Fundamentos de la programación

2 Tipos e instrucciones I

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Índice

Un ejemplo de programación

Luis Hernández Yáñez

Una computadora de un coche

Instrucciones que entiende:

<instrucción> ::= <inst> `;`

<inst> ::= `Start` | `Stop` | <avanzar>

<avanzar> ::= `Go` <dirección> <num> `Blocks`

<dirección> ::= `North` | `East` | `South` | `West`

<num> ::= `1` | `2` | `3` | `4` | `5`

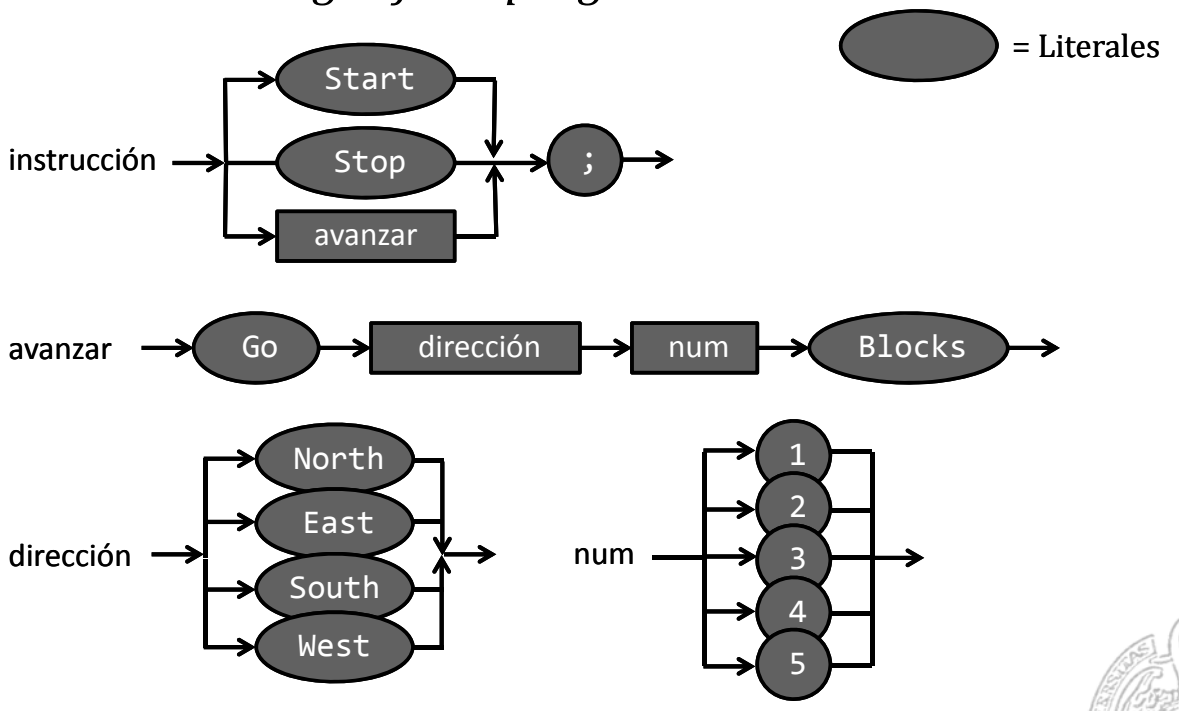
Ejemplos:

```
Start;
Go North 3 Blocks;
Stop;
```

Luis Hernández Yáñez

Un ejemplo de programación

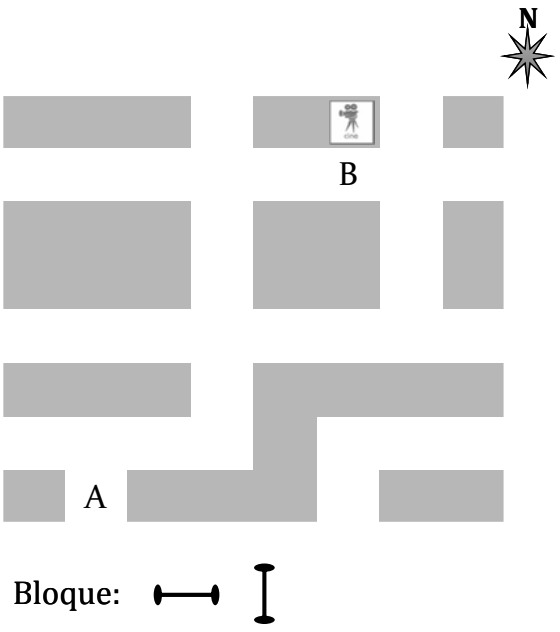
Sintaxis del lenguaje de programación



Un ejemplo de programación

El problema a resolver

Estando el coche en la posición A, conseguir llegar al Cine Tívoli (B)



¿Qué pasos hay que seguir?

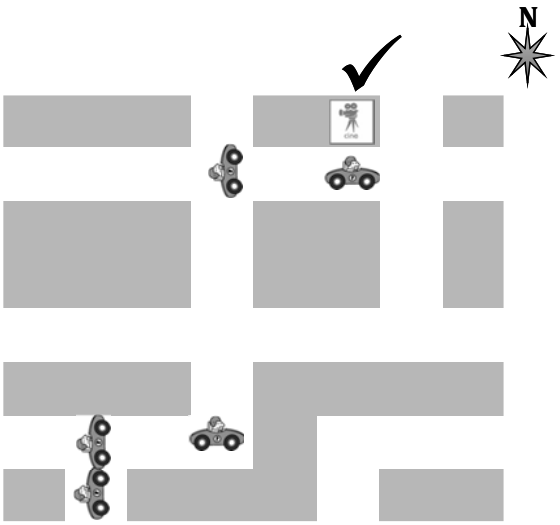
Arrancar
Ir un bloque al Norte
Ir dos bloques al Este
Ir cinco bloques al Norte
Ir dos bloques al Este
Parar

Un ejemplo de programación

El algoritmo

Secuencia de pasos que hay que seguir para resolver el problema

1.- Arrancar
2.- Ir un bloque al Norte
3.- Ir dos bloques al Este
4.- Ir cinco bloques al Norte
5.- Ir dos bloques al Este
6.- Parar



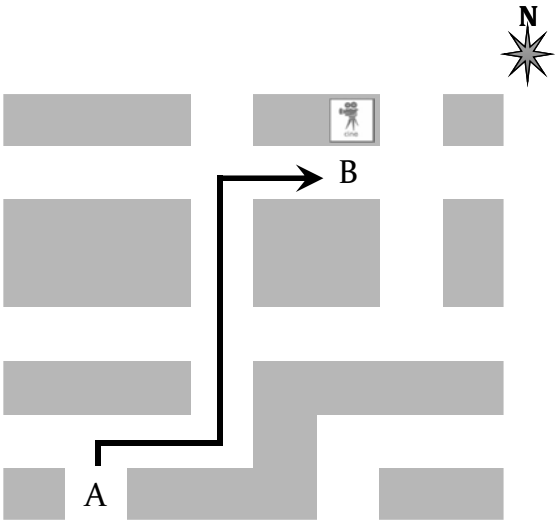
Esos pasos sirven tanto para una persona como para una computadora.

Un ejemplo de programación

El programa

Instrucciones escritas en el lenguaje de programación

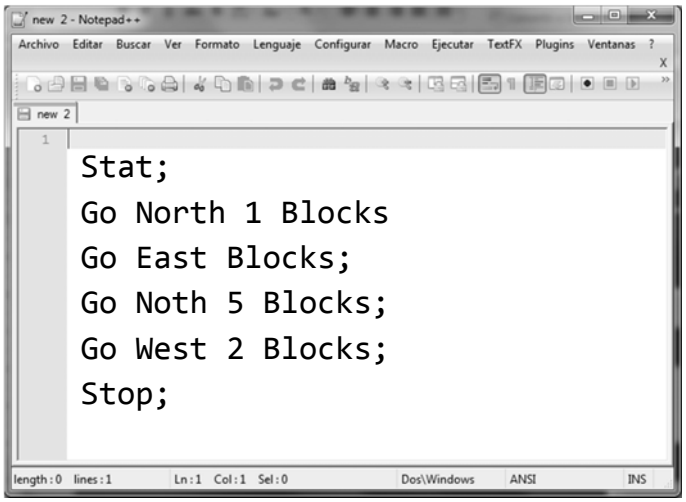
```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```



Un ejemplo de programación

El programa

Escribimos el código del programa en un editor
y lo guardamos en un archivo:



Copiamos el archivo
en una llave USB
y lo llevamos al coche

Un ejemplo de programación

La compilación

Introducimos la llave USB en el coche
y pulsamos el botón de ejecutar el programa:

```
Stat;
----^ Unknown word.
Go North 1 Blocks
-----------------^ ; missing.
Go East Blocks;
--------^ Number missing.
Go Noth 5 Blocks;
-------^ Unknown word.
Go West 2 Blocks;
Stop;
There are errors. Impossible to run the program.
```

Errores
de sintaxis

Un ejemplo de programación

Depuración

Editamos el código para corregir los errores sintácticos:

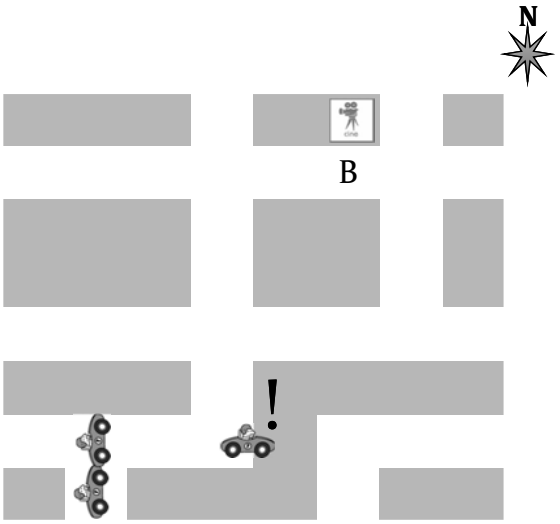
```
Stat;                          Start;
Go North 1 Blocks              Go North 1 Blocks;
Go East Blocks;        ⇒       Go East 3 Blocks;
Go Noth 5 Blocks;              Go North 5 Blocks;
Go West 2 Blocks;              Go West 2 Blocks;
Stop;                          Stop;
```


Un ejemplo de programación

La ejecución

Se realiza lo que pide cada instrucción:

```
Start;
Go North 1 Blocks;
Go East 3 Blocks;
```



Error de ejecución

¡Una instrucción no se puede ejecutar!

Un ejemplo de programación

Depuración

Editamos el código para arreglar el error de ejecución:

```
Start;                          Start;
Go North 1 Blocks;              Go North 1 Blocks;
Go East 3 Blocks;      ⇒        Go East 2 Blocks;
Go North 5 Blocks;              Go North 5 Blocks;
Go West 2 Blocks;               Go West 2 Blocks;
Stop;                           Stop;
```

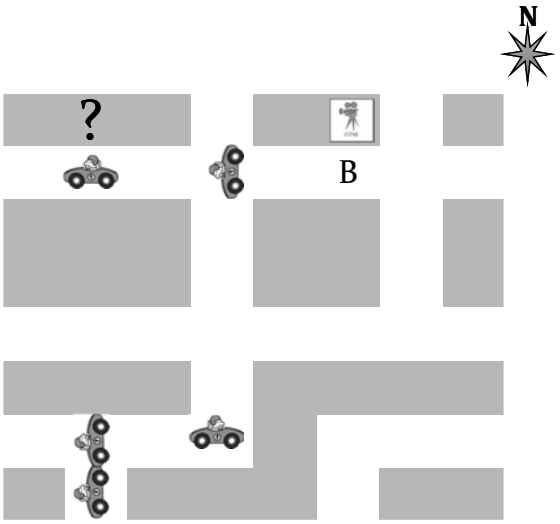
Luis Hernández Yáñez

Un ejemplo de programación

La ejecución

Se realiza lo que pide cada instrucción:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go West 2 Blocks;
Stop;
```



Error lógico

¡El programa no llega al resultado deseado!

