

2

Tipos e instrucciones

Doble Grado en Matemáticas e informática

Alberto de la Encina
(adaptadas del original de Luis Hernández Yáñez)



Facultad de Informática
Universidad Complutense



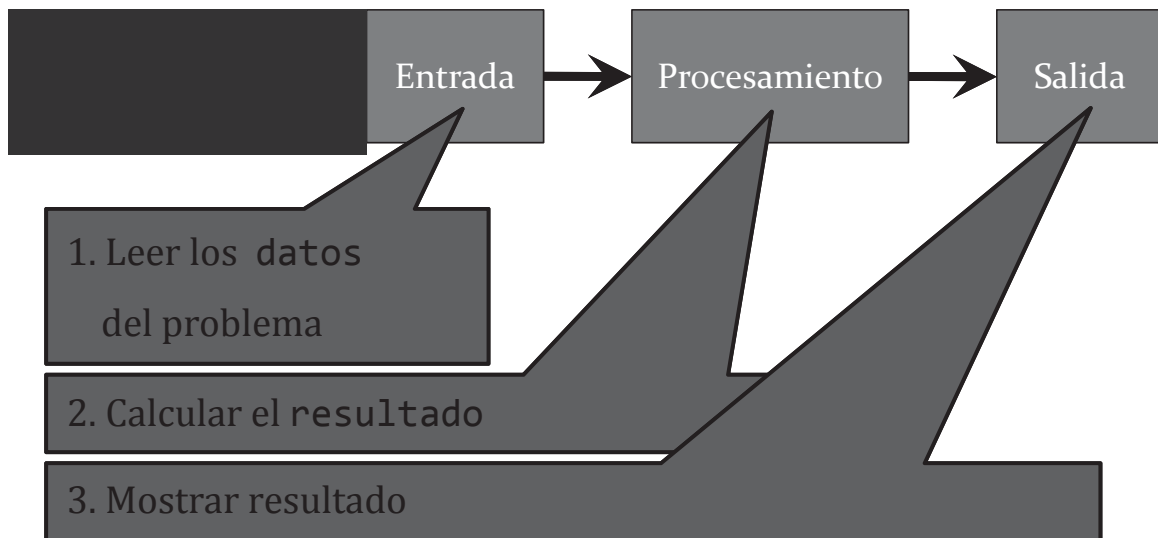
Índice

Un esquema inicial	2	Operadores relacionales	82
Dispositivo de salida	15	Operadores lógicos	84
Literales	18	Instrucción condicional (if)	90
Variables	20	Bloques de código ({...})	92
Lectura de datos	24	Instrucción iterativa (while)	95
Asignación y expresiones	30	Entrada/salida por consola	98
Los datos de los programas	40	Funciones definidas	
Identificadores	41	por el programador	111
Tipos de datos	44		
Definición de variables	53		
Instrucción de asignación	58		
Operadores	62		
Más sobre expresiones	68		
Constantes	74		
La biblioteca cmath	77		
Operadores con caracteres	80		



Un esquema inicial

Muchos programas se ajustan a un sencillo esquema



Un esquema inicial

Programa con E/S por consola.

Los datos se manejan mediante variables.

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

Biblioteca para E/S por consola

```
int main()
```

```
{
```

```
// Definición de variables
// Solicitar y leer los datos
// Calcular el resultado
// Mostrar el resultado
```

← ¡Tu código aquí!

```
return 0;
```

```
}
```



Variables del problema

Problema

Mostrar en la pantalla un mensaje que pida la base del triángulo. El usuario introducirá la base con el teclado. Mostrar en la pantalla un mensaje que pida la altura del triángulo. El usuario introducirá la altura con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla.

Datos del problema -> variables del programa

- pantalla (cout consola o flujo de salida)
- mensaje que pida la base del triángulo (texto)
- base (real)
- teclado (cin consola o flujo de entrada)
- altura (real)
- área del triángulo (real)

El usuario no es un dato, es quien maneja el programa.
cout y cin están definidas en la biblioteca iostream



Acciones del problema

Problema

Mostrar en la pantalla un texto que pida la base del triángulo. El usuario introducirá la base con el teclado. Mostrar en la pantalla un texto que pida la altura del triángulo. El usuario introducirá la altura con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla..

Acciones del problema -> instrucciones del programa

- mostrar en la pantalla: se envía algo a la pantalla con el insertor <<.
`cout << ...`
- leer del el teclado: se lee algo del teclado con el extractor >>.
`cin >> ...`
- calcular el área del triángulo: expresión aritmética.
`area = base * altura / 2`

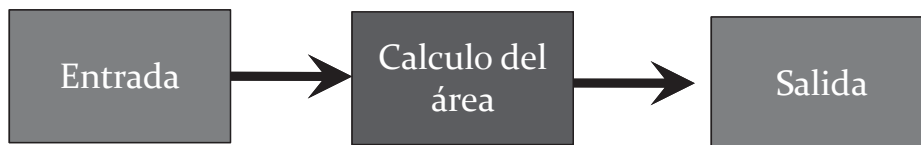
introducir no es una acción del programa, el programa lee.



Programa

Secuencia de instrucciones:

- 1 - definir las variables (nombres y tipos de datos)
base, altura (y area) para datos de tipo real.
- 2 y 3 - solicitar y leer la base (operadores cout << y cin >>)
- 4 y 5 - solicitar y leer la altura (operadores cout << y cin >>)
- 6 - calcular el área (expresiones aritméticas)
- 7 - mostrar el área (operador cout <<)



Programa

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

int main()
{
    float base, altura, area;
    cout << "Introduzca la base del triángulo: ";
    cin >> base;
    cout << "Introduzca la altura del triángulo: ";
    cin >> altura;

    area = base * altura / 2;

    cout << "El área de un triángulo de base " << base << " y altura "
         << altura << " es: " << area << endl;

    return 0;
}
```

Las instrucciones terminan en ;

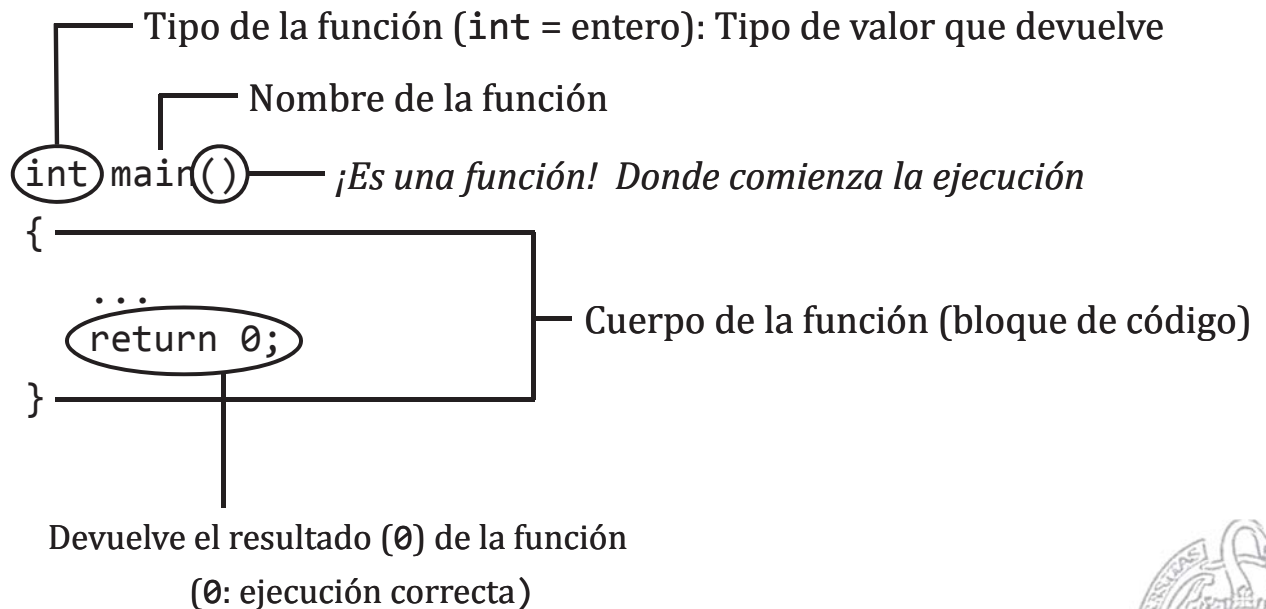
// 1. Variables..
// 2. Mostrar ...
// 3. Leer del...
// 4. Mostrar ...
// 5. Leer del...
// 6. Calcular...
// 7. Mostrar...



El programa principal

La función **main()**

contiene las instrucciones que hay que ejecutar



Las bibliotecas

C++ nos proporciona mucho código para reutilizar:

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

```
int main()    // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

Bibliotecas de funciones a nuestra disposición (STL).
Al incluirlas podemos usar lo que contienen.



Las bibliotecas

Inclusión

Se incluyen con la *directiva* `#include`:

`#include <iostream>` ← Es una directiva: empieza por #
(Utilidades de entrada/salida por consola)

Para mostrar o leer datos hay que incluir la biblioteca `iostream`

Espacios de nombres

Cualifican los nombres con un prefijo (`std::cout`).

En `iostream` hay espacios de nombres; podemos pedir que se cualifiquen automáticamente con la instrucción

`using namespace std;` ← Es una instrucción: termina en ;

Siempre usaremos el espacio de nombres estándar (`std`)

Muchas bibliotecas no tienen espacios de nombres



Documenta el código

Comentarios (se ignoran):

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

```
int main()    // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

Hasta el final de la línea: `// Comentario hasta fin línea`

De varias líneas: `/* Comentario de varias
 líneas seguidas */`



Indentación

Uso de espacios en blanco

Los elementos del lenguaje se separan por uno o más *espacios en blanco* (espacios, tabuladores, saltos de línea, ...).

Usa sangría (indentación) para el código en los bloques (2 espacios).

El compilador los ignora.

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

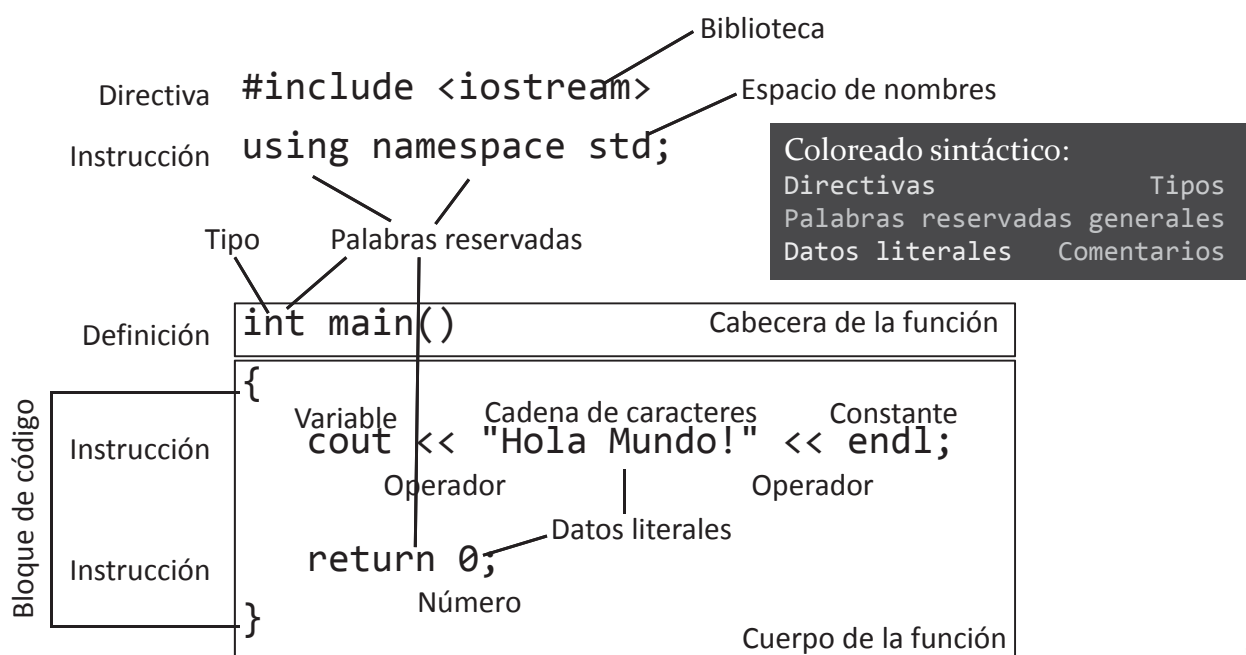
```
int main()
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

```
#include <iostream> using namespace std;
int main(){cout<<"Hola Mundo!"<<endl;
return 0;}
```

¿Cuál se lee mejor?
¡El estilo importa!



