FACULTAD DE INFORMÁTICA

El lenguaje C++
Una introducción para programadores

TEMA

Programación orientada a objetos — Unidad 0

Autor: Luis Hernández Yáñez

FdI UCM

Contenido

FdI UCM

Contenido

FdI UCM

Contenido

FdI UCM

Contenido

FdI UCM

Introducción al lenguaje C++

La base de C++: el lenguaje C

Lenguaje estructurado, pero no estrictamente estructurado en bloques (no se pueden definir funciones dentro de funciones).

Compartimentalización de código (funciones) y datos (variables locales).

Incluye las construcciones típicas de los lenguajes estructurados (distintos tipos de condicionales y bucles).

Componente estructural básico: la función.

Otra forma de estructuración: el bloque de código.

```
if (x < 10)  {
    cout << "muy pequeño, pruebe de nuevo";
    reini_contador(-1);
}
```

Un bloque de código (delimitado por llaves)

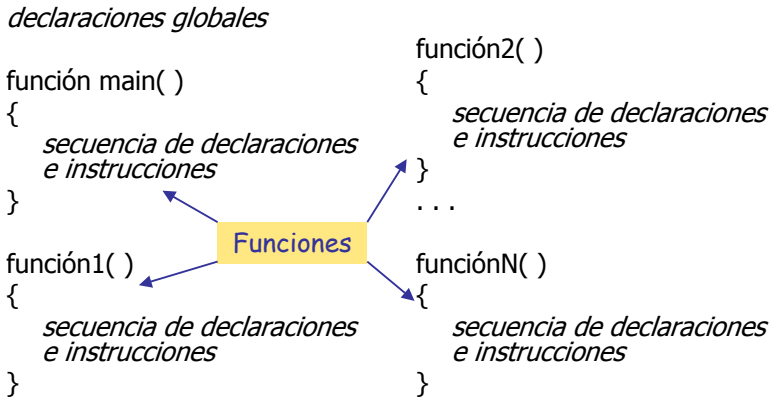
Se distingue entre mayúsculas y minúsculas.

Palabras clave: en minúsculas.

FdI UCM

Introducción al lenguaje C++

Forma general de un módulo de programa



FdI UCM

Introducción al lenguaje C++

Las bibliotecas de funciones

Los compiladores de C/C++ proporcionan bibliotecas de funciones.

Cada biblioteca de funciones tiene asociada un archivo de definición que se denomina *archivo de cabecera* (o simplemente cabecera).

Los archivos de cabecera en C tienen la extensión **.h** y contienen las declaraciones de los elementos de programa (datos y funciones) que la biblioteca exporta (pone a disposición).

Para utilizar algo de una biblioteca en un módulo de programa hay que colocar al principio una directiva de preprocesamiento `#include` seguida del nombre del correspondiente archivo de cabecera entre comillas dobles o ángulos:

```
#include <cabecera>
```

Programación modular (compilación separada de módulos)

Las bibliotecas de funciones

Por ejemplo, para mostrar datos en la pantalla podemos usar el operador **<<** con el elemento **cout**, que representa la salida estándar (la pantalla normalmente):

```
cout << dato;
```

Pero para poder usar **cout** y su operador **<<** (que, por cierto, se denomina *insertor*) debemos incluir la biblioteca donde están definidos; se trata de la biblioteca cuyas declaraciones se encuentran en el archivo de cabecera **iostream.h**:

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

Una vez que se incluye el archivo de cabecera se conocen las declaraciones de la biblioteca y, por tanto, se puede usar lo que está definido en ella.

Bibliotecas de funciones y espacios de nombres

Las bibliotecas más modernas de C++ tienen archivos de cabecera que no llevan la extensión **.h**; por ejemplo, la biblioteca **iostream** se rescribió para mejorarla y para distinguirla de la antigua su archivo de cabecera no lleva la extensión **.h**:

```
#include <iostream>
```

En las bibliotecas modernas pueden estar definidos *espacios de nombres*, que son algo así como secciones con nombre dentro de las que se incluyen las declaraciones; los espacios de nombres permiten usar varias veces un mismo identificador, cada una en un espacio de nombres distinto; se hace necesario en esos casos indicar el espacio de nombres que se usa:

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

std es el espacio de nombres estándar.

