



# PROGRAMACIÓN I

## C++

**UCM**

Grado en Estadística Aplicada. EUE.



2

## Tema 2.- Introducción a la Programación

# Introducción a la programación en C++

3

## Tema 2

- Aspectos prácticos: introducción al entorno de desarrollo C++
- Estructura de un programa
- Elementos básicos: palabras reservadas, identificadores, constantes literales, operadores y delimitador, y comentarios
- Tipos de datos básicos (carácter, enteros, reales, valores de verdad, cadenas de caracteres): dominio y operaciones; compatibilidad; prioridades.
- Variables, constantes y expresiones
- Instrucciones básicas: asignación (y variantes), entrada y salida



4

# Características del Lenguaje C++





# Entorno de programación

5

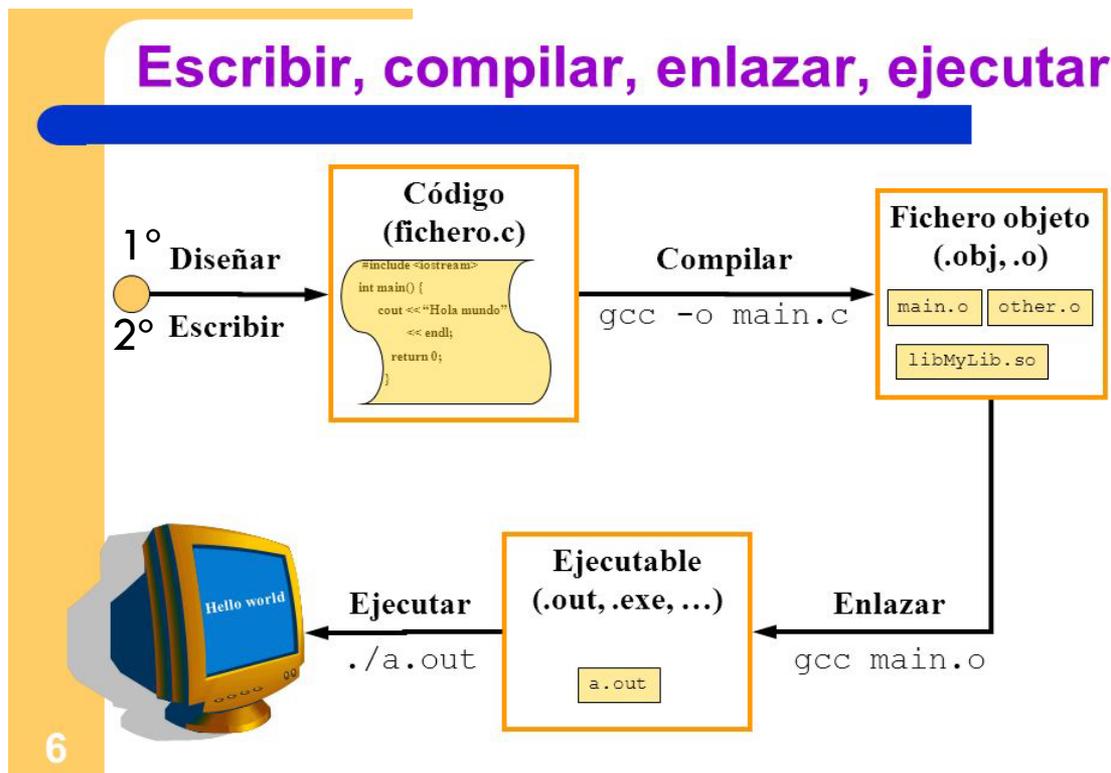
- El sistema consiste en 3 partes
  - ▣ Entorno de desarrollo (**Dev-C ++**)  
<http://www.bloodshed.net/devcpp.html>
  - ▣ Lenguaje
  - ▣ Biblioteca estándar
  
- Los programas C++ tienen las siguientes fases
  - ▣ **Edición**. Se escribe el programa (.cpp, .cxx, .cc o .C)
  - ▣ **Compilación**. Traducción del código a lenguaje máquina.
  - ▣ **Enlace**. Enlaza el código objeto para producir una imagen ejecutable
  - ▣ **Ejecución**. Crea un programa(.exe)



# Entorno de programación

6

- Los programas C++ tienen las siguientes fases
  - **Edición**. Se escribe el programa (.cpp, .cxx, .cc o .C)
  - **Compilación**. Traducción del código a lenguaje máquina.
  - **Enlace**. Enlaza el código objeto para producir una imagen ejecutable
  - **Ejecución**. Crea un programa(.exe)

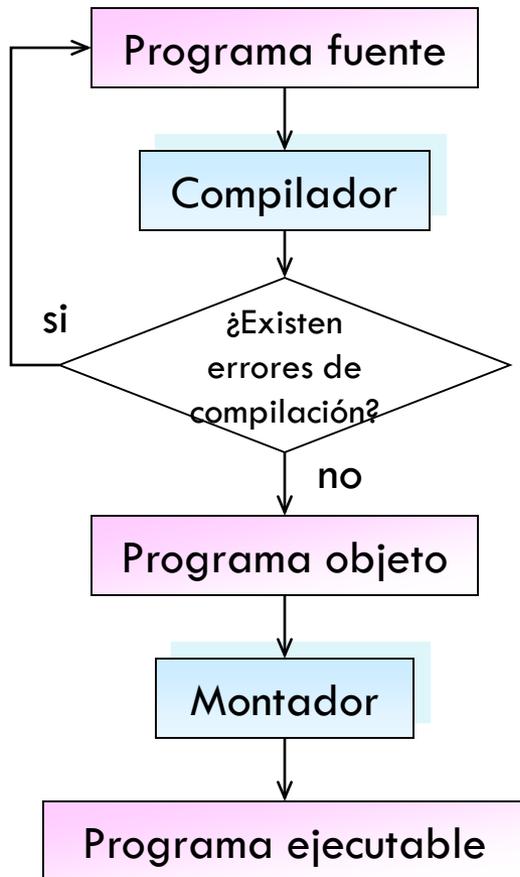


6

C++

# Fases en la ejecución de un programa

7

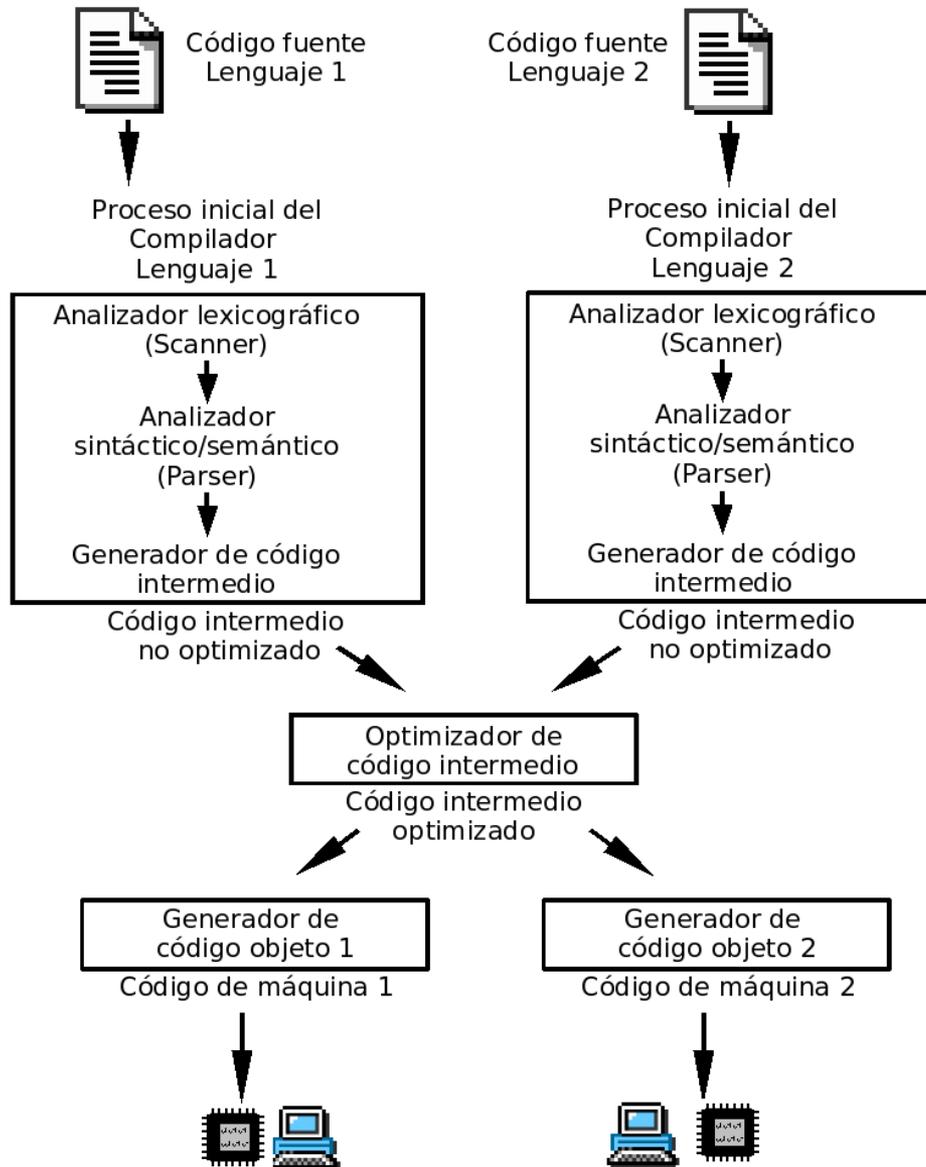


1. Escritura del programa fuente mediante un editor.
2. Traducir el programa mediante un compilador.
3. Verificar y corregir los errores de compilación.
4. Obtención del programa objeto.
5. Obtener el programa ejecutable mediante el montador.
6. Se ejecuta el programa y si no existen errores, se tendrá una salida.



# Entorno de programación

8



# Entorno de programación

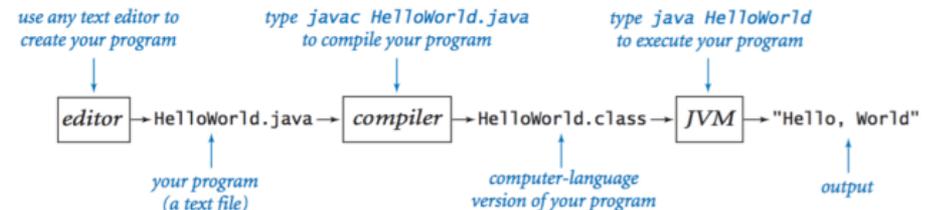
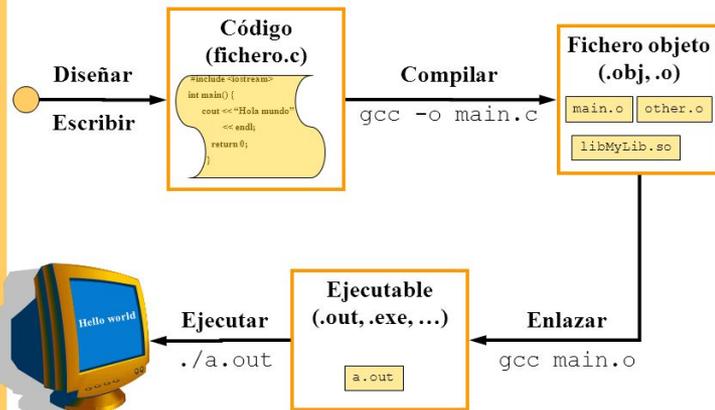
9

- Los programas C++ tienen las siguientes fases
  - **Edición.** Se escribe el programa (.cpp, .cxx, .cc o .C)
  - **Compilación.** Traducción del código a lenguaje máquina.
  - **Enlace.** Enlaza el código objeto para producir una imagen ejecutable
  - **Ejecución.** Crea un programa(.exe)

Compilador

vs Máquina interprete

## Escribir, compilar, enlazar, ejecutar



# Características del lenguaje C++

10

- Se distingue entre mayúsculas y minúsculas.  
(sensible a mayúsculas *case sensitive*)
- Fuertemente tipado.
- Palabras clave: siempre en minúsculas.
- Lenguaje estructurado  
pero no estrictamente estructurado en bloques  
(no se pueden definir funciones dentro de otras funciones).
- Todas las sentencias y declaración de variables terminan en punto y coma(;).
- La ejecución siempre comienza con la función main().



