



# Variables

## Conversiones

Universidad Católica Boliviana

MSc, José Jesús Cabrera Pantoja

# Recap

- Componentes de una computadora
- Compiladores
- Interpretes
- Representación de la información

# Memorias



# Representación de texto

- Ya que las computadoras manejan solo los números binarios se ha buscado la forma de estandarizar el mapeo de texto a números decimales
- American Standard Code for Information Interchange (ASCII)
- Los primeros treinta y dos caracteres de la tabla ASCII no se pueden imprimir. Corresponden a cosas como "comienzo de texto".



# Representación de texto

Character	ASCII code
A	65
B	66
C	67
D	68
E	69

# Sistemas Numéricos

- Decimal
- Binario
- Hexadecimal

# Sistemas Numericos

Cuadro 1: Sistemas numéricos para Sistemas Embebidos

Sistema numérico	Base	Símbolos	Interpretación de "11"
Binario	2	0 1	3
Decimal	10	0 1 2 3 4 5 6 7 8 9	11
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 a b c d e f	17

# Sistemas Numericos: Hex

- Cada numero Hexadecimal tiene 4 bits:
  - 0b1111 -> 0xF
  - 0b0000 -> 0x0
  - 0b0000 1111 -> 0x0F



# Sistemas Numericos

Cuadro 1: Sistemas numéricos para Sistemas Embebidos

Sistema numérico	Base	Símbolos	Interpretación de "11"
Binario	2	0 1	3
Decimal	10	0 1 2 3 4 5 6 7 8 9	11
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 a b c d e f	17

$$153_b = 1 * b^2 + 5 * b^1 + 3 * b^0$$

$$153_{16} = 1 * 16^2 + 5 * 16^1 + 3 * 16^0$$

```
1 int myNumber = 339;  
2 int myNumber = 0x153;  
3 int myNumber = 0b101010011;
```

# Hoy veremos

- Input-Output en C++
- Variables
- Tipos de variables
- Casting (Conversiones)
- Entradas



# Variables

# C++



```
1 #include <iostream>
2
3
4 int main() {
5     std::cout << "Hello World";
6     return 0;
7 }
```



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```

# C++



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello World";
7     return 0;
8 }
```

# C++



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello World";
7     return 0;
8 }
```

Utilidades (Incluirlas) iostream

# C++



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello World";
7     return 0;
8 }
```

Punto de entrada de nuestro programa. Se llama programa principal o función principal (main)

# C++

6

```
cout << "Hello World";
```

Muestra el  
contenido  
en la  
consola  
(Terminal)  
**Content  
Output**

Inserta el  
contenido  
que viene  
después de  
estos  
símbolos en  
**cout**

Contenido que es escrito en  
comillas dobles "Hola"

Final de la instrucción!



# C++



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello World";
7     return 0;
8 }
```