Un ejemplo de programa

Directiva de preprocesamiento

Biblioteca de E/S por consola
Este programa usa la definición de **cout** como salida estándar y el operador **<<** que se le aplica

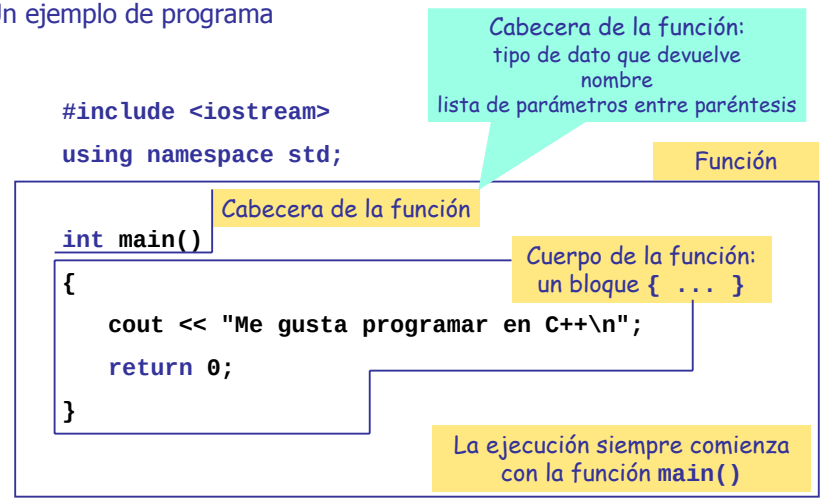
La biblioteca puede tener definido *espacios de nombres*.
Si no se "usa" (**using namespace**) un espacio de nombres (**std** en este caso), los elementos importados deben ser cualificados (**std::cout**, por ejemplo)

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

int main()
{
    cout << "Me gusta programar en C++\n";
    return 0;
}
```

Las palabras reservadas las mostraremos en azul para resaltarlas

Un ejemplo de programa



Introducción al lenguaje C++

Un ejemplo de programa

El punto y coma (;) es un terminador de instrucciones (declaraciones o ejecutables)

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

int main()
{
   cout << "Me gusta programar en C++\n";
   return 0;
}
```

cout: flujo de salida estándar (normalmente la pantalla)

<<: operador de inserción (*insertor*) para flujos de salida Envía al flujo el operando derecho

Constante de cadena de caracteres: **"..."**

Secuencias de escape (constantes de carácter de barra invertida)

Devuelve el valor-resultado

>>> Prog0001

El lenguaje C++: Datos, operadores y expresiones

Tipos de datos

- **char**
  caracteres y cualquier cantidad de 8 bits.
- **int**
  cantidades enteras (entre –32768 y 32767).
- **float**
  números reales (entre 1.18E–38 y 3.4E+38, y sus negativos).
- **double**
  números reales más grandes
  (entre 2.23E–308 y 1.79E+308, y sus negativos).
- **bool**
  valores lógicos (**true** / **false**).
- **void**
  nada (indica, por ejemplo, que una función no devuelve valor).

El lenguaje C++: Datos, operadores y expresiones

Modificadores de tipos (un)**signed** **short**/**long**

Se altera el significado de un tipo base

unsigned short int: 0 a 255

unsigned int: 0 a 65535

long int: –2147483648 a 2147483647

unsigned long int: 0 a 4294967295

long double: 3.37E-4932 a 1.18E+4932, y sus negativos

modificador **int** se abrevia a *modificador* (**long int** como **long**)

Nombres de identificadores

- Longitud: entre uno y 32 caracteres.
- Primer carácter: una letra o un subrayado
- Resto de caracteres: letras, números o subrayados