11

# Estructura de un programa





# Estructura básica de un programa

12

- Un programa en C++ consta de uno o más archivos de texto, cuyas líneas forman:
  - **Instrucciones, separadas por ;**
    - **Declarativas**, definen variables, tipos, clases, funciones,...
    - **Ejecutables**, son las que se convierten en código ejecutable: operaciones, sentencias de flujo, asignaciones, llamadas a funciones, etc...
    - **Compuestas**, grupo de sentencias, encerrados entre `{ }`
  - **Directivas**, información que le pasamos al compilador, llevan una `#` delante
  - **Comentarios**, `//` una línea y `/* */` varias líneas
  - **Expresiones**, combinación de constantes, variables, operadores, funciones y paréntesis. Todas las expresiones tienen un tipo



# Estructura de un programa

13

Componente  
estructural  
básico: la función

**Funciones**

**main**

Es obligatoria  
Punto de llamada  
del sistema  
operativo

```
Directivas de preprocesador
Declaraciones globales ( variables globales, funciones, ...)

función main()
{
    secuencia de declaraciones e instrucciones
}

función1()
{
    secuencia de declaraciones e instrucciones
}
...
funciónN()
{
    secuencia de declaraciones e instrucciones
}
```



# Mi primer programa "Hola Mundo!"

14

```
//Isabel Riomoros
// Este es mi primer programa
#include <iostream>
using namespace std;
int main()
{
    cout << "Hola Mundo!" << endl;
    system("pause");
    return 0;
}
```

Hola Mundo!

Pulse enter para continuar!



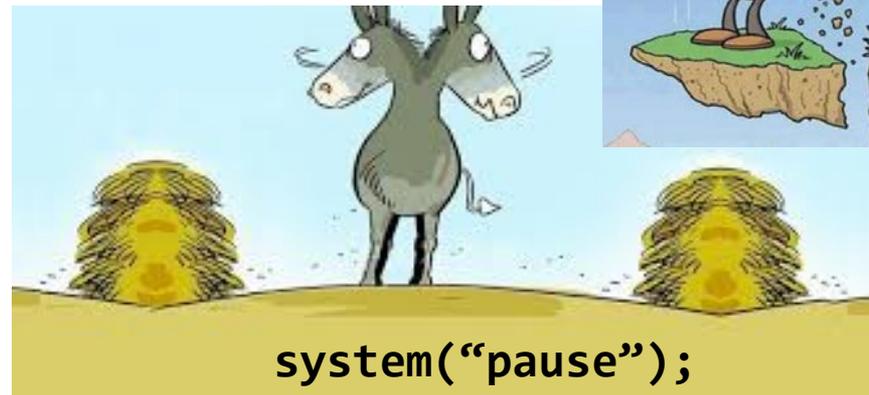
# Mi primer programa "Hola Mundo!"

15

```
//Isabel Riomoros
// Este es mi primer programa
#include <iostream>
using namespace std;
int main()
{
    cout << "Hola Mundo!" << endl;
    system("pause");
    return 0;
}
```

Hola Mundo!

Pulse enter para continuar!





# Mi primer programa

16

```
//primer programa en C++
#include <iostream>
// Comienza la ejecución del programa
int main()
{
    cout << "Hola mundo!\n";
    return 0;
}
//Fin de la ejecución del programa
```

**Comentario**- Propósito del programa

**Directiva del preprocesador**

**<< Operador de inserción de flujo**

**\ carácter de escape**

**\n línea nueva**

**El valor 0, indica que el programa termina de forma exitosa**



# Ejemplo: Intercambiar el valor de dos variables

17

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

int main()
{
    int x, y;
    int aux;

    cin >> x >> y;

    aux = x;
    x = y;
    y = aux;

    cout << x << y;

    return 0;
}
```

Directiva de preprocesamiento

Declaración de variables locales

Flujo de entrada

Flujo de salida por estándar (pantalla)

<< : operador de inserción para flujos de salida

Valor que se devuelve al S.O. Es la salida de la función main.



# Directivas del preprocesador

18

- Los compiladores de C++ proporcionan bibliotecas de funciones.
- Cada biblioteca de funciones tiene asociada un archivo de definición que se denomina *cabecera*.

```
# include <iostream.h>
```

**Indica al compilador que lea las directivas antes de compilar la función principal**

Si las librerías son heredadas de C se pone de forma obligatoria **.h**, o delante del nombre de la librería la letra **c**, p.e, `stdlib.h` o `cstdlib`

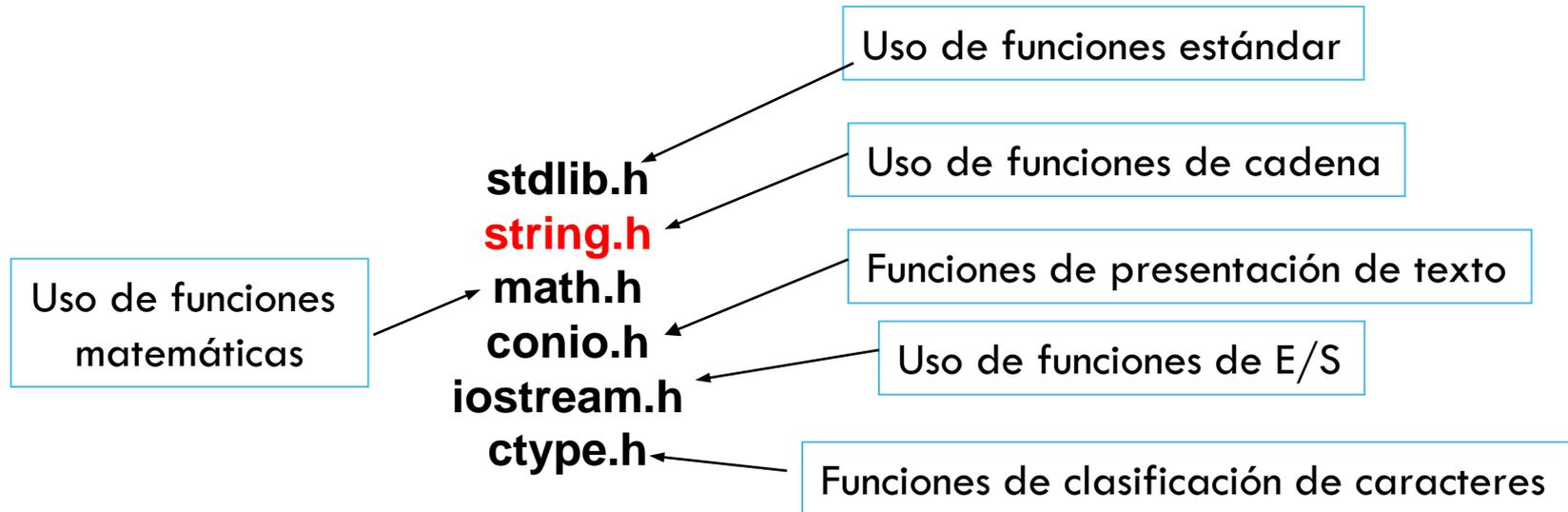
**Las directivas más usuales son:**

```
# include  
# define
```



# Directivas del preprocesador

- Existen archivos de cabecera estándar muy utilizados



El uso más frecuente en C++ de las directivas del preprocesador es la inclusión de archivos de cabecera, pero también se usan para definir macros, **nombres de constantes**, etc.



# Directivas del preprocesador

20

- Nombres de cabeceras
  - ▣ las viejas (de C o de C++ no estándar)
  - ▣ las nuevas (de C++ estándar que usan namespace std)

C C++ no estándar	C++ estándar ( <i>using namespace std</i> )	
<code>#include &lt;stdlib.h&gt;</code>	<code>#include &lt;cstdlib&gt;</code>	Funciones de propósito general
<code>#include &lt;math.h&gt;</code>	<code>#include &lt;cmath&gt;</code>	Funciones matemáticas
<code>#include &lt;stdio.h&gt;</code>	<code>#include &lt;cstdio&gt;</code>	Manipular datos de E/S
<code>#include &lt;iostream.h&gt;</code>	<code>#include &lt;iostream&gt;</code>	Uso de funciones de E/S
<code>#include &lt;string.h&gt;</code>	<code>#include &lt;string&gt;</code>	Uso de funciones de cadena



# La función main()

21

- Todo programa C++ tiene una única función main() que es el punto inicial de entrada al programa. Llamada desde el sistema operativo o comando.
- Una función C++ es un subprograma que devuelve un valor, un conjunto de valores o realiza una tarea específica.

```
#include <iostream.h>
```

```
main()  
{  
  ...  
  ...  
}
```

Las sentencias escritas entre llaves se denominan **bloque**

Llamadas a otras funciones

```
#include <iostream.h>
```

```
int main()  
{  
  entrada_datos();  
  proceso_datos();  
  return 0;  
  ...  
}
```

- Si se intenta declarar dos funciones main() dentro del programa se produce error.



## Ejercicio fin de sección

22

- Realiza un programa en el que te presentes a tus compañeros. Es decir que salga por pantalla.
  - Nombre
  - Dirección
  - Teléfono
  - Email
  - Un saludo
  
- Realiza un programa que intercambie dos variables de tipo carácter, sabiendo que una variable carácter se la denomina **char**



# Elementos básicos

- ❑ Palabras reservadas
- ❑ Identificadores
- ❑ Constantes literales, operadores y delimitadores
- ❑ Comentarios y espacios en blanco





# Palabras reservadas

24

- Son un conjunto de palabras que tienen un significado predeterminado para el compilador, y solo pueden ser utilizadas con dicho sentido.
- Se escriben en minúscula

Por ejemplo:

***using, namespace, const, double, int, char, bool, void, for, while, do, if, switch, case, default, return, typedef, enum, struct, etc.***





# Identificadores

25

- Son nombres elegidos por el programador para representar entidades (tipos, constantes, variables, funciones, etc) en el programa.
- Se construyen mediante una secuencia de letras y dígitos, siendo el **primer carácter una letra**.
- El carácter '\_' se considera como una letra, sin embargo, los nombres que comienzan con dicho carácter se reservan para situaciones especiales, por lo que no deberán utilizarse en programas.
- Se utilizan para nombrar a:
  - Constantes Simbólicas
  - Tipos
  - Variables
  - Funciones
  - Campos de Registros



# Elección del identificador

26

- ❑ Deben de ser significativos
- ❑ No pueden contener espacios en blanco, ni ciertos caracteres especiales como letras acentuadas, la letra eñe, las barras \ o /, etc.
- ❑ El compilador determina la máxima longitud que pueden tener (por ejemplo, 31 caracteres)[uhhh]
- ❑ Sensibles a mayúsculas y minúsculas.
- ❑ No se podrá dar a un dato el nombre de una palabra reservada.
- ❑ No es recomendable usar el nombre de algún identificador usado en las bibliotecas estándar (por ejemplo, cout)
- ❑ El identificador de un dato debe reflejar su semántica (contenido).
- ❑ Usaremos minúsculas, salvo la primera letra de nombres compuestos (y los identificadores de las constantes).
- ❑ No se nombrarán dos datos con identificadores que difieran únicamente en la capitalización, o un sólo carácter.





# Constantes, operadores, delimitadores

27

## Constantes literales

- Son valores que aparecen explícitamente en el programa, y podrán ser lógicos, numéricos, caracteres y cadenas.  
Ejemplo: true, false, 0, 25, 166.386, " Pts", ' ', etc.

