# 2. Introducción a C++

Las cosas importantes.

## 2.1 Escribiendo un programa en C++

Considere el siguiente programa que calcula el cuadrado de un número ingresado por el usuario:

■ **Ejemplo 2.1** [Mi primer programa en C++] Este ejemplo muestra las diferentes elementos que componen un programa en C++.

```
#include <iostream>   //esto es una libreria
using namespace std;

int cuadrado(int n);  //este es el prototipo de una función

int main()        //esta es la función principal
{
    int num;      //esta es una variable
    cin >> num;   //esta es la función para capturar datos
    int res=cuadrado(num);    //aquí se llama a la otra función
    cout<<num<<" elevado al cuadrado es "<<res<<endl;   //esta es la
        función para imprimir datos
    return 0;
}

int cuadrado(int n)     //esta es la implementación de una función
{
    int r=n*n;
    return r;
}
```

- 

la salida del programa es...
Hay que introdicir las secciones empezando por lo de compilar...

## 2.2 La compilación

Cuando se diseña un programa en C++, el primer paso implica escribir el código del programa en un archivo de texto (archivo de código fuente). En general, se debe usar la extensión apropiada en el archivo de código fuente, para indicar que se trata de código C++. Dicha extensión de archivo, consiste de un punto seguido por un grupo de caracteres específicos.


En el caso de C++ las extensiones de los archivos pueden ser .c, .cc, .cxx, .cpp o .c++, dependiendo del compilador que se vaya a emplear. Tradicionalmente, la extensión más empleada es .cpp.

Antes de poder correr el programa es necesario traducir el archivo de código fuente en un archivo ejecutable, es decir, en un archivo que puede ser cargado en la memoria y ser ejecutado en el sistema. Este proceso involucra dos etapas principales:

1. El primer paso consiste en invocar un programa denominado compilador, el cual se encarga de traducir el código fuente en código objeto. El código objeto, está representado por uno o varios archivos binarios (dependiendo de cuantos archivos fuente compongan nuestro programa) que contienen instrucciones en lenguaje máquina que puede ejecutar el procesador del sistema.

2. En general, todo programa emplea una o varias librerías. Una librería contiene rutinas o funciones auxiliares qur eventualmente requiere emplear un programa como, por ejemplo, obtener la raíz cuadrada de un número o imprimir texto en la pantalla. Las librerías, existen en forma de código objeto y se hallan disponibles en el sistema. El segundo paso para obtener el archivo ejecutable final, consiste entonces en combinar el código objeto obtenido en la etapa de compilación, con el código objeto de las librerías respectivas. Esta etapa, la realiza un programa denominado enlazador o linker en inglés.

A continuación se ilustra, en la Figura 2.1, el proceso de compilación descrito anteriormente:
Hay que hacer el salto para explicar cada componente de un programa estándar.

## 2.3 Librerías

Para realizar un programa en C++ es necesario incluir las librerías que contienen las funciones básicas para ejecutar el programa. Incluir las librerías es lo primero que se debe hacer al realizar un programa. Las librerías de uso más común en C++ son:

- iostream: para manejar la entrada y la salida estándar, se suele especificar que se usará el namespace std.
- cstring: para manejar cadenas de caracteres tradicionales.
- string: contiene una clase para manejar cadenas de caracteres.
- cmath: para usar funciones matemáticas.

