CAPÍTULO 2

# Conceptos básicos de los programas en C++



## Introducción

Este capítulo comienza con un repaso de los conceptos teóricos y prácticos relati vos a la estructura de un programa en C++ dada su gran importancia en el desarrollo de aplicaciones; se describen también los elementos básicos que componen un programa; tipos de datos en C++ y cómo se declaran; concepto de constantes y su declaración; concepto y declaración de v ariables; tiempo de vida o duración de v ariables; operaciones básicas de entrada/salida y espacios de nombres.

## 2.1. Estructura general de un programa en C++

Un programa en C++ se compone de una o más funciones, de las cuales una debe ser obligatoriamente `main( )`. Una función en C++ es un grupo de instrucciones que realizan una o más acciones. Un programa contiene una serie de directivas `#include` que permiten incluir en el mismo, archivos de cabecera que a su v ez contienen funciones y datos predefinidos en ellos.



**EJEMPLO 2.1.** *Estructura de un programa C++.*

```
#include <iostream.h>          archivo de cabecera  iostream.h (o simplemente, <iostrean>)

int main( )                    cabecera de función
{                              nombre de la función
...                            sentencias de la función
}
int funcl ( )
{
...
}
int func2 ( )
{
...
}
```

Un programa C++ puede incluir: directivas de preprocesador; declaraciones globales; la función `main( )`; funciones definidas por el usuario; comentarios del programa.

**EJEMPLO 2.2.** *Primer programa en C++.*

```
#include <iostream>
using namespace std;
// Este programa visualiza Bienvenido a la programacion en C++
// Pronto se van a escribir programas en  C++
int main()
{
    cout << "Bienvenido a la programacion en C++\n";
    cout << "Pronto se van a escribir programas en C++\n";
    system("PAUSE");     //si no compila con Dev C++ sustituya esta línea
    return EXIT_SUCCESS; //y la siguiente por la sentencia
}                        //count << "Presione ENTER (INTRO) para terminar"; cin.get(); return 0
```

La directiva `#include <iostream>` se refiere a la inclusión del archi vo externo denominado `iostream.h` en el que se declaran las funciones y datos que contiene iostream, entre otros, la información relativa al objeto `cout`. `using namespace std` sirve para usar las declaraciones estándar del espacio nombrado (consola) . Las dobles barras inclinadas `(//)` sirven para indicar comentarios que son ignorados por el compilador . La sentencia `main( )` es obligatoria en cada programa C++, e indica el comienzo del programa. Las llaves { ... } encierran el cuerpo de la función`main( )` y son necesarias en todos los programasC++.

```
cout << "Bienvenido a la programacion en C++\n";
cout << "Pronto se van a escribir programas en C++\n";
```

Las sentencias anteriores en vían los mensajes a la salida estándar mediante el objeto `cout`. El símbolo `<<` se denomina *operador de salida* u *operador de inserción*. Inserta el mensaje en el flujo de salida. El símbolo `'\n'` es el símbolo de *nueva línea*. La sentencia `system("PAUSE")` obliga a que se visualice el resultado de la ejecución hasta que se pulse una tecla. La sentencia `return EXIT_SUCCESS`, retorna el control al sistema operati vo, una vez terminada la ejecución.

## 2.1.1. DIRECTIVAS DEL PREPROCESADOR

El *preprocesador* en un programa C++ consta de *directivas* que son instrucciones al compilador. Todas las directivas del *preprocesador* comienzan con el signo de *libro* o "*almohadilla*" (#) y no terminan en punto y coma, ya que no son instrucciones del lenguaje C++.

La directiva `#include` indica al compilador que lea el archivo fuente (archivo cabecera o de inclusión) que viene a continuación de ella y su contenido lo inserte en la posición donde se encuentra dicha directi va. Estos archivos se denominan *archivos de cabecera* o *archivos de inclusión*. Estas instrucciones son de la forma `#include<nombrearch.h>` *o bien* `#include "nombrearch.h"`. El archivo de cabecera más frecuente es `iostream.h`, ya que proporciona al compilador C++ la información necesaria sobre las funciones de biblioteca `cin` y `cout`, así como otras rutinas de biblioteca que realizan operaciones de entrada y salida.



