

# INTRODUCCIÓN AL LENGUAJE C++

## 1.- ESPECIFICACIÓN DE PROGRAMAS

### ESTRUCTURA Y ORGANIZACIÓN DE UN PROGRAMA

- Los programas en C++ pueden dividirse en varios ficheros aunque por ahora vamos a suponer que cada programa está escrito en un único fichero (por ejemplo, Pitágoras.cpp).
- Se pueden incluir comentarios en lenguaje natural.

```
/* De esta manera puedo incluir comentarios en varias líneas
de mi código fuente*/
// comentario 1 sola línea
```

El texto de un comentario no es procesado por el compilador

- Al principio del fichero se indica que vamos a usar una serie de recursos definidos en un fichero externo, biblioteca o librería.

```
#include <iostream>
#include <conio.h>
using namespace std;
```

- A continuación aparece `int main(){` que indica que comienza el programa principal. Este se extiende desde la llave abierta `{`, hasta encontrar la correspondiente llave cerrada.
- Dentro del programa principal van las **sentencias** que componen el programa. Una sentencia es una parte del código fuente que el compilador puede traducir en una instrucción en código binario
- Las sentencias van obligatoriamente separadas por punto y coma (;).
- El compilador va ejecutando las sentencias secuencialmente de arriba abajo. Aunque en el siguiente tema se verá como realizar saltos, es decir, interrumpir la estructura secuencial.
- Cuando llega a la llave cerrada `}` correspondiente a `main()`, si no ha habido problemas, el programa terminará de ejecutarse y el sistema operativo libera los recursos asignados a dicho programa.

### TIPOS DE SENTENCIAS

- Sentencias de declaración de datos. Los datos son como variables matemáticas. Cada dato que usemos en un programa debe declararse al principio (después de `main`), asociándole un tipo de dato concreto.

```
double lado1;
double lado2;
double hip;
```

Declara tres datos (variables) de tipo real que el programador puede usar en el programa

- Sentencias de cálculo, a través de la **asignación** =

```
lado1=7;
lado2=5;
```

```
hip = sqrt(lado1*lado1 + lado2*lado2);
```

Asigna a la variable hip el resultado de evaluar lo que aparece en la derecha de la asignación.

NOTA: No confundir = con la Igualdad en Matemáticas

- Sentencias para mostrar información en el periférico de salida establecido por defecto. En la asignatura este periférico será la pantalla, pero podría ser un fichero en disco o dispositivo externo (Impresora, Ploter, Puerto FireWire, etc.).

Se construye usando **cout**, que es un recurso externo incluido en la biblioteca **iostream**.

- Lo que se vaya a imprimir va separado de **cout** por **<<**.
- Lo que haya dentro de un par de comillas dobles se muestra tal cual, excepto los caracteres precedidos de **\**. Por ejemplo, **\n** hace que el cursor salte al principio de la línea siguiente.

```
cout<< "Bienvenido. Salto a la siguiente línea ->\n";  
cout<<"\nEmpiezo en una nueva línea";
```

- Los números se escriben tal cual.

```
cout<< 3.14159;
```

- Si ponemos una variable, se imprime su contenido

```
cout<< "Introduzca la longitud del primer cateto: ";  
.....  
hip = sqrt(lado1*lado1 + lado2*lado2);  
cout<< "la hipotenusa vale: ";  
cout<< hip
```

**También puede ponerse esto en una única sentencia**

```
cout<< "la hipotenusa vale: "<<hip;
```

- Aunque no es recomendable también pueden ponerse expresiones como:

```
cout<< "la hipotenusa vale: "<< sqrt(lado1*lado1 + lado2*lado2);
```

- Es importante dar un formato adecuado a la salida de datos en pantalla. El usuario del programa debe entender claramente el significado de todas sus salidas.

```
//MAL:  
cout<<totalventas<<numeroventas;  
//BIEN:  
cout<<"\nSuma total de ventas= "<<totalventas;  
cout<<"\nNumero total de ventas= "<<numeroventas;
```

- Sentencias para leer datos del dispositivo de entrada establecido por defecto. Por ahora, será el teclado. Se construyen usando **cin**, que es un recurso externo incluido en la biblioteca **iostream**.

