

# **Tipos de Datos y Operadores**

# Tipos de Datos en C++

Un programa trabaja con datos con unas determinadas características. No es lo mismo procesar números naturales, números reales o cadenas de caracteres.

En cada caso tratamos con datos de tipo diferente.

La elección de un determinado tipo u otro para una determinada entidad del programa **proporciona información** acerca de:

- ☐ El **rango de posibles valores** que puede tomar.
- ☐ El **conjunto de operaciones y manipulaciones** aplicables.
- ☐ El **espacio de almacenamiento** necesario para almacenar dichos valores.
- ☐ La **interpretación del valor almacenado**.

# Tipos de Datos en C++

Los datos se clasifican en:

- ❑ **Simple:** valores indivisibles, es decir, no se dispone de operadores para acceder a parte de ellos: [int](#), [float](#), [double](#), [char](#), [bool](#)
- ❑ **Compuestos:** están formados como un agregado o composición de otros tipos, ya sean simples o compuestos: [arreglos](#), [cadenas](#), [estructuras](#)

## Tipos simples predefinidos

Tipo	Bytes	Bits	Min. Valor	Max. Valor
bool	1	8	false	true
char	1	8	-128	127
short	2	16	-32768	32767
int	4	32	-2147483648	2147483647
long	4	32	-2147483648	2147483647
long long	8	64	-9223372036854775808	9223372036854775807
unsigned char	1	8	0	255
unsigned short	2	16	0	65535
unsigned	4	32	0	4294967295
unsigned long	4	32	0	4294967295
unsigned long long	8	64	0	18446744073709551615
float	4	32	1.17549435e-38	3.40282347e+38
double	8	64	2.2250738585072014e-308	1.7976931348623157e+308
long double	12	96	3.36210314311209350626e-4932	1.18973149535723176502e+4932

# Variables

- Son posiciones de la memoria donde se almacena un valor de un cierto tipo.
- En C++ toda variable debe **declararse** antes de ser usada, dándole un nombre (identificador) y asignándole un tipo.

## Declaración de Variables:

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

El valor inicial o expresión es opcional, pero debe ser válido (del tipo correcto).

- ✓ La declaración reserva espacio de memoria para almacenamiento y debe situarse al principio de un bloque.
- ✓ Si la declaración contiene el signo “=” se le proporciona un valor inicial a la variable → **Inicialización**.
- Es posible acceder a cualquier variable declarada, **pero si no se ha INICIALIZADO**, referirá al valor (“**basura**”) que tiene almacenado el espacio de memoria.

Ejemplos de declaraciones de variables:

- int suma = 0;
- char valor;
- int dni, edad, contador;

# Constantes

C++ tiene 3 tipos de constantes:

- **Constantes literales:** se escriben directamente en el programa. Pueden ser:
  - **Constantes booleanas:** false, true
  - **Constantes enteras:** 123, 1860L (long), 0773 (octal), 0AFF3 (hexadecimal)
  - **Constantes de coma flotante:** 23.4, -3.1932, -1.76E12
  - **Constantes de caracteres:** se encierran entre comillas simples: 'a', 'Z', '3', ':'.
  - **Constantes de cadena:** Las cadenas se encierran entre comillas dobles y se representan por una secuencia de caracteres, más un carácter nulo al final insertado automáticamente por el compilador: "es cadena", "z".

# Constantes

- **Constantes definidas:** son nombres (identificadores) asociados a valores literales constantes.

Se definen mediante: **#define <nombre> <valor>**

*Ejemplo:* #define PI 3.141592

No terminan  
con “,”

- **Constantes declaradas:** son como una variable pero su valor no puede ser modificado.