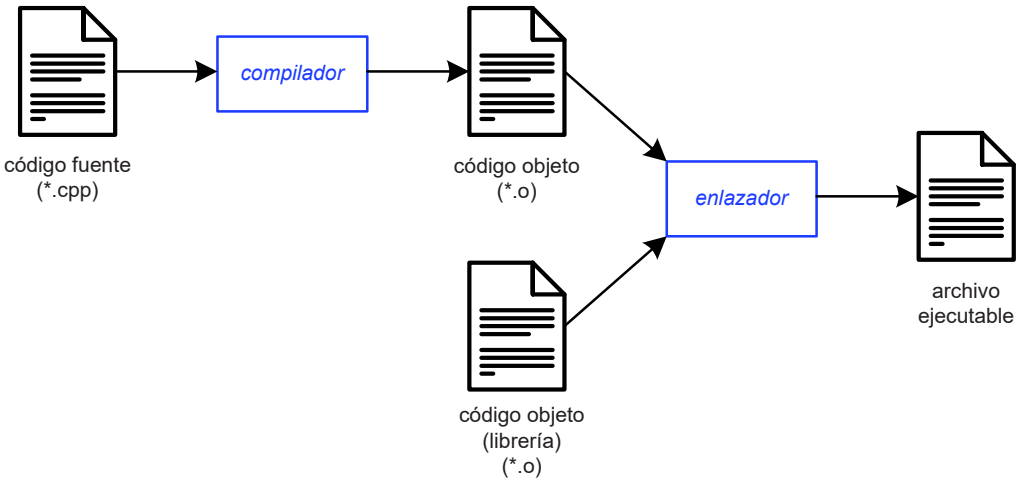


Figura 2.1: Proceso de compilación

- fstream: para leer y escribir archivos.

Para incluir una librería se usa la instrucción #include. Si es una librería estándar se coloca el nombre de la librería entre <>. Si se va a incluir una librería personalizada se coloca entre comillas dobles (“ ”).

La librerías suelen estar organizadas en secciones llamadas namespace, especificar el namespace que se va a usar en el programa hace más cortas las llamadas a las funciones de la librería. El namespace más común es el estancas (std). El siguiente ejemplo muestra la manera más común de iniciar un programa en C++.

■ **Ejemplo 2.2** [Definición de librerias] Encabezado típico de un programa en C++, donde se incluye la librería iostream y una librería externa definida por el usuario.

```
#include <iostream> //librería que siempre se incluye
#include "libpropia.h"//así se incluyen las librerías personalizadas
using namespace std;  /*se especifica que se va a usar el namespace
    std*/
```

■

## 2.4 Funciones

Una función es un conjunto de instrucciones, las cuales son ejecutadas secuencialmente cuando se ejecuta dicha función. La función principal es la función main, esta debe existir en todo programa de C++, es la función que se ejecuta al iniciar el programa y cuando esta se termina, también termina el programa. Esta función es entonces la encargada de llamar a todas las demás funciones que

componen el programa.

Además de la función main y las demás funciones creadas por el programador, existen muchas librerías que contienen funciones de utilidad para el programador. Este es el caso de las funciones de entrada y salida de datos disponibles en la librería iostream.

Más adelante se explicarán con mayor profundidad la función main y como crear otras funciones. De momento tenga en cuenta como iniciar la función main y las siguientes características del lenguaje:


En lenguaje C y C++, se debe poner ";" para indicarle al compilador el fin de una instrucción.


En lenguaje C y C++, los bloques de instrucciones deben ir encerradas entre llaves "{ }" y todas las instrucciones de una función deben estar agrupadas en un bloque.


En lenguaje C y C++ se usa la instrucción return para que una función devuelva un valor al ser ejecutada, el valor retornado debe corresponder al tipo de dato definido en el prototipo de la función.


En lenguaje C y C++ existen palabras reservadas que corresponden a instrucciones del lenguaje y no pueden ser utilizadas como nombres de funciones o nombres de variables. (citar el c++ reference) [**cppreference**].

### 2.4.1 Captura y visualización de datos

Las funciones cin y cout que se usan respectivamente para leer la entrada estándar y escribir la salida estándar, es decir, la terminal. Estas funciones hacen parte del espacio de nombres std de la librería iostream. Para asignar la variable en que cin almacena el valor ingresado se usa el operador $>>$. Para asignar la información a mostrar en terminal con cout se usa el operador $<<$, este operador puede ser implementado en cascada para asignar varios datos de diferentes tipos en una sola línea de código. A continuación se verá un ejemplo sencillo del uso de estas funciones.

