general en C++ se pueden declarar las variables en el lugar donde se utilizan y cuando se procede así, dichas variables tienen vigencia (existen) únicamente en el bloque donde han sido declarados. Así, cuando se declara el contador dentro de un ciclo "for", dicho contador sólo existe mientras se ejecuta ese ciclo, por lo tanto no puede ser empleado fuera del mismo.

Compilando y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
C:\HPV\programas\c++\2-2012\tema8>g++ -c ejem2.cpp

C:\HPV\programas\c++\2-2012\tema8>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 5
              Numero? 1.2
     El Chebyshev es: 11.25312

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 6
              Numero? 3.5
     El Chebyshev es: 51841

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 2
              Numero? 7.1
     El Chebyshev es: 99.82

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 12
              Numero? 0.3
     El Chebyshev es: -0.870430994432

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 0
              Numero? 1
     El Chebyshev es: 1

C:\HPV\programas\c++\2-2012\tema8>ejem2
               Orden? 4.5
               Error: El orden debe ser entero.
               Orden? -6
               Error: El orden debe ser positivo.
               Orden? 2
              Numero? 1.1
     El Chebyshev es: 1.42
```

8.4. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema8" (tanto para los ejemplos como para los ejercicios). No olvide que a partir de este tema los programas deben estar identados.

1. Elabore un programa que, empleando librerías tipo C y la estructura "for" calcule la sumatoria de los primeros "n" números enteros positivos.

$$s=\sum_{i=1}^{n} i$$

2. Elabore un programa que, empleando librerías tipo C++ y la estructura "for", calcule al productoria de los primeros "n" números impares:

$$s=\prod_{i=1,3,5}^{2*n-1} i$$

3. Elabore un programa que, empleando librerías tipo C y la estructura "for", lea el número de elementos "n" y calcule el valor de la siguiente sumatoria:

$$p=\sum_{x=1,3,5}^{n}\sum_{y=2,4,6}^{n}\sqrt{x^2+y^2}$$

4. Elabore un programa que, empleando librerías tipo C++ y la estructura "for", calcule el fibonaci enésimo de un número entero positivo "n":

$$F_n=F_{n-1}+F_{n-2}$$
$$F_1=F_2=1$$

5. Elabore un programa que, empleando libreías tipo C y la estructura "for", calcule el Legendre enésimo (n) de un número real (x):

$$\mathrm{Le}_n(x)=\left(2-\frac{1}{n}\right)\cdot x\cdot \mathrm{Le}_{n-1}(x)-\left(1-\frac{1}{n}\right)\cdot \mathrm{Le}_{n-2}(x)$$
$$\mathrm{Le}_0(x)=1$$
$$\mathrm{Le}_1(x)=x$$

9. RECURSIVIDAD 1

En este tema se estudia otra forma de resolver problemas repetitivos: la recursividad. En general, un concepto es recursivo cuando en su definición se emplea el concepto que se está definiendo. Desde el punto de vista de la programación, una función es recursiva cuando se llama a sí misma.

La recursividad no es imprescindible en la solución de problemas iterativos, sin embargo, en ocasiones los problemas pueden ser resueltos de manera más clara y sencilla empleando este método.

Puesto que una solución más sencilla es también más comprensible, el empleo de estos métodos puede facilitar el mantenimiento y actualización de los programas, que es justamente el propósito de la programación estructurada.

Es importante comprender que la recursividad se trata sobre todo de otra forma de pensar, otra forma de abordar los problemas repetitivos. Mientras que con las estructuras estándar pensamos en cuantas veces o hasta cuando se debe repetir un determinado proceso, en la recursividad se piensa en la forma de encontrar la solución aprovechando una solución o valor conocido.

Algo que es muy importante al momento de plantear una solución recursiva es el no considerar como parte del problema el proceso recursivo en sí, es decir que no se debe considerar como parte del problema el cómo la función se llama a si misma. **En el planteamiento recursivo sólo interesa que efectivamente logre el resultado buscado, no como lo logra**.

En toda solución recursiva es importante que exista una condición de finalización pues de lo contrario el planteamiento recursivo resultaría en un proceso infinito: la función se llamaría a sí misma indefinidamente o hasta que se consuma toda la memoria disponible.

Algo que también se debe tener en mente es que las soluciones recursivas son menos eficientes que las iterativas: consumen más memoria y requieren más tiempo. Esto porque cada vez que una función se llama a sí misma crea una copia de las variables y parámetros de la función, lo que consume memoria y requiere tiempo, pero además cuando devuelven el resultado (cuando terminan) se destruyen las variables y parámetros de la función, lo que también consume tiempo.

9.1. EJEMPLOS

1. Elabore un programa que, empleando la recursividad y librerías tipo C, calcule el factorial de un número entero.

Este es el ejemplo clásico de recursividad. Desde el punto de vista iterativo la recursividad se define (como ya se vio en el anterior tema) como una productoria:

$$n! = \prod_{i=1}^{n} i$$

Que se resuelve con ciclo "for" que va desde 1 hasta "n". Desde el punto de vista recursivo, el factorial se define como:

$$n! = (n-1)! \cdot n$$

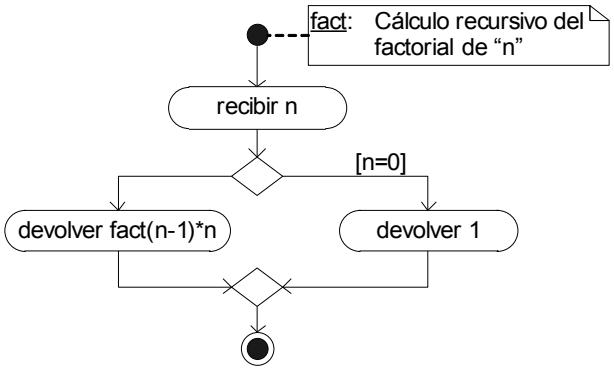
Es decir que el factorial de un número es igual al factorial del número anterior multiplicado por el número. Por ejemplo el factorial de 5 es:

$$5! = 4! \cdot 5 = 24 \cdot 5 = 120$$

Qué es el resultado correcto, por lo tanto el planteamiento recursivo es correcto. Como ya se dijo **en el planteamiento recursivo sólo interesa que se logre el resultado buscado,** en este caso por ejemplo, no es parte del problema el saber cómo se calcula el factorial de 4, sólo interesa verificar que al multiplicar el factorial de 4 por 5 se obtiene el factorial de 5.

Como también se dijo, **en toda solución recursiva se debe contar además con una condición de finalización,** pues de lo contrario el proceso es infinito. En este caso se sabe (por definición) que el factorial de 0 es 1, por lo tanto esa es la condición de finalización natural del problema, es decir que el proceso debe concluir cuando "n" sea 0 devolviendo el resultado 1.

El algoritmo para calcular el factorial, de forma recursiva, es:



Sólo con el propósito de ilustrar lo que ocurre en la recursividad, y comprender así mejor en qué consiste, se muestra a continuación los pasos que sigue el proceso recursivo para calcular el factorial de 5.

En la primera llamada a *fact* el parámetro *n* toma el valor 5 y como no es igual a 0, se ejecuta la instrucción:

*fact(5-1)*5 = fact(4)*5*

Ahora, para calcular el factorial de 4 (fact(4)) la función se llama a sí misma y en esta llamada (n=4), se ejecuta la instrucción:

*fact(4-1)*4 = fact(3)*4*

Una vez más, para calcular el factorial de 3, la función se llama a sí misma y sigue así hasta que "n" es igual a 0:

*fact(3-1)*3 = fact(2)*3*

*fact(2-1)*2 = fact(1)*2*

*fact(1-1)*1 = fact(0)*1*

Ahora la función se llama a sí misma con "n" igual a 0 y como se cumple la condición (n es igual a 0) la función devuelve el resultado 1. Este resultado es devuelto a la función que hizo la llamada (la copia de la función para "n" igual a 1) y con este resultado se puede calcular ahora el factorial de 1:

*fact(0)*1 = 1*1 = 1*

Este resultado (1) es devuelto a la función que hizo la llamada (la copia de la función para "n" igual a 2), igual que el caso anterior, con este resultado se puede calcular ahora el factorial de 2:

*fact(1)*2 = 1*2 = 2*

Y se prosigue así hasta llegar a la primera llamada recursiva:

*fact(2)*3 = 2*3 = 6*
*fact(3)*4 = 6*4 = 24*
*fact(4)*5 = 24*5 = 120*

Siendo este el resultado devuelto por la función recursiva. **Como se puede ver el proceso recursivo en si es moroso y hasta confuso, pero como se dijo, dicho proceso no es parte del problema,** del mismo se encarga automáticamente el lenguaje de programación.

Observe también que en el algoritmo se ha resuelto exclusivamente el problema sin tomar en cuenta casos especiales o excepcionales. Esto es algo que siempre se debe hacer al resolver un problema de manera recursiva: **la función recursiva sólo debe resolver el problema en sí, los casos especiales y/o excepcionales deben ser tratados en otra función no recursiva**. Si los casos especiales no se trataran en una función no recursiva, al llamarse la función a si misma "n" veces, dichos casos serían tratados "n" veces, lo que sería ilógico pues en la primera llamada ya se sabe si se trata, ono, de un caso especial.

El código respectivo, tomando en cuenta las anteriores consideraciones, es el siguiente:

```
c:\HPU\programas\c++\2-2012\tema9>notepad ejem1.cpp
```

```
#include <cstdio>
#include <cmath>

unsigned leerNumero(){
  double n;
  do{
    printf("%20s:? ","Número");
    scanf("%lf",&n);
    if (fmod(n,1)) {
      printf("%20s: %s\n","Error","El número debe ser entero.");
      continue;
    }
    if (n<0){
      printf("%20s: %s\n","Error","El número debe ser positivo.");
      continue;
    }
    break;
  } while(true);
  return (unsigned)n;
}

double factorial(unsigned n){
  if (n==0) return 1; else return factorial(n-1)*n;
}

void mostrarFactorial(double r){
  printf("%20s: %.12g\n","Factorial",r);
}

int main(){
  unsigned n;
  double r;
```

```
    printf("***** Calculo del factorial *****\n");
    printf("Escriba 0 para salir\n\n");
    do{
      n=leerNumero();
      r=factorial(n);
      mostrarFactorial(r);
      printf("\n");
    }while(n!=0);
    return 0;
}
```

Observe que en este caso los datos se piden dentro de un ciclo "do-while" hasta que el número introducido sea 0. Ello permite hacer correr el programa con varios valores sin necesidad de ejecutar una y otra vez el programa.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
c:\HPV\programas\c++\2-2012\tema9>g++ -c ejem1.cpp

c:\HPV\programas\c++\2-2012\tema9>g++ -o ejem1 ejem1.o

c:\HPV\programas\c++\2-2012\tema9>ejem1
***** Calculo del factorial *****
Escriba 0 para salir

      N·mero:? 5
Factorial: 120

      N·mero:? 10
Factorial: 3628800

      N·mero:? 90
Factorial: 1.48571596448e+138

      N·mero:? 7.3
       Error: El n·mero debe ser entero.
      N·mero:? -45
       Error: El n·mero debe ser positivo.
      N·mero:? 0
Factorial: 1
```

A pesar de que el proceso recursivo no forma parte del problema (pues del mismo se encarga el lenguaje), es importante comprender claramente como opera pues con ese conocimiento resulta más sencillo aplicar el razonamiento recursivo.

Para ello la mejor alternativa consiste en hacer correr el programa paso a paso (depurar el programa). En este tema se empleará el depurador directamente desde la línea de comando. El depurador "gdb.exe" se encuentra en la carpeta "bin" de "MinGW", sin embargo, es conveniente actualizar el mismo pues presenta algunos errores que han sido corregidos en las últimas versiones. Para ello se puede bajar la última versión (para Windows) de http://www.equation.com/servlet/equation.cmd?fa=gdb y copiarla en la carpeta "bin" de "MinGW" (**se recomienda hacer una copia de la versión anterior por si se tuviera algún problema con la nueva versión**).

Para depurar el programa primero debe ser vuelto a compilar, pero en el modo de depuración, para lo cual simplemente se añada la instrucción "-g" delante de las instrucciones habituales:

```
c:\HPV\programas\c++\2-2012\tema9>g++ -g -c ejem1.cpp

c:\HPV\programas\c++\2-2012\tema9>g++ -g -o ejem1 ejem1.o
```

Cuando se compila en el modo de depuración (con "-g") los archivos resultantes son más grandes de lo habitual porque contienen información adicional que puede ser ejemplada por el depurador ("gdb.exe"). Por esta razón, sólo se compila en este modo cuando se está probando el programa, pero una vez probado, la versión final debe ser compilada en el modo normal (eclipse y otros ambientes de desarrollo compilan por defecto en el modo de depuración).

Para iniciar la depuración se escribe "gdb" seguido del nombre del programa que se quiere depurar (en este caso "ejem1"):

```
c:\HPV\programas\c++\2-2012\tema9>gdb ejem1
```

Con lo que aparece en pantalla un mensaje similar al siguiente:

```
GNU gdb (GDB) 7.4.50.20120410
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.ht
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-pc-mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
This binary was built by Equation Solution <http://www.Equation.com>...
Reading symbols from c:\HPV\programas\c++\2-2012\tema9\ejem1.exe...done.
(gdb)
```

Que informa, entre otras cosas, de la versión del depurador que se está empleando (en este caso 7.4.5). Como se puede ver, después del mensaje, el indicador de la línea de comando cambia a "(gdb)".

En este modo (en el modo de depuración), si se quiere depurar el programa desde el principio se ejecuta el comando "start":

```
(gdb) start
Temporary breakpoint 1 at 0x401436: file ejem1.cpp, line 33.
Starting program: c:\HPV\programas\c++\2-2012\tema9\ejem1.exe
[New Thread 3884.0xf5c]

Temporary breakpoint 1, main () at ejem1.cpp:33
33          printf("***** Calculo del factorial *****\n");
```

Como se puede ver este comando inicia la depuración del programa y detiene la ejecución en la primera línea de la función "main" (línea 33), creando para ello un punto de ruptura (un "breakpoint") temporal.

Ahora se puede ejecutar las instrucciones de esta línea (la línea 33) y pasar a la siguiente con el comando "step" o el comando "next". La diferencia entre estos comandos es que "step" ejecuta las instrucciones y si en dichas instrucciones existe una llamada a otra función ingresa a la misma, mientras que "next" ejecuta las instrucciones pero no ingresa a ninguna función.

Como en este ejemplo se quiere depurar la función "factorial", se ejecuta el programa con "next" hasta llegar a la línea donde se llama a "factorial":

```
(gdb) next
***** Calculo del factorial *****
34          printf("Escriba 0 para salir\n\n");
(gdb) next
Escriba 0 para salir

36          n=leerNumero();
(gdb) next
              N·mero:? 5
37          r=factorial(n);
```

Ahora, para ingresar a la función "factorial" se emplea "step":

```
(gdb) step
factorial (n=5) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
```

Como se puede ver, el programa pasa a la primera línea (línea 23) de la función "factorial", además, como se puede ver, muestra el valor del parámetro "n", que en este caso es igual a 5.

Ahora, para ver qué es lo que ocurre en la función recursiva, se continua la ejecución con "step" (para que vuelva a ingresar a la función cuando se ejecuten las llamadas recursivas) y como "step" ha sido el último comando ejecutado, se ejecuta con la tecla "Enter":

```
(gdb)
factorial (n=4) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)
factorial (n=3) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)
factorial (n=2) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)
factorial (n=1) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)
factorial (n=0) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
```

Como se puede ver, la función se llama a si misma disminuyendo el valor de "n" en cada llamada hasta que "n" es cero. Ahora como "n" es cero, la función termina y van terminando también todas las llamadas recursivas hasta que el resultado es devuelto a la función principal:

```
(gdb)
24      }
(gdb)
24      }
(gdb)
24      }
(gdb)
24      }
(gdb)
24      }
(gdb)
24      }
(gdb)
main () at ejem1.cpp:38
38          mostrarFactorial(r);
```

Tal como se ve, el depurador no muestra el resultado devuelto, sino simplemente la llave que cierra el módulo y una vez que terminan todas las llamadas recursivas sale a la función "main" y pasa a la línea 38. En esta línea se llama a la función "mostrarFactorial" y como en este caso no se quiere depurar dicha función se la ejecuta con "next" (recuperando la instrucción con el cursor hacia arriba):

```
(gdb) next
          Factorial: 120
39          printf("\n");
```

Que muestra el resultado correcto (para n=5) y dado que en este ejemplo no se quiere depurar otro módulo, se continua ejecutando el programa en el modo normal, escribiendo pare ello el comando "continue":

```
(gdb) continue
Continuing.

          N·mero:? 0
          Factorial: 1
```

```
[Inferior 1 (process 3884) exited normally]
```

Al terminar la ejecución del programa el depurador informa que el programa ha concluido normalmente y queda en espera de la siguiente instrucción.

La mayoría de las veces sólo se quiere depurar una parte del programa, por lo que normalmente se emplean puntos de ruptura ("breakpoint") en el lugar donde se quiere detener la ejecución de programa (y comenzar la depuración). Por ejemplo, para detener la ejecución del programa en la función "factorial" se escribe:

```
(gdb) break factorial
Breakpoint 2 at 0x4013b4: file ejem1.cpp, line 23.
```

Entonces "gdb" informa que ha creado un punto de ruptura (el punto de ruptura número 2) en la línea 23 del programa (que es la primera línea de la función "factorial").

Una vez creado el punto de ruptura se puede ejecutar el programa en el modo normal con el comando "run" y al introducir un dato (en este ejemplo el número 5), el programa se detiene en la función factorial (en el punto de ruptura creado):

```
(gdb) run
Starting program: c:\HPV\programas\c++\2-2012\tema9\ejem1.exe
[New Thread 6232.0x1e08]
***** Calculo del factorial *****
Escriba 0 para salir

              N·mero:? 5

Breakpoint 2, factorial (n=5) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
```

Ahora se puede depurar el programa con los comandos "step" (como se vio previamente) o "next", según sea conveniente. Aquí e empleará el comando "next" (hasta que el valor de "n" sea cero):

```
Breakpoint 2, factorial (n=5) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb) next

Breakpoint 2, factorial (n=4) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)

Breakpoint 2, factorial (n=3) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)

Breakpoint 2, factorial (n=2) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)

Breakpoint 2, factorial (n=1) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
(gdb)

Breakpoint 2, factorial (n=0) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
```

Ahora, para ver los resultados que devuelven las diferentes llamadas recursivas se emplea el comando "finish" (este comando ejecuta la función actual hasta que termina y muestra el valor devuelto):

```
(gdb) finish
Run till exit from #0  factorial (n=0) at ejem1.cpp:23
0x004013cd in factorial (n=1) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
Value returned is $7 = 1
```

Como se puede ver "finish" informa que ha ejecutado la función factorial para "n" igual a 0 y que dicha función devuelve 1 a la función que hizo la llamada (la función para n=1). Continuando la depuración con este comando, hasta que la función a la que se devuelve el resultado es "main", se obtiene:

```
(gdb) finish
Run till exit from #0  0x004013cd in factorial (n=1) at ejem1.cpp:23
0x004013cd in factorial (n=2) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
Value returned is $8 = 1
(gdb)
Run till exit from #0  0x004013cd in factorial (n=2) at ejem1.cpp:23
0x004013cd in factorial (n=3) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
Value returned is $9 = 2
(gdb)
Run till exit from #0  0x004013cd in factorial (n=3) at ejem1.cpp:23
0x004013cd in factorial (n=4) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
Value returned is $10 = 6
(gdb)
Run till exit from #0  0x004013cd in factorial (n=4) at ejem1.cpp:23
0x004013cd in factorial (n=5) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
Value returned is $11 = 24
(gdb)
Run till exit from #0  0x004013cd in factorial (n=5) at ejem1.cpp:23
0x00401461 in main () at ejem1.cpp:37
37           r=factorial(n);
Value returned is $12 = 120
```

De esta manera se puede ver con mayor claridad como, en el proceso recursivo, el resultado devuelto por una función es empleado para calcular el nuevo valor del factorial. Así al ejecutar por tercera vez el comando "finish" se ve que el resultado devuelto por la función (para n=3) es 6, el cual al ser multiplicado por el valor actual de "n" (4) da 24, que, como se puede ver, es el resultado devuelto a la función que hizo la llamada (la función para n=5).

Ahora que se ha comprobado la lógica y el control del programa se encuentra en el módulo principal (main), se puede continuar la ejecución del programa en el modo normal (con continue):

```
(gdb) continue
Continuing.
          Factorial: 120

             N·mero:? 7

Breakpoint 2, factorial (n=7) at ejem1.cpp:23
23          if (n==0) return 1; else return factorial(n-1)*n;
```

Pero como se puede ver, el programa vuelve a detenerse en el breakpoint 2. Si se quiere continuar con la ejecución normal del programa (sin que se detenga en el breakpoint) se puede inhabilitar o suprimirlo definitivamente el mismo.

Para inhabilitar temporalmente un breakpoint se emplea el comando "disable" seguido del número de breakpoint a inhabilitar (o del nombre de la función en la que se creo el breakpoint). Así para inhabilitar el breakpoint 2 se escribe:

```
(gdb) disable 2
```

Ahora, al ejecutar el comando "continue" el programa no se detiene más en el breakpoint:

```
(gdb) continue
```

```
Continuing.
          Factorial: 5040

             N-mero:? 6
          Factorial: 720

             N-mero:? 0
          Factorial: 1

[Inferior 1 (process 6976) exited normally]
```

Para volver a habilitar un breakpoint se emplea el comando "enable":

```
(gdb) enable 2
```

Ahora, al hacer correr el programa se detiene en el breakpoint:

```
(gdb) run
Starting program: c:\HPV\programas\c++\2-2012\tema9\ejem1.exe
[New Thread 7420.0x1ea0]
***** Calculo del factorial *****
Escriba 0 para salir

             N-mero:? 5

Breakpoint 2, factorial (n=5) at ejem1.cpp:23
23        if (n==0) return 1; else return factorial(n-1)*n;
```

Para suprimir definitivamente uno o más breakpoints se emplea el comando "delete" o "clear". Así para suprimir definitivamente este breakpoint se escribe:

```
(gdb) delete 2
```

Después de esta orden el breakpoint no existe más, por lo que al ejecutar el comando "continue" el programa corre en el modo normal:

```
(gdb) continue
Continuing.
          Factorial: 120

             N-mero:? 7
          Factorial: 5040

             N-mero:? 0
          Factorial: 1

[Inferior 1 (process 7420) exited normally]
```

Con "clear" se puede escribir el nombre de la función para suprimir el breakpoint (si es que el breakpoint ha sido creado en una función, como en este ejemplo). Una vez suprimido el breakpoint (con delete o clear) no puede ser recuperado, por lo que si se quiere detener el programa nuevamente en ese punto se debe crear un nuevo breakpoint.

Si se quiere terminar la depuración del programa (porque se ha ingresado a un ciclo infinito o porque ya se ha encontrado el error) se emplea el comando "kill".

Para salir del modo de depuración se escribe el comando "quit" (o abreviado "q"):

```
(gdb) quit

c:\HPV\programas\c++\2-2012\tema9>
```

El depurador "gdb" cuenta con muchos otros comandos que pueden ser estudiados si se pide ayuda, con "help", estando en el modo de depuración (o bajando la documentación del depurador, disponible en la misma página donde se encuentra el ejecutable (http://www.equation.com/servlet/equation.cmd?fa=gdb).

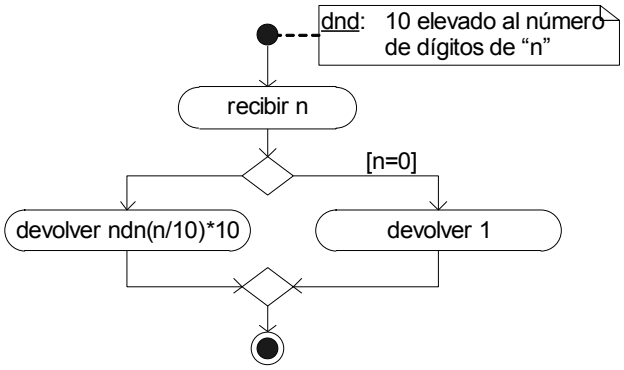
2. Elabore un programa que, empleando la recursividad y librerías tipo C++, calcule el valor de 10 elevado al número de dígitos existentes en un número entero positivo. Por ejemplo, si el número es 458329, el resultado deberá ser: 10^6 (1000000).

Si se tiene el valor de 10 elevado al número de dígitos del número sin su último dígito, el resultado se calcula simplemente multiplicando dicho valor por 10. Por ejemplo si el número es 458329 y se conoce el valor de 10 elevado al número de dígitos de 45832 (esto es 100000) entonce el valor de 10 elevado al número de dígitos de 458329 es: 100000*10 = 1000000, es decir:

$$10^{nd(458329)} = 10^{nd(45832)} * 10 = 10^5 * 10 = 10^6$$

Al devolver el planteamiento recursivo el resultado correcto, se sabe que la lógica es correcta, sin embargo y como ya se dijo, para completar el planteamiento recursivo se requiere de una condición de finalización. Puesto que 10 elevado al número de dígitos de 0 es 1 ($10^0 = 1$), esa será la condición de finalización que completa el planteamiento recursivo.

Con las anteriores consideraciones el algoritmo es:



Y el código respectivo, es:

```
c:\HPU\programas\c++\2-2012\tema9>notepad ejem2.cpp
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

unsigned leerNumero(){
  double n;
  while(true){
    cout<<setw(25)<<"Numero"<<"? ";
    cin>>n;
    if(fmod(n,1)){
      cout<<setw(25)<<"Error"<<": El numero debe ser entero."<<endl;
      continue;
    }
    if(n<0){
      cout<<setw(25)<<"Error"<<": El numero debe ser positivo."<<endl;
      continue;
    }
    break;
  }
  return (unsigned)n;
}
```

```
double dnd(unsigned n){
  if (n==0) return 1; else return dnd(n/10)*10;
}

void mostrarNumero(double n){
  cout<<setw(25)<<"10^(Numero de digitos)"<<": "<<n<<endl;
}

int main(){
  unsigned n;
  double r;
  cout<<"***** 10 elevado al número de dígitos *****"<<endl;
  cout<<"Introduzca 0 para terminar"<<endl<<endl;
  do{
    n=leerNumero();
    r=dnd(n);
    mostrarNumero(r);
    cout<<endl;
  }while(n!=0);
  return 0;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
c:\HPU\programas\c++\2-2012\tema9>g++ -c ejem2.cpp

c:\HPU\programas\c++\2-2012\tema9>g++ -o ejem2 ejem2.o

c:\HPU\programas\c++\2-2012\tema9>ejem2
***** 10 elevado al n·mero de dÝgitos *****
Introduzca 0 para terminar

                  Numero? 12345
  10^(Numero de digitos): 100000

                  Numero? 1234567
  10^(Numero de digitos): 1e+07

                  Numero? 123.45
                   Error: El numero debe ser entero.
                  Numero? -12345
                   Error: El numero debe ser positivo.
                  Numero? 0
  10^(Numero de digitos): 1
```

No olvide hacer correr el programa paso a paso para comprender mejor la lógica recursiva involucrada.

9.2. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema9" (tanto para los ejemplos como para los ejercicios). No olvide emplear el depurador "gdb" para encontrar errores y comprender la lógica de los programas elaborados.

1. Elabore y pruebe un programa que, empleando librerías tipo C++ y el ra-

zonamiento recursivo, encuentre la sumatoria de los primeros "n" números enteros positivos.

2. Elabore y pruebe un programa que, empleando librerías tipo C y el razonamiento recursivo, encuentre la sumatoria de los primeros "n" números impares positivos.

3. Elabore y pruebe un programa que, empleando librerías tipo C++ y el razonamiento recursivo, calcule la potencia entera "n" de un número real "x", tomando en cuenta que x^n = x*x*x*x*... (n veces).

4. Elabore y pruebe un programa que, empleando librerías tipo C y el razonamiento recursivo, reciba un número entero positivo y devuelva el número con sus dígitos invertidos. El porgrama debe hacer uso de la función "dndr" elaborada en el ejemplo.

5. Elabore y pruebe un programa que, empleando librerías tipo C++ y el razonamiento recursivo, reciba un número entero positivo y devuelva el primer dígito de dicho número (denomine a la función recursivo "pdr").

6. Elabore y pruebe un programa que, empleando librerías tipo C y el razonamiento recursivo, reciba un número entero positivo y devuelva dicho número sin su primer dígito (denomine a la función recursiva "spdr").

7. Elabore y pruebe un programa que, empleando librerías tipo C++ y el razonamiento recursivo, reciba un número entero positivo y devuelva el número con sus dígitos invertidos. El programa debe hacer uso de las funciones "pdr" y "spdr" creadas en los anteriores ejercicios.

10. RECURSIVIDAD 2

En este tema continua el repaso la recursividad con algunos ejemplos y ejercicios que permitirán asimilar y comprender mejor el concepto recursivo.

10.1. EJEMPLOS

1. Elabore un programa que, empleando la recursividad y librerías tipo C++, calcule la potencia entera de un número real.

Desde el punto de vista recursivo el problema se resuelve multiplicando por "*x*" la potencia previa, es decir:

$$x^n = x^{n-1} \cdot x$$

Y dado que cualquier número elevado a 0 es uno, esa puede ser la condición de finalización:

$$x^0 = 1$$

No obstante, esta solución, aunque correcta, es ineficiente porque implica la repetición innecesaria de operaciones, por ejemplo para calcular $3.45^{2500000}$, se requieren ¡2 millones quinientas mil llamadas recursivas!, por lo tanto se deben crear y destruir ¡2 millones quinientas mil funciones!, sin embargo, si se toma en cuenta que $3.45^{2500000}$ es igual a $(3.45^{1250000})^2$, se puede reducir el número de llamadas en ¡1 millón ciento veinticinco mil!, y si se aplica el mismo procedimiento a la potencia que se encuentra entre paréntesis el número de llamadas se reduce aún más.

En general entonces, la potencia entera de un número real puede ser calculada aplicando la expresión:

$$x^n = \left(x^{\frac{n}{2}}\right)^2 \quad \text{si n es par}$$

$$x^n = \left(x^{\frac{n}{2}}\right)^2 \cdot x \quad \text{si n es impar}$$

Así por ejemplo el valor de x^{4561} puede ser calculado con:

$$x^{4561} = (x^{(4561-1)/2})^2 * x = (x^{2280})^2 * x$$

Para calcular x^{2280} se aplica el mismo procedimiento:

$$x^{2280} = (x^{2280/2})^2 = (x^{1140})^2$$

Prosiguiendo de esta manera los siguientes cálculos son:

$$x^{1140} = (x^{1140/2})^2 = (x^{570})^2$$

$$x^{570} = (x^{570/2})^2 = (x^{285})^2$$

$$x^{285} = (x^{(285-1)/2})^2 * x = (x^{142})^2 * x$$

$$x^{142} = (x^{142/2})^2 = (x^{71})^2$$

$$x^{71} = (x^{71/2})^2 * x = (x^{35})^2 * x$$

$$x^{35} = (x^{(35-1)/2})^2 * x = (x^{17})^2 * x$$

$$x^{17} = (x^{(17-1)/2})^2 * x = (x^{8})^2 * x$$

$$x^{8} = (x^{8/2})^2 = (x^{4})^2$$

$$x^4 = (x^{4/2})^2 = (x^2)^2$$

$$x^2 = (x^{2/2})^2 = (x^1)^2$$

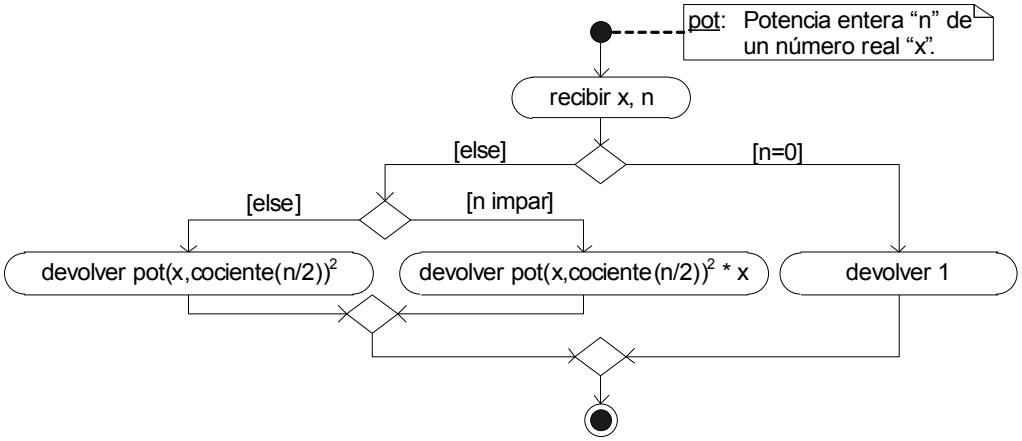
$$x^1 = x$$

En consecuencia, aplicando este procedimiento se requieren 13 repeticiones y 18 multiplicaciones, en lugar de ¡4561 repeticiones y 4561 multiplicaciones! (una reducción del 99.6%).

Como se puede observar este planteamiento es recursivo y por ello la forma "natural" de implementarlo es con dicha propiedad.

Como se sabe, para completar el planteamiento recursivo se requiere una condición de finalización y dado que x^0 es 1, esa será la condición de finalización.

El algoritmo que resuelve el problema es:



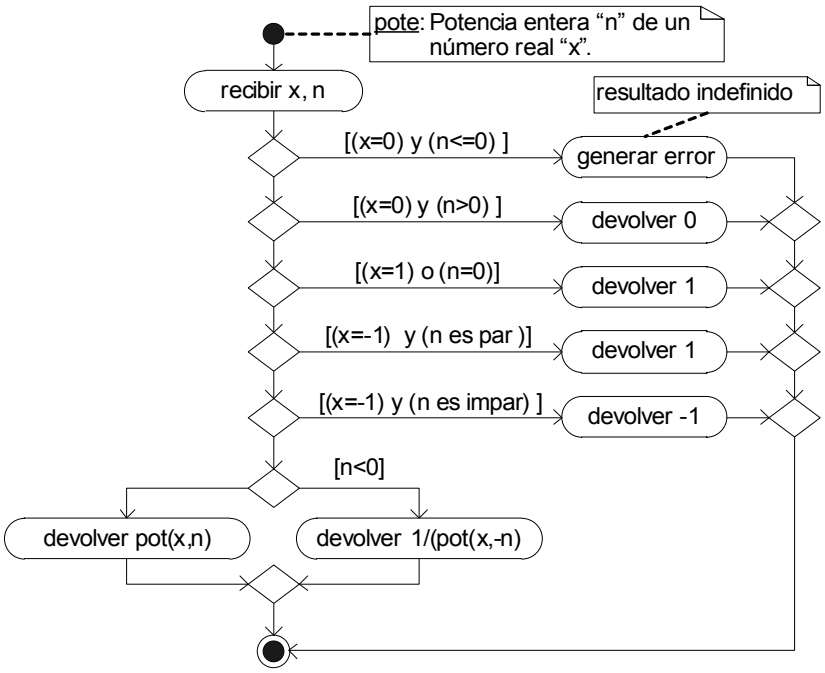
En este problema, sin embargo, existen muchos casos que no han sido tomados en cuenta tales como cuando la base es 1 (donde sin importar la potencia la solución es 1), o cuando la base es cero (donde, sin importar la potencia la solución es 0) o cuando la potencia es negativa (siendo la solución la inversa del valor calculado), o cuando la base y la potencia son cero (resultado indeterminado), etc., etc. Como ya se dijo previamente, todos los casos especiales deben ser analizados y tratados en un módulo independiente, no en el módulo recursivo (porque sería un error lógico).

El algoritmo del módulo que analiza los diferentes casos se presenta en la siguiente página y es desde este módulo que se hace la llamada al módulo recursivo (después de analizar todos los posibles casos).

El código del programa es:

```
C:\HPV\programas\c++\2-2012\tema10>notepad ejem1.cpp
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
using namespace std;

double leerBase(){
  double x;
  cout<<setw(20)<<"Base"<<"? ";
  cin>>x;
```




```
  return x;
}

int leerPotencia(){
  double n;
  do{
    cout<<setw(20)<<"Potencia:"<<"? ";
    cin>>n;
    if(fmod(n,1)){
      cout<<setw(20)<<"Error"<<": La potencia debe ser entera."<<endl;
      continue;
    }
    break;
  }while(true);
  return (int)n;
}

//función para calcular el cuadrado de un número
double sqr(double x){return x*x;}

//Función recursiva para el cálculo de la potencia
double potEntr(const double x, unsigned n){
  if (n==0) return 1;
  else return n%2 ? sqr(potEntr(x,n/2))*x : sqr(potEntr(x,n/2));
}

//Función principal, no recursiva, para el cálculo de la potencia
double potEnt(double x, int n){
  if (x==0 && n<=0) {
    cout<<setw(20)<<"Error"<<": Resultado indefenido."<<endl;
    exit(EXIT_FAILURE);}
  if (x==0 && n>0) return 0;
  if (x==1 || n==0) return 1;
```

```
  if (x==-1 && !n%2) return 1;
  if (x==-1 && n%2) return -1;
  return n<0 ? 1/potEntr(x,-n) : potEntr(x,n);
}

void mostrarResultado(double r){
  cout<<setw(20)<<"Potencia entera"<<": "
    <<setprecision(12)<<r<<endl<<endl;
}

int main(){
  double x,r;
  int n;
  cout<<"***** Potencia entera de x^n *****"<<endl;
  cout<<"Introduzca base y potencia = 0 para salir"<<endl<<endl;
  do{
    x=leerBase();
    n=leerPotencia();
    r=potEnt(x,n);
    mostrarResultado(r);
  }while(x!=0 || n!=0);
  return EXIT_SUCCESS;
}
```

En este programa se ha creado la función "sqr" para facilitar el cálculo del cuadrado de la fórmula recursiva. Observe también que la función "potEntr" recibe la base como constante, esto por dos razones: a) Para la función, dicho valor efectivamente es una constante (su valor no debe ser modificado dentro de la función) y b) Cuando una función recibe un parámetro como constante no se crea una copia del valor original, sino que se emplea el valor original, de forma similar a los parámetros por referencia, sólo que en este caso dicho valor no puede ser modificado. De esa manera se evita crear innecesariamente "n" copias de "x" (una por cada llamada recursiva).