Elementos del programa



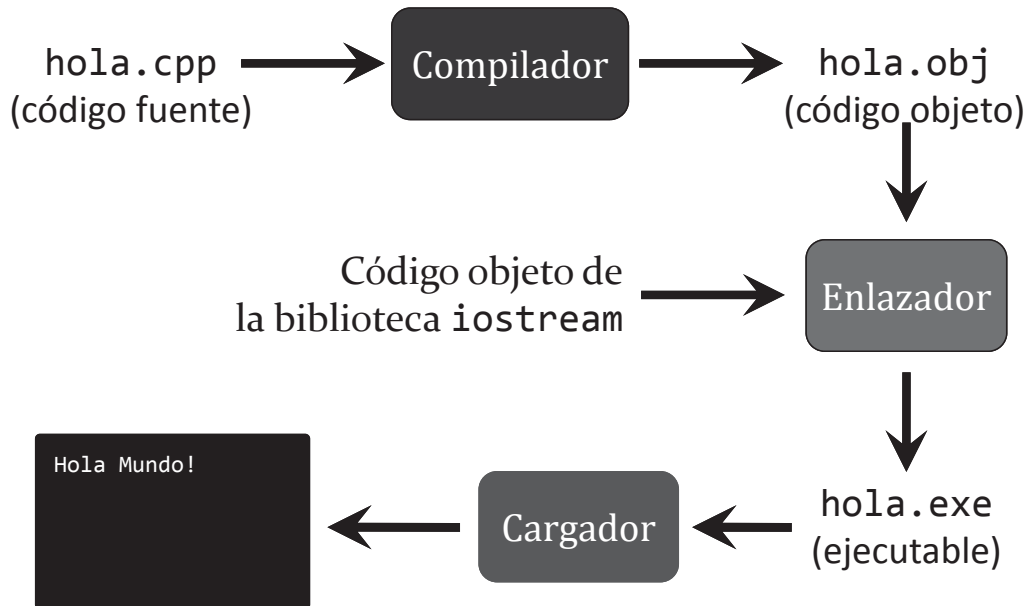
Las instrucciones terminan en ;



Compilación y enlace

cl fuente.cpp

compila y enlaza el programa del archivo fuente.cpp



El dispositivo de salida

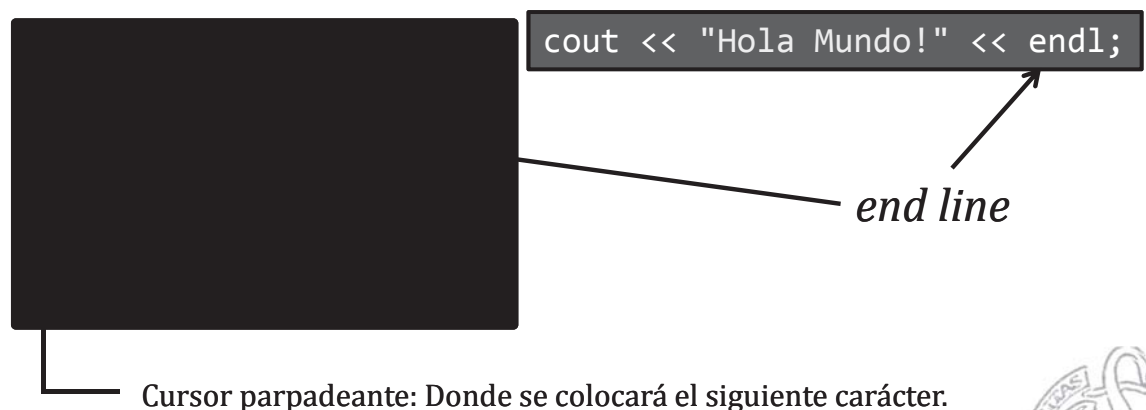
`cout` (ostream)

standard charoutput stream

Ventanas de consola o terminal → Líneas de caracteres .

Las aplicaciones en modo texto se ejecutan dentro de ventanas:

- ✓ Windows: ventanas de consola (*Símbolo del sistema*).
- ✓ Linux: ventanas de terminal.



Visualización de datos

El insertor <<

```
cout << ...;
```

Inserta texto en la consola de salida .

Transforma los datos a su representación textual.

La representación textual de los datos enviados aparecerá a continuación de la posición del cursor.

Las operaciones de inserción se pueden encadenar:

```
cout << ... << ... << ...;
```

Recuerda: las instrucciones terminan en ;



Visualización de datos

Con el insertor << podemos mostrar...

- ✓ Cadenas de caracteres literales.

Textos encerrados entre comillas dobles: "..."

```
cout << "Hola Mundo!";
```

¡Las comillas no se muestran!

Se muestran los caracteres dentro de las comillas.

- ✓ Números literales.

Con o sin decimales, con signo o no: 123, -37, 3.1416, ...

```
cout << "Pi = " << 3.1416;
```

¡Punto decimal, NO coma!

Se muestran los caracteres que representan el número.

- ✓ endl

El cursor pasa a la siguiente línea.



Literales

Un literal es un dato concreto que se escribe mediante una sintaxis específica para representar los datos de cada tipo.

Literales numéricos

Enteros (sin decimales):

Signo opcional seguido de una secuencia de dígitos.

3 143 -12 67321 -1234

No se usan puntos de millares

Reales (con decimales):

Signo opcional seguido de una secuencia de dígitos, un punto decimal y otra secuencia de dígitos.

3.1416 357.5 -1.333 2345.6789 -404.1

Punto decimal (3.1415), NO coma decimal (3,1415)



Literales

Ejemplo

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

```
int main()
{
    cout << "133 + 1234 = 1367" << endl;

    cout << 133 << " + " << 1234 << " = " << 1367 << endl;

    cout << "133 + 1234 = " << 133 + 1234 << endl;

    return 0;
}
```

Diagrama de anotaciones:

- Literal (texto) apunta a "133 + 1234 = 1367"
- Literal (número) apunta a 1367
- Operación numérica apunta a 133 + 1234
- Salto de línea apunta a endl



Variables

Datos que se mantienen en memoria

Una variable contiene un dato al que se accede por medio de un nombre (el identificador de la variable).

El dato puede modificarse cuando se quiera.

```
edad = 19; // edad es una variable y 19 es un literal
```

Las variables hay que definirlas

¿Qué tipo de dato queremos?

- ✓ Un valor numérico entero: `int`
- ✓ Un valor numérico real: `float`
- ✓ Un carácter : `char`



Variables

Definición: tipo nombre;

Al declarar una variable establecemos el tipo de dato que puede contener y el identificador (nombre) que usaremos para acceder.

```
int unidades;
```

```
float precio;
```

Para cada variable se reserva espacio suficiente en memoria para el tipo de datos.

Unos tipos de datos requieren más bytes que otros.

LAS VARIABLES NO TOMAN UN VALOR INICIAL

No se deben usar hasta que se les haya dado algún valor.

¿Dónde se definen las variables?

Siempre, antes del primer uso.

Normalmente al principio de la función.

	Memoria
unidades	?
precio	?
...	...



Variables

Definición

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

int main()
{
    int unidades;
    float precio, total;
    ...

    return 0;
}
```

	Memoria
unidades	?
precio	?
total	?
...	...

Podemos definir varias del mismo tipo separando los nombres con comas.



Variables

Capacidad de las variables

int de 32 bits (4 bytes)

Entre -2.147.483.648 y 2.147.483.647.

-2147483648 .. 2147483647

float de 4 bytes

Entre $1,18 \times 10^{-38}$ y $3,40 \times 10^{+38}$

y sus negativos $-1,18 \times 10^{-38}$ y $-3,40 \times 10^{+38}$

[+|-] 1.18e-38 .. 3.40e+38

Notación científica

Problemas de precisión -> **double** (8 bytes).



Lectura de datos

`cin (istream)`

char input stream

La lectura de valores se realiza en variables:

```
cin >> unidades;
```



Memoria
unidades 12
...



Lectura de datos

El extractor >>

```
cin >> var;
```

Lee un dato, del tipo de la variable, de la consola de entrada (teclado -> secuencia de caracteres).

Transforma los caracteres introducidos por el usuario en un dato del tipo de la variable. La entrada termina con Intro.

Se ignoran los espacios en blanco iniciales.

Las operaciones de extracción se pueden encadenar:

```
cin >> var1 >> var2 >> var3;
```

Se leerán, en orden, los datos de cada una de las variables.

¡El destino del extractor debe ser SIEMPRE una variable!



Lectura de datos

Lectura de valores enteros (int)

Se leen dígitos hasta encontrar un carácter que no lo sea

12abc↵ 12 abc↵ 12 abc↵ 12↵

Se asigna el valor 12 a la variable

El resto queda pendiente para la siguiente lectura

Lectura de valores reales (float)

Se leen dígitos, el punto decimal y otros dígitos

39.95.5abc↵ 39.95 abc↵ 39.95↵

Se asigna el valor 39,95 a la variable; el resto queda pendiente



Lectura de datos

¿Qué pasa si el usuario se equivoca?

La lectura producirá un fallo y
el programa no funcionará como debe.

Una aplicación profesional dispondría de código
de comprobación de las entradas por teclado, y ayuda.



Para evitar errores, solicita y lee cada dato



Lectura de datos

Facilitar la entrada

Indicar al usuario para cada dato qué se espera que introduzca:

```
int unidades;  
float precio, total;  
cout << "Introduce las unidades: ";  
cin >> unidades;  
cout << "Introduce el precio: ";  
cin >> precio;  
cout << "Unidades: " << unidades << endl;  
cout << "Precio: " << precio << endl;
```



Buenos hábitos de programación:

Ser amigable con el usuario 😊

Tras escribir cada dato el usuario pulsará la tecla Intro.