Su sintaxis más elemental es la siguiente

```
cin>> variable;
```

Por ejemplo, `cin>> lado1`, espera a que el usuario introduzca un valor real (double) desde el teclado (dispositivo de entrada) y, cuando se pulsa la tecla intro, lo almacena en la variable `lado1`. Conforme se va escribiendo el valor, es se te muestra en pantalla, incluyendo el salto de línea.

Las entradas de datos deben etiquetarse adecuadamente:

```
//MAL:  
cin>>lado1;  
cin>>lado2;
```

```
//BIEN:  
cout<<"Introduzca la longitud del primer cateto: ";  
cin>> lado1;  
cout<< "Introduzca la longitud del segundo cateto: ";  
cin>>lado2;
```

- Estructura básica de un programa (los corchetes delimitan secciones opcionales):

```
/*Breve descripción en lenguaje natural de lo que hace el  
programa */  
[Inclusión de recursos externos (Bibliotecas o librerías)]  
  
using namespace std;  
  
int main(){  
  
    [Declaración de datos]  
  
    [Sentencias del programa separadas por ;]  
  
}
```

## 2.- DATOS Y TIPOS DE DATOS

Al trabajar con un lenguaje de alto nivel (c++), no haremos referencia a la secuencia de 0 y 1 que codifican un valor concreto, sino a lo que representa para nosotros.

Un dato es un conjunto de celdas o posiciones de memoria que tiene asociado un nombre (identificador) y un contenido (valor).

El compilador reconoce distintas categorías de datos (numéricas, texto, etc.). Necesitamos almacenar muchos tipos de información (enteros, reales, caracteres, cadenas de caracteres, etc.). Para cada tipo el lenguaje ofrece un tipo de dato.

Por ejemplo en c++:

### TIPOS DE DATOS SIMPLES

- Caracteres: `char` (también es un entero)
- Enteros: `int`
- Números en coma flotante: `float`, `double`,
- Booleanos: `bool`

## TIPOS DE DATOS COMPUESTOS

- Vectores y matrices
- Cadena de caracteres
- Registros
- Clases

Por ahora sólo usaremos los tipos de datos simples pero más adelante veremos algunos compuestos.

## DECLARACIÓN DE VARIABLES

Cada variable que se use en un programa debe declararse al principio (después de main), asociándole un tipo de dato concreto y un identificador.

- Al declarar una variable, el compilador reserva una zona de memoria para trabajar con ella. Ninguna otra variable podrá utilizar dicha zona.
- Cada variable debe estar asociada a un único tipo de dato (el tipo no puede cambiarse durante la ejecución). El valor que se le puede asignar a una variable depende del tipo de dato con el que es declarado.

Formas de declarar variables:

- Un único identificador por declaración

**<tipo> <identificador1>**

Ejemplos:

```
double lado1
double lado2
```

- Varios identificadores por declaración

**<tipo> <identificadores separados por coma>**

Ejemplo:

```
double lado1, lado2, hip;
```

- Dando un valor inicial por declaración

**<tipo> <identificador> = <valor\_inicial>**

**<tipo> <identificador>=<valor\_inicial>, <identificador>=<valor\_inicial>,...**

Ejemplo:

```
double dato=4.5;
```

Es equivalente a:

```
double dato;
dato= 4.5;
```

Cuando se declara una variable y no se inicializa, ésta no tiene ningún valor asignado por defecto. Puesto que desconocemos su valor, lo podremos considerar como basura, y lo representaremos gráficamente por ?.

### **Normas para la elección del identificador**

Cada dato necesita un identificador único. Un identificador de un dato esta formado por caracteres alfanuméricos con las siguientes restricciones:

- Debe empezar por letra o subrayado (\_).
- No pueden contener espacios en blanco ni ciertos caracteres especiales como letras acentuadas, la letra ñ, las barras / o \, etc
- El compilador determina la máxima longitud que pueden tener.
- Sensibles a mayúsculas y minúsculas
- No se podrá dar a un dato el nombre de una palabra reservada.