C++ distingue entre minúsculas y mayúsculas

El lenguaje C++: Datos, operadores y expresiones

Declaración de variables

[*modificadores*] *tipo lista_de_variables*;

lista_de_variables:
uno o más identificadores separados por comas.

```
int i, j, l;
short int si;
unsigned int ui;
double balance, beneficio, perdida;
```

El punto y coma es un *terminador* de instrucciones

- Variables locales:
  Declaradas dentro de las funciones *o de los bloques*.
- Parámetros:
  Definidos en la lista de parámetros formales de las funciones.
- Variables globales:
  Declaradas fuera de todas las funciones.

Variables locales

Reglas de ámbito: como en Pascal

```
void f()
{
  int t;
  ...
  if(t == 1) {
    char s[80];  // esta existe sólo en este bloque
    ...
  }
  /* aquí no se conoce a s */
}
```

Parámetros

```
int calcula(int i, float f)
{
  ...
}
```

Comentarios

De múltiples líneas:

```
/* ... */
```

De una sola línea:

```
// ...
```

(hasta el final de la línea)

El modificador de acceso `const`

Permite definir constantes.

Inicialización de la constante

```
const int cuenta = 100;
```

Los especificadores de clase de almacenamiento

[especificador] [modificadores] tipo variables;

auto **static** **register** **extern**

- **auto** (por defecto)
  Se asigna memoria durante toda la ejecución.
- **static**
  Se mantiene su memoria entre distintas llamadas a la función.
- **register**
  Se intenta alojarla en un registro de la UCP.

`extern`

Indica que las variables ya se han declarado en otro módulo.

Archivo 1

```
int x, y;
char c;

int main()
{
   . . .
}

void func1()
{
   x = 123;
}
```

Archivo 2

```
extern int x, y;
extern char c;

void func22()
{
   x = y / 10;
}

void func23()
{
   y = 10;
}
```


Instrucciones de asignación

variable = *expresión*;

Conversión automática de tipos.

Constantes literales

Tipo de dato	Ejemplos
`char`	`'a' '9' '\n'`
`int`	`1 123 21000 -234`
`long int`	`35000 -34`
`unsigned int`	`10000 987 40000`
`float`	`123.23 4.34e-3`
`double`	`12312333 -0.9876324`
`bool`	`true false`

Constantes de barra invertida

Para representar caracteres especiales:

`\n` salto de línea
`\t` tabulación
...

Además, constantes de cadena: `"Esto es una prueba"`

El lenguaje C++: Datos, operadores y expresiones

Conversión automática de tipos

Promoción de tipo:
cuando los dos operandos son de tipos distintos, el del tipo *menor* se promociona al tipo *mayor*.

Tipos de mayor a menor:

```
long double
double
float
long int
int
short int
char
```


El lenguaje C++: Datos, operadores y expresiones

Operadores aritméticos

`+` `-` `*` `/` `%` (módulo)

Operadores de incremento (`++`) y de decremento (`--`).

Si precede al operando, se aplica la operación antes de acceder al operando en la expresión:

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

Si sigue al operando, se aplica la operación después de que se acceda al operando en la expresión:

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


El lenguaje C++: Datos, operadores y expresiones

Operadores relacionales y lógicos

En C++, *Cierto* es cualquier valor distinto de 0. *Falso* es 0.

Operadores relacionales:

Operador	Acción
`>`	Mayor
`>=`	Mayor o igual
`<`	Menor
`<=`	Menor o igual
`==`	Igual
`!=`	No igual

Operadores lógicos:

Operador	Acción
`&&`	Y
`||`	O
`!`	NO

El lenguaje C++: Datos, operadores y expresiones

Expresiones

Los operadores, las constantes y las variables son lo que constituyen las *expresiones*.
Una expresión es cualquier combinación válida de esos elementos (como en otros lenguajes).

Moldes (*casts*)

Fuerzan a que una expresión sea de un tipo determinado.