```
const tipo_de_dato nombre_de_la_constante = valor_de_la_constante;
```

```
const float E = 2.7183; //declaramos la constante con su valor
```

## Operadores

- Símbolos con significado propio **según el contexto** en el que se utilicen.
- **Ejemplo:** = << >> \* / % + - < > <= >= == != ++ -- . ,  
etc.

## Delimitadores

- Símbolos que indican comienzo o fin de una entidad.  
Ejemplo: ( ) { } ; , < >



# Comentarios y espacios en blanco

28

- Los espacios en blanco, tabuladores, nueva línea, retorno de carro, avance de página y los comentarios son ignorados por el compilador, excepto en el sentido en que separan elementos.
- Los comentarios en un programa es texto que el programador escribe para facilitar la comprensión, o remarcar algún hecho importante a quien lea nuestro programa, y son, por lo tanto, ignorados por el compilador.

**Los comentarios en C++ se expresan de dos formas diferentes:**

- **Comentarios hasta fin de línea:** los símbolos `//` marcan el comienzo del comentario, que se extiende hasta el final de la línea.
  - `//` acumula el siguiente número
  - `suma = suma + n; //` acumula el valor de 'n'
- **Comentarios enmarcados:** los símbolos `/*` marcan el comienzo del comentario, que se extiende hasta los símbolos del fin del comentario `*/`
  - `/*`
  - `*` acumula el siguiente número
  - `*/`
  - `suma = suma + n; /*` acumula el valor de 'n' `*/`



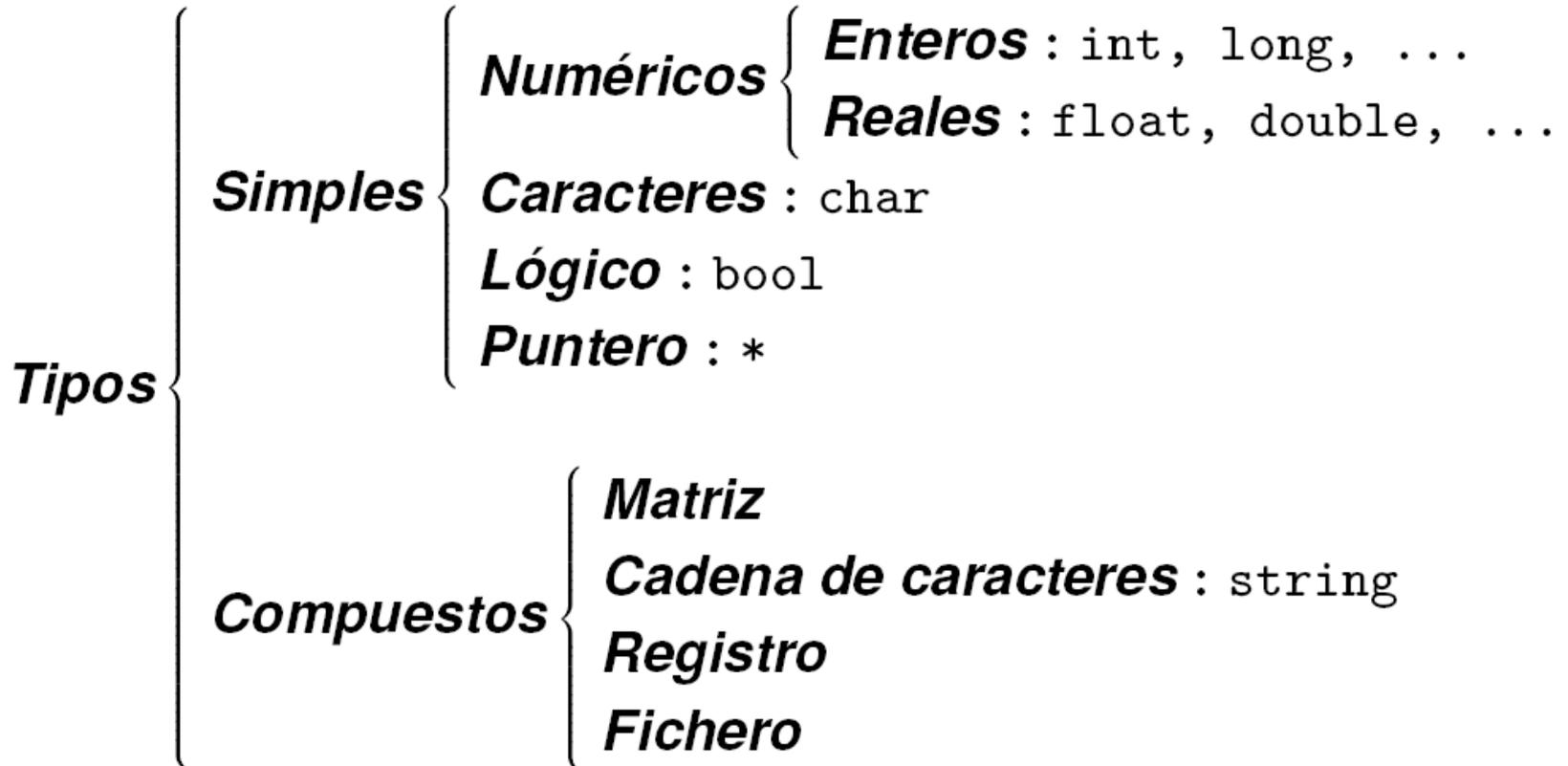


29

# Tipos de datos básicos

- ❑ Carácter,
- ❑ Enteros
- ❑ Reales
- ❑ Valores de verdad
- ❑ Cadenas de caracteres
- ❑ Compatibilidad de tipos





C++ es un lenguaje fuertemente tipado **C++**

# Tipos de datos básicos

31

- El **tipo de dato** **determina la naturaleza del valor** que puede tomar una variable.
- El **tipo de dato** **define un dominio de valores y las operaciones** que se pueden realizar con éstos valores.
- C++ dispone de unos **tipos** de datos **predefinidos** (simples) y permite al programador crear otros tipos de datos.





# Tipos de datos básicos

32

## Tipo de datos básicos

- **int** ( Números enteros )
- **float**( Números reales )
- **double**( Números reales más grandes que float )
- **bool**( Valores lógicos )
- **char** ( Caracteres y cualquier cantidad de 8 bits )
- **void**( Nada. Sirve para indicar que una función no devuelve valores )



# Tipo entero: int

int <identificador>

33

`_Bool`



7 0

ゼロビット目のみが意味を持ちます。1～7ビット目は不定値となります。

`-lang=c` オプションと `-strict_std` オプションを同時に指定している場合は、`_Bool` 型は C90 となります。

`char`



7 0

`signed char` (`unsigned` では符号ビットなし)



7 0

`short` (`unsigned` では符号ビットなし)



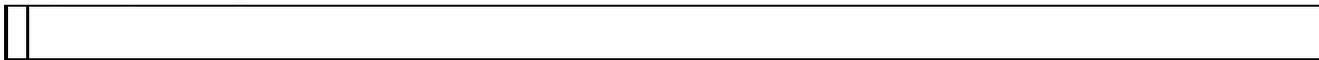
15 0

`int, long` (`unsigned` では符号ビットなし)



31 0

`long long` (`unsigned` では符号ビットなし)



63 0

`unsigned char`

`char (signed)`

`short (signed)`

`unsigned short`

`unsigned (int)`

`int (signed)`

`unsigned long`

`long (int)`



`-lang=c` オプションと `-strict_std` オプションを同時に指定している場合は、`long long` 型は C90 違反でエラーとなります。



# Tipo entero: int

int <identificador>

34

## Complemento a uno:

Complemento a uno  
con enteros de 4 bits

El número obtenido  
al invertir todos los bits

Rango  
 $-2^{N-1}$   
A  
 $2^{N-1}-1$ .

Binario	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

## Complemento a dos:

Complemento a dos  
con enteros de 4 bits

Agregando 1 = 011111

Rango  
 $-2^{N-1}$   
A  
 $2^{N-1}-1$ .

Binario	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8



# Tipo entero: int

int <identificador>

35

**Tamaño en bytes:** 4 bytes (32 bits)

**Dominio:** son todos los números enteros entre los valores

- 2.147.483.648 y 2.147.483.647

**Operaciones:**

		<b>int × int → int</b>
+	Suma	
-	Resta	
*	Producto	
/	División entera	
%	Resto de la división entera (módulo)	
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	<b>int → int</b>

Otros enteros

**unsigned char**

**char (signed)**

short (signed)

**unsigned short**

**unsigned (int)**

int (signed)

**unsigned long**

long (int)



# Tipo entero: cociente y resto de la división entera

36

## Cociente / división

```
int i = 43, j = 2;
```

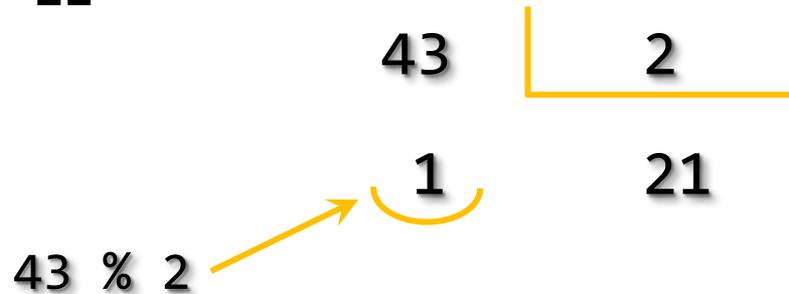
```
cout << i / j; // Muestra 21
```

## Resto % modulo

No se obtienen decimales → Queda un resto

$43 \% 2 = 1$

$43 / 2 = 21$



# Tipo entero: int

## Operadores de incremento y decremento

37

Suma una unidad a su argumento

++

--

Resta una unidad a su argumento

```
m = m + 1;  
m++;  
++m;
```

Son equivalentes



- Si precede al operando se realiza la operación ++ o -- y luego se realiza la asignación (**Prefijo**)

```
x = 10;  
y = ++x; // y vale 11
```

- Si sigue al operando, se realiza la asignación y posteriormente se realiza la operación ++ o -- (**Postfijo**)