Fin del programa. En caso  
de una ejecución exitosa

# C++



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "To be, or not to be," << endl;
7     cout << "that is the question" << endl;
8     return 0;
9 }
```

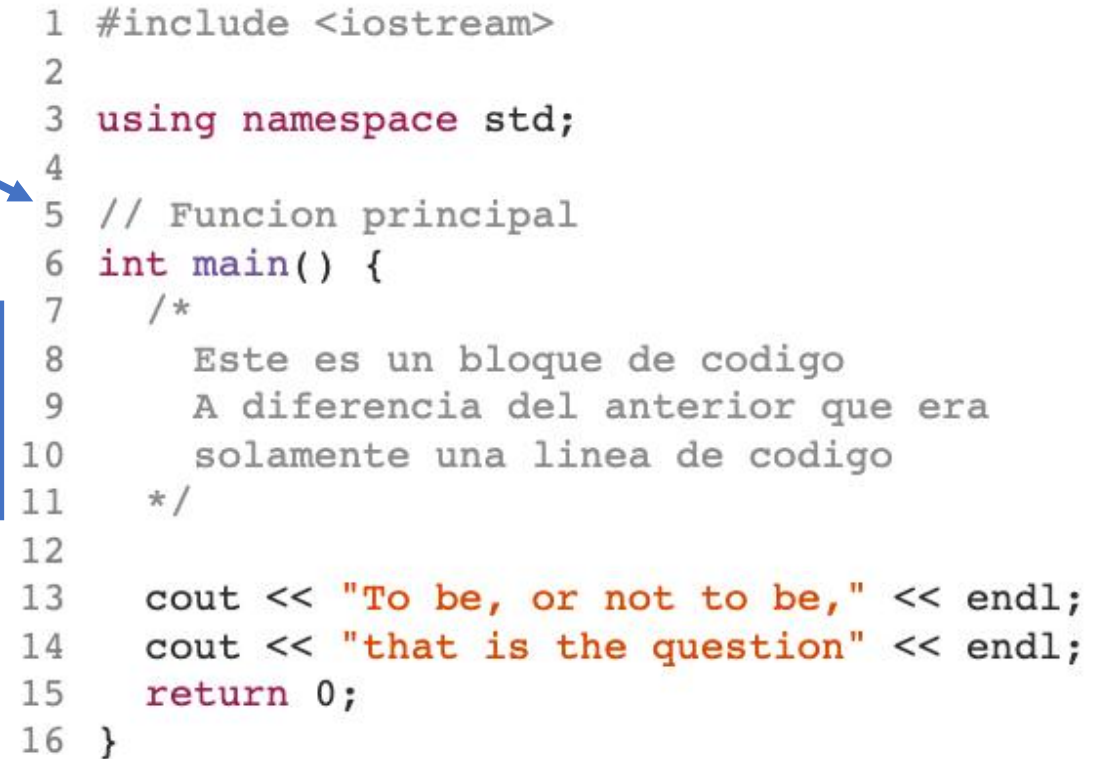
# Comentarios

- Comentarios en línea (//)
- Bloques de comentarios

/\*

aquí su comentario

\*/



```
1 #include <iostream>
2
3 using namespace std;
4
5 // Funcion principal
6 int main() {
7     /*
8         Este es un bloque de codigo
9         A diferencia del anterior que era
10        solamente una linea de codigo
11    */
12
13    cout << "To be, or not to be," << endl;
14    cout << "that is the question" << endl;
15    return 0;
16 }
```

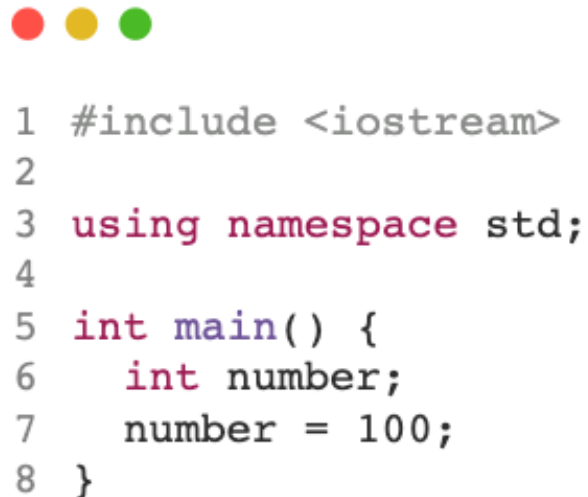
# Blindaje

Error!!!

```
1 #include <iostream>
2
3 using namespace std;
4
5 // Funcion principal
6 int main() {
7     /*
8         Este es un bloque de codigo
9         A diferencia del anterior que era
10        solamente una linea de codigo
11     */
12
13     cout << "Quote from "Hamlet":" << endl;
14     cout << "To be, or not to be," << endl;
15     cout << "that is the question" << endl;
16     return 0;
17 }
```

# Variables en C++


- Declaración de una variable:
  - **tipo\_de\_variable nombre\_de\_variable**
- Inicialización de una variable:

A code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The window contains C++ code with line numbers 1 through 8 on the left. The code includes a header, namespace declaration, and a main function with a variable declaration and assignment.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int number;
7     number = 100;
8 }
```