```
Introduce las unidades: 12  
Introduce el precio: 39.95  
Unidades: 12  
Precio: 39.95
```

¡¡¡Lectura correcta!!!



Lectura de datos

¿Qué pasa si el usuario se equivoca?

```
Introduce las unidades: abc  
Introduce el precio: Unidades: 0  
Precio: 1.79174e-307
```

No se puede leer un entero → Error

La lectura falla

La ejecución del programa
sigue de forma anómala.

```
Introduce las unidades: 12abc  
Introduce el precio: Unidades: 12  
Precio: 0
```

Se leen dos dígitos → 12 para unidades

No se puede leer un real → Error

```
Introduce las unidades: 12.5abc  
Introduce el precio: Unidades: 12  
Precio: 0.5
```

Se leen dos dígitos → 12 para unidades

Se lee .5 → 0,5 para precio



Asignación y expresiones

Asignación de valores a las variables (operador =)

`variable = expresión;` ← Instrucción: termina en ;

```
unidades = 12; // guardar 12 en unidades
precio = 39.95; // precio recibe el valor 39.95
total = unidades * precio; // Toma el valor 479.4
```

1. Se evalúa la expresión y se obtiene como resultado un valor
2. El resultado se guarda en la memoria de la variable.

¡A la izquierda del = debe ir siempre una variable!

¡A la derecha del = debe ir siempre una expresión del mismo tipo que la variable!



Expresiones

Combinación de operandos, operadores y funciones:

`unidades * precio * 1.21`

Al evaluar una expresión, el nombre de una variable representa el valor que contiene. (`12 * 39.95 * 1.21 -> 580.074`)

El resultado de la evaluación se puede guardar en una variable.

`total = unidades * precio * 1.21 ; // 580.074`

Unos operadores se evalúan antes que otros (precedencia).

A igual prioridad se evalúan de izquierda a derecha.

Paréntesis para forzar el orden de evaluación:

`unidades1 + unidades2 * precio`
`(unidades1 + unidades2) * precio`



Expresiones

Precedencia de los operadores

* y / tienen mayor precedencia que + y - (se evalúan antes).

```
unidades1 = 1;
unidades2 = 2;
precio = 40.0;
total = unidades1 + unidades2 * precio;
      → 1 + 2 * 40,0 → 1 + 80,0 → 81,0
```

Los paréntesis fuerzan la evaluación:

```
total = (unidades1 + unidades2) * precio;
      → (1 + 2) * 40,0 → 3 * 40,0 → 120,0
```



Asignación y expresiones

Ejemplo de uso de variables y expresiones

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
    precio = 39.95;
    total = unidades * precio;
    cout << unidades << " x " << precio << " = "
         << total << " (con IVA " << total * 1.21 << ")";

    return 0;
}
```



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
```

	Memoria
unidades	?
precio	?
total	?
	...



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
```

	Memoria
unidades	12
precio	?
total	?
	...



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
    precio = 39.95;
```

	Memoria
unidades	12
precio	39.95
total	?
	...



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
    precio = 39.95;
    total = unidades * precio;
```

	Memoria
unidades	12
precio	39.95
total	479.4
	...



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
    precio = 39.95;
    total = unidades * precio;
    cout << unidades << " x " << precio << " = "
         << total << endl;
```

	Memoria
unidades	12
precio	39.95
total	479.4
	...



Asignación y expresiones

Ejemplo de uso de variables

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

int main()
{
    int unidades;
    double precio, total;
    unidades = 12;
    precio = 39.95;
    total = unidades * precio;
    cout << unidades << " x " << precio << " = "
         << total << endl;
    return 0;
}
```

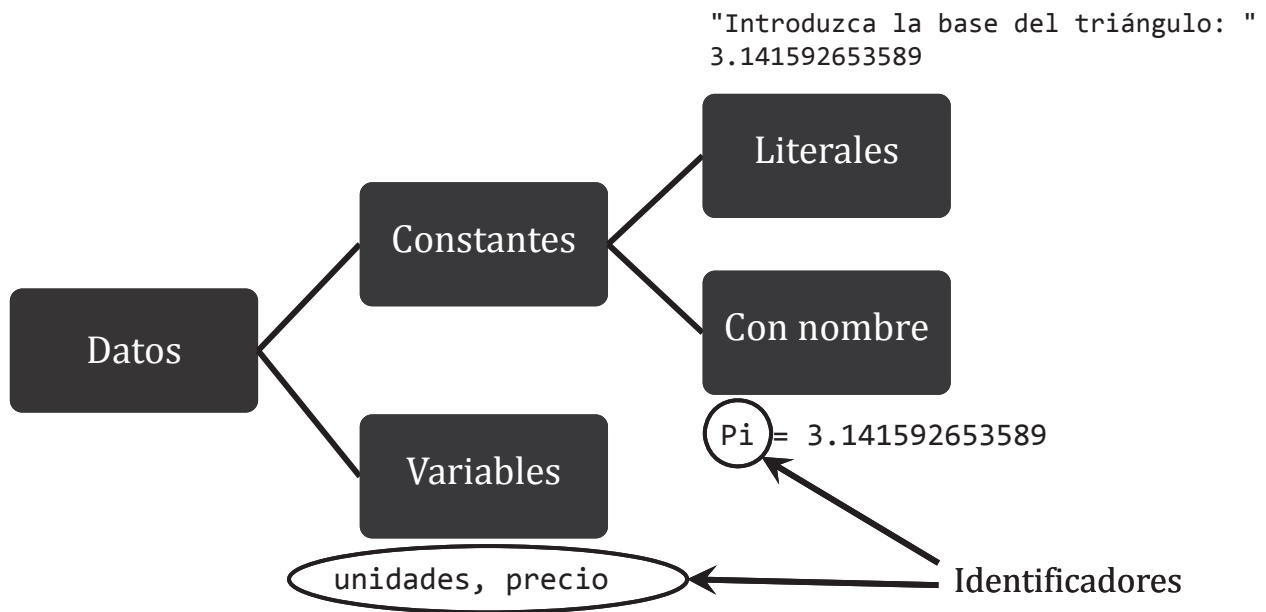
12 x 39.95 = 479.4

-



Los datos de los programas

Variabilidad de los datos



Identificadores

Identificadores

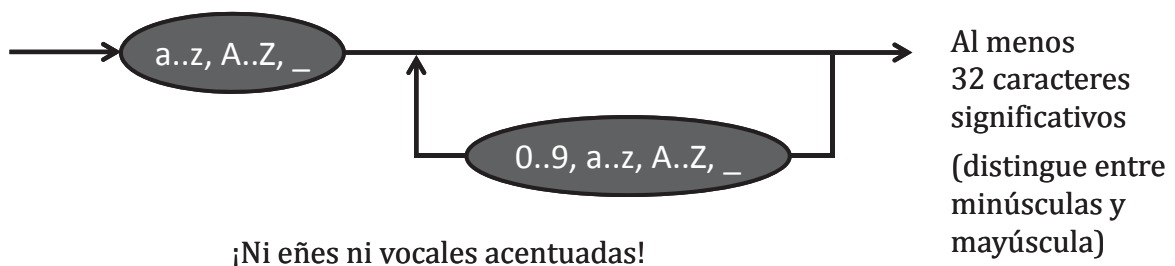
Distintos de las palabras reservadas del lenguaje

Las variables y las constantes se definen mediante

- Un identificador, el nombre. Se debe usar nombres descriptivos
- Un tipo de datos.

Sintaxis de los identificadores alfanuméricos:

unidades1 _cont altura Altura alturaA _cont_1



No pueden repetirse identificadores en un mismo ámbito (bloque).



Identificadores

Palabras reservadas del lenguaje C++

asm auto bool break case catch char class const
const_cast continue default delete do double
dynamic_cast else enum explicit extern false
float for friend goto if inline int long
mutable namespace new operator private protected
public register reinterpret_cast return short
signed sizeof static static_cast struct switch
template this throw true try typedef typeid
typename union unsigned using virtual void
volatile while



Identificadores

¿Qué identificadores son válidos y cuáles no?

balance ✓	interesAnual ✓
_base_imponible ✓	años ✗
EDAD12 ✓	salario_1_mes ✓
__edad ✓	cálculoNómina ✗
valor%100 ✗	AlgunValor ✓
100caracteres ✗	valor? ✗
_12_meses ✓	____valor ✓



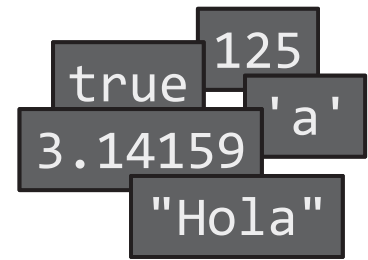
Tipos de datos

Tipos

Cada dato del programa es de un tipo concreto.

Cada tipo establece:

- El conjunto (intervalo) de valores válidos.
- El conjunto de operaciones que se pueden realizar.



En las expresiones con datos de distintos tipos compatibles se llevan a cabo transformaciones automáticas de tipos (*promoción de tipo*).

- Un entero se promociona a un real.
- Un carácter se promociona a una cadena de caracteres (texto)



Tipos de datos

Tipos de datos básicos

- `int`: para números enteros (sin parte decimal) ✓
1363, -12, 49
- `float`: para números reales (con parte decimal) ✓
12.45, -3.1932, 1.16E+02
- `double`: para números reales con mayor intervalo y precisión ✓
- `char`: para caracteres
'a', '{', '\t'
- `bool`: para valores lógicos (cierto/falso)
true, false
- `void`: *nada*, ausencia de tipo, ausencia de dato (*funciones*)



Tipos de datos

Tipos de datos definidos en la biblioteca estándar

- `string`: para cadenas de caracteres (biblioteca `<string>`).
"Hola Mundo!"
- `istream (cin), ostream (cout)` (biblioteca `<iostream>`).



Tipos de datos

char

Caracteres

Intervalo de valores: Juego de caracteres (ASCII extendido)

1 byte

Literales:

```
'a', '%', '\\t'
```

Constantes de barra invertida (o *secuencias de escape*):

Caracteres de control.

'\t' = tabulador, '\n' = salto de línea, ...

Juego de caracteres:

```
!"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz
{|}~
```

ASCII (códigos 32..127)

CüēäääâcêëèîîlÄÉæÆöôûüÿÜü£Ø×f
áíóúñÑ=90%¼;|<>>■||-ÄÄ@||§¶¥
└─┐┌─āÄ┐┌─┐┌─┐┌─┐DÉE'íîī┐┌─┐ì
óþööðµþUÚÝ'-±¼¶÷÷÷÷÷÷÷÷÷÷

ISO-8859-1
(ASCII extendido: códigos 128..255)



Tipos de datos

`char`

Caracteres

Intervalo de valores: Juego de caracteres (ASCII extendido)

1 byte

0 .. 255 (ASCII 0..127 y extensión 128..255)

Juego de caracteres (página de códigos):

asociación de un número a cada símbolo

Código de un carácter: `int(carácter)`

`int('a')`, `int('0')`, `int('\n')`, `int('\0')`

Carácter de un código: `char(código)`

`char(97)`, `char(48)`, `char(10)`, `char(0)`

Consulta el código windows -1252 o cp 1252



Tipos de datos

`bool`

Valores lógicos

Sólo dos valores posibles:

- Cierto (*true*)
- Falso (*false*)

Literales:

`true`, `false`

Cualquier número distinto de 0 es equivalente a `true` y el número 0 es equivalente a `false`.