Luis Hernández Yáñez

Un ejemplo de programación

Depuración

Editamos el código para arreglar el error lógico:

```
Start;                          Start;
Go North 1 Blocks;              Go North 1 Blocks;
Go East 2 Blocks;      ⇒        Go East 2 Blocks;
Go North 5 Blocks;              Go North 5 Blocks;
Go West 2 Blocks;               Go East 2 Blocks;
Stop;                           Stop;
```

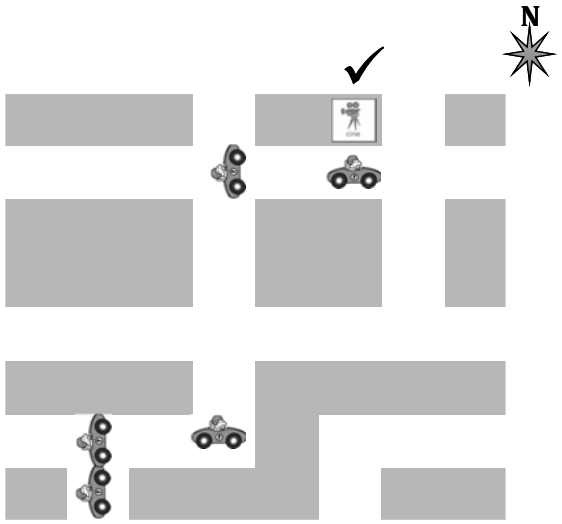
Luis Hernández Yáñez

Un ejemplo de programación

La ejecución

Se realiza lo que pide cada instrucción:

```
Start;
Go North 1 Blocks;
Go East 2 Blocks;
Go North 5 Blocks;
Go East 2 Blocks;
Stop;
```



¡Conseguido!

Luis Hernández Yáñez

El primer programa en C++

Luis Hernández Yáñez

El primer programa en C++

Hola Mundo!

De vuelta en el programa que muestra un saludo en la pantalla:

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

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

   return 0;
}
```

Luis Hernández Yáñez

BY NC SA

El primer programa en C++

Hola Mundo!

La única instrucción que produce algo tangible:

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

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

   return 0;
}
```


El primer programa en C++

`cout` (`iostream`) c*haracter* out*put stream*

Visualización en la pantalla: operador << (*insertor*)

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

`<< "Hola Mundo!" << endl;`

```
Hola Mundo!
_
```

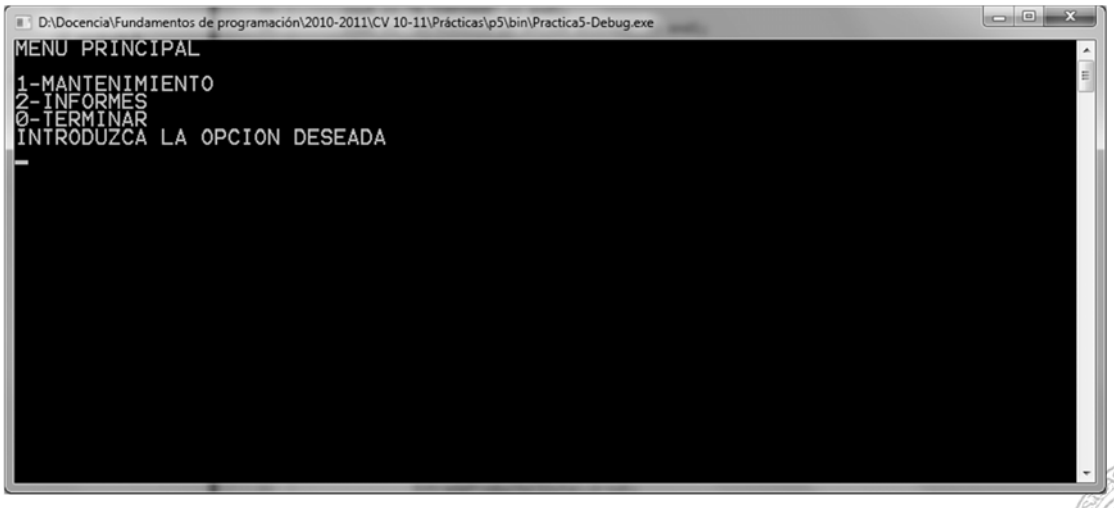
`endl` → *end line*

El dispositivo de salida

Pantalla en modo texto

➔ Líneas de 80 caracteres (textos)

Aplicación en modo texto

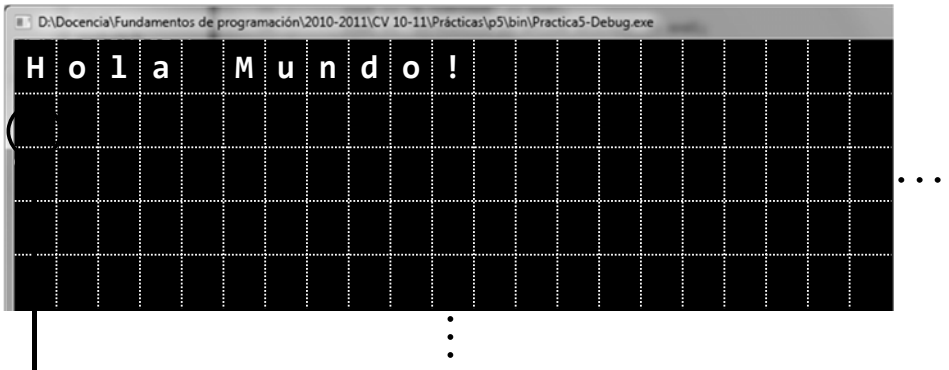


El dispositivo de salida

Ventanas de consola o terminal

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 textos en la pantalla de modo texto

Representación textual de los datos

A partir de la posición del cursor

Line wrap (continúa en la siguiente línea si no cabe)

Se pueden encadenar:

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

Recuerda: las instrucciones terminan en `;`

Luis Hernández Yáñez

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!
- ✓ Números literales
  Con o sin decimales, con signo o no: `123`, `-37`, `3.1416`, ...
  ```
  cout << "Pi = " << 3.1416;
  ```
  Se muestran los caracteres que representan el número
  ¡Punto decimal, NO coma!
- ✓ `endl`

Luis Hernández Yáñez

El primer programa en C++

El programa principal

La función `main()`: *donde comienza la ejecución...*

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

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

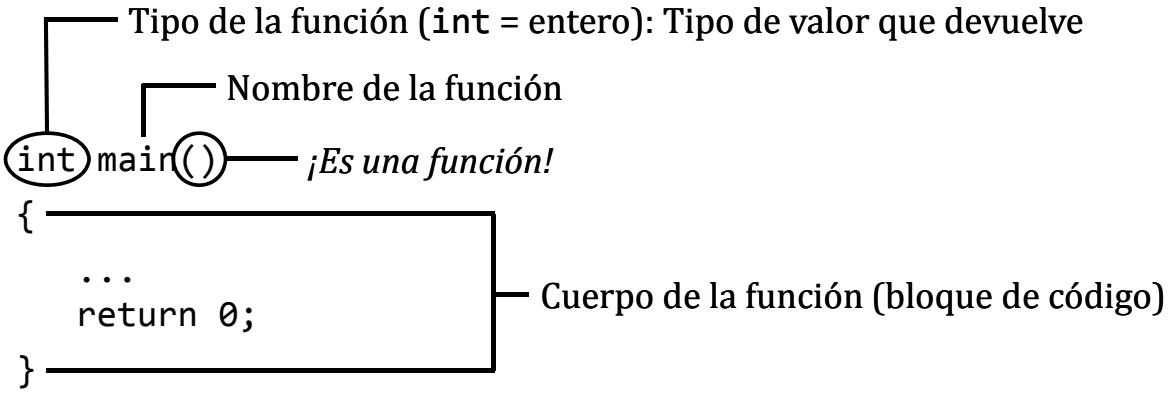
Contiene las instrucciones que hay que ejecutar

Luis Hernández Yáñez

El primer programa en C++

El programa principal

La función `main()`:



`return 0;` Devuelve el resultado (`0`) de la función

Luis Hernández Yáñez

El primer programa en C++

Documentando el código...

Comentarios (se ignoran):

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

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

Hasta el final de la línea: `// Comentario de una línea`

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

Luis Hernández Yáñez

El primer programa en C++

La infraestructura

Código para reutilizar:

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

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

← Una directiva: empieza por #

Bibliotecas de funciones a nuestra disposición

Luis Hernández Yáñez

El primer programa en C++

Bibliotecas

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

```
#include <iostream>
```

(Utilidades de entrada/salida por consola)

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

Espacios de nombres

En `iostream` hay espacios de nombres; ¿cuál queremos?