### **3.- OPERADORES Y EXPRESIONES**

#### TERMINOLOGÍA EN MATEMÁTICAS

Notaciones utilizadas en los operadores matemáticos:

- **Prefija.** El operador va antes de los argumentos. Estos suelen cerrarse entre paréntesis.  
Seno(3), tangente(x), media(valor1, valor2);
- **Infija.** El operador va entre los argumentos  
3+5, x/y

Según el número de argumentos, diremos que un operador es :

- **Unario.** Solo tiene un argumento.  
Seno(3), tangente(x)
- **Binario.** Tiene dos argumentos.  
Media(valor1,valor2), 3+5. x/y
- **N-ario.** Tiene mas de dos argumentos.

#### OPERADORES EN UN LENGUAJE DE PROGRAMACIÓN

Cuando hablamos de matemáticas pensamos en valores numéricos enteros o reales, pero en general los operadores matemáticos trabajan con cualquier tipo de dato (lógico, matrices, complejos, etc) y son mas generales ya que, por ejemplo podrían modificar el dato sobre el que se aplican.

Los lenguajes de programación proporcionan operadores que permiten manipular los datos.

#### **Tipos de operadores:**

- Los definidos en el núcleo del compilador.  
No hay que incluir ninguna biblioteca. Suelen ser infijos  
Ejemplos: + (Suma) – (Resta) \*(producto) etc..
- Los definidos en bibliotecas externas.  
Suelen ser prefijos y si hay varios argumentos suelen separarse por comas.

`sqrt(4) sin(6) pow(3,6)`  
Las funciones pertenecen a la biblioteca `cmath` o `math.h`.

## EXPRESIONES:

Una expresión es una combinación de datos y operadores sintácticamente correcta, que el compilador evalúa y devuelve un valor.

```
3
3+5
lado1
lado1*lado1
lado1*lado1+lado2*lado2
sqrt(lado1*lado1 + lado2*lado2)
```

Así pues, los operadores y funciones actúan, en general, sobre expresiones. Las expresiones pueden aparecer a la derecha de una asignación, pero no a la izquierda.

**3+5 = lado //Error de compilación**

NOTA. El operador = No representa una igualdad matemática.

¿Qué haría lo siguiente?

```
double dato = 4;
dato = dato + 3;
```

Una expresión NO es una sentencia de un programa:

- Expresión: `sqrt(lado1*lado1 + lado2*lado2)`
- Sentencia: `hip = sqrt(lado1*lado1 + lado2*lado2)`

Cuando el compilador evalúa una expresión, devuelve un valor de un tipo de dato (entero, real, carácter, etc.) Diremos que la expresión es de dicho tipo de dato. Por ejemplo:

3+5 es una expresión entera  
3.5 + 6.7 es una expresión de reales

```
int main(){
    double dato_real;
    int dato_entero;
    dato_real=3.5 + 6.7;
    dato_entero= 3 + 5;
    .....
}
```

Nota: Cuando se usa una expresión dentro de un `cout`, el compilador detecta el tipo de dato y la imprime de forma adecuada.

```
cout<< "\nResultado = "<< 3+5;
cout<< "\nResultado = "<<3.5 + 6.7;
```

#### 4.- TIPOS DE DATOS COMUNES EN C++

**Enteros:** Son números que tienen un rango limitado, el computador no puede representar números excesivamente altos. Los enteros matemáticamente son los números que representan el conjunto Z. En c++ se declaran con la palabra reservada `int`

```
int <identificador> // declara un identificador como una variable entera
```

**Reales:** Los números reales también tienen un rango limitado por la capacidad del computador. Estos son los números que componen el conjunto R. Se componen de una parte entera y una parte real o decimal. Se declaran con la palabra reservada `double` o `float`.

```
double <identificador> // declara un identificador como una variable real
```

Los **operadores** comunes para los números reales y enteros son los siguientes: +, -, \* y /

Hay operadores exclusivos para los números enteros que son:

% → representa el resto de la división de dos enteros.

++ → Aumenta en uno el valor de una variable entera.

-- → Disminuye en uno el valor de una variable entera.

Estos operadores tienen unas reglas de preferencia:

() ,

- (operador unario de cambio de signo)

\* / %

+ -

Cualquier operador de una fila superior tiene más prioridad que cualquier de la fila inferior. Los operadores de la misma fila tienen la misma prioridad. Se evalúan de izquierda a derecha según aparecen en una expresión.