Tipos de datos

string

Cadenas de caracteres

Literales: "Hola", "Introduce el numerador: ", "X142FG5TX?%A"



Son secuencias de caracteres. No es un tipo básico,

Hay que incluir la biblioteca `string` con el espacio de nombres `std`:

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



¡Ojo!

Las comillas tipográficas (apertura/cierre) “...” NO sirven
Asegúrate de utilizar comillas rectas: "..."



Tipos de datos

Recuerda: C++ distingue entre mayúsculas y minúsculas

`int` es la palabra reservada de C++ que permite declarar datos enteros.

`Int`, `INT` o `inT` no son palabras reservadas de C++.

`true` es la palabra reservada de C++ que representa el valor lógico *verdadero*.

`True` o `TRUE` no son palabras reservadas de C++.



Tipos de datos

Modificadores de tipos

- signed / unsigned : con signo (por defecto) / sin signo
- short / long : reduce/amplía el intervalo de valores

Tipo	Intervalo
int	-2147483648 .. 2147483647
unsigned int	0 .. 4294967295
short int	-32768 .. 32768
unsigned short int	0 .. 65535
long int	-2147483648 .. 2147483647
unsigned long int	0 .. 4294967295
double	+/- 2.23e-308 .. 1.79e+308
long double	+/- 3.37E-4932 .. 1.18E+4932



Definición de variables

[modificadores] tipo lista_de_variables;

[...] indica que eso es opcional.

Termina en ;



int i, j, l;

short int unidades;

unsigned short int monedas;

double balance, beneficio, perdida;



Programación con buen estilo:

Utiliza identificadores descriptivos.

Un espacio tras cada coma

Los nombres de las variables, en minúsculas

Varias palabras, capitaliza cada inicial: interesPorMes



Variables

Ejemplo de tipos de datos

```
#include <iostream>
#include <string>
using namespace std; // Un solo using... para las dos bibliotecas

int main()
{
    unsigned int entero = 3; // se puede inicializar (asignar)
    double real = 2.153;     // en la definición
    char caracter = 'a';
    bool cierto = true;
    string cadena = "Hola";
    cout << "Entero: " << entero << endl;
    cout << "Real: " << real << endl;
    cout << "Carácter: " << caracter << endl;
    cout << "Booleano: " << cierto << endl;
    cout << "Cadena: " << cadena << endl;

    return 0;
}
```

```
D:\FP\Tema2>tipos
Entero: 3
Real: 2.153
Caracter: a
Booleano: 1
Cadena: Hola
D:\FP\Tema2>
```

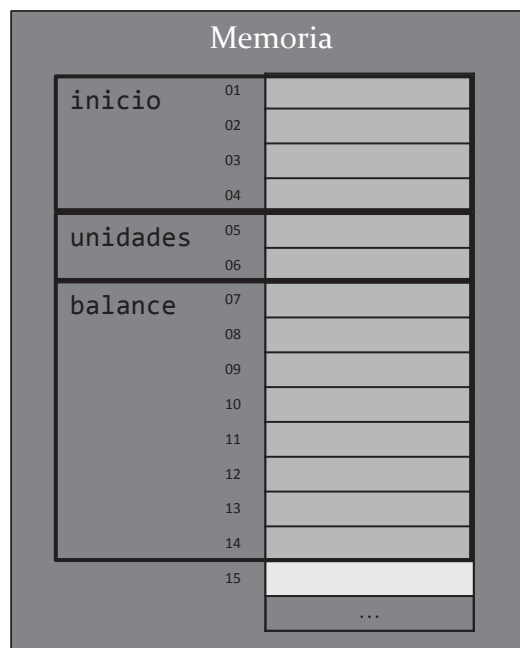


Variables

Datos y memoria

Cuando se define una variable se le reserva suficiente memoria para el tipo de datos declarado.

```
int inicio;
short int unidades;
double balance;
```



Variables

Inicialización de variables

¡En C++ las variables no toman automáticamente un valor inicial!

¡Toda variable debe ser inicializada antes de acceder a su valor!

¿Cómo se inicializa una variable?

- Al asignarle un valor (al definirla o en una instrucción posterior)
- Al leer un valor (`cin >>`)

Inicialización en la propia definición, a continuación del identificador:



```
int i = 0, j, l = 26;  
short int unidades = 100;
```

En particular, una expresión puede ser un literal



Variables

Uso de las variables

Acceso al valor de la variable:

- ✓ Nombre de la variable en una expresión

```
cout << balance;
```

```
resultado = interesPorMes * meses / 100;
```

Modificación del valor de la variable:

- ✓ Nombre de la variable a la izquierda del operador de asignación (=)

```
balance = 1214;
```

```
porcentaje = valor / 30;
```

- ✓ Nombre de la variable a la derecha del operador de extracción (>>)

```
cin >> unidades;
```

Las variables tiene que haber sido previamente declaradas



Instrucción de asignación

El operador =



A la izquierda siempre una variable, que recibe el valor resultante de evaluar la expresión (del tipo de la variable).

```
int i, j = 2;  
i = 23 + j * 5; // i toma el valor 33
```



Instrucción de asignación

Posibles errores:

```
int a, b;  
  
char c;  
  
String s;  
  
5 = a;          // ERROR: un literal no puede recibir un valor  
a + 23 = 5;     // ERROR: no puede haber una expresión a la izda.  
b = "abc";      // ERROR: un entero no puede guardar una cadena  
b = 23 5;       // ERROR: expresión no válida (falta operador)  
c = "abc";      // ERROR: un char no puede guardar una cadena  
c = a;          // ERROR: un char no puede guardar un entero  
s = "a "b" c";  // ERROR: expresión no válida
```

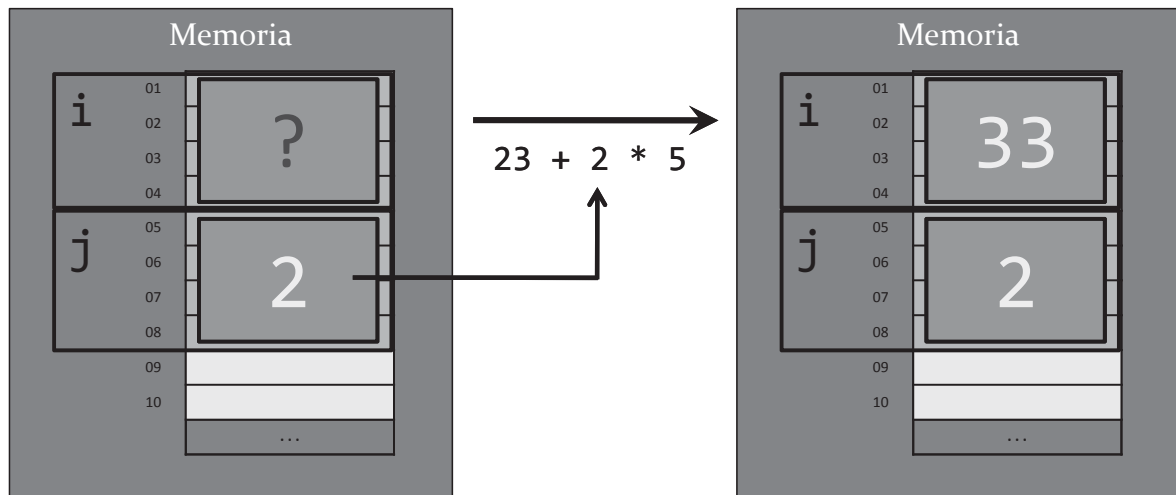


Instrucción de asignación

Variables, asignación y memoria

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

```
i = 23 + j * 5;
```



Instrucción de asignación

Ejemplo: Intercambio de valores

Se necesita una variable auxiliar.

```
double a = 3.45, b = 127.5, aux;
```

a	3.45
b	127.5
aux	?

```
aux = a;
```

```
a = b;
```

```
b = aux;
```

a	3.45
b	127.5
aux	3.45

a	127.5
b	127.5
aux	3.45

a	127.5
b	3.45
aux	3.45



Operadores

Operaciones sobre valores de los tipos

Cada tipo determina las operaciones posibles

Tipos de datos numéricos (`int`, `float` y `double`, con sus variantes):

- Asignación (`=`)
- Operadores aritméticos (`+`, `-`, `*`, `/` y `%` para tipos enteros)
- Operadores relacionales (`==`, `!=`, `<`, `<=`, `>`, `>=`)

Tipo de datos `bool`:

- Asignación (`=`)
- Operadores lógicos (`&&`, `||`, `!`)

Tipo de datos `char`:

- Asignación (`=`)
- Incremento/decremento (código siguiente/anterior)
- Operadores relacionales (`==`, `!=`, `<`, `<=`, `>`, `>=`)

Tipo de datos `string`:

- Asignación (`=`)
- Operadores relacionales (`==`, `!=`, `<`, `<=`, `>`, `>=`)



Operadores aritméticos

Operadores monarios y operadores binarios

Operadores monarios (o unarios)

Operador Operando

Operando Operador

- Cambio de signo (`-`)

`-saldo` `-RATIO` `-(3 * a - b)` `-5`

- Incremento y decremento sólo se aplican a variables (prefijo y postfijo):

`++interes` `--meses` `j++`

Operadores binarios

Operando_izquierdo Operador Operando_derecho

Los operandos pueden ser literales, constantes, variables o expresiones:

`2 + 3` `a * RATIO` `-a + b`

`(a % b) * (c / d)`



Operadores aritméticos

Operadores para tipos de datos numéricos

Operador	Operandos	Posición	int	float / double
-	1 (monario)	Prefijo	Cambio de signo	
+	2 (binario)	Infijo	Suma	
-	2 (binario)	Infijo	Resta	
*	2 (binario)	Infijo	Producto	
/	2 (binario)	Infijo	División entera	División real
%	2 (binario)	Infijo	Módulo	No aplicable
++	1 (monario)	Prefijo / postfijo	Incremento	
--	1 (monario)	Prefijo / postfijo	Decremento	



Operadores aritméticos

¿División entera o división real?



Si ambos operandos son enteros, / hace un división entera:

```
int i = 23, j = 2;  
cout << i / j; // Muestra 11
```

Si alguno de los operandos es real, / hace una división real:

```
int i = 23;  
double j = 2;  
cout << i / j; // Muestra 11.5
```



Operadores aritméticos

Módulo (resto de la división entera)

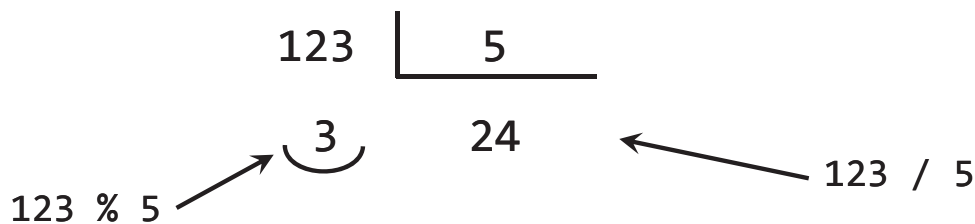
%

Ambos operandos han de ser enteros:

```
int i = 123, j = 5;  
cout << i % j; // Muestra 3
```

División entera: $\text{dividendo} = \text{divisor} * \text{cociente} + \text{resto}$

$$123 = 5 * 24 + 3$$



Operadores aritméticos

Operadores de incremento y decremento

++/--

Se incrementa/decrementa la variable numérica en una unidad.