Se definen mediante:

**const <tipo-dato> <nombre-constante> = <valor-constante>;**

*Ejemplos:* const int Meses = 12;

const int Semana = 7;

const float PI=3.141592;

# Expresiones

- Un programa se basa en la realización de una serie de cálculos con los que se producen los resultados deseados.
- Dichos resultados se almacenan en variables y a su vez son utilizados para nuevos cálculos, hasta obtener el resultado final.
- Los cálculos se producen mediante la **evaluación de expresiones** en las que se mezclan **operadores** con **operandos** (constantes literales, constantes simbólicas o variables).
- En caso de que en una misma expresión se utilice más de un operador habrá que conocer la **precedencia de cada operador** (aplicando el de mayor precedencia), y en caso de que haya varios operadores de igual precedencia, habrá que conocer su **asociatividad** (si se aplican de izquierda a derecha o de derecha a izquierda).

# Reglas de Precedencia

## (Ordenados de mayor a menor precedencia)

Mayor  
precedencia



Menor  
precedencia

Operador	Tipo de Operador	Asociatividad
[] -> .	Binarios	Izq. a Dch.
! ~ - *	Unarios	Dch. a Izq.
* / %	Binarios	Izq. a Dch.
+ -	Binarios	Izq. a Dch.
<< >>	Binarios	Izq. a Dch.
< <= > >=	Binarios	Izq. a Dch.
== !=	Binarios	Izq. a Dch.
&	Binario	Izq. a Dch.
^	Binario	Izq. a Dch.
	Binario	Izq. a Dch.
&&	Binario	Izq. a Dch.
	Binario	Izq. a Dch.
? :	Ternario	Dch. a Izq.



# Significado y tipos de Operadores

Operador	Significado
[ ]	Selección de elemento en una tabla
.	Selección de miembro de una tupla
( )	Paréntesis
-	Cambio de Signo
!	Negación
*	Multiplicación
/	División entera
/	División real
%	Módulo
+	Suma
-	Resta
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
==	Igual
!=	Diferente
&	Conjunción lógica bit a bit
	Disyunción lógica bit a bit
&&	Conjunción lógica
	Disyunción lógica

- **Aritméticos**
- **Relacionales (comparaciones)**
- **Lógicos**
- **Condicionales**
- **De manipulación de bits**

# RESUMEN de Operadores C++

Operador	Descripción	Asociatividad
() [] -> .	Llamada a función Elemento de un arreglo Referencia al apuntador de miembro en una estructura Referencia a un miembro de una estructura	Izquierda a derecha
++ -- - ! ~ (type) sizeof & *	Incremento Decremento Menos unario Negación lógica Complemento a uno Conversión de tipo (molde) Tamaño del almacenamiento La dirección de Indirección	Derecha a izquierda
* / %	Multiplicación División Módulo (residuo)	Aritméticos
+ -	Adición Sustracción	
<< >>	Desplazamiento a la izquierda Desplazamiento a la derecha	Izquierda a derecha
< <= > >=	Menor que Menor que o igual a Mayor que Mayor que o igual a	Relacionales
== !=	Igual a Diferente a	
&	AND bit por bit	Izquierda a derecha
*	OR exclusivo bit por bit	Izquierda a derecha
	OR inclusivo bit por bit	Izquierda a derecha
&&	AND lógico	Lógicos
	OR lógico	
?:	Expresión condicional	Derecha a izquierda
= += -= *= /= %= &= ^=  = <<= >>=	Asignación Asignación Asignación Asignación Asignación	Derecha a izquierda
,	Coma	Izquierda a derecha

## Salida de Datos

- El estándar de salida (stdout) es la pantalla.
- En el archivo iostream están definidas macros, constantes, variables y funciones para la salida.
- La función **cout** deriva a pantalla los valores a mostrar

*cout << cadena;*

*cout << dato;*

*cout << endl;*

Los datos pueden ser expresiones, variables, constantes.

### Salida de datos formateada

- Para dar formato a la salida:

**#include <iomanip>**

- **setprecision (int p):** Fija la precisión de salida
- **setw (int w):** Fija el ancho de salida
- **setfill (int f) :** Fija el carácter de relleno
- **fixed:** Permite alinear los puntos decimales

# Salida de datos formateada

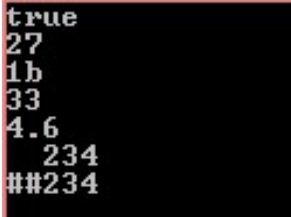
- En ocasiones puede interesar **controlar la forma en la que se muestran** los datos, especificando un determinado formato.

