FACULTAD DE INFORMÁTICA

El lenguaje C++
Una introducción para programadores

SUPLEMENTO

Programación orientada a objetos — Unidad 0

Autor: Luis Hernández Yáñez

FdI UCM

Contenido

FdI UCM

Más sobre datos, operadores y expresiones

El operador ?

Exp1 **?** *Exp2* **:** *Exp3*;

a menudo es una alternativa para el **`if...else`**

Se evalúa *Exp1*: Si es cierta, se evalúa *Exp2*; si no, se evalúa *Exp3*.

```
x = 10;
y = x > 9 ? 100 : 200;
```

y toma el valor 100 porque *x* es mayor que 9.

Los operadores de punteros & y *

Para obtener la dirección de memoria de una variable y el valor que hay en una dirección de memoria (los veremos más adelante).

El operador de tiempo de compilación *sizeof*

Devuelve la longitud, en bytes, de la variable o del especificador de tipo entre paréntesis al que precede. Fomenta la portabilidad.

FdI UCM

Más sobre datos, operadores y expresiones

La coma como operador

Encadena varias expresiones.

```
x = (y = 3, y + 1);
```

Hacer esto (`y = 3`) y esto (`y + 1`)

primero asigna el valor 3 a *y* y luego asigna el valor 4 a *x*.
(la coma tiene menor precedencia que la asignación).

Los operadores . y ->

Para referenciar elementos (campos) de las estructuras.
-> se usa cuando se acceden a través de punteros (ya lo veremos).

( ) y [ ]

Los paréntesis son operadores que aumentan la precedencia de las operaciones que contienen.

Los corchetes sirven para la indexación de arrays (ya lo veremos).

Más sobre el `for`

Uso del operador coma en el `for`:

```
for(x=0, y=0; x+y<10; x++) {
  ...
}
```

La condición puede ser cualquier expresión relacional:

```
void conexion()
{
  char cad[20]; // una cadena de caracteres
  int x;
  for(x = 0; x < 3 && strcmp(cad, "clave"); x++) {
    cout << "Introduzca su contraseña: ";
    cin >> cad;
  }
  if(x == 3) cerrar();
}
```

Determina lo que hace esta función sabiendo que `strcmp()` devuelve `0` si las dos cadenas son iguales y distinto de `0` si son distintas

Salida inmediata del programa

`exit()` Terminación inmediata del programa

```
void exit(int estado); // toma el código de terminación

void menu()
{
  char c;
  do {
    cout << "1 - Comprobar ortografía\n";
    cout << "2 - Corregir errores\n";
    cout << "3 - Salir\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;
      case '3':
        exit(0);  // vuelta al sistema operativo
    }
  } while(c != '1' && c != '2');
}
```


Funciones insertadas

Funciones insertadas (*inline*)

Se declaran empezando con la palabra reservada **`inline`**.

En lugar de generar una llamada a la función cada vez que encuentra una invocación, el compilador sustituye la invocación por el código de la función. *EFICIENCIA EN TIEMPO DE EJECUCIÓN*

Para funciones sencillas.

```
inline int cuadrado(int i);
{
  return i*i;
} ...
```

```
cuad = cuadrado(x);
...
```

Al compilar se sustituye por:

```
cuad = x*x;
```


Más sobre el preprocesador

Directivas de compilación condicional

`#if`, **`#else`**, **`#elif`** y **`#endif`** ***`elif`*** = ***`else if`***

```
#include <iostream>
using namespace std;
#define MAX 10
int main()
{
#if MAX > 99
  cout << "compilado para arrays mayores de 99\n";
#else
  cout << "compilado para arrays pequeños\n";
#endif
  return 0;
}
```

Si se cumple la condición se compila esa sección; si no, no se compila

Más sobre el preprocesador

#ifdef y *#ifndef*

```
#include <iostream>
using namespace std;
#define LUIS 10
int main()
{
#ifdef LUIS
  cout << "Hola Luis\n";
#else
  cout << "Hola, quienquiera que seas\n";
#endif
#ifndef PEDRO
  cout << "PEDRO no definido\n";
#endif
  return 0;
}
```

Si está definida la macro (o si no lo está), se compila esa sección; admite **else**

#undef

#undef *nombre_de_macro*