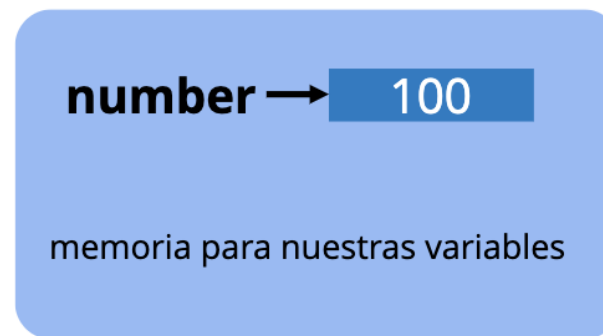
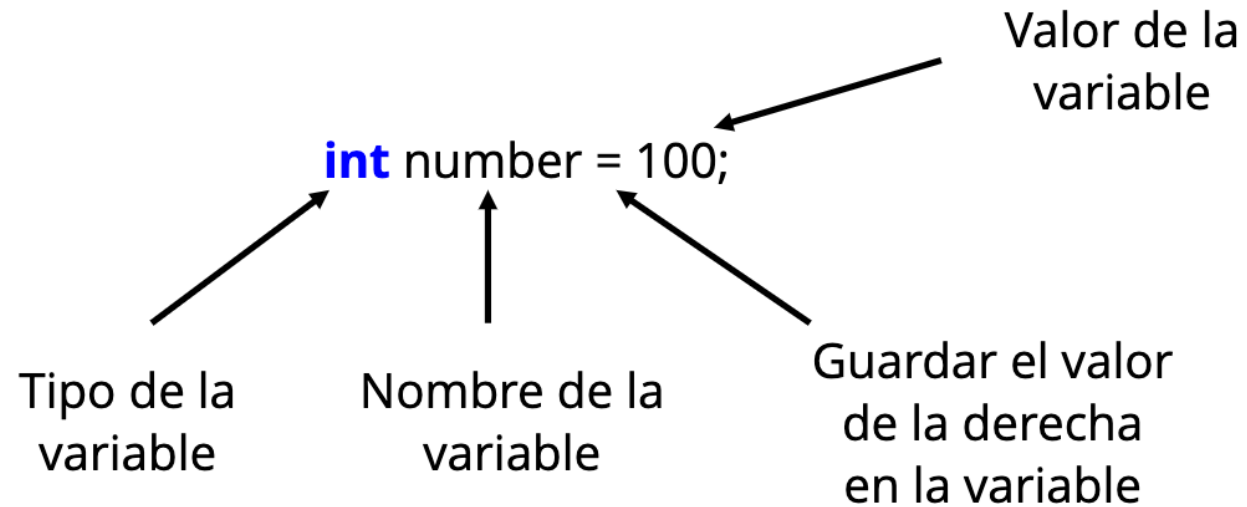
# Variables en C++

- Declaración e inicialización de una variable:
  - **tipo\_de\_variable nombre\_de\_variable = 100;**



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int number = 100;
7 }
```

# Variables en C++



RAM


# Variables: Strings



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6
7     // Delcaracion de una variable y definicion de su valor
8     string text = "Hello from C++ ^_^"s;
9
10    // Uso de una variable
11    cout << text << endl;
12    return 0;
13 }
```




# Variables: Strings




```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6
7     // Delcaracion de una variable y definicion de su valor
8     string text_begin = "Hello from"s;
9     string text_end = "C++ ^_^"s;
10
11     // Mostrar ambas variables sin olvidar de agregar el espacio
12     cout << text_begin << " "s << text_end << endl;
13     return 0;
14 }
```

# Variables: Strings



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Declaracion e inicializacion de la variable text
7     string text = "Hello from C++ ^_^";
8
9     // Mostrar la variable text
10    cout << text << endl;
11
12    // Escribimos un nuevo valor en la variable text.
13    // El valor de la variable text cambia a Happy coding!
14    text = "Happy coding!";
15    cout << text << endl;
16 }
```

# Variables: Enteros



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Declaracion de una variable current_amount
7     int current_amount;
8
9     // Inicializacion de la variable current_amount igual a 100
10    current_amount = 100;
11
12    // Mostrar el valor de la variable current_amount
13    cout << "Your current amount is: " << current_amount << endl;
14
15    // Actualizar el valor de la variable current_amount
16    current_amount = 120;
17
18    // Mostrar el nuevo valor de la variable current_amount
19    cout << "Your current amount is: " << current_amount << endl;
20 }
```