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

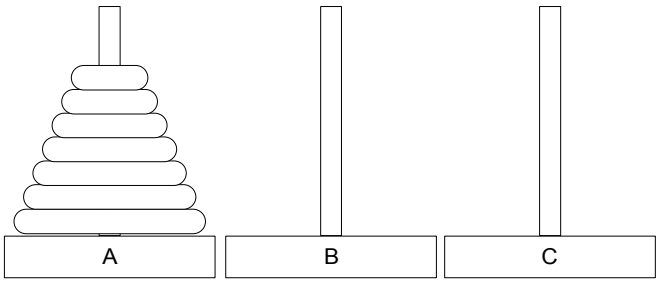
```
C:\HPV\programas\c++\2-2012\tema10>g++ -c ejem1.cpp

C:\HPV\programas\c++\2-2012\tema10>g++ -o ejem1 ejem1.o

C:\HPV\programas\c++\2-2012\tema10>ejem1
***** Potencia entera de x^n *****
Introduzca base y potencia = 0 para salir

               Base? 3
          Potencia:? 9
    Potencia entera: 19683

               Base? 2.3
          Potencia:? 20
    Potencia entera: 17161558.3133

               Base? 5.4
          Potencia:? -4
    Potencia entera: 0.00117604776447

               Base? 0
          Potencia:? 23
    Potencia entera: 0

               Base? 34
```

```
        Potencia:? 0
Potencia entera: 1

            Base? 4.3
        Potencia:? 7.1
            Error: La potencia debe ser entera.
        Potencia:? 7
Potencia entera: 27181.8611107

            Base? 0
        Potencia:? 0
            Error: Resultado indefenido.
```

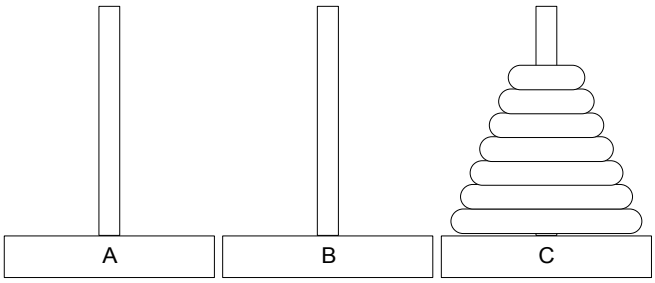
No olvide depurar el programa con "gdb" haciendo correr el mismo paso a paso y/o colocando puntos de ruptura (breakpoints) donde se tenga dudas o se quiera analizar la lógica. Emplee esta herramienta para responder a las siguientes preguntas: ¿Es necesaria la instrucción "else" en la función "potEntr"? ¿Por qué la función "PotEntr" recibe un entero sin signo? ¿Por qué en la función "potEnt" no se emplea la instrucción "else" en ningún caso? ¿Como se resolvería el problema si no se crea la función "sqr"?

2. Elabore un programa que, empleando la recursividad y librerías tipo C, devuelva un texto con los movimientos necesarios para resolver el juego de las torres de Hanoi.

El juego de las torres de Hanoi consta de tres postes y una serie de aros de diferente diámetro, que se encuentran ordenados como se muestra en la figura:



El objetivo del juego consiste en mover estos aros al tercer poste (el poste C), de manera que queden en el mismo orden en el que se encontraban en el poste original (el poste A):



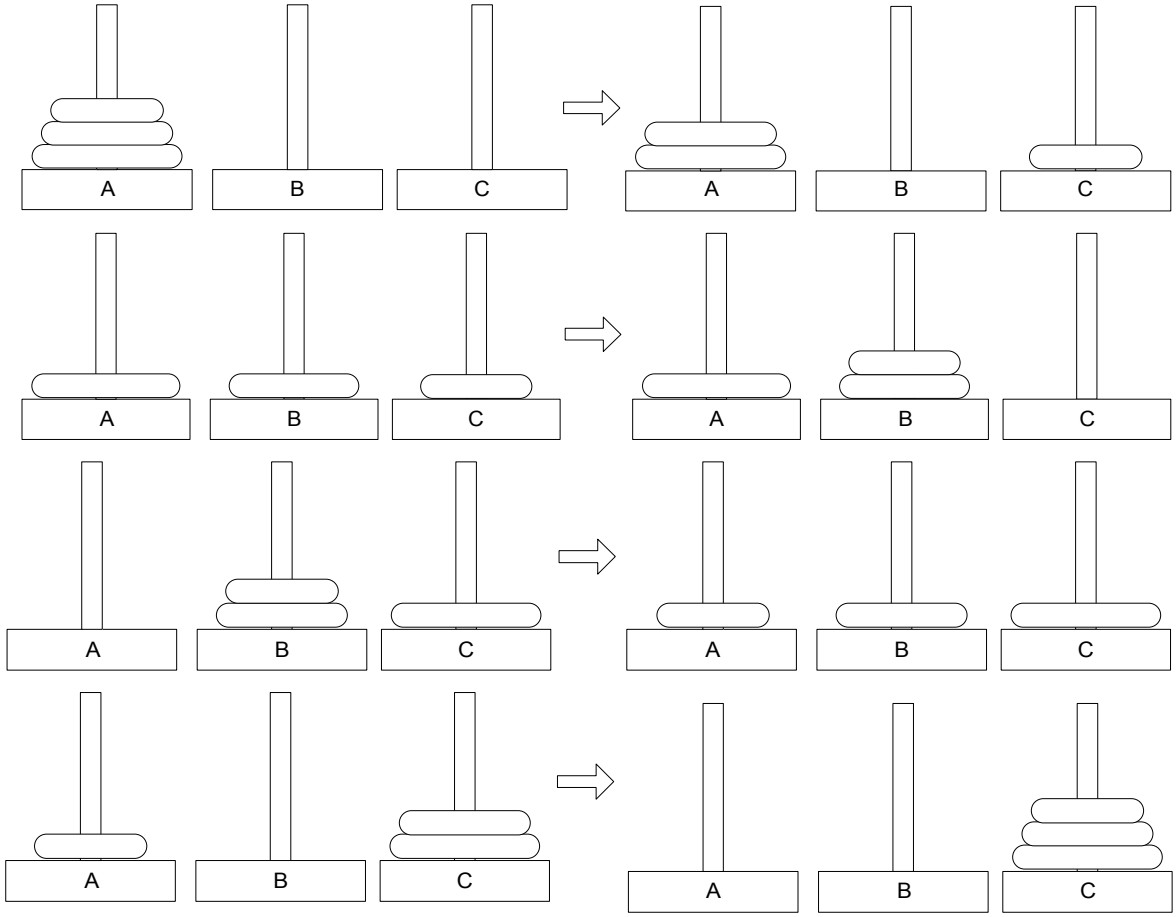
Las reglas del juego son simples:

a) Sólo está permitido mover un aro a la vez.

b) No se puede colocar un aro de mayor diámetro sobre otro de menor diámetro.

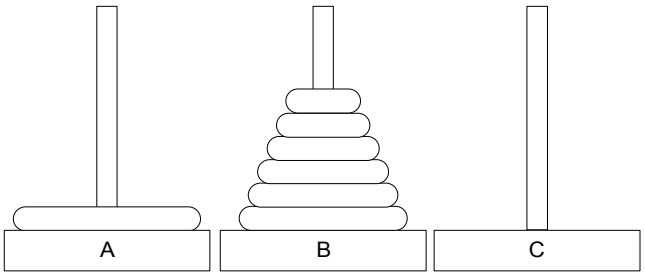
c) En concordancia con la segunda regla, el aro de mayor tamaño sólo puede ser movido a un poste vacío.

Para comprender mejor la forma en que se resuelve el juego, se muestra a continuación los movimientos que se deben realizar para llevar 3 aros desde el poste “A” hasta el poste “C”.

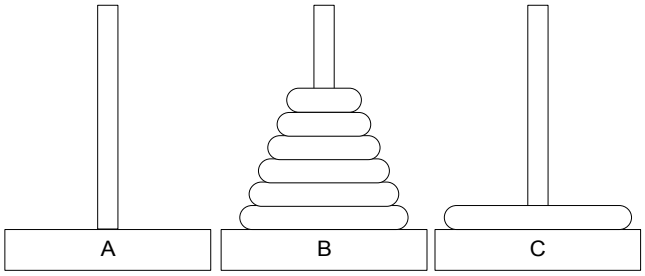


Es decir que en este caso los movimientos necesarios para ordenar los aros en el poste "C" son: "A->C", "A->B", "C->B", "A->C", "B->A", "B->C" y "A->C", que se entiende como: mover el aro superior del poste "A" al poste "C", luego mover el aro superior del poste "A" al poste "B", después mover el aro superior del poste "C" al poste "B", mover el aro superior del poste "A" al poste "C", mover el aro superior del poste "B" al poste "A", mover el aro superior del poste "B" al poste "C" y finalmente mover el aro del poste "A" al poste "C".

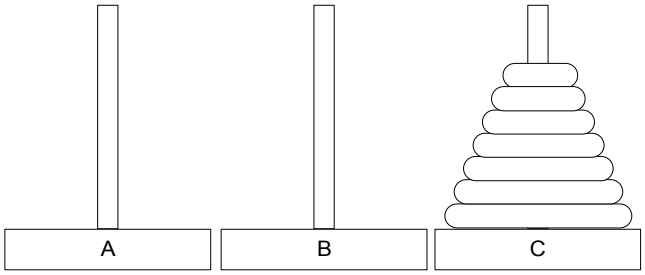
La solución más sencilla de este problema es por naturaleza recursiva: Si ya se han movido “n-1” aros del poste “A” al poste “B”:



Se pude llevar el aro restante al poste “C”:



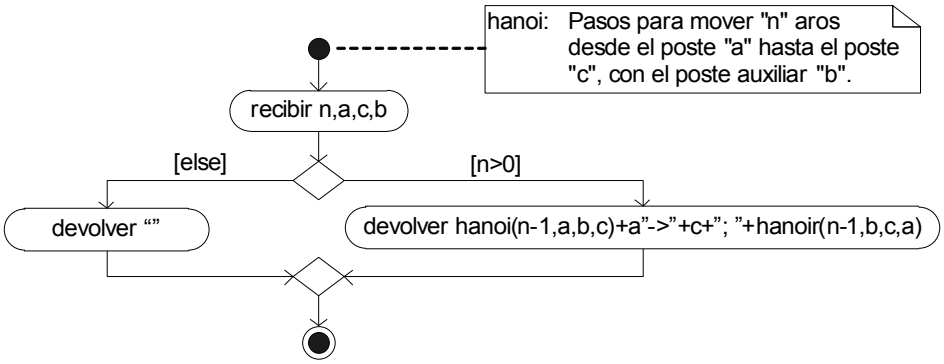
Entonces se pueden mover los "n-1" aros del poste "B" hasta el poste "C", con lo que se logra el objetivo:



En cierto sentido se puede pensar que se está haciendo trampa, porque aparentemente no se están respetando las reglas del juego (mover un aro a la vez), pero no es así, lo que se está haciendo es aprovechar la propiedad recursiva: para mover los "n-1" aros (uno por uno) el módulo se llamará a si mismo, cuantas veces sea necesario, pero de eso se encarga el proceso recursivo, no el programador (o jugador).

Finalmente, para que el razonamiento recursivo quede completo se requiere una condición de finalización. En este caso se sabe que cuando un palo queda sin aros ya no se realiza ningún movimiento, por lo que esa será la condición de finalización.

El algoritmo que implementa esta solución recursiva es:



El código respectivo es:

```
C:\HPU\programas\c++\2-2012\tema10>notepad ejem2.cpp
```

```
#include <cstdio>
#include <string>
#include <cmath>
#include <cstdlib>
using namespace std;

unsigned leerNumero(){
  double n;
```

```
  do{
    printf("%20s? ","Numero de aros");
    scanf("%lf",&n);
    if (fmod(n,1)){
      printf("%20s: %s\n","Error","El numero debe ser entero.");
      continue;
    }
    if (n<0){
      printf("%20s: %s\n","Error","El numero debe ser positivo.");
      continue;
    }
    break;
  }while(true);
  return (unsigned)n;
}

string hanoi(unsigned n, string a, string c, string b){
  if (n>0) return hanoi(n-1,a,b,c)+a+"->"+c+"; "+hanoi(n-1,b,c,a);
  else return "";
}

void mostrarMovimientos(string &s){
  printf("%20s: \n","Los movimientos necesarios son");
  unsigned m=s.length(),i=0,nm=60;
  do{
    printf("%s\n",s.substr(i,nm).c_str());
    i+=nm;
  }while (i<m);
  printf("\n");
}

int main(){
  unsigned n;
  string s;
  printf("\n**** Torres de Hanoi *****\n");
  printf("Introduzca 0 para salir\n\n");
  do{
    n=leerNumero();
    s=hanoi(n,"A","C","B");
    mostrarMovimientos(s);
  }while(n!=0);
  return EXIT_SUCCESS;
}
```

Observe que en este programa, a pesar de que las operaciones de entrada/saida se las realiza con librerías tipo C, se incluye la instrucción "using namespace std". Esto se debe a que la librería "string", que es donde se encuentra el tipo "string" empleado en el programa, es una librería C++ y como tal sus nombres están definidos en el espacio de nombres estándar (std). El problema puede ser resuelto empleando librerías tipo C únicamente, pero ello implica la manipulación de vectores y punteros, algo que se verá en temas posteriores.

El número de caracteres que se muestra en cada fila (60), en la función "mostrarMovimientos", se fija en la variable "nm". Para mostrar ese número de caracteres, en cada repetición del ciclo "do-while", se llama al método

"substr", el cual recibe dos parámetros: la posición del primer carácter a extraer (almacenado en el contador "i", el cual comienza en 0) y el número de caracteres a extraer (guardado, como se dijo, en la variable "nm"), pero además, como en este ejemplo se imprimen los caracteres con "printf", son convertidos al tipo *char (que es el tipo admitido por "printf") con el método "c_str". Este último paso no es necesario cuando se imprimen los resultados con "cout".

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
C:\HPV\programas\c++\2-2012\tema10>g++ -c ejem2.cpp

C:\HPV\programas\c++\2-2012\tema10>g++ -o ejem2 ejem2.o

C:\HPV\programas\c++\2-2012\tema10>ejem2

***** Torres de Hanoi *****
Introduzca 0 para salir

      Numero de aros? 3
Los movimientos necesarios son:
A->C; A->B; C->B; A->C; B->A; B->C; A->C;

      Numero de aros? 5
Los movimientos necesarios son:
A->C; A->B; C->B; A->C; B->A; B->C; A->C; A->B; C->B; C->A;
B->A; C->B; A->C; A->B; C->B; A->C; B->A; B->C; A->C; B->A;
C->B; C->A; B->A; B->C; A->C; A->B; C->B; A->C; B->A; B->C;
A->C;

      Numero de aros? 7
Los movimientos necesarios son:
A->C; A->B; C->B; A->C; B->A; B->C; A->C; A->B; C->B; C->A;
B->A; C->B; A->C; A->B; C->B; A->C; B->A; B->C; A->C; B->A;
C->B; C->A; B->A; B->C; A->C; A->B; C->B; A->C; B->A; B->C;
A->C; A->B; C->B; C->A; B->A; C->B; A->C; A->B; C->B; C->A;
B->A; B->C; A->C; B->A; C->B; C->A; B->A; C->B; A->C; A->B;
C->B; A->C; B->A; B->C; A->C; A->B; C->B; C->A; B->A; C->B;
A->C; A->B; C->B; A->C; B->A; B->C; A->C; B->A; C->B; C->A;
B->A; B->C; A->C; A->B; C->B; A->C; B->A; B->C; A->C; B->A;
C->B; C->A; B->A; C->B; A->C; A->B; C->B; C->A; B->A; B->C;
A->C; B->A; C->B; C->A; B->A; B->C; A->C; A->B; C->B; A->C;
B->A; B->C; A->C; A->B; C->B; C->A; B->A; C->B; A->C; A->B;
C->B; A->C; B->A; B->C; A->C; B->A; C->B; C->A; B->A; B->C;
A->C; A->B; C->B; A->C; B->A; B->C; A->C;

      Numero de aros? 4.3
            Error: El numero debe ser entero.
      Numero de aros? -4
            Error: El numero debe ser positivo.
      Numero de aros? 4
Los movimientos necesarios son:
A->B; A->C; B->C; A->B; C->A; C->B; A->B; A->C; B->C; B->A;
C->A; B->C; A->B; A->C; B->C;

      Numero de aros? 0
Los movimientos necesarios son:
```

Como se puede ver en los resultados, el número de movimientos necesarios incrementa geométricamente con el número de aros, así, para 20 aros se requieren más de un millón de movimientos (haga la prueba).

Al igual que el ejercicio anterior, para comprender mejor la lógica involucrada, depure y haga correr el programa paso a paso empleando "gdb", recuerde que para ello debe compilar y crear el ejecutable añadiendo la directiva "-g", es decir:

```
C:\HPV\programas\c++\2-2012\tema10>g++ -g -c ejem2.cpp
```

```
C:\HPV\programas\c++\2-2012\tema10>g++ -g -o ejem2 ejem2.o

C:\HPV\programas\c++\2-2012\tema10>gdb ejem2 -silent
Reading symbols from C:\HPV\programas\c++\2-2012\tema10\ejem2.exe...done.
(gdb)
```

En este caso se inicia la depuración empleando el interruptor "-silent" (o "quiet" o "-q") simplemente para que "gdb" no muestre el mensaje donde informa sobre el número de versión y tipo de licencia del programa.

10.2. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema10" (tanto para los ejemplos como para los ejercicios). No olvide emplear el depurador "gdb" para encontrar errores y comprender mejor la lógica involucrada.

1. Siguiendo un razonamiento similar al de la potencia entera, se puede calcular el exponente (e^x) de un número real (siendo "x" positivo). En este caso el valor de "x" se reduce hasta que sea menor o igual a 0.125, entonces el exponente se calcula con:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{10}}{10!}$$

   Elabore y pruebe un programa que, empleando la recursiva y librerías tipo C++, calcule el exponente de un número real siguiendo el anterior planteamiento (el exponente del número reducido debe ser calculado en otra función).
2. El cálculo de la potencia entera de un número real (x^n) puede ser implementado también de forma iterativa: Mientras la potencia entera sea mayor a 0 se calcula su cociente entre 2, contando (en la variable "c") el número de veces que se repite el proceso y guardando en un vector ("v") la información con relación a si la potencia es par (0) o impar (1). Luego se inicia un acumulador ("r") en 1 y en un ciclo, que se repite "c" veces, se guarda en el acumulador el resultado de multiplicar r*r si el correspondiente valor de "v" es 0, o el resultado de multiplicar r*r*x si el correspondiente valor de "v" es 1.
   Por ejemplo para calcular x^9, las divisiones y valores a almacenar en el vector son: 9/2=4, v[0]=1; 4/2=2, v[1]=0; 2/2=1, v[2]=0; 1/2=0, v[3]=1 y como el proceso se repite 4 veces, el resultado se calcula con (recordando que r=1): v[3]=1 => r=1*1*x=x; y[2]=0 => r=x*x=x^2; y[1]=0 => r=x^2*x^2=x^4; v[0]=1 => r=x^4*x^4*x=x^9.
   Como no se sabe de antemano el número de veces que se debe repetir el proceso se declara un vector con unos 500 elementos.
   Elabore y pruebe un programa que, empleando una función iterativa y librerías tipo C, calcule la potencia entera de un número real siguiendo el anterior razonamiento.
3. En el problema de las torres de Hanoi, la condición de finalización "natural" ocurre cuando el número de aros a mover es uno, en cuyo caso se mueve dicho aro desde el origen hasta el destino. Elabore y pruebe un programa que, empleando la recursividad y librerías tipo C++, resuelva el problema de las torres de Hanoi empleando la condición "natural" de finalización.
4. Aunque es una solución más complicada, sólo se la da a manera de ejercicio (no se debería emplear en ningún caso práctico), el problema de

las torres de Hanoi, puede ser resuelto también si se sigue el siguiente planteamiento: a) Se mueven “n-2” aros desde el poste de origen (A) hasta el poste de destino (C); b) Se mueve el penúltimo aro desde el poste de origen (A) al poste auxiliar (B); d) Se mueven “n-2” aros del poste de destino (C) al poste auxiliar (B); e) Se mueve el aro restante del poste de origen (A) hasta el poste de destino (C); f) Se mueven los “n-1” aros desde el poste auxiliar (B) hasta el de destino (C).

En este planteamiento, sin embargo, se debe tener el cuidado de considerar como condiciones de finalización los casos “n=0” y “n=1” (pues pueden darse ambos casos).

Elabore y pruebe una programa que, empleando la recursividad y librerías tipo C, resuelva el problema de las torres de Hanoi siguiendo este razonamiento.

11. RECURSIVIDAD 3

En este tema se completa el estudio de la recursividad con algunos ejemplos y ejercicios que permitirán ampliar y consolidar los conocimientos adquiridos. Se introduce además el mecanismo que se emplea en C++ para el tratamiento de los errores.

11.1. TRATAMIENTO DE ERRORES

Cuando se resuelve un problema no siempre es posible encontrar la solución buscada y/o realizar la tarea requerida, debido a una serie de factores tales como: los datos proporcionados no son correctos, se supera cierto límite o restricción, ocurren errores de redondeo o de propagación, no se ha considerado un determinado caso o simplemente no existe una solución para los datos proporcionados.

Una forma conveniente de hacer conocer de la ocurrencia de un error dentro de una función (o programa) es mediante la generación de errores, de esa manera, el que utiliza la función puede tomar las acciones pertinentes a fin de lidiar con el error producido.

Para generar un error en C++ se emplea la instrucción "throw", que tiene la siguiente sintaxis:

```
throw valor_o_expresión;
```

Esta instrucción provoca la inmediata finalización de la función, pero no devuelve ningún resultado, sino que genera una excepción con el "valor" o resultado de la "expresión". El "valor_o_expresión" puede ser de cualquier tipo válido en C++.

Las excepciones son un tipo de evento y los eventos son sucesos que ocurren tanto interna como externamente y que pueden afectar al programa (y/o al sistema operativo). En este caso, este tipo de evento informa sobre la ocurrencia de un error durante la ejecución del programa.

Los errores pueden ser tratados tanto en el lugar donde se generan como fuera de ellos y para realizar dicho tratamiento se emplea la estructura "*try-catch*":

```
try {
  instrucciones_normales;
} catch (tipo_de_dato_del_error variable) {
  instrucciones_en_caso_de_error;
}
```

Las "instrucciones_normales" son las instrucciones que se escriben para resolver el problema o llevar a cabo la tarea requerida (todo el código escrito hasta el momento corresponde a este tipo de instrucciones).

Las "instrucciones_en_caso_de_error" son las instrucciones que se ejecutan cuando ocurre algún error mientras se ejecutan las "instrucciones_normales" y en los casos más sencillos se trata simplemente de algún mensaje que informa sobre el error producido.

Si no se produce ningún error mientras se ejecutan las "instrucciones_normales", entonces el bloque "catch" (y las "instrucciones_en_caso_de_error") no se ejecutan.

La generación y tratamiento de errores es una técnica conveniente también para la validación de datos.

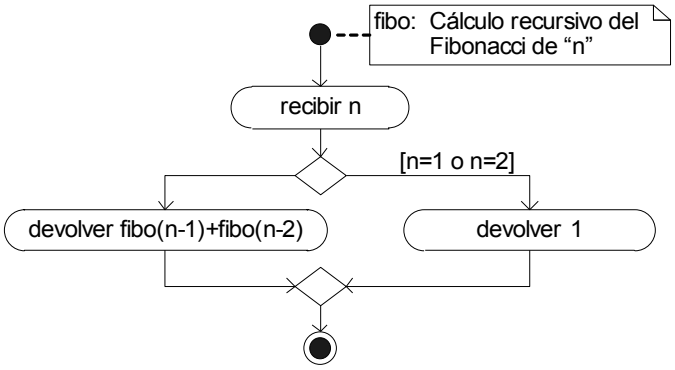
11.2. EJEMPLOS

1. Elabore un programa que, empleando la recursividad y librerías C++, calcule el Fibonacci de un número entero.

Como se recordará, la ecuación de definición del Fibonacci es:

$$F_n = F_{n-1} + F_{n-2}$$
$$F_1 = F_2 = 1$$

Esta es de hecho una definición recursiva: Se calcula un nuevo Fibonacci en base a otros dos valores conocidos. Por lo tanto el problema puede ser resuelto directamente aplicando el siguiente algoritmo:



Siendo el código respectivo:

```
c:\HPV\programas\c++\2-2012\tema11>notepad ejem1.cpp
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdio>
using namespace std;

unsigned leerNumero(){
  double n;
  try {
    cout<<setw(20)<<"Numero"<<"? ";
    cin>>n;
    if (fmod(n,1)) throw 1;
    if (n<0) throw 2;
    return (unsigned)n;
  } catch (int e) {
    cout<<setw(20)<<"Error";
    switch (e) {
      case 1:
         cout<<": El numero debe ser entero."<<endl;
         break;
      case 2:
         cout<<": El numero debe ser positivo."<<endl;
         break;
    }
    return leerNumero();
  }
}
```

```
//Función recursiva para el cálculo del Fibonacci
double fibonaccir(unsigned n){
  return n==0 || n==1 ? 1 : fibonaccir(n-1)+fibonaccir(n-2);
}

//Función que analiza los casos especiales y llama a la función recursiva
double fibonacci(unsigned n){
  if (n==0) throw 3;
  return fibonaccir(n);
}

void mostrarFibonacci(double f){
  cout<<setw(20)<<"El fibonacci es"<<": "<<f<<endl<<endl;
}

int main(){
  unsigned n;
  double f;
  cout<<endl<<"***** Calculo recursivo del Fibonacci *****"<<endl;
  cout<<"Introduzca 1 para salir"<<endl<<endl;
  do{
    try{
      n=leerNumero();
      f=fibonacci(n);
      mostrarFibonacci(f);
    } catch(int e){
        if (e==3) cout<<setw(20)<<"Error"
          <<": No existe el Fibonacci de 0."<<endl;
    }
  }while(n!=1);
  return EXIT_SUCCESS;
}
```

Observe que en este ejemplo en el módulo "leerNumero" se valida el dato mediante el tratamiento de errores y la recursividad, para ello el dato se lee dentro de un bloque "try-catch" y si es real o negativo, se genera un error (los errores 1 y 2 respectivamente), entonces el programa salta a la instrucción "catch" donde se muestra el mensaje de error respectivo y luego se vuelve a llamar a la función "leerNumero" (llamada recursiva), proceso que se repite hasta que el dato leído es correcto, es decir un número entero y positivo.

Por supuesto, en este ejemplo, se puede hacer la validación de manera que el número leído sea entero y mayor a 0 (pues no existe el Fibonacci de 0), sin embargo, dado que no existe un tipo entero positivo que comience en 1, el lugar más adecuado para tratar este caso es la función "fibonacci". En esta función se verifica si el número recibido (n) es cero y de ser así se genera un error (el error 3), sin embargo, este error no se trata en esta función, sino en el módulo principal (main), que es desde donde se llama a la función fibonacci.

Observe también que la anterior comprobación no se la hace en la función recursiva (fibonaccir), sino en la función "fibonacci", que es desde donde se llama a la función recursiva. Como se recordará, se procede así para evitar el realizar dicha comprobación en cada llamada recursiva (lo que constituiría un error lógico, pues no se debe volver a comprobar algo que ya ha sido comprobado previamente).

Finalmente, en el módulo principal, se leen los datos en un ciclo "do-while" hasta que el número introducido es 1, sin embargo, dentro del mismo se trata el error generado por la función "fibonacci" (el error número 3) de manera que cuando se produce el control salta al bloque "catch" y muestra el mensaje de error respectivo, continuando luego con la ejecución del siguiente ciclo.

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
c:\HPU\programas\c++\2-2012\tema11>g++ -c ejem1.cpp

c:\HPU\programas\c++\2-2012\tema11>g++ -o ejem1 ejem1.o

c:\HPU\programas\c++\2-2012\tema11>ejem1

***** Calculo recursivo del Fibonacci *****
Introduzca 1 para salir

            Numero? 5
     El fibonacci es: 5

            Numero? 9
     El fibonacci es: 34

            Numero? 20
     El fibonacci es: 6765

            Numero? 0
             Error: No existe el Fibonacci de 0.
            Numero? 7.2
             Error: El numero debe ser entero.
            Numero? -7
             Error: El numero debe ser positivo.
            Numero? 1
     El fibonacci es: 1
```

Como se puede ver (y como se dijo) cuando se introduce el número 0, se muestra el mensaje de error y el programa vuelve a pedir otro número, sin que el programa se interrumpa. Es en este aspecto donde difiere el comando "exit" (que si interrumpe la ejecución del programa) del tratamiento de errores (try-catch).

Como los resultados obtenidos son correctos, se podría concluir que el programa es correcto, sin embargo, si se intenta calcular el Fibonacci de números más grandes, por ejemplo de 50:

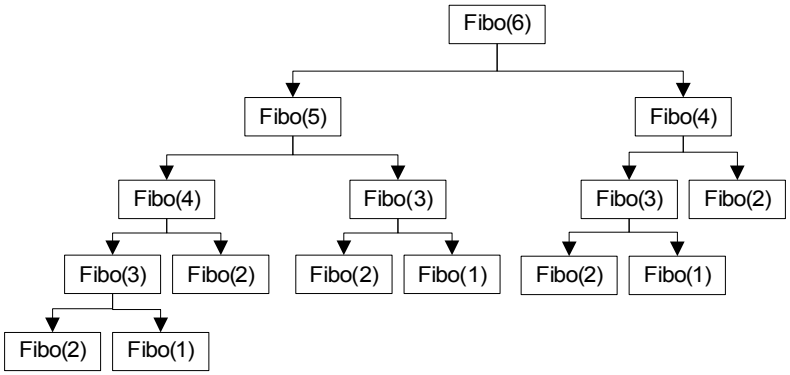
```
c:\HPU\programas\c++\2-2012\tema11>ejem1

***** Calculo recursivo del Fibonacci *****
Introduzca 1 para salir

            Numero? 50
```

No se obtiene una respuesta inmediata y el programa queda como colgado y por lo general es necesario interrumpir manualmente el mismo pulsando las teclas Ctrl+C:

```
            Numero? 50
^C
c:\HPU\programas\c++\2-2012\tema11>
```

¿Por qué no se encuentra este Fibonacci? Lo que ha sucedido es que la solución propuesta es en realidad errónea: tiene un error lógico.

Para comprender el por qué la lógica es errónea se debe analizar el proceso recursivo. Por ejemplo, las llamadas recursivas que se hacen para calcular el Fibonacci de 6 son:

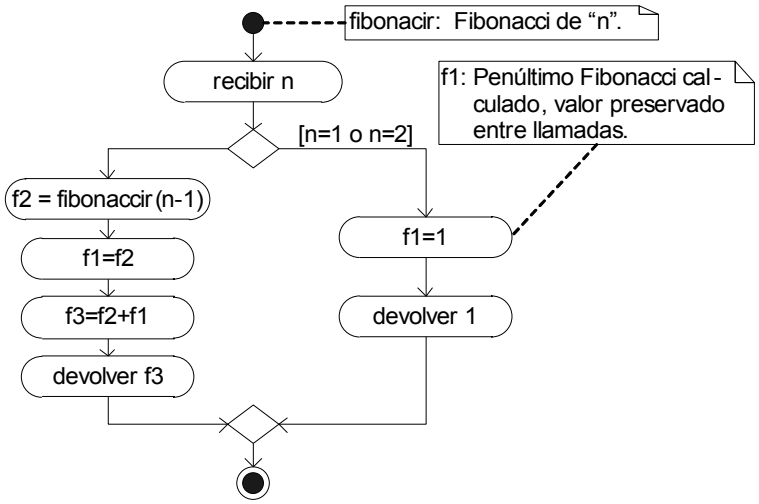


Es decir que en el cálculo del Fibonacci de 6, se calcula 3 veces el Fibonacci de 1, 5 veces el Fibonacci de 2, 3 veces el Fibonacci de 3 y 2 veces el Fibonacci de 4. En consecuencia se realiza un total de 14 llamadas recursivas y en las mismas se vuelven a calcular los mismos valores, lo que constituye un error pues ningún programa, con una lógica correcta, calcula más de una vez el mismo valor.

Este error incrementa geométricamente con el número de Fibonacci a calcular, así en el Fibonacci de 6 se calculan 8 veces los mismos valores, pero en el Fibonacci de 34 se calculan !más de 11 millones de veces los mismos valores! Esta es la razón por la cual no se obtiene ningún resultado inmediato para el Fibonacci de 50, pues en ese caso se están calculando los mismos valores ¡miles de millones de veces!

Este error no se hace evidente con los números pequeños porque los procesadores actuales ejecutan las instrucciones a tal velocidad que la respuesta, para dichos valores, parece inmediata.

Es necesario entonces plantear otra solución de manera que se evite la repetición de cálculos. Una solución posible consiste en hacer que la función se llame a sí misma mientras que el número sea mayor a 2 y emplear luego los resultados, a medida que son devueltos por cada una de las llamadas recursivas, para calcular los nuevos valores, guardando el penúltimo valor en una variable, es decir:



Como se indica en el algoritmo el valor de “f1” debe ser preservado entre llamadas (para guardar el penúltimo valor calculado). Esto se puede conseguir en C/C++ a través de las variables locales estáticas. Alternativamente en C++ se pueden emplear espacios de nombres (namespace), creando uno para

la función y la variable "f1" y empleando entonces dicha variable dentro de la función.

En este ejemplo se emplearán variables locales estáticas, que son variables locales, pero que conservan su valor entre llamadas. Las variables locales estáticas se declaran como cualquier variable local pero precediendo la declaración con la palabra "static".

El código de la función que implementa este cambio es:

```
//Función recursiva para el cálculo del Fibonacci
double fibonaccir(unsigned n){
  static double f1=1,f2,f3;
  if (n==1 || n==2) {
    f1=1;
    return 1;
  }
  f2=fibonaccir(n-1);
  f3=f2+f1;
  f1=f2;
  return f3;
}
```

No se muestran las otras funciones del programa pues no cambian con relación a la solución propuesta previamente. Note que, aun cuando sólo se requiere una variable estática "f1", las tres han sido declaradas en esta forma, esto porque con ello se ahorra tiempo evitando que en cada llamada recursiva se creen nuevas variables.

Haciendo correr el programa con algunos valores de prueba, se obtiene:

```
***** Calculo recursivo del Fibonacci *****
Introduzca 1 para salir

            Numero? 5
    El fibonacci es: 5

            Numero? 20
    El fibonacci es: 6765

            Numero? 50
    El fibonacci es: 1.25863e+10

            Numero? 100
    El fibonacci es: 3.54225e+20

            Numero? 1
    El fibonacci es: 1
```

Y como se puede ver ahora se obtienen las soluciones no sólo para números pequeños, sino también grandes, sin embargo, y como se hace evidente, la lógica se torna mucho más compleja, con lo que se pierde la razón por la cual se emplea la recursividad: resolver los problemas de forma más sencilla. Este es uno de los casos donde, a pesar de que la definición es recursiva, la solución más sencilla es iterativa y esa es la solución por la que se debería optar en un caso práctico.