`(`*tipo*`)` *expresión*

`(float) x/2` asegura que el resultado es **float**

Hay otras formas de moldes que se verán en su momento.

El lenguaje C++: Datos, operadores y expresiones

Inicialización de variables

En C++ NO se inicializan por defecto las variables.

En las declaraciones de variables se puede colocar detrás del nombre un signo igual seguido de un valor (el valor inicial).

```
int i = 10, j = -1;
```

Inicialización dinámica

En C++, las variables locales y las variables globales pueden inicializarse durante la ejecución:

```
...
int n = atoi(gets(cad));
long int pos = ftell(fp);
double d = 1.02 * cont / deltax;
```

Expresiones (incluyendo llamadas a funciones)

El lenguaje C++: Datos, operadores y expresiones

Espaciado y paréntesis

Para mayor legibilidad, se pueden añadir tabulaciones y espacios a discreción.

El uso de paréntesis redundantes o adicionales no produce errores ni disminuye la velocidad de ejecución de una expresión.

Abreviaturas de C

Simplificación de la escritura de ciertos tipos de asignaciones:

`x = x + 10;` equivale a `x += 10;`

La forma general en C de la abreviatura de

variable = variable operador expresión;

es *variable operador= expresión;*

PRECEDENCIA DE LOS OPERADORES

Mayor	Moldes
	`++ -- (prefijos)`
	`!`
	`* / %`
	`+ -`
	`< <= > >=`
	`== !=`
	`&&`
	`||`
Menor	`= += -= ...`

El lenguaje C++: Entrada/salida por consola

Archivo de cabecera: `iostream` (con espacio de nombres `std`)

Salida por pantalla:

cout << *dato*

```
cout << "Hola\n";
```

`<<` Operador de inserción (*insertor*)

Se pueden *concatenar* operadores <<

```
int i;
...
cout << "Hola\n" << i;
```

`cout << endl` provoca un salto de línea

Entrada por teclado:

cin >> *variable*

```
int i;
...
cin >> i;
```

`>>` Operador de extracción (*extractor*)

El lenguaje C++: Un ejemplo de programa

Programa que pida la base y la altura de un triángulo, calcule su área y muestre los datos y el resultado.

```
#include <iostream>
using namespace std;
int main()
{
```

```
  return 0;
}
```

El lenguaje C++: Un ejemplo de programa

Programa que pida la base y la altura de un triángulo, calcule su área y muestre los datos y el resultado.

```
#include <iostream>
using namespace std;
int main()
{
  float base, altura, area;

  return 0;
}
```

Los datos

El lenguaje C++: Un ejemplo de programa

Programa que pida la base y la altura de un triángulo, calcule su área y muestre los datos y el resultado.

```
#include <iostream>
using namespace std;
int main()
{
  float base, altura, area;
  cout << "Introduce la base: ";
  cin >> base;
  cout << "Introduce la altura: ";
  cin >> altura;

  return 0;
}
```

Entrada de datos

El lenguaje C++: Un ejemplo de programa

Programa que pida la base y la altura de un triángulo, calcule su área y muestre los datos y el resultado.

```
#include <iostream>
using namespace std;
int main()
{
  float base, altura, area;
  cout << "Introduce la base: ";
  cin >> base;
  cout << "Introduce la altura: ";
  cin >> altura;
  area = base * altura / 2;

  return 0;
}
```

Cálculos

El lenguaje C++: Un ejemplo de programa

Programa que pida la base y la altura de un triángulo, calcule su área y muestre los datos y el resultado.

```
#include <iostream>
using namespace std;
int main()
{
  float base, altura, area;
  cout << "Introduce la base: ";
  cin >> base;
  cout << "Introduce la altura: ";
  cin >> altura;
  area = base * altura / 2;
  cout << "El área de un triángulo de base " << base
       << " y altura " << altura << " es " << area;
  return 0;
}
```

Salida de datos

>>> Prog0002

El lenguaje C++: Control de la ejecución

Bifurcación: la instrucción `if`

```
if (expresión) instrucción;
else instrucción;
```

```
if (expresión) {
   secuencia de instrucciones
}
else {
   secuencia de instrucciones
}
```