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

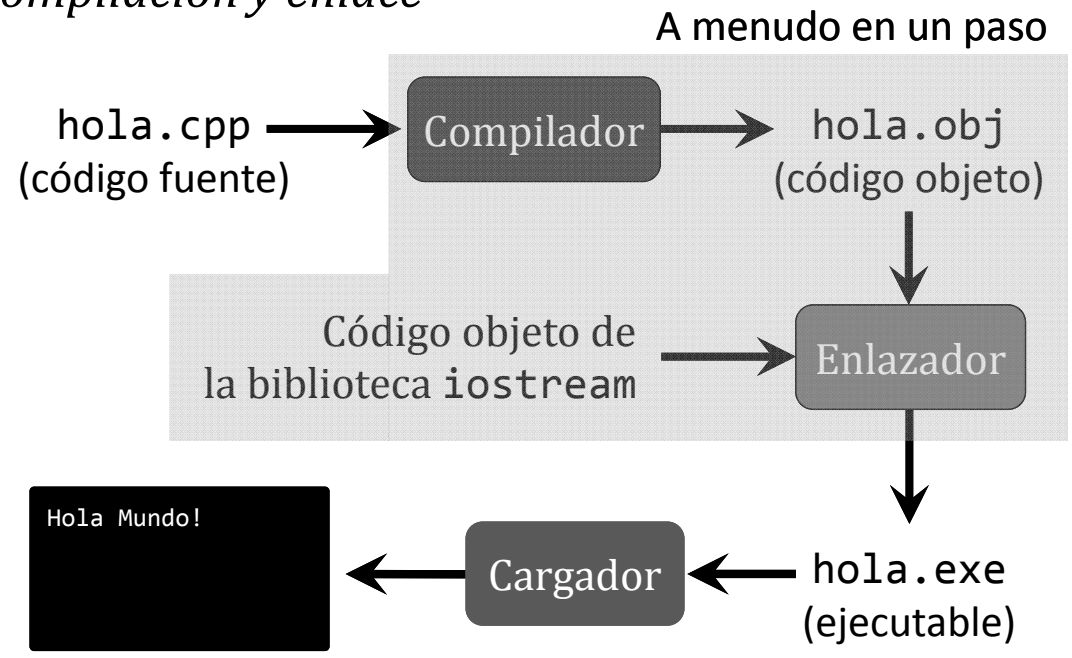
← Es una instrucción: termina en `;`

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

Muchas bibliotecas no tienen espacios de nombres

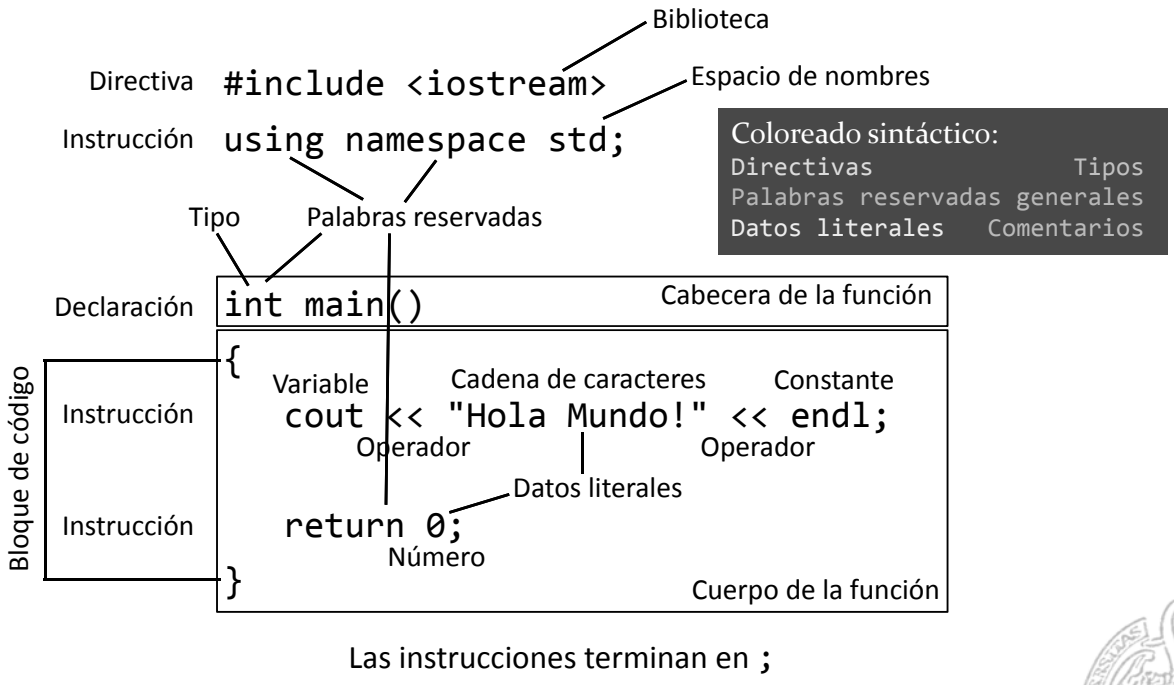
El primer programa en C++

Compilación y enlace



El primer programa en C++

Elementos del programa



Las instrucciones terminan en ;

El primer programa en C++

Uso de espacio en blanco

Separación de elementos por uno o más *espacios en blanco* (espacios, tabuladores y saltos de línea)

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?

Las líneas de código del programa

Luis Hernández Yáñez

Programa mínimo

Programa con E/S por consola

Una plantilla para empezar:

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

int main()
{
   
   return 0;
}
```

¡Tu código aquí!

Luis Hernández Yáñez

BY NC SA

El Quijote...

... recitado en la consola

Mostrar los textos con `cout <<`:

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

int main()
{
   cout << "En un lugar de la Mancha," << endl;
   cout << "de cuyo nombre no quiero acordarme," << endl;
   cout << "no ha mucho tiempo que vivía un hidalgo de los de
lanza en astillero, ..." << endl;
   return 0;
}
```


Líneas de código

Introducción del código del programa

Terminamos cada línea de código con un salto de línea (↲):

```
#include <iostream>↲
using namespace std;↲
↲
int main()↲
{↲
   cout << "En un lugar de la Mancha," << endl;↲
   cout << "de cuyo nombre no quiero acordarme," << endl;↲
   cout << "no ha mucho tiempo que vivía un hidalgo de los de
lanza en astillero, ..." << endl;↲
   return 0;↲
}↲
```


Líneas de código

Introducción del código del programa

No hay que partir una cadena literal entre dos líneas:

```
cout << "no ha mucho tiempo que vivía un hidalgo de↲
los de lanza en astillero, ..." << endl;↲
```

¡La cadena no termina (1ª línea)!

¡No se entiende `los` *(2ª línea)!*

Veamos cómo nos muestra los errores el compilador...

Programar pensando en posibles cambios

Mantenimiento y reusabilidad

- ✓ Usa espacio en blanco para separar los elementos:

```
cout << "En un lugar de la Mancha," << endl;
```

mejor que

```
cout<<"En un lugar de la Mancha,"<<endl;
```

- ✓ Usa sangría (indentación) para el código de un bloque:

```
{
    cout << "En un lugar de la Mancha," << endl;
    ...
    return 0;
}
```

Tab ó 3 esp.

¡El estilo importa!

Cálculos en los programas

Luis Hernández Yáñez

Cálculos en los programas

Operadores aritméticos

- `+` Suma
- `-` Resta
- `*` Multiplicación
- `/` División

Operadores binarios

`operando_izquierdo operador operando_derecho`

Operación	Resultado
`3 + 4`	`7`
`2.56 - 3`	`-0.44`
`143 * 2`	`286`
`45.45 / 3`	`15.15`

Luis Hernández Yáñez

BY NC SA

Cálculos en los programas

Números literales (concretos)

- ✓ Enteros: sin parte decimal

  Signo negativo (opcional) + secuencia de dígitos

  `3  143  -12  67321  -1234`

  No se usan puntos de millares

- ✓ Reales: con parte decimal

  Signo negativo (opcional) + secuencia de dígitos
  + punto decimal + secuencia de dígitos

  `3.1416  357.0  -1.333  2345.6789  -404.1`

Punto decimal (3.1416), NO coma (~~3,1416~~)

Cálculos en los programas

cálculos.cpp

Ejemplo

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

int main()
{
   cout << "133 + 1234 = " << 133 + 1234 << endl;
   cout << "1234 - 111.5 = " << 1234 - 111.5 << endl;
   cout << "34 * 59 = " << 34 * 59 << endl;
   cout << "3.4 * 5.93 = " << 3.4 * 5.93 << endl;
   cout << "500 / 3 = " << 500 / 3 << endl; // Div. entera
   cout << "500.0 / 3 = " << 500.0 / 3 << endl; // Div. real

   return 0;
}
```

Un texto

Un número

Cálculos en los programas

```
D:\FP\Tema02>g++ -o cálculos cálculos.cpp
Información: se resuelve std::cout  al enlaza
ación)
c:/mingw/bin/../lib/gcc/mingw32/4.5.0/../../.
importación automática se activó sin especifi
a de órdenes.
Esto debe funcionar a menos que involucre est
ferencíen símbolos de DLLs auto-importadas.

D:\FP\Tema02>cálculos
133 + 1234 = 1367
1234 - 111.5 = 1122.5
34 * 59 = 2006
3.4 * 5.93 = 20.162
500 / 3 = 166
500.0 / 3 = 166.667
```

División entera

División real

Cálculos en los programas

¿División entera o división real?

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 siempre que el tipo de división sea el que quieres

Variables

Luis Hernández Yáñez

Variables

Datos que se mantienen en memoria

Variable: dato que se accede por medio de un nombre

Dato literal: un valor concreto

Variable: puede cambiar de valor (*variar*)

```
edad = 19; // variable edad y literal 19
```

Las variables deben ser declaradas

¿Qué tipo de dato queremos mantener?

- ✓ Valor numérico sin decimales (entero): tipo `int`
- ✓ Valor numérico con decimales (real): tipo `double`

Declaración: *`tipo nombre;`*

Luis Hernández Yáñez

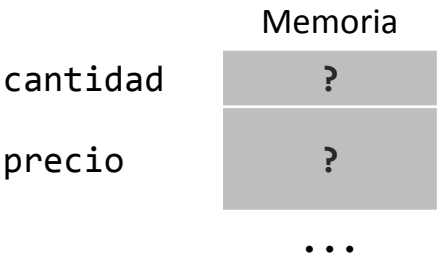
Variables

Declaración de variables

tipo nombre;

```
int cantidad;
double precio;
```

Se reserva espacio suficiente



LAS VARIABLES NO SE INICIALIZAN

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

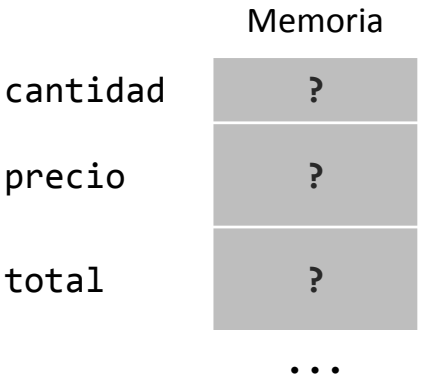
¿Dónde colocamos las declaraciones?

Siempre, antes del primer uso
Habitualmente al principio de la función

Variables

Declaración de variables

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

int main()
{
   int cantidad;
   double precio, total;



   return 0;
}
```



Podemos declarar varias de un mismo tipo separando los nombres con comas

Variables

Capacidad de las variables

`int`

-2.147.483.648 ... 2.147.483.647

`-2147483648 .. 2147483647`

`double`

$2{,}23 \times 10^{-308}$ ... $1{,}79 \times 10^{+308}$ y sus negativos

`[+|-] 2.23e-308 .. 1.79e+308` ← Notación científica

Problemas de precisión

Luis Hernández Yáñez

Variables

Asignación de valores a las variables (operador =)