# Variables: Enteros



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int number = 10;
7     cout << "Number = " << number << endl;
8
9     number = 20;
10    cout << "Number = " << number << endl;
11
12    number = 30;
13    cout << "Number = " << number << endl;
14 }
```

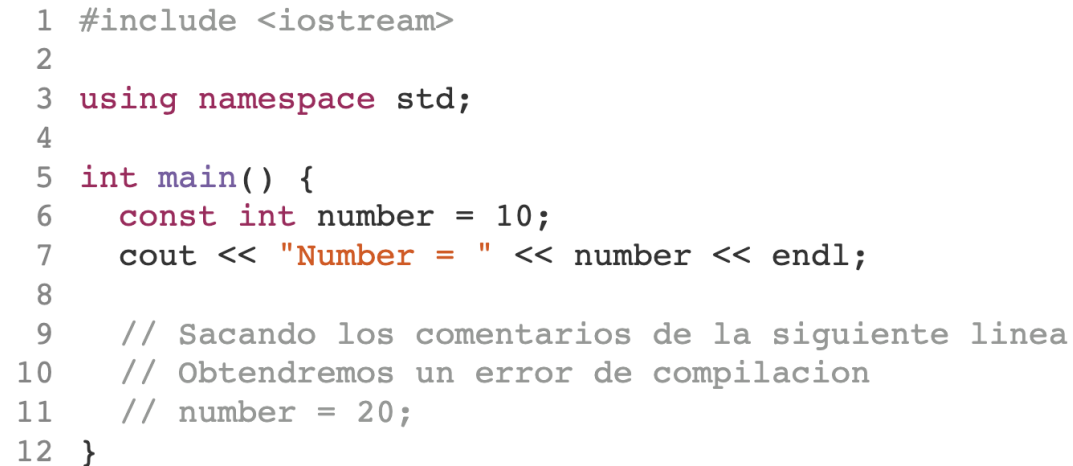
# Variables: Constantes Enteros

- Las constantes son similares a las variables excepto que no podemos cambiar su valor durante la ejecución del código.
- En C++, podemos usar la palabra clave **const** para declarar una constante. La sintaxis básica para crear una constante es:
- **const**    **tipo\_de\_constante**    **nombre\_de\_variable = 100;**

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     const int number = 10;
7     cout << "Number = " << number << endl;
8
9     // Sacando los comentarios de la siguiente linea
10    // Obtendremos un error de compilacion
11    // number = 20;
12 }
```

# Variables: Constantes Enteros

- **Error común de programación:** en C++, debe inicializar una constante en el momento de su declaración. Si no inicializa una constante en el momento de crearla, se producirá un error.



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     const int number = 10;
7     cout << "Number = " << number << endl;
8
9     // Sacando los comentarios de la siguiente linea
10    // Obtendremos un error de compilacion
11    // number = 20;
12 }
```

# Memoria

- Considere una hoja de Excel que consta de una gran cantidad de celdas donde cada celda se usa para almacenar datos. Podemos ubicar cada celda en la hoja de Excel usando un número de fila y columna.
- La memoria de la computadora es como una hoja de Excel que contiene celdas de datos dispuestos en un orden lógico.
- Cada celda en la memoria puede almacenar **1 byte** de datos. Como sabemos, **1 byte = 8 bits**; por lo tanto, cada celda puede almacenar cualquier valor entre 0 y 255.

# Memoria

- Como sabemos cuanta memoria hay que guardar en este programa?



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int number = 10;
7     cout << "Number = " << number << endl;
8 }
```



# Variables: Tipos

- El **tipo de datos** le dice al compilador qué tipo de datos puede almacenar una variable en particular. El compilador asigna la memoria a la variable en función de su tipo de datos.