La cláusula `else` puede no existir

Cierto: distinto de cero Falso: cero

```
int main()
{
  int magico = 123; /* número mágico */
  int intento;
  cout << "Adivina el número mágico: ";
  cin >> intento;
  if(intento == magico)
    cout << "** Correcto **";
  else cout << ".. Incorrecto ..";
  return 0;
}
```

Diferencia con Pascal: ; antes del `else`

El lenguaje C++: Control de la ejecución

Programa que pida tres números y los muestre de mayor a menor.

```
#include <iostream>
using namespace std;
int main()
{
  float r1, r2, r3, tmp;
  cout << "Introduce el primer número: ";
  cin >> r1;
  cout << "Introduce el segundo número: ";
  cin >> r2;
  cout << "Introduce el tercer número: ";
  cin >> r3;
  if(r1 < r2) {
    tmp = r1;
    r1 = r2;
    r2 = tmp;
  }
  if(r1 < r3) {
    tmp = r1;
    r1 = r3;
    r3 = tmp;
  }
  if(r2 < r3) {
    tmp = r2;
    r2 = r3;
    r3 = tmp;
  }
  cout << r1 << " " << r2 << " " << r3 << endl;
  return 0;
}
```

```
float r1, r2, r3, tmp;
```


El lenguaje C++: Control de la ejecución

Programa que pida tres números y los muestre de mayor a menor.

```
#include <iostream>
using namespace std;
int main()
{
  float r1, r2, r3, tmp;
  cout << "Introduce el primer número: ";
  cin >> r1;
  cout << "Introduce el segundo número: ";
  cin >> r2;
  cout << "Introduce el tercer número: ";
  cin >> r3;
  if(r1 < r2) {
    tmp = r1;
    r1 = r2;
    r2 = tmp;
  }
  if(r1 < r3) {
    tmp = r1;
    r1 = r3;
    r3 = tmp;
  }
  if(r2 < r3) {
    tmp = r2;
    r2 = r3;
    r3 = tmp;
  }
  cout << r1 << " " << r2 << " " << r3 << endl;
  return 0;
}
```

```
cout << "Introduce el primer número: ";
cin >> r1;
cout << "Introduce el segundo número: ";
cin >> r2;
cout << "Introduce el tercer número: ";
cin >> r3;
```


El lenguaje C++: Control de la ejecución

Programa que pida tres números y los muestre de mayor a menor.

```
#include <iostream>
using namespace std;
int main()
{
  float r1, r2, r3, tmp;
  cout << "Introduce el primer número: ";
  cin >> r1;
  cout << "Introduce el segundo número: ";
  cin >> r2;
  cout << "Introduce el tercer número: ";
  cin >> r3;
  if(r1 < r2) {
    tmp = r1;
    r1 = r2;
    r2 = tmp;
  }
  if(r1 < r3) {
    tmp = r1;
    r1 = r3;
    r3 = tmp;
  }
  if(r2 < r3) {
    tmp = r2;
    r2 = r3;
    r3 = tmp;
  }
  cout << r1 << " " << r2 << " " << r3 << endl;
  return 0;
}
```

```
if(r1 < r2) {
  tmp = r1;
  r1 = r2;
  r2 = tmp;
}
if(r1 < r3) {
  tmp = r1;
  r1 = r3;
  r3 = tmp;
}
if(r2 < r3) {
  tmp = r2;
  r2 = r3;
  r3 = tmp;
}
```