Forma prefija: Se incrementa/decrementa antes de acceder.

```
int i = 10, j;  
i=i+1;  
j = ++i; // Se incrementa antes de copiar el valor de i en j  
cout << i << " - " << j; // Muestra 11 - 11
```

Forma postfija: Se incrementa/decrementa después de acceder.

```
int i = 10, j;  
j=i;  
i=i+1;  
j = i++; // Se incrementa después de copiar el valor de i en j  
cout << i << " - " << j; // Muestra 11 - 10
```

Sólo se pueden aplicar sobre variables (modifican el operando).



Programación con buen estilo:

Es preferible NO utilizar los operadores ++ y -- en expresiones con otros operandos/operadores.



Más sobre expresiones

Evaluación de expresiones

¿Cómo evaluamos una expresión como $3 + 5 * 2 / 2 - 1$?

¿En qué orden se aplican los distintos operadores?

¿De izquierda a derecha?

¿De derecha a izquierda?

¿Unos operadores antes que otros?

Resolución de ambigüedades:

- Precedencia de los operadores (prioridad):
Unos operadores se han de evaluar antes que otros (mayor precedencia).
- Asociatividad de los operadores (orden a igual prioridad):
Los operadores de igual prioridad se evalúan de izquierda a derecha o de derecha a izquierda de acuerdo con su asociatividad.

Paréntesis: rompen la precedencia y asociatividad

Siempre se evalúan en primer lugar las subexpresiones parentizadas.



Más sobre expresiones

Precedencia y asociatividad de los operadores

Precedencia	Operadores	Asociatividad
Mayor prioridad ↑ Menor prioridad	++ -- (postfijos)	Izquierda a derecha
	++ -- (prefijos)	Derecha a izquierda
	- (cambio de signo)	
	* / %	Izquierda a derecha
	+ -	Izquierda a derecha

$3 + 5 * 2 / 2 - 1 \rightarrow 3 + 10 / 2 - 1 \rightarrow 3 + 5 - 1 \rightarrow 8 - 1 \rightarrow 7$

Diagram illustrating the evaluation steps with arrows indicating the order of operations:

- Step 1: $3 + 5 * 2 / 2 - 1$. Arrows point to $5 * 2$ and $2 / 2$ (Same precedence: Izquierda antes).
- Step 2: $3 + 10 / 2 - 1$. An arrow points to $10 / 2$ (Mayor precedencia).
- Step 3: $3 + 5 - 1$. Arrows point to $3 + 5$ and $5 - 1$ (Same precedence: Izquierda antes).
- Step 4: $8 - 1$.
- Step 5: 7 .



Más sobre expresiones

Evaluación de expresiones

Suponiendo que la variable entera a contiene en este momento el valor 3...

$$\begin{array}{rcl} (3 + 5) * a++ + 12 / a++ - (a * 2) & & \\ \downarrow & & \downarrow \\ 8 * a++ + 12 / a++ - (3 * 2) & & \\ \downarrow & & \downarrow \\ 8 * 3 + 12 / 3 - 6 & & \\ \downarrow & & \downarrow \\ 24 + 4 - 6 & & \\ \downarrow & & \downarrow \\ 28 - 6 & & \\ \downarrow & & \downarrow \\ 22 & & \end{array}$$

Primero, los paréntesis...

Ahora, los ++ (mayor prioridad)

a pasa a valer 5



Programación con buen estilo:
Evita usar los operadores ++
y -- en expresiones compuestas.



Más sobre expresiones

Una fórmula

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

int main()
{
    double x, f;
    cout << "Introduce el valor de X: ";
    cin >> x;
    f = 3 * x * x / 5 + 6 * x / 7 - 3;
    cout << "f(x) = " << f << endl;
    return 0;
}
```

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$



Más sobre expresiones

Abreviaturas aritméticas

Cuando una expresión tiene la forma:

```
variable = variable operador op_derecho;
```

La misma

Se puede abreviar como: *variable* *operador*= *op_derecho*;

Ejemplos:

```
a = a + 12;      a += 12;
```

`a = a * 3;` `a *= 3;`

`a = a - 5;` \equiv `a -= 5;`

```
a = a / 37;      a /= 37;
```

```
a = a % b;      a %= b;
```

Operadores (prioridad)	Asociatividad
++ -- (postfijos) Llamadas a funciones	Izda. a dcha.
++ -- (prefijos) - (cambio de signo)	Dcha. a izda.
* / %	Izda. a dcha.
+ -	Izda. a dcha.
= += -= *= /= %=	Dcha. a izda.

Igual precedencia que la asignación.



Más sobre expresiones

Desbordamiento

Si una expresión produce un valor mayor del máximo permitido por el tipo correspondiente (o menor del mínimo), habrá desbordamiento.

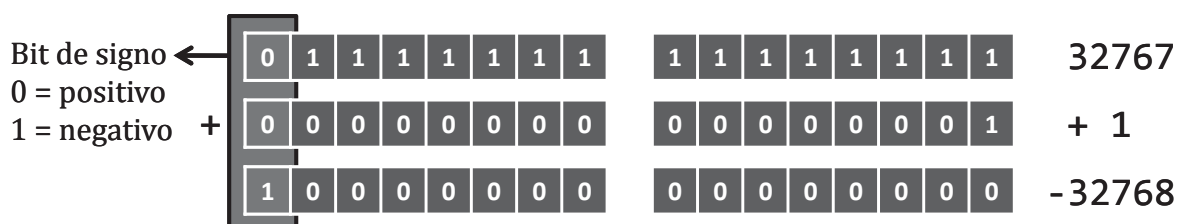
¡El resultado no será válido!

```
short int i = 32767; // Valor máximo para short int
```

```
i = i+1; cout << i;    // Muestra -32768
```

Los valores short int se codifican en memoria (2 bytes)

en *Complemento a 2* (FC); al incrementar se pasa al valor mínimo.



Debemos asegurarnos de utilizar tipos que contengan todo el conjunto de valores posibles para nuestros datos.



Constantes

Declaración de constantes

Modificador de acceso const

Declaramos variables inicializadas a las que no dejamos variar:



```
const short int Meses = 12;  
const double Pi = 3.141592,  
              RATIO = 2.179 * Pi;
```

Una constante, una vez declarada, no puede aparecer a la izquierda de una asignación, ni a la derecha del extractor >>.



Programación con buen estilo:

Pon en mayúscula la primera letra de una constante o todo su nombre.



Constantes

¿Por qué utilizar constantes con nombre?

- ✓ Aumentan la legibilidad del código:

```
cambioPoblacion = (0.1758 - 0.1257) * poblacion;      versus  
cambioPoblacion = (RatioNacimientos - RatioMuertes) * poblacion;
```

- ✓ Facilitan la modificación del código:

```
...  
double compra1 = bruto1 * 18 / 100;  
double compra2 = bruto2 * 18 / 100;  
double total = compra1 + compra2 - 18;  
cout << total << " (IVA: " << 18 << "%)" << endl;
```

3 cambios ←

```
...  
const int IVA = 18; const int PROMO = 18;  
double compra1 = bruto1 * IVA / 100;  
double compra2 = bruto2 * IVA / 100;  
double total = compra1 + compra2 - PROMO;  
cout << total << " (IVA: " << IVA << "%)" << endl;
```

¿Cambio del IVA al 21%?

1 cambio ←



Constantes

Ejemplo de uso de constantes

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

int main()
{
    const double Pi = 3.141592;
    double radio = 12.2, circunferencia;
    circunferencia = 2 * Pi * radio;
    cout << "Circunferencia de un circulo de radio "
         << radio << ": " << circunferencia << endl;

    const double Euler = 2.718281828459; // Número e
    cout << "Numero e al cuadrado: " << Euler * Euler << endl;

    const int IVA = 21;
    int unidades = 12;
    double precio = 39.95, neto, porIVA, total;
    neto = unidades * precio;
    porIVA = neto * IVA / 100;
    total = neto + porIVA;
    cout << "Total compra: " << total << endl;

    return 0;
}
```



La biblioteca cmath

Funciones matemáticas

La biblioteca `cmath` contiene muchas funciones matemáticas que podemos utilizar en nuestras expresiones.

Para poder utilizarlas hay que incluir la biblioteca:

```
#include <cmath>
```

Una llamada de una función tiene esta forma:

nombre(argumentos) Los argumentos irán separados por comas.

Por ejemplo, `abs(a)` → Valor absoluto de `a`.

Una llamada a función es un término más de una expresión.

```
f = 3 * pow(x, 2) / 5 + 2 * sqrt(x);
```



La biblioteca cmath

<code>abs(x)</code>	Valor absoluto de x
<code>pow(x, y)</code>	x elevado a y
<code>sqrt(x)</code>	Raíz cuadrada de x
<code>ceil(x)</code>	Menor entero que es mayor o igual que x
<code>floor(x)</code>	Mayor entero que es menor o igual que x
<code>exp(x)</code>	e^x
<code>log(x)</code>	Ln x (logaritmo natural de x)
<code>log10(x)</code>	Logaritmo en base 10 de x
<code>sin(x)</code>	Seno de x
<code>cos(x)</code>	Coseno de x
<code>tan(x)</code>	Tangente de x
<code>round(x)</code>	Redondeo al entero más próximo
<code>trunc(x)</code>	Pérdida de la parte decimal (entero)



La biblioteca cmath

Un ejemplo

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

int main()
{
    double x, y, f;
    cout << "Introduzca el valor de X: ";
    cin >> x;
    cout << "Introduzca el valor de Y: ";
    cin >> y;
    f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))
        / abs(x * y) - cos(y);
    cout << "f(x, y) = " << f << endl;
    return 0;
}
```

$$f(x, y) = 2x^5 + \frac{\sqrt{x^3}}{|x \times y|} - \cos(y)$$



Operaciones con caracteres

Operadores para caracteres (tipo char)

*Asignación, incremento/decremento (código siguiente/anterior)
y los operadores relacionales (==, !=, <, <=, >, >=)*

Funciones para caracteres (biblioteca ctype)

- `isalnum(c)` true si c es una letra o un dígito; si no, false.
- `isalpha(c)` true si c es una letra; si no, false.
- `isdigit(c)` true si c es un dígito; si no, false.
- `islower(c)` true si c es una letra minúscula; si no, false.
- `isupper(c)` true si c es una letra mayúscula; si no, false.
- `toupper(c)` devuelve la mayúscula de c.
- `tolower(c)` devuelve la minúscula de c.

...



Operaciones con caracteres

Un ejemplo `#include <ctype>`