**Ejemplo 1**

```
/* Salida de datos con diferentes formatos */
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
    bool x = true;
    cout << boolalpha << x << endl; // escribe los booleanos como 'false' o 'true'

    cout << dec << 27 << endl; // escribe 27 (decimal)
    cout << hex << 27 << endl; // escribe 1b (hexadecimal)
    cout << oct << 27 << endl; // escribe 33 (octal)

    cout << setprecision(2) << 4.567 << endl; // escribe 4.6
    cout << dec << setw(5) << 234 << endl; // escribe " 234"
    cout << setfill('#') << setw(5) << 234; // escribe "##234"
    return 0;
}
```



The terminal output of the program is as follows:

```
true
27
1b
33
4.6
 234
##234
```

# Entrada de Datos

- Los datos de entrada pueden tener diversas fuentes: teclado, archivos,...
- El estándar de entrada (stdin) es el teclado.
- En el archivo iostream están definidas funciones para la entrada.
- La función **cin** permite la entrada de datos:

*cin >> var1 >> var2;*

Equivalente a

*cin >> var1;*



*cin >> var2;*

# Expresiones

**Asignación:** Variable = expresión

- **Expresión**: es un elemento de un programa que retorna un valor.
  - ✓ Puede ser una constante, una variable o una fórmula que se ha evaluado, cuyo resultado se puede asignar a la variable de la izquierda.

Ejemplos:  $a = 3 * z + \text{sum}(x)$ ;     $a = 3$ ;     $a = b$ ;

- La asignación destruye el valor anterior. Así, si se ejecuta:  $a=b$ ;  $b=a$ ; luego de estas dos acciones no se ha intercambiado el valor de a y b.
- La asignación puede tener notaciones abreviadas:
  - $a++$ ;     Incrementa en 1 el valor de a  $\rightarrow a = a+1$ ;
  - $a = b = c = 56$ ;
  - $a *= 3$ ;     Almacena en a el valor de  $a*3 \rightarrow a = a*3$ ;

# Notación abreviada de asignación

En asignaciones donde se suma, resta, multiplica o divide, se puede usar una **notación abreviada**:

Operador **prefijo** de incremento/decremento

Sentencia	Equivalencia
<code>++variable;</code>	<code>variable = variable + 1;</code>
<code>--variable;</code>	<code>variable = variable - 1;</code>
<code>variable++;</code>	<code>variable = variable + 1;</code>
<code>variable--;</code>	<code>variable = variable - 1;</code>
<code>variable += expresion;</code>	<code>variable = variable + (expresion);</code>
<code>variable -= expresion;</code>	<code>variable = variable - (expresion);</code>
<code>variable *= expresion;</code>	<code>variable = variable * (expresion);</code>
<code>variable /= expresion;</code>	<code>variable = variable / (expresion);</code>

Operador **postfijo** de incremento/decremento

# Operadores pre y post fijo

Cuando el operador ++ aparece antes de una variable se llama **operador de prefijo para incremento**; cuando aparece después de una variable se llama **operador de postfijo para incremento**.

- La distinción entre un operador de prefijo y uno de postfijo es importante cuando la variable que es incrementada **se usa en una expresión de asignación**.

- Ejemplo operador de prefijo: **valor = ++n;**

El valor de n es incrementado en 1 y luego el nuevo valor de n es asignado a la variable valor. La expresión **valor = ++n;** es equivalente a:

```
n=n+1;    //primero incrementa n
valor=n;  //y luego asigna el valor de n a
```

- La expresión **valor = n++;** usa un operador de postfijo para incremento, invirtiendo este procedimiento. Asigna primero el valor actual de n a valor y luego incrementa el valor de n en 1.