# Variables: Tipos

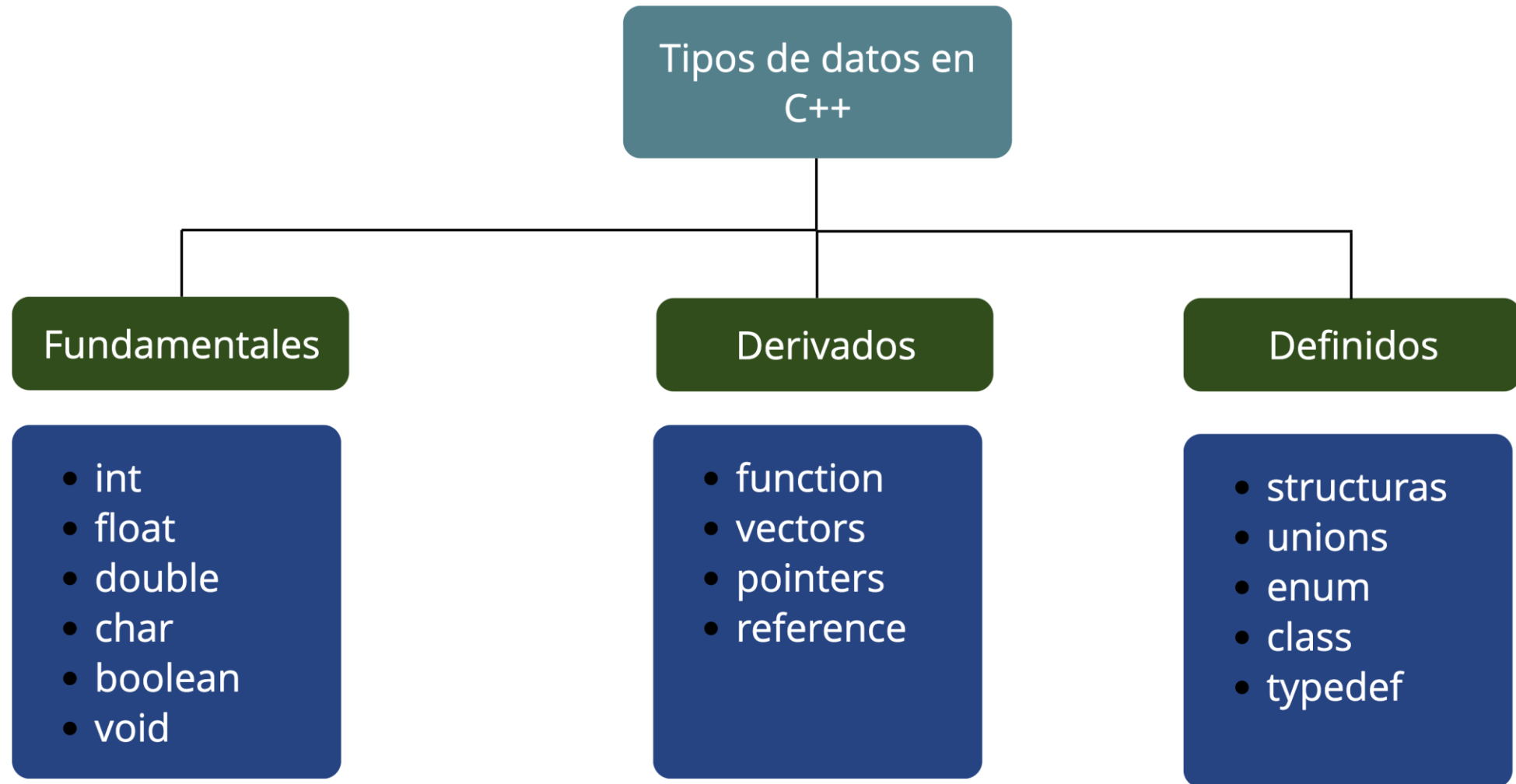
- Los **tipos de datos** hacen las dos cosas siguientes:
  - Especifica el tipo de valor que puede almacenar una variable (int, string)
  - Reserva el número de bytes para una variable en memoria, es decir, una variable de tipo **int** reservará **cuatro bytes** consecutivos en memoria.
  - Con **4 bytes**, podemos representar cualquier valor desde -2147483648 hasta 2147483647. Por lo tanto, el rango de valores que puede almacenar una variable depende de su tipo de datos.

# Variables: Tipos

- C++ admite los siguientes tipos de datos:
  - Tipos de datos primitivos o fundamentales
  - Tipos de datos derivados
  - Tipos de datos definidos por el usuario

# Variables: Tipos de datos

ng



# Tipos de datos: Fundamentales



```
1 // almacena los numeros en 4 bytes
2 int integer_number = 100;
3
4 // almacena los numeros en 4 bytes
5 float float_number = 10.7;
6
7 // almacena los numeros en 8 bytes
8 double double_number = 10.65417;
```

# Diferencia entre float y double

- La precisión de un número **float** es el número de dígitos que se pueden almacenar después de un punto decimal.
- Un **float** puede almacenar siete dígitos después de un punto decimal con precisión. Mientras que el **double** puede almacenar 15 dígitos después de un punto decimal con precisión. Se recomienda utilizar double para valores de coma flotante.
- **Nota:** Podemos almacenar un número en notación científica en tipos de datos dobles o flotantes. El número después de **e** muestra la potencia de 10.