La directiva `#define` indica al preprocesador que defina un *item* de datos u operación para el programa C++. Por ejemplo, la directiva `#define TAM 10` sustituirá el valor `10` cada vez que `TAM` aparezca en el programa.

## 2.1.2. DECLARACIONES GLOBALES

Las *declaraciones globales* indican al usuario que las constantes o variables así declaradas son comunes a todas las funciones de su programa. Se sitúan antes de la función `main( )`. La zona de declaraciones globales puede incluir declaraciones de v ariables además de declaraciones de prototipos de función. Un prototipo es una declaración de una función y consiste en una definición de la función sin cuerpo y terminado con un punto y coma.

**EJEMPLO 2.3.** *Una estructura modelo que incluye declaraciones globales y prototipos de funciones.*

```
#include <iostream.h>

                                        //Definir macros
#define pi 3.14 15

                                        //Declaraciones globales
```

```
float  radio, altura;
                                                    //prototipo de función
float area_circulo( float radio);
float volumen_cilindor( float radio, float altura);

int main()
{
    // ...
}
```

## 2.1.3. FUNCIÓN `main( )`

Cada programa C++ tiene una función `main( )` que es el punto inicial de entrada al programa. Su estructura es:

```
int main( )
{
    ...  bloque de sentencias
}
```



Además de la función `main()`, un programa C++ consta de una colección de *subprogramas* (en C++ siempre son funciones). Todas las sentencias de C++ situadas en el cuerpo de la función `main( )`, o de cualquier otra función, deben terminar en punto y coma.

**EJEMPLO 2.4.** *Programa que visualiza el nombre y la dirección.*

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

int main(int argc, char *argv[])
{
  cout << "Lucas Sánchez García\n";
  cout << "Calle Marquillos de Mazarambroz, \n";
  cout << "Mazarambroz, TOLEDO \n";
  cout << "Castilla la Mancha, ESPAÑA\n";
  system("PAUSE");
  return EXIT_SUCCESS;
}
```



**EJEMPLO 2.5.** *Un ejemplo de algoritmo que incluye un pr ototipo de una función, su codificación y una llamada.*

```
#include <cstdlib>                          //cstdl:b contiene funciones de entrada y salida
#include <iostream>
using namespace std;

void prueba( );

int main(int argc, char *argv[])
{
   prueba( );
   system("PAUSE");
   return EXIT_SUCCESS;
}

void prueba( )
{
    cout << "Mis primeros pasos  en C++ \n";
}
```

La ejecución del programa anterior es:

```
Mis primeros pasos  en C++
```

## 2.1.4. FUNCIONES DEFINIDAS POR EL USUARIO

C++ proporciona funciones predefinidas que se denominan *funciones de biblioteca* y otras *definidas por el usuario*. Las *funciones de biblioteca* requieren que se incluya el archi vo donde está su declaración tales como `matht.h` o `stdio.h`. Las *funciones definidas por el usuario* se invocan por su nombre y los parámetros actuales opcionales que tengan. Estos parámetros actuales deben coincidir en número orden y tipo con los parámetros formales de la declaración. Después de que la función sea llamada, el código asociado con la función se ejecuta y , a continuación, se retorna a la función llamadora. En C++, las funciones definidas por el usuario requieren una *declaración* o *prototipo* en el programa, que indica al compilador el tipo de la función, el nombre por el que será invocada, así como el número y tipo de sus argumentos. Todos los programas C++ constan de un conjunto de funciones que están controladas por la función `main( )`.



**EJEMPLO 2.6.** *El algoritmo siguiente incluye un prototipo de una función* `suma`*, un programa* `main` *y con sentencias y una llamada a la función, así como la codificación de la función* `suma`*.*

```
#include <cstdlib>
#include <iostream>
using namespace std;
float suma(float a, float b);

int main(int argc, char *argv[])
{
   float numero, numero1, sumadenumeros;

   numero = 2;
   numero1 = 3;
   sumadenumeros = suma(numero, numero1);
   cout << " la suma de "  << numero << " y " << numero1 << endl;
   cout << " es : " << sumadenumeros << endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
float suma(float a, float b)
{
   return a + b;
}
```



El resultado de ejecución del programa anterior es:

```
la suma de 2 y  3
 es : 5
```

## 2.1.5. COMENTARIOS

Un *comentario* es cualquier información añadida a su archi vo fuente e ignorada por el compilador . En C estándar los comentarios comienzan por `/*` y terminan con la secuencia `*/`. En C++ se def ine una línea de comentario comenzando con una doble barra inclinada ( `//`). Todo lo que viene después de la doble barra inclinada es un comentario y el compilador lo ignora.

> Los archivos de cabecera en C++ tienen normalmente una e xtensión `.h` y los archivos fuente, la extensión `.cpp`.

**EJEMPLO 2.7.** *Programa con comentarios en C y C++.*

```
#include <iostream.h>                    // archivo cabecera donde se encuentra cout
using namespace std;                     // usa la consola como dispositivo estándar

int main(int argc, char *argv[])         // número de argumentos y cuales
{
   // este programa sólo muestra en pantalla el mensaje
   //hola mundo cruel
   cout << " Hola mundo cruel";          // visualiza el mensaje
   system("PAUSE");                      // archivo de ejecución retenido hasta pulsar tecla
   return EXIT_SUCCESS;                  // retorna el control al sistema operativo.
}
```



# 2.2. Los elementos de un programa en C++

Los elementos básicos de un programa C++ son: **identificadores**; **palabras reservadas**; **comentarios**; **signos de puntuación**; **separadores** y **archivos cabecera**.

## 2.2.1. IDENTIFICADOR

Un *identificador* es una secuencia de caracteres, letras, dígitos y subrayados ( _ ) que comienza siempre por un carácter . Las letras mayúsculas y minúsculas son diferentes. Pueden tener cualquier longitud, pero el compilador ignora a partir del 32. No pueden ser palabras reservadas. Son ejemplos de identificadores los siguientes:

```
nombre_Alumno     Letra_indice     Dia          Mayor     menor
Fecha_Compra      alfa             Habitacion24 i         j
```

## 2.2.2. PALABRAS RESERVADAS

C++ como la mayoría de los lenguajes tiene reserv ados algunos identificadores con un significado especial, que sólo pueden ser usados con ese cometido. Una *palabra reservada,* tal como `void` (ausencia de tipo, o tipo genérico), es una característica del lenguaje C++. Una palabra reserv ada no se puede utilizar como nombre de identif icador, objeto o función. Son palabras reservadas de C++ las siguientes:



```
asm          delete       if           return       try
auto         do           inline       short        typedef
bool         double       int          signed       union
break        else         longmutable  sizeof       unsigned
case         enum         namespace    static       virtual
catch        explicit     new          struct       void
char         extern       operator     switch       volatile
class        float        private      template     wchar_t
const        for          protected    this         while
continue     friend       public       throw
default      goto         register
```

## 2.2.3. COMENTARIOS

Un comentario, estilo C++ comienza con `//` y termina al final de la línea en que se encuentra el símbolo. Un comentario, estilo C y soportados también por C++, encerrados entre `/*` y `*/` puede extenderse a lo largo de varias líneas.

### 2.2.4. SIGNOS DE PUNTUACIÓN Y SEPARADORES

Todas las sentencias de C++ deben terminar con un punto y coma (`;`). Los separadores son espacios en blanco, tabulaciones, retornos de carro y avances de línea. Otros signos de puntuación son:

```
!    %    ^    &    *    (    )    -    +    =    {    }    ~
[    ]    \    ;    ´    :    <    >    ?    ,    .    /
```

### 2.2.5. ARCHIVOS DE CABECERA

Un *archivo de cabecera* es un archivo especial que contiene las declaraciones de objetos y funciones de la biblioteca que son añadidos en el lugar donde se insertan. Un archivo de cabecera se inserta con la directiva `#include`. El nuevo ANSI C++ ha cambiado el convenio (notación) original de los archivos de cabecera. Es posible utilizar sólo los nombres de las bibliotecas sin el sufijo `.h`; es decir, se puede usar `iostream`, `cmath`, `cassert` y `cstlib` en lugar de `iostream.h`, `math.h`, `assert.h` y `stdlib.h` respectivamente.



## 2.3. Tipos de datos en C++

C++ no soporta un gran número de tipos de datos predefinidos, pero tiene la capacidad para crear sus propios tipos de datos. Los tipos de datos simples o básicos de C++ son: *enteros*; *números de coma flotante* (*reales*) y *caracteres*, y se muestran en la Tabla 2.2. Existen tres tipos adicionales en C++ que se tratarán más adelante:

| | |
|---|---|
| `enum` | constante de enumeración; |
| `bool` | constante falso-verdadero; |
| *`void`* | tipo especial que indica ausencia de tipo; |

Tabla 2.1. Tipos de datos simples de C++

| Tipo básico | Tipo | Ejemplo | Tamaño en bytes | Rango. Mínimo..Máximo |
|---|---|---|---|---|
| Carácter | `char` | 'C' | 1 | 0..255 |
| | `short` | -15 | 2 | -128..127 |
| | `int` | 1024 | 2 | -32768..32767 |
| Entero | `unsigned int` | 42325 | 2 | 0..65535 |
| | `long` | 262144 | 4 | -2147483648..2147483637 |
| | `float` | 10.5 | 4 | $3.4*(10^{-38})..3.4*(10^{38})$ |
| Real | `double` | 0.00045 | 8 | $2.7*(10^{-308})..2.7*(10^{308})$ |
| | `long double` | 1e-8 | 8 | igual que `double` |



### 2.3.1. ENTEROS (INT)

Los tipos enteros se almacenan internamente en 2 bytes de memoria. La Tabla 2.2 resume los tres tipos enteros básicos, junto con el rango de valores y el tamaño en bytes usual (depende de cada compilador C++).

Tabla 2.2. Tipos de datos enteros en C++

| Tipo C++ | Rango de valores | Uso recomendado |
|---|---|---|
| `int` | -32.768 .. +32.767 | Aritmética de enteros, bucles `for`, conteo. |
| `unsigned int` | 0 .. 65.535 | Conteo, bucles `for`, índices. |
| `short int` | -32.768 .. +32.767 | Aritmética de enteros, bucles `for`, conteo. |
| `long` | -2147483648..2147483647 | Entero largo. Números grandes. |
| `unsigned long` | 0 .. +4294967295 | Entero largo positivo. Números grandes. |

> **A tener en cuenta:** el tipo `char` sólo admite los modificadores `unsigned` y `signed`; el tipo `float` y `double` no admiten el modificador `unsigned`; los enteros y enteros largos admiten el modificador `unsined` y `signed`; el tipo `bool` no admite modificadores.

### Declaración de variables

La forma más simple de una declaración de variable en C++ es poner primero el tipo de dato y, a continuación, el nombre o los nombres de las variables separados por comas. Si se desea dar un valor inicial a la variable, éste se pone a continuación del identificador de la variable precedido del signo igual:

*<tipo de dato> <nombre de variable> = <valor inicial>*

Se pueden también declarar múltiples variables en la misma línea:

*<tipo_de_dato> <nom_var1>, <nom_var2> ... <nom-varn>*



**EJEMPLO 2.8.** *Declaración e inicialización de variables enteras en C++.*