```
int main()
{
    char character1 = 'A', character2 = '1', character3 = '&';
    cout << "Caracter 1 (A).-" << endl;
    cout << "Alfanumerico? " << isalnum(character1) << endl;
    cout << "Alfabetico? " << isalpha(character1) << endl;
    cout << "Digito? " << isdigit(character1) << endl;
    cout << "Mayuscula? " << isupper(character1) << endl;
    character1 = tolower(character1);
    cout << "En minuscula: " << character1 << endl;
    cout << "Caracter 2 (1).-" << endl;
    cout << "Alfabetico? " << isalpha(character2) << endl;
    cout << "Digito? " << isdigit(character2) << endl;
    cout << "Caracter 3 (&).-" << endl;
    cout << "Alfanumerico? " << isalnum(character3) << endl;
    cout << "Alfabetico? " << isalpha(character3) << endl;
    cout << "Digito? " << isdigit(character3) << endl;
    return 0;
}
```

`1 ≡ true / 0 ≡ false`



Expresiones lógicas (*booleanas*)

Operadores relacionales (*expresiones de tipo bool*)

Operando_izquierdo Operador_relacional Operando_derecho

Los operandos tiene que ser expresiones del mismo tipo (compatibles).

El resultado es de tipo bool (true o false)

Condición <-> Expresión de tipo bool

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	distinto de

Operadores (prioridad)	Asociatividad
++ -- (postfijos) Llamadas a funciones	Izda. a dcha.
++ -- (prefijos) - (cambio de signo)	Dcha. a izda.
* / %	Izda. a dcha.
+ -	Izda. a dcha.
< <= > >=	Izda. a dcha.
== !=	Izda. a dcha.
= += -= *= /= %=	Dcha. a izda.



Expresiones lógicas (*booleanas*)

Operadores relacionales

Menor prioridad que los operadores aditivos y multiplicativos.

bool resultado;

int a = 2, b = 3, c = 4;

resultado = a < 5; // 2 < 5 → true

resultado = a * b + c >= 12; // 10 >= 12 → false

resultado = a * (b + c) >= 12; // 14 >= 12 → true

resultado = a != b; // 2 != 3 → true

resultado = a * b > c + 5; // 6 > 9 → false

resultado = a + b == c + 1; // 5 == 5 → true



Error común de programación:

Confundir el operador de igualdad (==) con el operador de asignación (=).



Expresiones lógicas (*booleanas*)

Operadores lógicos (*booleanos*)

Se aplican a valores `bool` (*condiciones*):

El resultado es de tipo `bool` (`true` o `false`).

!	NO (negación)	Monario
&&	Y (conjunción)	Binario
	O (disyunción)	Binario

Operadores (prioridad)	Asociatividad
++ -- (postfijos) Llamadas a funciones	Izda. a dcha.
++ -- (prefijos) ! - (cambio de signo)	Dcha. a izda.
* / %	Izda. a dcha.
+ -	Izda. a dcha.
< <= > >=	Izda. a dcha.
== !=	Izda. a dcha.
&&	Izda. a dcha.
	Izda. a dcha.
= += -= *= /= %=	Dcha. a izda.



Expresiones lógicas (*booleanas*)

Operadores lógicos – Tablas de verdad

!	&&																												
<table><tr><th></th><th>true</th><th>false</th></tr><tr><th>true</th><td>false</td><td>true</td></tr><tr><th>false</th><td>true</td><td>false</td></tr></table>		true	false	true	false	true	false	true	false	<table><tr><th></th><th>true</th><th>false</th></tr><tr><th>true</th><td>true</td><td>false</td></tr><tr><th>false</th><td>false</td><td>false</td></tr></table>		true	false	true	true	false	false	false	false	<table><tr><th></th><th>true</th><th>false</th></tr><tr><th>true</th><td>true</td><td>true</td></tr><tr><th>false</th><td>true</td><td>false</td></tr></table>		true	false	true	true	true	false	true	false
	true	false																											
true	false	true																											
false	true	false																											
	true	false																											
true	true	false																											
false	false	false																											
	true	false																											
true	true	true																											
false	true	false																											
NO (<i>Not</i>)	Y (<i>And</i>)	O (<i>Or</i>)																											

```
bool cond1, cond2, resultado;
int a = 2, b = 3, c = 4;
resultado = !(a < 5);           // !(2 < 5) → !true → false
cond1 = (a * b + c) >= 12;      // 10 >= 12 → false
cond2 = (a * (b + c)) >= 12;    // 14 >= 12 → true
resultado = cond1 && cond2;      // false && true → false
resultado = cond1 || cond2;     // false || true → true
```



Expresiones lógicas (*booleanas*)

Ejemplo

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

int main()
{
    int num;
    cout << "Introduce un numero entre 1 y 10: ";
    cin >> num;
    if ((num >= 1) && (num <= 10))
        cout << "Numero dentro del intervalo de valores validos";
    else
        cout << "Numero no valido!";

    return 0;
}
```

¡Encierra las condiciones
simples entre paréntesis!

Condiciones equivalentes

```
((num >= 1) && (num <= 10))
((num > 0) && (num < 11))
((num >= 1) && (num < 11))
((num > 0) && (num <= 10))
```



Expresiones lógicas (*booleanas*)

Propiedades (álgebra de Boole)

- $\text{true} \ \&\& \ c \approx c$
- $\text{false} \ || \ c \approx c$
- $\text{false} \ \&\& \ c \approx \text{false}$
- $\text{true} \ || \ c \approx \text{true}$
- $c \ \&\& \ !c \approx \text{false}$
- $c \ || \ !c \approx \text{true}$
- $c \ \&\& \ c \approx c$
- $c \ || \ c \approx c$
- $!! \ c \approx c$
- $!(a \ || \ b) \approx !a \ \&\& \ !b$
- $!(a \ \&\& \ b) \approx !a \ || \ !b$
- $||$ y $\&\&$ son asociativos, distributivos, conmutativos

Cortocircuito de expresiones:
evaluación por necesidad



No conmutativos



Evaluación perezosa

Shortcut Boolean Evaluation

`true || X ≡ true`

`(n == 0) || (x >= 1.0 / n)`

Si `n` es 0: ¿división por cero? (segunda condición)

Como la primera sería `true`: ¡no se evalúa la segunda!

`false && X ≡ false`

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

Si `n` es 0: ¿división por cero? (segunda condición)

Como la primera sería `false`: ¡no se evalúa la segunda!



¿Año bisiesto?

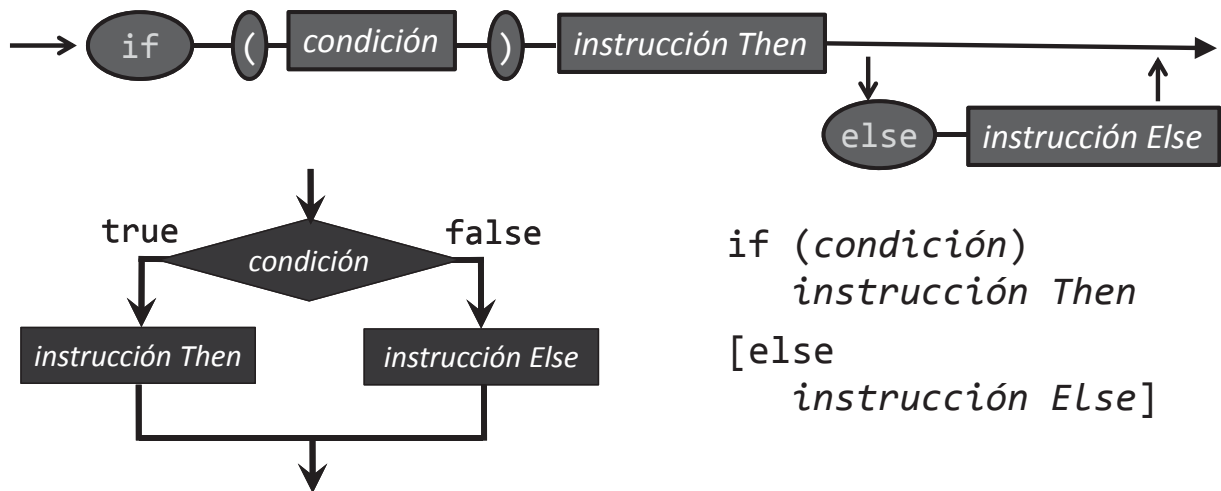
Calendario Gregoriano: bisiesto si divisible por 4, excepto el último de cada siglo (divisible por 100), salvo que sea divisible por 400

```
bool esBisiesto(int anio) {
    bool auxEsBisiesto =
        ((anio % 4) == 0) && // Divisible por 4
        ((anio % 100) != 0) || // No divisible por 100
        ((anio % 400) == 0) // Divisible por 400
    )
    return auxEsBisiesto;
}
```



Instrucción condicional

Si la condición es cierta, hacer esto...; si no, hacer esto otro...



- 1- Se evalúa la condición
- 2- Si el resultado es
true → ejecutar *inst. Then*
false → ejecutar *inst. Else*



La instrucción if

Ejemplos

```
int main()
{
    int num;
    cout << "Entero: ";
    cin >> num;
    if (num % 2 == 0)
        cout << "es par" << endl;
    else
        cout << "es impar" << endl;
    return 0;
}
```

```
int main()
{
    int op1 = 13, op2 = 4;
    int opcion;
    cout << "1 - Sumar" << endl;
    cout << "2 - Restar" << endl;
    cout << "Opcion: ";
    cin >> opcion;
    if (opcion == 1)
        cout << op1 + op2 << endl;
    else if (opcion == 2)
        cout << op1 - op2 << endl;
    else
        cout << "Error" << endl;
    return 0;
}
```



Bloques de código

¿Hay que hacer más de una cosa?

Si en alguna de las dos ramas del `if` es necesario poner varias instrucciones, se necesita crear un bloque de código:

```
{
  Tab o | instrucción1
  2 ó 3 → | instrucción2
  esp.   | ...
         | instrucciónN
}
```



Cada instrucción simple terminada en ;

Ejemplo:

```
int num, den;
cin >> num >> den;
if (den != 0)
```

```
{
  float div = float(num) / den;
  cout << endl << "Resultado:";
  cout << div;
}
```

```
cout << endl;
```

Cada bloque crea un nuevo ámbito en el que las declaraciones son locales.