En la función main del ejemplo inicial del capítulo se puede ver un ejemplo claro del uso de estas funciones. Repasemos el uso de estas funciones con el siguiente ejemplo.

■ **Ejemplo 2.3** [Captura y visualización de datos]...

```
int a;
cin>>a;   //lee un entero y lo almacena en a
cout<<"El valor ingresado es: "<<a<<endl; /*imprime el texto seguido
    del valor de a y un salto de línea*/
```

■

## 2.5 Variables

Una variable es un espacio de memoria que se reserva para guardar información. Este espacio de memoria está identificado por una dirección de memoria. Para facilitar el manejo de las variables, el programador le asigna un nombre, el cual debe ser único en el ámbito de dicha variable (el bloque de instrucciones donde fue declarada).

### 2.5.1 Declaración e inicialización de variables

Para declarar, crear, una variable basta con especificar el tipo de variable seguido del nombre que se dará a la variable. Si se van a usar modificadores, se usa primero el de signo y después el de tamaño. Inicializar una variable es asignarle un valor inicial, esto se puede hacer en la misma línea que se declara. Además es posible declarar e inicializar varias variables del mismo tipo en una sola línea, esto se logra colocando una sola vez el tipo de variable y separando las variables por comas.

En el ejemplo inicial del capítulo se puede observar la declaración de las variables num, res, y r, en este caso todas variables tipo int. Veamos a continuación un ejemplo más completo.

■ **Ejemplo 2.4** [Inicialización de variables] A continuación se muestra la forma como se defines variables de tipo int y char. De igual forma se muestra el uso de los modificadores de signo y tamaño.

```
int contador=0;//crea un entero llamado contador y lo inicializa en
    0
unsigned long long int largo; /*crea un entero sin signo de 8 bytes
    llamado largo.*/
char a='a',c,z='z'; /*crea las variables tipos carácter  a, c y z e
    inicializa a y z, pero no c.*/
```

■

### 2.5.2 Ámbito de una variable

Debido a que las variables ocupan espacio de memoria, normalmente se les da un ámbito limitado, es decir, solo pueden ser usadas dentro del bloque de instrucciones en que fueron declaradas, ya que fuera de este bloque se libera el espacio de memoria. Estos bloques de instrucciones generalmente corresponden a funciones o estructuras de control. Este tipo de variables son conocidas como variables locales.

Existe otro tipo de variables conocido como variables globales, estas variables son visibles desde cualquier función del programa y existen desde el inicio hasta el fin del programa. Estas variables se crean al declararlas fuera de un bloque de instrucción, es decir, fuera de una función.

En el ejemplo inicial del capítulo se puede observar claramente las limitaciones de ámbito de las variables, ya que r es una variable local de la función cuadrado, es necesario crear una variable res para almacenar el valor que la función retorna y poder usarlo en la función main. Esto se podría evitar con una variable global, pero generalmente no se recomienda usar variables globales ya que malgastan memoria, hacen más difícil el seguimiento del programa y facilitan la aparición de bugs.

### 2.5.3 Sensibilidad a mayúsculas y minúsculas

Una característica importante de C++ es que es un lenguaje sensible a mayúsculas y minúsculas.

■ **Ejemplo 2.5** [Sensibilidad a las mayúsculas (case sensitive)]...

```
int a=1;
int A=2;
//a y A son variables diferentes
```

■

### 2.5.4 Caracteres y cadenas de caracteres

En C++ los caracteres simples se encierran en comillas simples (’ ’), mientras que las cadenas de caracteres se encierran en comillas dobles (“ ”).