```
x = 10;  
y = x++; // y vale 10
```



- **Tamaño en bytes:** 4 bytes
- **Dominio:** son todos los números reales que contienen una coma decimal comprendidos entre los valores:

$$3,4 \times 10^{-38} \quad \text{y} \quad 3,4 \times 10^{38}$$

- **Operaciones:**

		<b>float × float → float</b>
+	Suma	
-	Resta	
*	Producto	
/	División en coma flotante	
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	<b>float → float</b>



- ❑ **Tamaño en bytes: 8 bytes**
- ❑ **Dominio:** son todos los números reales que contienen una coma decimal comprendidos entre los valores:

$$1,7 \times 10^{-308} \quad \text{y} \quad 1,7 \times 10^{308}$$

- ❑ **Operaciones:**

		<b>double × double → double</b>
+	Suma	
-	Resta	
*	Producto	
/	División en coma flotante	
- , +	Signo negativo, positivo	
++	Incrementación	
--	Decrementación	<b>double → double</b>

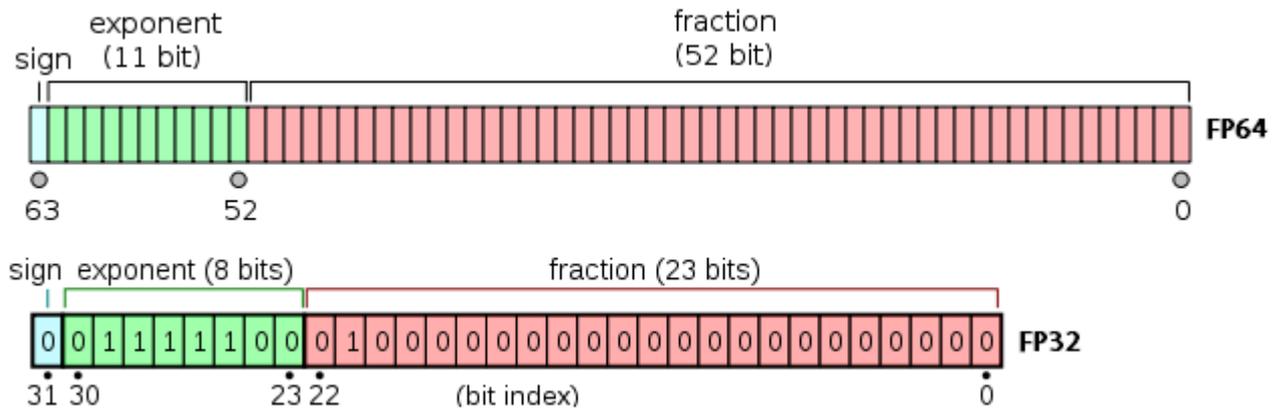


# Tipos de datos básicos

40

4,98 E-02

Tipo: tamaño y forma



Nombre  
(significativo)

Dirección de memoria

Palabra de memoria

Número	Característica	Mantisa	Logaritmo
45.600	4	0,6590	4,6590
4.560	3	0,6590	3,6590
456	2	0,6590	2,6590
45,6	1	0,6590	1,6590
4,56	0	0,6590	0,6590
0,456	-1	0,6590	0,6590-1
0,0456	-2	0,6590	0,6590-2
0,00456	-3	0,6590	0,6590-3



# Rango de los enteros y los reales

41

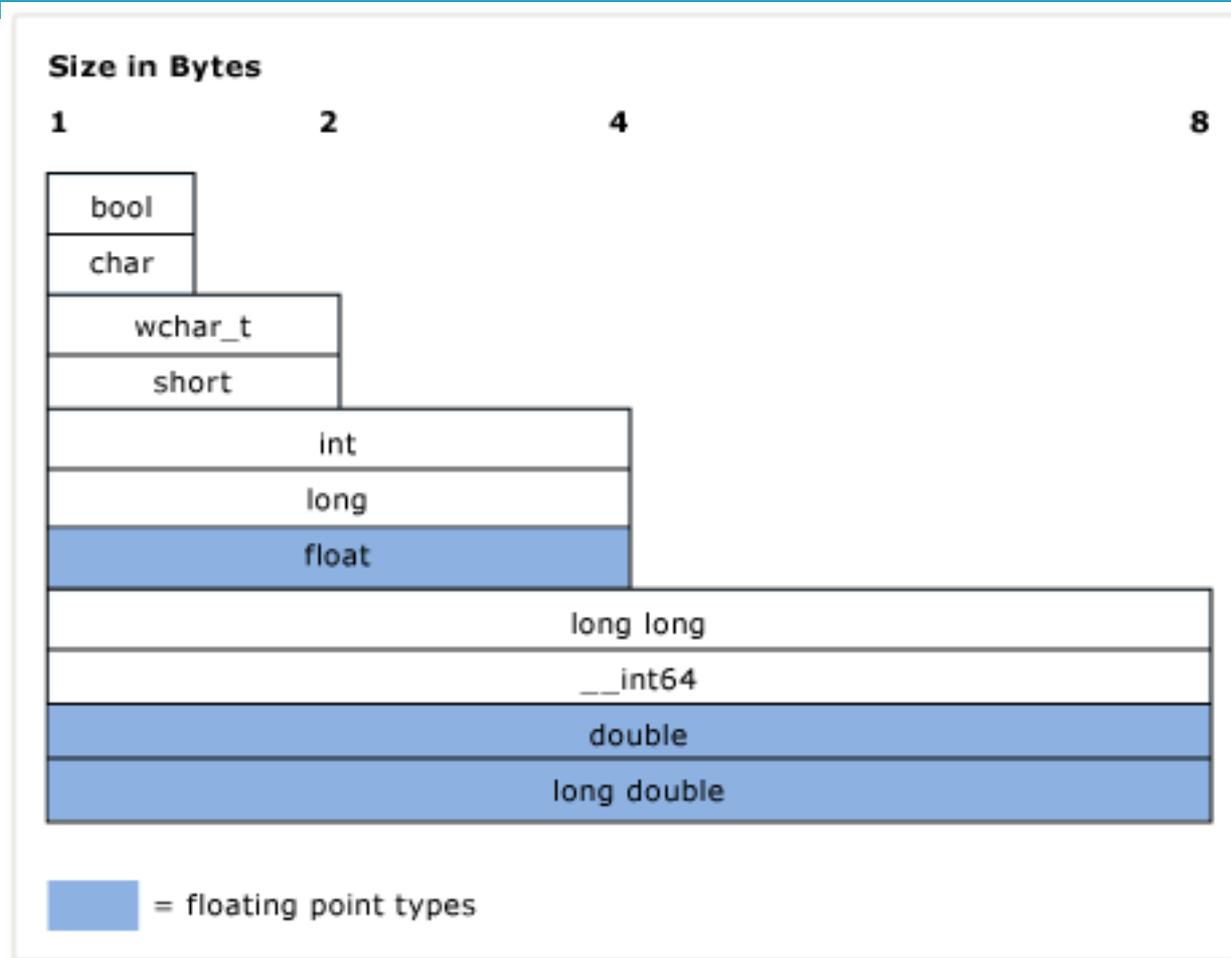
Tipo	bits	Rango / Tipo de uso	Constante
<b>unsigned</b> char	8	$0 \leq X \leq 255$	UCHAR_MAX
char (signed)	8	$-128 \leq X \leq 127$	SCHAR_MIN; SCHAR_MAX
short (signed)	16	$-32.768 \leq X \leq 32.767$	SHRT_MIN; SHRT_MAX
<b>unsigned</b> short	16	$0 \leq X \leq 65.535$	USHRT_MAX
<b>unsigned</b> (int)	32	$0 \leq X \leq 4.294.967.295.$	UINT_MAX
int (signed)	32	$-2.147.483.648 \leq X \leq 2.147.483.647$	INT_MIN; INT_MAX
<b>unsigned</b> long	32	$0 \leq X \leq 4.294.967.295$	LONG_MAX
long (int)	32	$-2.147.483.648 \leq X \leq 2.147.483.647$	LONG_MIN; LONG_MAX
float	32	$1.18e-38 \leq  X  \leq 3.40e38$	
double	64	$2.23e-308 \leq  X  \leq 1.79e308$	
long double	80	$3.37e-4932 \leq  X  \leq 1.18e4932$	

En limits.h están definidas las constantes que delimitan los rangos de los enteros. `#include <limits.h>`



# Rango de los enteros y los reales

42



En `limits.h` están definidas las constantes que delimitan los rangos de los enteros. `#include <limits.h>`



```
C:\ F:\c++ clases\programas\pruebaConstantes.exe
```

```
LONG_MAX    = 2147483647  
INT_MAX     = 2147483647  
USHRT_MAX   = 65535  
SHRT_MAX    = 32767  
SCHAR_MAX   = 127
```

```
Presione una tecla para continuar . . . _
```



# ¿División entera o división real?

44

Ambos operandos enteros → División entera

Algún operando real → División real

División	Resultado
$500 / 3$	166
$500.0 / 3$	166.667
$500 / 3.0$	166.667
$500.0 / 3.0$	166.667

Comprueba que el tipo de división es el que quieres



# Funciones predefinidas en las librerías

45

- Además de éstas operaciones, C++ dispone de un gran conjunto de **funciones matemáticas**.

- Funciones**

**abs:** int  $\rightarrow$  int

Calcula el valor absoluto de un número

**ceil:** double  $\rightarrow$  double

Calcula el número entero mayor o igual que el dado

**floor:** double  $\rightarrow$  double

Redondea por defecto el valor de un número

**fmod:** double  $\times$  double  $\rightarrow$  double

Calcula el resto de la división real de dos números

**sqrt:** double  $\rightarrow$  double

Calcula la raíz cuadrada de un número

**pow:** double  $\times$  double  $\rightarrow$  double

Calcula la potencia de un número

```
#include <math.h>
```

```
{  
x = abs(-7)           // x vale 7  
y = ceil (5.2)       // y vale 6  
z = floor (5.2)      // z vale 5  
resto = fmod(5.0, 2.0) // resto vale 1  
}
```

**math.h**  
**float.h**  
**complex.h**



# Funciones predefinidas en las librerías

46

- Redondear un número a la centésima cifra=3

```
resultado= ceil(resultado * pow(10,cifras))/pow(10,cifras);
```

```
cout<<"X = "<<fabs(7.5)<<" - 'fabs(7.5): Valor absoluto'"<<endl;
```

```
cout<<"X = "<<floor(7.5)<<" - 'floor(7.5): Redondea hacia abajo'"<<endl;
```

```
cout<<"X = "<<fabs(0.0)<<" - 'fabs(0.0) Valor absoluto'"<<endl;
```

```
cout<<"X = "<<ceil(0.0)<<" - 'ceil(0.0): Redondea hacia arriba'"<<endl;
```

```
cout<<"X = "<<fabs(-6.4)<<" - 'fabs(-6.4): Valor absoluto'"<<endl;
```

```
cout<<"X = "<<ceil(-6.4)<<" - 'ceil(-6.4): Redondea hacia arriba'"<<endl;
```

```
cout<<"X = "<<ceil(-fabs(-8 + floor(-5.5)))<<" - ceil(-fabs(-8 + floor(-5.5)))"
```

```
<<endl;
```

□



Ceil (2.3) = 3.0

Ceil(3.8) = 4.0

Ceil(-2.3) = -2.0

Ceil(-3.8) = -3.0

Uebun | rennans.com | funcionesMath.cpp

C:\ F:\c++ clases\programas\funcionesMath.exe