Bloques de código

Posición de las llaves: cuestión de estilo

```
if (den != 0)
{
  float div = float(num)/den;
  cout << endl << "Resultado:";
  cout << div;
}
cout << endl;
```

```
if (den != 0) {
  float div = float(num)/den;
  cout << endl <<"Resultado:";
  cout << div;
}
cout << endl;
```

No te dejes engañar

```
if (den != 0)
  float div = float(num)/den;
  cout << endl << "Resultado:";
  cout << div;
```

≡

```
if (den != 0)
  float div = float(num)/den;
  cout << endl << "Resultado:";
  cout << div;
```

Aunque la sangría pueda dar a entender, a simple vista, que las tres instrucciones se ejecutan en el caso *then* del `if`, las instrucciones `cout` se ejecuta *en cualquier caso* (sea `den` distinto de cero o no)



Bloques de código

Ejemplo

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

int main()
{
    int op1 = 13, op2 = 4, resultado;
    int opcion;
    char operador;
    cout << "1 - Sumar" << endl;
    cout << "2 - Restar" << endl;
    cout << "Opcion: ";
    cin >> opcion;
    if (opcion == 1) {
        operador = '+';
        resultado = op1 + op2;
    }
    else {
        operador = '-';
        resultado = op1 - op2;
    }
    cout << op1 << operador << op2 << " = " << resultado << endl;

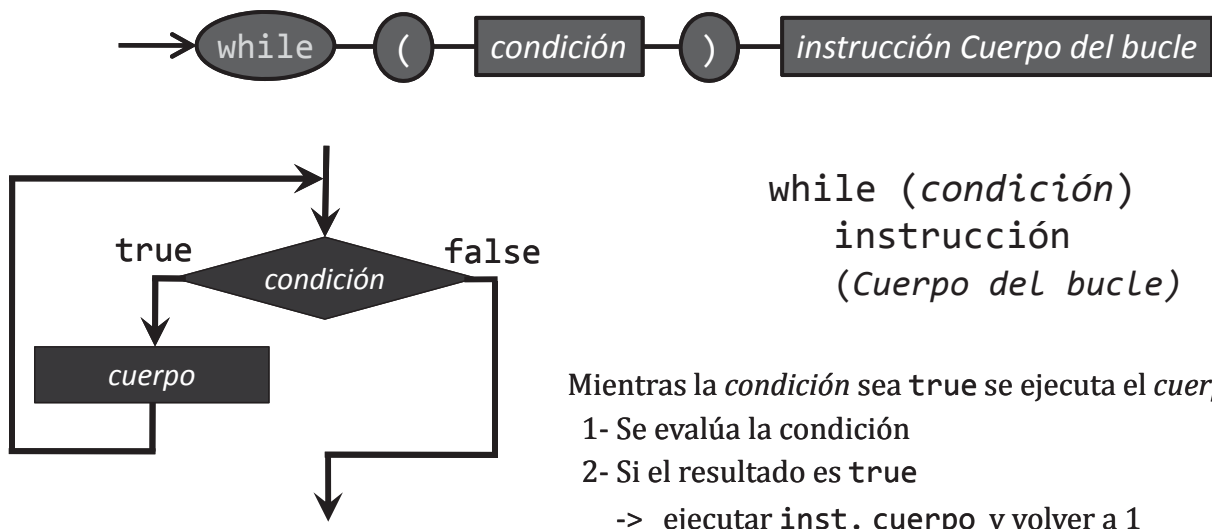
    return 0;
}
```



Instrucción iterativa

Iteración condicional:

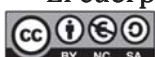
Mientras que la condición sea cierta, repetir esto...



Si inicialmente la *condición* es *false*, el *cuerpo* no se ejecuta ninguna vez.

El número de iteraciones debe ser finito: la condición tiene que cambiar de *true* a *false*

El cuerpo del bucle tiene que modificar las variables de la condición



La instrucción while

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

int main()
{
    int x, z, p, i;
    ... cin >> x;
    ... cin >> z;
    i = 0; p = 1;
    while (i < z) {
        p = p * x;
        i = i + 1;
    }
    cout << "..." << p << endl;

    return 0;
}
```

$$p = \prod_{i=1}^z x$$

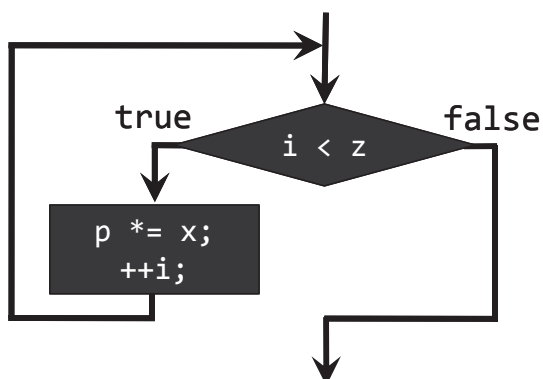
$p = x * .. * x$
i veces



La instrucción while

```
i = 0; p = 1;
while (i < z) {
    p = p * x;
    i = i + 1;
}
```

$$p = \prod_{i=1}^z x$$



x	2
z	4
i	4
p	16

$p = x * .. * x$
i veces

$p = x * .. * x$
i veces
i = z

-> $p = \text{pow}(x, z)$



Entrada/salida por consola (teclado/pantalla)

Flujos de texto

`cin` (de tipo `istream`) y `cout` (de tipo `ostream`) son variables definidas en la biblioteca `iostream`, con los operadores:

>> *Extractor*: Operador binario para entrada de datos.

El operando izquierdo es un flujo de entrada (`cin`) y el operando derecho una variable. El resultado de la operación es el propio flujo de entrada.



<< *Insertor*: Operador binario para salida de datos.

El operando izquierdo es un flujo de salida (`cout`) y el operando derecho una expresión. El resultado de la operación es el propio flujo de salida.



La asociatividad de ambos operadores es de izquierda a derecha.



Entrada/salida por consola

Flujos de texto

`cin` y `cout` son objetos complejos que constan de

- una secuencia de caracteres (buffer),
- un cursor (posición en la secuencia, elemento en curso) y
- variables de estado de error

Y se utilizan mediante los operadores y funciones definidos en la biblioteca.

La lectura y escritura puede hacerse

- carácter a carácter, mediante las funciones básicas `get()` y `put()` respectivamente, o
- mediante los operadores `>>` y `<<`, que realizan un proceso de transformación del texto a otros tipos de datos, de acuerdo a un determinado formato del texto.



Lectura de datos desde el teclado

Extractor



Comienza a extraer caracteres en la posición en que se encuentra el cursor.

Se salta los *espacios en blanco* (espacios, tabuladores o saltos de línea) y según cual se el tipo de datos declarado para la variable de la derecha

- **char**: asigna el carácter a la variable, quedando el cursor a continuación.
- **int**: lee los dígitos seguidos que haya (con posible signo inicial) y transforma la secuencia de dígitos en un valor entero que asigna a la variable. El cursor queda a continuación del último dígito leído.
- **float**: lee la parte entera; y si hay un punto lee los dígitos que le sigan para la parte decimal; transforma la secuencia en un valor real que se asigna a la variable. El cursor queda a continuación del último dígito leído.
- **bool**: Si lo leído es 1, la variable toma el valor **true**; con cualquier otra entrada toma el valor **false**. No se suelen leer este tipo de variables.

En la lectura de datos numéricos, si tras saltar los *espacios en blanco* no se encuentra un dígito o carácter válido, se produce un error que deja al flujo en modo fallo. Si el flujo está en modo fallo, no realiza ninguna operación hasta que se restablezca.

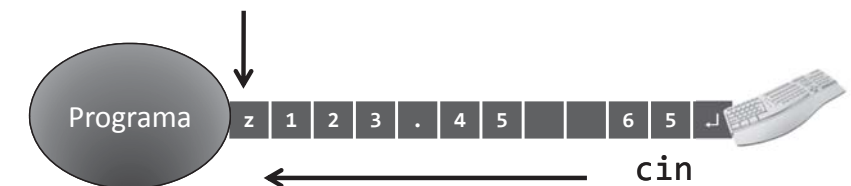


Entrada por teclado

Flujos de entrada

```
int i;  
char c;  
double d;  
cin >> c >> d >> i;
```

```
    {  
cin >> d >> i;  
    {  
cin >> i;
```



Se lee el carácter 'z' y se asigna a la variable c; resultado: cin

Se leen los caracteres 123.45, se convierten al valor 123.45 (válido para double) y se asigna a la variable d; resultado: cin

Se saltan los espacios, se leen los caracteres 65, se convierten al valor 65 (válido para int) y se asigna a la variable i



Entrada por teclado

Ejecución de funciones con el operador punto (.)

Algunos tipos de datos (como `istream`, `ostream` o `string`) requieren que algunas de sus funciones (métodos) se llamen con la notación del operador punto:

`variable.función(argumentos)`

Por ejemplo:

```
char c;
cin.get(c); // Llama a la función get() sobre la variable cin
La función get() lee el siguiente carácter sin saltar espacios en blanco
y lo asigna a la variable c (también los caracteres de espacio en blanco)

cin.sync(); // descarta el resto de caracteres pendientes

cin.ignore(1, '\n');      cin.ignore(INT_MAX, '\n');
// descarta 1/INT_MAX caracteres o hasta salto de línea
```



Estados de error (flags)

`cin.fail()`; // Llama a la función `fail()` sobre la variable `cin`
La función `fail()` devuelve un booleano indicando si el flujo se encuentra en modo fallo.

`cin.clear()`; // Llama a la función `clear()` sobre la variable `cin`
La función `clear()` restablece el flujo, eliminando el estado de fallo.

Por ejemplo:

```
int num;
cout << "Introduce un número entero: ";
cin >> num;
while (cin.fail()) {
    cin.clear(); cin.sync();
    cout << "Numero no valido!";
    cout << "Introduce un número entero: ";
    cin >> num;
}
cin.sync();
```



Entrada por teclado

Lectura de cadenas de caracteres (string) con el extractor

Comienza a extraer caracteres en la posición en que se encuentra el cursor. Se salta los *espacios en blanco*, lee los caracteres que haya, hasta encontrar un espacio en blanco. El cursor queda en dicho espacio.

```
string nombre, apellidos;  
cout << "Nombre: ";  
cin >> nombre;  
cout << "Apellidos: ";  
cin >> apellidos;  
cout << "Nombre completo: "  
    << nombre << " "  
    << apellidos << endl;
```

```
Nombre: Luis Antonio  
Apellidos: Nombre completo: Luis Antonio
```

apellidos toma la cadena "Antonio"

```
string nombre, apellidos;  
cout << "Nombre: ";  
cin >> nombre;  
cin.sync();  
cout << "Apellidos: ";  
cin >> apellidos;  
cout << "Nombre completo: "  
    << nombre << " "  
    << apellidos << endl;
```

```
Nombre: Luis Antonio  
Apellidos: Hernández Yáñez  
Nombre completo: Luis Hernández
```

¿Cómo leer varias palabras?



Entrada por teclado

Lectura de cadenas de caracteres (string)

Para leer incluyendo espacios en blanco se usa la función `getline()`:
`getline(cin, cadena)`

Se guardan en la *cadena* todos los caracteres que haya desde el cursor hasta el final de la línea (Intro).

```
string nombre, apellidos;  
cout << "Nombre: ";  
getline(cin, nombre);  
cout << "Apellidos: ";  
getline(cin, apellidos);  
cout << "Nombre completo: "  
    << nombre << " "  
    << apellidos << endl;
```

```
D:\FP\Tema2>string  
Nombre: Luis Antonio  
Apellidos: Hernández Yáñez  
Nombre completo: Luis Antonio Hernández Yáñez
```



Salida por pantalla

Insertor



- Se calcula el resultado de la expresión.
- Se convierte (si es necesario) a su representación textual y
- se envían los caracteres al flujo de salida. Se insertan a continuación del cursor, quedando el cursor al final.

```
int meses = 7;  
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
```

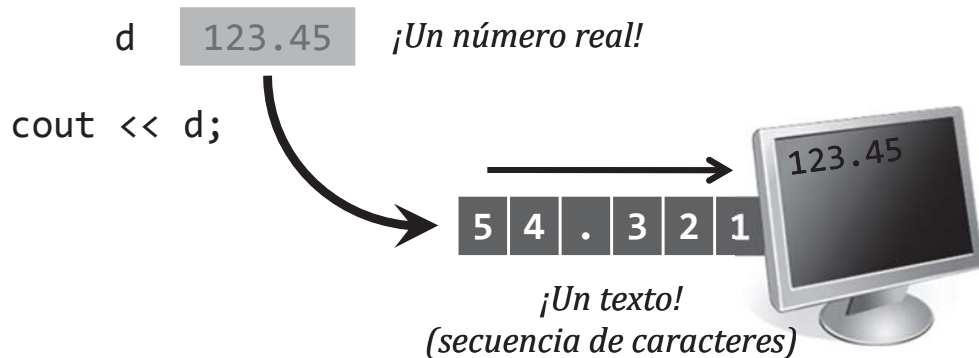
La biblioteca `iostream` define la constante `endl` como un salto de línea.