■ **Ejemplo 2.6** [Uso de caracteres]... **Ej:**
’A’ y “Arriba”.

```
char c='A'; //es un carácter.
char m[]="Arriba";  //es una cadena de caracteres.
```

■

### 2.5.5 Comentarios

En C++ existen dos formas de colocar comentarios, los comentarios de línea y los bloques de comentarios. Los primeros son comentarios que se inician con // y se extienden hasta el final de la línea, la siguiente línea vuelve a ser parte del programa. Los bloques de comentarios pueden extenderse un número indeterminado de líneas, se abre con /* y se cierran con */, todo lo que este contenido entre esos símbolos se considera comentario.

■ **Ejemplo 2.7** [Comentando el código]...

```
instruccion1; //Esto es un comentario de una linea
instruccion2; /*Esto
                también es
                un comentario*/
```

■

### 2.5.6 Valores lógicos ...

En C++ existen los valores lógicos true y false. Un valor se considera falso (false) cuando es cero, es decir, todos los bits que lo componen son 0, en caso contrario se considera verdadero (true), es decir, al menos uno de sus bits es 1.

■ **Ejemplo 2.8** [Valores lógicos]

```
int a=1;  //tiene el valor lógico: verdadero
int b=0;  //tiene el valor lógico: falso
int c=15; //tiene el valor lógico: verdadero
```

■

## 2.6 Tipos de datos

En C++ es necesario definir el tipo de cada variable que se va a usar. El tipo de variable define el espacio de memoria que esta ocupa y la forma como se interpreta la información contenida en ella.

### 2.6.1 Tipos estándar

Los tipos de variable estándar de C++ son descritos en la Tabla 2.1.

| Nombre | Bytes | Rango |
|---|---|---|
| char | 1 | -127 a 127 |
| char16_t | 2 | -32767 a 32767 |
| char32_t | 4 | -2147483648 a 2147483647 |
| wchar_t | Normalmente 2 | 0 a 65,535 |
| int | Normalmente 4 | -2147483648 a 2147483647 |
| float | Normalmente 4 | +/-3.4E+/-38 (7 digitos) |
| double | 8 | +/-1.7E+/-308(15 digits) |
| bool | 1 | true o false |
| void | 0 | Sin valor |

Tabla 2.1: Tipos de variable estándar de C++.

Los tipos char, char16_t, char32_t, wchar_t e int corresponden a números enteros, por lo que en las operaciones matemáticas son tratados como números enteros. Un int se imprime como número entero, pero los tipos char se imprimen como los carácteres ASCII correspondientes a su valor.

El tipo char se usa para caracteres UTF-8, el char16_t para UTF-16, el char32_t para UTF-32 y el wchar_t para otros conjuntos de caracteres extendidos.

Los tipos float y double corresponden a números de punto flotante, es decir, números que son expresados como un número decimal multiplicado por una potencia de 10. La precisión y el rango del double son mayores que los del float. Al ser imprimidos se muestra su representación como potencia únicamente cuando es un número muy grande o muy pequeño para ser impreso cómodamente en representación tradicional.

El tipo bool se usa para almacenar valores lógicos, estos se usan como condiciones en estructuras de control.

El tipo void se usa para especificar que una función no retorna ningún valor o que no necesita parámetros de entrada. No se pueden definir variables tipo void.

### 2.6.2 Modificadores de signo

C++ trata por defecto a todas las variables como datos con signo. Es posible utilizar los modificadores signed y unsigned para modificar como son interpretados estos datos y, por lo tanto, modificar su rango. Estos modificadores se pueden usar sobre las variables tipo char e int, definiendo entonces los nuevos tipos de variables tal como se muestra en la Tabla 2.2.

| Nombre | Bytes | Rango |
|---|---|---|
| signed char | 1 | -127 a 127 |
| unsigned char | 1 | 0 a 255 |
| signed int | 4 | -2147483648 a 2147483647 |
| unsigned int | 4 | 0 a 4,294,967,295 |

Tabla 2.2: Modificadores de signo en C++.