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

```
cantidad = 12; // int
precio = 39.95; // double
total = cantidad * precio; // Asigna 479.4
```

cantidad ← 12

Concordancia de tipos: ~~`cantidad = 12.5;`~~

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

Luis Hernández Yáñez

Expresiones

Luis Hernández Yáñez

Expresiones

Secuencias de operandos y operadores

operando operador operando operador operando ...

```
total = cantidad * precio * 1.18;
```

Expresión

A igual prioridad se evalúan de izquierda a derecha

Paréntesis para forzar ciertas operaciones

```
total = cantidad1 + cantidad2 * precio;
total = (cantidad1 + cantidad2) * precio;
```

≠

Unos operadores se evalúan antes que otros

Luis Hernández Yáñez

Expresiones

Precedencia de los operadores

```
cantidad1 = 10;
cantidad2 = 2;
precio = 40.0;
```

* y / se evalúan antes que + y -

```
total = cantidad1 + cantidad2 * precio;
```

* antes que + → 10 + 2 * 40,0 → 10 + 80,0 → 90,0

```
total = (cantidad1 + cantidad2) * precio;
```

+ antes que * → (10 + 2) * 40,0 → 12 * 40,0 → 480,0

Variables y expresiones

variables.cpp

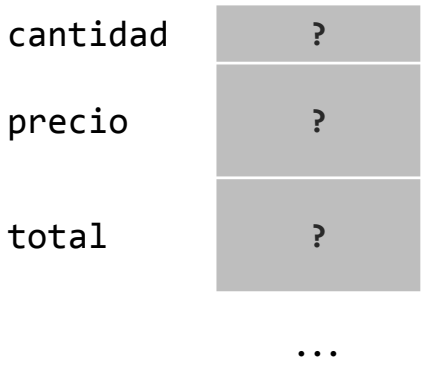
Ejemplo de uso de variables y expresiones

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

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

   return 0;
}
```


Variables y expresiones

Ejemplo de uso de variables

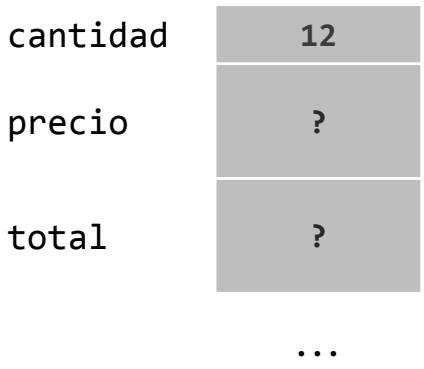
```
#include <iostream>
using namespace std;

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



Variables y expresiones

Ejemplo de uso de variables

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

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



Variables y expresiones

Ejemplo de uso de variables

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

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

Memoria

cantidad	12
precio	39.95
total	?
	...

Luis Hernández Yáñez

Variables y expresiones

Ejemplo de uso de variables

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

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

Memoria

cantidad	12
precio	39.95
total	479.4
	...

Luis Hernández Yáñez

Variables y expresiones

Ejemplo de uso de variables

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

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

	Memoria
cantidad	12
precio	39.95
total	479.4
	...

```
D:\FP\Tema2>variables
12 x 39.95 = 479.4
```


Variables y expresiones

Ejemplo de uso de variables

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

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

   return 0;
}
```

```
Símbolo del sistema

D:\FP\Tema2>g++ -o variables variables.cpp
Información: se resuelve std::cout  al enlazar
ación)
c:/mingw/bin/../lib/gcc/mingw32/4.5.0/../../..
importación automática se activó sin especific
a de órdenes.
Esto debe funcionar a menos que involucre estr
ferencíen símbolos de DLLs auto-importadas.

D:\FP\Tema2>variables
12 x 39.95 = 479.4

D:\FP\Tema2>_
```


Lectura de datos desde el teclado

Luis Hernández Yáñez

Valores proporcionados por el usuario

`cin` (`iostream`) *character* `in`*put stream*

Lectura de valores de variables: operador `>>` (*extractor*)

```
cin >> cantidad;
```

Memoria

cantidad	12
	...


```
1 2
_
```

Luis Hernández Yáñez

Valores proporcionados por el usuario

El extractor >>

```
cin >> variable;
```

Transforma los caracteres introducidos en datos

Cursor parpadeante: lugar de lectura del siguiente carácter

La entrada termina con Intro (cursor a la siguiente línea)

¡El destino del extractor debe ser SIEMPRE una variable!

Se ignoran los espacios en blanco iniciales

Valores proporcionados por el usuario

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

Recomendación: Lee cada variable en una línea `12↲`

Lectura de valores reales (`double`*)*

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

Recomendación: Lee cada variable en una línea `39.95↲`

Valores proporcionados por el usuario

¿Qué pasa si el usuario se equivoca?

El dato no será correcto

Aplicación profesional: código de comprobación y ayuda

Aquí supondremos que los usuarios no se equivocan

En ocasiones añadiremos comprobaciones sencillas


Para evitar errores, lee cada dato en una instrucción aparte

Valores proporcionados por el usuario

¿Qué pasa si el usuario se equivoca?

```
int cantidad;
double precio, total;
cout << "Introduce la cantidad: ";
cin >> cantidad;
cout << "Introduce el precio: ";
cin >> precio;
cout << "Cantidad: " << cantidad << endl;
cout << "Precio: " << precio << endl;
```

¡Amigable con el usuario!
¿Qué tiene que introducir?

```
Introduce la cantidad: abc
Introduce el precio: Cantidad: 0
Precio: 1.79174e-307
```

No se puede leer un entero → 0 para `cantidad` y Error
La lectura del precio falla: `precio` no toma valor (*basura*)

Valores proporcionados por el usuario

¿Qué pasa si el usuario se equivoca?

```
Introduce la cantidad: 12abc
Introduce el precio: Cantidad: 12
Precio: 0
```

12 para `cantidad`
No se puede leer un real
→ 0 para `precio` y Error

```
Introduce la cantidad: 12.5abc
Introduce el precio: Cantidad: 12
Precio: 0.5
```

12 para `cantidad`
`.5` → 0,5 para `precio`
Lo demás queda pendiente

```
Introduce la cantidad: 12
Introduce el precio: 39.95
Cantidad: 12
Precio: 39.95
```

¡¡¡Lectura correcta!!!

Programa con lectura de datos

División de dos números

Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo

Algoritmo.-

Datos / cálculos

1. Pedir el `numerador`

Variable `numerador` (`double`)

2. Pedir el `denominador`

Variable `denominador` (`double`)

3. Realizar la división, guardando el `resultado`

Variable `resultado` (`double`)

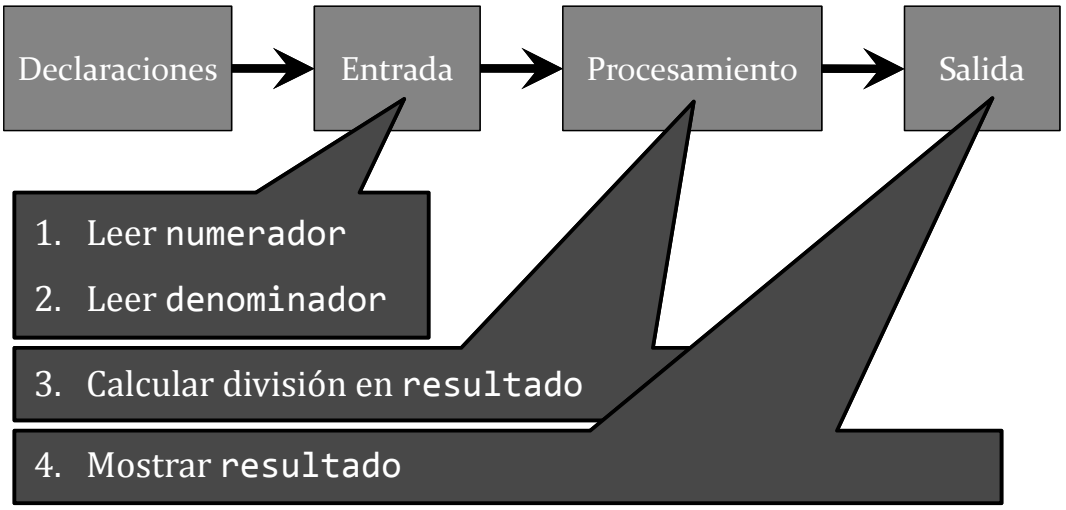
```
resultado = numerador / denominador
```

4. Mostrar el `resultado`

Un esquema general

Entrada-Proceso-Salida

Muchos programas se ajustan a un sencillo esquema:



Programa con lectura de datos

División de dos números

Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo.

1. Leer `numerador`

```
cin >> numerador;
```

2. Leer `denominador`

```
cin >> denominador;
```

3. Calcular división en `resultado`

```
resultado = numerador / denominador;
```

4. Mostrar `resultado`

```
cout << resultado;
```


Programa con lectura de datos Implementación

División de dos números

división.cpp

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

int main()
{
   double numerador, denominador, resultado;
   cout << "Numerador: ";
   cin >> numerador;
   cout << "Denominador: ";
   cin >> denominador;
   resultado = numerador / denominador;
   cout << "Resultado: " << resultado << endl;
   return 0;
}
```

```
Numerador: 129
Denominador: 2
Resultado: 64.5
```

Declaraciones	`double numerador, denominador, resultado;`

Entrada	`cout << "Numerador: ";`
	`cin >> numerador;`
	`cout << "Denominador: ";`
	`cin >> denominador;`

Procesamiento	`resultado = numerador / denominador;`
Salida	`cout << "Resultado: " << resultado << endl;`

Fundamentos de la programación

Resolución de problemas

Resolución de problemas

Análisis / Diseño

Problema

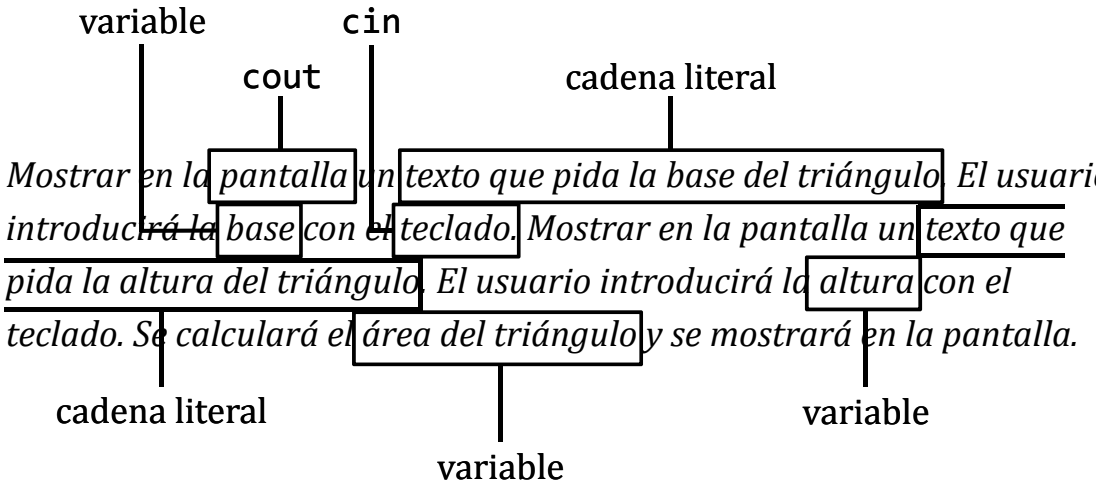
Dadas la base y la altura de un triángulo, mostrar su área