Para comprender correctamente la lógica involucrada en este caso, es conveniente hacer correr el programa paso a paso. Recuerde que para ello se debe compilar el programa con el interruptor -g:

```
c:\HPV\programas\c++\2-2012\tema11>g++ -g -c ejem1b.cpp

c:\HPV\programas\c++\2-2012\tema11>g++ -g -o ejem1b ejem1b.o

c:\HPV\programas\c++\2-2012\tema11>gdb ejem1b -silent
```

```
Reading symbols from c:\HPU\programas\c++\2-2012\tema11\ejem1b.exe...done.
(gdb)
```

En este caso es conveniente fijar un breakpoint en la función "fibonacci" (la función no recursiva) pues es desde la misma que se hace la llamada recursiva, por lo que el proceso de depuración puede comenzar en la misma:

```
(gdb) break fibonacci
Breakpoint 1 at 0x4016e4: file ejem1b.cpp, line 44.
```

Ahora, se hace correr el programa:

```
(gdb) run
Starting program: c:\HPU\programas\c++\2-2012\tema11\ejem1b.exe
[New Thread 8088.0x7f0]

***** Calculo recursivo del Fibonacci *****
Introduzca 1 para salir

              Numero? 7
```

Con lo que el programa se detiene en la función "fibonacci":

```
Breakpoint 1, fibonacci (n=7) at ejem1b.cpp:44
44          if (n==0) throw 3;
```

Como se puede ver la función recibe el número "n" con un valor igual a 7. Ahora se continua la ejecución del programa paso a paso con la instrucción "step", o lo que es lo mismo con su abreviación "s":

```
(gdb) s
45          return fibonaccir(n);
```

Como era de esperar (al no ser "n" cero) pasa a la siguiente línea donde se hace la llamada a la función recursiva "fibonaccir". Ejecutando esta instrucción se ingresa a dicha función (recuerde que para repetir la última instrucción simplemente se pulsa la tecla Enter):

```
(gdb)
fibonaccir (n=7) at ejem1b.cpp:32
32          if (n==1 || n==2) {
```

Como se puede observar, en esta primera llamada "n" vale 7 y como no es ni 1 ni 2, no se ejecutan las instrucciones de la estructura "if", sino que el programa continua con la instrucción que le continua:

```
(gdb)
36          f2=fibonaccir(n-1);
```

Donde se hace la primera llamada recursiva (cuyo resultado se guarda en la variable "f2"), ejecutando esta instrucción se ingresa a una copia de la función "fibonaccir", pero donde "n" vale ahora 6 (n-1 = 7-1 = 6):

```
(gdb)
fibonaccir (n=6) at ejem1b.cpp:32
32          if (n==1 || n==2) {
```

Y como "n" no es ni 1 ni 2, pasa a la siguiente instrucción, donde se vuelve a llamar a si mismo, continuando de esta forma hasta que "n" es 2:

```
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=5) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=4) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
```

```
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=3) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=2) at ejem1b.cpp:32
32          if (n==1 || n==2) {
```

Ahora, como "n" es 2, se cumple la condición y en consecuencia se ejecutan las instrucciones de la estructura "if":

```
(gdb)
33            f1=1;
(gdb)
34            return 1;
(gdb)
40        }
(gdb)
37          f3=f2+f1;
```

Por lo tanto "f1" toma el valor 1 y la función devuelve el número 1, devolviendo el control a la función que hizo la llamada, donde ahora se puede calcular el valor de "f3" pues "f1" y "f2" ya tienen valores:

```
(gdb)
38          f1=f2;
```

Para ver dichos valores se puede emplear la instrucción "printf", que es muy similar a la función "printf" de C, sólo que no requiere paréntesis:

```
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=3; f1=1; f2=1; f3=2
```

Con lo que se comprueba que "n" vale 3, "f1" 1, "f2" 1 (el resultado devuelto por la función recursiva) y "f3" 2 (el nuevo valor calculado). Ahora se guarda este valor en la variable "f1" y se devuelve a la función que hizo la llamada, donde se calcula el nuevo valor de "f3":

```
(gdb) s
39          return f3;
(gdb)
40        }
(gdb)
37          f3=f2+f1;
(gdb)
38          f1=f2;
```

Una vez más es conveniente analizar los nuevos valores con "printf", por supuesto no es necesario volver a escribir el comando, simplemente se recupera el comando anterior (o los comandos ateriores) pulsando la tecla del cursor hacia arriba:

```
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=4; f1=1; f2=2; f3=3
```

Continuando de esta manera, hasta que "n" es 7, se obtiene:

```
(gdb) s
39          return f3;
(gdb)
40        }
(gdb)
37          f3=f2+f1;
(gdb)
38          f1=f2;
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=5; f1=2; f2=3; f3=5
(gdb) s
39          return f3;
(gdb)
```

```
40      }
(gdb)
37          f3=f2+f1;
(gdb)
38          f1=f2;
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=6; f1=3; f2=5; f3=8
(gdb) s
39          return f3;
(gdb)
40      }
(gdb)
37          f3=f2+f1;
(gdb)
38          f1=f2;
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=7; f1=8; f2=8; f3=13
(gdb)
39          return f3;
(gdb) s
40      }
(gdb)
fibonacci (n=7) at ejem1b.cpp:46
46      }
```

Con lo que la última llamada recursiva devuelve el valor 13 (el valor de la variable "f3") a la función que le llamó, siendo este valor a su vez devuelto por la función no recursiva. Continuando la ejecución del program con "continue" o su abreviación "c", se obtiene:

```
(gdb) c
Continuing.
     El fibonacci es: 13

               Numero?
```

Para comprender el por qué es necesario asignar el valor "1" a la variable "f1" cuando "n" es 1 o 2, se hace correr el programa con otro valor, por ejemplo 6:

```
               Numero? 6

Breakpoint 1, fibonacci (n=6) at ejem1b.cpp:44
44          if (n==0) throw 3;
```

Entonces, haciendo correr paso a paso (con "step" o su abreviación "s") hasta que "n" es 2, se obtiene:

```
(gdb) s
45          return fibonaccir(n);
(gdb)
fibonaccir (n=6) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=5) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=4) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
fibonaccir (n=3) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
36          f2=fibonaccir(n-1);
(gdb)
```

```
fibonaccir (n=2) at ejem1b.cpp:32
32          if (n==1 || n==2) {
(gdb)
33            f1=1;
```

Si ahora se imprimen los valores de las variables:

```
(gdb) printf "n=%d; f1=%.0f; f2=%.0f; f3=%.0f\n",n,f1,f2,f3
n=2; f1=8; f2=8; f3=13
```

Se puede ver que "f1" vale 8, por lo tanto, si no se vuelve a iniciar en 1, el proceso recursivo sólo daría los resultados correctos la primera vez que se ejecuta, pues siempre guarda el penúltimo valor calculado. Esa es la razón por la cual la variable "f1" debe ser iniciada en 1, cada vez que "n" vale 1 o 2. Una vez comprendida la razón, se puede continuar con la ejecución del programa con "continue" (o su abraviación "c"):

```
(gdb) c
Continuing.
     El fibonacci es: 8

               Numero? 1

Breakpoint 1, fibonacci (n=1) at ejem1b.cpp:44
44          if (n==0) throw 3;
(gdb) c
Continuing.
     El fibonacci es: 1

[Inferior 1 (process 8088) exited normally]
```

Por supuesto, la depuración hecha es sólo un ejemplo, en función a las dudas individuales, cada estudiante puede detener el programa donde más lo requiera e imprimir (o mostrar con "display") las variables que le interesen.

Como ya se dijo, el problema puede ser resuelto también (en C++) empleando espacios de nombre, en cuyo caso las modificaciones que se deben hacer en el programa son:

```
//Espacio de nombres donde se declaran las 3 variables
namespace fibo {
  double f1=1,f2,f3;
}

//Uso del espacio de nombres creado
using namespace fibo;

//Función recursiva donde se emplean las variables del espacio fibo
double fibonaccir(unsigned n){
  if (n==1 || n==2) {
    f1=1;
    return 1;
  }
  f2=fibonaccir(n-1);
  f3=f2+f1;
  f1=f2;
  return f3;
}
```

Con el cual, como era de esperar, se obtienen los mismos resultados que con la anterior versión. Se reitera que el uso de espacios de nombre es exclusivo de C++.

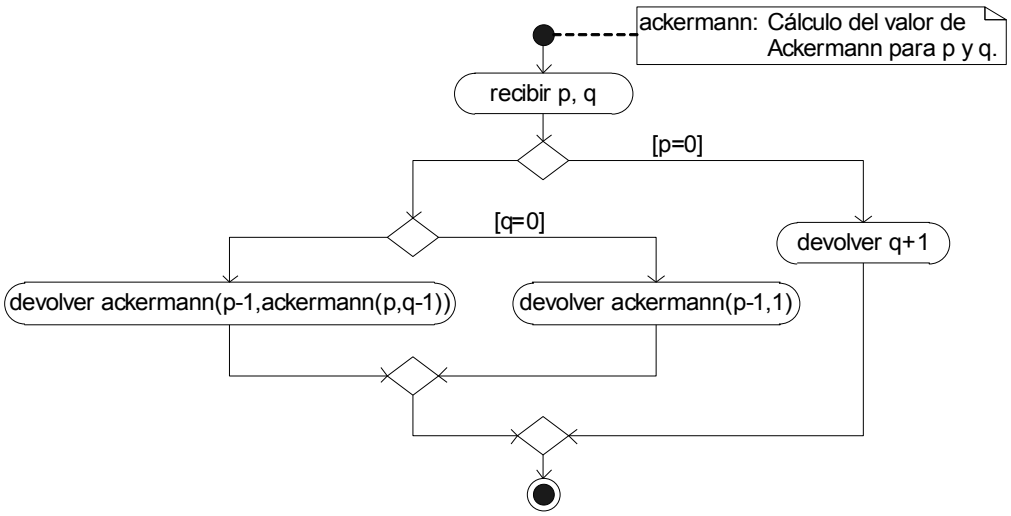
2. Elabore un programa que, empleando librerías tipo C y la recursividad, calcule el valor de la función de Ackermann para dos números enteros dados.

La función de Ackermann está definida por las siguiente expresión:

$$a(p,q)=\begin{cases} q+1 & \text{si p=0} \\ a(p-1,1) & \text{si p<>0 y q=0} \\ a(p-1,a(p,q-1)) & \text{si p<>0 y q<>0} \end{cases}$$

Esta función, al igual que el Fibonacci, es doblemente recursiva y presenta problemas similares, sin embargo, al tener dos parámetros, la lógica se torna más compleja, por lo que el evitar la repetición de cálculos y optimizar el proceso requiere de fórmulas matemáticas aún más complejas, que además no cubren todos los casos.

Por lo tanto, en este caso, la forma más práctica de resolver el problema es aplicar directamente la expresión:



Siendo el código respectivo:

```
c:\HPV\programas\c++\2-2012\tema8>notepad ejem2.cpp
```

```
#include <cstdio>
#include <cmath>
#include <cstdlib>
using namespace std;

unsigned leerNumero(){
  double n;
  try {
    scanf("%lf",&n);
    if (fmod(n,1)) throw 1;
    if (n<0) throw 2;
    return (unsigned)n;
  } catch(int e){
    printf("%20s: ","Error");
    switch(e){
      case 1: printf("El número debe ser entero.\n"); break;
      case 2: printf("El número debe ser positivo.\n"); break;
    }
    return leerNumero();
```

```
  }
}

unsigned ackermann(unsigned p, unsigned q){
  if (p==0)
    return q+1;
  else
    if (q==0)
      return ackermann(p-1,1);
    else
      return ackermann(p-1,ackermann(p,q-1));
}

void mostrarAckermann(double r){
  printf("%20s: %.12g\n\n","Ackerman",r);
}


int main(){
  printf("\n***** Funcion de Ackermann *****\n");
  printf("Introduzca ambos datos como 0 para salir\n\n");
  unsigned p,q,r;
  do{
    printf("%20s? ","Primer parametro");
    p=leerNumero();
    printf("%20s? ","Segundo parametro");
    q=leerNumero();
    r=ackermann(p,q);
    mostrarAckermann(r);
  } while(p!=0 || q!=0);
  return EXIT_SUCCESS;
}
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
c:\HPU\programas\c++\2-2012\tema11>g++ -c ejem2.cpp

c:\HPU\programas\c++\2-2012\tema11>g++ -o ejem2 ejem2.o

c:\HPU\programas\c++\2-2012\tema11>ejem2

***** Funcion de Ackermann *****
Introduzca ambos datos como 0 para salir

    Primer parametro? 1
   Segundo parametro? 3
            Ackerman: 5

    Primer parametro? 3
   Segundo parametro? 1
            Ackerman: 13

    Primer parametro? 3
   Segundo parametro? 3
            Ackerman: 61

    Primer parametro? 3
   Segundo parametro? 4
            Ackerman: 125

    Primer parametro? 4
   Segundo parametro? 1
```

```
          Ackerman: 65533

 Primer parametro? 0
Segundo parametro? 0
          Ackerman: 1
```

Como se puede comprobar, el resultado para Ackermann(4,1) demora bastante en ser calculado. Esto se debe (como en el caso del Fibonacci) a las millones de llamadas recursivas que se realizan y dicho número incrementa tanto con el valor de "p" que no se obtiene un resultado para el Ackermann(5,1) pues las llamadas recursivas son tantas que la memoria se agota.

11.3. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema11" (tanto para los ejemplos como para los ejercicios). No olvide emplear el depurador "gdb" para encontrar errores y comprender mejor la lógica involucrada.

1. La función Bessel de primera clase ("J") y orden "n" puede ser calculada aplicando la siguiente fórmula recursiva:

$$J_{n+1}(x)=\frac{2n}{x}J_n(x)-J_{n-1}(x)$$

Donde los valores de la función para "n=0" y "n=1" pueden ser calculados con las siguientes series:

$$J_0(x)=1-\frac{x^2}{2^2}+\frac{x^4}{2^2\cdot 4^2}-\frac{x^6}{2^2\cdot 4^2\cdot 6^2}+\ldots$$

$$J_1(x)=\frac{x}{2}-\frac{x^3}{2^2\cdot 4}+\frac{x^5}{2^2\cdot 4^2\cdot 6}-\frac{x^7}{2^2\cdot 4^2\cdot 6^2\cdot 8}+\ldots$$

Escriba un programa que, empleando una función recursiva, calcule la función "J" de Bessel de orden "n". Los valores de J_0 y J_1 deben ser calculados en módulos independientes y se debe evitar que en la función recursiva se calcule el mismo valor más de una vez.

2. El cálculo del Chebyshev, tiene una lógica muy similar al del Fibonacci y al de la función de Bessel. Elabore un programa que calcule el Chebyshev enésimo (n) de un número real (x), mediante un módulo recursivo que evite calcular los mismos valores.

$$Ch_n(x)=2\cdot x\cdot Ch_{n-1}(x)-Ch_{n-2}(x)$$
$$Ch_0(x)=1$$
$$Ch_1(x)=x$$

3. El máximo común divisor de dos números "A" y "B" puede ser calculado aplicando el algoritmo de Euclides que se resumen en las siguientes tres ecuaciones:

$$MCD(A,A)=A$$
$$MCD(A,B)=MCD(B,A) \qquad si\ \ A>B$$
$$MCD(A,B)=MCD(A,B-A) \qquad si\ \ A<B$$

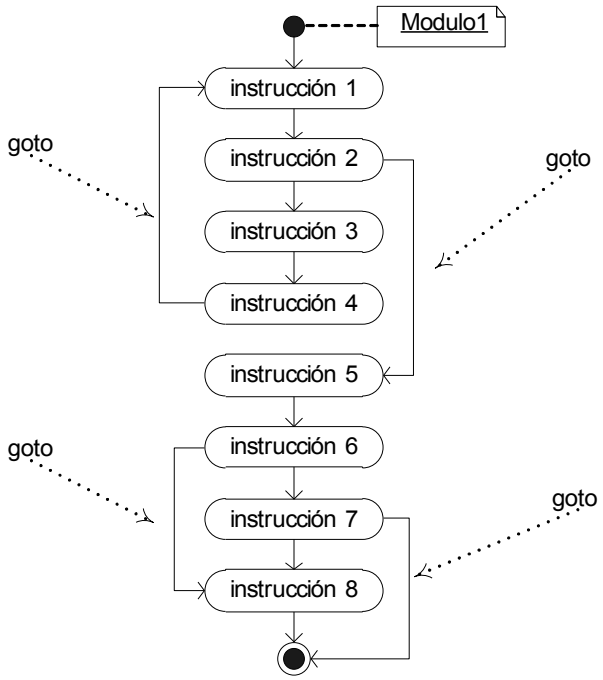
Elabore un programa que, aplicando el algoritmo de Euclides, calcule el máximo común divisor de dos números dados.

12. EL COMANDO GOTO

En C/C++ se puede saltar a cualquier lugar dentro de un programa empleando el comando *goto*. Con este comando no sólo se pueden resolver problemas iterativos, sino que es posible emular (y también romper) cualquiera de las estructuras estándar.

Es justamente esa versatilidad la que puede dar lugar a un código desordenado y es la razón por la que este comando no forma parte de la programación estructurada. No obstante, en ocasiones su uso puede dar lugar a una solución no sólo más eficiente, sino también más clara y es en estas ocasiones (y sólo en esas) donde debe ser empleada.

El comando *goto* no tiene una representación propia en un diagrama de actividades: cualquier salto (no secuencial) hacia adelante o hacia atrás, puede ser codificado con este comando. Así por ejemplo los flujos no secuenciales de la siguiente figura pueden ser interpretados como comandos *goto*:



Para saltar a cualquier lugar dentro de un módulo se requiere crear una etiqueta en dicho lugar. Una etiqueta es un identificador cualquiera (o un número) seguido de dos puntos:

```
identificador:
```

Para saltar a la etiqueta se escribe:

```
goto nombre_de_la_etiqueta;
```

En la práctica, **el comando "goto" debe ser empleado sólo cuando se logra un código más claro**, no obstante, en este tema, y sólo con la finalidad de adquirir práctica, se empleará inclusive cuando con su uso no se logre ninguna ventaja real.

12.1. EJEMPLOS

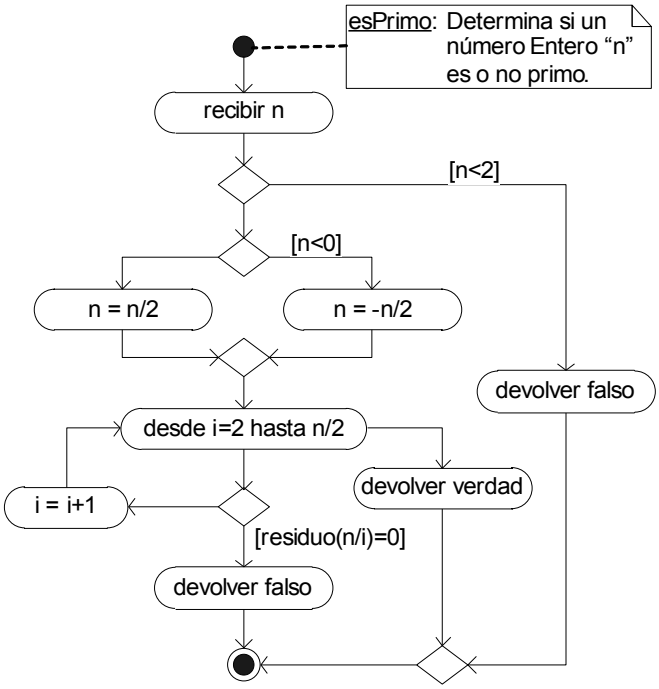
1. Elabore un programa que, empleando el comando goto y librerías tipo C++, determine si un número entero dado es o no primo.

Un número es primo cuando sólo es divisible entre 1 y si mismo. Como ningún número es divisible entre un número mayor a su mitad, se puede afirmar que un número es primo si no es divisible entre ningún número comprendido entre 2 y la mitad del número.

Entonces, para determinar si un número es o no primo, se puede emplear un ciclo "for" que vaya desde 2 hasta "n/2" y en el cual se compruebe si el número es divisible entre alguno de dichos números, sin embargo, si se encuentra un divisor, dicho ciclo debe concluir inmediatamente, porque en ese momento se sabe que el número no es primo y por lo tanto sería innecesario e ilógico buscar otros divisores.

Por ejemplo si el número es 1000000000, el ciclo no debe repetirse ni una vez, pues como es divisible entre 2 (lo que se sabe en el primer ciclo), se puede afirmar que no es primo y en consecuencia no es necesario probar ningún otro divisor. Si una vez determinado que un número no es primo, se continúan probando otros divisores (es decir no se termina el ciclo) se haría un trabajo absurdo, pues se seguiría buscando una respuesta cuando ya se la tiene y si esto sucede con 1000000000 el proceso se repite, innecesariamente, ¡500 millones de veces!

La forma más sencilla de resolver el problema es mediante un ciclo for que vaya desde 2 hasta la mitad del número, pero el cual concluya inmediatamente en caso de encontrarse un divisor. Este planteamiento (algoritmo) en forma de diagrama de actividades es:



Por supuesto, la forma más sencilla y lógica de codificar este algoritmo es mediante un ciclo for, del cual, si se encuentra un divisor, se sale con la instrucción "return", sin embargo, en este caso se empleará el comando goto (se reitera que se hace esto sólo con el propósito de adquirir práctica en el uso de dicho comando).

El código respectivo es:

```
c:\HPV\programas\c++\2-2012\tema12>notepad ejem1.cpp
```

```
#include <iostream>
#include <iomanip>
```

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

long long int leerNumero(){
  double n;
  uno:
    cout<<setw(20)<<"Numero"<<"? ";
    cin>>n;
    if (fmod(n,1)) {
      cout<<setw(20)<<"Error"<<": El numero debe ser entero."<<endl;
      goto uno;
    }
  return (long long int)n;
}

bool esPrimo(long long int n){
  long long int i=2, li;
  bool r=false;
  if (n>-2 && n<2) goto salir;
  li= n<0 ? -n/2 : n/2;
  ciclo:
    if (i++ > li) goto finciclo;
    if (n%i==0) goto salir;
    goto ciclo;
  finciclo: r=true;
  salir: return r;
}

bool mostrarSiPrimo(bool r){
  cout<<setw(20)<<"El numero"<<(r ? "  Es" : "  No es")<<" Primo."<<endl;
}

int main(){
  long long int n; bool r;
  cout<<endl<<"***** Determina si un número es o no primo *****"<<endl;
  cout<<"Introduzca 0 para salir"<<endl<<endl;
  ciclo:
    n=leerNumero();
    r=esPrimo(n);
    mostrarSiPrimo(r);
    if (n!=0) goto ciclo;
  return 0;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
c:\HPU\programas\c++\2-2012\tema12>g++ -c ejem1.cpp

c:\HPU\programas\c++\2-2012\tema12>g++ -o ejem1 ejem1.o

c:\HPU\programas\c++\2-2012\tema12>ejem1

***** Determina si un n·mero es o no primo *****
Introduzca 0 para salir

             Numero? 7
          El numero  Es Primo.
             Numero? 78
```

```
El numero  No es Primo.
   Numero? 113
El numero  Es Primo.
   Numero? 432527
El numero  Es Primo.
   Numero? 578423
El numero  No es Primo.
   Numero? 3.45
    Error: El numero debe ser entero.
   Numero? 0
El numero  No es Primo.
```

2. Elabore un programa que, empleando el comando goto y librerías tipo C, calcule la integral de una función por el método de Simpson.

La ecuación para el cálculo de la integral por el método de Simpson es:

$$\int_a^b f(x)\cdot dx = \frac{h}{3}\left(f(a) + 4\sum_{i=2,4,6}^{n} f(x_i) + 2\sum_{i=3,5,7}^{n-1} f(x_i) + f(b) \right)$$

En este método se obtiene el valor aproximado de la integral dividiendo el área bajo la curva de la función "f" en "n" segmentos de ancho "h":

$$h = \frac{b-a}{n}$$

Los valores sucesivos de x_i son los diferentes valores de la variable independiente en cada uno de los segmentos y se obtienen sumando "h" al valor anterior: $x_i = x_{i-1} + h$, siendo el primer valor el límite inferior: $x_1 = a$.

En teoría, mientras menor sea el ancho de los segmentos "h" (o lo que es lo mismo, mientras mayor sea "n") más exacto es el resultado obtenido, en la práctica no obstante, esto sólo es cierto hasta cierto límite (debido a los errores de redondeo).

Otro aspecto que se debe tomar en cuenta en este método es que el valor de "n", debe ser par.

Para resolver el problema básicamente se debe programar la ecuación y al existir sumatorias, la forma más sencilla de hacerlo es con un ciclo "for" y el algoritmo con dicha estructura se presenta en la siguiente página, sin embargo, y como ya se dijo, en este tema se elabora el código empleando el comando "goto", no obstante, en un caso real debe emplearse la estructura "for" (y no "goto") por ser la más adecuada para resolver este problema en particular.

Para probar el método se calculará el valor de la siguiente integral:

$$\int_a^b f(x)\cdot dx = \int_a^b \left(x^3 + 2\cdot x^2 + 3\cdot x + 4 \right)\cdot dx$$

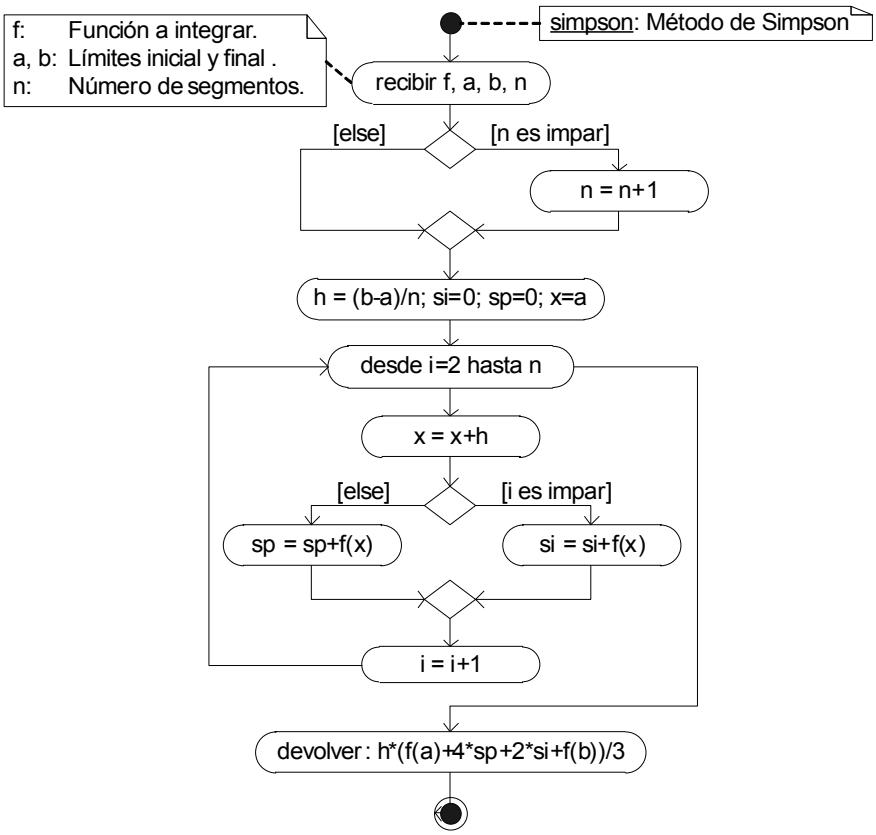
Y para corroborar los resultados son correctos se calculará el valor de la siguiente expresión, que es la solución analítica de la integral:

$$\int_a^b \left(x^3 + 2\cdot x^2 + 3\cdot x + 4 \right)\cdot dx = \left(\frac{x^4}{4} + \frac{2\cdot x^3}{3} + \frac{3\cdot x^2}{2} + 4\cdot x \right)\Bigg|_a^b$$

El código es:

```
c:\HPV\programas\c++\2-2012\tema12>notepad ejem2.cpp
```

```
#include <cstdio>
#include <cmath>
```




```
void leerLimites(double &a, double &b){
  uno:
    printf("%20s? ","Limite inferior");
    scanf("%lf",&a);
    printf("%20s? ","Limite superior");
    scanf("%lf",&b);
    if (a>=b) {
      printf("%20s: %s\n","Error",
        "El limite superior debe ser mayor al inferior.");
      goto uno;
    }
}

unsigned leerDivisiones(){
  double n;
  uno:
    printf("%20s? ","Numero de divisiones");
    scanf("%lf",&n);
    if (fmod(n,1)){
      printf("%20s: %s\n","Error",
        "El numero de divisiones debe ser entero.");
      goto uno;
    }
    if (n<0){
      printf("%20s: %s\n","Error",
        "El numero de divisiones debe ser positivo.");
      goto uno;
    }
```

```
  return (unsigned)n;
}

double simpson(double (*f)(double), double a, double b, unsigned n){
  n= n%2 ? n+1 : n;
  double h=(b-a)/n, si=0, sp=0, x=a;
  unsigned i=2;
  ciclo:
    if (i>n) goto finciclo;
    x+=h;
    if (i%2) si+=f(x); else sp+=f(x);
    i++;
    goto ciclo;
  finciclo: return h*(f(a)+4*sp+2*si+f(b))/3;
}


double ecuacion(double x){
  return x*x*x+2*x*x+3*x+4;
}

double ecuacionIntegrada(double x){
  return x*x*x*x/4+2*x*x*x/3+3*x*x/2+4*x;
}

double integralConocida(double a, double b){
  return ecuacionIntegrada(b)-ecuacionIntegrada(a);
}

void mostrarResultado(double r1, double r2){
  printf("%20s: %.15g\n","Integral calculada",r1);
  printf("%20s: %.15g\n\n","Integral conocida",r2);
}

int main(){
  printf("\n***** Metodo de Simpson *****\n");
  printf("Introduzca un número de divisiones igual a 0 para salir\n\n");
  double a,b,r1,r2;
  unsigned n;
  ciclo:
    leerLimites(a,b);
    n=leerDivisiones();
    if (n==0) goto finciclo;
    r1=simpson(ecuacion,a,b,n);
    r2=integralConocida(a,b);
    mostrarResultado(r1,r2);
    goto ciclo;
  finciclo: printf("Programa Terminado.\n");
  return 0;
}
```

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba se obtiene:

```
c:\HPV\programas\c++\2-2012\tema12>g++ -c ejem2.cpp
```

```
c:\HPV\programas\c++\2-2012\tema12>g++ -o ejem2 ejem2.o

c:\HPV\programas\c++\2-2012\tema12>ejem2

***** Metodo de Simpson *****
Introduzca un numero de divisiones igual a 0 para salir

     Limite inferior? 1
     Limite superior? 20
Numero de divisiones? 10
  Integral calculada: 46006.9166666667
   Integral conocida: 46006.9166666667

     Limite inferior? 1
     Limite superior? 40
Numero de divisiones? 15
  Integral calculada: 685220.25
   Integral conocida: 685220.25

     Limite inferior? 1
     Limite superior? -30
              Error: El limite superior debe ser mayor al inferior.
     Limite inferior? 1
     Limite superior? 30
Numero de divisiones? 10.2
              Error: El numero de divisiones debe ser entero.
Numero de divisiones? -30
              Error: El numero de divisiones debe ser positivo.
Numero de divisiones? 10
  Integral calculada: 221963.583333333
   Integral conocida: 221963.583333333

     Limite inferior? 1
     Limite superior? 2
Numero de divisiones? 0
Programa Terminado.
```

3. Elabore un programa que, empleando el comando goto y librerías tipo C++, calcule la función "j" de Bessel de primera especie y orden "n":

$$j_n(x)=\sum_{k=0}^{\infty}\frac{(-1)^k\cdot\left(\frac{x}{2}\right)^{n+2k}}{k!\cdot(n+k)!}$$

A primera vista esta función parece muy diferente a las series resueltas en temas previos, pero si se desarrolla la sumatoria:

$$j_n(x)=\frac{(-1)^0\cdot\left(\frac{x}{2}\right)^{n+2*0}}{0!\cdot(n+0)!}-\frac{(-1)^1\cdot\left(\frac{x}{2}\right)^{n+2*1}}{1!\cdot(n+1)!}+\frac{(-1)^2\cdot\left(\frac{x}{2}\right)^{n+2*2}}{2!\cdot(n+2)!}-\frac{(-1)^3\cdot\left(\frac{x}{2}\right)^{n+2*3}}{3!\cdot(n+3)!}+\cdots\infty$$

Y se llevan a cabo algunas operaciones:

$$j_n(x)=\frac{\left(\frac{x}{2}\right)^{n}}{(n)!}-\frac{\left(\frac{x}{2}\right)^{n+2}}{1!\cdot(n+1)!}+\frac{\left(\frac{x}{2}\right)^{n+4}}{2!\cdot(n+2)!}-\frac{\left(\frac{x}{2}\right)^{n+6}}{3!\cdot(n+3)!}+\cdots\infty$$

Se puede ver que tiene una forma muy parecida a las series de Taylor, básicamente la única diferencia es que el primer término, al no ser un valor simple, debe ser calculado previamente.

Como en cualquier serie, la principal dificultad radica en determinar la regla para calcular un término en base al anterior. Las operaciones para calcular el segundo, tercer y cuarto términos a partir del primero son:

$$\frac{\left(\frac{x}{2}\right)^{n}}{(n)!}\cdot\frac{-\left(\frac{x}{2}\right)^{2}}{1\cdot(n+1)}=-\frac{\left(\frac{x}{2}\right)^{n+2}}{(1)!\cdot(n+1)!}$$

$$-\frac{\left(\frac{x}{2}\right)^{n+2}}{(1)!\cdot(n+1)!}\cdot\frac{-\left(\frac{x}{2}\right)^{2}}{2\cdot(n+2)}=\frac{-\left(\frac{x}{2}\right)^{n+4}}{(2)!\cdot(n+2)!}$$

$$\frac{\left(\frac{x}{2}\right)^{n+4}}{(2)!\cdot(n+2)!}\cdot\frac{-\left(\frac{x}{2}\right)^{2}}{3\cdot(n+3)}=-\frac{\left(\frac{x}{2}\right)^{n+6}}{(3)!\cdot(n+3)!}$$

De donde se deduce que para calcular un nuevo término se debe multiplicar el término anterior por:

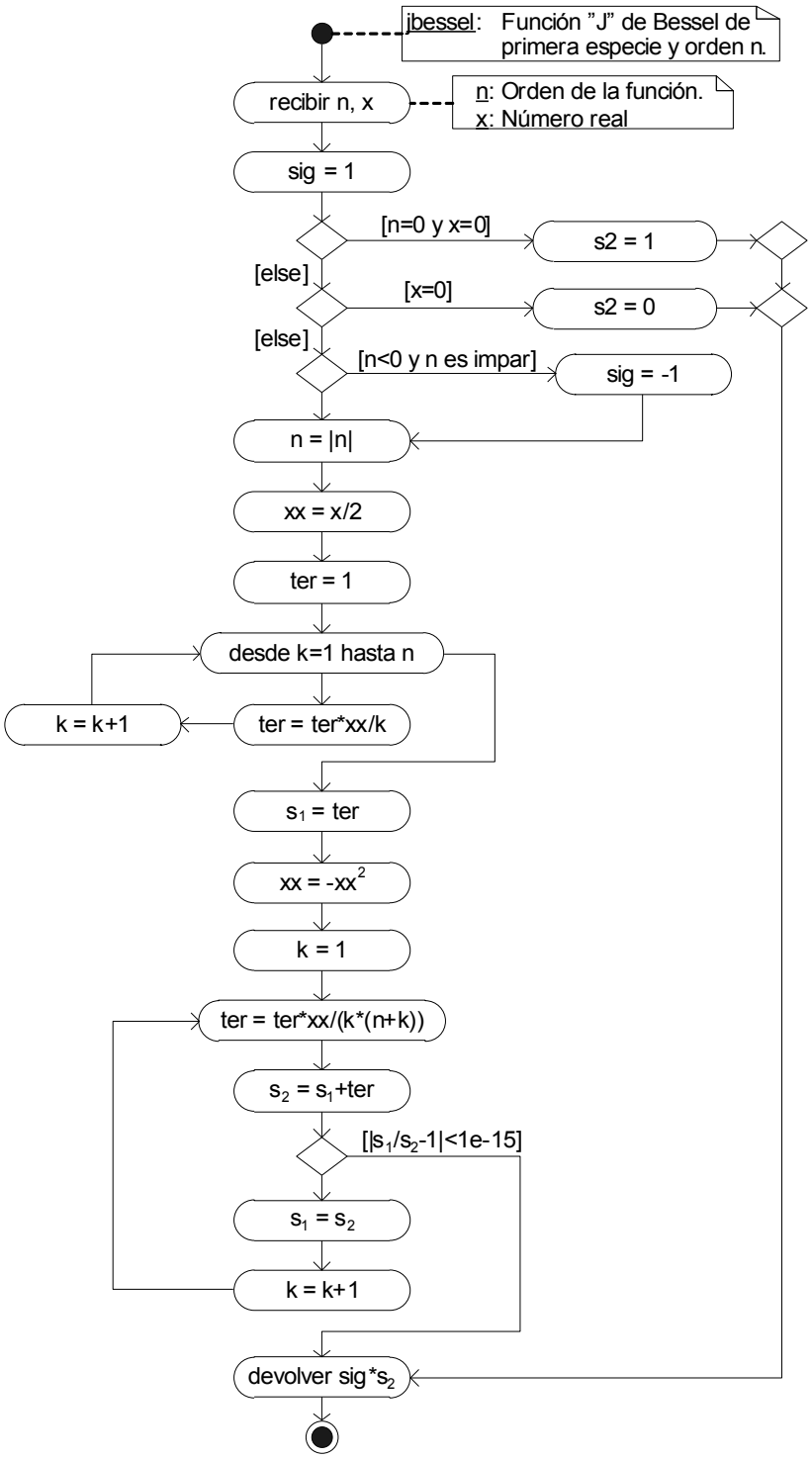
$$\frac{-\left(\frac{x}{2}\right)^{2}}{k\cdot(n+k)}$$

Donde "k" es contador, que comienza en 1, y que se incrementa de uno en uno.

Tomando en cuenta la regla deducida y considerando que el primer término tiene que ser calculado antes de comenzar el cálculo de la serie, se ha elaborado el algoritmo que se presenta en la siguiente página, siendo el código respectivo:

```
c:\HPU\programas\c++\2-2012\tema12>notepad ejem3.cpp
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

double leerNumero(){
  double x;
  cout<<setw(20)<<"Numero real"<<"? ";
  cin>>x;
  return x;
}

int leerOrden(){
  double n;
  uno:
    cout<<setw(20)<<"Orden"<<"? ";
    cin>>n;
    if (fmod(n,1)){
      cout<<setw(20)<<"Error"<<": El orden tiene que ser entero."<<endl;
      goto uno;
    }
  return (int)n;
}

double jbessel(int n, double x){
```




```
double ter,s1,s2,xx;
int k=1,sig=1;
if (n==0 && x==0) {s2=1; goto salir;}
if (x==0) {s2=0; goto salir;}
if (n<0 && n%2) sig=-1;
n=abs(n);
xx=x/2;
ter=1;
```

```
    if (k>n) goto finciclo1;
    ter*=xx/k;
    k++;
    goto ciclo1;
  finciclo1: s1=ter;
  xx=-xx*xx;
  k=1;
  ciclo2:
    ter*=xx/(k*(n+k));
    s2=s1+ter;
    if (abs(s1/s2-1)<1e-15) goto salir;
    s1=s2;
    k++;
    goto ciclo2;
  salir: return sig*s2;
}

void mostrarResultado(double j){
  cout<<setw(20)<<"Valor de la funcion"<<": "<<
    setprecision(16)<<j<<endl<<endl;
}

int main(){
  cout<<endl<<"***** Funcion J de Bessel *****"<<endl;
  cout<<"Introduzca orden y numero iguales a 0 para salir"<<endl<<endl;
  double x,j;
  int n;
  ciclo:
    n=leerOrden();
    x=leerNumero();
    j=jbessel(n,x);
    mostrarResultado(j);
    if (n!=0 || x!=0) goto ciclo;
  cout<<"Programa terminado."<<endl;
  return 0;
}
```

Como en los casos anteriores, el primer ciclo de la función Bessel puede ser programado con la estructura "for" y el segundo con un ciclo infinito. Se reitera que en estos ejemplos (y en este tema) se emplea el comando "goto" sólo con el propósito de adquirir práctica con el mismo.

Compilando, creando el ejecutable y haciendo correr el programa con algunos valores de prueba, se obtiene:

```
c:\HPV\programas\c++\2-2012\tema12>g++ -c ejem3.cpp

c:\HPV\programas\c++\2-2012\tema12>g++ -o ejem3 ejem3.o

c:\HPV\programas\c++\2-2012\tema12>ejem3

***** Funcion J de Bessel *****
Introduzca orden y numero iguales a 0 para salir

              Orden? 0
        Numero real? 5.78
 Valor de la funcion: 0.08545331724981572

              Orden? 5
        Numero real? -3.23
```

```
Valor de la funcion: -0.05841990974577606

               Orden? 12
        Numero real? 7.28
Valor de la funcion: 0.00391263526763217

               Orden? 3
        Numero real? 0
Valor de la funcion: 0

               Orden? 3.2
               Error: El orden tiene que ser entero.
               Orden? 3
        Numero real? 2.2
Valor de la funcion: 0.1623254728332875