### 2.6.3 Modificadores de tamaño

El tamaño de una variable tipo int puede ser aumentado o disminuido usando los modificadores short y long, con estos modificadores se obtienen los siguientes tipos de variables mostrados en la Tabla 2.3:

| Nombre | Bytes | Rango |
|---|---|---|
| short int | 2 | –32,768 a 32,767 |
| long int | 4 | 0 a -2147483648 a 2147483647 |
| long long int | 8 | –9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 |

Tabla 2.3: Modificadores de tamaño en C++.

Se puede observar que long int es equivalente a int. Para definir variables de estos tipos no es necesario colocar int, con short, long o long long basta. Estos tipos de variables también pueden ser modificados usando signed y unsigned.

También existe el long double, pero este es exactamente igual a un double normal.

## 2.7 Operadores

### 2.7.1 Aritméticos

Corresponden a las operaciones matemáticas básicas entre dos valores.

- Adición (+): suma dos cantidades.
- Substracción (-): resta dos cantidades.
- Multiplicación (*): multiplica dos cantidades.
- División (/): divide dos cantidades, no aproxima el resultado de divisiones entre enteros.
- Modulo (%): retorna el modulo (residuo) de la división de dos cantidades.

■ **Ejemplo 2.9** [Uso de los operadores aritméticos]...

```
int a=3, b=8;
b+a;  //el resultado es 11
b-a;  //el resultado es 5
b*a;  //el resultado es 24
b/a;  //el resultado es 2
b%a;  //el resultado es 2
```

■

### 2.7.2 Incremento y decremento

Los operadores incremento (++) y decremento (--) se usan para cambiar en una unidad al valor de una variable. Si preceden al nombre de la variable tienen el efecto de predecremento y preincremento, es decir, primero cambian el valor de la variable y después se realizan operaciones con ella. Por el contrario si están después de la variable, primero realizan las demás operaciones con la variable y luego realizan el incremento o decremento.

■ **Ejemplo 2.10** [Uso de los operadores de incremento y decremento]...

```
int a=3, b=0;
b=a++;  //al realizar la operación, b=3 y a=4
a=++b;  //al realizar la operación, b=4 y a=4
```

■

### 2.7.3 Relación y comparación

Se usan para comparar dos valores, retornan verdadero (true) si la condición se cumple y falso (false) en caso contrario.

- Igual a (==)
- No igual a (!=)
- Mayor que (>)
- Menor que (<)
- Mayor o igual que (>=)
- Menor o igual que (<=)

■ **Ejemplo 2.11** [Uso de los operadores de relación y comparación]...

```
int a=3, b=8;
b==a; //el resultado es falso
b!=a; //el resultado es verdadero
b>a;  //el resultado es verdadero
b<a;  //el resultado es falso
b>=a; //el resultado es verdadero
b<=a; //el resultado es falso
```

■

### 2.7.4 Lógicos

Se usan para realizar operaciones entre valores lógicos (verdadero o falso) y retornan un valor lógico.

- Y (&&): retorna verdadero si ambos operandos son verdaderos, en caso contrario retorna falso.
- O (||): retorna falso si ambos operandos son falsos, en caso contrario retorna verdadero.
- No (!): niega el valor lógico.

■ **Ejemplo 2.12** [Uso de los operadores lógicos]...

```
bool a=true, b=false;
b&&a; //el resultado es falso
a&&a; //el resultado es verdadero
b||a; //el resultado es verdadero
b||b; //el resultado es falso
```

```
!a;   //el resultado es falso
!b;   //el resultado es verdadero
```

■

### 2.7.5 Bit a bit

Estos se usan para realizar operaciones lógicas bit a bit.