```
int valor, valor1, valor2 = 100, valor3 = 25, x, y;
short int  Numero_asociado = 345, numer, libre;
long numero_grande1, numero_grade2 = 345678;
```

C++ permite escribir constantes enteras en *octal* (base 8) o *hexadecimal* (base 16). Una constante octal es cualquier número que comienza con un 0 y contiene dígitos en el rango de `1` a `7`. Una constante hexadecimal comienza con `0x` y va seguida de los dígitos `0` a `9` o las letras `A` a `F` (o bien `a` a `f`). La Tabla 2.3 muestra ejemplos de constantes enteras representadas en sus notaciones `decimal`, `hexadecimal` y `octal`.

Tabla 2.3. Constantes enteras en tres bases diferentes

| Base 10 decimal | Base 16 hexadecimal (Hex) | Base 8 octal |
|---|---|---|
| 8 | 0x08 | 010 |
| 10 | 0x0A | 012 |
| 16 | 0x10 | 020 |
| 65536 | 0x10000 | 0200000 |
| 24 | 0x18 | 030 |



## 2.3.2. TIPOS EN COMA FLOTANTE (`float/double`)

Los tipos de datos "coma (*punto*) flotante" representan números reales que contienen una coma (un punto) decimal, tal como `2.123`, o números muy grandes, tales como `2.43 e +18 = 2,43*10`$^{18}$. La declaración de las variables de coma flotante es igual que la de variables enteras. C++ soporta tres formatos de coma flotante. El tipo `float` requiere `4` bytes de memoria, `double` requiere `8` bytes y `long` double requiere `10` bytes. La Tabla 2.4 muestra los tipos de datos en coma flotante.

Tabla 2.4. Tipos de datos en coma flotante

| Tipo C | Rango de valores | Precisión |
|---|---|---|
| `float` | $3.4 \times 10^{-38} \ldots 3.4 \times 10^{38}$ | 7 dígitos |
| `double` | $2.7 \times 10^{-308} \ldots 2.7 \times 10^{308}$ | 15 dígitos |
| `long double` | $3.4 \times 10^{-4932} \ldots 2.1 \times 10^{4932}$ | 19 dígitos |

## 2.3.3. CARACTERES (`char`)

Un *carácter* es cualquier elemento de un conjunto de caracteres predefinidos o alfabeto. La mayoría de las computadoras utilizan el conjunto de caracteres ASCII. C++ procesa datos carácter utilizando el tipo de dato `char`. Internamente, los caracteres se almacenan como números enteros en el rango `-128` a `+127` y se asocian con el código ASCII; por tanto se pueden rea-

lizar operaciones aritméticas con datos tipo `char`. El lenguaje C++ proporciona el tipo `unsigned char` para representar valores de `0` a `255` y así representar todos los caracteres ASCII. Se pueden realizar operaciones aritméticas con datos tipo `char`.

**EJEMPLO 2.9.** *Define e inicializa una variable de tipo* ***char****, a continuación se pasa a mayúscula.*

```
char car = 'c', car1 = 'D';
car = car - 32;
car1 = car1 + 32;
```

El ejemplo convierte `'c'` (código ASCII 99) a `'C'` (código ASCII 67), y `'D'` (código ASCII 68) a `'d'` (código ASCII 100).

**EJEMPLO 2.10.** *Diferencia entre char y unsigned char.*

El programa usa dos variables, `dato`, y `dato1` de tipo `char` y `unisgned char` respectivamene. Asigna a cada variable un número natural comprendido entre `115` y `129`, visualizando ambas variables como enteras y como carácter. Cuando no está activado el bit de signo, aparecen números negativos que no se corresponden con los números ASCII asociados.



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

char dato;                              // carácter
unsigned char dato1;                    //declaración de tipo carácter sin signo

int main(int argc, char *argv[])
{
  cout << "   char     unsigned char " << endl;
  cout << " ASCII  CAR  ASCII    CAR " << endl;
  for (int i = 115; i < 130; i++)
   {
     dato = i; dato1 = i;
     cout << (int) dato << "      " << dato << "      ";
     cout << (int) dato1 << "      " << dato1<< endl;
   }
  system("PAUSE");
  return EXIT_SUCCESS;
}
```



Resultado de ejecución:

```
  char      unsigned char
ASCII  CAR  ASCII    CAR
15      s     115     s
16      t     116     t
17      u     117     u
18      v     118     v
19      w     119     w
20      x     120     x
21      y     121     y
22      z     122     z
23      {     123     {
24      |     124     |
25      }     125     }
26      ~     126     ~
27      ⌂     127     ⌂
128     Ç      128     Ç
127     ü      129     ü
Presione una tecla para continuar .
```

### 2.3.4. EL TIPO DE DATO `bool`

La mayoría de los compiladores de C++ incorporan el tipo de dato `bool` cuyos valores posibles son: "*verdadero*" (`true`) y falso" (`false`). El tipo `bool` proporciona la capacidad de declarar variables lógicas, que pueden almacenar los valores verdadero y falso. Si en el compilador de C++ no está disponible el tipo `bool`, deberá utilizar el tipo de dato `int` para representar el tipo de dato `bool`. C++ utiliza el valor entero `0` para representar falso y cualquier valor entero distinto de cero (normalmente `1`) para representar verdadero. De esta forma, se pueden utilizar enteros para escribir expresiones lógicas de igual forma que se utiliza el tipo `bool`.

