Introducción a la Programación en C
–Funciones–

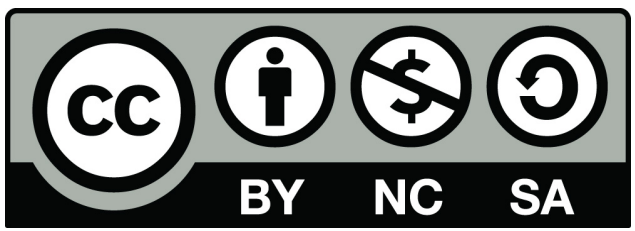
Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es



Contenidos

Introducción:

- Hasta ahora los programas realizados contienen un único bloque de código: el de la función `main`
- Escribir todo el código en un único bloque tiene desventajas claras:
  - La complejidad del código se incrementa debido a la dependencia entre sentencias
  - Número de errores elevado y difíciles de encontrar
  - Alta complejidad para desarrollar código por un grupo de programadores
- La **programación estructurada** surge como solución a los problemas anteriores

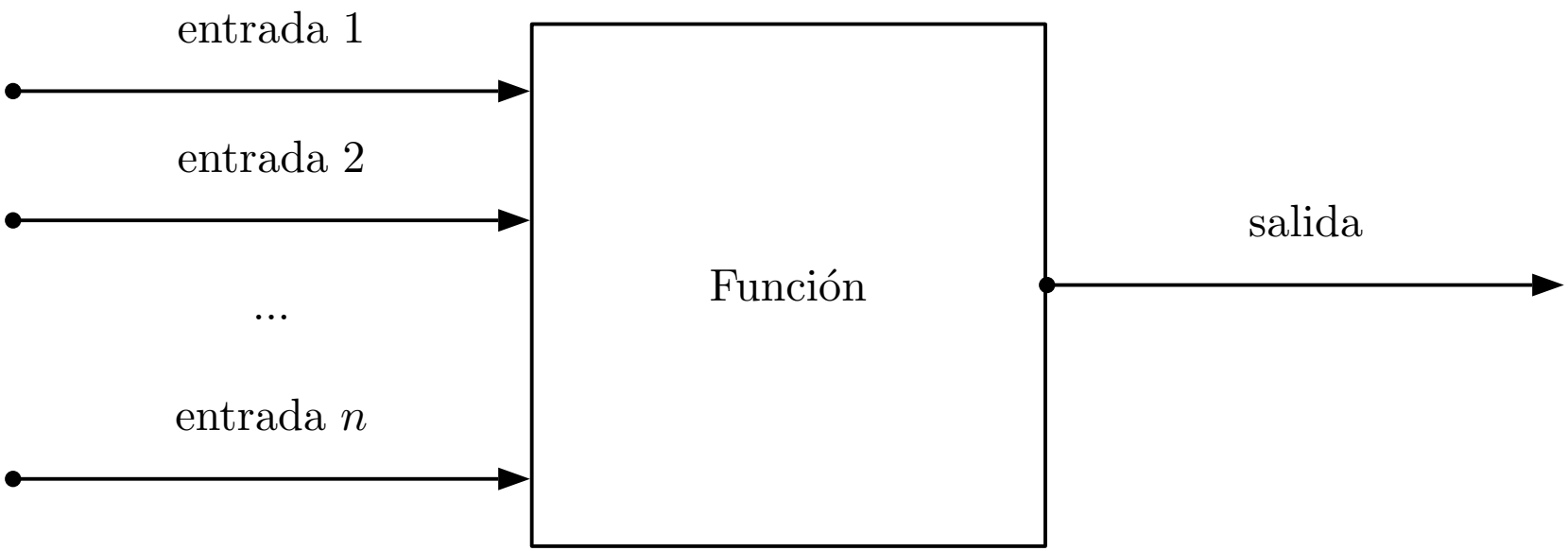
Programación Estructurada:

Función

Una función es un pequeño programa que obtiene unos resultados a partir de unos parámetros de entrada

- Las **funciones** constituyen las unidades mínimas con sentido y denominación propia dentro de un programa
- Las funciones se agrupan en **módulos** (archivos de código que incluyen funciones relacionadas lógicamente entre sí que se compilan por separado y se enlazan entre sí para formar un programa)

Programación Estructurada:



Programación Estructurada:

Para definir funciones hay que tener en cuenta lo siguiente:

- **El lenguaje C se basa en el uso de funciones**. No se puede escribir ninguna línea de código ejecutable (excluyendo declaraciones y definiciones) que no pertenezcan a una función
- **Todas las funciones devuelven algo**, aunque sea un valor vacío (*i.e.*, `void`)
- No se puede definir una función dentro de otra función
- En todo programa escrito en C debe existir una función `main`

Funciones:

Una función se define tal como sigue:

```
tipoDatoRetorno idFunción(tipoParám1 idParám1, tipoParám2 idParám2,..., tipoParámN idParámN) {
        sentencias
}
```

- `tipoDatoRetorno`: es el tipo de datos a devolver por la función
- `idFunción`: es el identificador de la función
- `tipoParám1, tipoParám2,..., tipoParámN`: son los tipos de datos de los parámetros
- `idParám1, idParám2,..., idParámN`: son los identificadores de los parámetros. Se conocen como **parámetros formales**

Funciones: Ejemplo

```
1 int multiply(int a, int b) {
2         int multiplication = a * b;
3         return multiplication;
4 }
```

Explicación:

1. Define una función cuyo identificador es `multiply`, que recibe dos parámetros de tipo `int` identificados como `a` y `b` y devuelve un `int`
2. Multiplica los dos valores recibidos por parámetro y los asigna a una variable, `multiplication`
3. Devuelve el valor asignado a la variable `multiplication`

Funciones: Sentencia return

La sintaxis de la sentencia **`return`** es como sigue:

```
return expresión;
```

- **`expresión`** debe ser del mismo tipo que el tipo a devolver por la función
- La sentencia **`return`** puede estar situada en cualquier lugar del cuerpo de una función
- Cuando se ejecuta una sentencia return sucede lo siguiente:
  1. Finaliza la ejecución de la función
  2. Se devuelve el valor de **`expresión`** a la función que realizó la llamada

Llamadas a Funciones:

Para ejecutar una función es necesario realizar una llamada desde otra función. La sintaxis es la siguiente:

```
identificadorFunción(expresiónParám1, expresiónParám2,...,
                     expresiónParámN);
```

- `identificadorFunción`: es el identificador de la función a llamar
- `expresiónParám1, expresiónParám2,..., expresiónParámN`: son expresiones empleadas como parámetros en la llamada. Se conocen como **parámetros reales**

Llamadas a Funciones:

1. El valor resultante de evaluar los parámetros reales es copiado en las variables de la función que actúan como parámetros formales

2. Una vez instanciados los parámetros formales se procede a ejecutar el conjunto de sentencias de la función (hasta llegar al final de la función o encontrar la sentencia `return`)

Llamadas a Funciones:

```
1  #include <stdio.h>
2
3  float calculateMaximum(float a, float b) {
4          float max;
5          if (a > b) {
6                  max = a;
7          } else {
8                  max = b;
9          }
10         return max;
11 }
12
13 void main(void) {
14         float x;
15         float y;
16         printf("Introduce el valor de x: ");
17         scanf(" %f", &x);
18         printf("Introduce el valor de y: ");
19         scanf(" %f", &y);
20         float maximum = calculateMaximum(x, y);
21         printf("El maximo es %f\n", maximum);
22 }
```

Prototipos:

- Si se quiere utilizar una función es suficiente con definirla para que pueda ser llamada desde cualquier punto del programa
- El compilador necesita saber (i) el tipo de dato que devuelve la función y (ii) los parámetros que acepta
- Cuando una función llama a otra definida anteriormente, el compilador conoce sus parámetros y el tipo de dato a devolver
- Cuando una función llama a otra definida posteriormente hay que emplear los **prototipos de funciones**

Prototipos:

Un prototipo de función se define tal como sigue:

```
tipoDatoRetorno idFunción(tipoParám1 idParám1, tipoParám2 idParám2,..., tipoParámN idParámN);
```

- En el prototipo de una función se indica el identificador de la función, el tipo de dato a devolver y los parámetros que acepta, finalizando con punto y coma
- Un prototipo declara una función y la definición incluye el código de la misma
- Habitualmente los prototipos de las funciones se incluyen al principio del módulo. De esta manera se podrá llamar a todas las funciones del módulo sin preocuparse de dónde están situadas