Salida por pantalla

Representación textual de los datos

El valor `double` 123.45 se guarda en memoria en binario.
Su representación textual es: '1' '2' '3' '.' '4' '5'
`double d = 123.45;`



Salida por pantalla

Flujos de salida



T o t a l : 1 2 3 . 4 5 ↵ M e s e s : 7
cout ←

Programa

```
int meses = 7;
cout << "Total: " << 123.45 << endl << " Meses: " << meses;
cout << 123.45 << endl << " Meses: " << meses;
cout << endl << " Meses: " << meses;
cout << " Meses: " << meses;
cout << meses;
```



Salida por pantalla

Formato de la representación textual

Las bibliotecas `iostream` e `iomanip` permiten establecer *opciones de formato* que afectan a la E/S que se realice a continuación.

Biblioteca	Identificador	Propósito
iostream	<code>showpoint</code> / <code>noshownpoint</code>	Mostrar o no el punto decimal para reales sin parte decimal (añadiendo 0 a la derecha).
	<code>fixed</code> / <code>scientific</code>	Notación de punto fijo / científica (reales).
	<code>boolalpha</code>	Valores <code>bool</code> como <code>true</code> / <code>false</code> .
	<code>left</code> / <code>right</code>	Ajustar a la izquierda/derecha.
iomanip	<code>setw(anchura)*</code>	Nº de caracteres (anchura) para el dato.
	<code>setprecision(p)</code>	Precisión: Nº de dígitos (antes y después del .). Con <code>fixed</code> o <code>scientific</code> , nº de decimales.

***`setw()` sólo afecta al siguiente dato que se escriba, mientras que los otros afectan a todos**



Salida por pantalla

Formato de la salida (ejemplo)

```
bool fin = false;
cout << fin << "->" << boolalpha << fin << endl;
double d = 123.45;
char c = 'x';
int i = 62;
cout << d << c << i << endl;
cout << "|" << setw(8) << d << "|" << endl;
cout << "|" << left << setw(8) << d << "|" << endl;
cout << "|" << setw(4) << c << "|" << endl;
cout << "|" << right << setw(5) << i << "|" << endl;
double e = 96;
cout << e << " - " << showpoint << e << endl;
cout << scientific << d << fixed << endl;
cout << setprecision(8) << d << endl;
```

0->>false

123.45x62

| 123.45|

|123.45 |

|x |

| 62|

96 - 96.0000

1.234500e+002

123.45000000



Funciones en C++

Los programas pueden incluir otras funciones además de `main()`

Forma general de una función en C++:

```
tipo nombre(parámetros) // Cabecera
{
    // Cuerpo
}
```

- ✓ *Tipo* de dato que devuelve la función como resultado
- ✓ *Parámetros* para proporcionar datos a la función
- Declaraciones de variables separadas por comas
- ✓ *Cuerpo*: secuencia de declaraciones e instrucciones
¡Un bloque de código!



Datos en las funciones

- ✓ Datos locales: declarados en el cuerpo de la función
Datos auxiliares que utiliza la función (puede no haber)
- ✓ Parámetros: declarados en la cabecera de la función
Datos de entrada de la función (puede no haber)

Ambos son de uso exclusivo de la función y no se conocen fuera

```
double f(int x, int y) {  
    // Declaración de datos locales:  
    double resultado;  
  
    // Instrucciones:  
    resultado = 2 * pow(x, 5) + sqrt(pow(x, 3)  
        / pow(y, 2)) / abs(x * y) - cos(y);  
  
    return resultado; // Devolución del resultado  
}
```

$$f(x, y) = 2x^5 + \frac{\sqrt{\frac{x^3}{y^2}}}{|x \times y|} - \cos(y)$$



Argumentos

Llamada a una función con parámetros

Nombre(Argumentos)

- Tantos argumentos entre los paréntesis como parámetros
- Orden de declaración de los parámetros
- Cada argumento: mismo tipo que su parámetro
- Cada argumento: expresión válida (se pasa el resultado)

```
double valor = f(2, 3);
```

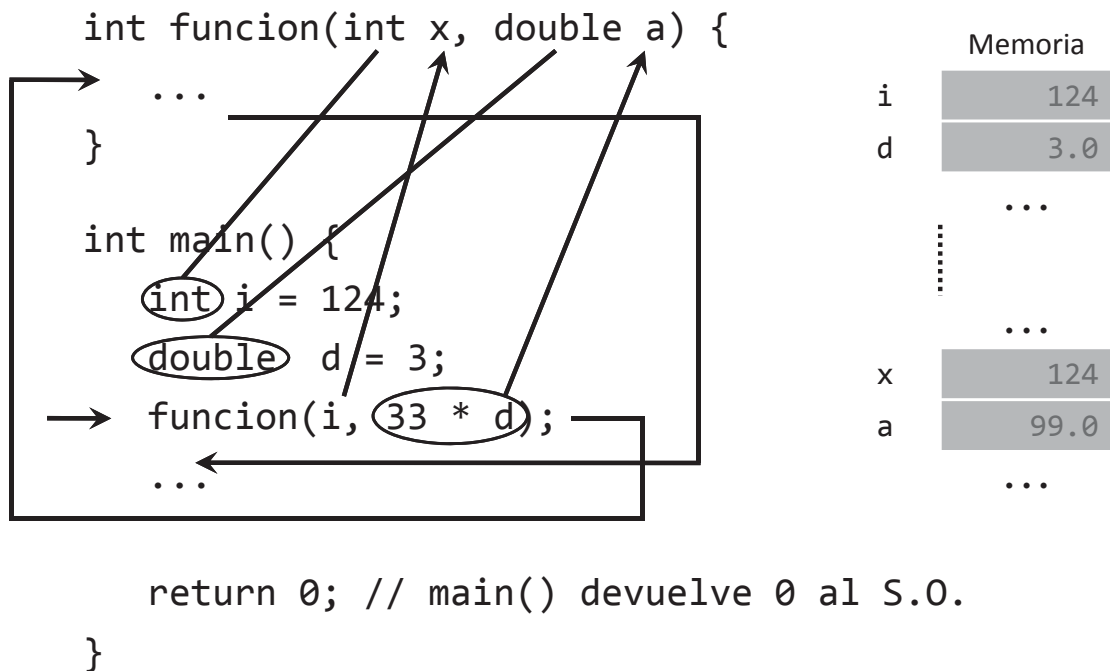
Ejecución de una llamada:

1. Se crean las variables locales (incluidos los parámetros)
2. Se evalúan los argumentos
3. Se copian los valores resultantes en los correspondientes parámetros
4. Se ejecuta el cuerpo de la función
5. Se regresa el resultado y se eliminan las variables locales



Paso de argumentos

Se copian los argumentos en los parámetros



Resultado de la función

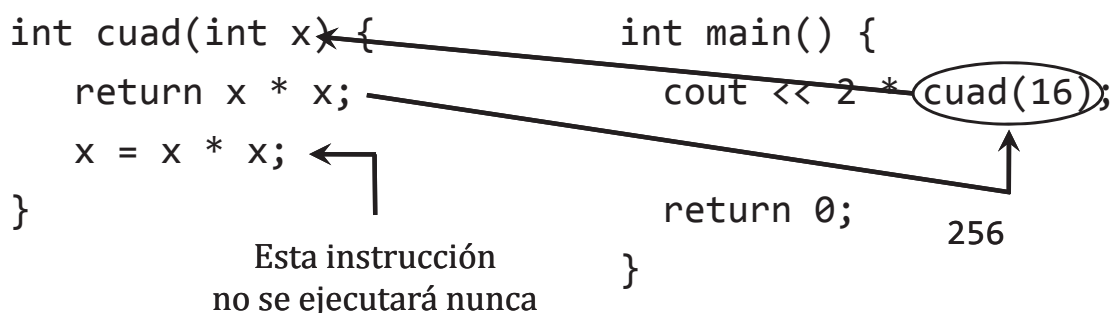
La función ha de devolver un resultado

La función termina su ejecución devolviendo un resultado

La instrucción **return** (*sólo una en cada función*)

- Devuelve el dato que se pone a continuación (tipo de la función)
- Termina la ejecución de la función

El dato devuelto sustituye a la llamada de la función:



Prototipos de las funciones

¿Qué funciones hay en el programa?

¿Son correctas las llamadas a funciones del programa?

- ¿Existe la función?
- ¿Concuerdan los argumentos con los parámetros?
- Prototipo de función: Cabecera de la función terminada en ;

```
double f(int x, int y);  
int funcion(int x, double a)  
int cuad(int x);  
int leerInt();  
...
```

Los prototipos antes de main()

Las definiciones de las funciones después de main()



Un programa con funciones

```
#include <iostream>  
using namespace std;  
#include <cmath>  
  
// Prototipos de las funciones  
int leerInt();  
bool par(int num);  
bool letra(char car);  
double formula(int x, int y);
```

```
int main() {  
    int numero = leerInt();  
    if (par(numero)) {  
        cout << "Par";  
    }  
    else {  
        cout << "Impar";  
    }  
    cout << endl;  
    ...  
}
```



Excepto para main()



Un programa con funciones

```
char character;
cout << "Carácter: ";
cin >> character;

if (!letra(character)) {
    cout << "no ";
}
cout << "es una letra" << endl;

int x, y;
x = leerInt();
y = leerInt();
cout << "f(x, y) = " << formula(x, y) << endl;
// Los argumentos pueden llamarse igual o no que los parámetros

return 0;
}
...
```



Un programa con funciones

```
// Implementación de las funciones
int leerInt() {
    int num;
    cout << "Introduce un número entero: ";
    cin >> num;
    while (cin.fail()) {
        cin.clear(); cin.sync();
        cout << "Numero no valido!" << "Introduce un número entero: ";
        cin >> num;
    }
    cin.sync();
    return num;
}

bool par(int num) {
    bool esPar = (num % 2 == 0);
    return esPar;
}
```



Un programa con funciones

```
bool letra(char car) {  
    return ((car >= 'a') && (car <= 'z') ||  
            (car >= 'A') && (car <= 'Z'))  
}  
  
double formula(int x, int y) {  
    double f;  
  
    f = 2 * pow(x, 5) + sqrt(pow(x, 3) / pow(y, 2))  
        / abs(x * y) - cos(y);  
  
    return f;  
}
```






Acerca de *Creative Commons*



Licencia CC (Creative Commons)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

-  **Reconocimiento (*Attribution*):**
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
-  **No comercial (*Non commercial*):**
La explotación de la obra queda limitada a usos no comerciales.
-  **Compartir igual (*Share alike*):**
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