```
X = 7.5 - 'fabs(7.5): Valor absoluto'  
X = 7 - 'floor(7.5): Redondea hacia abajo'  
X = 0 - 'fabs(0.0) Valor absoluto'  
X = 0 - 'ceil(0.0): Redondea hacia arriba'  
X = 6.4 - 'fabs(-6.4): Valor absoluto'  
X = -6 - 'ceil(-6.4): Redondea hacia arriba'  
X = -14 - ceil(-fabs(-8 + floor(-5.5)))  
Presione una tecla para continuar . . .
```



# Funciones en math.h

48

---

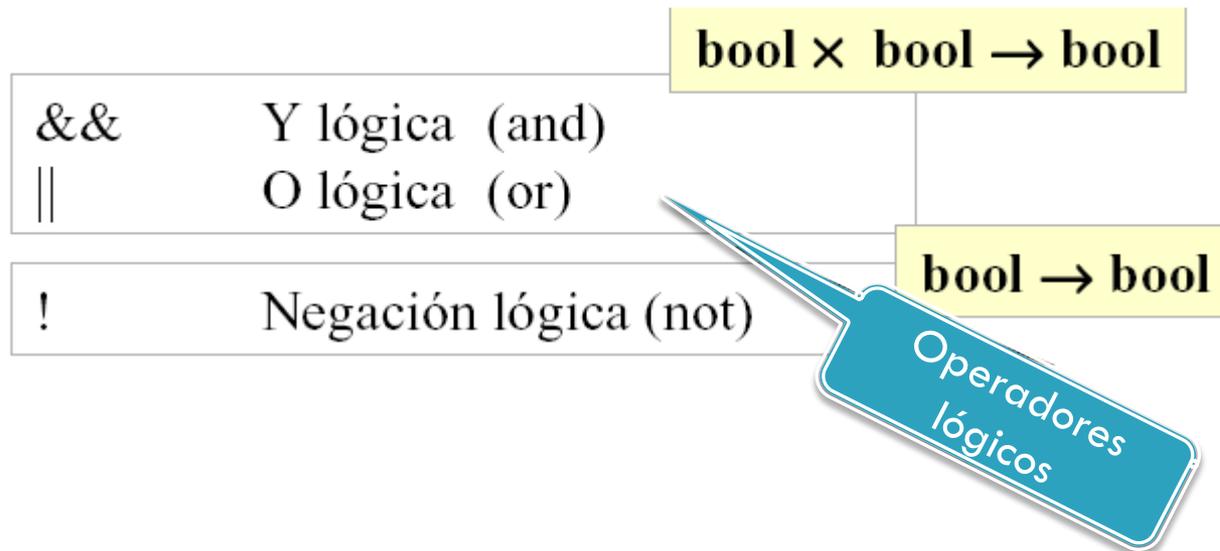
<code>double sin(double r);</code>	seno, $\sin r$ (en radianes)
<code>double cos(double r);</code>	coseno, $\cos r$ (en radianes)
<code>double tan(double r);</code>	tangente, $\tan r$ (en radianes)
<code>double asin(double x);</code>	arco seno, $\arcsin x$ , $x \in [-1, 1]$
<code>double acos(double x);</code>	arco coseno, $\arccos x$ , $x \in [-1, 1]$
<code>double atan(double x);</code>	arco tangente, $\arctan x$
<code>double atan2(double y, double x);</code>	arco tangente, $\arctan y/x$
<code>double sinh(double r);</code>	seno hiperbólico, $\sinh r$
<code>double cosh(double r);</code>	coseno hiperbólico, $\cosh r$
<code>double tanh(double r);</code>	tangente hiperbólica, $\tanh r$
<code>double sqrt(double x);</code>	$\sqrt{x}$ , $x \geq 0$
<code>double pow(double x, double y);</code>	$x^y$
<code>double exp(double x);</code>	$e^x$
<code>double log(double x);</code>	logaritmo neperiano, $\ln x$ , $x > 0$
<code>double log10(double x);</code>	logaritmo decimal, $\log x$ , $x > 0$
<code>double ceil(double x);</code>	menor entero $\geq x$ , $\lceil x \rceil$
<code>double floor(double x);</code>	mayor entero $\leq x$ , $\lfloor x \rfloor$
<code>double fabs(double x);</code>	valor absoluto de $x$ , $ x $
<code>double ldexp(double x, int n);</code>	$x2^n$
<code>double frexp(double x, int* exp);</code>	inversa de ldexp
<code>double modf(double x, double* ip);</code>	parte entera y fraccionaria
<code>double fmod(double x, double y);</code>	resto de $x/y$



# Tipo booleano: bool

49

- ❑ **Tamaño en bytes:** 1 byte
- ❑ **Dominio:** dos únicos valores: { true, false }
- ❑ **Operaciones:**



- ❑ No todos los compiladores de C++ tienen éste tipo de dato

Falso → Cero  
Verdadero → Distinto de cero



# Tipo booleano: bool

50

## Tabla de verdad

A	B	!A	A && B	A    B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F



## Operadores relacionales

==	Igual a
!=	Distinto
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

El resultado es booleano



# Tipo booleano: bool

52

- ❑ **Escritura de valores bool**
- ❑ Tipo de datos se utiliza sobre todo para evaluar expresiones lógicas.
- ❑ Si escribimos un dato de tipo bool cuyo valor es true, en consola se visualiza el valor 1. Para visualizar **true** debemos utilizar la función : **boolalpha**.

- ❑ La lectura es análogo.

```
#include <iostream>
using namespace std;
int main () {
    bool b=true;
    cout << boolalpha << b << endl;
    cout << noboolalpha << b << endl;
    return 0; }
```

true

1





# Ejercicios

53

Escribir expresiones booleanas para indicar:

1. Error es falso
2. Un número es par
3. Número de tres dígitos
4. Tres dígitos y par
5. Dados tres números comprobar si forman un triangulo
6. Comprobar si un número de tres cifras es capicúa



- **Tamaño en bytes:** 1 byte
- **Dominio:** dígitos, letras mayúsculas, letras minúsculas y signos de puntuación. Tipos de datos básicos en C++  
 $0 < 1 < 2 \dots < 9 < A < B < \dots < Z < a < b < \dots < z$
- Internamente, los caracteres se almacenan como números.
- El tipo **char** representa valores en el rango -128 y 127 y se asocian con el código ASCII.
  - ▣ **Así, el carácter 'A' se almacena como el número 65, etc ...**
- **Operaciones:**

Dado que los caracteres se almacenan internamente como números enteros, se pueden realizar operaciones aritméticas con los datos de tipo char. Se puede sumar un entero a un carácter para obtener otro código ASCII diferente.
- **Ejemplos:**
  - Para convertir una letra minúscula en mayúscula basta con restar 32.  
 $'a' - 32 = 'A'$
  - Para convertir una letra mayúscula en minúscula basta con sumar 32.  
 $'B' + 32 = 'b'$
  - Para convertir el carácter '4' en el número 4 basta con restar 48.  
 $'4' - 48 = 4$



# Tipo carácter: char

55

## Funciones:

**isdigit:** char → bool

Devuelve TRUE si el carácter es: '0', ... , '9'

**isalpha:** char → bool

Devuelve TRUE si el carácter es: 'A', ... , 'Z', 'a', ... , 'z'.

**islower:** char → bool

Devuelve TRUE si el carácter es una letra minúscula: 'a', ... , 'z'.

**isupper:** char → bool

Devuelve TRUE si el carácter es una letra mayúscula: 'A', ... , 'Z'.

**tolower:** char → char

Convierte un carácter mayúscula en minúscula.

**toupper:** char → char

Convierte un carácter minúscula en mayúscula.

```
#include <ctype>
```

```
{
```

```
...
```

```
char c = 'A';
```

```
c = tolower (c); // c vale 'a'
```

```
t = isdigit(c); // t vale 0 (FALSE)
```

```
...
```

```
}
```

```
char c = 65;
```

El archivo de cabecera que contiene éstas funciones es: **ctype.h**



# Tabla ASCII

## Standard Code for Information Interchange

56

- La tabla ASCII es comúnmente utilizada como base para la representación de los caracteres, donde los números del 0 al 31 se utilizan para representar caracteres de control, y los números del 128 al 255 se utilizan para representar caracteres extendidos.

Rep	Simb	Rep	Simb	Rep	Simb	Rep	Simb
0	\0	32	SP	64	@	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	\a	39	'	71	G	103	g
8	\b	40	(	72	H	104	h
9	\t	41	)	73	I	105	i
10	\n	42	*	74	J	106	j
11	\v	43	+	75	K	107	k
12	\f	44	,	76	L	108	l
13	\r	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL



# Prioridad de operadores

57

## Enteros y reales

### Prioridad de los operadores:

++, --	10*5++
-, + (unario)	-3
*, /, %	3*5
+, -	6+7



= 10\*6

## Lógicos

### Prioridad de los operadores

!  
&&, ||



# Prioridad de operadores

58

Operador	Tipo de Operador	Asociatividad
! ~ -	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.



# Evaluación de expresiones

59

$((3 + 5) * 4 + 12) / 4 - (3 * 2 - 1)$  Primero, los paréntesis...

$(8 * 4 + 12) / 4 - (6 - 1)$   
\* antes que -

$(32 + 12) / 4 - 5$   
\* antes que +

$44 / 4 - 5$

$11 - 5$   
/ antes que -

$6$

👁️👁️ Pon espacio antes y después de cada operador binario



# Una fórmula

60

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

```
int main()
{
    double x, f;
    cout << "Introduce el valor de X: ";
    cin >> x;
    f = 4 * x * x / 6 + 9 * x / 5 - 8;
    cout << "f(x) = " << f << endl;
    return 0;
}
```

$$f(x) = \frac{4x^2}{6} + \frac{9x}{5} - 8$$




Usa paréntesis para mejorar la legibilidad:

```
f = (4 * x * x / 6) + (9 * x / 5) - 8;
```



# Modificadores de tipos de datos

61

- Los tipos de datos **int**, **double** y **char** tienen variaciones o **modificadores** de tipos de datos, permiten un uso más eficiente de los tipos de datos.
- Son modificadores los siguientes: de los tipos de datos **signed - unsigned – short - long**
- Rango de valores

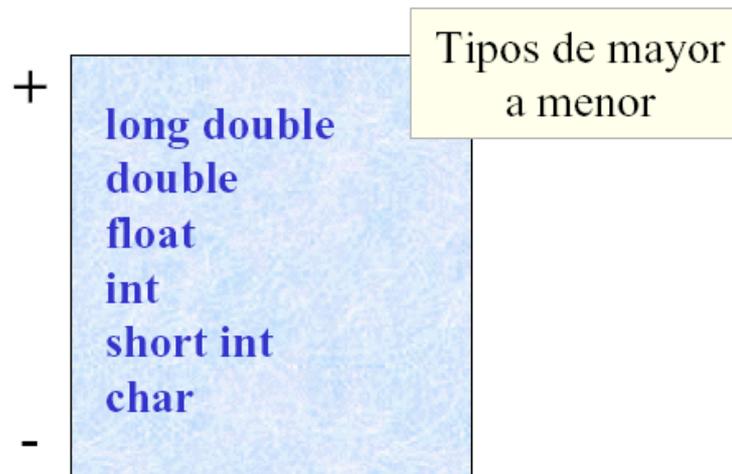
<b>unsigned int</b>	0 ... 65625
<b>long double</b>	$-3,37 \times 10^{-4932} \dots 3,37 \times 10^{4932}$
<b>long int</b>	-2147483648 ... 2147483647



# Modificadores de tipos de datos

62

- Los ordenadores realizan numerosas operaciones para la resolución de problemas,
  - Operaciones aritméticas y lógicas.
  - Operaciones condicionales.
  - ...
- Además, puede ocurrir que en una misma expresión concurren varios tipos de datos. Ante ésta situación, debemos saber cómo se comporta el compilador.
- **Cuando los dos operandos son de tipos distintos, el de tipo *menor* se promociona al de tipo *mayor*.**



# Memoria que ocupan los tipos de datos

63

Tipo de datos	Datos almacenados	Nº de Bytes
char	caracteres	1
int	enteros	2
float	reales	4
double	reales	8
bool	lógicos	1

Esto no siempre es cierto, depende del ordenador y del compilador.