Refinamiento

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

Resolución de problemas

Objetos: Datos que maneja el programa



Resolución de problemas

Datos que maneja el programa: tipos

Objeto	*Tipo*	*¿Varía?*	*Nombre*
Pantalla		Variable	`cout`
"Introduzca la base del triángulo: "		Constante	*ninguno*
Base del triángulo	`double`	Variable	`base`
Teclado		Variable	`cin`
"Introduzca la altura del triángulo: "		Constante	*ninguno*
Altura del triángulo	`double`	Variable	`altura`
Área del triángulo	`double`	Variable	`area`

Resolución de problemas

Operaciones (acciones)

```
cout << ...                    cin >> ...
```

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.

```
area = base * altura / 2
```


El algoritmo

Secuencia de acciones que ha de realizar el programa para conseguir resolver el problema

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo

Luis Hernández Yáñez

El programa

```
#include <iostream>
using namespace std;
int main()
{
   Declaraciones
   Algoritmo traducido a código en C++
   return 0;
}
```

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura del triángulo
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo

Luis Hernández Yáñez

triángulo.cpp

El programa: implementación

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

int main()
{
  double base, altura, area;                  // Declaraciones
  cout << "Introduzca la base del triángulo: ";          // 1
  cin >> base;                                           // 2
  cout << "Introduzca la altura del triángulo: ";        // 3
  cin >> altura;                                         // 4
  area = base * altura / 2;                              // 5
  cout << "El área de un triángulo de base " << base     // 6
       << " y altura " << altura << " es: " << area << endl;

  return 0;
}
```

```
D:\FP\Tema02>triángulo
Introduzca la base del trißngulo: 34.7
Introduzca la altura del trißngulo: 12
El ßrea de un trißngulo de base 34.7 y altura 12 es: 208.2
```

¿triβngulo?

Recuerda: las instrucciones terminan en ;

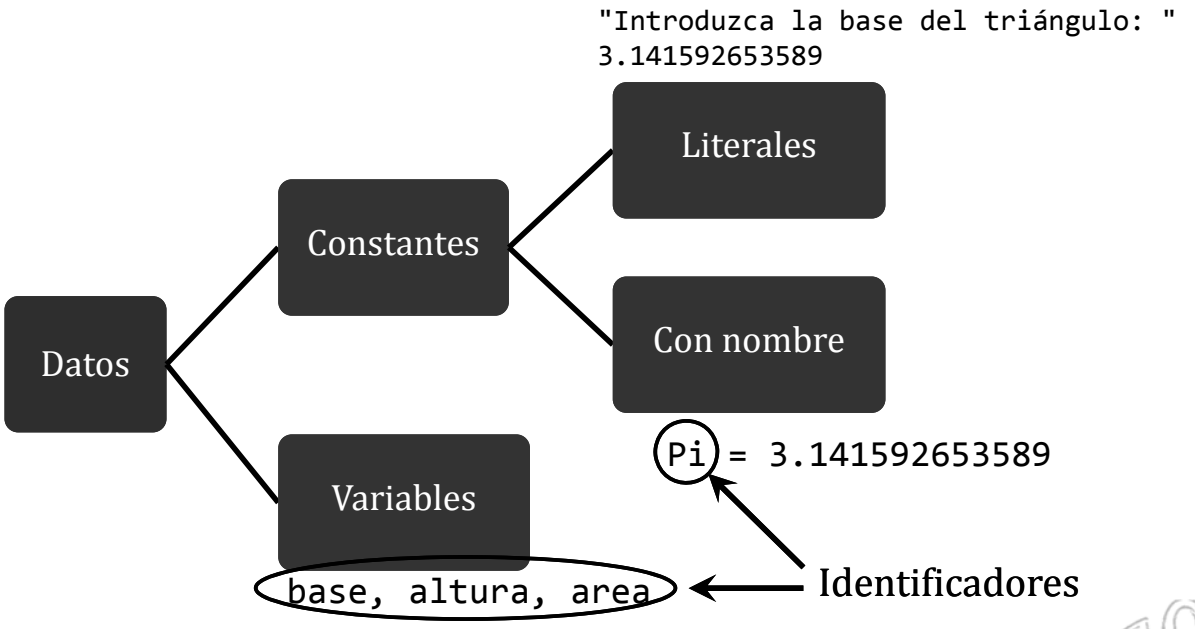
Luis Hernández Yáñez

Los datos de los programas

Luis Hernández Yáñez

Los datos de los programas

Variabilidad de los datos



Fundamentos de la programación

Identificadores

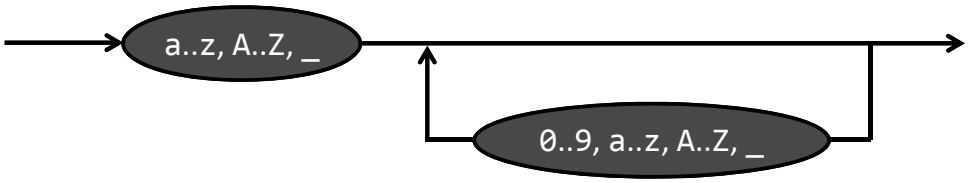
Identificadores

≠ palabras reservadas

Para variables y constantes con nombre

- *Nombre* de un dato (para accederlo/modificarlo)
- Deben ser descriptivos

Sintaxis:




```
cantidad  prrecio  total  base  altura  area  numerador
```

Al menos 32 caracteres significativos

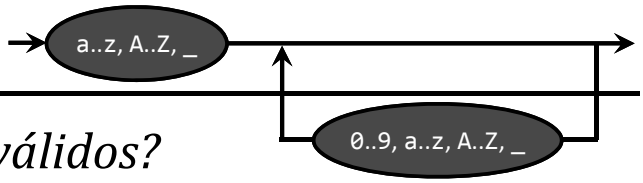
¡Ni eñes ni vocales acentuadas!

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?

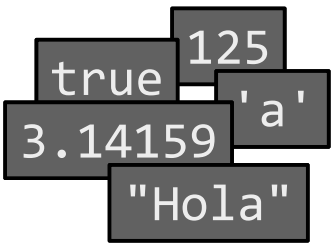
`balance`	✓	`interesAnual`	✓
`_base_imponible`	✓	`años`	✗
`EDAD12`	✓	`salario_1_mes`	✓
`__edad`	✓	`cálculoNómina`	✗
`valor%100`	✗	`AlgunValor`	✓
`100caracteres`	✗	`valor?`	✗
`_12_meses`	✓	`____valor`	✓

Fundamentos de la programación

Tipos de datos

Tipos de datos

Tipos



Cada dato, de un tipo concreto

Cada tipo establece:

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

Expresiones con datos de distintos tipos (compatibles):
Transformación automática de tipos (*promoción de tipo*)

Anexo del Tema 2: detalles técnicos

Tipos de datos básicos

`int`
Números enteros (sin decimales) `1363, -12, 49` ✓

`float`
Números reales `12.45, -3.1932, 1.16E+02`

`double`
Números reales (mayores intervalo y precisión) ✓

`char`
Caracteres `'a', '{', '\t'`

`bool`
Valores lógicos (verdadero/falso) `true, false`

`string`
Cadenas de caracteres (biblioteca `string`) `"Hola Mundo!"`

`void`
Nada, ausencia de tipo, ausencia de dato (*funciones*)

char

Caracteres

Intervalo de valores: Juego de caracteres (ASCII)

1 byte

Literales:

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

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

Caracteres de control

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

ASCII (códigos 32..127)

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

bool

Valores lógicos

Sólo dos valores posibles:

- Verdadero (*true*)
- Falso (*false*)

Literales:

```
true   false
```

Cualquier número distinto de `0` es equivalente a `true`

El `0` es equivalente a `false`

Mayúsculas y minúsculas

C++ distingue entre mayúsculas y minúsculas

`int`: palabra reservada de C++ para declarar datos enteros

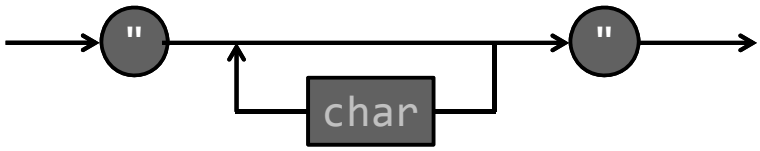
`Int`, `INT` o `inT` no son palabras reservadas de C++

`true`: palabra reservada de C++ para el valor *verdadero*

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

`string` *Cadenas de caracteres*

```
"Hola"  "Introduce el numerador: "  "X142FG5TX?%A"
```



Secuencias de caracteres

Programas con variables de tipo `string`:

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

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

Tipos de datos básicos: ejemplo

tipos.cpp

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

int main()
{
   int entero = 3; // Podemos asignar (inicializar) al declarar
   double real = 2.153;
   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>
```

¿Cuántos números hay en total en el programa?
¿Y caracteres? ¿Y cadenas? ¿Y booleanos?

Modificadores de tipos

- `signed` / `unsigned` : con signo (por defecto) / sin signo
- `short` / `long` : menor / mayor 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`

Declaración y uso de variables

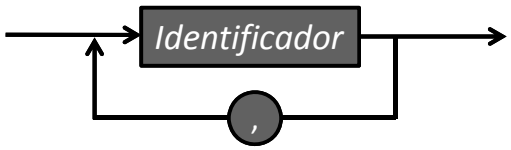
Luis Hernández Yáñez

Declaración de variables

```
[modificadores] tipo lista_de_variables;
```

Opcional

lista_de_variables




```
int i, j, l;
short int unidades;
unsigned short int monedas;
double balance, beneficio, perdida;
```

Programación con buen estilo:
Identificadores descriptivos
Espacio tras cada coma
Nombres de las variables en minúsculas
(Varias palabras: capitaliza cada inicial: `interesPorMes`)

Luis Hernández Yáñez

Datos y memoria

Se reserva memoria suficiente para cada tipo de dato

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

Inicialización de variables

¡En C++ las variables no se inicializan automáticamente!

¡Una variable debe ser haber sido inicializada antes de ser accedida!

¿Cómo se inicializa una variable?