- Y (&): el bit del resultado es 1 si los bits de ambos operandos son 1, en caso contrario el resultado es 0.
- O (|): el bit del resultado es 0 si los bits de ambos operandos son 0, en caso contrario el resultado es 1.
- O exclusiva ($\wedge$): el bit del resultado es 1 si los bits de ambos operandos son diferentes, en caso contrario el resultado es 0.
- No ($\sim$): niega el valor del bit.
- Corrimiento a izquierda ($<<$): mueve los bits una cantidad de posiciones a la izquierda y coloca un 0 en los bits más a la derecha.
- Corrimiento a derecha ($>>$): mueve los bits una cantidad de posiciones a la derecha y coloca un 0 en los bits más a la izquierda.

■ **Ejemplo 2.13** [Operaciones a nivel de bit]...

```
char a=15, b=85;//en representación de bits: a=00001111, b=01010101
a&b;  //el resultado es 00000101
a|b;  //el resultado es 01011111
a^b;  //el resultado es 01011010
~b;   //el resultado es 10101010
b<<2; //el resultado es 01010100
b>>2; //el resultado es 00010101
```

■

### 2.7.6 Asignación

Se usan para asignar valor a una variable, además de la asignación simple (=), también existe la asignación compuesta (+=, -=, *=, /=, %=, $>>=$, $<<=$, &=, $\wedge=$, |=), esta permite usar el valor actual de la variable en una operación para asignarle un nuevo valor.

■ **Ejemplo 2.14** [Operaciones de asignación]...

```
int a=3, b=8;
b=a;  //entonces b=3
b+=a; //entonces b=11
b-=a; //entonces b=5
b<<=a;  //entonces b=64
b>>=a;  //entonces b=1
```

■

### 2.7.7 Otros

- Agrupación (()): se usa para agrupar expresiones, dando un orden de ejecución a las operaciones.

- Índice ([ ]): se usa para acceder a los datos de un arreglo.
- Scope (::): se usa para acceder a miembros de una clase o de un espacio de nombres.
- Acceso a miembros (. o ->): se usa para acceder a miembros de un objeto o una estructura, "." se usa para acceder desde el objeto o estructura y "->" para acceder a través de un apuntador.
- Nuevo (new): se usa para reservar espacio en memoria dinámica, se define el tipo de dato a almacenar en dicho espacio y el número de espacios a reservar. Retorna un puntero al espacio reservado.
- Eliminar (delete): se usa para liberar espacio en memoria dinámica, recibe el puntero generado por new.
- Tamaño de (sizeof): retorna el tamaño en bytes que ocupa una variable.
- Ternario (?:): forma resumida de un if, si se cumple una condición retorna un valor, si no se cumple retorna otro.
- Coma (,): se usa para separar dos expresiones donde solo se espera una.
- Casting ((tipo)): convierte un valor a un tipo diferente. Se usa para evitar perdida de información en operaciones y para adaptar entradas y salidas de funciones.

■ **Ejemplo 2.15** [Otros operadores]...

```
(1+2)*3;       //realiza primero 1+2 y luego multiplica por 3.
arreglo[2];      //retorna el tercer elemento de arreglo.
std::cout<<"algo";  /*accede a cout que está definido en el
   namespace std*/
objeto.atributo;  //accede a atributo desde el objeto
puntero->atributo;  /*accdede a atributo a través de un puntero al
   objeto*/
int *ptr=new int[3];/*reserva espacio para 3 datos int, ptr apunta
   al primero de estos datos*/
delete[] ptr;    //libera la memoria anteriormente reservada
sizeof(int);     //retorna el tamaño de una variable tipo int
a>4?2:1;       //si a>4 retorna 2, en caso contrario retorna 1
int a=0,b=2;     //crea dos variables enteras y las inicializa
(int)b;        /*trata a b como si fuese int, independientemente del
   tipo que sea b.*/
```

■

### 2.7.8 Precedencia

Una expresión puede tener múltiples operadores, el orden en que las operaciones son realizadas está determinado por la precedencia de los operadores. Los operadores organizados en orden de mayor precedencia a menor precedencia es la siguiente:

1. Scope.
2. Posincremento/posdecremento, agrupación, índice, acceso a miembros.
3. Preincremento/predecremento, negación lógica, negación por bit, new, delete, sizeof, casting, referencia.
4. Escalamiento (multiplicación, división, modulo)
5. Adición (suma y resta)
6. Corrimientos

7. Comparaciones (mayor, menor, mayor igual, menor igual)
8. Igualdad (igual, no igual)
9. Y bit a bit
10. O exclusiva
11. O bit a bit
12. Y lógica
13. O lógica
14. Asignación simple y compleja, operador ternario

## 2.8 Control de flujo

Las estructuras de control de flujo que se veran a continuacion permiten modificar el orden en que las instrucciones del programa son ejecutadas.

### 2.8.1 if

Se usa para ejecutar una secuencia de instrucciones si se cumple una determinada condición. La forma general de un if en C++ es la siguiente:

**Sintaxis 2.8.1 — if.** ...

```
if(condición 1){
  //instrucciones a ejecutar si se cumple la condición 1
}
else if(condición 2){
  /*instrucciones a ejecutar si NO se cumple la condición 1 y SI
     se cumple la condición 2*/
}
else if(condición 3){
  /*instrucciones a ejecutar si NO se cumplen la condiciones 1 o
     2, pero SI se cumple la condición 3*/
}
else{
  /*instrucciones a ejecutar si no se cumple ninguna de las
     condiciones anteriores*/
}
```

Se pueden colocar tantos else if como sea necesario. Si el else o el else if no son necesarios, no es necesario colocarlos.

C++ Luego de definir la condición de las estrcuturas de control, se abre una llave "{" y se cierra "}" cuando se terminan las instrucciones dentro de esa estructura de control. Si dentro de la estructura de control solo va una instrucción, el uso de llaves es opcional

```
if(x>0)
{       //inicio de estructura de control.
  x--;  //instrucción 1.
  y--;  //instrucción 2.
```

```
}       //fin de estructura de control.

if(x>0) //inicio de estructura de control.
  x--;  //instrucción 1.
        //fin de estructura de control.
```

■ **Ejemplo 2.16** [Uso del if] Suponiendo que a sea un int...

```
if(a==0){
  cout<<"cero";
}
else if(a>0){
  cout<<"mayor que cero";
}
else{
  cout<<"menor que cero";
}
```

■

### 2.8.2 switch

Se usa para ejecutar una secuencia de instrucciones según el valor actual de una variable. La forma general de un switch en C++ es la siguiente:

**Sintaxis 2.8.2 — switch.** ...

```
switch(variable){
  case valor1:{
    //instrucciones a ejecutar si variable==valor1
    break;
  }
  case valor2:{
    //instrucciones a ejecutar si variable==valor2
    break;
  }
  default:{
    /*instrucciones a ejecutar si variable es diferente de todos
       los valores definidos*/
    break;
  }
```

La instrucción puede ser usada con cualquier tipo de variable y los valores de los case deben ser valores válidos para este tipo de variable. Puede existir tantos case como sea necesario, mientras que los valores no se repitan. Si no es necesario tener un caso por defecto, puede omitirse el default.

■ **Ejemplo 2.17** [Uso del switch] Suponiendo que a sea un char...

```
switch(a){
  case 'a':{
    cout<<"es a";
```

```
    break;
  }
  case 'b':{
    cout<<"es b";
    break;
  }
  default:{
    cout<<"no es a ni b";
    break;
  }
}
```

■

### 2.8.3 while

Se usa para ejecutar una secuencia de instrucciones continuamente, mientras que se cumpla una condición. La forma general de un while en C++ es la siguiente:

**Sintaxis 2.8.3 — while.** ...

```
while(condición){
  //secuencia de instrucciones
}
```

■ **Ejemplo 2.18** [Uso del while]...

```
int a=7;
while(a>0){
  cout<<a<<" es mayor a 0";
  a--;
}
```

■

### 2.8.4 do while

Es similar a un ciclo while, se diferencian en que el do while evalúa la condición luego de ejecutar la secuencia de instrucciones y el while lo hace antes de ejecutarla. La forma general de un do while en C++ es la siguiente:

**Sintaxis 2.8.4 — do while.** ...

```
do{
  //secuencia de instrucciones
}while(condición);
```

Debido a que la condición se evalúa al final, la secuencia de instrucciones se ejecuta al menos una vez.