# Diferencia entre float y double



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos variables de distintos tipos
7     int integer_number = 10;
8     float float_number = 10.5;
9
10    // Guardamos valores en notacion cientificia
11    // e representa exponente 10
12    float float_scientific = 97e4;
13    double double_number = 10.5;
14
15
16    // Mostramos los valores de la variable
17    cout << "int = " << integer_number << endl;
18    cout << "float = " << float_number << endl;
19    cout << "float_scientific = " << float_scientific << endl;
20    cout << "double = " << double_number << endl;
21
22 }
```

# Tipos de datos: Fundamentales



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     char character = 'A';
7     bool boolean = true;
8
9     // Mostrar los valores en la pantalla
10    cout << "char = " << character << endl;
11    cout << "bool = " << boolean << endl;
12
13 }
```



# Char y Boolean

- El tipo de datos **char** contiene un solo carácter del conjunto ASCII
- El tipo de datos **booleano** almacena un valor lógico. Puede almacenar **true** y **false**. También podemos usar 1 para representar **true** y 0 para representar **false**
- Nota: los **char** y **booleans** también se almacenan como **números**.

# Char y Boolean

- Que devolverá este código?



```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     long a = false;
7     int b = 'C';
8
9     cout << "variable a = " << a << endl;
10    cout << "variable b = " << b << endl;
11    return 0;
12 }
```

# Operaciones



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     double x = 1;
7     double y = 3;
8     cout << x + y << endl // 4
9         << x - y << endl // -2
10        << x * y << endl // 3
11        << x / y << endl; // 0.333333
12 }
```

# Pruebe le codigo

- Que devolverá este codigo?



```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int number = 18.9;
6
7     cout << "Number = " << number << endl;
8 }
```

# Pruebe le codigo


- Que devolverá este codigo?



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int x = 42;
7     double y = x; // 42.0
8     double z = -(x + y + 0.8); // -84.8
9
10    // El redondeo se realiza al lado del cero
11    // los decimales se cortan
12    int w = z; // -84
13 }
```

# Conversiones

- Se puede convertir explícitamente **double** a **int**. Hay una operación **static\_cast** para esto. Se aplica así:



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int x;
7
8     // Entrada al usuario
9     cin >> x;
10
11     // el resultado de la operacion (x + 0.5) se convertira a double,
12     // luego static_cast<int>(...) explicitamente se convertira a
13     // int – cortando la parte decimal
14     cout << static_cast<int>(x + 0.5) << endl;
15 }
```

# Conversiones



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos una variable de tipo char
7     char character = 'A';
8
9     // Declaramos una variable de tipo int
10    int ASCII;
11
12    // Convertimos char type a int (explicitamente)
13    ASCII = (int) character;
14
15    // Mostramos el valor de la variable
16    cout << "ASCII value = " << ASCII;
17
18 }
```

# Tipos de datos: Fundamentales

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-9223372036854775808 to 9223372036854775807
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 18446744073709551615
long long int	8bytes	$-(2^{63})$ to $(2^{63})-1$
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	



# Tipo de datos: Modificaciones

- Que pasa si queremos un numero entero mas grande?
- No se puede?

	int	4bytes	-2147483648 to 2147483647
--	-----	--------	---------------------------

- Pruebe el siguiente código

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     // Creamos una variable
6     int number = 2147483649;
7
8     // Mostremos el valor
9     cout << number;
10
11 }
```

# Tipo de datos: Modificaciones


- C++ admite los siguientes modificadores de tipo de datos:
  - long
  - short
  - unsigned
  - signed

# Modificadores: long

- **long** se utiliza para aumentar la longitud de un tipo de datos a 4 bytes más. Podemos usar **long** con tipos de datos **int** y **double**. Usemos un modificador largo con tipos de datos integrados.

# Modificadores: long

- Pruebe el siguiente código




```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos dos variables de tipo int y long int
7     int integer = 2147483649;
8     long int long_integer = 2147483649;
9
10    // Mostramos el contenido de las variables
11    cout << "integer = " << integer << endl;
12    cout << "long_integer = " << long_integer << endl;
13
14 }
```

# Modificadores: short

- **short** reduce la longitud disponible de un tipo de datos a 2 bytes. Podemos usar short con un tipo de datos **int**.

# Modificadores: short

- Pruebe el siguiente código