               Orden? 0
        Numero real? 0
Valor de la funcion: 1

Programa terminado.
```

12.2. EJERCICIOS

En todos los ejercicios debe escribir los programas en "notepad", compilarlos y enlazarlos (crear el ejecutable) con el comando g++ y hacerlos correr desde la ventana de comandos. Los archivos (programa fuente, objeto y ejecutable) deben ser creados, compilados y ejecutados en el directorio "tp_tema12" (tanto para los ejemplos como para los ejercicios). No olvide emplear el depurador "gdb" para encontrar errores y comprender mejor la lógica involucrada.

1. Elabore un programa que, empleando el comando "goto" y librerías tipo C, determine si un número es o no perfecto. Un número es perfecto si la suma de sus divisores (sin incluir el propio número), es igual al número. Por ejemplo el número 6 es perfecto porque la suma de sus divisores (1+2+3) es igual a 6.
2. Un método más sencillo, aunque menos exacto, para calcular la integral numérica de una función es el método del trapecio:

$$\int_a^b f(x)\cdot dx = \frac{h}{2}\left(f(x_1) + 2\sum_{i=2}^{n} f(x_i) + f(x_{n+1}) \right)$$

$$h = \frac{b-a}{n}$$

$$x_1 = a\,;\ x_2 = x_1 + h\,;\ x_3 = x_2 + h\,;\ \cdots\ x_i = x_{i-1} + h\,;\ x_{n+1} = b$$

A diferencia del método de Simpson, en este método el número de divisiones "n" puede ser par o impar (por lo que no es necesario verificar aspecto).
Elabore y pruebe un programa que, empleando el comando goto y librerías tipo C++, encuentre la integral numérica de una función por este método. Pruebe el método con la siguiente función:

$$\int_a^b \left(\frac{3\cdot x^2 + 2\cdot x - 4}{5\cdot x^3 - 4x^2} \right) \cdot dx$$

Los resultados obtenidos deben ser verificados con el resultado obtenido a partir de la integral analítica de la ecuación.

3. Elabore y pruebe un programa que, empleando el comando goto y librerías tipo C, calcule la función "I" de Bessel de orden "n" para un número real "x". La ecuación de definición de dicha función es la siguiente:

$$I_n(x)=\sum_{k=0}^{\infty}\frac{\left(\frac{x}{2}\right)^{n+2k}}{k!\cdot(n+k)!}$$

Recuerde desarrollar esta función (igual que se hizo con la función J de Bessel) para deducir la regla que permite generar los términos de la serie.

13. MATRICES 1

En este tema se da inicio al estudio de los datos estructurados, ***un dato es estructurado si está conformado por dos o más datos simples o estructurados.*** En ese sentido las matrices son datos estructurados porque están conformadas por dos o más datos del mismo tipo.

Para hacer referencia a una matriz, se emplearán indistintamente los términos *matriz* o *array*. Para hacer referencia a los vectores se emplearán igualmente dichos términos o el término "vector", pues, como se sabe, un vector es simplemente una matriz que tiene una sola fila (o una sola columna).

Desde otro punto de vista, se puede considerar que las matrices son vectores cuyos elementos son a su vez vectores (las filas). La mayoría de los lenguajes de programación (incluido C++) implementan las matrices de acuerdo a este ultimo punto de vista.

13.1. DECLARACIÓN Y USO DE MATRICES

En C++ existen dos tipos de matrices: las matrices estáticas y las matrices dinámicas. La diferencia fundamental es que las matrices estáticas tienen un número fijo de elementos, mientras que las dinámicas fijan y/o pueden cambiar el número de elementos durante la ejecución del programa.

Las matrices estáticas se declaran igual que cualquier otra variable pero especificando (entre corchetes) el número de elementos, por ejemplo las siguientes declaraciones crean un vector con 20 elementos de tipo entero y una matriz con 5 filas y 4 columnas de elementos de tipo doble:

```
int x[20];
double a[4][5];
```

También es posible declarar y asignar al mismo tiempo, valores a las matrices:

```
float y[5] = {1.1,2.2,3.3,4.4,5.5};
int b[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

Se pueden asignar también sólo algunos de los valores, así en el siguiente ejemplo se declara un vector con 12 elementos pero sólo se asigna 4 de ellos:

```
string z[12] = {"enero","febrero","marzo","abril"};
```

Pero no se puede asignar un mayor número de valores que el número de elementos declarado. Por ejemplo el siguiente código generaría un error:

```
char c[5] = {'A','B','C','D','E','F','G'};//Valores iniciales en exceso
```

Es posible igualmente declarar e iniciar un vector sin especificar el número de elementos:

```
short int r[] = {1,2,3,4,5,6,7};
```

Pero en el caso de las matrices se debe especificar el número de elementos de la segunda dimensión (el número de columnas) y posteriores si hubieran:

```
short int d[][4] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

Esto se debe a que las matrices estáticas (de dos o más dimensiones) son en realidad vectores, no existe un carácter o indicador que separe una fila de otra, por eso C++ necesita saber cuántos elemento existen en cada colum-

na para realizar las operaciones respectivas.

Para utilizar los elementos de una matriz, simplemente se escribe el nombre de la matriz y el índice (número) que identifica al elemento entre corchetes. Por ejemplo, en el siguiente segmento de código se guardan dos números reales en los dos primeros elementos del vector, se calcula el cuadrado del primer elemento y se guarda en el tercer elemento, luego se calcula la raíz cuadrada de la suma de los dos primeros elementos y se guarda en el cuarto, finalmente se muestran los valores calculados:

```
double x[4];
x[0]=2.1;
x[1]=3.4;
x[2]=x[0]*x[0];
x[3]=sqrt(x[0]+x[1]);
cout<<setw(20)<<"x[2] = "<<setprecision(4)<<x[2]<<endl;
cout<<setw(20)<<"x[3] = "<<setprecision(4)<<x[3]<<endl;
```

Es importante recordar que en C/C++ el primer elemento de un vector es siempre el elemento número 0.

13.2. PUNTEROS

El correcto entendimiento y uso de punteros es crítico para la exitosa programación en C/C++. Los punteros están estrechamente ligados a los vectores, de hecho un puntero puede ser tratado como un vector y un vector (declarado como parámetro de una función) puede ser tratado como un puntero.

Ahora bien, **un puntero es una variable que almacena una dirección de memoria**, es decir que un puntero no guarda la información en sí, sino el lugar donde se encuentra dicha información.

El tipo de dato que por lo general se asocia a un puntero, simplemente informa con relación al tipo de dato que se espera en la dirección de memoria del puntero, pero no es el tipo de dato que se guarda en el puntero. Como ya se dijo un puntero sólo guarda una dirección de memoria, no un tipo de dato en sí, por esta razón, sin importar el tipo de dato al que esté asociado un puntero, siempre ocupa la misma cantidad de memoria (32 o 64 bits, en función de la arquitectura de la computadora).

Nada garantiza que en la dirección de memoria de un puntero se encuentre efectivamente el tipo de dato al que se encuentra asociado, de hecho es posible declarar punteros que no estén asociados a ningún tipo de dato (punteros puros). Por eso se debe tener mucho cuidado cuando se trabaja con punteros pues el escribir datos de un tipo en una dirección de memoria donde se esperan datos de otro tipo (o no se esperan datos), causa por lo general el colapso del sistema operativo.

Para declarar una variable puntero se emplea la sintaxis:

```
tipo_de_dato *nombre_de_la_variable;
```

Donde el "*" identifica a la variable como una variable puntero. Una declarada la variable puntero, se le puede asignar la dirección de otra variable, obteniendo dicha dirección con el operador "&". Este operador obtiene la dirección de memoria de la variable a la que precede, así por ejemplo, en el siguiente segmento de código se asigna la dirección de memoria de la variable "x" a la variable puntero "p":

```
int x=467;
int *p;
p = &x;
```

Ahora, se puede acceder al número 467 tanto con la variable "x", como con la variable "p", pero para acceder al valor al que apunta un puntero se debe preceder el mismo con el operador "*", así "*p" es el número 467, mientras que "p" (sin el asterisco) es la dirección de memoria donde se encuentra este número.

Por lo tanto, ahora "x" y "*p" son la misma variable (ambos hacen referencia al mismo número), en consecuencia, cualquier modificación a "*p" es una modificación a "x" y viceversa, así si a "*p" se le suma 3 y luego se muestra el valor de "x":

```
*p+=3;
cout<<"x = "<<x<<endl;
```

Se obtiene:

```
x = 470
```

Si ahora se le suma 10 a "x" y se imprime "*p":

```
x+=10;
cout<<"*p= "<<*p<<endl;
```

Se obtiene:

```
*p= 480
```

Con lo que se comprueba que cualquier modificación a "*p" es una modificación a "x" y viceversa.

Es importante tener clara la diferencia entre la variable puntero "p" y el valor al que apunta la variable puntero "*p", así, si se imprime el valor de "p":

```
cout<<p<<endl;
```

Se obtiene:

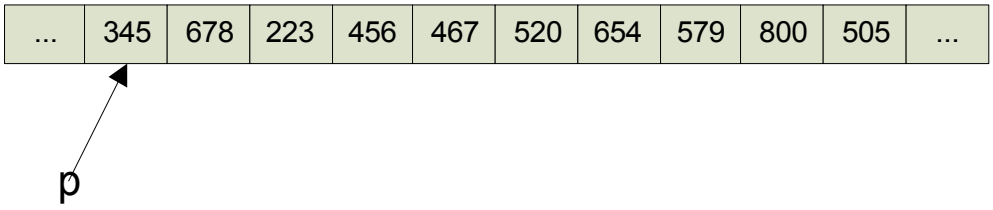
```
0x28ff44
```

Que es una dirección de memoria (mostrada como un número hexadecimal) y no el valor al que apunta (que ahora es el número 480).

13.2.1. Aritmética de punteros

Existen dos operaciones aritméticas que se pueden hacer con punteros: la suma y la resta. Cuando se suma (o resta) un número a un puntero, no se suma (o resta), como se podría pensar, dicho número a la dirección de memoria, sino que la dirección de memoria incrementa (o disminuye) en ese número de posiciones, en función al dato al que está asociado.

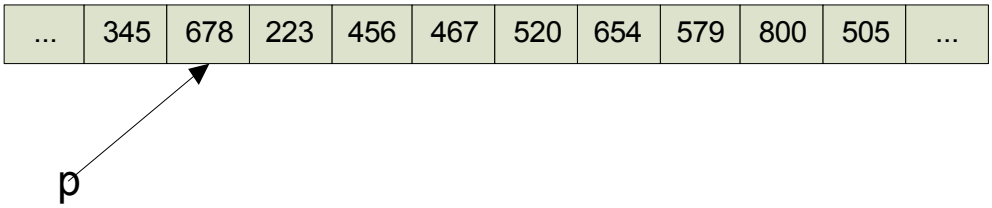
Por ejemplo si se tiene la variable "p", apuntando al número entero 345, y la memoria se encuentra ocupada por números enteros como se muestra en la figura:



Y se incrementa su valor en 1:

```
p++;
```

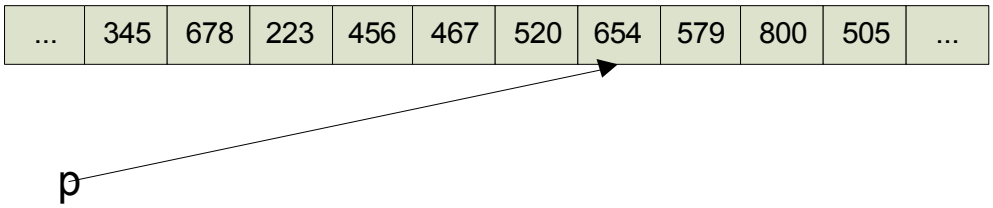
Incrementa una posición de memoria, de manera que queda apuntando al segundo número entero, es decir a 678:



Si ahora se incrementa el valor de "p" en 5:

```
p+=5;
```

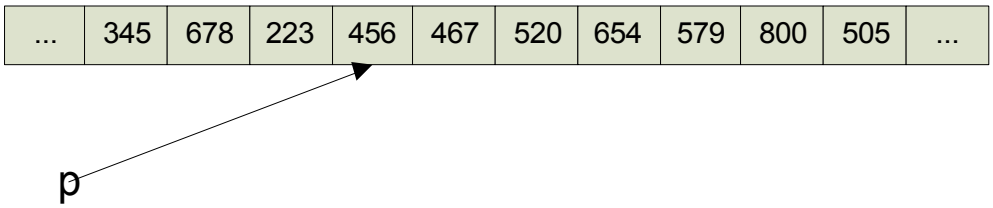
Incrementa 5 posiciones de memoria más, de manera que queda apuntando al número 654:



Finalmente, si se le resta 3:

```
p-=3;
```

Disminuye 3 posiciones de manera y queda apuntando al número 456:



Si en este momento se imprimiera el valor al que apunta el puntero "*p", se obtendría el número 456.

13.2.2. Asignación dinámica de memoria (variables dinámicas)

En la mayoría de las aplicaciones prácticas la dirección de memoria que se le asigna a un puntero no es la dirección de otra variable (como se ha visto en los ejemplos), sino la dirección de un nuevo espacio de memoria que se reserva para almacenar información en el mismo.

En C, dicha memoria se reserva con las funciones "malloc":

```
puntero=(tipo_de_dato* )malloc(número_de_bytes)
```

La cual reserva un espacio de memoria suficiente como para guardar el "número_de_bytes" especificado, devolviendo la dirección de memoria de dicho espacio como un puntero sin tipo, por lo que debe ser convertido al tipo adecuado con (tipo_de_dato *). La memoria puede ser reservada también con "calloc":

```
puntero=(tipo_de_dato* )calloc(número_de_elementos, bytes_de_un_elemento)
```

Para obtener el tamaño (el número de bytes) de un elemento se puede emplear la función "sizeof" en una de las siguientes formas:

```
sizeof(tipo_de_dato o variable)
```

```
sizeof variable
```

Para cambiar el espacio de memoria reservada (ampliándola o reduciéndola) se emplea la función "realloc", la cual devuelve un puntero al nuevo espacio de memoria.

```
nuevo_puntero=realloc(puntero, nuevo_número_de_bytes)
```

Finalmente, el espacio reservado se libera (vuelve a estar disponible) con la función "free":

```
free(puntero)
```

Por ejemplo, en el siguiente segmento de código se declaran tres punteros, dos de tipo real y uno de tipo entero. Para los punteros reales se reserva memoria para un número, mientras que para el puntero entero se reserva memoria para cuatro. Se asignan valores (empleando aritmética de punteros para los números enteros) y se calcula la raíz cuadrada de los números enteros más dos veces el número real, guardando el resultado en la dirección reservada para el segundo puntero real. Finalmente se muestra el resultado y se libera la memoria reservada.

```
double *x, *y;
int *k;
x=(double *)malloc(sizeof(double));
y=(double *)malloc(sizeof(double));
k=(int *)calloc(4,sizeof(int));
*x=3.45;
*k=12; *(k+1)=14; *(k+2)=16; *(k+3)=18;
*y=sqrt(*k+*(k+1)+*(k+2)+*(k+3))+2*(*x);
cout<<setw(20)<<*y<<endl;
free(x); free(y); free(k);
```

Siendo el resultado:

```
14.646
```

En C++, el manejo de la memoria dinámica se la hace con "new" y "delete". La memoria se reserva con "new", empleando una de las siguientes formas:

```
puntero = new tipo_de_dato;
puntero = new tipo_de_dato(valor_inicial);
puntero = new tipo_de_dato[número_de_elementos];
```

A diferencia de "malloc" y "calloc", "new" devuelve un puntero del tipo especificado, por lo que no es necesario hacer el moldeo de tipos. La segunda forma, además de reservar la memoria, asigna el "valor_inicial" a dicho espacio. Las últimas forma reserva memoria suficiente para almacenar "n" elementos del tipo especificado.

La memoria reservada con "new" debe ser liberada con "delete", empleando una de las siguientes formas:

```
delete puntero;
delete []puntero;
```

La primera forma se emplea cuando se ha reservado memoria para un sólo elemento, mientras que la segunda cuando se ha reservado memoria para 2 o más datos.

Así, el siguiente segmento de código hace lo mismo (y produce el mismo resultado) que el ejemplo anterior, pero empleando "new" y "delete" en lugar de "malloc", "calloc" y "free":

```
double *x, *y;
int *k;
x=new double;
y=new double;
k=new int[4];
*x=3.45;
*k=12; *(k+1)=14; *(k+2)=16; *(k+3)=18;
*y=sqrt(*k+*(k+1)+*(k+2)+*(k+3))+2*(*x);
cout<<setw(20)<<*y<<endl;
delete x; delete y; delete []k;
```

En general es más seguro reservar y liberar memoria con las funciones "new" y "delete", porque por un lado se evita el moldeo de tipos (al devolver un puntero del tipo adecuado), por otro se reserva la cantidad de memoria correcta en función al tipo de dato proporcionado (no se requiere de "sizeof") y además se reserva memoria de la pila, el cuál depende sólo de la memoria física disponible en el equipo, no del montículo (como ocurre con "malloc") que es la memoria libre asignada al programa.

13.2.3. Algunos errores comunes con punteros

Los punteros permiten gran libertad al programador, pero al mismo tiempo dejan bajo su responsabilidad labores que, con variables normales, estarían a cargo del compilador. Por eso, cuando se trabaja con punteros, es muy fácil cometer errores, los cuales no sólo producen resultados inesperados, sino que además pueden provocar el colapso del sistema operativo.

Uno de los errores que más frecuentemente se comete es el uso de un puntero no inicializado, es decir, un puntero al que no se le ha asignado una dirección de memoria. Por ejemplo el siguiente segmento de código:

```
float z, *p;
z=10.234;
*p=z;
```

Es erróneo y sin embargo no genera ningún error al momento de compilarlo. Es erróneo porque al puntero "p" no se le ha asignado una dirección de memoria, por lo tanto al hacer la asignación "*p=z", el valor "z" se escribe en una dirección de memoria desconocida. Si el código se ejecuta sólo un par de veces, es probable que el error pase inadvertido, pero si se ejecuta frecuentemente, en algún momento el valor será escrito en un sector de memoria ocupado por el programa o por el sistema operativo, causando que el programa y/o el sistema operativo se cuelguen.

Otro error frecuente se comete cuando se olvida que la aritmética de punteros cambia la dirección del puntero. Por ejemplo, el siguiente segmento de código, calcula correctamente la sumatoria de los números del vector, pero no su productoria:

```
int s,r,*p,x[10]={1,2,3,4,5,6,7,8,9,10};
s=suma(x);
cout<<"Sumatoria: "<<s<<endl;
p=x;
s=*p;
while(*p!=10) {
  p++;
  s=s+(*p);
} while(*p!=10);
cout<<"Sumatoria: "<<s<<endl;
r=*p;
while (*p!=10) {
```

```
    p++;
    r=r*(*p);}
  cout<<"Productoria: "<<r<<endl;
```

Antes de ingresar a la estructura "while", el puntero "p" apunta al primer elemento del vector (el número 1), el cual se asigna a la variable "s", luego, en el primer ciclo se incrementa el puntero (con p++) de manera que apunta al segundo elemento (el número 2) el cual se suma a la variable "s" (s=s+(*p)), entonces en el siguiente ciclo se vuelve a incrementar el puntero (p++) de manera que apunta al tercer elemento, el cual se suma también a la variable "s", continuando de esta forma hasta que el puntero apunta al décimo elemento (el número 10).

De esa manera se obtiene, correctamente, la sumatoria, sin embargo, al calcular la productoria, el puntero ya está apuntando el último elemento del (el número 10), pues esa es la condición de finalización del primer ciclo, por lo que el segundo ciclo no se ejecuta ni una vez, devolviendo como resultado de la productoria el número 10.

Para obtener el resultado correcto es necesario reasignar al puntero, antes de comenzar el cálculo de la productoria, la dirección del primer elemento, es decir hacer la asignación:

```
p=x;
```

Si la condición del ciclo fuera otra (por ejemplo *p<10) el anterior error puede dar lugar a un ciclo infinito (dependiendo de los valores aleatorios que se encuentren en memoria después del último elemento del vector).

Como se puede observar en estos ejemplos, cuando se trabaja con punteros, es muy fácil cometer errores que, aparte de no ser detectados por el compilador, tienen efectos que van desde pasar inadvertidos, hasta bloquear el sistema operativo, por eso es necesario tener especial cuidado cuando se escribe código que involucra punteros.

13.2.4. El puntero genérico

El puntero genérico es un puntero puro, pues sólo tiene una dirección de memoria. No está asociado a ningún tipo de dato, por lo que puede recibir cualquier otro tipo de puntero, esta propiedad lo hace particularmente útil en funciones de carácter general.

Un puntero genérico se declara como un puntero "void":

```
void *nombre_del_puntero;
```

Por ejemplo, el siguiente segmento de código crea e inicializa dos punteros de tipo "char" e "int" que son asignados e impresos empleando un puntero genérico:

```
char *c;
int *i;
void *p;
c = new char('A');
i = new int(92);
p=c;
cout<<*((char *)p)<<endl;
p=i;
cout<<*((int *)p)<<endl;
```

Observe que para imprimir el valor al que apunta el puntero genérico debe ser moldeado a un puntero con tipo (char* e int*), luego se accede a su valor con el operador "*" (siendo esta la razón de los dos paréntesis).

13.3. OPERACIONES FRECUENTES CON MATRICES

Puesto que las matrices son datos estructurados algunas de las operaciones que se dan por sentadas cuando se trabaja con datos simples, constituyen un proceso cuando se trabaja con datos estructurados.

Por ejemplo para mostrar los elementos de un vector no se puede emplear directamente una instrucción sino que se debe mostrar elemento a elemento en un ciclo repetitivo. Así, si se trata de un vector de números enteros, pueden ser mostrados con la siguiente función:

```
void mostrarVectInt(int x[], int n){
  int c=0;
  for (int i=0;i<n;i++) {
    cout<<setw(8)<<x[i];
    c+=8;
    if (c>=72) {cout<<endl; c=0;}
  }
  cout<<endl;
}
```

Que al hacer correr con un vector como el siguiente:

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28};
mostrarVectInt(x,28);
```

Produce:

```
   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      26      27
  28
```

En esta función a cada número se le asigna un ancho de 8 caracteres. El número de caracteres impresos se acumula en la variable "c" y si llega (o supera) los 72, se ínserta una nueva línea y se reinicia el acumulador "c" en 0. Por supuesto, el ancho de cada número (8), así como el ancho de cada fila (72) pueden ser cambiados según convenga.

Igualmente, para leer los elementos de una matriz se debe introducir, uno a uno, los elementos en un ciclo repetitivo. Por ejemplo, la siguiente función permite leer "n" elementos de un vector de números reales:

```
double *leerVectDouble(unsigned n) {
  double *x;
  x=new double[n];
  for (unsigned i=0;i<n;i++){
    cout<<setw(6)<<"x["<<i<<"]? ";
    cin>>x[i];
  }
  return x;
}
```

Sin embargo, en ocasiones se requieren vectores con cientos, miles e inclusive millones de elementos, por supuesto, el introducir cientos o miles de datos manualmente no es posible en la práctica, más aún si se considera que al introducirlos se pueden cometer errores. En dichos casos la mejor alternativa consiste en generarlos al azar o, si se trata de información precisa, importarlos desde un archivo (algo que se verá posteriormente).

Para generar una matriz de números aleatorios, se deben generar elementos aleatorios y para ese fin la mayoría de los lenguajes cuentan con una función que devuelve un número aleatorio (o pseudo-aleatorio) comprendido entre ciertos límites.

En el caso de C++ se tiene la función "rand()" (librería "cstdlib") que genera un número entero, pseudo-aleatorio, comprendido entre 0 y RAND_MAX. Este número puede ser convertido en un número entero comprendido entre dos límites dados ("li"=límite inferior y "ls"=límite superior) con la siguiente expresión:

```
(rand()*(ls-li)/RAND_MAX+li;
```

O en un número real aleatorio, con la expresión:

```
(double)rand()/RAND_MAX*(ls-li)+li;
```

Como ya se dijo, la función "rand()" genera por defecto números pseudoaleatorios, es decir que en realidad genera siempre la misma serie de números. Para que C++ genere números realmente aleatorios se debe cambiar el valor de la semilla con "srand()", siendo un buen valor el tiempo del sistema (pues siempre está cambiando). El tiempo del sistema se obtiene con la función "time()" (de la librería "ctime"):

```
time_t t; time(&t); srand(t);
```

En otras ocasiones (como cuando se compara la eficiencia de dos métodos) resultan de mayor utilidad los números pseudo-aleatorios, es decir una secuencia de números que siempre se repite. En esos casos es conveniente iniciar la semilla en un número fijo, generalmente 0:

```
srand(0);
```

Ya sea que se genere una matriz aleatoria o pseudoaletoria, el proceso para generar números es el mismo: recorrer la matriz (empleando una estructura iterativa) para cada uno de sus elementos, generando y guardando un número aleatorio (o pseudoaleatorio) en cada uno de ellos. Por ejemplo, la siguiente función genera un vectore de "n" números aleatorios comprendidos entre 1 y 200:

```
int *genVectInt(unsigned n){
  int *x;
  x=new int[n];
  for (unsigned i=0;i<n;i++)
    x[i]=rand()*199/RAND_MAX+1;
  return x;
}
```

Que al ser llamada desde el siguiente segmento de código:

```
int *x,*y;
time_t t;
time(&t);
srand(t);
x=genVectInt(14);
mostrarVectInt(x,14);
y=genVectInt(18);
mostrarVectInt(y,18);
```

Genera los siguientes vectores de números aleatorios:

```
  81   180    54    26   117   167    84   103   144
  24   175    57   193   159
  64    64    82   193   190     5    48    63    99
 103   168   187    51    96   169    87   118   178
```

Otra de las operaciones que es directa con datos de tipo simple, pero que requiere un proceso con datos estructurados, es la copia de un vector en otro. Al igual que en los casos anteriores, para copiar los elementos de un vector en otro se requiere de una estructura repetitiva, donde en cada repetición se copia un elemento de un vector al otro. Por ejemplo, la siguiente función copia "n" elementos del vector de números reales "x" en el vector de números reales "y":

```
void copiarVector(int *x, int *y, unsigned n){
  for (unsigned i=0;i<n;i++)
    y[i]=x[i];
}
```

Que al ser llamada desde el siguiente segmento de código:

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
int y[20];
copiarVector(x,y,20);
mostrarVectInt(y,20);
```

Produce el resultado:

```
   1       2       3       4       5       6       7       8       9
  10      11      12      13      14      15      16      17      18
  19      20
```

Si bien las operaciones estudiadas no son complicadas no son tan directas como con los datos simples, pero además, las funciones elaboradas no tienen carácter general, por ejemplo, la función "mostrarVectInt" sólo permite mostrar vectores de tipo entero, para mostrar otro tipo de vectores, por ejemplo reales, se debe elaborar otra función similar pero que reciba un vector de tipo "double" en lugar de "int".

Para crear funciones de carácter más general se puede recurrir a la sobrecarga de funciones y al uso de plantillas.

13.4. SOBRECARGA DE FUNCIONES

La sobrecarga de funciones permite declarar dos o más funciones con el mismo nombre pero con diferentes tipos de parámetros y/o diferente número de de parámetros.

Por ejemplo en lugar de crear las funciones mostrarVectInt, mostrarVectorUnsigned, mostrarVectFloat y MostrarVectorDouble, se pueden crear cuatro funciones con el mismo nombre "mostrarVector" sólo que con diferentes tipos de parámetros:

```
void mostrarVector(int x[], int n){
  int c=0;
  for (int i=0;i<n;i++) {
    cout<<setw(8)<<x[i];
    c+=8;
    if (c>=72) {cout<<endl; c=0;}
  }
  cout<<endl;
}

void mostrarVector(unsigned x[], int n){
  int c=0;
  for (int i=0;i<n;i++) {
    cout<<setw(8)<<x[i];
    c+=8;
```

```
      if (c>=72) {cout<<endl; c=0;}
    }
    cout<<endl;
  }

  void mostrarVector(float x[], int n){
    int c=0;
    for (int i=0;i<n;i++) {
      cout<<setw(8)<<x[i];
      c+=8;
      if (c>=72) {cout<<endl; c=0;}
    }
    cout<<endl;
  }

  void mostrarVector(double x[], int n){
    int c=0;
    for (int i=0;i<n;i++) {
      cout<<setw(8)<<x[i];
      c+=8;
      if (c>=72) {cout<<endl; c=0;}
    }
    cout<<endl;
  }
```

Una vez creadas, el compilador sabe a cual de ellas llamar en base al tipo de dato del vector. Así, en el siguiente segmento de código se crean cuatro vectores (con cuatro tipos de datos) y todos ellos se muestran llamando a "mostrarVector":

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
double y[]={1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2};
float z[]={3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8,3.9,4.0};
unsigned w[]={10,11,12,13,14,15,16,17,18,19,20,21,22,23,24};
mostrarVector(x,20);
mostrarVector(y,12);
mostrarVector(z,10);
mostrarVector(w,15);
```

Con lo que los cuatro vectores se muestran correctamente:

```
       1       2       3       4       5       6       7       8       9
      10      11      12      13      14      15      16      17      18
      19      20
     1.1     1.2     1.3     1.4     1.5     1.6     1.7     1.8     1.9
       2     2.1     2.2
     3.1     3.2     3.3     3.4     3.5     3.6     3.7     3.8     3.9
       4
      10      11      12      13      14      15      16      17      18
      19      20      21      22      23      24
```

De esa manera sólo se tiene que recordar un nombre para la función (en lugar de cuatro o más), sin embargo, siempre existe la posibilidad de olvidar sobrecargar la función para algún tipo de dato en particular o inclusive, puede suceder que se quiera emplear la función con un nuevo tipo de dato.

La sobrecarga de funciones constituye una buena solución para casos como estos, sin embargo, una solución más sencilla y de carácter más general se la consigue con las plantillas.

13.5. PLANTILLAS

Las plantillas permiten crear funciones de carácter general al hacer posible la creación de comodines para los tipos de datos. Estos comodines pueden ser empleados dentro de las funciones igual que un tipo estándar, luego, cuando se llama a la función con tipos de datos concretos, el compilador reemplaza los comodines por dichos tipos de datos.

Los comodines se crean de acuerdo a la siguiente sintaxis:

```
template <class nombre_comodín_1, class nombre_comodín_2, ...>
```

Por ejemplo, para crear la plantilla de una función que calcule el cuadrado de cualquier tipo de dato numérico, se puede puede escribir:

```
template <class tipo1>
tipo1 sqr(tipo1 x){
  return x*x;
}
```

Donde el comodín del tipo de dato se ha denominado "tipo1". Ahora, se puede emplear esta plantilla con datos numéricos de diferentes tipos, como ocurre en el siguiente segmento de código:

```
  double x=3.45;
  cout<<"Cuadrado de "<<x<<" : "<<sqr(x)<<endl;
  int y=-73;
  cout<<"Cuadrado de "<<y<<" : "<<sqr(y)<<endl;
  unsigned z=98;
  cout<<"Cuadrado de "<<z<<" : "<<sqr(z)<<endl;
```

Que al ser ejecutado devuelve los resultados correctos:

```
Cuadrado de 3.45 : 11.9025
Cuadrado de -73 : 5329
Cuadrado de 98 : 9604
```

De igual manera, empleando comodines, se puede crear una plantilla para la función que muestra los elementos de un vector:

```
template <class tipo2>
void mostrarVector(tipo2 *x, unsigned n){
  unsigned c=0;
  for (unsigned i=0;i<n;i++){
    cout<<setw(8)<<x[i];
    c+=8;
    if (c>=72) {cout<<endl; c=0;}
  }
  cout<<endl;
}
```

Que al ser llamado desde el siguiente segmento de código:

```
  int x[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
  double y[]={1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2};
  float z[]={3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8,3.9,4.0};
  unsigned w[]={10,11,12,13,14,15,16,17,18,19,20,21,22,23,24};
  mostrarVector(x,20);
  mostrarVector(y,12);
  mostrarVector(z,10);
  mostrarVector(w,15);
```

Muestra correctamente los elementos sin importar el tipo de dato:

```
       1       2       3       4       5       6       7       8       9
```

```
 10    11    12    13    14    15    16    17    18
 19    20
1.1   1.2   1.3   1.4   1.5   1.6   1.7   1.8   1.9
  2   2.1   2.2
3.1   3.2   3.3   3.4   3.5   3.6   3.7   3.8   3.9
  4
 10    11    12    13    14    15    16    17    18
 19    20    21    22    23    24
```

Como se puede observar el crear y mantener funciones empleando plantillas resulta más sencillo que con la sobrecargadas de funciones (pues se evita reescribir dos o más veces el mismo código), sin embargo, es necesario tener en cuenta que las plantillas no son funciones. Las funciones propiamente las crea el compilador cuando se usa la plantilla con algún tipo de dato en concreto y es el compilador que se encarga de crear tantas funciones sobrecargadas como tipos de datos diferentes se empleen con la plantilla.

13.6. PARÁMETROS POR DEFECTO

En ocasiones algunos de los datos que se emplean dentro de una función son casi siempre los mismos, como ocurre con la función "mostrarVector" con el ancho de cada número (8) y el ancho de la fila (72), de manera que pueden ser escritas como constantes, tal como se ha hecho en dicho ejemplo. El problema es que para emplear otro ancho se debe modificar el código, algo que no es recomendable, pues es una posible fuente de errores y además implica la recompilación del programa.

Una alternativa más conveniente es el empleo de parámetros por defecto. Los parámetros por defecto son aquellos a los que se les asigna un valor al momento de declararlos y es ese el valor que toman en caso de que no se le mande otro.

Los parámetros por defecto se declaran igual que un parámetro estándar excepto que se les asigna un valor (con el operador =), además, los parámetros por defecto deben ser declarados al final de la lista de parámetros, después de los parámetros normales.

Empleando parámetros por defecto para fijar ancho de los elementos (ae) y el ancho de cada línea (al), se puede modificar la plantilla "mostrarVector" de la siguiente forma:

```
template <class tipo2>
void mostrarVector(tipo2 *x, unsigned n, unsigned ae=8, unsigned al=72){
  unsigned c=0;
  for (unsigned i=0;i<n;i++){
    cout<<setw(ae)<<x[i];
    c+=ae;
    if (c>=al) {cout<<endl; c=0;}
  }
  cout<<endl;
}
```

En el siguiente segmento de código se emplea esta plantilla, con elementos de tipo entero, primero sin cambiar los parámetros por defecto, luego cambiando el ancho de los elementos a 16 y finalmente cambiando el ancho de los elementos a 16 y de las líneas a 48:

```
int x[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
mostrarVector(x,20);
mostrarVector(x,20,16);
mostrarVector(x,20,16,48);
```

Que al ser ejecutada produce:

```
 1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18
19 20

 1  2  3  4
5

 6  7  8  9
10

11 12 13 14
15

16 17 18 19
20


 1  2  3
 4  5  6
 7  8  9
10 11 12
13 14 15
16 17 18
19 20
```

De esa manera se tiene una plantilla de carácter general para mostrar vectores de cualquier tipo, donde se puede modificar el ancho de los elementos y de las filas.

Ahora bien, para emplear esta plantilla en un programa sería necesario reescribirla (o copiarla), algo que no es eficiente y que va en contra del principio de la modularidad (que a más de facilitar el desarrollo busca evitar la repetición de código). Para evitar ello se debe recurrir al empleo de archivos de cabecera y bibliotecas de archivo.

13.7. ARCHIVOS DE CABECERA Y BIBLIOTECAS

Los archivos de cabecera son archivos de texto que pueden ser incluidos en cualquier programa con la instrucción "#include". Los archivos de cabecera tienen la extensión "*.h" (o "*.hpp") y por lo general contienen la declaración de constantes, identificadores y funciones (prototipos), sin embargo pueden contener cualquier instrucción válida en C/C++.

Constituyen una primera alternativa para evitar el reescribir código, pues en dichos archivos se puede escribir el código que se quiere esté a disposición de otros programas. Con ello se evita la copia manual de la función o funciones que se quieren reutilizar, sin embargo, al ser un archivo de texto, una vez insertado en el programa debe ser compilado (conjuntamente el programa) para que pueda ser ejecutado. Por lo tanto las funciones de un archivo de cabecera son vueltos a compilar en cada uno de los programas que los importa.

Para evitar el volver a compilar una y otra vez las mismas funciones, se deben crear bibliotecas. Las bibliotecas son colecciones de funciones compiladas que pueden ser incorporadas directamente en otros programas (o en otras bibliotecas). Además, cuando se incluye una biblioteca en un programa (o en otra biblioteca), no se incluyen todas las funciones que contiene sino sólo aquellas que efectivamente son empleadas por el programa donde es importado, lo que reduce el tamaño del archivo resultante.

En la práctica los archivos de cabecera y las bibliotecas se emplean conjuntamente: En los archivos de cabecera se escriben las declaraciones de funciones, constantes y otros identificadores, mientras que en las bibliotecas (que suelen tener el mismo nombre que los archivo de cabecera pero

con la extensión "*.cpp") se escribe el código de las funciones declaradas en el archivo de cabecera (así como el código de todas las funciones que estas funciones requieran).

En esta forma (archivos de cabecera – bibliotecas) han sido empleadas desde el primer programa elaborado en la materia. Así cuando en el código fuente (en el programa) se escribe la instrucción "#include <iostream>", se instruye al compilador que incluya el archivo de cabecera "iostream" y al momento de crear el ejecutable, el compilador incorpora en el programa todas las funciones que se emplean en el programa de la librería respectiva.

Estrictamente hablando en C/C++ se emplea el término "biblioteca" (algunas veces denominada "librería") para hacer referencia a una biblioteca de archivos compilados incorporados en un archivo único (un archivo "*.a"). En esta materia se empleará el término "biblioteca" para hacer referencia a una biblioteca de funciones, en ese sentido una biblioteca puede constar de un solo archivo compilado (un archivo objeto "*.o") o de varios (un archivo "*.a").

Para comprender mejor el proceso que se sigue en la creación y uso de librerías, se creará una con dos funciones ("cuad" y "cubo") para calcular el cuadrado y el cubo de números enteros y reales.

Primero se crea el archivo de cabecera "lib1.h":

```
C:\HPV\programas\c++\2-2012\tema13>notepad lib1.h
```

Y en el mismo se escribe lo siguiente:

```
#ifndef LIB1_H_
#define lib1_H_

int cuad(int);
double cuad(double);
int cubo(int);
double cubo(double);

#endif
```

En este código la instrucción "#ifndef" (si no está definido), sirve para averiguar si el nombre LIB1_H_ no está definido, si es así, es decir si el nombre LIB1_H_ no está definido, se ejecutan las instrucciones dentro del bloque #ifndef – #endif, caso contrario estas instrucciones se ignoran.