```
#include <stdio.h>
#include <iostream.h>
```

```
int main()
```

```
{
```

```
    int i;
```

```
    i = sizeof( int ) * 8;
```

```
    cout << "Tamaño (en bits) del tipo int = ";
```

```
    cout << i;
```

```
    return 0;
```

```
}
```

Memoria que ocupan los tipos de datos



# Conversión de tipos

64

- Cuando se opera con datos de diferente tipo,
  - ▣ ambos son convertidos al tipo de dato de precisión más alta
  - ▣ El resultado es convertido al tipo de datos que lo almacena

## Por ejemplo:

```
int x, z; double y;
```

```
z=x+y;
```

1. convierte x en double
2. Se realiza la suma entre doubles
3. Se guardará la parte entera del resultado en z

*No es conveniente dejar en manos del compilador la conversión de tipos, el programador puede forzar a conversión*

p.e.      (int) y

**(nuevoTipoDeDatos) variableAconvertir**



# Moldes (*casts*)

65

Fuerzan una conversión de tipo:

*tipo*(*expresión*)

El valor resultante de la *expresión* se trata como un valor del *tipo*

```
int a = 3, b = 2;  
cout << a / b; // Muestra 1 (división entera)  
cout << double(a) / b; // Muestra 1.5 (división real)
```



# Variables, constantes y expresiones



# Constantes

67

- Una **constante** es un objeto cuyo valor no puede ser modificado
- Los nombres de las constantes se suelen escribir en mayúsculas
- Las constantes pueden aparecer como
  - *constantes literales*, son aquellas cuyo valor aparece directamente en el programa,
  - *constantes simbólicas*, aquellas cuyo valor se asocia a un identificador, a través del cual se representa
- **Constantes lógicas (bool):**  
false, true
- **Constantes carácter (char),** el símbolo constante aparece entre comillas simples:  
'a', 'b', ..., 'z',  
'A', 'B', ..., 'Z',  
'0', '1', ..., '9',  
' ', '.', ',', ':', ...



# Constantes

68

```
const <tipo_de_dato> <nombre_de_constante> = <valor>;
```

```
...  
const int DIAS = 7;  
const char VACIO = ' ' ;  
const char PORCENTAJE = '% ' ;
```

Ejemplo

- **Constantes cadenas de caracteres literales**, la secuencia de caracteres aparece entre comillas dobles

"Hola Pepe"

"Hola\nJuan\n"

"Hola " "Mara"



# Constantes

69

## Constantes definidas

- Se declaran mediante la directiva **#define**

```
#define <nombre_de_constante> <valor>
```

Ejemplos

```
...  
#define pi 3.14  
#define fin 'F'  
...
```

No se especifica el tipo de dato

No aparece ; al final de la sentencia

No aparece el símbolo =

Es más recomendable utilizar **const** en lugar de **#define** ya que el compilador genera código más eficiente.



# Constantes

70

- **Constantes enteras**, pueden ser expresadas en
  - decimal (base 10)
  - hexadecimal (base 16)
  - octal (base 8).
  - El sufijo **L** se utiliza para especificar **long**, el sufijo **LL** se utiliza para especificar **long long**, el sufijo **U** se utiliza para especificar **unsigned**, el sufijo **UL** especifica **unsigned long**, y el sufijo **ULL** especifica **unsigned long long**:  
123, -1520, 2345U, 30000L, 50000UL, 0x10B3FC23 (hexadecimal), 0751 (octal)
  
- **Constantes reales**, números en punto flotante. El sufijo **F** especifica float, y el sufijo **L** especifica long double:  
3.1415, -1e12, 5.3456e-5, 2.54e-1F, 3.25e200L



# Constantes

71

- Las **constantes simbólicas** se declaran indicando la palabra reservada **const** seguida por su tipo, el nombre simbólico (o identificador) con el que nos referiremos a ella y el valor asociado tras el símbolo (**=**).

## Ejemplos de constantes simbólicas:

```
const bool OK = true;
const char SONIDO = '\a';
const short ELEMENTO = 1000;
const int MAXIMO = 5000;
const long ULTIMO = 100000L;
const long long TAMANO = 1000000LL;
const unsigned short VALOR = 100U;
const unsigned FILAS = 200U;
const unsigned long COLUMNAS = 200UL;
const unsigned long long NELMS = 2000ULL;
const float N_E = 2.7182F;
const double LOG10E = log(N_E);
const long double N_PI = 3.141592L;
const Color COLOR_DEFECTO = ROJO;
```



# Las variables

72

- Una variable es un espacio reservado en la memoria del ordenador para contener valores que pueden cambiar durante la ejecución de un programa
- Los tipos determinan cómo se manipulará la información contenida en esas variables
- El tipo nos dice a nosotros y al compilador cómo debe interpretarse y manipularse la información binaria almacenada en la memoria de un ordenador
- Una declaración de variables es una instrucción, puede aparecer entremezclada con otras instrucciones
- La variable declarada existe inmediatamente después de la declaración hasta el momento en que se acabe el bloque
- Recién declarada tiene un valor indefinido, es recomendable definirla lo más próximo al lugar dónde se va a utilizar y darle un valor inicial
- Se suele escribir en minúscula



# Variables

73

Toda variable utilizada en un programa debe ser declarada previamente. En C++, ésta declaración puede situarse en cualquier parte del programa.

Dependiendo de dónde se definan,  
tenemos varios tipos:

**Variables globales**  
**Variables locales**  
**Parámetros**

Declarar una variable

`<tipo_de_dato> <nombre_de_variable> ;`

`<tipo_de_dato> <lista de variables> ;`

`<tipo_de_dato> <nombre_de_variable> = valor;`

`int x;`

`char x, y, z;`

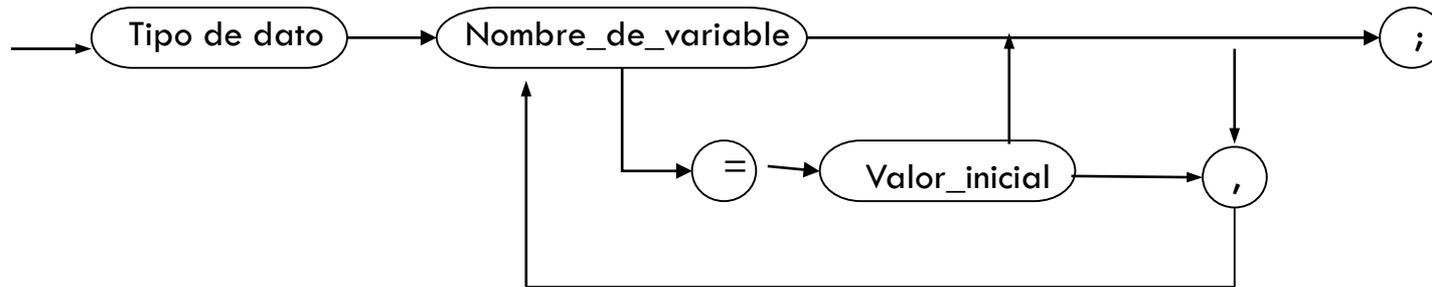
`long int i = 10, j, k = 0;`

**En C++ las variables no se  
actualizan automáticamente**



# Variables

74



## ***Declaraciones locales***

Son *variables locales* aquellas que están declaradas dentro de las funciones o de los bloques.

## ***Declaraciones globales ( variables globales, funciones, ...)***

La zona de declaraciones globales de un programa puede incluir declaraciones de variables y declaraciones de funciones (*prototipos*).



# Ejemplos

75

## Constantes

```
// -- Constantes -----  
const bool DEPURACION = true;  
const char LETRA = 'a';  
const unsigned KBYTE = 1024;  
const int ESCALA = -1;  
const double ERROR_PRECISION = 1.56E-7;  
// -- Principal -----  
int main ()  
{  
    // Acciones  
}
```

## Variables

```
// -- Principal -----  
int main ()  
{  
    bool logico = false;  
    char caracter = 'z';  
    unsigned natural_1, natural_2;  
    int entero = 56;  
    double real;  
    // Acciones  
}
```



# Operadores y expresiones

Instrucción de asignación

Operador de dirección

Referencias



# Instrucciones de asignación

`<nombre_de_variable> = <expresión> ;`

Puede ser otra variable, una constante o una operación entre variables y constantes.

- El **operador asignación (=)** asigna el valor de la expresión derecha a la variable situada en la izquierda de la instrucción.
- Podemos tener en C++ varios operadores de asignación:

`=      +=      -=      *=      /=      %=`



# Instrucciones de asignación

## Ejemplos:

<code>m = n;</code>	<code>// asigna el valor de n a m</code>
<code>m += n;</code>	<code>m = m + n; // suma m y n y lo asigna a la variable m</code>
<code>m -= n;</code>	<code>m = m - n; // resta m menos n y lo asigna a la variable m</code>
<code>m *= n;</code>	<code>m = m * n; // multiplica m por n y lo asigna a la variable m</code>
<code>m /= n;</code>	<code>m = m / n; // divide m entre n y lo asigna a la variable m</code>
<code>m %= n;</code>	<code>m = m % n; // calcula el resto de la div. entera y lo asigna a la variable m</code>

Instrucción abreviada.



# Instrucciones de asignación

## Más ejemplos:

Podemos dar valores a varias variables a la vez

```
m = n = t = 10;    // Damos a las variables m, n y t el valor 10  
m = n = t = a;    // las variables m, n y t toman el valor de la variable a
```

También podemos asignar a varias variables el valor de otra de un sólo golpe



# Ejemplos

80

```
// -- Principal -----  
int main ()  
{  
    int x1 = 4 + 2 * 6; // Inicializac  
    x1 = x1 * 3 % 2 - 3; // Asignación  
    ++x1;           // x1 = x1 + 1;  
    --x1;           // x1 = x1 - 1;  
    x1 += 5;        // x1 = x1 + 5;  
    x1 -= 3;        // x1 = x1 - 3;  
    x1 *= 2 + 5;    // x1 = x1 * (2 + 5);  
    x1 /= 2;        // x1 = x1 / 2;  
    x1 %= 2;        // x1 = x1 % 2;  
}
```

```
#include <iostream>  
#include <string>  
using namespace std;  
// -- Principal -----  
int main ()  
{  
    char letra;  
    cin >> letra;  
    int valor = int(letra) + 1;  
    letra = char(valor);  
    letra = char(letra + 1);  
    cout << letra << endl;  
}
```



# Instrucciones de Entrada / Salida



# Instrucciones de Entrada / Salida

- En C++ la entrada y salida se lee y escribe **en flujos**. Cuando se incluye la biblioteca **iostream** en el programa, se definen automáticamente dos flujos:

Flujo **cin** (se utiliza para la entrada de datos)

Flujo **cout** (se utiliza para la salida de datos)

Permiten la comunicación del ordenador con el exterior para tomar datos o devolver resultados

- Esta biblioteca también nos proporciona dos operadores, uno de *inserción* (`<<`), que inserta datos en el flujo **cout** y otro operador de extracción (`>>`) para extraer valores del flujo **cin** y almacenarlos en variables.