- Al leer su valor (`cin >>`)
- Al asignarle un valor (instrucción de asignación)
- Al declararla

Inicialización en la propia declaración:


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

En particular, una expresión puede ser un literal

Uso de las variables

Obtención del valor de una variable

- ✓ Nombre de la variable en una expresión

```
cout << balance;
cout << interesPorMes * meses / 100;
```

Modificación del valor de una variable

- ✓ Nombre de la variable a la izquierda del =

```
balance = 1214;
porcentaje = valor / 30;
```

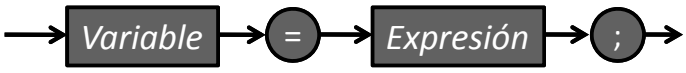
Las variables han de haber sido previamente declaradas

Fundamentos de la programación

Instrucciones de asignación

Instrucciones de asignación

El operador =



A la izquierda, SIEMPRE una variable

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

Luis Hernández Yáñez

Instrucciones de asignación

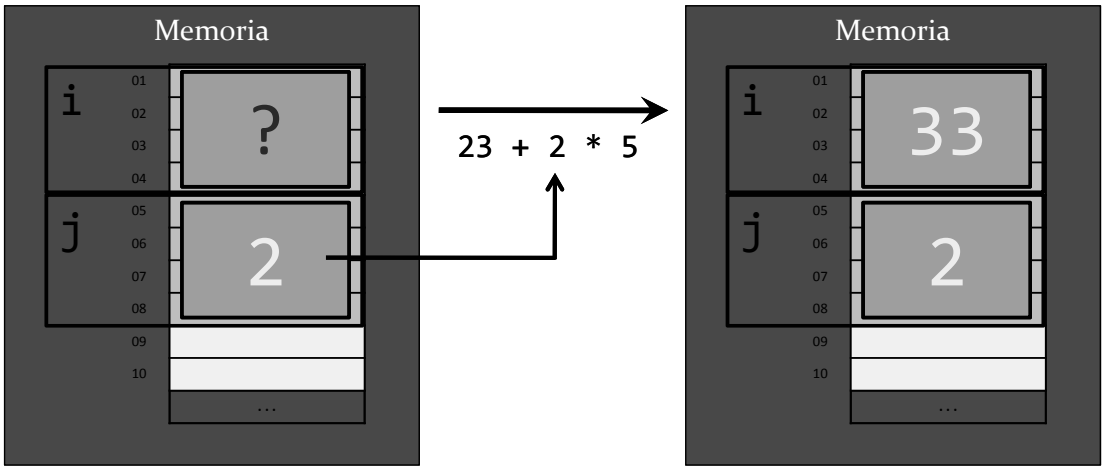
Errores

```
int a, b, c;
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
c = 23  5;
   // ERROR: expresión no válida (falta operador)
```

Luis Hernández Yáñez

Variables, asignación y memoria

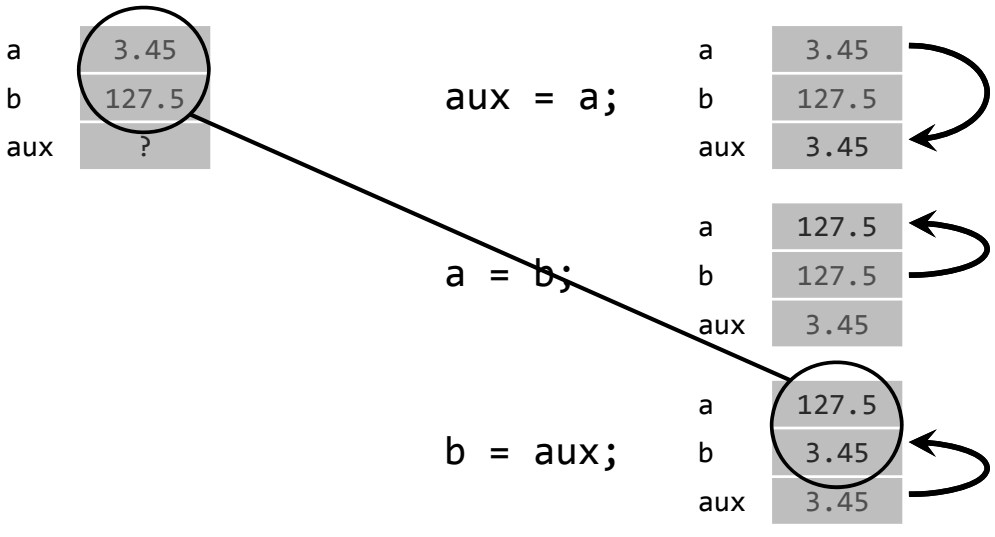
```
int i, j = 2;
i = 23 + j * 5;
```



Ejemplo: Intercambio de valores

Necesitamos una variable auxiliar

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



Operadores

Luis Hernández Yáñez

Operadores

Operaciones sobre valores de los tipos

Cada tipo determina las operaciones posibles

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

- Asignación (=)
- Operadores aritméticos
- Operadores relacionales (menor, mayor, igual, ...)

Tipo de datos `bool`:

- Asignación (=)
- Operadores lógicos (Y, O, NO)

Tipos de datos `char` y `string`:

- Asignación (=)
- Operadores relacionales (menor, mayor, igual, ...)

Luis Hernández Yáñez

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	Div. entera	División real
%	2 (binario)	Infijo	Módulo	No aplicable
++	1 (monario)	Prefijo / postfijo	Incremento	
--	1 (monario)	Prefijo / postfijo	Decremento	

Operadores aritméticos

Operadores monarios y operadores binarios

Operadores monarios (*unarios*)

— Cambio de signo (`-`):

Delante de variable, constante o expresión entre paréntesis

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

— Incremento/decremento (sólo variables) (prefijo/postfijo):

```
++interes      --meses      j++      // 1 más ó 1 menos
```

Operadores binarios

— Operando izquierdo operador operando derecho
Operandos: literales, constantes, variables o expresiones

```
2 + 3      a * RATIO      -a + b
(a % b) * (c / d)
```


Operadores aritméticos

¿División entera o división real?

/

Ambos operandos enteros: división entera

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

Algún operando real: división real

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

Luis Hernández Yáñez

Operadores aritméticos

Módulo (resto de la división entera)

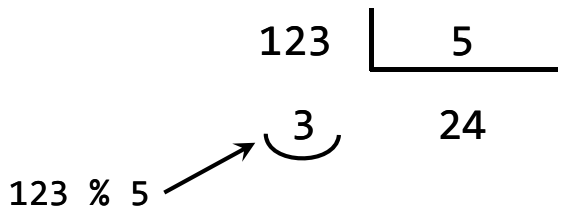
%

Ambos operandos han de ser enteros

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

División entera:

No se obtienen decimales → Queda un resto



Luis Hernández Yáñez

Operadores aritméticos

Operadores de incremento y decremento

++/--

Incremento/decremento de la variable numérica en una unidad

Prefijo: Antes de acceder

```
int i = 10, j;
j = ++i; // Incrementa antes de copiar
cout << i << " - " << j; // Muestra 11 - 11
```

```
i=i+1;
j=i;
```

Postfijo: Después de acceder

```
int i = 10, j;
j = i++; // Copia y después incrementa
cout << i << " - " << j; // Muestra 11 - 10
```

```
j=i;
i=i+1;
```

No mezcles ++ y -- con otros operadores

Operadores aritméticos: ejemplo

operadores.cpp

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

int main() {
   int entero1 = 15, entero2 = 4;
   double real1 = 15.0, real2 = 4.0;
   cout << "Operaciones entre los números 15 y 4:" << endl;
   cout << "División entera (/): " << entero1 / entero2 << endl;
   cout << "Resto de la división (%): " << entero1 % entero2 << endl;
   cout << "División real (/): " << real1 / real2 << endl;
   cout << "Num = " << real1 << endl;
   real1 = -real1;
   cout << "Cambia de signo (-): " << real1 << endl;
   real1 = -real1;
   cout << "Vuelve a cambiar (-): " << real1 << endl;
   cout << "Se incrementa antes (++ prefijo): " << ++real1 << endl;
   cout << "Se muestra antes de incrementar (posfijo ++): "
        << real1++ << endl;
   cout << "Ya incrementado: " << real1 << endl;
   return 0;
}
```


Más sobre expresiones

Luis Hernández Yáñez

Orden de evaluación

¿En qué orden se evalúan los operadores?

```
3 + 5 * 2 / 2 - 1
```

¿De izquierda a derecha?

¿De derecha a izquierda?

¿Unos antes que otros?

Precedencia de los operadores (prioridad):

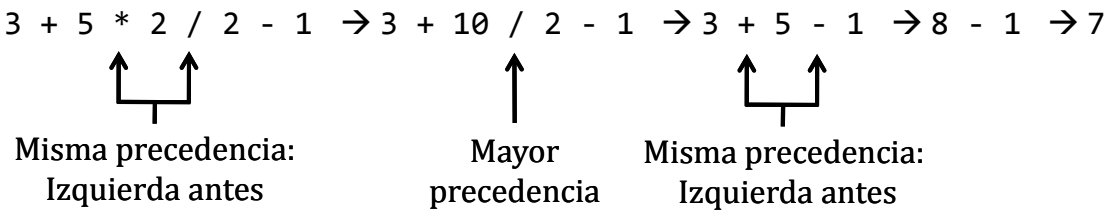
- Se evalúan antes los de mayor precedencia
- ¿Y si tienen igual prioridad?
  - Normalmente, de izquierda a derecha

Paréntesis: fuerzan a evaluar su subexpresión

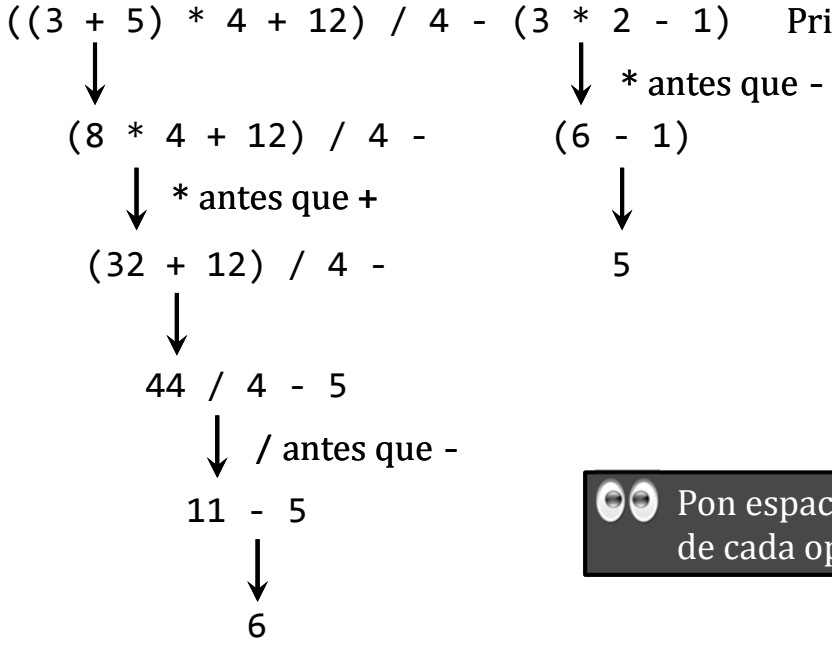
Luis Hernández Yáñez

Precedencia de los operadores

Precedencia	Operadores
Mayor prioridad	`++ --` (postfijos)
	`++ --` (prefijos)
	`-` (cambio de signo)
	`* / %`
Menor prioridad	`+ -`



Evaluación de expresiones



Pon espacio antes y después de cada operador binario

Una fórmula

fórmula.cpp

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