El lenguaje C++: Control de la ejecución

Programa que pida tres números y los muestre de mayor a menor.

```
#include <iostream>
using namespace std;
int main()
{
  float r1, r2, r3, tmp;
  cout << "Introduce el primer número: ";
  cin >> r1;
  cout << "Introduce el segundo número: ";
  cin >> r2;
  cout << "Introduce el tercer número: ";
  cin >> r3;
  if(r1 < r2) {
    tmp = r1;
    r1 = r2;
    r2 = tmp;
  }
  if(r1 < r3) {
    tmp = r1;
    r1 = r3;
    r3 = tmp;
  }
  if(r2 < r3) {
    tmp = r2;
    r2 = r3;
    r3 = tmp;
  }
  cout << r1 << " " << r2 << " " << r3 << endl;
  return 0;
}
```

```
cout << r1 << " " << r2 << " " << r3 << endl;
```

>>> Prog0003

El lenguaje C++: Control de la ejecución

Selección múltiple: la instrucción `switch`

Similar al **`case`** de Pascal

```
switch (variable) {
  case constante1:
    secuencia de instrucciones
    break;
  case constante2:
    secuencia de instrucciones
    break;
  . . .
  default:
    secuencia de instrucciones
}
```

La instrucción **`break`** termina la ejecución del **`switch`**.

Si un caso no termina con **`break`**, cuando le toque ejecutarse, su ejecución proseguirá con la secuencia de instrucciones del siguiente caso (si existe).

El lenguaje C++: Control de la ejecución

Ejemplo de `switch`

```
char menu()
{
  char c;
  cout << "1 - Comprobar ortografía\n";
  cout << "2 - Corregir errores\n";
  cout << "Introduce tu opción: ";
  cin >> c;
  switch(c) {
    case '1':
      comprobar(); // llamada a otra función
      break;
    case '2':
      corregir(); // llamada a otra función
      break;
    default:
      cout << "Opción no válida";
      c = '0';
  }
  return c;
}
```


El lenguaje C++: Control de la ejecución

El bucle `for`

`for``(`*inicialización*`;` *condición*`;` *incremento*`)` *instrucción*`;`

La *inicialización* normalmente es una instrucción de asignación que se utiliza para inicializar la variable de control del bucle.

La *condición* es una expresión relacional que determina cuándo finaliza el bucle.

El *incremento* define cómo cambia la variable de control cada vez que se repite el bucle.

```
int main()
{
  for(int x = 1; x <= 100; x++)
    cout << x;
  return 0;
}
```

Declaración de la variable de control directamente dentro del **`for`**

Aunque la forma de las secciones puede ser bastante complicada, usaremos esta forma sencilla

El lenguaje C++: Control de la ejecución

El bucle `while`

Similar al **`while...do`** de Pascal

```
while(condición) instrucción;
```

```
void esperar_caracter()
{
  char c;
  c = '\0';  // inicializa c al carácter nulo
  while(c != 'A')
    cin >> c;
}
```


El lenguaje C++: Control de la ejecución

El bucle `do...while`

Similar al **`repeat...until`** de Pascal pero con el sentido contrario

```
do {
  secuencia de instrucciones
} while(condición);
```

```
do {
  cin >> num;
} while(num > 100);
```


El lenguaje C++: Control de la ejecución

La instrucción `break`

Además de terminar los **`case`** de una instrucción **`switch`**, sirve para forzar la terminación inmediata de un bucle

```
for(t = 0; t < 100; t++) {
    cout << t;
    if(t == 10) break; // salir del for
}
```

La instrucción `continue`

Fuerza una nueva iteración en un bucle

```
do {
  cin >> x;
  if(x < 0) continue;  // ignorar lo que sigue y volver
  cout << x;           // a comprobar la condición
} while(x != 100);
```


El lenguaje C++: Estructuras

Estructuras

Similares a los registros (**`record`**) de Pascal

```
struct etiqueta { // etiqueta es el nombre de tipo
  tipo nombre-de-campo;
  tipo nombre-de-campo;
  ...
} lista-de-variables;
```

Se puede omitir la etiqueta o la lista de variables, pero no ambas cosas (no tendría sentido)

```
struct pieza {
  int numeroModelo;
  int numeroPieza;
  float coste;
} pieza1, pieza2;
```