```
...  
cin >> a;  
cin >> a >> b >> c;  
...
```

```
...  
cout << x;  
cout << x << y << z << endl;  
cout << " x vale: " << x;  
cout << "Hola\n";  
...
```

Salto de línea

\n también provoca salto de línea



Para evitar errores, lee cada dato en una instrucción aparte

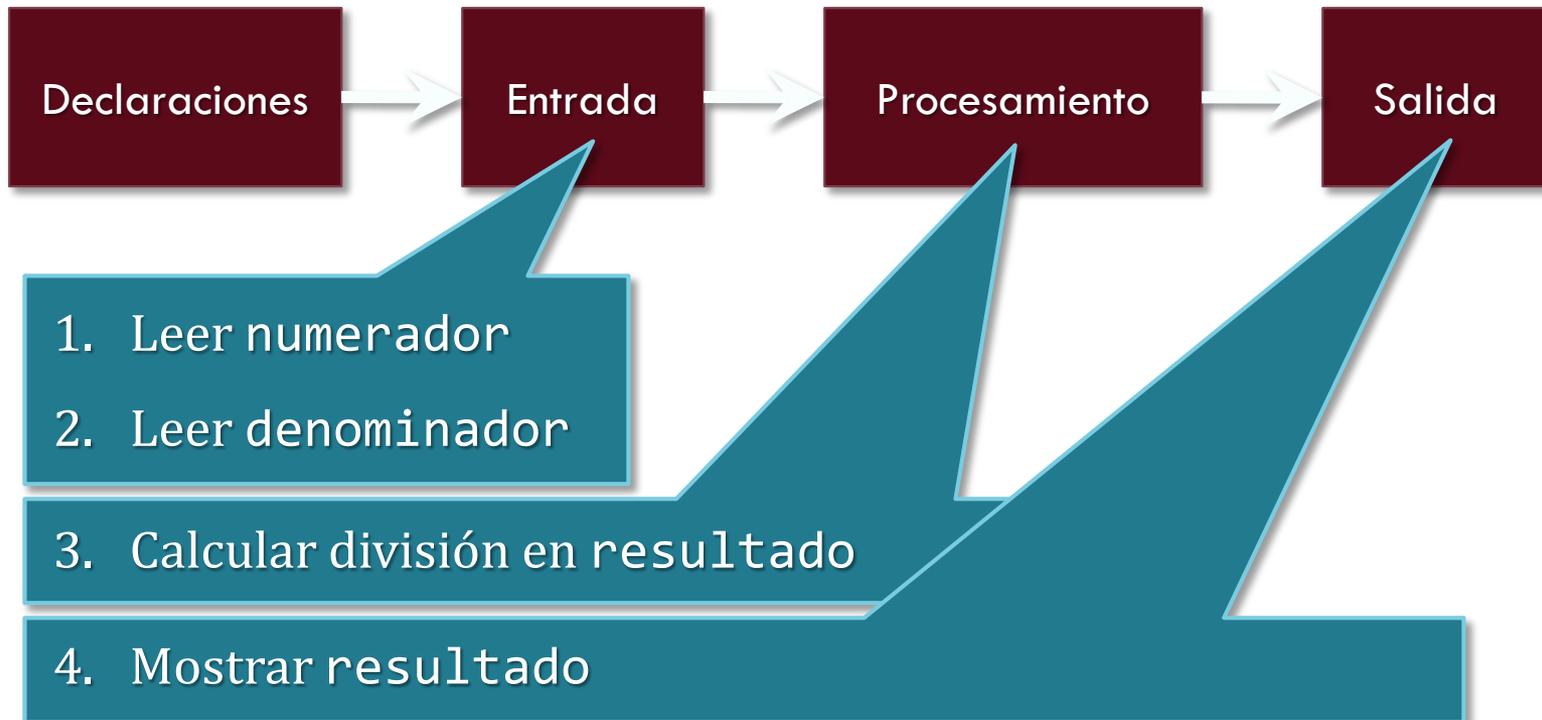


# Dividir dos números

83

## Entrada-Proceso-Salida

- Muchos programas se ajustan a un sencillo esquema:



# Dividir dos enteros

84

```
//Sumar dos numeros enteros
#include<iostream>
using namespace std;
int main(){
    int num1,num2,division;
    cout << "Introduce un número entero ";
    cin >> num1;
    cout << endl<<"Introduce un numero entero ";
    cin >> num2;
    producto = num1/num2;
    cout << "\n El producto de "<<num1<<" x "<<num2<<"
        es"<<suma<<endl;
    system("pause");
    return 0;
}
```



# Entrada de datos

85

- **getchar()** espera hasta pulsar un carácter
- **kbhit()** espera hasta pulsar una tecla
- **Getline(cin, cadena)** Lee todo hasta el siguiente carácter de nueva línea y coloca el resultado en la variable de cadena especificado. Lee líneas de texto que incluyen espacios en blanco

```
#include<iostream>
#include<string>
using namespace std;
int main() {
    string frase;
    cout << "Introduce una frase: \n";
    getline(cin,frase);
    cout << "Tu frase es: " << frase;
    getchar();
    return 0;
}
```



# Salida formateada

86

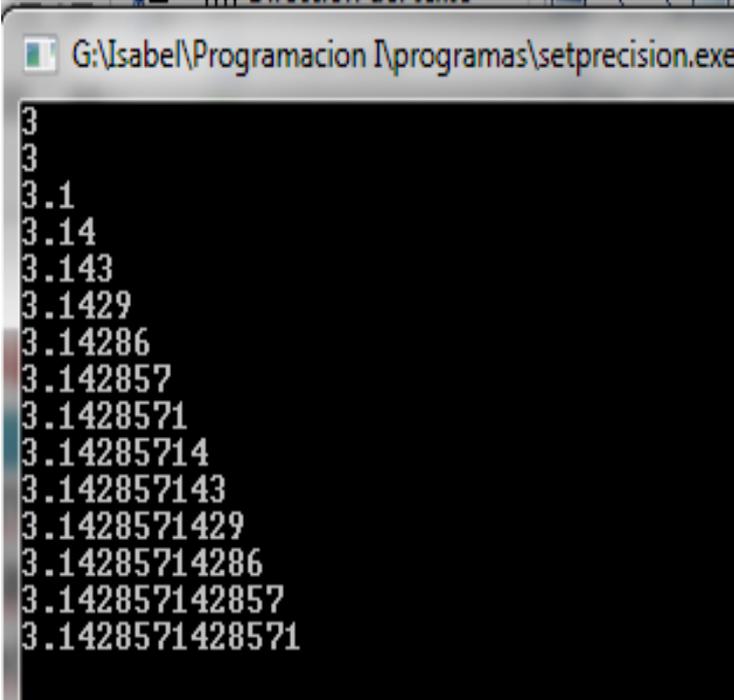
- **setprecision()**. Para indicar el número de dígitos significativos en un dato en punto flotante. Afecta a todos los datos que se introduzcan con posterioridad.
  
- **setw()**. Permite indicar el número de espacios que se emplearán para escribir un dato, alineando al mismo a la derecha dentro de dicho espacio.
  - Si el espacio requerido es mayor que el indicado, el modificador se ignora. Sólo afecta al dato que se indica a continuación.
  
- **setfill()**: Para especificar el carácter de relleno que se empleará para los espacios no usados al escribir un dato (según un modificador setw()).



# Ejemplo setprecision

87

```
#include <iostream>    // para operaciones de E/S
#include <iomanip>      // define diferentes manipuladores
using namespace std;
int main()
{
    int i;
    double variable1=22, variable2=7;
    for(i=0; i<15; i++)
    {
        cout << setprecision(i);
        cout << variable1/variable2 << endl;
    }
    return 0;
}
```



```
G:\Isabel\Programacion I\programas\setprecision.exe
3
3
3.1
3.14
3.143
3.1429
3.14286
3.142857
3.1428571
3.14285714
3.142857143
3.1428571429
3.14285714286
3.142857142857
3.1428571428571
```



# Ejemplo setw y setfill

88

```
#include <iostream>           // para operaciones de E/S
#include <iomanip>             // define diferentes manipuladores
using namespace std;
int main()
{
    cout << setfill('.');      // rellenar con puntos
    cout << "Lista de notas\n" << endl;
    cout << "Julio Iglesias" << setw(20) << "5" << endl;
    cout << "Shakira Pérez" << setw(21) << "8" << endl;
    cout << "Alejandro Sanz" << setw(20) << "7" << endl;
    cout << "Jarabe de Palo" << setw(20) << "5" << endl;
    cout << "Jhon Travolta" << setw(21) << "8" << endl;
    cout << "Enrique Bunbuny" << setw(19) << "9" << endl;
    cout << setfill('\0');     // se restablece el carácter de llenado
    cout << setw(20) << "Fin" << endl;

    return 0;
}
```



# Ejemplo setw y setfill

89

```
G:\Isabel\Programacion I\programas\manipuladores2.exe
Lista de notas
Julio Iglesias.....5
Shakira Púrez.....8
Alejandro Sanz.....7
Jarabe de Palo.....5
Jhon Travolta.....8
Enrique Bunbuny.....9
                Fin

-----
Process exited with return value 0
Press any key to continue . . .
```



# Salida formateada

90

## SALIDA FORMATEADA DE DATOS

formato aplicado a la salida de datos. incluir la biblioteca estándar **iomanip**

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    cout << boolalpha; // escribe los valores booleanos como 'false' o 'true'
    cout << dec << 27; // escribe 27 (decimal)
    cout << hex << 27; // escribe 1b (hexadecimal)
    cout << oct << 27; // escribe 33 (octal)
    cout << setprecision(2) << 4.567; // escribe 4.6
    cout << setw(5) << 234; // escribe  234
    cout << setfill('#') << setw(5) << 234; // escribe ##234
}
```



# Instrucciones de Entrada / Salida

- C++ utiliza **secuencias de escape** para visualizar caracteres que no están representados por los símbolos tradicionales.
- Las más utilizadas las mostramos en la siguiente tabla:

<code>\n</code>	Retorno de carro y avance de línea
<code>\t</code>	Tabulación
<code>\a</code>	Alarma
<code>\"</code>	Dobles comillas
<code>\\</code>	Barra inclinada

```
...  
cout << "Hola\n";  
cout << "Lunes\t Martes\t Miercoles\t ";  
cout << "\a" ;  
...
```



# Generar números aleatorios

92

- Activar el generador de números aleatorios  
**srand(time(NULL));**

```
variable = limiteInferior + rand() % (limiteSuperior + 1 -  
limiteInferior) ;
```

## Ejemplos

- Número aleatorios entre **0 y 50**:  
`num=rand()%51;`
- Número aleatorios entre **1 y 100**:  
`num=1+rand()%(101-1);`
- Número aleatorios entre **250 y 420**:  
`num=250+rand()%(421-250);`



### Código Fuente: Pitagoras.cpp

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

int main() {
    double lado1;
    .....
    cout << "Introduzca la longitud ..."
    cin >> lado1;
    .....
}
```



**Compilador**



### Programa Ejecutable: Pitagoras.exe

```
10011000010000
10010000111101
00110100000001
11110001011110
11100001111100
11100101011000
00001101000111
00011000111100
```



# Errores en un programa

94

- ❑ **Errores en tiempo de compilación.** Ocasionados por un fallo de sintaxis en el código fuente. No se genera el programa ejecutable.

```
/* CONTIENE ERRORES */  
#include <iostream>  
USING namespace std;  
int main{ }(  
    double lado1:  
    double lado 2,  
    double hip:  
    lado1 = 2;  
    lado2 = 3  
    hip = sqrt(lado1**lado1 + ladv2*ladp2);  
    cout >> "La hipotenusa vale >> hip;  
)
```



# Errores en un programa

95

- ❑ **Errores en tiempo de ejecución.** Se ha generado el programa ejecutable, pero se produce un error durante la ejecución.