```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos dos variables (int, short int)
7     int integer = 32768;
8     short int short_integer = 32768;
9
10    // Se muestra las variables
11    cout << "integer = " << integer << endl;
12    cout << "short_integer = " << short_integer << endl;
13
14 }
```

# Modificadores: unsigned

- **unsigned** nos permite almacenar solo valores positivos. Podemos usar **unsigned** con tipos de datos **char** e **int**.
- Con **unsigned int**, podemos almacenar cualquier valor de 0 a 4294967295. Con **unsigned char**, podemos almacenar cualquier valor de 0 a 255

# Modificadores: unsigned

- Pruebe el siguiente código





```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos variables
7     int integer = -10;
8     unsigned int unsigned_integer = -10;
9
10    char character = 'A';
11    unsigned char unsigned_character = 'B';
12
13    // Mostramos las variables
14    cout << "integer = " << integer << endl;
15    cout << "unsigned_integer = " << unsigned_integer << endl;
16
17    cout << "character = " << character << endl;
18    cout << "unsigned_character = " << unsigned_character << endl;
19
20 }
```



# String: Secuencias de escape

- Una secuencia de escape consta de dos o más caracteres que se utilizan para modificar el formato de la salida.
- El primer carácter de la secuencia de escape es la barra invertida \. Los caracteres restantes determinan qué hará realmente nuestra secuencia de escape. Aquí hay una lista de las secuencias de escape más utilizadas:

# Tipo de variables: string

Escape sequence	Description	Representation
Simple escape sequences		
\'	single quote	byte 0x27 in ASCII encoding
\"	double quote	byte 0x22 in ASCII encoding
\?	question mark	byte 0x3f in ASCII encoding
\\	backslash	byte 0x5c in ASCII encoding
\a	audible bell	byte 0x07 in ASCII encoding
\b	backspace	byte 0x08 in ASCII encoding
\f	form feed - new page	byte 0x0c in ASCII encoding
\n	line feed - new line	byte 0x0a in ASCII encoding
\r	carriage return	byte 0x0d in ASCII encoding
\t	horizontal tab	byte 0x09 in ASCII encoding
\v	vertical tab	byte 0x0b in ASCII encoding
Numeric escape sequences		
\nnn	arbitrary octal value	byte <i>nnn</i> (1~3 octal digits)
\o{n...} (since C++23)		byte <i>n...</i> (arbitrary number of octal digits)
\xn...	arbitrary hexadecimal value	byte <i>n...</i> (arbitrary number of hexadecimal digits)
\x{n...} (since C++23)		
Conditional escape sequences <sup>[1]</sup>		
\c	Implementation-defined	Implementation-defined
Universal character names		
\unnnn	arbitrary <a href="#">Unicode</a>  value; may result in several code units	code point U+ <i>nnnn</i> (4 hexadecimal digits)
\u{n...} (since C++23)		code point U+ <i>n...</i> (arbitrary number of hexadecimal digits)
\Unnnnnnnn		code point U+ <i>nnnnnnnn</i> (8 hexadecimal digits)
\N{name} (since C++23)	arbitrary character listed in <a href="#">ISO/IEC 10646</a> 	character named by <i>name</i> (see <a href="#">below</a> )

# Tipo de variables: string



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos variables
7     string text = "Hey12345";
8
9     // Mostramos la variable
10    cout << text;
11
12 }
```