`pieza`: nombre del tipo de estructura

Elementos o campos de la estructura

variables de este tipo de estructura

```
struct pieza {
  ...
}; // sin variables
```

```
struct { // sin etiqueta
  ...
} info_pieza;
```

El lenguaje C++: Estructuras

Declaración de variables de un tipo de estructura

```
struct tipo variable;
struct pieza info_pieza;
```

En este tipo de declaraciones se puede omitir `struct`:

```
pieza info_pieza;
```

Referencia a los elementos de una estructura

Se utiliza el operador punto (•)

variable-de-estructura`.` *nombre-de-campo*

```
info_pieza.coste
```


El lenguaje C++: Enumeraciones

Enumeraciones

```
enum etiqueta {lista_de_símbolos} lista_de_variables;
  // etiqueta es el nombre del tipo de enumeración
```

```
enum moneda { centimo, dos_centimos, cinco_centimos,
              diez_centimos, veinte_centimos,
              medio_euro, euro};
enum moneda dinero;
...
dinero = diez_centimos;
if(dinero == medio_euro)
  cout << "es la mitad de un euro\n";
```

Se pueden usar las enumeraciones para definir constantes, asignando valores a los símbolos de la enumeración:

```
enum { MAX = 100 };
```


El lenguaje C++: Enumeraciones

Los símbolos internamente son valores enteros.

Si se quieren mostrar las cadenas que significan los símbolos se ha de utilizar un `switch`:

```
switch(dinero) {
  case centimo: cout << "céntimo"; break;
  case dos_centimos: cout << "dos céntimos";
    break;
  case cinco_centimos: cout << "cinco céntimos";
    break;
  case diez_centimos: cout << "diez céntimos";
    break;
  case veinte_centimos: cout << "veinte céntimos";
    break;
  case medio_euro: cout << "medio euro"; break;
  case euro: cout << "euro";
}
```


El lenguaje C++: `typedef`

La instrucción *`typedef`*

```
typedef tipo nombre;
```

Simplemente define un nuevo nombre para un tipo.

```
typedef float balance;
balance negativo;
```

El lenguaje C++: Funciones

Forma general de una función

tipo nombre **(** *lista de parámetros*) **{**

cuerpo de la función

}

No se pueden anidar funciones
(declarar una función dentro de otra)

Prototipo de una función

A excepción de **`main()`**, en el módulo de programa debe aparecer el prototipo de cada función antes de que se utilice por primera vez esa función (a menudo al principio del módulo):

tipo nombre **(** *lista de parámetros*) **;**

El prototipo de una función informa de la existencia de la función (implementada más adelante), el tipo de dato que devuelve y los parámetros que tiene definidos.

El lenguaje C++: Funciones

```
#include <iostream>
using namespace std;
int potencia(int x);

int main()
{
  ...
}

int potencia(int x)
{
  ...
}
```

Prototipo

Implementación de la función

El lenguaje C++: Funciones

La instrucción `return`

- Fuerza la salida inmediata de la función.
- Sirve también para devolver un valor (el resultado).

Vuelta de una función

- Tras ejecutar la última instrucción (funciones tipo **`void`**).
- Al ejecutar una instrucción **`return`**.

Puede haber más de una instrucción **`return`**.

Valores devueltos por las funciones

Todas las funciones, excepto las de tipo **`void`**, devuelven un valor (por medio de un **`return`**).

Lo que devuelve `main()` Al sistema operativo

Código de terminación del programa (un entero).

El lenguaje C++: Funciones

Invocación de una función

Nombre de la función seguido de los argumentos entre paréntesis.

Tantos argumentos como parámetros, correspondiéndose según el orden de declaración y con concordancia de tipo.

`potencia(k)`

¿Dónde puede colocarse la llamada a una función?

Funciones **`void`**:
No pueden usarse en expresiones.

```
void espacios();
...
espacios();
```

Funciones que no son de tipo **`void`**:
pueden usarse en expresiones o no.
Si no se usan en expresiones, el valor se pierde.

```
i = j + potencia(k);

potencia(k);
```

Como si fuera un procedimiento de Pascal

El lenguaje C++: Funciones

La declaración (cabecera) de la función

Tipo de dato que devuelve:
cualquier tipo estándar o previamente declarado.