```
int datoEntero;  
int otraVariable;  
datoEntero = 0;  
otraVariable = 7 / datoEntero;  
.....
```



# Errores en un programa

96

- ❑ **Errores lógicos.** Se ha generado el programa ejecutable, pero el programa ofrece una solución equivocada.

.....

```
lado1 = 4;
```

```
lado2 = 9;
```

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

.....



# Biblioteca ctype (características del tipo char)

97

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

---

<code>bool isalnum(char ch);</code>	<code>(isalpha(ch)    isdigit(ch))</code>
<code>bool isalpha(char ch);</code>	<code>(isupper(ch)    islower(ch))</code>
<code>bool iscntrl(char ch);</code>	caracteres de control
<code>bool isdigit(char ch);</code>	dígito decimal
<code>bool isgraph(char ch);</code>	caracteres imprimibles excepto espacio
<code>bool islower(char ch);</code>	letra minúscula
<code>bool isprint(char ch);</code>	caracteres imprimibles incluyendo espacio
<code>bool ispunct(char ch);</code>	carac. impr. excepto espacio, letra o dígito
<code>bool isspace(char ch);</code>	espacio, <code>'\r'</code> , <code>'\n'</code> , <code>'\t'</code> , <code>'\v'</code> , <code>'\f'</code>
<code>bool isupper(char ch);</code>	letra mayúscula
<code>bool isxdigit(char ch);</code>	dígito hexadecimal

---

<code>char tolower(char ch);</code>	retorna la letra minúscula correspondiente a ch
<code>char toupper(char ch);</code>	retorna la letra mayúscula correspondiente a ch

---



# Biblioteca cstdlib

98

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

<code>int system(const char orden[]);</code>	orden a ejecutar por el sistema operativo
<code>int abs(int n);</code>	retorna el valor absoluto del número <code>int n</code>
<code>long labs(long n);</code>	retorna el valor absoluto del número <code>long n</code>
<code>void srand(unsigned semilla);</code>	inicializa el generador de números aleatorios
<code>int rand();</code>	retorna un aleatorio entre 0 y <code>RAND_MAX</code> (ambos inclusive)

```
#include <cstdlib>
#include <ctime>
using namespace std;
// -----
// inicializa el generador de números aleatorios
inline unsigned ini_aleatorio()
{
    srand(time(0));
}
// -----
// Devuelve un número aleatorio entre 0 y max (exclusive)
inline unsigned aleatorio(unsigned max)
{
    return unsigned(max*double(rand())/(RAND_MAX+1.0));
}
// -----
// Devuelve un número aleatorio entre min y max (ambos inclusive)
inline unsigned aleatorio(unsigned min, unsigned max)
{
    return min + aleatorio(max-min+1);
}
// -----
```



# Bibliografía y enlaces

99

Eckel, B., Thinking in C++. 2ª Edición. Prentice-Hall. 2000.

Disponible en versión electrónica en <http://www.bruceeckel.com/>

<http://www.tecnun.es/asignaturas/Informat1/AyudaInf/Index.htm#lenguajes>

C++ con clase. <http://c.conclase.net>

C Plus Plus (en inglés) <http://www.cplusplus.com>

C++ Reference (en inglés) <http://www.cppreference.com>

Zator (libro programación) <http://www.zator.com/Cpp/>



# Datos de una persona

100

```
#include <iostream>
using namespace std;
int main() {
    char Nombre[30];
    int Edad;
    char Telefono[9];
    cout <<"Introduce tu nombre, edad y número de teléfono"<<endl;
    cin >> Nombre >> Edad >> Telefono;
    cout << "Nombre:" << Nombre << endl;
    cout << "Edad:" << Edad << endl;
    cout << "Teléfono:" << Telefono << endl;
    return 0;
}
```



# Ejemplos

101

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int x1, x2;
    cout << "Introduzca dos numeros: ";
    cin >> x1 >> x2;
    cout << "Valor: " << x1 << ' ' << (x2 * 3) << endl;
    cout << "FIN" << endl;
    char c;
    cin.get(c);
}
```



# Error sintáctico

102

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int dividendo, divisor;
    int cociente;
    cin >> dividendo >> divisor;
    cociente = / dividendo divisor;
    cout << cociente << endl;
}
```

ERROR SINTÁCTICO



# Error en tiempo de ejecución

103

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int dividendo, divisor;
    int cociente;
    cin >> dividendo >> divisor;
    cociente = dividendo / divisor;
    cout << cociente << endl;
}
```

RT\_ERROR: DIVISIÓN POR CERO



# Error semántico

104

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int dividendo, divisor;
    const int cociente = 3;
    cin >> dividendo >> divisor;
    cociente = dividendo / divisor;
    cout << cociente << endl;
}
```

ERROR SEMÁNTICO



# Expresiones lógicas

105

```
int main ()
{
    int num;
    cin >> num;
    bool error = false;
    bool par = (num % 2) == 0;
    bool tres_digitos = (num >= 100) && (num <= 999);
    bool tres_digitos_par = tres_digitos && par;
    bool tres_digitos_par = ((num >= 100) && (num <= 999)) && ((num % 2) == 0);
    bool primo_10 = ((num >= 2) && (num <= 3)) || (num == 5) || (num == 7);
    bool divisor_10 = (num > 0) && ((10 % num) == 0);
}
```



# Pasar una cantidad de pesetas a euros

106

```
//- fichero: euros.cpp -----  
#include <iostream>  
#include <cstdlib>  
using namespace std;  
const double EUR_PTS = 166.386;  
int main()  
{  
    cout << "Introduce la cantidad (en euros): ";  
    double euros;  
    cin >> euros;  
    double pesetas = euros * EUR_PTS;  
    cout << euros << " Euros equivalen a " << pesetas << " Pts" << endl;  
    system("pause");  
    return 0;  
}  
//- fin: euros.cpp -----
```



# Decidir si un entero dado es o no par

107

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int numero;
    cin >> numero;
    int resto = numero % 2;
    bool par = (resto == 0);
    // par = ((numero % 2) == 0);
    cout << numero << " es par: " << boolalpha << par << endl;
}
```



# Dada una letra mayúscula convertirla a minúscula

108

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    char letra_mayuscula;
    cin >> letra_mayuscula;
    // Suponemos que es una letra Mayúscula
    int distancia = int(letra_mayuscula) - int('A');
    char letra_minuscula = char(int('a') + distancia);
    // letra_minuscula = char(int('a') + (int(letra_mayuscula) - int('A')));
    // letra_minuscula = char('a' + (letra_mayuscula - 'A'));
    cout << letra_mayuscula << " -> " << letra_minuscula << endl;
}
```



# Ejercicio

109

Escribe un programa que lea de teclado dos números enteros ( $x$  e  $y$ ) y un carácter ( $c$ ), y escriba true si cumplen las siguientes propiedades, y false en caso contrario:

- a)  $x \in \{3; 4; 5; 6; 7\}$
- b)  $X \in \{1; 2; 3; 7; 8; 9\}$
- c)  $X \in \{1; 3; 5; 7; 9\}$
- d)  $X \in \{2; 5; 6; 7; 8; 9\}$
- e)  $X \in \{3; 4; 6; 8; 9\}$ ,  $y \in \{6; 7; 8; 3\}$
- f) Ni  $x$  ni  $y$  sean mayores que 10
- g)  $x$  no sea múltiplo de  $y$
- h)  $c$  es una letra mayúscula
- i)  $c$  es una letra
- j)  $c$  es un alfanumérico (letra o dígito)



```

# include < iostream >
using namespace std ;
int main ()
{
int x, y;
char c;
cout << " Introduzca dos numeros naturales : ";
cin >> x >> y;
cout << " Introduzca un caracter : ";
cin >> c;
bool prop_a = (x >= 3 && x <= 7);
bool prop_b = (x >= 1 && x <= 3) || (x >= 7 && x <= 9);
bool prop_c = (x >= 1 && x <= 9) && (x % 2 == 1);
bool prop_d = (x == 2) || (x >= 5 && x <= 9);
bool prop_e = ((x >= 3 && x <= 9 && x != 5 && x != 7) && ((y >= 6 && y <= 8) || y == 3));
bool prop_f = (x <= 10 && y <= 10);
bool prop_g = ! (y != 0 && x % y == 0);
bool prop_h = (c >= 'A' && c <= 'Z');
bool prop_i = (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
bool prop_j = ((c >= 'A' && c <= 'Z')
|| (c >= 'a' && c <= 'z')
|| (c >= '0' && c <= '9'));
cout << boolalpha ;
cout << "(a) " << x << " pertenece a { 3, 4, 5, 6, 7 } : " << prop_a << endl ;
cout << "(b) " << x << " pertenece a { 1, 2, 3, 7, 8, 9 } : " << prop_b << endl ;
cout << "(c) " << x << " pertenece a { 1, 3, 5, 7, 9 } : " << prop_c << endl ;
cout << "(d) " << x << " pertenece a { 2, 5, 6, 7, 8, 9 } : " << prop_d << endl ;
cout << "(e) " << x << " pertenece a { 3, 4, 6, 8, 9 }, "
<< y << " pertenece a { 6, 7, 8, 3 } : " << prop_e << endl ;
cout << "(f) Ni " << x << " ni " << y << " sean mayores que 10: " << prop_f << endl ;
cout << "(g) " << x << " no sea multiplo de " << y << " : " << prop_g << endl ;
cout << "(h) " << c << " es una letra mayuscula : " << prop_h << endl ;
cout << "(i) " << c << " es una letra : " << prop_i << endl ;
cout << "(j) " << c << " es un alfanumerico : " << prop_j << endl ;

```

```
#include <iostream.h>
#include <stdlib.h>
Using namespace std;
int main()
{
    int a, b;

    cout << endl << "Introduce el primer numero: " << endl;
    cin >> a;
    cout << endl << "\nIntroduce el segundo numero: " << endl;
    cin >> b;

    cout << endl << "La suma es:" << a+b;
    cout << endl << "La resta es: " << a-b;
    cout << endl << "La multiplicacion es: " << a*b;
    cout << endl << "La division entera es: " << a/b;
    cout << endl << "El resto de la division entera es: " << a%b << endl;

    system("PAUSE");
    return 0;
}
```

```

#include <iostream.h>
#include <stdlib.h>
Using namespace std;
int main()
{
// Otra manera de definir constantes
// CONST float PI=3.1415;

float altura, base, radio;

cout << endl << "Introduzca la base del triangulo " << endl;
cin >> base;
cout << endl << "Introduzca la altura del triangulo " << endl;
cin >> altura;
cout << endl << "El area del triangulo es " << base*altura/2;

cout << endl << "Introduzca el radio del circulo " << endl;
cin >> radio;
cout << endl << "El area del circulo es " << PI*radio*radio << endl;

system("PAUSE");
return 0;
}

```

# Comprobar si un número de tres cifras es capicúa

113

```
#include <iostream>
#include <string>
using namespace std;
// -- Principal -----
int main ()
{
    int numero;
    cin >> numero;
    int digito_1 = numero % 10;
    int digito_3 = numero / 100;
    bool tres_cap = (numero >= 100) && (numero <= 999) && (digito_1 == digito_3);
    cout << numero << " tiene 3 dígitos y es capicua: " << boolalpha << tres_cap << endl;
}
```