**Carácter:** Es un conjunto finito y ordenado de caracteres: letras minúsculas, mayúsculas, dígitos del 0 al 9 y otros caracteres especiales. Se representan con la palabra reservada `char`. Cada carácter tiene un código numerado de 0 a 255. Buscar código ASCII en la web.

En c++, el tipo `char` es realmente un tipo entero pequeño. Cuando hacemos

```
char c; //declara una variable de tipo caracter
c='A';
```

estamos asignando a la variable `c` el número de orden en la tabla ASCII del carácter 'A', es decir el 65.

```
char c;
c='A'; // c contiene el valor 65
```

De hecho también podríamos poner:

```
char c;
```

`c= 65; // c contiene el valor 65`

En definitiva el tipo char es un tipo de dato entero pequeño ya que esta pensado para almacenar los números de órdenes de la tabla ASCII.

**Booleano:** Es un tipo de dato muy común en los lenguajes de programación que se utiliza para representar los valores verdadero y falso que suelen estar asociados a una condición. En c++ se representa con la palabra reservada **bool**.

Un dato lógico solo puede tomar dos valores: true (verdadero) y false (falso). Una expresión lógica es una expresión cuyo resultado es un tipo de dato lógico.

Los operadores en este tipo de dato son los clásicos de la lógica (NO, Y, O) que en c++ son los operadores `!`, `&&`, `||`, respectivamente.

Otros operadores para este tipo de dato son los relacionales, que son los habituales para la comparación de expresiones numéricas. Pueden aplicarse a operadores tanto enteros, reales, como de caracteres y tienen el mismo sentido que en Matemáticas. El resultado es de tipo bool.

`==` (igual), `!=` (distinto), `<`, `>`, `<=` (menor o igual) y `>=` (mayor o igual)

**Algunos ejemplos:**

- La expresión `(4<5)` devuelve valor true.
- La expresión `(4>5)` devuelve el valor false.

## 5.- ESTRUCTURAS DE CONTROL

**Estructura secuencial:** Las instrucciones se van ejecutando sucesivamente, siguiendo el orden de aparición de estas. No hay saltos.

**Estructura condicional:** Una condición es una expresión lógica.

Una ejecución condicional consiste en la ejecución de una (o más) sentencias dependiendo de la evaluación de una condición.

Existen tres tipos: Simple, doble y múltiple.

**Simple:**

```
if (condición){  
    <sentencias>  
}
```

Donde **condición** es una expresión lógica que puede ser verdadera o falsa y `<sentencias>` serán las instrucciones que se ejecutarán en el caso de que la condición sea verdadera.

**Doble:**

Funciona igual que la simple  
Pero cuando la condición es  
Falsa se ejecutan las sentencias  
Que hay dentro del else.

```
if (condicion){  
    <sentencias>;  
}  
else {  
    <sentencias>;  
}
```

### **Anidamiento de estructuras condicionales:**

Dentro de un bloque if (else), puede incluirse otra estructura condicional, anidándose tanto como permita el compilador. Hay que tener cuidado con las llaves, tabular bien los programas para que, a la hora de que ocurran errores poder saber bien dónde está el fallo. Ejemplo:

```
if (condición1){
    Sentencias if 1;
    if(condición2){
        Sentencias if 2;
    }
    else{
        Sentencias else 2;
    }
}
else {
    Sentencias else1;
}
```

Ejemplos sentencias condicionales:

```
//comprueba si un numero es mayor o menor que 0
#include <iostream>
#include <conio.h>
using namespace std;
int main(){
    double valor;
    cout<<"Introduce un numero: ";
    cin>> valor;
    if(valor==0){ //Comprobamos si es igual a cero
        cout<<" Es igual a cero";
    }
    if (valor>0){ //Comprobamos si es mayor que cero
        cout<<"Mayor que cero";
    }
    else{ //si no se cumple ninguna de las anteriores es que es menor
        cout<<"Menor que cero";
    }
    getch();
    return 0;
}
```

El programa anterior tiene una sentencia condicional simple y una sentencia condicional doble.