Lista de parámetros formales:
declaraciones de los parámetros separadas por comas.

Cada declaración de parámetro:
tipo nombre
donde *tipo* puede incluir modificadores y especificador.

En el prototipo se pueden (se suelen) omitir los nombres de los parámetros:

```
float potencia(float, int);  // prototipo

float potencia(float x, int i)
{ ...
```

Si no hay parámetros, se pone **void** o nada

El lenguaje C++: Funciones

Parámetros por valor

El mecanismo normal de paso de argumentos a funciones.

```
int cuad(int);
...
int cuad(int x)
{
  ...
```

El argumento puede ser cualquier expresión:

```
...
k = cuad(13);
k = cuad(i);
k = cuad(21 * i + j);
```


El lenguaje C++: Funciones

Parámetros por referencia

Se coloca **&** detrás del tipo en la declaración del parámetro:

```
void inter(int&, int&);
...
void inter(int& x, int& y)
{
  int tmp;
  tmp = x;
  x = y;
  y = tmp;
}
```

El equivalente a poner **var** en la declaración de un parámetro en Pascal

El argumento debe ser una variable

```
inter(a, b);
```


El lenguaje C++: Funciones

Recursión (o recursividad)

```
int fact(int n)
{
  int resp;
  if(n == 1) return(1);
  resp = fact(n - 1) * n;
  return(resp);
}
```

¡No se utiliza el nombre de la función para devolver valores!

El lenguaje C++: Funciones

Sobrecarga de funciones

Funciones con el mismo nombre pero con diferente definición de parámetros.

El compilador sabe cuál utilizar por los argumentos que se usan en la llamada.

```
int cuadrado(int i);
double cuadrado(double d);
long int cuadrado(long int l);
int main()
{
  cout << cuadrado(10) << "\n";
  cout << cuadrado(11.0) << "\n";
  cout << cuadrado(9L) << "\n";
  return 0;
} ...
```

Diferente número de parámetros o parámetros de distintos tipos

Para indicar que se trata de un **long int** y no un **int**

El lenguaje C++: Funciones

Argumentos implícitos (*por defecto*)

```
void f(int i = 1)
{ ... }
```

Si se ponen en el prototipo, no se vuelven a poner en la implementación de la función

Ahora, **f()** puede llamarse de una de dos formas:

```
f(10);   // pasa un valor explícito
f();     // la función usa el valor implícito
```

Todos los parámetros que se declaren con argumentos implícitos deben encontrarse al final de la lista de parámetros:

```
void f(int i, int j=2, int k=3); // CORRECTO
void f(int i=1, int j, int k=3); // INCORRECTO
```

Una vez asignado un valor implícito, debe haber también para los parámetros que siguen

El lenguaje C++: Funciones

Sobrecarga de operadores

tipo **operator***símbolo* **(***operandos***)**

```
struct punto {
  int x, y;
};

punto operator+(punto p1, punto p2)
{
  punto tmp;

  tmp.x = p1.x + p2.x;
  tmp.y = p1.y + p2.y;

  return tmp;
}
```

Parámetros

¿ Por qué no poder escribir **Matriz1 + Matriz2** en lugar de **suma(Matriz1,Matriz2)** ?

```
struct punto p1, p2, p3;
...
p3 = p1 + p2;
```


El lenguaje C++: El preprocesador

Directivas de preprocesamiento

Instrucciones dirigidas al compilador en el código fuente de un programa en C++.
Todas las directivas empiezan con el símbolo #.

#define

#define *nombre cadena*

```
#define CIERTO 1
#define FALSO 0
```

Antes de compilar, cada ocurrencia del nombre se sustituye por la cadena

#include

Inclusión de archivos fuente.

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

El compilador localiza los archivos de forma diferente si se usan "" o <> (*consultar*)