```
#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;
}
```

Usa paréntesis para mejorar la legibilidad:

```
f = (3 * x * x / 5) + (6 * x / 7) - 3;
```


Abreviaturas aritméticas

variable = variable operador op_derecho;

La misma ≡

variable operador= op_derecho;

Asignación	*Abreviatura*
`a = a + 12;`	`a += 12;`
`a = a * 3;`	`a *= 3;`
`a = a - 5;`	`a -= 5;`
`a = a / 37;`	`a /= 37;`
`a = a % b;`	`a %= b;`

Igual precedencia que la asignación

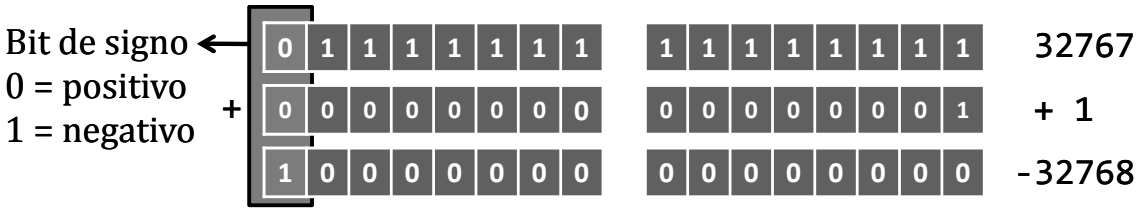
De momento, mejor evitarlas

Desbordamiento

¿Valor siguiente al máximo?

Valor mayor del máximo (o menor del mínimo) del tipo

```
short int i = 32767; // Valor máximo para short int
i++; // 32768 no cabe en un short int
cout << i; // Muestra -32768
```



Fundamentos de la programación

Constantes

Constantes

Declaración de constantes Modificador de acceso `const`

Variables inicializadas a las que no dejamos variar


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

La constante no podrá volver a aparecer a la izquierda de un =

Programación con buen estilo:
Pon en mayúscula la primera letra de una constante o todo su nombre

¿Por qué utilizar constantes con nombre?

- ✓ Aumentan la legibilidad del código

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

- ✓ Facilitan la modificación del código

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

3 cambios

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

¿Cambio del IVA al 21%?

1 cambio

Constantes: ejemplo

constantes.cpp

```
#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 círculo de radio "
        << radio << ": " << circunferencia << endl;
   const double Euler = 2.718281828459; // Número e
   cout << "Número e al cuadrado: " << Euler * Euler << endl;
   const int IVA = 21;
   int cantidad = 12;
   double precio = 39.95, neto, porIVA, total;
   neto = cantidad * precio;
   porIVA = neto * IVA / 100;
   total = neto + porIVA;
   cout << "Total compra: " << total << endl;
   return 0;
}
```


Fundamentos de la programación

La biblioteca cmath

Funciones matemáticas

`#include <cmath>`

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

La biblioteca cmath

`mates.cpp`

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

int main() {
  double x, y, f;
  cout << "Valor de X: ";
  cin >> x;
  cout << "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;
}
```

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

`pow()` con argumento entero:

Usa el molde `double()`:
`pow(double(i), 5)`

Pon un espacio detrás de cada coma en las listas de argumentos

Operaciones con caracteres

Luis Hernández Yáñez

Operaciones con caracteres `char`

Asignación, `++`/`--` y operadores relacionales

Funciones para caracteres (biblioteca `cctype`)

`isalnum(c)`	`true` si `c` es una letra o un dígito
`isalpha(c)`	`true` si `c` es una letra
`isdigit(c)`	`true` si `c` es un dígito
`islower(c)`	`true` si `c` es una letra minúscula
`isupper(c)`	`true` si `c` es una letra mayúscula
	`false` en caso contrario
`toupper(c)`	devuelve la mayúscula de `c`
`tolower(c)`	devuelve la minúscula de `c`

…

Luis Hernández Yáñez

cc BY NC SA

Operaciones con caracteres

caracteres.cpp

```
...
#include <cctype>

int main() {
   char caracter1 = 'A', caracter2 = '1', caracter3 = '&';
   cout << "Carácter 1 (" << caracter1 << ").-" << endl;
   cout << "Alfanumérico? " << isalnum(caracter1) << endl;
   cout << "Alfabético? " << isalpha(caracter1) << endl;
   cout << "Dígito? " << isdigit(caracter1) << endl;
   cout << "Mayúscula? " << isupper(caracter1) << endl;
   caracter1 = tolower(caracter1);
   cout << "En minúscula: " << caracter1 << endl;
   cout << "Carácter 2 (" << caracter2 << ").-" << endl;
   cout << "Alfabético? " << isalpha(caracter2) << endl;
   cout << "Dígito? " << isdigit(caracter2) << endl;
   cout << "Carácter 3 (" << caracter3 << ").-" << endl;
   cout << "Alfanumérico? " << isalnum(caracter3) << endl;
   cout << "Alfabético? " << isalpha(caracter3) << endl;
   cout << "Dígito? " << isdigit(caracter3) << endl;
   return 0;
}
```

$1 \equiv \texttt{true} \,/\, 0 \equiv \texttt{false}$

Luis Hernández Yáñez

Fundamentos de la programación

Operadores relacionales
(condiciones simples)

Luis Hernández Yáñez

Expresiones lógicas (*booleanas*)

Operadores relacionales

Comparaciones (*condiciones*)

Condición simple ::= Expresión Operador_relacional Expresión

Concordancia de tipo entre las expresiones

Resultado: `bool` (`true` o `false`)

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

Operadores (prioridad)
...
* / %
+ -
< <= > >=
== !=
= += -= *= /= %=

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
```

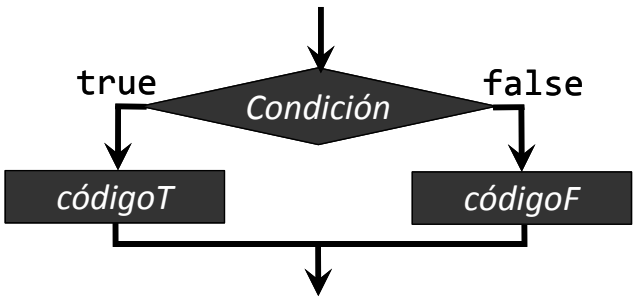
No confundas el operador de igualdad (==) con el operador de asignación (=)

Toma de decisiones (`if`)

Luis Hernández Yáñez

Hacer esto... o hacer esto otro...

Selección: bifurcación condicional




```
if (condición) {
⟼códigoT
}
else {
⟼códigoF
}
```

Opcional: puede no haber else

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

Luis Hernández Yáñez

La instrucción if

selección.cpp

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

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

```
D:\FP\Tema02>selección
1 - Sumar
2 - Restar
Opción: 1
17

D:\FP\Tema02>selección
1 - Sumar
2 - Restar
Opción: 2
9
```

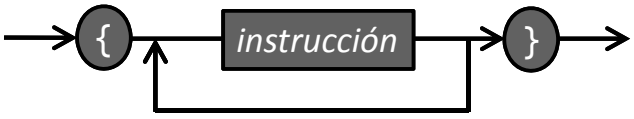

Fundamentos de la programación

Bloques de código

Bloques de código

Agrupación de instrucciones

Grupo de instrucciones a ejecutar en una rama del `if`




```
{
   intrucción1
   intrucción2
   ...
   intrucciónN
}
```

Tab ó 3 esp.

```
int num, total = 0;
cin >> num;
if (num > 0)
{
   cout << "Positivo";
   total = total + num;
}
cout << endl;
```

Ámbito local
(declaraciones locales)

Bloques de código

Posición de las llaves: cuestión de estilo

```
if (num > 0)
{
   cout << "Positivo";
   total = total + num;
}
cout << endl;
```

```
if (num > 0) {
   cout << "Positivo";
   total = total + num;
}
cout << endl;
```

No necesitamos las llaves si sólo hay una instrucción

```
if (num > 0) {
   cout << "Positivo";
}
```

≡

```
if (num > 0)
   cout << "Positivo";
```

Usaremos siempre llaves por simplicidad...

Evita poner el `if` y la instrucción objetivo en la misma línea:

```
if (num > 0) cout << "Positivo";
```

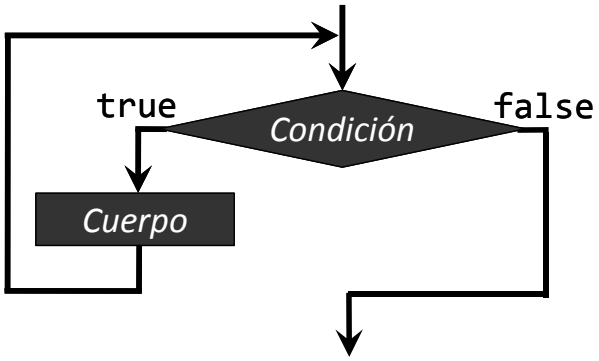

Bucles (`while`)

Luis Hernández Yáñez

Mientras la condición sea cierta, repetir...

Repetición o iteración condicional

Bloque de código




```
while (condición) {
⟷cuerpo
}
```

Si la condición es `false` al empezar, no se ejecuta el cuerpo ninguna vez

Luis Hernández Yáñez

La instrucción while

serie.cpp

$$\sum_{i=1}^{n} i$$

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

int main() {
   int i = 1, n = 0, suma = 0;
   while (n <= 0) { // Sólo n positivo
      cout << "¿Cuántos números quieres sumar? ";
      cin >> n;
   }
   while (i <= n) {
      suma = suma + i;
      i++;
   }
   cout << "Sumatorio de i (1 a " << n << ") = "
        << suma << endl;
   return 0;
}
```

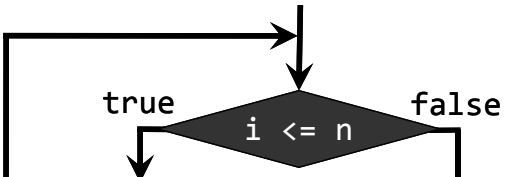
```
D:\FP\Tema02>serie
¿Cuántos números quieres sumar? -3
¿Cuántos números quieres sumar? 0
¿Cuántos números quieres sumar? 5
Sumatorio de i (1 a 5) = 15
```