Las instrucciones que comienzan con un "#" son directivas de compilación (o directivas de preprocesamiento) y mediante ellas se le instruye al compilador las tareas que debe realizar antes de compilar el código. La instrucción "#include" es también una directiva de compilación y como ya se ha visto, con la misma se le instruye al compilador que incluya en el programa (antes de compilar) el archivo que se le indica.

La instrucción #endif, simplemente marca el final de la instrucción #ifndef, o de #ifdef (si está definido, el complemento de #ifndef) o de #if (si lógico). Dentro de estos bloques se pueden emplear también las instrucciones "#else" (caso contrario) y "#elif" (else if).

La instrucción #define define un nombre (conocida como macro) y su sintaxis es la siguiente:

```
#define nombre_de_la_macro secuencia_de_caracteres
```

Donde la secuencia de caracteres puede estar en blanco (como en el archivo de cabecera) o ser cualquier constante o secuencia de instrucciones vá-

lidas en C/C++. Observe que las directivas de compilación no terminan con ";", esto se debe a que dichas instrucciones sólo pueden ocupar una línea, por lo tanto las mismas concluyen cuando termina la línea (es decir cuando se encuentra el código de un salto de línea).

Por ejemplo las siguientes instrucciones definen las constantes UNO, SALUDO y NUM_ELEMENTOS con los valores 1, "Hola Mundo!" y 500 respectivamente:

```
#define UNO 1
#define SALUDO "Hola Mundo!"
#define NUM_ELEMENTOS 500
```

No es obligatorio que el nombre de las macros estén en mayúsculas, sin embargo, es una norma defacto que así sea, de esa manera es más fácil identificarlas dentro de un programa. Una vez definida una constante (macro), puede ser empleada dentro de un programa de manera similar a una variable, por ejemplo, la siguiente instrucción define un vector de números reales con 500 elementos:

```
double x[NUM_ELEMENTOS];
```

Que antes de ser compilada se transforma en:

```
double x[500];
```

Es decir que el compilador, antes de proceder a la compilación, reemplaza, en todos los lugares donde aparecen las constantes, sus valores.

Las macros pueden tener también parámetros (de manera similar a una función), por ejemplo la siguiente macro permite sumar dos números:

```
#define SUM(a,b) (a)+(b);
```

Que puede ser empleada dentro de un programa de manera similar a una función, por ejemplo, en el siguiente segmento de código se emplea la macro para sumar dos números reales y dos números enteros:

```
double x=2.3, y=4.5, z;
int a=-23, b=56, c;
z=SUM(x,y);
c=SUM(a,b);
```

Aunque se emplean y comportan de manera similar a una función, no se debe olvidar que en realidad son constantes (macros), y que en consecuencia, antes la compilación, su nombre es reemplazado por su contenido, es decir, que las dos sentencias de asignación anteriores se transforman en:

```
z=(x)+(y);
c=(a)+(b);
```

En este caso los paréntesis no son necesarios, sin embargo, y dado que se trata de una sustitución directa, es conveniente emplearlos a fin de vitar errores debido al orden de prioridad de los operadores.

Cuando se quiere quitar la definición de una macro se puede emplea la instrucción #undef, seguida del nombre de la macro a remover.

Una vez que se ha repasado brevemente las directivas de compilación se puede comprender mejor lo que se ha hecho en el archivo de cabecera.

En primer lugar se averigua si el nombre "LIB1_H_" no está definido (#ifndef). Este nombre se define en este archivo y como se puede ver es el nombre del archivo en mayúsculas, donde la extensión se separa con un guión bajo y donde el nombre termina con otro guión bajo, por lo tanto, al ser definido en este archivo, se podría suponer que no está definido y que en consecuencia esta pregunta está por demás, sin embargo, los archivos de ca-

becera pueden ser incluidos en varios archivos dentro de un mismo proyecto, incluido otros archivos de cabecera, por lo que puede suceder (y con frecuencia es así) que el archivo ya haya sido incluido en otro archivo y que en consecuencia las declaraciones que contiene ya estén en memoria. En ese caso, si otro archivo intentara incorpora nuevamente las mismas declaraciones, se produciría un error, porque en C/C++ no se puede hacer la misma declaración 2 o más veces.

En consecuencia este nombre "LIB1_H_" se define únicamente para determinar si las declaraciones del archivo ya han sido incluidas (o no) dentro del programa: si está definido, las declaraciones ya han sido incluidas, caso contrario no.

Luego, en el archivo de cabecera se escriben todas las declaraciones que se quieren exportar (que se puedan emplear desde otros archivos), en este ejemplo los nombres de las cuatro funciones (dos funciones sobrecargadas).

Una vez creado el archivo de cabecera se crea el archivo fuente correspondiente:

```
C:\HPV\programas\c++\2-2012\tema13>notepad lib1.cpp
```

Y en el mismo se escribe el código:

```
#include "lib1.h"

int cuad(int x){
  return x*x;
}

double cuad(double x){
  return x*x;
}

int cubo(int x){
  return x*x*x;
}

double cubo(double x){
  return x*x*x;
}
```

Como se puede ver, lo primero que se hace en este archivo es incluir el archivo de cabecera "lib1.h". Observe que el nombre de este archivo está encerrado entre comillas (no entre "<>"). En C/C++, los archivos de cabecera creados por el usuario se incluyen escribiendo su nombre entre comillas. Esto le instruye al compilador que los busque comenzando en el lugar donde se encuentra el archivo que los incluye.

Una vez creado el archivo fuente se compila en el modo habitual:

```
C:\HPV\programas\c++\2-2012\tema13>g++ -c lib1.cpp
```

Con lo que se crea el archivo "lib1.o" que es el archivo que contiene las funciones compiladas (la biblioteca de funciones).

Ahora, para emplear la librería desde un programa, se incluye el archivo de cabecera "lib1.h":

```
#include <iostream>
#include <iomanip>
#include "lib1.h"
```

```
using namespace std;

int main(){
  double x=2.5;
  cout<<setw(20)<<"El cuadrado de "<<x<<" es: "<<cuad(x)<<endl;
  cout<<setw(20)<<"El cubo de "<<x<<" es: "<<cubo(x)<<endl;
  int a=-3;
  cout<<setw(20)<<"El cuadrado de "<<a<<" es: "<<cuad(a)<<endl;
  cout<<setw(20)<<"El cubo de "<<a<<" es: "<<cubo(a)<<endl;
  return 0;
}
```

Se compila el programa:

```
C:\HPV\programas\c++\2-2012\tema13>g++ -c proglib1.cpp
```

Con lo que se crea el programa objeto "proglib1.o", y se crea el ejecutable indicándole al compilador que enlace los programas objeto "proglib1.o" con "lib1.o" (la biblioteca creada):

```
C:\HPV\programas\c++\2-2012\tema13>g++ -o proglib1 proglib1.o lib1.o
```

Con lo que se crea el ejecutable "proglib1.exe", y haciendo correr el mismo se obtiene:

```
C:\HPV\programas\c++\2-2012\tema13>proglib1
     El cuadrado de 2.5 es: 6.25
         El cubo de 2.5 es: 15.625
     El cuadrado de -3 es: 9
         El cubo de -3 es: -27
```

Se podría pensar en emplear plantillas en lugar de sobrecargar las funciones, sin embargo, se debe recordar que las plantillas son solo moldes a partir de los cuales se construyen las funciones reales, por lo tanto no pueden ser compiladas. En este sentido las plantillas son esencialmente declaraciones y como tales pueden ser incluidas en los archivos de cabecera, pero no en una librería, por lo que las funciones generadas a partir de ellas deben ser compiladas conjuntamente el programa que las incluye, lo que consume más tiempo que el enlazar funciones ya compiladas.

Empleando plantillas y un archivo de cabecera, el anterior ejemplo puede ser resuelto creando primero el archivo de cabecera:

```
C:\HPV\programas\c++\2-2012\tema13>notepad lib1b.h
```

En el cual se escriben las plantillas:

```
#ifndef LIB1B_H_
#define LIB1B_H_

template <class T>
T cuad(T x){
  return x*x;
}

template <class T>
T cubo(T x){
  return x*x*x;
}

#endif
```

Ahora se crea el programa que hace uso de las mismas:

```
C:\HPV\programas\c++\2-2012\tema13>notepad proglib1b.cpp
```

```
#include <iostream>
#include <iomanip>
#include "lib1b.h"
using namespace std;

int main(){
  double x=2.5;
  cout<<setw(20)<<"El cuadrado de "<<x<<" es: "<<cuad(x)<<endl;
  cout<<setw(20)<<"El cubo de "<<x<<" es: "<<cubo(x)<<endl;
  int a=-3;
  cout<<setw(20)<<"El cuadrado de "<<a<<" es: "<<cuad(a)<<endl;
  cout<<setw(20)<<"El cubo de "<<a<<" es: "<<cubo(a)<<endl;
  return 0;
}
```

Que se compila en la forma habitual:

```
C:\HPV\programas\c++\2-2012\tema13>g++ -c proglib1b.cpp
```

Y se crea el ejecutable, también en la forma habitual:

```
C:\HPV\programas\c++\2-2012\tema13>g++ -o proglib1b proglib1b.o
```

Como era de esperar, al hacer correr el programa ("proglib1b.exe") se obtienen los mismos resultados que cuando se emplea una librería:

```
C:\HPV\programas\c++\2-2012\tema13>proglib1b
     El cuadrado de 2.5 es: 6.25
         El cubo de 2.5 es: 15.625
     El cuadrado de -3 es: 9
         El cubo de -3 es: -27
```

La única diferencia, que en un ejemplo tan pequeño como el presente es imperceptible, es el tiempo consumido en la creación del ejecutable.

Entonces cuando se quiere reutilizar plantillas, deben ser incluidas únicamente en archivos de cabecera, no en las bibliotecas porque no pueden ser compiladas (por lo menos no directamente).

Tanto las funciones sobrecargadas como las plantillas constituyen opciones viables al momento de evitar la reescritura de código. Las plantillas son más sencillas de crear y reutilizar que las funciones sobrecargadas, sin embargo, no pueden ser compiladas, por lo que suelen consumir más recursos del sistema, además, las funciones sobrecargadas (y las funciones normales) al ser compiladas en bibliotecas quedan protegidas, pues no es posible modificarlas (al menos no directamente).

13.8. EJERCICIOS

1. Empleando funciones sobrecargadas cree el archivo de cabecera "vectores_a.h" y la biblioteca respectiva ("vectores_a.o") con la función "mostrarVector", para mostrar los elementos de un vector de "n" números enteros, reales o de caracteres, permitiendo establecer el ancho de cada elemento y el ancho de cada línea. Pruebe la librería mostrando los elementos de un vectores de 15 números enteros, otro de 10 elementos de números reales y otro con 12 elementos de tipo carácter.

2. Empleando plantillas cree el archivo de cabecera "vectores_b.h" con la función "mostrarVector", para mostrar los "n" elementos de un vector. Pruebe la función con un vector de 10 números enteros sin signo, otro con 7 números enteros con signo, otro con 8 elementos de tipo real y otro con 11 elementos de tipo carácter.

3. Añada al archivo de cabecera "vectores_a.h" y a la biblioteca respectiva, una función sobrecargada para leer los elementos de un vector de números enteros, reales y de caracteres. Los elementos del vector deben ser tratados como punteros. Pruebe la función leyendo y mostrando los elementos de un vector con 7 números enteros, otro con 10 números reales y otro con 15 elementos de tipo carácter.

4. Añada al archivo de cabecera "vectores_b.h" una plantilla para leer los elementos de un vector. Los elementos del vector deben ser tratados como punteros. Pruebe la plantilla leyendo y mostrando los elementos de un vector con 10 caracteres, otro con 5 números reales y otro con 7 números enteros sin signo.

5. Añada al archivo de cabecera "vectores_a.h" (y a la biblioteca respectiva) una función sobrecargada para generar un vector con "n" elementos de tipo entero, de tipo real y de tipo carácter, comprendidos entre dos límites dados. Pruebe la función generando y mostrando 3 vectores con 20 elementos de tipo entero, 30 elementos de tipo real y 50 elementos de tipo carácter.

6. Añada al archivo de cabecera "vectores_b.h" una plantilla para generar los elementos de un vector comprendidos entre 2 límites dados. Pruebe la plantilla generando y mostrando un vector con 30 elementos de tipo carácter, otro con 20 elementos de tipo entero y otro con 10 de tipo real.

7. Añada al archivo de cabecera "vectores_a.h" (y a la biblioteca respectiva) una función sobrecargada para copiar los elementos de un vector de números enteros, números reales y caracteres. Los elementos del vector deben ser tratados como punteros y por defecto debe copiar todos los elementos del vector, sin embargo debe permitir especificar también el número de elementos a copiar. Pruebe la función generando 3 vectores con 10 elementos de números enteros, reales y caracteres y copiando en otras variables todos los elementos del primer vector, los 7 elementos del segundo y los 5 elementos del tercero.

8. Añada al archivo de cabecera "vectores_b.h" una plantilla para copiar los elementos de un vector. Los elementos del vector deben ser tratados como punteros y por defecto debe copiar todos los elementos del vector, sin embargo debe permitir especificar también el número de elementos a copiar. Pruebe la función generando 3 vectores con 20 elementos de números enteros, reales y caracteres y copiando en otras variables los primeros 5 elementos del primer vector, los 7 elementos del segundo y todos los elementos del tercero.

9. Cree el archivo de cabecera "ejer9_a.h" y la biblioteca respectiva, con una función sobrecargada que invierta el orden de los elementos de un vector de números enteros, reales y caracteres. Pruebe el archivo creando 3 vectores con 5 elementos de tipo entero, real y carácter, introduciendo sus valores por teclado y mostrando los vectores con sus elementos originales e invertidos.

10. Cree el archivo de cabecera “ejer9_b.h” con una plantilla que invierta el orden de los elementos de un vector. Los elementos del vector deben ser tratados como punteros. Pruebe la función creando 3 vectores con 7 elementos de tipo entero, real y carácter, introduciendo sus valores por teclado y mostrando los vectores con sus elementos originales e invertidos.

14. MATRICES 2

Habiendo estudiado las operaciones más frecuentes con matrices, así como la reutilización de código, en este tema se profundizan dichos conceptos y se introduce el ambiente de desarrollo "Eclipse".

14.1. BREVE INTRODUCCIÓN A ECLIPSE

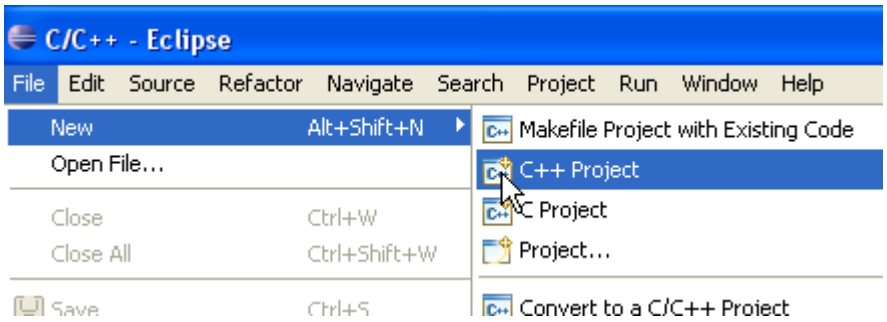
Si bien los programas que se elaboran en la materia son pequeños y pueden ser creados y mantenidos sin problema desde la línea de comando, a medida que incrementan en tamaño y complejidad, el mantenimiento se hace cada vez más difícil y en aplicaciones reales (con decenas, centenas o miles de archivos) es casi imposible.

Por eso, en el desarrollo de aplicaciones reales se recurre, casi siempre, a los ambientes de desarrollo y uno de dichos ambientes es Eclipse, el cual, además de ser muy completo, es libre.

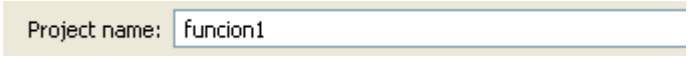
Para escribir código C++ en Eclipse, primero se ingresa a eclipse (haciendo doble clic en el archivo "eclipse-C++.exe" dentro de la carpeta de "eclipse.c++" (carpeta es la carpeta dejada en el centro de estudiantes).

La primera vez que se ingresa a Eclipse aparece la ventana de bienvenida, para comenzar a trabajar con eclipse se debe hacer clic en el icono:

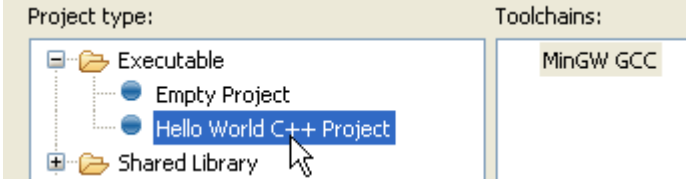
Una vez en el ambiente de desarrollo de eclipse se crea un nuevo proyecto:



Se da un nombre al proyecto:

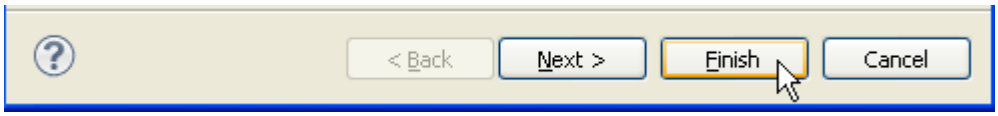


Y se selecciona el tipo de proyecto. Sobre todo en los primeros proyectos se recomienda seleccionar como tipo de proyecto "Hello World C++ Project":



Se hace clic en el botón finish (en realidad se puede añadir información adicional sobre el proyecto haciendo click en "next", pero dicha informa-

ción no es relevante por el momento):



Con ello eclipse crea el siguiente código:

```
//============================================================================
// Name        : funcion1.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
    cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
    return 0;
}
```

Las primeras líneas (en verde) son comentarios, que como se sabe, no forman parte del ejecutable, por lo que incluso pueden ser suprimidas. Como se puede ver "eclipse" crea el código de un programa que imprime en pantalla el mensaje "Hola Mundo" en inglés, donde ya está incluida la librería "iostream" y el espacio de nombres "namespace".

La diferencia obvia, como normalmente ocurre con los ambientes de desarrollo, es que los diferentes componentes del programa están claramente identificados con diferentes colores y estilos.

Para crear programas simplemente se borran las partes que no son de utilidad y se escribe y/o añade código propio. Por ejemplo, se puede ambiar el programa para que muestre el mensaje en español:

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

int main(){
  cout << "¡¡¡Hola Mundo!!!" << endl;
  return 0;
}
```

Una vez escrito el programa, se debe guardar el mismo: Ctrl+S o File ->Save o clic en el botón guardar:

Luego se compila y construye el ejecutable: Project ->Build project o clic en el botón construir:

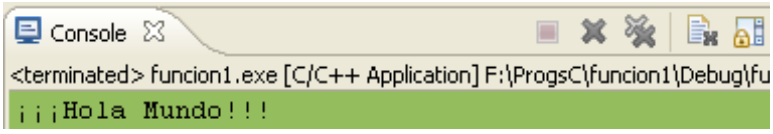
Esta es, por supuesto, la diferencia más notable con relación a la elaboración de programas en la línea de comando: el programa se compila y se crea el ejecutable con un solo clic.

Los resultados del proceso de compilación y creación del ejecutable se presentan en la consola (en la ventana inferior central de Eclipse) y en el mismo se informa si la compilación ha tenido éxito o si ha ocurrido algún error. Básicamente se muestran los mismos mensajes de error que aparecen en la línea de comando cuando se compilan y crean ejecutables.

Si existe algún error, el código debe ser corregido en el lugar correspondiente. Esta es otra de las diferencias notables: los mensajes de error y el código se encuentran en la misma ventana y basta hacer doble clic en el mensaje de error para saltar al lugar donde se produjo el mismo, o simplemente echar un vistazo al código, pues en el mismo los errores también están resaltados.

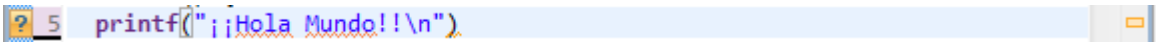
Finalmente, una vez corregidos todos los errores, se hace correr el programa (se ejecutar el programa): Ctrl+F11 o Run->Run o clic en el icono Ejecutar:

Con lo que se obtiene la salida en la consola de la computadora (¡todo en el mismo lugar!)

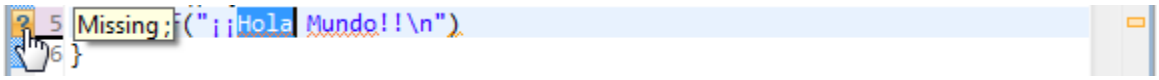


En realidad, en un ambiente de desarrollo como Eclipse, los tres pasos: la compilación, creación del ejecutable y la ejecución del programa, puede ser llevada a cabo en uno solo: haciendo clic en el botón ejecutar.

Otra de las ventajas de los ambientes de desarrollo es la que proporcionan ayuda continua a medida que se escribe un programa. Por ejemplo, al escribir la siguiente instrucción:

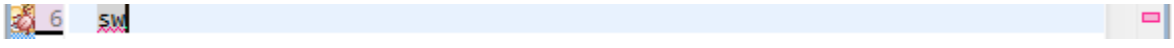


Eclipse nos informa de varias cosas: a) Que ha reconocido el comando "printf", pues se encuentra resaltado como una instrucción; b) Que el texto (entre comillas) ha sido reconocido como tal, pues se encuentra también en un color diferente; c) Que el paréntesis que acaba de cerrarse corresponde al paréntesis abierto en "printf" (dicho paréntesis aparece remarcado) y d) Que existe un error de sintaxis hasta donde se ha escrito el código, por eso aparece un signo de interrogación en la parte izquierda y un recuadro en la parte derecha. El error detectado puede ser visto colocando el puntero sobre el signo de interrogación (o sobre el recuadro):



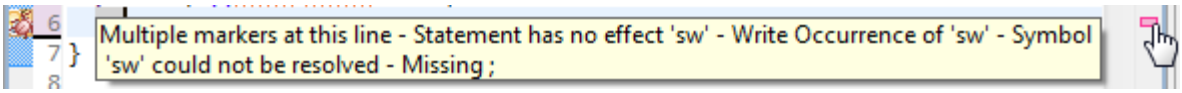
Como se puede ver, en este caso Eclipse informa que falta el punto y coma al final de la instrucción.

Eclipse además permite completar código automáticamente (lo que no siempre es bueno, sobre todo cuando realmente se quiere aprender un lenguaje). Por ejemplo, para escribir código utilizando la estructura "switch", se puede comenzar escribiendo las letras "sw":



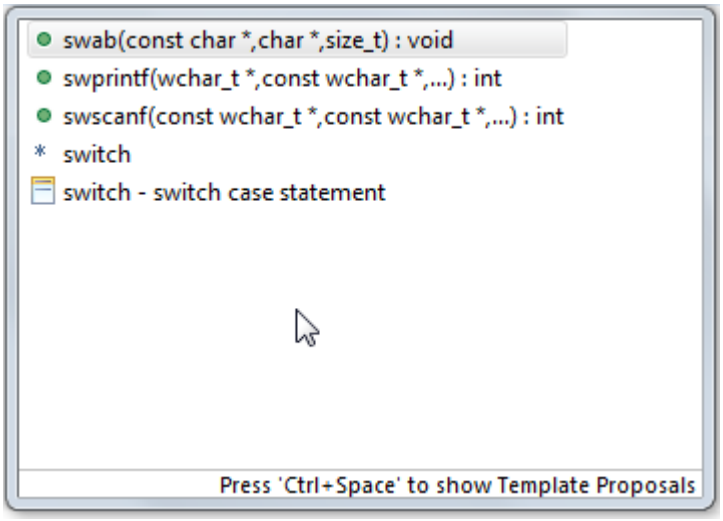
Donde el bichito que aparece en el lado izquierdo (y el recuadro rojo en

el derecho) informan que, existe algún problema con el código. El problema detectado puede ser visto colocando el puntero sobre el bichito (o en el recuadro rojo):

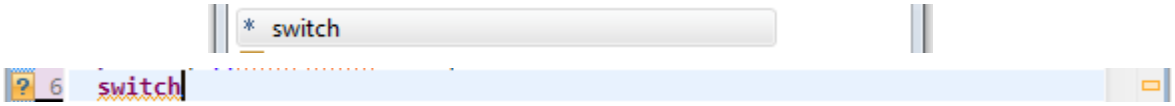


En este caso Eclipse informa que no ha reconocido como comando o variable "sw" (porque no lo es) y además no ha encontrado el punto y coma al final de la sentencia. Por supuesto, en este caso, se sabe que existen dichos errores porque el código está aún incompleto.

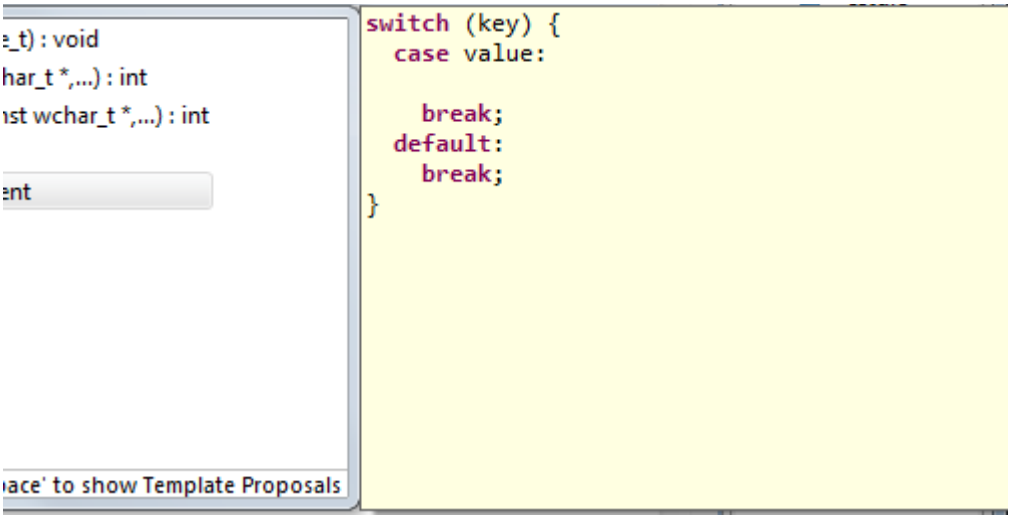
Si ahora, estando el cursor después de "sw", se mantiene pulsa la tecla "ctrl" y se pulsa la tecla "espacio", aparece la ventana:



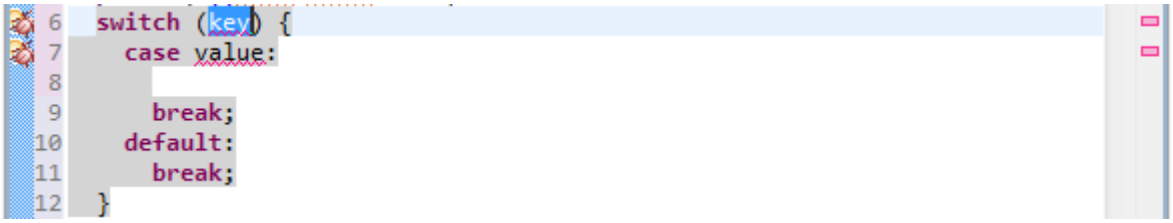
Donde Eclipse muestra las instrucciones que comienzan con "sw". Como se puede ver existe para "switch" dos opciones. La primera es simplemente la palabra. Si se selecciona y hace enter la misma es escrita en el código:



La segunda es la estructura completa, al seleccionarla aparece una nueva ventana, que nos muestra la estructura:



Y al pulsar la tecla "enter" esta estructura es insertada en el código:



Este código también se inserta (sin que aparezcan las ventanas) si se escribe la palabra "switch" y se pulsan las teclas "ctrl+espacio".

Observe también que Eclipse identa automáticamente el código, por lo que en este ambiente es más difícil escribir un programa no identado que uno identado (recuerde que desde el tema 8 los programas tienen que estar identados)

Eclipse tiene además muchas otras características y funciones, algunas de las cuales serán introducidas a medida que se elaboren programas en el mismo.

A partir de este tema los programas pueden ser desarrollados indistintamente ya sea en Eclipse o desde la línea de comando, si se prefiere trabajando desde la línea de comando se recomienda el uso de un editor más avanzado como notepad++ (el mismo que cuenta con resaltado de texto, identación automática, autocompleatado y otras herramientas más).

14.2. BIBLIOTECAS Y ARCHIVOS DE CABECERA DE USO FRECUENTE

En el anterior tema las bibliotecas y los archivos de cabecera han sido creados y utilizados dentro del mismo directorio. No obstante, las bibliotecas y archivos de cabecera de uso frecuente deben estar disponibles para los programas que así lo requieran, por lo que es conveniente que se conveniente agruparlos en directorios con bibliotecas relacionadas.

En lo sucesivo, se asumirá que dichos archivos se encuentran dentro del directorio "include".

En concordancia con los ejemplos desarrollados hasta el momento, este directorio se encontrará dentro de "C:\HPV\programas\c++\2-2012\", sin embargo, este es sólo un directorio de referencia, por lo que se puede emplear cualquier otro (siempre y cuando dentro del mismo esté el directorio "include", así como los directorios creados para cada tema).

Inicialmente, este directorio contendrá el archivo de cabecera "matrices.h", con las plantillas de uso general elaboradas hasta el momento, es decir:

```
#ifndef MATRICES_H_
#define MATRICES_H_

#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

namespace matrices{

  //Tipo de dato generado en el error
  typedef int ERROR;
```

```
//Constantes correspondientes a los errores
//Límites erróneos en le generación de elementos
const ERROR ERROR_LIMITES=1;
//Dígitos de redondeo muy grande
const ERROR ERROR_REDONDEO=2;

//Plantilla para mostrar "n" elementos del vector "x" mostrando,
//por defecto desde el primer elemento (pe=0). Los elementos se
//muestran por defecto con un ancho de 8 caracteres (ae) y
//72 caracteres por línea (al)
template <class T>
void mostrarVector(T *x, unsigned n, unsigned pe=0, unsigned ae=8,
    unsigned al=72) throw()
{
  unsigned c=0;
  for (unsigned i=0;i<n;i++){
    cout<<setw(ae)<<x[i];
    c+=ae;
    if (c>=al) {cout<<endl; c=0;}
  }
  if (c!=al) cout<<endl<<endl; else cout<<endl;
}

//Plantilla para generar un vector con "n" elementos aleatorios
//comprendidos entre los límites "li" (inferior) y "ls" (superior)
template <class T>
T *generarVector(unsigned n, T li, T ls) throw(ERROR)
{
  if (ls<=li) throw ERROR_LIMITES;
  T d=ls-li, *x=new T[n];
  for (unsigned i=0;i<n;i++)
    x[i]=(T)rand()*d/RAND_MAX+li;
  return x;
}

//Plantilla para leer, desde teclado, "n" elementos en el vector "x"
template <class T>
void leerVector(T* &x, unsigned n) throw()
{
  x=new T[n];
  for (unsigned i=0;i<n;i++) {
    cout<<"x["<<i<<"]? ";
    cin>>x[i];
  }
}

//Plantilla para devolver un vector con una copia de "n" elementos
//del vector "x". Por defecto los elemento se copian comenzando con
//el primer elemento, el elemento 0 (pe).
template <class T>
T *copiarVector(T *x, unsigned n, unsigned pe=0) throw()
{
  T *y=new T[n]; n+=pe;
  for (unsigned i=pe;i<n;i++)
    y[i]=x[i];
  return x;
}
```

```
#endif
```

Como ya se vio en el anterior tema, las plantillas no pueden ser compiladas, por lo que, por el momento, no se requiere la biblioteca respectiva.

Observe que en la plantilla "leerVector" se recibe la variable donde se lee el vector, la variable "x", como un puntero por referencia. Se procede así para que la dirección de memoria que se reserva dentro del módulo quede guardada en la variable original (la variable externa). Si no se recibiera la variable por referencia, la memoria sería reservada y los datos leídos, pero la dirección de memoria de dichos datos se perdería definitivamente.

En lugar de recibir la variable por referencia se podría pensar en recibir sólo el número de elementos y devolver un puntero (como se hace con la plantilla "generarVector"), sin embargo, eso no es posible en este caso porque, cuando se trabaja con plantillas, al menos uno de los parámetros debe ser del tipo (o tipos) definido para la plantilla. La otra alternativa, que sí funcionaría, sería el crear funciones sobrecargadas, eso sí teniendo el cuidado de sobrecargar la función para todos los posibles tipos de datos.

Como ya se dijo, para emplear este archivo de cabecera es suficiente, por el momento, que sea incluido en el programa (con "#include"), sin embargo, puesto que este archivo se encuentra en el directorio "include" y no en el directorio donde se escribe el programa, es necesario informar de ello al compilador. Con ese fin, se pueden seguir varios caminos, el más sencillo consiste en escribir el camino relativo o completo en la directiva "#include", por ejemplo, en el siguiente programa se le da la localización relativa (empleando, sólo como ejemplo, notepad++):

```
C:\HPU\programas\c++\2-2012\tema14>notepad++ prue1.cpp
```

```
#include "../include/matrices.h"
using namespace matrices;

int main(){
  int *x;
  x=generarVector(20,1,100);
  mostrarVector(x,10);
  int *y;
  y=copiarVector(x,5);
  mostrarVector(y,5);
  double *z;
  z=generarVector(100,-10.0,10.0);
  mostrarVector(z,100,0,12);
  int *a;
  leerVector(a,9);
  mostrarVector(a,9);
  delete(x);
  delete(y);
  delete(z);
  delete(a);
  return 0;
}
```

Luego se procede a la compilación y enlazado en la forma habitual:

```
C:\HPV\programas\c++\2-2012\tema14>g++ -c prue1.cpp

C:\HPV\programas\c++\2-2012\tema14>g++ -o prue1 prue1.o
```

Y haciendo correr el programa se obtiene:

```
C:\HPV\programas\c++\2-2012\tema14>prue1
       1       56       20       81       58       48       35       89       82
      74

       1       56       20       81       58

      9.7705    -1.08615    -7.61834    -9.90661    -9.82177     -2.4424
    0.633259     1.42369     2.03528     2.14331    -6.67531      3.2609
   -0.984222    -2.95755    -8.85922     2.15369     5.66637     6.05213
    0.397656      -3.961     7.51946     4.53352     9.11802     8.51436
    0.787072    -7.15323   -0.758385    -5.29344     7.24479    -5.80798
     5.59313     6.87307     9.93591      9.9939     2.22999    -2.15125
    -4.67574    -4.05438     6.80288    -9.52513    -2.48268    -8.14753
     3.54411     -8.8757    -9.82421      8.3758    -4.48225    -4.54207
     1.75817     3.82366     6.75222     4.52986   -0.301218    -5.89282
     4.87472   -0.630818   -0.840785     8.98312     4.88876    -7.83441
     1.98096     -2.2953     4.70016     2.17933      1.4481    -2.77322
     -6.9689    -5.49791    -1.49693     6.05762    0.342112      9.7998
     5.03098    -3.08878    -6.62038     3.14615   -0.162053    -8.72921
     3.99518   0.0961333    -7.05008     8.99167    -7.16849     8.10236
     3.85784    -3.93902    -1.46886    -8.59249     9.33226     3.66375
    -6.93533     7.54509     6.43361     1.64098    -6.17298    -6.44215
     6.34388   -0.494705    -6.88894   0.0784326

x[0]? 1
x[1]? 2
x[2]? 3
x[3]? 4
x[4]? 5
x[5]? 6
x[6]? 7
x[7]? 8
x[8]? 9
       1        2        3        4        5        6        7        8        9
```

Alternativamente, se puede incluir el archivo en la forma habitual, es decir:

```
#include "matrices.h"
using namespace matrices;
```

Y al momento de compilar, se le instruye al compilador, con la opción "-I", que busque los archivos de cabecera en el directorio indicado (en este caso en "include"):

```
C:\HPV\programas\c++\2-2012\tema14>g++ -c prue1.cpp -I../include

C:\HPV\programas\c++\2-2012\tema14>g++ -o prue1 prue1.o
```

Donde, al igual que en el anterior ejemplo, los dos puntos ".." representan al directorio padre (recuerde también que un punto "." representa al directorio actual). Observe que se ha empleado como separador de directorios "/" y no "\", esto porque el compilador (GNU) proviene de los sistemas operativos Linux y en dichos sistemas se emplea "/" para separar los directorios (sin embargo, se puede emplear también "\").

Note también que en este programa no sólo se incluye el archivo "matrices.h" sino también el espacio de nombres "matrices", porque es en espacio donde se han declarado las plantillas y constantes de esta librería, por lo tanto no es suficiente incluir el archivo, sino que también se debe usar el espacio de nombres respectivo.

Suponga ahora que se quieren crear cuatro funciones: 1) para generar un número entero aleatorio, 2) para genera un número real aleatorio, 3) para redondear un número real a un determinado número de dígitos después del punto y 4) para redondear los primeros "n" elementos de un vector de números reales a un determinado número de dígitos después del punto.

Las dos primeras funciones podrían ser implementadas mediante una plantilla, sin embargo, dado que sólo se debe considerar dos tipos de datos, es más eficiente implementarlos a través de funciones sobrecargadas. Para las dos últimas, las plantillas no constituyen una buena opción, pues sólo tiene sentido redondear números reales (no enteros, caracteres, cadenas, etc.), por lo que la forma más eficiente de implementarlas es también con funciones sobrecargadas.

Primero se añaden los prototipos de estas funciones (las declaraciones de estas funciones) en el archivo de cabecera "include.h":

```
//Genera un númeo entero comprendido entre los límites "li" y "ls"
int generarNumero(int li, int ls) throw(ERROR);

//Genera un número real comprendido entre los límites "li" y "ls"
double generarNumero(double li, double ls) throw(ERROR);

//Redondea el número real "x" a "n" dígitos depués del punto, siendo
//el número de dígitos por defecto 0
void redondear(double &x, unsigned n=0) throw(ERROR);

//Redondea los primeros "n" elementos del vector "x" a "m" dígitos
//después del punto
void redondear(double *x, unsigned n, unsigned m=0) throw(ERROR);
```

Estas declaraciones deben ser añadidas después de la última plantilla ("copiarVector") y antes de cerrar el bloque correspondiente al espacio de nombres "matrices". Observe, que al igual que en las plantillas, en los prototipos se informa con relación a los parámetros por defecto y al tipo de error generado, de esa manera el usuario de las funciones sabe no sólo qué datos mandar, sino qué tipos de errores tratar.

Luego se escribe el código de estas funciones en otro archivo: "matrices.cpp" (dentro del directorio "include"):