**valor = n++;** es equivalente a:

```
valor=n;  //asigna el valor de n a
n=n+1;    //y luego incrementa n
```

C++ también proporciona un operador de decremento -- que funciona de la misma manera, aunque decrementando en 1 la variable.



# Operadores Relacionales

- Son: **==**, **!=**, **<**, **>**, **>=**, **<=**

## Forma de utilización:

**Exp1 operador-relacional Exp2**

- Las expresiones y el operador deben ser compatibles.
- Sirven para comprobar una condición.
- Devuelven 0 (condición falsa) o 1 (condición verdadera).
- Los caracteres se comparan utilizando el código ASCII.
- La comparación alfabética de cadenas debe hacerse con la función *strcmp (cad1, cad2)*.
- Ejemplos de expresiones relacionales válidas:
  - `b * b > 5 * c;`
  - `inicial != 'A';`
  - `medida == 12.6;`
  - `'a' < 'B' ;`

# Operadores Lógicos

- También llamados operadores booleanos, son:

**! (not)      && (and)      || (or)**

- Las evaluaciones se corresponden con el **álgebra de Boole** y con sus prioridades.
- Se trabaja con **evaluación en cortocircuito**: se evalúa el operando de la izquierda y si éste determina en forma unívoca la expresión, el de la derecha no se evalúa.

Ej: `x=0; if ((x>0) && (log (x) <8)) ....`

Como el operando de la izquierda es falso, la expresión completa es falsa → no se evalúa el operando de la derecha.

✓ Utilidades:

Ahorrar tiempo en la evaluación de condiciones complejas

Evitar errores: `(n!=0) && ( x < 1.0/ n)`

Si n es 0 entonces la expresión `n!= 0` es falsa → la 2da expresión no se evalúa.

- Los resultados de expresiones booleanas se pueden almacenar en variables mediante asignación:

`int edad, MayorEdad;`

`MayorEdad = (edad > 18);`

Cuando `edad > 18` asigna 1 a `MayorEdad`, sino le asigna 0.

# Conversión de tipos

- Frecuentemente se necesita convertir un valor de un tipo a otro, sin cambiar el valor que representa.
- Se hacen conversiones automáticas:
  - Cuando se asigna un valor de un tipo a una variable de otro tipo.
  - Cuando se combinan tipos mixtos en expresiones.
  - Cuando se pasan argumentos a funciones.

**Promoción integral**: int, unsigned int, long, unsigned long, float, double

El tipo de dato que viene primero en esta lista se convierte al que viene segundo. Por ejemplo, si los tipos operandos son int y long, el operando int se convierte en long.

- **Conversión implícita** (automática): cuando los tipos básicos están mezclados en asignaciones y expresiones.

Los datos de tipos más bajos se convierten a datos de tipo más alto.

*Ejemplo:*

```
int i = 12;  
double x = 4;
```

```
x = x + i;
```

```
x = i / 5;
```

En una sentencia de asignación, el resultado final de los cálculos se reconvierte al tipo de la variable al que está siendo asignado.

*/\* el valor de i se convierte en double antes de la suma \*/*

*/\* primero se hace una división entera i / 5 == 2, luego el 2 se convierte a tipo double: 2.0 y se asigna a x \*/*

# Conversión de tipos

- Conversión explícita: cuando se fuerza la conversión con un operador de *conversión de tipos* (cast):

✓ Formato: **(tipo\_dato\_al\_cual\_convertir) valor o variable**

*Ejemplo:*

- (float) i;
- (int) 3.4;
- precio = (int) 16.77 + (int) 23.99

**Los operadores C++ respetan reglas de prioridad (precedencia) y de asociatividad**

# **LEER**

**❑ Capítulos 1, 2 y 3**

**Libro: “Fundamentos de Programación  
con el Lenguaje de Programación C++” –  
Benjumea-Roldan**

**❑ Capítulos 1, 2 y 3**

**Libro: “C++ para ingeniería y ciencias” –  
Gary J. Bronson, 2da Edición,**