Prototipos:

```
1   #include <stdio.h>
2
3   float readNumber();
4   float calculateMaximum(float a, float b);
5
6   void main(void) {
7           float x = readNumber();
8           float y = readNumber();
9           float maximum = calculateMaximum(x, y);
10          printf("El maximo es %f\n", maximum);
11  }
12
13  float readNumber() {
14          float number;
15          printf("Introduce un numero: ");
16          scanf(" %f", &number);
17          return number;
18  }
19
20  float calculateMaximum(float a, float b) {
21          float max;
22          if (a > b) {
23                  max = a;
24          } else {
25                  max = b;
26          }
27          return max;
28  }
```

Paso de Parámetros:

En la mayoría de los lenguajes de alto nivel se cuenta con dos formas de realizar el paso de parámetros a una función:

- Paso de parámetros por valor
- Paso de parámetros por referencia

Paso de Parámetros: Por Valor

En el lenguaje de programación C todos los parámetros se pasan por valor

- Cuando se pasa un parámetro a una función, éste no resultará modificado una vez termine dicha función
- Es posible emular el paso de parámetros por referencia en el lenguaje C haciendo uso de variables de tipo puntero
- Cuando se pasa un parámetro por valor, lo que se hace es copiar el valor del parámetro real en el parámetro formal. A continuación, la función trabaja con el parámetro formal
- Si se modifica internamente el valor del parámetro formal, estos cambios no afectan a los parámetros reales, que seguirán sin cambios cuando finalice la función

Paso de Parámetros: Por Valor

```
1   #include <stdio.h>
2
3   void myFunction(int a, int b);
4
5   void main(void) {
6           int x = 10;
7           int y = 20;
8           printf("Antes de la llamada: %d %d\n", x, y);
9           myFunction(x, y);
10          printf("Despues de la llamada: %d %d\n", x, y);
11  }
12
13  void myFunction(int a, int b) {
14          a = 0;
15          b = 0;
16          printf("Dentro de la llamada: %d %d\n", a, b);
17  }
```

Paso de Parámetros: Por Referencia

Lo que se pasa a la función es una referencia a la dirección de memoria donde se almacena dicho dato

- La función utiliza dicha referencia para modificar el dato real, en lugar de trabajar sobre una copia del dato
- El programador debe realizar el paso de parámetros por referencia de forma explícita mediante punteros

La función **puede modificar el parámetro** que se pasa a la función

Paso de Parámetros: Por Referencia

```
1  #include <stdio.h>
2
3  void increment(int *value);
4
5  void main(void) {
6          int x = 10;
7          printf("Antes de la llamada: %d\n", x);
8          increment(&x);
9          printf("Despues de la llamada: %d\n", x);
10 }
11
12 void increment(int *value) {
13         *value = *value + 1;
14         printf("Dentro de la llamada: %d %d\n", *value);
15 }
```

Paso de Parámetros: Paso de Arrays

- Un array se puede pasar como parámetro de una función
- Hay que recordar que un array representa la dirección de comienzo del mismo
- Cuanso se pasa un array a una función, lo que realmente se está pasando es la dirección del primer elemento
- Dentro de una función **se pueden modificar** los valores del array
- Si se desea que una función no pueda modificar los valores de un array se debe poner la palabra reservada `const` delante del tipo de datos del mismo
- Habitualmente se debe pasar el tamaño del array a las funciones que los manipulan

Paso de Parámetros: Paso de Arrays

```
1  #include<stdio.h>
2  #define ELEMENTS 10
3
4  double getAverage(int array[], int length) {
5          double average = 0.0;
6          int i;
7          for (i = 0; i < length; i++) {
8                  average += array[i];
9          }
10         return average / length;
11 }
12
13 void main() {
14         int numbers[ELEMENTS], i;
15         printf("Introduce the numbers:\n");
16         for (i = 0; i < ELEMENTS; i++) {
17                 scanf(" %d", &numbers[i]);
18         }
19         double average = getAverage(numbers, ELEMENTS);
20         printf("The average value in the array is %.2f\n", average);
21 }
```

Errores Habituales:

- **Utilizar punto y coma en la definición de una función**. Cuando se define una función, su cabecera no termina en punto y coma. El punto y coma se utiliza en los prototipos de las funciones
- **Llamar a una función con un número diferente de parámetros**. Cuando se pasan a una función más o menos parámetros de los que espera, el compilador sólo avisa, pero no genera un error de compilación
- **Devolver un valor que no coincide con el tipo de la función**. Cuando devuelve un valor utilizando `return` hay que asegurarse de que el resultado de la expresión correspondiente coincide con el tipo a devolver por la la función
- **No utilizar `return` en una función que devuelve un valor**

Introducción a la Programación en C
–Funciones–

Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es