```
C:\HPV\programas\c++\2-2012\include>notepad++ matrices.cpp
```

```
#include <cmath>
#include "matrices.h"

namespace matrices{

int generarNumero(int li, int ls) throw(ERROR)
{
  if (li>=ls) throw ERROR_LIMITES;
  return rand()*(ls-li)/RAND_MAX+li;
}

double generarNumero(double li, double ls) throw(ERROR)
{
  if (li>=ls) throw ERROR_LIMITES;
  return rand()*1.0/RAND_MAX*(ls-li)+li;
}

void redondear(double &x, unsigned n) throw(ERROR)
```

```
{
  if (n>15) throw ERROR_REDONDEO;
  if (n==0)
    x=round(x);
  else {
    double d=1;
    for (unsigned i=0;i<n;i++)
      d*=10;
    x=round(x*d)/d;
  }
}

void redondear(double *x, unsigned n, unsigned m) throw(ERROR)
{
  if (m>15) throw ERROR_REDONDEO;
  if (n==0)
    for (unsigned i=0;i<n;i++)
      x[i]=round(x[i]);
  else {
    double d=1;
    for (unsigned i=0;i<m;i++)
      d*=10;
    for (unsigned i=0;i<n;i++)
      x[i]=round(x[i]*d)/d;
  }
}

}
```

Se compila el archivo en la forma habitual:

```
C:\HPV\programas\c++\2-2012\include>g++ -c matrices.cpp
```

Con lo que se crea el archivo objeto "matrices.o" que puede ser enlazado a cualquier programa.

Como se dijo previamente, en realidad en C/C++ las bibliotecas están conformadas por 2 o más archivos compilados y aun cuando en este caso sólo se tiene un archivo, es posible crear la librería correspondiente:

```
C:\HPV\programas\c++\2-2012\include>ar -r libmatrices.a matrices.o
ar: creating libmatrices.a
```

Con lo que se crea la biblioteca "libmatrices.a". Como se puede observar, para crear una biblioteca se emplea el comando "ar" ("archiver"), normalmente con la opción "-r" que instruye reemplazar los archivos existentes por los nuevos. Observe que el nombre de la librería inica con "lib" y termina con la extensión ".a".

Una vez creada la librearía se crea el índice de los símbolos existentes en la misma (funciones, variables y otras declaraciones) de manera que el enlazado de la librería sea mucho más rápido, ello se consigue con el comando "ranlib":

```
C:\HPV\programas\c++\2-2012\include>ranlib libmatrices.a
```

En el caso de Windows, el comando "ranlib" no es realmente necesario, pues el índice es creado automáticamente con el comando "ar".

Una vez que se tiene el archivo objeto ("matrices.o") y/o la biblioteca ("libmatrices.a"), se enlazan en los programas que así lo requieran.

Para enlazar el archivo objeto (matrices.o) se procede en la forma habitual, sólo que, como no está en el mismo directorio que el programa, se le debe dar el camino para encontrarlo, por ejemplo para enlazar esta biblioteca con el siguiente programa:

```
C:\HPV\programas\c++\2-2012\tema14>notepad++ prue2.cpp
```

```
#include <iostream>
#include <cstdlib>
using namespace std;
#include "matrices.h"
using namespace matrices;

int main(){
  int a=generarNumero(100,300);
  cout<<"Numero entero generado: "<<a<<endl;
  double r=generarNumero(10.0,100.0);
  cout<<"Numero real generado: "<<r<<endl;
  redondear(r,2);
  cout<<"Numero real redondeado: "<<r<<endl;
  double *x;
  x=generarVector(10,1.0,20.0);
  cout<<"Vector generado: "<<endl;
  mostrarVector(x,10);
  redondear(x,10,3);
  cout<<"Vector redondeado: "<<endl;
  mostrarVector(x,10);
  delete x;
}
```

Primero se compila para crear el archivo objeto ("prue2.o"):

```
C:\HPV\programas\c++\2-2012\tema14>g++ -c prue2.cpp -I../include
```

El cual puede ser enlazado con "matrices.o" (no olvidando especificar el el directorio en el que se encuentra):

```
PV\programas\c++\2-2012\tema14>g++ -o prue2 prue2.o ../include/matrices.o
```

Que al ser ejecutado devuelve:

```
C:\HPV\programas\c++\2-2012\tema14>prue2
Numero entero generado: 100
Numero real generado: 60.7227
Numero real redondeado: 60.72
Vector generado:
 4.67278 16.3661 12.1152 10.1176 7.65554 18.0233  16.634 15.1855 4.30805
 17.3199

Vector redondeado:
   4.673  16.366  12.115  10.118   7.656  18.023  16.634  15.185   4.308
   17.32
```

Alternativamente se puede enlazar empleando la biblioteca:

```
\programas\c++\2-2012\tema14>g++ -o prue2 prue2.o -L../include -lmatrices
```

Donde "-L" se emplea para indicarle al compilador el lugar donde debe buscar las bibliotecas. Como se puede observar el nombre de la librería se precede con "-l" y no se escribe ni "lib" ni la extensión ".a".

Por supuesto, ambos métodos, generan el mismo resultado, sin embargo, si en lugar de un sólo archivo objeto se tienen varios, es más eficiente emplear bibliotecas.

14.3. EJEMPLOS

1. Añada a "matrices" una función que reciba un vector de números enteros o reales y devuelva la posición (el índice) del elemento con el mayor valor absoluto. Pruebe la función encontrando el mayor de 5 números enteros leídos por teclado y la posición del mayor número real de un vector con 40 números reales generados al azar y comprendidos entre -50 y 50.

Puesto que los números pueden ser enteros o reales se crea dos funcione sobrecargadas, cuyas cabeceras (añadidas a "matrices.h") son:

```
//Devuelve el índice del elemento con el mayor valor absoluto
unsigned mayor(int *x, unsigned n);
unsigned mayor(double *x, unsigned n);
```

El código respectivo, añadido a "matrices.cpp" es:

```
unsigned mayor(int *x, unsigned n){
  unsigned k=0;
  for (unsigned i=1;i<n;i++)
    if (abs(x[i])>abs(x[k])) k=i;
  return k;
}

unsigned mayor(double *x, unsigned n){
  unsigned k=0;
  for (unsigned i=1;i<n;i++)
    if (abs(x[i])>abs(x[k])) k=i;
  return k;
}
```

Con estas modificaciones se vuelven a crear el archivo objeto "matrices.o" y la librería "libmatrices.a":

```
C:\HPV\programas\c++\2-2012\include>g++ -c matrices.cpp

C:\HPV\programas\c++\2-2012\include>ar -r libmatrices.a matrices.o
```

Entonces se crea el programa de prueba (en este ejemplo en el directorio tema14):

```
C:\HPV\programas\c++\2-2012\tema14>notepad++ ejem1.cpp
```

```
#include <iostream>
using namespace std;
#include "matrices.h"
using namespace matrices;

int main(){
  int *v;
  unsigned m;
  cout<<endl<<"Introduzca 5 números enteros:"<<endl;
  leerVector(v,5);
  m=mayor(v,5);
  cout<<endl<<"El mayor valor en el vector es: "<<v[m]<<endl;
  double *x;
  x=generarVector(40,-50.0,50.0);
  cout<<endl<<"El vector con los valores generados es:"<<endl;
  mostrarVector(x,40,0,12);
  m=mayor(x,40);
  cout<<"El mayor valor en el vector es: "<<x[m]<<endl;
```

```
  delete v;
  delete x;
  return 0;
}
```

Que se compila y se crea el ejecutable:

```
\programas\c++\2-2012\tema14>g++ -c ejem1.cpp -I../include

\programas\c++\2-2012\tema14>g++ -o ejem1 ejem1.o -L../include -lmatrices
```

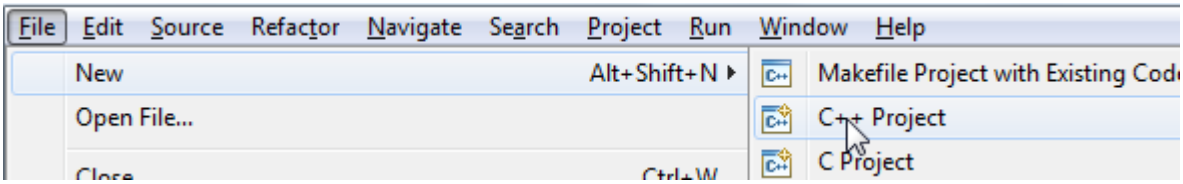
Y haciendo correr el programa, se obtiene:

```
C:\HPV\programas\c++\2-2012\tema14>ejem1

Introduzca 5 n·meros enteros:
x[0]? 1
x[1]? 2
x[2]? 3
x[3]? 2
x[4]? 1

El mayor valor en el vector es: 3

El vector con los valores generados es:
    -49.8749     6.35853    -30.6696     30.8741     8.50093    -2.0127
    -14.9709     39.5962      32.284     24.6605    -32.5892     35.8943
     21.0501      1.3535    -19.6005    -48.5015    -40.8597    -13.5548
    -35.2687    -33.4101     48.8525    -5.43077    -38.0917    -49.5331
    -49.1089     -12.212      3.1663     7.11844     10.1764     10.7166
    -33.3766     16.3045    -4.92111    -14.7877    -44.2961     10.7685
     28.3319     30.2606     1.98828     -19.805

El mayor valor en el vector es: -49.8749
```

Que, como se puede ver, devuelve los resultados esperados.

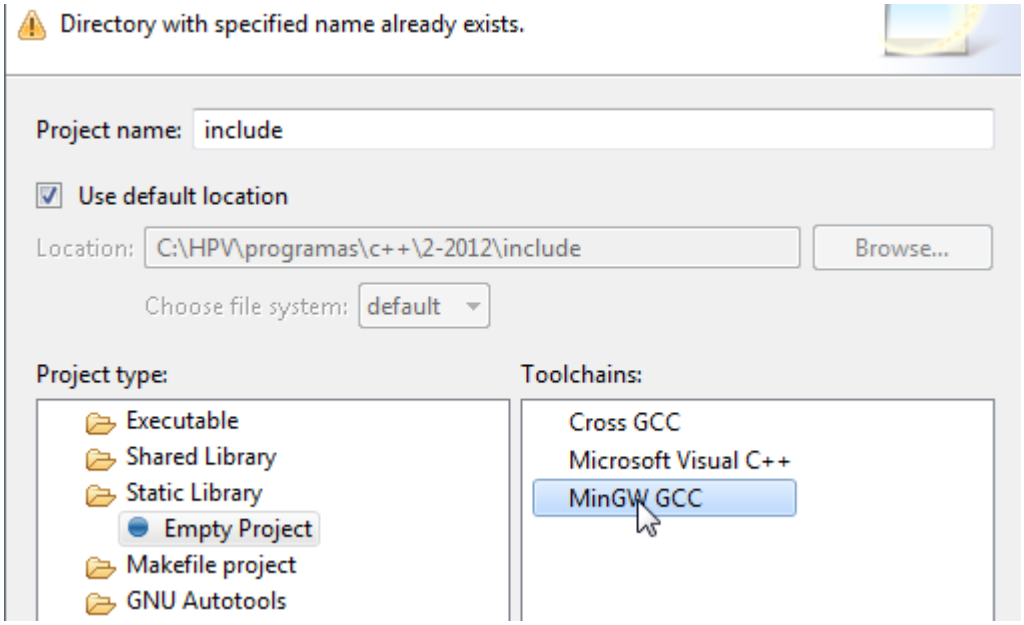
2. Añada a "matrices" una función que reciba un vector de números enteros o reales y devuelva la sumatoria de sus elementos. Pruebe la función calculando el promedio de 10 números enteros generados aleatoriamente y comprendidos entre 1 y 10 y con el promedio de 5 números reales introducidos por teclado.

En este caso el problema será resuelto empleando Eclipse. Puesto que Eclipse trabaja con proyectos y cada proyecto corresponde a un directorio, para la modificar y mantener la librería "matrices" se crea un nuevo proyecto:

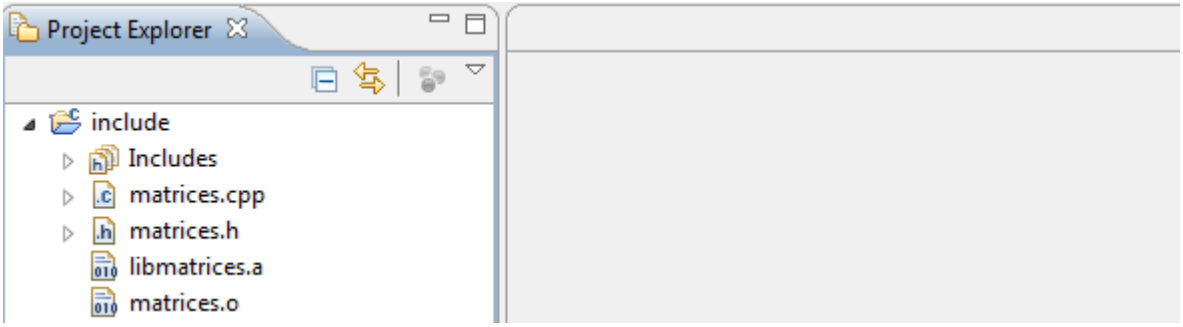


El nombre del proyecto deberá ser "include" (que corresponde al nombre del directorio ya creado para la librería "matrices") y que deberá estar ubicado en el mismo directorio donde se creó la misma, que en los ejemplos elaborados corresponde a "C:\HPV\programas\c++\2-2012".

La ventana de creación de nuevos proyectos deberá quedar aproximadamente como se muestra en la figura de la siguiente página y como se puede ver en la misma aparece una advertencia indicando que ya existe un directorio con el nombre especificado ("include"). En este caso se está creando el proyecto, a propósito, en el directorio existente (para seguir trabajando con los archivos ya creados), por lo que dicha advertencia puede ser ignorada.



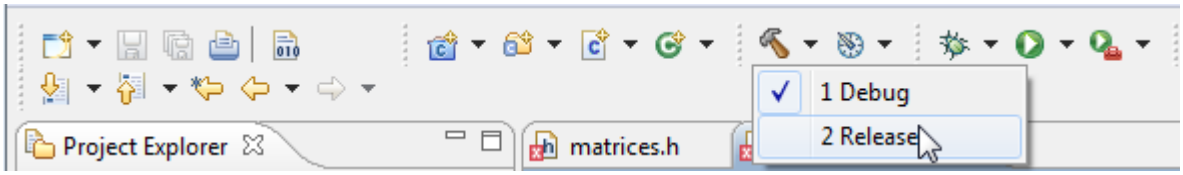
Se crea un proyecto de tipo biblioteca estática, porque ese es el tipo de biblioteca que se está empleando en la solución de los problemas. Entonces en la ventana del explorador de proyectos aparecerá algo similar a lo siguiente:



Donde, como se puede ver, están los archivos creados desde la línea de comando, además de una serie de directorios "Include" que corresponde a los directorios donde se encuentran los archivos de cabecera proporcionados conjuntamente el compilador.

En Eclipse, los archivos compilados se guardan por defecto en el directorio "Debug" (porque por defecto se compila en el modo de depuración) y en el directorio "Release" si se ha compilado en el modo de distribución (el modo normal). El modo de depuración equivale a la depuración con la opción "-g" desde la línea de comando y es el modo por defecto porque mientras se elabora un programa es frecuente emplear las herramientas de depuración.

Para compilar en el modo de distribución (el modo normal), simplemente se elige dicha opción:



Se debe compilar en este modo una vez que el programa ha sido depurado y probado.

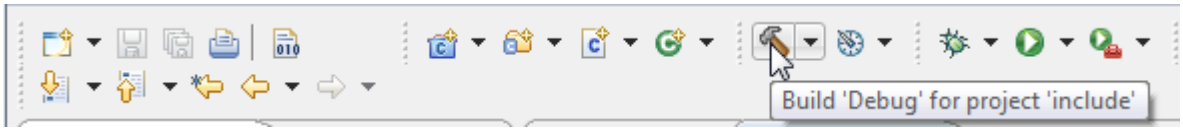
Para abrir cualquiera de los archivos de este proyecto, simplemente se hace doble clic en el mismo. Procediendo de esta manera, se añaden los siguientes prototipos al archivo de cabecera "matrices.h":

```
//Devuelve la sumatoria de los "n" elementos del vector "x"
long long int sum(int *x, unsigned n);
double sum(double *x, unsigned n);
```

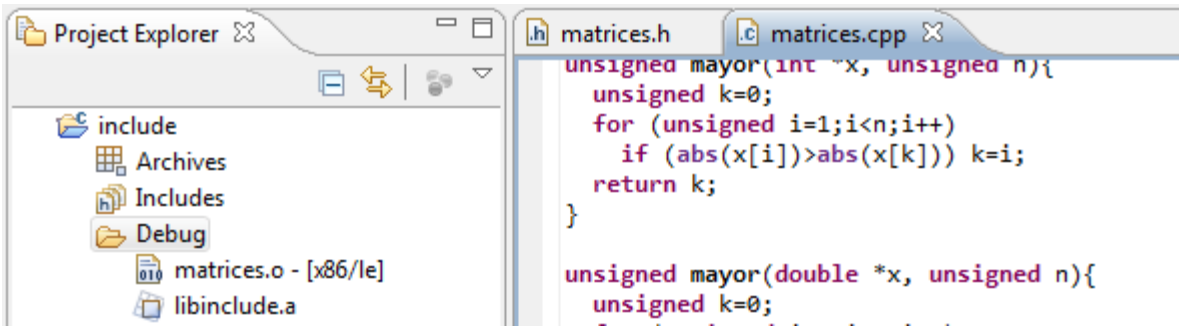
Y el código respectivo al archivo fuente "matrices.cpp":

```
long long int sum(int *x, unsigned n){
                          long long int s=0;
                  for (unsigned i=0;i<n;i++) s+=x[i];
                          return s;
}

double sum(double *x, unsigned n){
                         double s=0;
                  for (unsigned i=0;i<n;i++) s+=x[i];
                          return s;
}
```

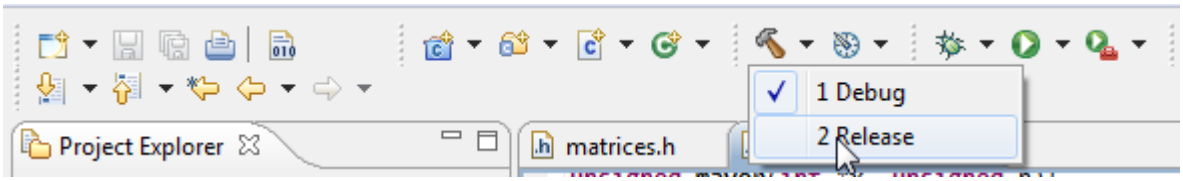
Y se compila la librería:



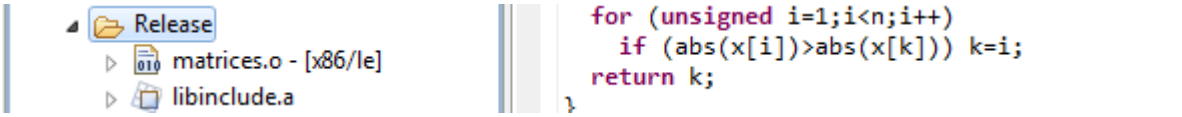
Entonces en el explorador del proyecto aparece el directorio "Debug" y dentro del mismo el archivo objeto "matrices.o" y la librería "libinclude.a" (Eclipse nombra a la biblioteca igual que al proyecto):



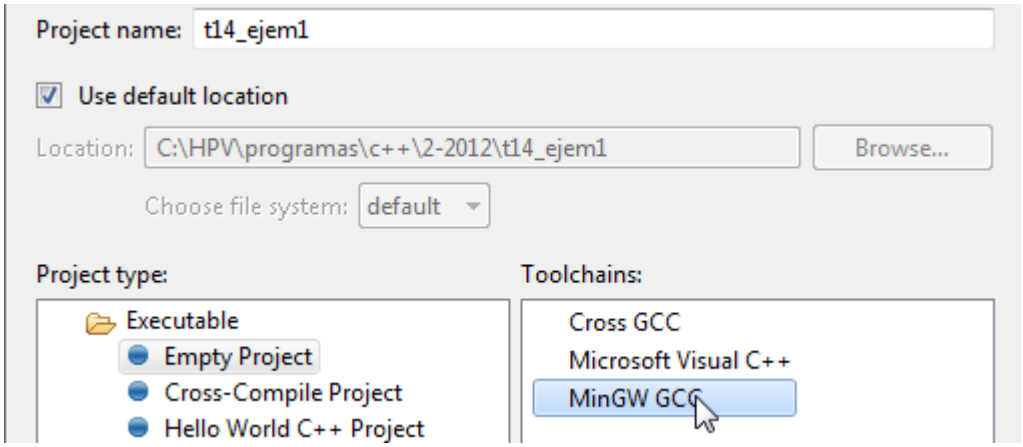
Si se compila en el modo de distribución ("release"):



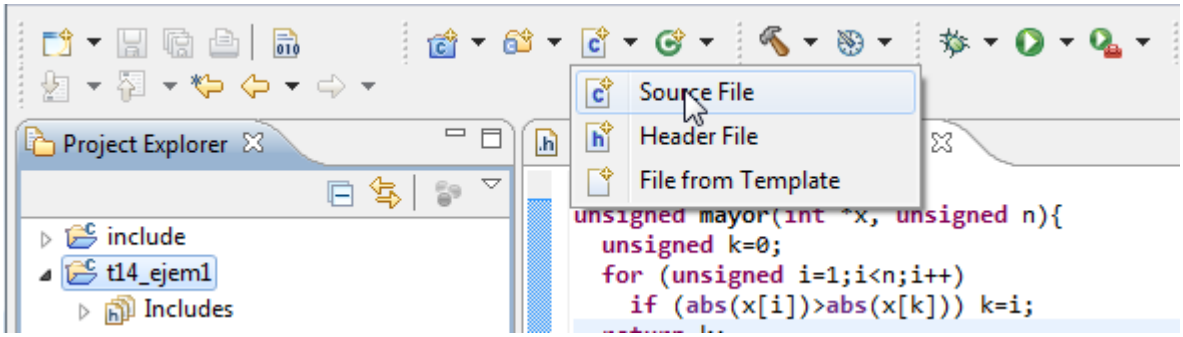
Se crea el directorio "Release" y dentro del mismo aparecen igualmente el archivo objeto "matrices.o" y la librería "libinclude.a", pero en un modo más compacto pues en los mismos no se guarda la información requerida para la depuración (3.84Kb comparados con 127Kb en el modo Debug).

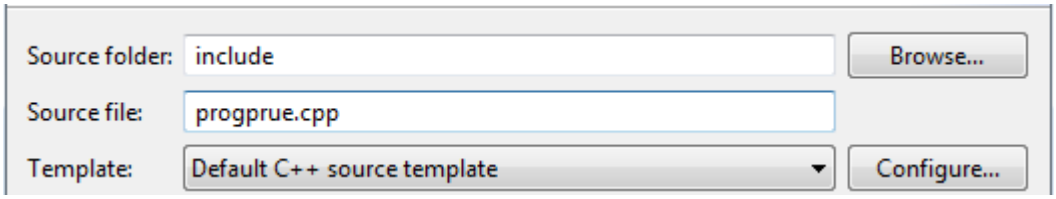


Ahora, se crea un nuevo proyecto para el programa de prueba:



Que, como todos los proyectos Eclipse, crea el directorio respectivo ("t14_ejem1") inicialmente vacío. Entonces se añade el programa de prueba "progprue.cpp":

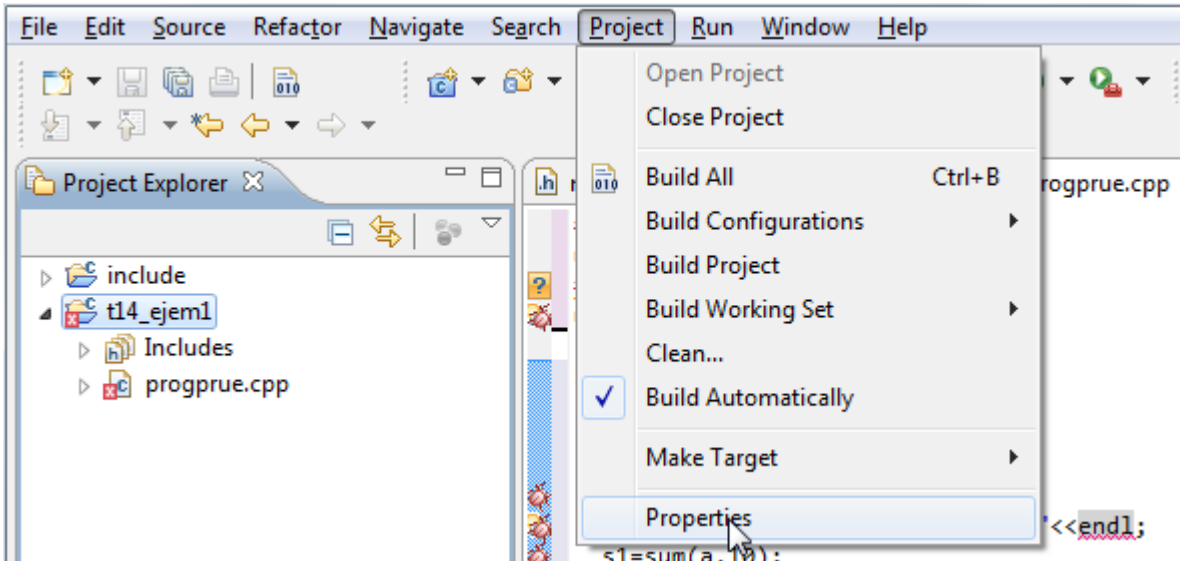




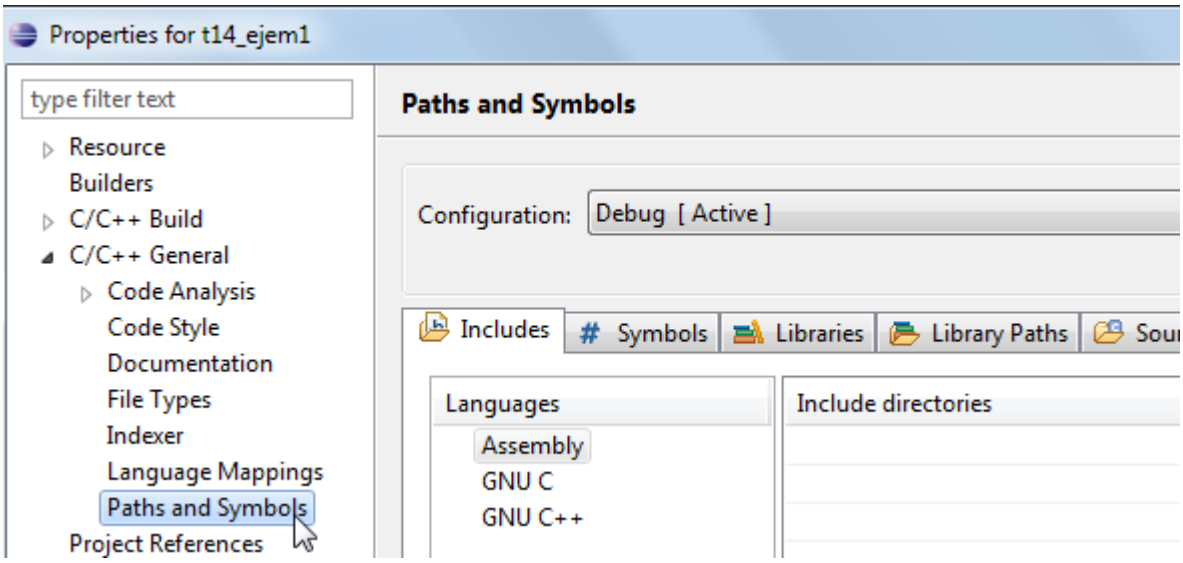
En el cual se escribe el código de prueba:

```
#include <iostream>
using namespace std;
#include "matrices.h"
using namespace matrices;

int main(){
  int *a;
  long long int s1;
  double *x,s2;
  a=generarVector(10,1,10);
  cout<<endl<<"Elementos a sumar:"<<endl;
  mostrarVector(a,10);
  s1=sum(a,10);
  cout<<"Sumatoria: "<<s1<<endl;
  cout<<endl<<"Números reales a sumar:"<<endl;
  leerVector(x,5);
  s2=sum(x,5);
  cout<<endl<<"sumatoria: "<<s2<<endl<<endl;
  return 0;
}
```

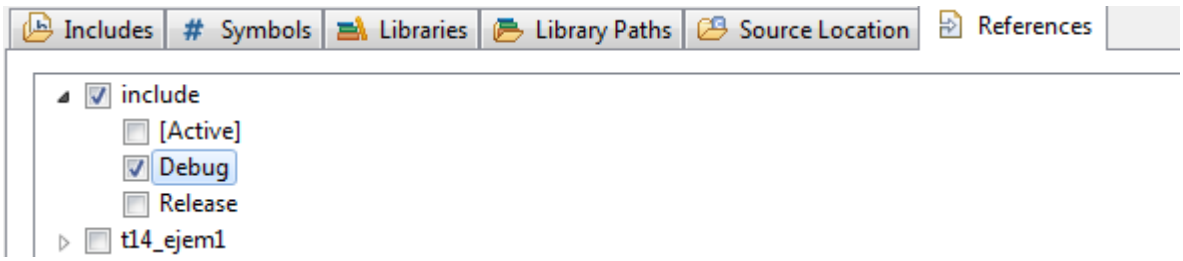
Que reporta muchos errores porque no puede encontrar el archivo de cabecera "matrices.h", debido a que se encuentra en otro proyecto. Para que Eclipse sepa donde buscar estos archivos, se modifican las propiedades del proyecto:



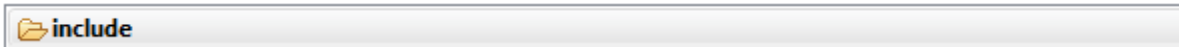
Y dentro del mismo modificar las opciones de "Path and Symbols":



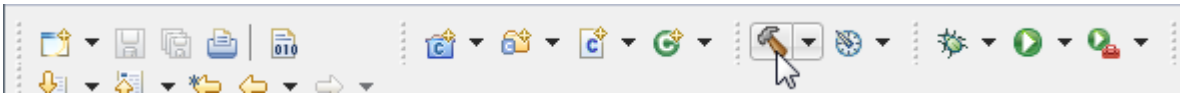
En esta página se elige la pestaña "References" y en la misma se selecciona la referencia "Debug" dentro de "include":



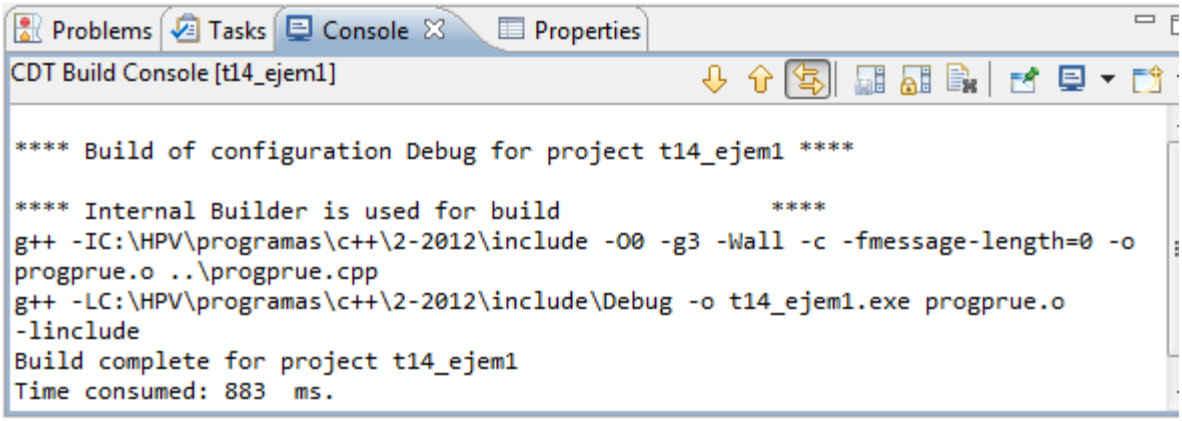
Con lo que la referencia respectiva se añade en las pestañas "Includes", "Libraries" y "Library Paths". Por ejemplo a la pestaña "Libraries" se añade:



Ahora, con esta modificación puede compilar el programa:



Con lo que el programa compila correctamente y en la consola se puede ver una información similar a la siguiente:



Que con excepción de las opciones -O0, -g3, -Wall y -fmessage-length=0 son las mismas instrucciones empleadas desde la línea de comando para compilar y enlazar el programa.

Haciendo correr el programa, se obtiene:

```
Elementos a sumar:
        1       6       2       8       6       5       4       9       8
        7

Sumatoria: 56

Números reales a sumar:
x[0]? 2.1
x[1]? 3.5
x[2]? 4.3
x[3]? 3.9
x[4]? 5.3

sumatoria: 19.1
```

Que son los resultados correctos.

3. Añada a "matrices" una función que reciba dos vector de números enteros o reales, los sume elemento a elemento y devuelva el vector resultante. La función debe generar un error si los dos vectores no tienen el mismo número de elementos. Pruebe la función sumando dos vectores de 7 números enteros leídos por teclado y dos vectores de 300 números reales generados al azar y comprendidos entre 0 y 100, pruebe también la función multiplicando dos vectores con 5 y 7 números enteros generados al azar y comprendidos entre 1 y 10.

Este problema se resuelve también en Eclipse y para ello se sigue el mismo procedimiento que en el anterior ejemplo.

La constante de error añadida a la cabecera "matrices.h" (después de la última constante de error) es:

```
//Matrices o vectores con dimensiones no compatibles
const ERROR ERROR_DIMENSIONES=3;
```

Y los prototipos de las funciones (añadidos al final del archivo) son:

```
//Suma los "n" elementos de los vectores "x" y "y"
int *sum(int *x, unsigned nx, int *y, unsigned ny) throw(ERROR);
double *sum(double *x, unsigned nx, double *y, unsigned ny) throw(ERROR);
```

El código respectivo, añadido al final del archivo "matrices.cpp" es:

```
int *sum(int *x, unsigned nx, int *y, unsigned ny) throw(ERROR)
{
  if (nx!=ny) throw ERROR_DIMENSIONES;
  int *z = new int[nx];
  for (unsigned i=0;i<nx;i++)
    z[i]=x[i]+y[i];
  return z;
}

double *sum(double *x, unsigned nx, double *y, unsigned ny) throw(ERROR)
{
  if (nx!=ny) throw ERROR_DIMENSIONES;
  double *z = new double[nx];
  for (unsigned i=0;i<nx;i++)
    z[i]=x[i]+y[i];
  return z;
}
```

Se guardan estas modificaciones y se construye la librería (con Build). Entonces se crea el programa de prueba (un nuevo proyecto) con el nombre t14_ejem2 y se añade al mismo el archivo "progprue.cpp", no olvidando modificar las propiedades del proyecto para incluir la referencia a "include" (igual que se hizo en el anterior ejemplo).

El código escrito en el archivo "progprue.cpp" es:

```
#include "matrices.h"
using namespace matrices;
#include <iostream>
using namespace std;

int main(){
  int *a,*b,*c,*d,*e,*f,err;
  double *x,*y,*z;
  try{
    cout<<endl<<"Elementos del primer vector: "<<endl;
    leerVector(a,7);
    cout<<endl<<"Elementos del segundo vector: "<<endl;
    leerVector(b,7);
    c=sum(a,7,b,7);
    cout<<endl<<"Vector resultante: "<<endl;
    mostrarVector(c,7);
    x=generarVector(300,0.0,100.0);
    cout<<"Elementos del primer vector de números reales: "<<endl;
    mostrarVector(x,300,0,12);
    y=generarVector(300,0.0,100.0);
    cout<<"Elementos del segundo vector de números reales: "<<endl;
    mostrarVector(y,300,0,12);
```

```
    z=sum(x,300,y,300);
    cout<<"Elementos del vector resultante: "<<endl;
    mostrarVector(z,300,0,12);
    d=generarVector(5,1,10);
    cout<<"Elementos del primer vector de números enteros: "<<endl;
    mostrarVector(d,5);
    e=generarVector(7,1,10);
    cout<<"Elementos del segundo vector de n´meros enteros: "<<endl;
    mostrarVector(e,7);
    f=sum(d,5,e,7);
    cout<<"Elementos del vector resultante: "<<endl;
    mostrarVector(f,7);
  } catch(ERROR er){
    if (er==ERROR_DIMENSIONES) {
      cout<<"Error: Vectores no compatibles."<<endl;
      err=1;
    }
  }
  delete a;
  delete b;
  delete c;
  delete d;
  delete e;
  delete x;
  delete y;
  delete z;
  if (!err) delete f;
  return 0;
}
```

Guardando, construyendo y haciendo correr el programa se obtiene:

```
Elementos del primer vector:
x[0]? 1
x[1]? 2
x[2]? 3
x[3]? 4
x[4]? 5
x[5]? 6
x[6]? 7

Elementos del segundo vector:
x[0]? 2
x[1]? 3
x[2]? 4
x[3]? 5
x[4]? 6
x[5]? 7
x[6]? 8