# Tipo de variables: string



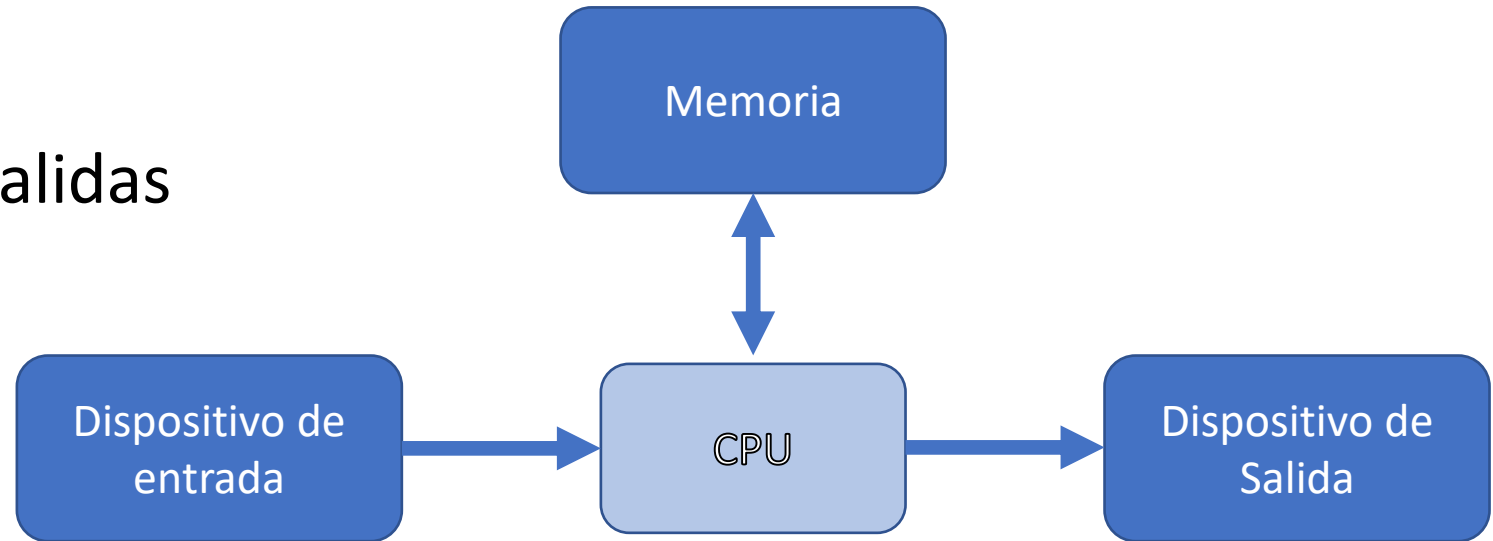
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos la variable string
7     // con texto y secuencias de escape
8     string text = "Hello\nI\tam\tJohn";
9
10    // Mostramos la variable
11    cout << text;
12 }
```



Entradas

# Componentes de una computadora

- Unidad de procesamiento Central (CPU)
- Memoria
- Dispositivos de entradas y salidas (I/O Devices)

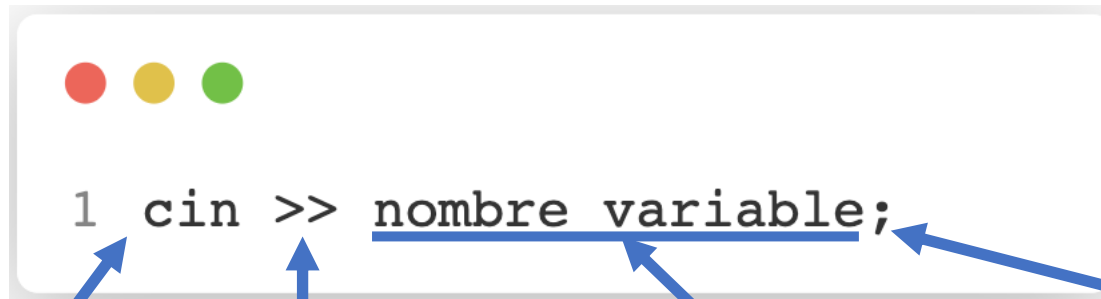




# Entradas

- La **operación de entrada** es exactamente lo contrario de la operación de salida. En la operación de entrada, tomamos datos del usuario y los almacenamos en la memoria.

# Entrada: C++



A diagram showing a C++ code snippet inside a window-like box with three colored dots (red, yellow, green) in the top-left corner. The code is `1 cin >> nombre variable;`. The text `nombre variable` is underlined. Four blue arrows point from text labels below to parts of the code: one to the line number `1`, one to the `cin` keyword, one to the `>>` operator, and one to the semicolon `;`.

```
1 cin >> nombre variable;
```

Toma (Pide)  
entrada del  
usuario

Extrae la entrada  
(valores) de **cin** y lo  
guarda dentro de  
la variable después  
de estos símbolos


Guarda los datos de la  
entrada del usuario

Final de la instrucción!



# Entrada: C++

- Pruebe el siguiente código



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     // Creamos una variable
7     float number;
8
9     // Mostramos un texto
10    cout << "Please enter your number:" << endl;
11
12    // Esperamos al usuario que introduzca una entrada
13    cin >> number;
14
15    // Mostramos la variable
16    cout << "You have entered: " << number;
17 }
```