**EJEMPLO 2.11.** *Uso del tipo bool.*

```
#include <iostream.h>
void main()
{
   bool b1,  b2 = false,  b3 = true;
   int i1 = b3, i4 = 10;

   cout << " b2:" << b2 << endl;    // muestra el valor de b2 que es 0
   ...
}
```



## 2.4. Constantes

Una constante es un objeto cuyo valor no puede cambiar a lo largo de la ejecución de un programa. En C++ existen cuatro tipos de constantes: *constantes literales*, *constantes definidas*, *constantes enumeradas*, *constantes declaradas*.

### 2.4.1. CONSTANTES LITERALES

Las constantes literales o constantes, en general, se clasifican en cuatro grupos, cada uno de los cuales puede ser de cualquiera de los tipos siguientes: enteras, reales, de caracteres, de cadena, enumeradas, definidas y declaradas.

**Constantes enteras.** Son una sucesión de dígitos precedidos o no por el signo + o – dentro de un rango determinado.

**EJEMPLO 2.12.** *Constantes numéricas.*

| | |
|---|---|
| `234, -456` | constantes enteras. |
| `12334L, 43567L` | constantes enteras largas; tienen el sufijo `L` |
| `0777, 0123` | constantes en octal; comienza por `0` y dígitos entre `0` y `7` |
| `0XFF3A, 0x2AB345` | constantes hexadecimal; comienzan por `0X` |
| `3456UL` | constante sin signo y larga; sufijo `U` sin signo |



**Constantes reales.** Son una sucesión de dígitos con un punto delante, al final o en medio y seguidos opcionalmente de un exponente: Por ejemplo, `82.347`, `.63`, `83.`, `47e-4`, `.25E7` y `62.e+4`.

**Constantes carácter.** Una constante carácter (`char`) es un carácter del código ASCII encerrado entre apóstrofes. Por ejemplo, `'A'`, `'a'`, `'b'`, `'c'`. Además de los caracteres ASCII estándar, una constante carácter soporta caracteres especiales que no se pueden representar utilizando su teclado, como por ejemplo los códigos ASCII altos y las secuencias de escape.

Tabla 2.5. Caracteres secuencias (códigos) de escape

| Código de Escape | Significado | Códigos ASCII | | | |
|---|---|---|---|---|---|
| | | Dec | | Hex | |
| `'\n'` | nueva línea | 13 | 10 | 0D | 0A |
| `'\r'` | retorno de carro | 13 | | 0D | |
| `'\t'` | tabulación | 9 | | 09 | |
| `'\v'` | tabulación vertical | 11 | | 0B | |

Tabla 2.5. Caracteres secuencias (códigos) de escape *(continuación)*

| Código de Escape | Significado | Códigos ASCII | |
|---|---|---|---|
| | | Dec | Hex |
| '\a' | alerta (pitido sonoro) | 7 | 07 |
| '\b' | retroceso de espacio | 8 | 08 |
| '\f' | avance de página | 12 | 0C |
| '\\' | barra inclinada inversa | 92 | 5C |
| '\'' | comilla simple | 39 | 27 |
| '\"' | doble comilla | 34 | 22 |
| '\?' | signo de interrogación | 34 | 22 |
| '\000' | número octal | todos | todos |
| '\xhh' | número hexadecimal | todos | todos |



**Constantes cadena.** Una *constante cadena* (también llamada *literal cadena* o simplemente *cadena*) es una secuencia de caracteres encerrados entre dobles comillas. Por ejemplo, `"123"`, `"14 de Julio de 2005"`, `"esto es un ejemplo de cadena"`. En memoria, las cadenas se representan por una serie de caracteres ASCII más un `0` o nulo `'\0'` que es definido en C++ mediante la constante `NULL` en diversos archivos de cabecera (normalmente `STDEF.H`, `STDIO.H`, `STDLIB.H` y `STRING.H`).

**Constantes definidas (simbólicas).** Las constantes pueden recibir nombres simbólicos mediante la directi va `#define`. Estas constantes son sustituidas por su v alor por el preprocesador (antes de comenzar la compilación del programa).

**EJEMPLO 2.13.** *Se ponen nombres simbólicos a constantes de interés.*

```
#define NUEVALINEA '\n'
#define e 2.81
#define pi 3.1415929          // valor de la constante Pi
#define cabecera "esto es un ejemplo de cabecera de programa"
```

C++ sustituye `'\n'`, `2.81`, `3.1415929`, `"esto es un ejemplo de cabecera de programa"`, cuando se encuentra las constantes simbólicas `NUEVALINEA`, `e`, `pi`, `cabecera`.