Vector resultante:
       3       5       7       9      11      13      15

Elementos del primer vector de números reales:
    0.125126     56.3585     19.3304     80.8741     58.5009     47.9873
     35.0291     89.5962      82.284     74.6605     17.4108     85.8943
     71.0501     51.3535     30.3995     1.49846     9.14029     36.4452
     14.7313     16.5899     98.8525     44.5692     11.9083    0.466933
     0.89114      37.788     53.1663     57.1184     60.1764     60.7166
```

16.6234	66.3045	45.0789	35.2123	5.70391	60.7685
78.3319	80.2606	51.9883	30.195	87.5973	72.6676
95.5901	92.5718	53.9354	14.2338	46.2081	23.5328
86.2239	20.9601	77.9656	84.3654	99.6796	99.9695
61.1499	39.2438	26.6213	29.7281	84.0144	2.37434
37.5866	9.26237	67.7206	5.62151	0.878933	91.879
27.5887	27.2897	58.7909	69.1183	83.7611	72.6493
48.4939	20.5359	74.3736	46.8459	45.7961	94.9156
74.4438	10.828	59.9048	38.5235	73.5008	60.8966
57.2405	36.1339	15.1555	22.5105	42.5153	80.2881
51.7106	98.999	75.1549	34.5561	16.8981	65.7308
49.1897	6.35395	69.9759	50.4807	14.7496	94.9583
14.1575	90.5118	69.2892	30.3049	42.6557	7.03757
96.6613	68.3187	15.3233	87.7255	82.168	58.2049
19.1351	17.7892	81.7194	47.5265	15.5553	50.3922
73.2017	40.5591	27.958	56.8743	68.2241	75.5852
72.1915	47.5295	12.302	36.7809	83.4681	3.50963
51.7014	66.2984	42.6222	10.4678	94.9339	92.1384
54.9547	34.5988	47.1725	37.4981	84.698	31.6874
45.6099	27.1889	98.2971	29.78	73.9189	56.7278
19.599	76.1315	83.9442	39.7656	50.09	89.0164
2.74667	99.4629	57.2588	5.05081	53.1327	19.4067
84.3043	62.6759	65.7613	19.7851	84.2158	12.3325
10.9928	74.3126	31.4066	94.1069	28.6081	33.6314
14.0263	73.3085	83.462	70.7999	60.0238	74.7215
25.2724	14.4475	0.161748	6.10065	80.6238	85.2626
21.0578	11.5604	55.3209	1.42521	11.3773	45.4512
75.222	68.6148	54.3443	7.38853	43.672	20.1941
69.6219	29.0353	43.6689	23.2429	57.7868	53.2579
62.8681	16.0192	50.4135	96.3042	69.5761	92.4802
18.9947	33.5948	17.835	99.5178	45.7442	99.8016
9.75066	62.5172	9.43937	43.7727	93.1516	4.84329
89.462	29.0017	22.7302	76.9066	41.0718	20.1971
62.8071	60.4144	45.1613	46.6353	59.7827	63.4724
85.4793	82.8791	62.4775	72.0908	56.5752	37.5134
18.4271	73.7907	55.5132	90.5087	24.2866	18.894
60.4724	69.8508	58.4613	35.1299	49.4461	8.03858
74.0745	61.2049	62.038	69.1122	80.4529	14.9113
57.6037	86.7733	91.1557	61.4704	72.7683	4.32142
66.7776	97.6531	31.5012	56.9201	30.5826	17.3925
10.8554	86.9045	85.1222	74.4316	15.4881	32.6914
7.93481	7.66015	64.098	82.0002	54.5091	44.8256
40.8979	29.8746	46.556	50.1205	15.2654	32.3038
73.7999	31.3883	82.6685	95.9075	87.3348	72.5028
30.0058	94.3999	12.7232	6.57369	78.4967	52.4583

Elementos del segundo vector de números reales:

60.9638	95.6114	7.22678	87.5637	65.3859	32.2123
10.4801	50.5051	22.7088	29.0292	91.998	55.1164
66.2801	11.4536	49.2538	37.9131	49.6811	79.3359
50.9262	38.2366	68.8162	53.2151	60.6281	39.5184
0.589007	70.7877	10.062	62.3066	86.3247	49.1501
74.7337	49.6902	38.0108	78.5363	55.2812	35.7097
95.5718	63.0848	17.658	37.4248	13.1626	74.3278
95.172	61.1988	2.78329	32.9844	5.59099	63.921
13.1626	84.7072	86.4315	59.6881	72.1641	85.3969
1.46794	12.6469	70.7907	61.7145	21.7566	6.59505

16.892	62.4104	34.0983	31.9407	36.7565	66.1
80.2393	80.6879	52.6536	61.1103	79.8181	90.0601
14.481	63.0177	40.2417	25.37	13.654	85.519
6.61641	42.7808	57.3351	30.2286	54.8051	22.5562
31.135	11.063	80.8039	13.4709	28.4249	78.811
89.523	78.9636	74.3797	61.5223	36.1126	85.6655
22.8492	86.3582	22.9438	24.955	54.2405	98.4832
5.38041	8.14234	52.4674	42.6801	9.46684	25.8797
89.1537	23.2765	14.655	12.5095	93.1639	8.01111
4.70901	5.87176	33.6406	91.4701	39.8602	43.2783
94.6165	83.7184	53.4227	84.2097	69.3533	39.7687
25.9163	0.433363	52.559	95.4802	39.8694	24.1096
58.5559	25.5135	68.4011	94.528	43.5499	89.0225
0.717185	94.0977	60.155	78.6157	57.6678	14.243
22.2327	38.3007	0.427259	41.792	8.22474	65.9932
85.5098	6.48518	81.106	66.2069	69.1488	80.2698
53.0137	68.5629	14.2766	68.9505	72.7897	77.7734
3.10678	86.8679	64.452	70.6565	8.54518	55.1988
94.7905	5.87786	27.4972	14.5177	98.178	61.9984
29.2245	92.2483	36.7534	69.454	21.8635	15.5919
24.0547	52.1439	90.228	10.6418	90.2646	44.1725
8.01111	78.2098	17.1789	97.4395	77.5872	87.0388
21.0639	45.6618	0.375378	75.0633	11.4048	40.4706
31.1136	99.2615	3.85754	25.2083	18.9276	24.7688
15.3508	62.0319	88.5372	93.9085	19.5654	77.9656
64.5558	65.6758	90.9146	45.5458	92.1293	66.4174
15.0761	63.6341	56.9536	42.204	94.3022	54.0574
57.857	53.6454	25.5287	39.6954	35.0261	3.66222
79.5251	19.6509	7.02841	38.9416	59.0747	7.0925
19.7668	15.5889	64.4337	45.4329	60.4297	69.7348
44.1298	68.4469	39.6527	83.5719	59.2212	19.9591
94.9461	87.6003	39.1705	98.7457	18.5308	89.5718
57.445	44.2061	62.7338	70.8518	4.96231	28.5562
26.0201	40.7636	89.5322	71.0196	72.8446	89.6054
39.3567	39.7412	90.3867	30.8603	38.8104	57.0574
35.3557	73.3695	74.5354	73.6381	73.9433	13.9317
20.0171	27.2591	68.0441	91.113	36.784	51.7655
10.9378	90.7437	20.307	51.796	65.4836	43.852
68.7643	9.04263	7.95618	7.57775	2.78024	35.5083
30.723	69.7287	14.2674	39.4848	6.78426	67.5741

Elementos del vector resultante:

61.0889	151.97	26.5572	168.438	123.887	80.1996
45.5092	140.101	104.993	103.69	109.409	141.011
137.33	62.8071	79.6533	39.4116	58.8214	115.781
65.6575	54.8265	167.669	97.7844	72.5364	39.9854
1.48015	108.576	63.2282	119.425	146.501	109.867
91.3572	115.995	83.0897	113.749	60.9851	96.4782
173.904	143.345	69.6463	67.6199	100.76	146.995
190.762	153.771	56.7186	47.2182	51.7991	87.4538
99.3866	105.667	164.397	144.053	171.844	185.366
62.6179	51.8906	97.412	91.4426	105.771	8.96939
54.4786	71.6727	101.819	37.5622	37.6354	157.979
107.828	107.978	111.444	130.229	163.579	162.709
62.9749	83.5536	114.615	72.2159	59.4501	180.435
81.0602	53.6088	117.24	68.7521	128.306	83.4529
88.3755	47.1969	95.9593	35.9813	70.9403	159.099

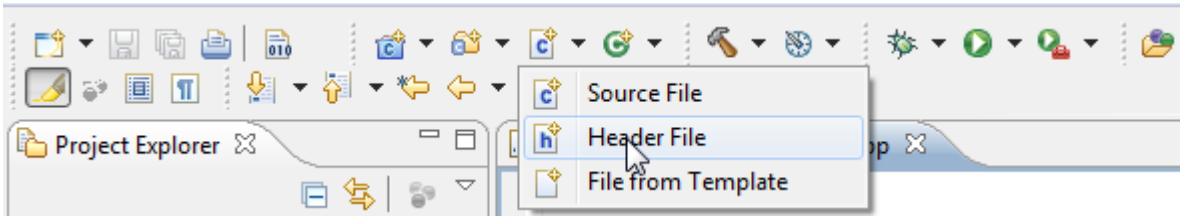
```
141.234     177.963     149.535     96.0784     53.0107     151.396
72.0389     92.7122     92.9197     75.4357     68.9901     193.442
19.5379     98.6541     121.757      72.985     52.1226     32.9173
185.815     91.5952     29.9783     100.235     175.332      66.216
23.8441      23.661      115.36     138.997     55.4155     93.6705
167.818     124.277     81.3807     141.084     137.577     115.354
98.1079     47.9629      64.861     132.261     123.338     27.6193
110.257     91.8119     111.023     104.996     138.484     181.161
55.6719     128.697     107.327     116.114     142.366     45.9304
67.8426     65.4897     98.7243      71.572     82.1436     122.721
105.109     82.6167      165.05     105.972     119.239     169.286
55.7604     168.026     71.5354     74.0013     125.922     97.1801
87.4111     149.544     130.213     90.4416      92.761     67.5314
105.783     80.1904     58.9038     108.625     126.786     95.6297
43.2508     165.557     120.215     140.254     81.8873     90.3134
49.3271     66.5914     90.3897     16.7425     170.888     129.435
29.0689     89.7702     72.4998     98.8647     88.9645      132.49
96.2859     114.277     54.7197     82.4519     55.0768     60.6647
100.735     128.297     47.5265     48.4512     76.7144     78.0267
78.2189     78.0511     138.951     190.213     89.1415     170.446
83.5505     99.2706      108.75     145.064     137.873     166.219
24.8268     126.151      66.393     85.9767     187.454     58.9007
147.319     82.6472     48.2589     116.602     76.0979     23.8594
142.332     80.0653     52.1897      85.577     118.857     70.5649
105.246      98.468     126.911     117.524     117.005     107.248
62.5568     142.238     95.1659     174.081     83.5078     38.8531
155.419     157.451     97.6318     133.876     67.9769     97.6104
 131.52     105.411     124.772     139.964     85.4152     43.4675
83.6238     127.537     180.688      132.49     145.613     93.9268
106.134     137.394     121.888     87.7804      69.393     74.4499
46.2111     160.274     159.658      148.07     89.4314     46.6231
27.9519     34.9193     132.142     173.113     91.2931     96.5911
51.8357     120.618      66.863     101.917     80.7489     76.1559
142.564     40.4309     90.6247     103.485     90.1151     108.011
60.7288     164.129     26.9906     46.0585     85.2809     120.032


Elementos del primer vector de números enteros:
       7       2       7       6       6

Elementos del segundo vector de n´meros enteros:
       5       1       5       1       9       5       9

Error: Vectores no compatibles.
```

Que son los resultados esperados.

4. Cree una biblioteca con una función que reciba los elementos de un vector de números enteros o reales y devuelva la desviación estándar de los mismos. Pruebe la función calculando la desviación estándar de 10 números enteros leídos por teclado y la desviación estándar de 500 números reales generados al azar y comprendidos entre 1 y 200. La desviación estándar está definida por la siguiente ecuación.

$$dstd = \sqrt{\frac{\sum_{i=1}^{n} \left(\overline{x} - x_i\right)^2}{n-1}}$$

Este ejemplo se resolverá también en Eclipse. Primero se crea un nuevo proyecto del tipo biblioteca estática (Static Library) con el nombre t14_lib_ejem4 y a la misma se le añade un archivo de cabeceracon el nombre "ejem4.h":



En el cual se escriben los prototipos de la función:

```
#ifndef EJEM4_H_
#define EJEM4_H_

//Calcula la desviación estándar de los "n" datos contenidos en el
//vector x
double desvstd(int *x, unsigned n) throw(int);
double desvstd(double *x, unsigned n) throw(int);

#endif
```

Observe que en este casos no se define un nuevo tipo de dato sino que se genera un error de tipo entero. Entonces se añade el archivo fuente ("source file") "ejem4.cpp" y en el mismo se escribe el código de las funciones:

```
#include <cmath>
#include "ejem4.h"

int cuad(int x){
  return x*x;
}

double cuad(double x){
  return x*x;
}

double desvstd(int *x, unsigned n) throw(int)
{
  if (n==0) throw 1;//Error vector vacío
  double p=0,s=0;
  for (unsigned i=0;i<n;i++)
    p+=x[i];
  p/=n;//Promedio de los elementos
  for (unsigned i=0;i<n;i++)
    s+=cuad(p-x[i]);
  return sqrt(s/(n-1));
}

double desvstd(double *x, unsigned n) throw(int)
{
  if (n==0) throw 1;//Error vector vacío
  double p=0,s=0;
  for (unsigned i=0;i<n;i++)
    p+=x[i];
  p/=n;//Promedio de los elementos
  for (unsigned i=0;i<n;i++)
```

```
    s+=cuad(p-x[i]);
  return sqrt(s/(n-1));
}
```

Note que a más del código de las dos funciones se han creado otras dos (que no son visibles desde otros programas) para calcular el cuadrado de un número real o entero.

Se crea la librería (construyéndola con Build), luego se crea un nuevo proyecto para el programa de prueba con el nombre "t4_ejem4" y se añade al mismo un archivo fuente ("source file") con el nombre "progprue.cpp". Se modifican las propiedades del proyecto de manera que haga referencia a las carpetas "Debug" tanto de "include" como de "t14_lib_ejem4" (para que el programa pueda emplear ambas librerías) y se escribe en el archivo fuente el siguiente código:

```
#include <iostream>
using namespace std;
#include "ejem4.h"
#include "matrices.h"
using namespace matrices;

int main(){
  int *a;
  double *x,d1,d2;
  try{
    cout<<endl<<"Datos del primer vector:"<<endl;
    leerVector(a,10);
    d1=desvstd(a,10);
    cout<<"Desviación estándar: "<<d1<<endl;
    cout<<endl<<"Vector de números reales:"<<endl;
    x=generarVector(500,1.,200.);
    mostrarVector(x,500,0,12);
    d2=desvstd(x,500);
    cout<<"Desviación estándar: "<<d2<<endl;
    double *y;
    desvstd(y,0);
  } catch(int er){
    cout<<endl<<"Error: Se ha mandado un vector vacío."<<endl;
  }
  delete a;
  delete x;
  return 0;
}
```

Finalmente se crea el ejecutable (con Build) y haciendo correr el programa se obtiene:

```
Datos del primer vector:
x[0]? 1
x[1]? 2
x[2]? 3
x[3]? 4
x[4]? 5
x[5]? 6
x[6]? 7
x[7]? 8
x[8]? 9
x[9]? 10
Desviación estándar: 3.02765
```

Vector de números reales:

1.249	113.153	39.4675	161.939	117.417	96.4947
70.708	179.297	164.745	149.574	35.6475	171.93
142.39	103.193	61.495	3.98193	19.1892	73.526
30.3153	34.0138	197.716	89.6928	24.6976	1.9292
2.77337	76.1982	106.801	114.666	120.751	121.826
34.0806	132.946	90.707	71.0724	12.3508	121.929
156.88	160.719	104.457	61.0881	175.319	145.609
191.224	185.218	108.331	29.3253	92.9541	47.8303
172.586	42.7106	156.152	168.887	199.362	199.939
122.688	79.0951	53.9764	60.1589	168.189	5.72494
75.7973	19.4321	135.764	12.1868	2.74908	183.839
55.9016	55.3064	117.994	138.545	167.685	145.572
97.5029	41.8665	149.003	94.2234	92.1342	189.882
149.143	22.5477	120.211	77.6618	147.267	122.184
114.909	72.9065	31.1594	45.7958	85.6055	160.773
103.904	198.008	150.558	69.7667	34.6272	131.804
98.8876	13.6444	140.252	101.457	30.3517	189.967
29.1735	181.118	138.886	61.3067	85.8849	15.0048
193.356	136.954	31.4935	175.574	164.514	116.828
39.0789	36.4006	163.622	95.5777	31.955	101.28
146.671	81.7126	56.6364	114.18	136.766	151.415
144.661	95.5838	25.481	74.194	167.102	7.98416
103.886	132.934	85.8181	21.831	189.919	184.355
110.36	69.8517	94.8732	75.6212	169.549	64.0579
91.7637	55.106	196.611	60.2621	148.099	113.888
40.002	152.502	168.049	80.1336	100.679	178.143
6.46587	198.931	114.945	11.0511	106.734	39.6194
168.766	125.725	131.865	40.3724	168.589	25.5417
22.8756	148.882	63.4991	188.273	57.93	67.9265
28.9124	146.884	167.089	141.892	120.447	149.696
51.292	29.7505	1.32188	13.1403	161.441	170.673
42.905	24.0052	111.089	3.83618	23.6408	91.4479
150.692	137.543	109.145	15.7032	87.9073	41.1863
139.548	58.7803	87.9012	47.2534	115.996	106.983
126.108	32.8781	101.323	192.645	139.456	185.036
38.7995	67.8536	36.4917	199.04	92.0309	199.605
20.4038	125.409	19.7844	88.1077	186.372	10.6381
179.029	58.7135	46.2331	154.044	82.7329	41.1923
125.986	121.225	90.871	93.8043	119.968	127.31
171.104	165.929	125.33	144.461	113.585	75.6516
37.6699	147.844	111.471	181.112	49.3304	38.5991
121.34	140.003	117.338	70.9084	99.3977	16.9968
148.408	122.798	124.456	138.533	161.101	30.6736
115.631	173.679	182.4	123.326	145.809	9.59963
133.887	195.33	63.6874	114.271	61.8594	35.6111
22.6023	173.94	170.393	149.119	31.8214	66.0559
16.7903	16.2437	128.555	164.18	109.473	90.2029
82.3867	60.4504	93.6464	100.74	31.3781	65.2846
147.862	63.4627	165.51	191.856	174.796	145.281
60.7115	188.856	26.3191	14.0816	157.208	105.392
122.318	191.267	15.3813	175.252	131.118	65.1025
21.8553	101.505	46.1906	58.7681	184.076	110.682
132.897	23.7927	99.0151	76.4472	99.8654	158.878
102.343	77.0909	137.944	106.898	121.65	79.6417
2.17212	141.867	21.0233	124.99	172.786	98.8086
149.72	99.8836	76.6415	157.287	111.01	72.0623

```
191.188     126.539     36.1394     75.4754     27.1936     148.912
190.392     122.786     6.53874      66.639     12.1261     128.203
27.1936     169.567     172.999     119.779     144.606      170.94
 3.9212     26.1673     141.874     123.812     44.2957     14.1241
34.6151     125.197     68.8557     64.5619     74.1454     132.539
160.676     161.569     105.781     122.609     159.838      180.22
29.8173     126.405      81.081     51.4864     28.1714     171.183
14.1667     86.1339     115.097     61.1549     110.062     45.8869
62.9586     23.0153       161.8      27.807     57.5656     157.834
179.151     158.138     149.016     123.429      72.864     171.474
46.4699     172.853     46.6582     50.6604     108.939     196.982
 11.707     17.2033      105.41     85.9335      19.839     52.5006
178.416     47.3202     30.1634      25.894     186.396     16.9421
10.3709     12.6848     67.9447     183.025     80.3218     87.1238
189.287       167.6     107.311     168.577     139.013     80.1397
52.5735     1.86239     105.592     191.006     80.3401     48.9781
117.526     51.7718     137.118     189.111     87.6643     178.155
 2.4272     188.254     120.709     157.445     115.759     29.3435
45.2431     77.2185     1.85025     84.1662     17.3672     132.327
171.165     13.9055     162.401     132.752     138.606     160.737
106.497      137.44     29.4104     138.211     145.851     155.769
 7.1825     173.867      129.26     141.606     18.0049     110.846
189.633      12.697     55.7194     29.8901     196.374     124.377
59.1568     184.574     74.1393     139.214     44.5083     32.0279
48.8688     104.766     180.554     22.1772     180.627     88.9033
16.9421     156.637     35.1859     194.905     155.399     174.207
42.9171      91.867       1.747     150.376     23.6955     81.5365
62.9161      198.53
```

Desviación estándar: 57.5139

Error: Se ha mandado un vector vacío.

Que son los resultados esperados.

14.4. EJERCICIOS

Los siguientes ejercicios pueden ser resueltos, según se prefiera, en Eclipse o con un editor de texto y la línea de comando. En todo caso, los ejercicios deben estar claramente identificados por su nombre y deben agruparse bajo un mismo directorio.

1. Añada a "matrices" una función que reciba un vector de números enteros o reales y devuelva el índice del elemento con el menor valor absoluto. Pruebe la función encontrando el menor de 10 números enteros leídos por teclado y el menor de 50 números reales generados al azar y comprendidos entre -70 y 60.

2. Añada a "matrices" una función que reciba un vector de números enteros o reales y devuelva el valor absoluto del vector, es decir la raíz cuadrada la sumatoria de los cuadrados de sus elementos. Pruebe la función calculando el valor absoluto de 6 números reales leídos por teclado y el valor absoluto de 200 números enteros generados al azar y comprendidos entre -100 y 100

3. Añada a "matrices" una función que reciba dos vectores de números enteros o reales, los multiplique elemento a elemento y devuelva el vector resultante de la multiplicación. La función debe generar un error si los

dos vectores no tienen el mismo número de elementos. Pruebe la función multiplicando dos vectores de 5 números enteros leídos por teclado, dos vectores con 150 números reales generados al azar y comprendidos entre -100 y 100 y con dos vectores con 6 y 10 números enteros generados al azar y comprendidos entre 1 y 20.

4. Añada a "matrices" una plantilla que reciba un vector e invierta el orden en que se encuentran sus elementos. Pruebe la plantilla invirtiendo un vector con 20 números enteros generados al azar y comprendidos entre -10 y 10 e invirtiendo un vector con 50 números reales generados al azar y comprendidos entre 0 y 20.

15. MATRICES 3

Continuando con el estudio de matrices, en este tema se estudian las matrices que tienen más de una dimensión, más específicamente las matrices con 2 dimensiones, pues son las matrices uso más frecuente en la práctica.

15.1. DECLARACIÓN Y USO DE MATRICES MULTIDIMENSIONALES

En C/C++, las matrices multidimensionales son vectores de vectores. En forma estática se declaran de acuerdo a la siguiente sintaxis:

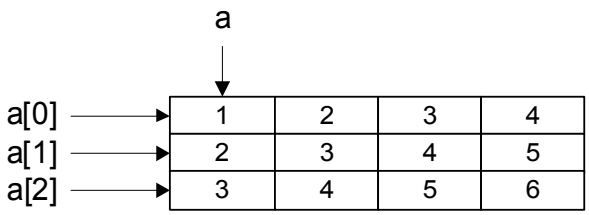
```
tipo_de_dato nombre_matriz[Nª_el_dim_1][N°_el_dim_2][N°_el_dim_3]...;
```

Al ser las matrices bidimensionales las de mayor aplicación práctica, el estudio de las matrices multidimensionales se centrará en las mismas, sin embargo, los principios y funciones que se estudien pueden ser extendidos (por analogía) a matrices de cualquier dimensión.

Cuando se declara una matriz estática en C/C++, se reserva un espacio de memoria continuo para todos los elementos y dicho espacio puede ser inicializado al momento de hacer la declaración. Por ejemplo, la siguiente instrucción crea una matriz de 3 filas por 4 columnas y le asigna valores iniciales:

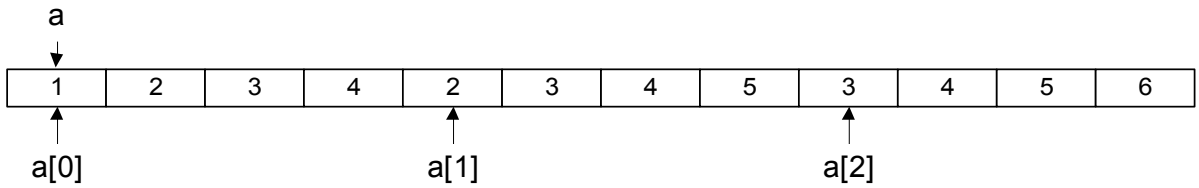
```
int a[3][4]={{1,2,3,4},{2,3,4,5},{3,4,5,6}};
```

Si cada espacio de memoria se representa como una casilla, esta matriz queda en memoria de la siguiente forma:



Donde, como se puede ver, el identificador "a" apunta al primer elemento de la matriz (recuerde que las matrices y los punteros están estrechamente relacionados en C/C++). Como también se puede ver, al ser las matrices un vector de vectores, los elementos del primer vector (el primer índice o dimensión) apuntan a los vectores (las filas) que contienen los elementos de la matriz: a[0] apunta a la primera fila, a[1] a la segunda y a[2] a la tercera.

Se recalca, sin embargo, que se trata de un bloque continuo de memoria, no existe ningún separador entre los datos, por lo que una representación gráfica más aproximada de la misma es:



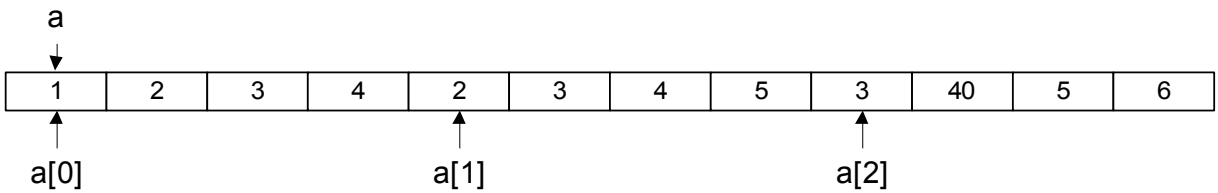
Para acceder a un determinado elemento se deben especificar las dos dimensiones, así para sumar el segundo elemento de la primera fila con el tercer elemento de la tercera fila, se escribe:

```
s=a[0][1]+a[2][2];
```

Se procede de igual forma, para asignar o cambiar el valor de los elementos de la matriz, por ejemplo, para asignar el número 40 al tercer elemento de la segunda fila, se escribe:

```
a[1][2]=40;
```

Con lo que la representación gráfica de los elementos de la matriz en memoria quedaría en la forma:



Al estar las matrices estrechamente relacionadas a los punteros, se puede trabajar con punteros, y la aritmética de punteros, en lugar de matrices, por ejemplo, en el siguiente segmento de código se emplea un puntero para mostrar en pantalla los elementos de la matriz "a":

```
int *b=(int*)a;
for (int i=0;i<3;i++){
  for (int j=0;j<4;j++)
    cout<<setw(8)<<*b++;
  cout<<endl;
}
```

Como se puede ver, se asigna la dirección de la matriz (el valor de "a") a un puntero de tipo entero. En este caso es necesario el moldeo de tipos porque la matriz es un puntero a un vector de 4 elementos (int*[4]), mientras que "b" es un puntero a un dato de tipo entero (int*). Luego, como se sabe que los elementos se encuentran en un bloque continuo, se muestra el valor al que apunta el puntero ("*b") y se incrementa su valor en 1 (b++) para que en el siguiente ciclo apunte al siguiente elemento.

Observe también que en una misma instrucción ("*b++") se obtiene el valor al que apunta el puntero y se incrementa su valor en 1, ello es posible debido al orden de prioridad de los operadores: el operador "*" tiene mayor orden de prioridad que el operador "++" (como sufijo), por lo tanto primero se obtiene el valor al que apunta el puntero y después (una vez obtenido el valor) se incrementa el valor del puntero en 1.

Para crear matrices dinámicas el procedimiento más directo consiste en crear un vector de vectores. Por ejemplo, con las siguientes instrucciones se crea una matriz de 5 filas por 6 columnas, sin valores iniciales:

```
int **a; const int nf=5,nc=6;
a = new int*[nf];
for (int i=0;i<nf;i++)
  a[i] = new int[nc];
```

Como se puede ver se requiere un puntero a puntero (**a) para trabajar con matrices dinámicas. Primero se reserva memoria para las filas (las direcciones de memoria de cada fila "*a") y luego se reserva memoria para los elementos de cada fila (las columnas que guardan los valores enteros "**a") y las direcciones de dichas filas se guardan en el espacio reservado para "*a".

De lo anterior se deduce que las matrices dinámicas pueden tener filas con diferentes números de columnas, pero además, dado que para cada fila se reserva un nuevo bloque de memoria, las filas de una matriz dinámica pueden encontrarse en diferentes sectores de memoria y no en un bloque continuo.

Una vez creadas se emplean de la misma forma que las matrices estáticas, así, en el siguiente segmento de código se le asignan valores y se muestran dichos valores en pantalla:

```
for (int i=0;i<nf;i++)
  for (int j=0;j<nc;j++)
    a[i][j]=i+j;
for (int i=0;i<nf;i++){
  for (int j=0;j<nc;j++)
    cout<<setw(8)<<a[i][j];
  cout<<endl;
}
```

Para liberar la memoria reservada se sigue el proceso inverso al de reservarla, es decir primero se libera la memoria de cada fila y finalmente se libera la memoria del vector de punteros:

```
for (int i=0;i<nf;i++)
  delete []a[i];
delete []a;
```

Al ser las matrices dinámicas punteros a punteros, se puede emplear aritmética de punteros en lugar de índices, así en el siguiente segmento de código se hace lo mismo que en los segmentos anteriores, es decir se crea la matriz, se le asignan valores iniciales, se muestran dichos valores y se libera la memoria reservada, pero empleando punteros y aritmética de punteros:

```
int **a; const int nf=5, nc=6;
a = new int*[nf];
for (int i=0;i<nf;i++)
  *(a+i) = new int[nc];
for (int i=0;i<nf;i++)
  for (int j=0;j<nc;j++)
    *(*(a+i)+j) = i+j;
for (int i=0;i<nf;i++){
  for (int j=0;j<nc;j++)
    cout<<setw(8)<<*(*(a+i)+j);
  cout<<endl;
}
for (int i=0;i<nf;i++)
  delete []*(a+i);
delete []a;
```

Donde, como se puede ver, "*(a+i)" es equivalente a "a[i]", de hecho, cuando se escribe la expresión "a[i]" el compilador la reescribe como "*(a+i)". Igualmente "*(*(a+i)+j)" es equivalente a "a[i][j]" o si se quiere a "*(a[i]+j)". Como se hace evidente en este ejemplo, si bien la aritmética de punteros puede ser más eficiente en algunos casos, el código resultante es menos legible y por lo tanto es más fácil cometer errores.

Es posible también cambiar los valores de los punteros de manera que queden apuntando al dato o datos deseados, así por ejemplo el siguiente segmento de código hace lo mismo que el anterior, pero se cambian las direcciones de los punteros para que apunten a los datos deseados:

```
int **a,**a0,*a1; const int nf=5, nc=6;
a0 = a = new int*[nf];
for (int i=0;i<nf;i++,a++)
  *a = new int[nc];
a=a0;
for (int i=0;i<nf;i++,a++){
```

```
    a1 = *a;
    for (int j=0;j<nc;j++,(*a)++)
      **a = i+j;
    *a = a1;
  }
  a=a0;
  for (int i=0;i<nf;i++,a++,cout<<endl){
    a1 = *a;
    for (int j=0;j<nc;j++,(*a)++)
      cout<<setw(8)<<**a;
    *a = a1;
  }
  a=a0;
  for (int i=0;i<nf;i++,a++)
    delete []*a;
  delete []a0;
```

En este caso, dado que el valor de los punteros cambia en los ciclos, es necesario guardar las direcciones iniciales en otras variables. Así la dirección de la matriz se guarda en la variable "a0" y la dirección de las filas en la variable "a1". Esta es la razón por la cual, después de los ciclos "for i", que es donde se incrementa el valor de "a" (a++), siempre se restituye la dirección original a la variable "a" (a=a0).

Igualmente antes de cada ciclo "for j", que es donde se cambia el valor de los punteros de fila ((*a)++), se guarda la dirección original en "a1" (a1=*a) y después del ciclo se restituye dicha dirección (*a=a1). Si no se restituyeran esta direcciones los punteros quedarían apuntando a direcciones desconocidas, ubicadas fuera de la matriz.

Es importante hacer notar que los punteros de las filas se incrementan con la expresión "(*a)++" y no "*a++", esto porque si se empleara esta expresión, primero se recuperaría la dirección del elemento "*a" (dirección que no se emplearía en nada) y luego se incrementaría el valor de "a" (el puntero a la matriz) y no el de "*a" (el puntero a la fila). Con los paréntesis se altera el orden de prioridad de los operadores de manera que se incremente el valor de "*a" en 1 (y el de "a").

Puesto que la mayoría de las aplicaciones prácticas trabajan con matrices regulares, es decir matrices que tienen el mismo número de columnas en todas las filas, es más eficiente reservar un bloque de memoria continuo, tal como ocurre con las matrices estáticas. Si el puntero donde se guarda la dirección de este bloque es "a", se puede acceder a cualquier elemento ubicado en la fila "i" y en la columna "j" con la igualdad:

$$a_{i,j} = a[i*nc+j]$$

O empleando notación de punteros:

$$a_{i,j} = *(a+i*nc+j)$$

Como de costumbre, se pueden acceder a los elementos de la matriz cambiando la dirección del puntero, en cuyo caso se debe preservar la dirección original para vitar su pérdida.

Es posible también emular el comportamiento de las matrices estáticas creando un vector con punteros a cada una de las filas, con lo que se puede emplear la nomenclatura estándar de las matrices, así, en el siguiente segmento de código se crea una matriz dinámica con 4 filas y 6 columnas y se accede a sus elementos siguiendo este procedimiento:

```
int *b,**a,nf=4,nc=6;
b = new int[nf*nc];
```

```
a = new int*[nf];
for (int i=0;i<nf;i++)
  a[i]= b+i*nc;
for (int i=0;i<nf;i++)
  for (int j=0;j<nc;j++)
    a[i][j]=i+j;
for (int i=0;i<nf;i++,cout<<endl)
  for (int j=0;j<nc;j++)
    cout<<setw(8)<<a[i][j];
delete []a;
delete []b;
```

Que al ser ejecutado muestra la siguiente matriz en pantalla:

```
0       1       2       3       4       5
1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

Observe que en este caso, a diferencia de una matriz creada como un vector de vectores, no se reserva memoria para cada uno de los elementos del vector "a", sino que se le asignan las direcciones de memoria ya existentes (mismas que han sido reservadas como un bloque continuo apuntado por "b"). Por esta misma razón, cuando se libera la memoria no se libera la memoria a la que apunta cada uno de los elementos del vector "a", sino simplemente se libera la memoria ocupada por dicho vector y la memoria ocupada por la matriz "b" (que es la memoria reservada para los elementos de la matriz).

15.2. ALGUNAS FUNCIONES DE UTILIDAD GENERAL PARA EL TRABAJO CON MATRICES

Como ya se vio en el anterior acápite existen diversas alternativas para el trabajo con matrices. Una de las más sencillas y eficientes, desde el punto de vista de su implementación, consiste en tratarlas como vectores, empleando las relaciones dadas en el anterior acápite para acceder sus elementos en base a los índices de fila y columna.

Sin embargo, la alternativa más sencilla, desde el punto de vista del usuario, es la de emular el comportamiento de las matrices estáticas permitiendo acceder a sus elementos con la notación estándar, tal como se vio en el último ejemplo del anterior acápite. Esta es la opción elegida para el trabajo con matrices en el presente tema.

Como no es ni lógico ni eficiente el repetir, una y otra vez, los mismos procedimientos para llevar a cabo las mismas tareas, se crearán algunas funciones de utilidad general para el trabajo con matrices.

Las propiedades que deben estar ligadas a las matrices, pues forman parte de ellas, son su número de filas y columnas. No es conveniente manejar dicha información por separado, porque constituyen una fuente de error y porque su mantenimiento se hace más difícil a medida que el programa crece. Como se verá posteriormente, una forma de asociar dichas propiedades a la matriz es mediante el uso de registros, la otra, que es la que se empleará en este tema, consiste en emplear parte de la memoria reservada para guardar dicha información.

En ese sentido, la primera función de carácter general a implementar, en forma de funciones sobrecargadas y añadida a la librería "matrices", es "crearMatriz", la cual reserva memoria para una matriz de números enteros, reales o de caracteres y reserva un espacio de memoria adicional donde guarda las dimensiones de la matriz creada. El proceso que se sigue es esencialmente el mismo ya sea que se trate de números enteros, reales o de

caracteres, por lo que puede ser implementado como una plantilla, sin embargo, dado que las plantillas no pueden ser compiladas y en consecuencia deben ser escritas en el archivo de cabecera, son más vulnerables, razón por la cual se ha optado por las funciones sobrecargadas a pesar de que con ello las soluciones son menos generales.

Para comprender mejor el proceso se analiza primero la función que crea una matriz de números enteros (escrito en el archivo "matrices.cpp"):

```
void crearMatriz(int** &a, int nf, int nc) throw(ERROR)
{
  if (nf<=0 || nc<=0) throw ERROR_DIMENSIONES;
  int *b;
  b = (int*)malloc(nf*nc*sizeof(int)+2*sizeof(int));
  if (b == NULL) throw ERROR_MEMORIA;
  a = (int**)malloc(nf*sizeof(int*));
  if (a == NULL) throw ERROR_MEMORIA;
  *b++ = nc;
  *b++ = nf;
  for (int i=0;i<nf;i++,b+=nc)
    a[i] = b;
}
```

Como se pude ver, la función recibe, por referencia, un puntero a un puntero de tipo entero "**a". Es importante recibir este parámetro por referencia porque en el mismo se devuelve la dirección de memoria de la matriz. Si no se recibiera el parámetro por referencia la dirección y en consecuencia la memoria reservada, se perdería definitivamente.

Dentro de la función se declara un puntero a enteros "*b". Empleando este puntero se reserva un bloque de memoria continuo para la matriz: "nf*nc*sizeof(int)" y se le añade un espacio de memoria para dos números enteros "2*sizeof(int)", este espacio de memoria adicional se emplea para guardar el número de filas y columnas de la matriz. Luego se reserva memoria para guardar "nf" direcciones de memoria en el puntero "a": "nf*sizeof(int*)".

En esta función se emplea "malloc" y no "new", porque "malloc" permite reservar el número de bytes que se desee, lo que permite reservar el espacio de memoria extra para las dimensiones, mientras que "new" sólo permite reservar un espacio de memoria que sea múltiplo del tipo base, es decir que "new" funcionaría para una matriz de números enteros, pero no para una de reales o de tipo char.

Al principio del bloque de memoria se guardan las dimensiones de la matriz "nc" (número de columnas) y "nf" (número de filas) y se deja el puntero "b" apuntando al primer espacio de memoria de la matriz (primera fila), entonces en "**a" se guardan las direcciones de memoria correspondientes a cada una de las filas y para ello se emplea un ciclo "for", donde en cada iteración se incrementa el valor de "b" en "nc" (para que en cada iteración apunte al principio de la siguiente fila). En consecuencia, al final del proceso, "a" contiene las direcciones de memoria de todas las filas de la matriz y en el espacio de memoria previo a la matriz se encuentran las dimensiones de la misma (como dos números enteros).