La instrucción while

Iteración condicional

$$\sum_{i=1}^{n} i$$

```
while (i <= n) {
   suma = suma + i;
   i++;
}
```



n	5
i	6
suma	15

```
Sumatorio de i (1 a 5) = 15
```


Entrada/salida por consola

Luis Hernández Yáñez

Flujos de texto (streams)

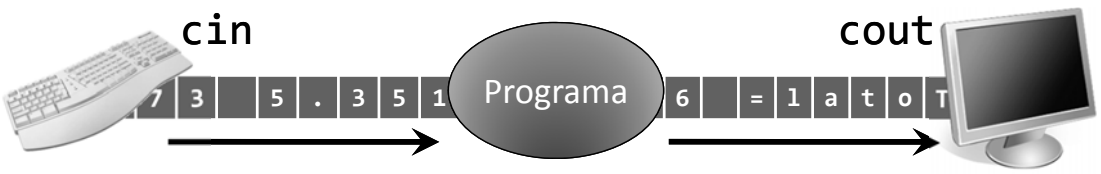
```
#include <iostream>
using namespace std;
```

Conectan la ejecución del programa con los dispositivos de E/S

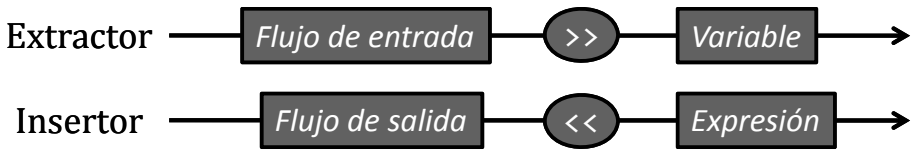
Son secuencias de caracteres

Entrada por teclado: flujo de entrada `cin` (tipo `istream`)

Salida por pantalla: flujo de salida `cout` (tipo `ostream`)

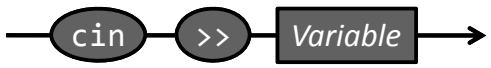


Biblioteca `iostream` con espacio de nombres `std`



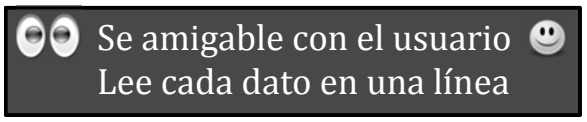
Luis Hernández Yáñez

Entrada por teclado



Salta los *espacios en blanco* (espacios, tabuladores o saltos de línea)

- `char`
  Se lee un carácter en la variable
- `int`
  Se leen dígitos y se transforman en el valor a asignar
- `float/double`:
  Se leen dígitos (quizá el punto y más dígitos) y se asigna el valor
- `bool`:
  Si se lee `1`, se asigna `true`; con cualquier otro valor se asigna `false`




```
cout << "Introduce tu edad: ";
cin >> edad;
```


Lectura de cadenas (`string`)

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

`cin >> cadena` termina con el primer espacio en blanco

`cin.sync()` descarta la entrada pendiente

```
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` recibe `"Antonio"`

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

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

¿Cómo leer varias palabras?

Siguiente página...

Entrada por teclado

Lectura sin saltar los espacios en blanco iniciales

Llamada a funciones con el operador punto (`.`) :

El operador punto permite llamar a una función sobre una variable

`variable.función(argumentos)`

Lectura de un carácter sin saltar espacios en blanco:

```
cin.get(c); // Lee el siguiente carácter
```

Lectura de cadenas sin saltar los espacios en blanco:

`getline(cin, cad);`

Lee todo lo que haya hasta el final de la línea (Intro)

Recuerda:

Espacios en blanco son espacios, tabuladores, saltos de línea, ...

Salida por pantalla

Representación textual de los datos

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

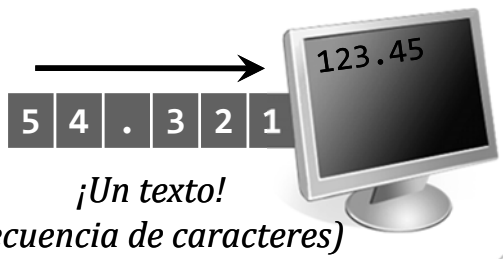
El valor `double 123.45` se guarda en memoria en binario

Su representación textual es: `'1' '2' '3' '.' '4' '5'`

```
double d = 123.45;
```

d 123.45 *¡Un número real!*

```
cout << d;
```



¡Un texto!
(secuencia de caracteres)

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

Salida por pantalla


```
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;
```

```
Total: 123.45
Meses: 7
```


Formato de la salida

`#include <iomanip>`

Constantes y funciones a enviar a `cout` para ajustar el formato de salida

Biblioteca	Constante/función	Propósito
`iostream`	`showpoint` / `noshowpoint`	Mostrar o no el punto decimal para reales sin decimales (`34.0`)
	`fixed`	Notación de punto fijo (reales) (`123.5`)
	`scientific`	Notación científica (reales) (`1.235E+2`)
	`boolalpha`	Valores `bool` como `true` / `false`
	`left` / `right`	Ajustar a la izquierda/derecha (por defecto)
`iomanip`	`setw(`*anchura*`)`*	Nº de caracteres (anchura) para el dato
	`setprecision(`*p*`)`	Precisión: Nº de dígitos (en total) Con `fixed` o `scientific`, nº de decimales

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

Formato de la salida

```
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 << endl;
cout << fixed << setprecision(8) << d << endl;
```

```
0->false


123.45x62
|  123.45|
|123.45  |
|x   |
|   62|

96 - 96.0000
1.234500e+002
123.45000000
```


Fundamentos de la programación

Funciones definidas por el programador

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!

Luis Hernández Yáñez

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
}
```

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

Luis Hernández Yáñez

Argumentos

Llamada a una función con parámetros

`Nombre(Argumentos)`

Al llamar a la función:

- 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)

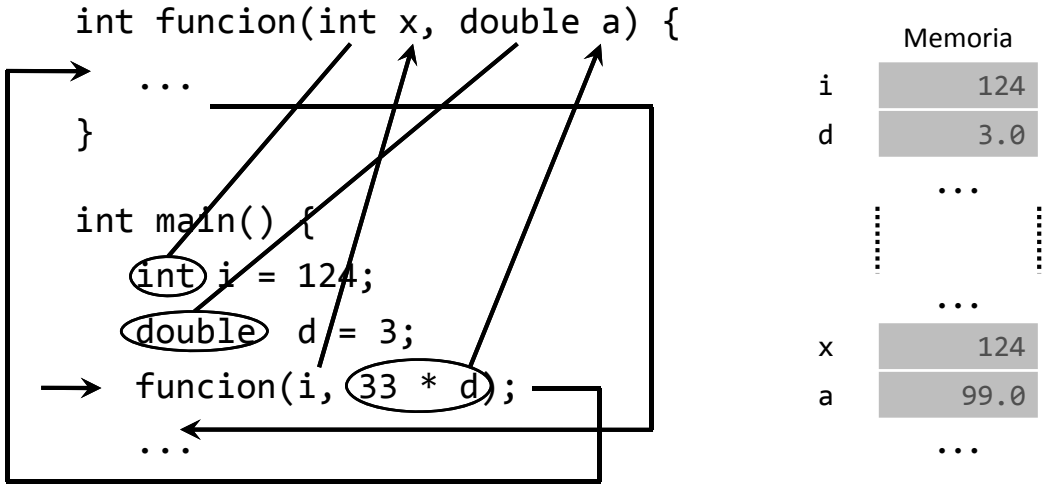
Se copian los valores resultantes de las expresiones en los correspondientes parámetros

Llamadas a la función: en expresiones de otras funciones

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


Paso de argumentos

Se copian los argumentos en los parámetros



Los argumentos no se modifican

Resultado de la función

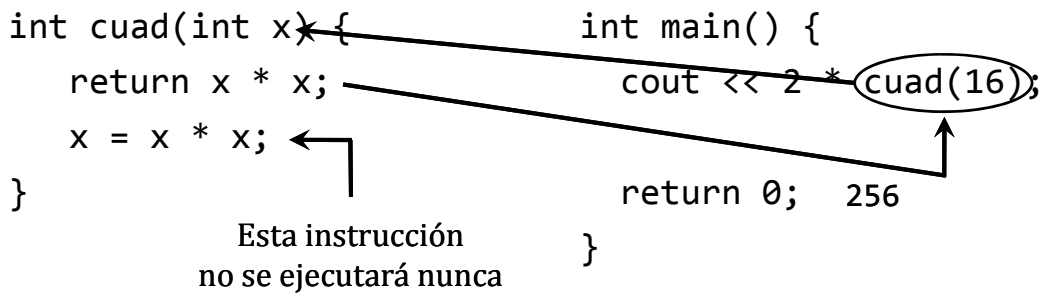
La función ha de devolver un resultado

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?

Colocaremos las funciones después de `main()`

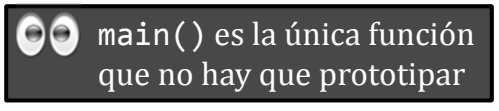
¿Son correctas las llamadas a funciones del programa?

- ¿Existe la función?
- ¿Concuerdan los argumentos con los parámetros?

→ Prototipos tras las inclusiones de bibliotecas

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);
...
```



Un programa con funciones

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

// Prototipos de las funciones (excepto main())
bool par(int num);
bool letra(char car);
int suma(int num);
double formula(int x, int y);

int main() {
   int numero, sum, x, y;
   char caracter;
   double f;
   cout << "Entero: ";
   cin >> numero;
   if (par(numero)) {
      cout << "Par";
   }
   ...
```


Un programa con funciones

```
   else {
      cout << "Impar";
   }
   cout << endl;
   if (numero > 1) {
      cout << "Sumatorio de 1 a " << numero << ": "
           << suma(numero) << endl;
   }
   cout << "Carácter: ";
   cin >> caracter;
   if (!letra(caracter)) {
      cout << "no ";
   }
   cout << "es una letra" << endl;
   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 propias

bool par(int num) {
   bool esPar;

   if (num % 2 == 0) {
      esPar = true;
   }
   else {
      esPar = false;
   }

   return esPar;
}
...
```

Luis Hernández Yáñez

Un programa con funciones

```
bool letra(char car) {
   bool esLetra;
   if ((car >= 'a') && (car <= 'z') || (car >= 'A') && (car <= 'Z')) {
      esLetra = true;
   }
   else {
      esLetra = false;
   }
   return esLetra;
}

int suma(int num) {
   int sum = 0, i = 1;
   while (i < num) {
      sum = sum + i;
      i++;
   }
   return sum;
}
...
```

Luis Hernández Yáñez

Un programa con funciones

funciones.cpp

```
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.