**Constantes enumeradas.** Las constantes enumeradas permiten crear listas de elementos afines. Cuando se procesa esta sentencia el compilador *enumera* los identificadores comenzando por `0`. Después de declarar un tipo de dato enumerado, se pueden crear variables de ese tipo, como con cualquier otro tipo de datos. En el ejemplo 2.13 `Miercoles` toma el valor `2`.



**EJEMPLO 2.14.** *Declaración de tipos enumerados e inicialización de variables de tipo enumerado.*

```
enum laborable  { Lunes, Martes, Miercoles, Jueves, Viernes}
enum festivo {Sabado, Domingo};
laborable diaL =lunes;
festivo  diaF = Domingo;
```

**Constantes declaradas; `const` y `volatile`.** El cualificador `const` permite dar nombres simbólicos a constantes. Su valor no puede ser modificado por el programa. Su formato es:

```
const tipo nombre = valor;
```

La palabra reserv ada `volatile`, modifica el tipo de acceso de una v ariable, permitiendo cambiar el v alor de la v ariable por medios no explícitamente especificados por el programa. Por ejemplo la dirección de una v ariable global que apunta a un puerto externo. El valor de una variable `volatile` puede ser modificado no sólo por el propio programa, sino también por el *hardware* o por el *software* del sistema. Su formato de declaración es:

```
volatile tipo nombre = valor;
```

Si se usa `const` y `volatile` juntos, se asegura que una variable no puede cambiar por programa, pero sí por medios externos al progama C++.

**Ejemplo:** `const volatile unsigned char *puerto=0x30;`.

## 2.5. Variables

En C++ una *variable* es una posición de memoria a que se le asocia un nombre (identificador) en el que se almacena un valor del tipo de dato del que se ha definido. El valor de una variable puede cambiar a lo largo de la ejecución del programa, siendo manipulada por los operadores aplicables al tipo del que ha sido definida la variable.

**Delaración.** Una *declaración* de una variable es una sentencia que proporciona información de la variable al compilador C++. Es preciso *declarar* las variables antes de utilizarlas. La sintaxis de declaración es:



```
tipo lista de variables;
```

Siendo *tipo* el nombre de un tipo de dato conocido por C++ y *lista de variables* una sucesión de nombres separadas por comas, y cada nombre de la lista un identificador de C++.

**Inicialización.** Las variables, pueden ser inicializadas al tiempo que se declaran. El formato general de una declaración de inicialización es:

```
tipo lista de inicialización;
```

Siendo *lista de inicialización* una sucesión *nombre_variable = expresión*. Además *expresión* es cualquier expresión válida cuyo valor es del mismo tipo que *tipo*.
Hay que tener en cuenta que los dos formatos de declaración pueden combinarse entre sí.

**EJEMPLO 2.15.** *Declaración de variables e inicialización de alguna de ellas.*

```
int x, z, t = 4;
float xx = 2.0, yy = 8, zz;
char Si = 'S', No = 'N', ch;
```



Las variables `x`, `z` y `t` son enteras. La variable `t` se inicializa a 4.
Las variables `xx`, `yy`, y `zz` son reales. Se inicializan `xx` y `zz` a `2.0`, y `8` respectivamente.
Las variables `Si`, `No` y `ch` son caracteres. Se inicializan las dos primeras a los caracteres `'S'` y `'N'` respectivamente.

## 2.6. Duración de una variable

Dependiendo del lugar de definición, las variables de C++, pueden ser utilizadas en la totalidad del programa, dentro de una función o sólo dentro de un bloque de una función. La zona de un programa en la que una variable está activa se denomina, normalmente, *ámbito* o *alcance* ("*scope*").

### 2.6.1. VARIABLES LOCALES

Las variables locales son aquellas que se definen dentro de una función. Las reglas por las que se rigen las variables locales son:

1. Las variables locales no pueden modificarse fuera de la función.
2. Los nombres de las variables locales no son conocidas fuera de donde se declaran.
3. Las variables locales no existen en memoria hasta que se ejecuta la función.