■ **Ejemplo 2.19** [Uso del do while]...

```
int a=7;
```

```
do{
  cout<<a<<" es mayor a 0";
  a--;
}while(a>0);
```

■

Si el valor inicial de a no cumpliera la condición a>0, las instrucciones se ejecutarían una vez. Por esto un ciclo do while no es una buena elección para lo que debería hacer este programa.

### 2.8.5 for

Es una variante del ciclo while. Suele usarse para ejecutar una secuencia de instrucciones una cantidad determinada de veces. La forma general de un for en C++ es la siguiente:

**Sintaxis 2.8.5 — for.** ...

```
for(instrucción 1; condición; instrucción 2){
  //secuencia de instrucciones
}
```

La instrucción 1 se ejecuta antes de entrar en el ciclo, la instrucción 2 es la última que se ejecuta en cada recorrido del ciclo, el cual se ejecuta sucesivamente mientras que se cumpla la condición. El while equivalente al ciclo for es el siguiente:

```
Instrucción 1;
while(condición){
  //secuencia de instrucciones
  Instrucción 2;
}
```

■ **Ejemplo 2.20** [Uso del for]...

```
for(int a=7;a>0;a--){
  cout<<a<<" es mayor a 0";
}
```

■

## 2.9 Ejemplos

■ **Ejemplo 2.21** [Conversión de segundos a formato hora]...

El siguiente programa lee constantemente una cantidad de segundos y lo convierte e imprime en fromato horas:minutos:segundos. Para salir del programa se ingresa 0. Ejemplo: Si el usuario ingresa 7587 el programa imprime: 2:6:27.

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

int main(){                 //función principal
    int input;              //variable para almacenar el dato ingresado
```

```
    int segundos;         //variable para almacenar los segundos
    int minutos;          //variable para almacenar los minutos
    int horas;            //variable para almacenar las horas

    cout<<"Ingrese la cantidad de segundos a convertir\nPara
       terminar ingrese 0\n";
    cin>>input;           //pide el dato y lo almacena en input

    while(input!=0){    //ejecuta el while si la función retorna
       true
        segundos=input%60;
        minutos=(input/60)%60;
        horas=input/(3600);

        cout<<"La hora correspondiente a "<<input<<" segundos\n";
        cout<<"Es: "<<horas<<":"<<minutos<<":"<<segundos<<endl; //
           imprime el dato convertido
        cout<<"Ingrese la cantidad de segundos a convertir\nPara
           terminar ingrese 0\n";
        cin>>input;   //y pide un dato nuevo nuevo
    }
    return 0;
}
```

■

## 2.10 En Resumen

Enfatizar en lo que definitivamente debe quedar claro antes de continuar...

## 2.11 Para prácticar

### 2.11.1 Ejercicios

**Ejercicio 2.1** Escriba un programa que pida dos números A y B e imprima en pantalla el residuo de la división A/B.
Ej: si se ingresan 8 y 3 se debe imprimir:
El residuo de la division 8/3 es: 2
■

**Ejercicio 2.2** Escriba un programa que pida un número N e imprima en pantalla si es par o impar.
Ej: si se ingresa 5 se debe imprimir:
5 es impar.
■

**Ejercicio 2.3** Escriba un programa que pida dos números A y B e imprima en pantalla el mayor.
Ej: si se ingresan 7 y 3 se debe imprimir:
El mayor es 7