Las funciones sobrecargadas tienen la misma lógica excepto que se requiere un puntero adicional de tipo entero ("c") para guardar las dimensiones de la matriz y hacer la aritmética de punteros (pues ahora los datos ya no son de tipo entero):

```
void crearMatriz(double** &a, int nf, int nc) throw(ERROR)
{
  if (nf<=0 || nc<=0) throw ERROR_DIMENSIONES;
```

```
    double *b; int *c;
    b = (double*)malloc(nf*nc*sizeof(double)+2*sizeof(int));
    if (b == NULL) throw ERROR_MEMORIA;
    a = (double**)malloc(nf*sizeof(double*));
    if (a == NULL) throw ERROR_MEMORIA;
    c = (int*)b;
    *c++ = nc;
    *c++ = nf;
    b = (double*)c;
    for (int i=0;i<nf;i++,b+=nc)
      a[i] = b;
  }

  void crearMatriz(char** &a, int nf, int nc) throw(ERROR)
  {
    if (nf<=0 || nc<=0) throw ERROR_DIMENSIONES;
    char *b; int *c;
    b = (char*)malloc(nf*nc*sizeof(char)+2*sizeof(int));
    if (b == NULL) throw ERROR_MEMORIA;
    a = (char**)malloc(nf*sizeof(char*));
    if (a == NULL) throw ERROR_MEMORIA;
    c = (int*)b;
    *c++ = nc;
    *c++ = nf;
    b = (char*)c;
    for (int i=0;i<nf;i++,b+=nc)
      a[i] = b;
  }
```

Como de costumbre, cuando se trabaja con librerías, los prototipos de estas funciones deben ser añadidos al archivo de cabecera "matrices.h":

```
  //Crea la matriz "a" con "nf" filas y "nc" columnas, genera el error 3
  //si "n" o "m" es menor o igual a 0 y el error 4 si no existe memoria
  //suficiente para la matriz
  void crearMatriz(int** &a, int nf, int nc) throw(ERROR);
  void crearMatriz(double** &a, int nf, int nc) throw(ERROR);
  void crearMatriz(char** &a, int nf, int nc) throw(ERROR);
```

Siendo las constantes añadidas:

```
  //No existe memoria suficiente para realizar la operación requerida
  const ERROR ERROR_MEMORIA=4;
  //El puntero que se está queriendo liberar es nulo
  const ERROR ERROR_NULO=5;
```

Puesto que la memoria reservada tiene un espacio extra para las dimensiones, debe ser liberada incluyendo dicho espacio. Con ese fin se crean las función sobrecargadas respectivas, cuyos prototipos (que deben ser añadidos al archivo "matrices.h") son:

```
//Libera la memoria reservada para la matriz "a"
  void liberarMatriz(int** &a) throw(ERROR);
  void liberarMatriz(double** &a) throw(ERROR);
  void liberarMatriz(char** &a) throw(ERROR);
```

Siendo el código respectivo:

```
void liberarMatriz(int** &a) throw(ERROR)
{
  if (a == NULL) throw ERROR_NULO;
  int *b=*a-2;
```

```
    free(b);
    free(a);
  }

  void liberarMatriz(double** &a) throw(ERROR)
  {
    if (a == NULL) throw ERROR_NULO;
    double *b = (double*)((int*)*a-2);
    free(b);
    free(a);
  }

  void liberarMatriz(char** &a) throw(ERROR)
  {
    if (a == NULL) throw ERROR_NULO;
    char *b=(char*)((int*)*a-2);
    free(b);
    free(a);
  }
```

Como se puede ver, a la dirección de memoria que tiene "a" se le resta dos posiciones (tratando al puntero como un puntero a entero), de manera que apunte al inicio del espacio de memoria reservado, esta dirección se asigna a un puntero del tipo de dato de la matriz y se libera la memoria empleando el mismo ("free(b)"), finalmente se libera la memoria reservada para "a" (las direcciones de memoria de cada fila).

Para conocer el número de filas y columnas de una matriz se crean funciones sobrecargadas (añadidas a "matrices.h"):

```
    //Devuelve el número de filas de la matriz "a"
    int numfil(int **a);
    int numfil(double **a);
    int numfil(char **a);

    //Devuelve el número de columnas de la matriz "a"
    int numcol(int **a);
    int numcol(double **a);
    int numcol(char **a);
```

Siendo el código respectivo (añadido a "matrices.cpp"):

```
int numfil(int **a){
  return (*(*a-1));
}
int numfil(double **a){
  return (*((int*)*a-1));
}
int numfil(char **a){
  return (*((int*)*a-1));
}

int numcol(int **a){
  return (*(*a-2));
}
int numcol(double **a){
  return (*((int*)*a-2));
}
int numcol(char **a){
  return (*((int*)*a-2));
```

```
}
```

Como se puede ver, para extraer el número de filas se le resta a la dirección de memoria guardada en "a" 1 (tratando dicha dirección como un puntero a entero) y se procede de forma similar para el número de columnas, sólo que ahora se le resta 2, de esa manera se leen los valores guardados en dichas posiciones al crear la matriz.

Otras funciones de utilidad para el trabajo con matrices son "mostrarMatriz", que muestra en pantalla los elementos de una matriz, "llenarMatriz" que llena la matriz con elementos generados al azar, "leerMatriz" que lee desde teclado los elementos de la matriz y "redondearMatriz" que redondea los elementos de una matriz de números reales al número de dígitos especificado.

Los prototipos de estas funciones (añadidos a "matrices.h") son:

```
//Muestra los elementos de la matriz "a" en un ancho de "w" caracteres
//mostrando "f" elementos por fila
void mostrarMatriz(int** a, int w=8, int f=10);
void mostrarMatriz(double** a, int w=8, int f=10);
void mostrarMatriz(char** a, int w=8, int f=10);

//Llena la matriz "a" con elementos aleatorios comprendidos entre "li"
//y "ls". Genera un error si "li" es mayor o igual a "ls".
void llenarMatriz(int** a, int li=1, int ls=10) throw(ERROR);
void llenarMatriz(double** a, double li=1., double ls=10.)throw(ERROR);
void llenarMatriz(char** a, char li='A', char ls='Z') throw(ERROR);

//Lee, desde teclado, los elementos de la matriz "a"
void leerMatriz(int **a);
void leerMatriz(double **a);
void leerMatriz(char **a);

//Redonde los elementos de la matriz "a" a "n" dígitos después del
//punto
void redondearMatriz(double **a, int n=0) throw(ERROR);
```

Siendo el código respectivo, añadido a "matrices.cpp":

```
void mostrarMatriz(int **a, int w, int f){
int nf=numfil(a);
int nc=numcol(a);
if (nc<f) f=nc;
int c=0;
cout<<"[";
for (int i=0;i<nf;i++){
  cout<<"[";
  for (int j=0;j<nc;j++){
    cout<<setw(w)<<a[i][j];
    if (++c == f) {
      c=0;
      if (j!=nc-1) cout<<endl<<"  ";}
  }
  if (c!=0) cout<<setw(w*(f-c+1))<<"]"; else cout<<setw(w)<<"]";
  c=0;
  if (i!=nf-1)
    cout<<endl<<" ";
  else
    cout<<"]"<<endl<<endl;
}
```

```
}
void mostrarMatriz(double **a, int w, int f){
  int nf=numfil(a);
  int nc=numcol(a);
  if (nc<f) f=nc;
  int c=0;
  cout<<"[";
  for (int i=0;i<nf;i++){
    cout<<"[";
    for (int j=0;j<nc;j++){
      cout<<setw(w)<<a[i][j];
      if (++c == f) {
        c=0;
        if (j!=nc-1) cout<<endl<<"  ";}
    }
    if (c!=0) cout<<setw(w*(f-c+1))<<"]"; else cout<<setw(w)<<"]";
    c=0;
    if (i!=nf-1)
      cout<<endl<<" ";
    else
      cout<<"]"<<endl<<endl;
  }
}
void mostrarMatriz(char **a, int w, int f){
  int nf=numfil(a);
  int nc=numcol(a);
  if (nc<f) f=nc;
  int c=0;
  cout<<"[";
  for (int i=0;i<nf;i++){
    cout<<"[";
    for (int j=0;j<nc;j++){
      cout<<setw(w)<<a[i][j];
      if (++c == f) {
        c=0;
        if (j!=nc-1) cout<<endl<<"  ";}
    }
    if (c!=0) cout<<setw(w*(f-c+1))<<"]"; else cout<<setw(w)<<"]";
    c=0;
    if (i!=nf-1)
      cout<<endl<<" ";
    else
      cout<<"]"<<endl<<endl;
  }
}

void llenarMatriz(int** a, int li, int ls) throw(ERROR)
{
  if (li>=ls) throw ERROR_LIMITES;
  int d=ls-li, nf=numfil(a), nc=numcol(a);
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++)
      a[i][j] = rand()*d/RAND_MAX+li;
}
void llenarMatriz(double** a, double li, double ls) throw(ERROR)
{
  if (li>=ls) throw ERROR_LIMITES;
  double d=ls-li; int nf=numfil(a),nc=numcol(a);
```

```
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++)
      a[i][j] = rand()*d/RAND_MAX+li;
}
void llenarMatriz(char** a, char li, char ls) throw(ERROR)
{
  if (li>=ls) throw ERROR_LIMITES;
  char d=ls-li; int nf=numfil(a),nc=numcol(a);
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++)
      a[i][j] = rand()*d/RAND_MAX+li;
}

void leerMatriz(int **a){
  int nf = numfil(a);
  int nc = numcol(a);
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++){
      cout<<"a["<<i<<","<<j<<"]? ";
      cin>>a[i][j];
    }
}
void leerMatriz(double **a){
  int nf = numfil(a);
  int nc = numcol(a);
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++){
      cout<<"a["<<i<<","<<j<<"]? ";
      cin>>a[i][j];
    }
}
void leerMatriz(char **a){
  int nf = numfil(a);
  int nc = numcol(a);
  for (int i=0;i<nf;i++)
    for (int j=0;j<nc;j++){
      cout<<"a["<<i<<","<<j<<"]? ";
      cin>>a[i][j];
    }
}

void redondearMatriz(double **a, int n) throw(ERROR){
  if (n>15 || n<0) throw ERROR_REDONDEO;
  int nf=numfil(a), nc=numcol(a);
  if (n==0)
    for (int i=0;i<nf;i++)
      for (int j=0;j<nc;j++)
        a[i][j]=round(a[i][j]);
  else {
    double d=1;
    for (int i=0;i<n;i++)
      d*=10;
    for (int i=0;i<nf;i++)
      for (int j=0;j<nc;j++)
        a[i][j]=round(a[i][j]*d)/d;
  }
}
```

En estas funciones, y sólo por claridad, se emplea la notación matricial

para acceder a los elementos de las matrices, por supuesto se puede emplear también la notación y la aritmética de punteros para lograr los mismos resultados.

A medida que se escriban funciones de utilidad general, se las irá añadiendo a la biblioteca "matrices".

15.3. EJEMPLOS

1. Añada a la biblioteca matrices, una función sobrecargada que intercambie los elementos de dos columnas dadas de una matriz de números enteros, reales o de caracteres.

La lógica que resuelve el problema es muy sencilla, se emplea un ciclo "for" para recorrer todas las filas de la matriz y que en cada repetición del ciclo se intercambian los elementos de las columnas dadas.

Antes de llevar a cabo el intercambio se debe verificar que ninguna de las columnas sea mayor al número de columnas de la matriz y que tampoco sean inferiores a cero, pues de ser así el intercambio es imposible.

También se debe verificar que los números de columnas sean diferentes, pues si son iguales no es necesario ningún intercambio.

Los prototipos, añadidos a "matrices.h", son:

```
//Intercambia las columnas "c1" y "c2" de la matriz "a"
void intercambiarColumnas(int **a, int c1, int c2) throw(ERROR);
void intercambiarColumnas(double **a, int c1, int c2) throw(ERROR);
void intercambiarColumnas(char **a, int c1, int c2) throw(ERROR);
```

Y el código respectivo, añadido a "matrices.cpp", es:

```
void intercambiarColumnas(int **a, int c1, int c2) throw(ERROR){
  int aux,nf=numfil(a), nc=numcol(a);
  if (c1>nc || c2>nc || c1<0 || c2<0) throw ERROR_DIMENSIONES;
  if (c1==c2) return;
  for (int i=0;i<nf;i++){
    aux=a[i][c1]; a[i][c1]=a[i][c2]; a[i][c2]=aux;
  }
}
void intercambiarColumnas(double **a, int c1, int c2) throw(ERROR){
  double aux; int nf=numfil(a), nc=numcol(a);
  if (c1>nc || c2>nc || c1<0 || c2<0) throw ERROR_DIMENSIONES;
  if (c1==c2) return;
  for (int i=0;i<nf;i++){
    aux=a[i][c1]; a[i][c1]=a[i][c2]; a[i][c2]=aux;
  }
}
void intercambiarColumnas(char **a, int c1, int c2) throw(ERROR){
  char aux; int nf=numfil(a), nc=numcol(a);
  if (c1>nc || c2>nc || c1<0 || c2<0) throw ERROR_DIMENSIONES;
  if (c1==c2) return;
  for (int i=0;i<nf;i++){
    aux=a[i][c1]; a[i][c1]=a[i][c2]; a[i][c2]=aux;
  }
}
```

Y el programa donde se prueban estas funciones, así como uno de los casos de error, es:

```
#include <iostream>
```

```
#include "matrices.h"
using namespace std;
using namespace matrices;

int main(){
  int **a;
  cout<<"Matriz Original:"<<endl;
  crearMatriz(a,6,7);
  llenarMatriz(a);
  mostrarMatriz(a);
  intercambiarColumnas(a,2,4);
  cout<<"Columnas 2 y 4 intercambiadas:"<<endl;
  mostrarMatriz(a);
  liberarMatriz(a);
  double **b;
  crearMatriz(b,4,5);
  llenarMatriz(b);
  redondearMatriz(b,2);
  cout<<"Matriz Original:"<<endl;
  mostrarMatriz(b);
  intercambiarColumnas(b,0,3);
  cout<<"Columnas 0 y 3 intercambiadas:"<<endl;
  mostrarMatriz(b);
  liberarMatriz(b);
  char **c;
  crearMatriz(c,7,6);
  llenarMatriz(c);
  cout<<"Matriz Original:"<<endl;
  mostrarMatriz(c);
  intercambiarColumnas(c,1,5);
  cout<<"Columnas 1 y 5 intercambiadas: "<<endl;
  mostrarMatriz(c);
  try{
    intercambiarColumnas(c,1,10);
    mostrarMatriz(c);
  } catch(ERROR e){
    cout<<"ERROR: Número de columnas erróneo!"<<endl;
  }
  liberarMatriz(c);
}
```

Con el cual se obtienen los siguientes resultados:

```
Matriz Original:
[[       1       6       2       8       6       5       4       ]
 [       9       8       7       2       8       7       5       ]
 [       3       1       1       4       2       2       9       ]
 [       5       2       1       1       4       5       6       ]
 [       6       6       2       6       5       4       1       ]
 [       6       8       8       5       3       8       7       ]]

Columnas 2 y 4 intercambiadas:
[[       1       6       6       8       2       5       4       ]
 [       9       8       8       2       7       7       5       ]
 [       3       1       2       4       1       2       9       ]
 [       5       2       4       1       1       5       6       ]
 [       6       6       5       6       2       4       1       ]
 [       6       8       3       5       8       8       7       ]]
```

```
Matriz Original:
[[     9.6    9.33    5.85    2.28    5.16       ]
 [    3.12    8.76    2.89    8.02    8.59       ]
 [    9.97      10     6.5    4.53     3.4       ]
 [    3.68    8.56    1.21    4.38    1.83       ]]

Columnas 0 y 3 intercambiadas:
[[    2.28    9.33    5.85     9.6    5.16       ]
 [    8.02    8.76    2.89    3.12    8.59       ]
 [    4.53      10     6.5    9.97     3.4       ]
 [    4.38    8.56    1.21    3.68    1.83       ]]

Matriz Original:
[[       Q       B       A       W       G       G       ]
 [       O       R       U       S       M       F       ]
 [       S       L       L       X       S       C       ]
 [       O       J       S       P       O       J       ]
 [       D       F       K       U       M       Y       ]
 [       S       I       E       Q       M       B       ]
 [       R       M       D       X       D       W       ]]

Columnas 1 y 5 intercambiadas:
[[       Q       G       A       W       G       B       ]
 [       O       F       U       S       M       R       ]
 [       S       C       L       X       S       L       ]
 [       O       J       S       P       O       J       ]
 [       D       Y       K       U       M       F       ]
 [       S       B       E       Q       M       I       ]
 [       R       W       D       X       D       M       ]]

ERROR: Número de columnas erróneo!
```

2. Añada a la biblioteca "matrices" una función sobrecargada que, dada una matriz, devuelva un vector con los elementos de la columna especificada.

Una vez más, el algoritmo es muy sencillo: se crea un vector con un número de elementos igual al número de filas de la matriz y luego, empleando un ciclo "for" que recorre las filas de la matriz, se copia en cada repetición del ciclo el elemento de la columna dada en el vector creado. Finalmente se devuelve el vector.

Igual que en el anterior ejemplo primero se debe verificar que la columna dada no sea mayor al número de columnas existentes en la matriz o que sea menor a 0, pues de ser así es imposible devolver la columna, por lo que se debe generar un error.

Los prototipos, añadidos a "matrices.h", son:

```
//Devuelve un vector con los elementos de la columna "c1" de la
//matriz "a"
int *columna(int **a, int c1) throw(ERROR);
double *columna(double **a, int c1) throw(ERROR);
char *columna(char **a, int c1) throw(ERROR);
```

Siendo el código respectivo, añadido a "matrices.cpp":

```
int *columna(int **a, int c1) throw(ERROR){
  int *v; int nf=numfil(a), nc=numcol(a);
  if (c1>nc || c1<0) throw(ERROR_DIMENSIONES);
  crearVector(v,nf);
```

```
  for (int i=0;i<nf;i++)
    v[i]=a[i][c1];
  return v;
}
double *columna(double **a, int c1) throw(ERROR){
  double *v; int nf=numfil(a), nc=numcol(a);
  if (c1>nc || c1<0) throw(ERROR_DIMENSIONES);
  crearVector(v,nf);
  for (int i=0;i<nf;i++)
    v[i]=a[i][c1];
  return v;
}
char *columna(char **a, int c1) throw(ERROR){
  char *v; int nf=numfil(a), nc=numcol(a);
  if (c1>nc || c1<0) throw(ERROR_DIMENSIONES);
  crearVector(v,nf);
  for (int i=0;i<nf;i++)
    v[i]=a[i][c1];
  return v;
}
```

El programa que prueba estos módulos, incluyendo un caso de error, es:

```
#include <iostream>
#include "matrices.h"
using namespace std;
using namespace matrices;

int main(){
  int **a,*x;
  cout<<"Matriz Original:"<<endl;
  crearMatriz(a,6,7);
  llenarMatriz(a);
  mostrarMatriz(a);
  x=columna(a,3);
  cout<<"Columna 3:"<<endl;
  mostrarVector(x);
  liberarMatriz(a);
  liberarVector(x);
  double **b,*y;
  crearMatriz(b,4,5);
  llenarMatriz(b);
  redondearMatriz(b,2);
  cout<<"Matriz Original:"<<endl;
  mostrarMatriz(b);
  y=columna(b,1);
  cout<<"Columna 1:"<<endl;
  mostrarVector(y);
  liberarMatriz(b);
  liberarVector(y);
  char **c,*z;
  crearMatriz(c,7,6);
  llenarMatriz(c);
  cout<<"Matriz Original:"<<endl;
  mostrarMatriz(c);
  z=columna(c,5);
  cout<<"Columnas 5: "<<endl;
  mostrarVector(z);
  liberarVector(z);
```

```
    try{
      z=columna(c,10);
      mostrarVector(z);
      liberarVector(z);
    } catch(ERROR e){
      cout<<"ERROR: Número de columna erróneo!"<<endl;
    }
    liberarMatriz(c);
}
```

Con el cual se obtiene:

```
Matriz Original:
[[       1       6       2       8       6       5       4       ]
 [       9       8       7       2       8       7       5       ]
 [       3       1       1       4       2       2       9       ]
 [       5       2       1       1       4       5       6       ]
 [       6       6       2       6       5       4       1       ]
 [       6       8       8       5       3       8       7       ]]

Columna 3:
[       8       2       4       1       6       5       ]

Matriz Original:
[[     9.6    9.33    5.85    2.28    5.16       ]
 [    3.12    8.76    2.89    8.02    8.59       ]
 [    9.97      10     6.5    4.53     3.4       ]
 [    3.68    8.56    1.21    4.38    1.83       ]]

Columna 1:
[    9.33    8.76      10    8.56       ]

Matriz Original:
[[       Q       B       A       W       G       G       ]
 [       O       R       U       S       M       F       ]
 [       S       L       L       X       S       C       ]
 [       O       J       S       P       O       J       ]
 [       D       F       K       U       M       Y       ]
 [       S       I       E       Q       M       B       ]
 [       R       M       D       X       D       W       ]]

Columnas 5:
[       G       F       C       J       Y       B       W       ]

ERROR: Número de columna erróneo!
```

3. Añada a la biblioteca matrices una función sobrecargada que permita insertar las filas de una matriz en la fila especificada dentro de otra matriz.

Para insertar una o más filas en una matriz se crea otra con las nuevas dimensiones y en la misma se copian las filas de la matriz original y de la matriz insertada en las posiciones correctas, luego se libera la memoria ocupada por la matriz original y se guarda la dirección del nuevo bloque de memoria en la variable que guardaba la dirección de la matriz original.

Los prototipos de las funciones sobrecargadas, añadidas a “matrices.h”, son:

```
//Inserta las filas la matriz "b" en la matriz "a" a partir de
```

```
    //la fila "nfi". Se genera un error si los números de columnas de "a"
    //y "b" son diferentes o si "nfi" está fuera de las dimensiones de "a"
    void insertarFilas(int** b, int** &a, int nfi) throw(ERROR);
    void insertarFilas(double** b, double** &a, int nfi) throw(ERROR);
    void insertarFilas(char** b, char** &a, int nfi) throw(ERROR);
```

El código comentado para el caso de las matrices enteras, añadido a "matrices.cpp", es:

```
  void insertarFilas(int** b, int** &a, int nfi) throw(ERROR){
    int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
    if (b==NULL) throw(ERROR_NULO);
    if (nca!=ncb || nfi>nfa || nfi<0) throw(ERROR_DIMENSIONES);
    int **c, nnf=nfa+nfb;
    //Matriz resultante con el nuevo número de filas
    crearMatriz(c,nnf,nca);
    //Se copian en "c" las filas iniciales de "a"
    for (int i=0;i<nfi;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i][j];
    //Se copia en "c" las filas de "b"
    for (int i=nfi;i<nfi+nfb;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=b[i-nfi][j];
    //Se copia en "c" las filas restantes de "a"
    for (int i=nfi+nfb;i<nnf;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i-nfb][j];
    //Se libera la memoria ocupada por "a"
    liberarMatriz(a);
    //Se asigna a "a" la dirección de "c"
    a = c;
  }
```

Y el código para números reales y caracteres, añadido también a "matrices.cpp" y que casi es el mismo que para enteros, es:

```
  void insertarFilas(double** b, double** &a, int nfi) throw(ERROR){
    int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
    if (b==NULL) throw(ERROR_NULO);
    if (nca!=ncb || nfi>nfa || nfi<0) throw(ERROR_DIMENSIONES);
    double **c; int nnf=nfa+nfb;
    crearMatriz(c,nnf,nca);
    for (int i=0;i<nfi;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i][j];
    for (int i=nfi;i<nfi+nfb;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=b[i-nfi][j];
    for (int i=nfi+nfb;i<nnf;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i-nfb][j];
    liberarMatriz(a);
    a = c;
  }
  void insertarFilas(char** b, char** &a, int nfi) throw(ERROR){
    int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
    if (b==NULL) throw(ERROR_NULO);
    if (nca!=ncb || nfi>nfa || nfi<0) throw(ERROR_DIMENSIONES);
    char **c; int nnf=nfa+nfb;
```

```
    crearMatriz(c,nnf,nca);
    for (int i=0;i<nfi;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i][j];
    for (int i=nfi;i<nfi+nfb;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=b[i-nfi][j];
    for (int i=nfi+nfb;i<nnf;i++)
      for (int j=0;j<nca;j++)
        c[i][j]=a[i-nfb][j];
    liberarMatriz(a);
    a = c;
  }
```

Un programa donde se prueban estas funciones, incluyendo un caso de error, es el siguiente:

```
  #include <iostream>
  #include "matrices.h"
  using namespace std;
  using namespace matrices;

  int main(){
    try {
      //Prueba para números enteros
      int **a,**b;
      crearMatriz(a,5,6);
      llenarMatriz(a);
      cout<<"Matriz a:"<<endl;
      mostrarMatriz(a);
      crearMatriz(b,2,6);
      llenarMatriz(b);
      cout<<"Matriz b:"<<endl;
      mostrarMatriz(b);
      insertarFilas(b,a,2);
      cout<<"Matriz 'b' insertada en la matriz 'a' a partir de la fila
  2"<<endl;
      mostrarMatriz(a);
      liberarMatriz(a);
      liberarMatriz(b);
      //Prueba para números reales
      double **c,**d;
      crearMatriz(c,5,5);
      llenarMatriz(c);
      cout<<"Matriz c:"<<endl;
      mostrarMatriz(c);
      crearMatriz(d,3,5);
      llenarMatriz(d);
      cout<<"Matriz d:"<<endl;
      mostrarMatriz(d);
      insertarFilas(d,c,0);
      cout<<"Matriz 'd' insertada en la matriz 'c' a partir de la fila
  0"<<endl;
      mostrarMatriz(c);
      liberarMatriz(c);
      liberarMatriz(d);
      //Prueba para caracteres
      char **e,**f;
      crearMatriz(e,6,7);
```

```
    llenarMatriz(e);
    cout<<"Matriz e:"<<endl;
    mostrarMatriz(e);
    crearMatriz(f,4,7);
    llenarMatriz(f);
    cout<<"Matriz f:"<<endl;
    mostrarMatriz(f);
    insertarFilas(f,e,6);
    cout<<"Matriz 'f' insertada en la matriz 'e' a partir de la fila
6"<<endl;
    mostrarMatriz(e);
    insertarFilas(f,e,20);
    liberarMatriz(e);
    liberarMatriz(f);
  } catch (ERROR e){
    cout<<"Fila a insertar errónea!!"<<endl;
  }
  return 0;
}
```

Con el cual se obtienen los siguientes resultados:

```
Matriz a:
[[       1       6       2       8       6       5       ]
 [       4       9       8       7       2       8       ]
 [       7       5       3       1       1       4       ]
 [       2       2       9       5       2       1       ]
 [       1       4       5       6       6       6       ]]

Matriz b:
[[       2       6       5       4       1       6       ]
 [       8       8       5       3       8       7       ]]

Matriz 'b' insertada en la matriz 'a' a partir de la fila 2
[[       1       6       2       8       6       5       ]
 [       4       9       8       7       2       8       ]
 [       2       6       5       4       1       6       ]
 [       8       8       5       3       8       7       ]
 [       7       5       3       1       1       4       ]
 [       2       2       9       5       2       1       ]
 [       1       4       5       6       6       6       ]]

Matriz c:
[[ 9.60311 9.33146 5.85418 2.28104 5.15873       ]
 [ 3.11795 8.76016 2.88641 8.01691 8.59288       ]
 [ 9.97116 9.99725 6.50349 4.53194 3.39592       ]
 [ 3.67553  8.5613 1.21369 4.38279 1.83361       ]
 [ 7.09485 1.50594  1.0791 9.26911 3.48299       ]]

Matriz d:
[[ 3.45607 6.29118 7.22065  8.5385 7.53844       ]
 [ 5.36445 2.84823 7.69362 5.21613 5.12165       ]
 [ 9.54241 7.69994 1.97452 6.39143 4.46712       ]]

Matriz 'd' insertada en la matriz 'c' a partir de la fila 0
[[ 3.45607 6.29118 7.22065  8.5385 7.53844       ]
 [ 5.36445 2.84823 7.69362 5.21613 5.12165       ]
 [ 9.54241 7.69994 1.97452 6.39143 4.46712       ]
 [ 9.60311 9.33146 5.85418 2.28104 5.15873       ]
```

```
 [ 3.11795 8.76016 2.88641 8.01691 8.59288       ]
 [ 9.97116 9.99725 6.50349 4.53194 3.39592       ]
 [ 3.67553  8.5613 1.21369 4.38279 1.83361       ]
 [ 7.09485 1.50594  1.0791 9.26911 3.48299       ]]

Matriz e:
[[       S       P       O       J       D       F       K       ]
 [       U       M       Y       S       I       E       Q       ]
 [       M       B       R       M       D       X       D       ]
 [       W       R       H       K       B       Y       R       ]
 [       D       V       U       O       E       E       U       ]
 [       L       D       M       S       K       G       O       ]]

Matriz f:
[[       R       S       S       L       D       J       U       ]
 [       A       M       Q       K       C       X       X       ]
 [       N       I       L       J       V       H       L       ]
 [       G       Y       H       S       O       E       T       ]]

Matriz 'f' insertada en la matriz 'e' a partir de la fila 6
[[       S       P       O       J       D       F       K       ]
 [       U       M       Y       S       I       E       Q       ]
 [       M       B       R       M       D       X       D       ]
 [       W       R       H       K       B       Y       R       ]
 [       D       V       U       O       E       E       U       ]
 [       L       D       M       S       K       G       O       ]
 [       R       S       S       L       D       J       U       ]
 [       A       M       Q       K       C       X       X       ]
 [       N       I       L       J       V       H       L       ]
 [       G       Y       H       S       O       E       T       ]]

Fila a insertar errónea!!
```

4. Añada a la biblioteca matrices una función sobrecargada que multiplique dos matrices de números reales o enteros y devuelva la matriz resultante.

La matriz resultante de la multiplicación de dos matrices "a" y "b", puede ser calculada con la expresión:

$$c_{ij} = \sum_{k=1}^{nca} a_{ik} \cdot b_{kj} \quad \begin{cases} i = 1 \rightarrow nfa \\ j = 1 \rightarrow ncb \text{, para cada i} \end{cases}$$

Como se puede ver, para programar esta expresión se requiere tres ciclos iterativos: el primero "i" que va desde la primera hasta la última fila de "a", el segundo "j" que va desde la primera hasta la última columna de "b" y el tercero "k" que va desde la primera hasta la última columna de "a".

Antes de llevar a cabo la multiplicación se debe verificar que las matrices sean multiplicables, es decir que el número de columnas de "a" sea igual al número de filas de "b" y de no ser así se debe generar un error.

Los prototipos de las funciones sobrecargadas, añadidas a "matrices.h" son:

```
//Devuelve el resultado de multiplicar las matrices "a" y "b"-
//Genera un error si las matrices no son compatibles
int **mulMat(int **a, int **b) throw(ERROR);
double **mulMat(double **a, double **b) throw(ERROR);
```

```
 double **mulMat(double **a, int **b) throw(ERROR);
 double **mulMat(int**a, double **b) throw(ERROR);
```

El código respectivo, añadido a "matrices.cpp", es:

```
int **mulMat(int **a, int **b) throw(ERROR){
  if (a==NULL || b==NULL) throw(ERROR_NULO);
  int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
  if (nca!=nfb) throw(ERROR_DIMENSIONES);
  int **c;
  crearMatriz(c,nfa,ncb);
  for (int i=0;i<nfa;i++)
    for (int j=0;j<ncb;j++){
      c[i][j]=0;
      for (int k=0;k<nca;k++)
        c[i][j]+=a[i][k]*b[k][j];
    }
  return c;
}
double **mulMat(double **a, double **b) throw(ERROR){
  if (a==NULL || b==NULL) throw(ERROR_NULO);
  int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
  if (nca!=nfb) throw(ERROR_DIMENSIONES);
  double **c;
  crearMatriz(c,nfa,ncb);
  for (int i=0;i<nfa;i++)
    for (int j=0;j<ncb;j++){
      c[i][j]=0;
      for (int k=0;k<nca;k++)
        c[i][j]+=a[i][k]*b[k][j];
    }
  return c;
}
double **mulMat(double **a, int **b) throw(ERROR){
  if (a==NULL || b==NULL) throw(ERROR_NULO);
  int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
  if (nca!=nfb) throw(ERROR_DIMENSIONES);
  double **c;
  crearMatriz(c,nfa,ncb);
  for (int i=0;i<nfa;i++)
    for (int j=0;j<ncb;j++){
      c[i][j]=0;
      for (int k=0;k<nca;k++)
        c[i][j]+=a[i][k]*b[k][j];
    }
  return c;
}
double **mulMat(int**a, double **b) throw(ERROR){
  if (a==NULL || b==NULL) throw(ERROR_NULO);
  int nfa=numfil(a), nca=numcol(a), nfb=numfil(b), ncb=numcol(b);
  if (nca!=nfb) throw(ERROR_DIMENSIONES);
  double **c;
  crearMatriz(c,nfa,ncb);
  for (int i=0;i<nfa;i++)
    for (int j=0;j<ncb;j++){
      c[i][j]=0;
      for (int k=0;k<nca;k++)
        c[i][j]+=a[i][k]*b[k][j];
    }
```

```
    return c;
  }
```

Como se puede ver en este caso se han creado 4 funciones sobrecargadas para permitir la multiplicación de matrices cuando ambas matrices son enteras, cuando ambas son reales, cuando la primera es real y la segunda entera y cuando la primera es entera y la segunda real.

Un programa que prueba estas funciones, incluyendo un caso de error, es:

```
#include <iostream>
#include "matrices.h"
using namespace std;
using namespace matrices;

int main(){
  try {
    //Prueba con matrices enteras
    int **a, **b, **c;
    int nfa=3,nca=4,nfb=4,ncb=5;
    crearMatriz(a,nfa,nca);
    llenarMatriz(a);
    cout<<"Matriz a:"<<endl;
    mostrarMatriz(a);
    crearMatriz(b,nfb,ncb);
    llenarMatriz(b);
    cout<<"Matriz b:"<<endl;
    mostrarMatriz(a);
    cout<<"Matriz c = a*b: "<<endl;
    c=mulMat(a,b);
    mostrarMatriz(c);
    //Prueba con matrices reales
    double **d, **e, **f;
    crearMatriz(d,nfa,nca);
    llenarMatriz(d);
    cout<<"Matriz d:"<<endl;
    mostrarMatriz(d);
    crearMatriz(e,nfb,ncb);
    llenarMatriz(e);
    cout<<"Matriz e:"<<endl;
    mostrarMatriz(e);
    cout<<"Matriz f = d*e: "<<endl;
    f=mulMat(d,e);
    mostrarMatriz(f);
    //Prueba con una matriz real y otra entera
    double **g;
    cout<<"Matriz g = d*b: "<<endl;
    g=mulMat(d,b);
    mostrarMatriz(g);
    //Prueba con una matriz entera y otra real
    double **h;
    cout<<"Matriz h = a*e: "<<endl;
    h=mulMat(a,e);
    mostrarMatriz(h);
    //Prueba cuando las matrices no son compatibles
    int **c1;
    cout<<"Matriz c1 = b*a: "<<endl;
    c1 = mulMat(b,a);
    mostrarMatriz(c1);
```

```
    liberarMatriz(h);
    liberarMatriz(g);
    liberarMatriz(c);
    liberarMatriz(a);
    liberarMatriz(b);
    liberarMatriz(f);
    liberarMatriz(d);
    liberarMatriz(e);
   } catch (ERROR e){
    cout<<"Error: Matrices no multiplicables."<<endl;
  }
  return 0;
}
```

Con el cual se obtienen los siguientes resultados:

```
Matriz a:
[[       1       6       2       8       ]
 [       6       5       4       9       ]
 [       8       7       2       8       ]]

Matriz b:
[[       1       6       2       8       ]
 [       6       5       4       9       ]
 [       8       7       2       8       ]]

Matriz c = a*b:
[[      83      67      65      79      89       ]
 [     124      98      86      85     105       ]
 [     136     104      88      95     101       ]]

Matriz d:
[[  5.0571  4.1691 1.51335 6.46916       ]
 [ 8.04987 8.22346 5.67895 3.71755       ]
 [ 8.88375 7.54009 9.60311 9.33146       ]]

Matriz e:
[[ 5.85418 2.28104 5.15873 3.11795 8.76016       ]
 [ 2.88641 8.01691 8.59288 9.97116 9.99725       ]
 [ 6.50349 4.53194 3.39592 3.67553  8.5613       ]
 [ 1.21369 4.38279 1.83361 7.09485 1.50594       ]]

Matriz f = d*e:
[[ 59.3326 80.1702  78.914 108.799 108.679       ]
 [ 112.307 126.319 138.292 154.345 206.948       ]
 [  147.55 165.131 160.342 204.384 249.471       ]]

Matriz g = d*b:
[[ 93.9178  73.952 63.8378 61.5708 72.2843       ]
 [ 122.906 84.6805 68.5808 112.212 99.8672       ]
 [ 167.542 125.091 107.323  133.82 150.588       ]]

Matriz h = a*e:
[[ 45.8892 94.5087 78.1768 127.055 97.9138       ]
 [ 86.4943 111.344 104.003 147.119 150.346       ]
 [ 89.7548 118.493 122.881 158.852 169.232       ]]

Matriz c1 = b*a:
Error: Matrices no multiplicables.
```

15.4. EJERCICIOS

1. Añada a la biblioteca "matrices" funciones sobrecargadas para intercambiar las filas de una matriz de números enteros, reales y de caracteres.

2. Añada a la biblioteca "matrices" funciones sobrecargadas que devuelvan en un vector los elementos de una fila de una matriz de números enteros, reales o de caracteres.

3. Añada a la biblioteca matrices funciones sobrecargadas que permita insertar las columnas de una matriz en la columna especificada dentro de otra matriz

4. Añada a la biblioteca "matrices" funciones sobrecargadas que devuelvan la suma de dos matrices de números enteros o reales.

5. Añada a la biblioteca "matrices" funciones sobrecargadas que devuelvan la transpuesta de una matriz de números enteros, reales o de caracteres.