UNIDAD 5 – FUNCIONES

ÍNDICE

1.- Introducción

El código de un programa escrito en C se divide en funciones. Aunque similares a los “métodos” de Java, las funciones no están asignadas ni a una clase ni a un objeto. Una función en C se distingue sólo por su nombre. Dos funciones con igual nombre y con diferente número y tipo de parámetros se considera una definición múltiple, y por tanto un error.

Las funciones suelen encapsular una operación más o menos compleja de la que se deriva un resultado. Para ejecutar esta operación, las funciones pueden precisar la invocación de otras funciones (o incluso de ellas mismas como es el caso de las funciones recursivas).

Las funciones en un programa son entidades que dado un conjunto de datos (los parámetros), se les encarga realizar una tarea muy concreta y se espera hasta obtener el resultado. Lo idóneo es dividir tareas complejas en porciones más simples que se implementan como funciones. La división y agrupación de tareas en funciones es uno de los aspectos más importantes en el diseño de un programa.

Las funciones en C tienen el siguiente formato:

```
tipo_del_resultado NOMBRE(tipo_param1 param1, tipo_param2 param2, ... )
{
   /* Cuerpo de la función */
}
```

Cuando se invoca una función se asignan valores a sus parámetros y comienza a ejecutar el cuerpo hasta que se llega al final o se encuentra la instrucción return. Si la función devuelve un resultado, esta instrucción debe ir seguida del dato a devolver. Por ejemplo

```
int search(int table[], int size)
{
   int i, j;
   if (size == 0)
   {
      return 0;
   }
   j = 0;
   for (i = 0; i < size; i++)
   {
      j += table[i];
   }
   return j;
}
```

La ejecución de la función comienza en la línea 4. Si el parámetro size tiene valor cero, la función termina y devuelve el valor cero. Si no, se ejecuta el bucle y se devuelve el valor de la variable j. El tipo de la expresión que acompaña a la instrucción return debe coincidir con el tipo del resultado declarado en la línea 1.

La llamada a una función se codifica con su nombre seguido de los valores de los parámetros separados por comas y rodeados por paréntesis. Si la función devuelve un resultado, la llamada se reemplaza por su resultado en la expresión en la que se incluye. Por ejemplo:

```
#include <stdio.h>
int main()
{
int suma(int a, int b)
{
   return (a + b);
}
   int c;
   c = suma(12, 32);
   printf("%d",c);
   return(0);
}
```

2.- Paso de parámetros a una función

Los parámetros son variables locales a los que se les asigna un valor antes de comenzar la ejecución del cuerpo de una función. Su ámbito de validez, por tanto, es el propio cuerpo de la función. El mecanismo de paso de parámetros a las funciones es fundamental para comprender el comportamiento de los programas en C.

Considera el siguiente programa:

```
int suma(int a, int b)
{
   return (a + b);
}
int main()
{
   int x = 10;
   int y = 20;
   int z;

   z = suma(x, y);
}
```

Los parámetros a y b declarados en la línea 1 son válidos únicamente en la expresión de la línea 3. Las variables x, y y z, por su lado, son válidas en el cuerpo de la función main (líneas 7 a 11).

El ámbito de las variables x, y y z (ámbito llamador), y el de las variables a y b (ámbito llamado) son totalmente diferentes. El ámbito llamador desaparece temporalmente cuando se invoca la función desde la liṅea 11. Durante esa ejecución, el ámbito llamado es el visible. Al terminar la función, el ámbito llamado desaparece y se recupera el ámbito llamador.

La comunicación entre estos dos ámbitos se realiza en la línea 11. Antes de comenzar la ejecución de la función, los valores de las variables del ámbito llamador son copiadas sobre las variables del ámbito llamado. Cuando termina la ejecución de la función, la expresión de la llamada en la línea 11 se reemplaza por el valor devuelto. En la siguiente figura se ilustra este procedimiento para el ejemplo anterior.

El paso de parámetros y la devolución de resultado en las funciones C se realiza *por valor*, es decir, *copiando* los valores entre los dos ámbitos.

3.- Tablas como parámetros a una función

La copia de parámetros del ámbito llamador al llamado tiene una excepción. Cuando una función recibe como parámetro una tabla, en lugar de realizarse un duplicado se copia su dirección de memoria. Como consecuencia, si una función modifica una tabla que recibe como parámetro, estos cambios sí son visibles en el ámbito llamador. El siguiente ejemplo ilustra esta situación:

```
#include <stdio.h>
#define NUMBER 10

void fill(int table[NUMBER], int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        table[i] = 0;
    }
}
int main()
{
```

```
    int i, data[NUMBER];
    for (i = 0; i < NUMBER; i++)
    {
        data[i] = 10;
    }
    fill(data, NUMBER);
    /* Valores de data todos a cero */
    for (i = 0; i < NUMBER; i++)
    {
        printf("%d\n",data[i]);
    }
}
```

Como el compilador trabaja sólo con la información contenida en un único fichero, a menudo es preciso “informar” al compilador de que en otro fichero existe una función. Esto se consigue insertando, en lugar de la definición de la función (que ya está presente en otro fichero), su prototipo. El prototipo de una función es una línea similar a la primera de su declaración: tipo del resultado, seguido del nombre de la función y de la lista de tipos de datos de los parámetros separados por comas y rodeados por paréntesis. Toda función que se invoca debe ir precedida o de su definición o de su prototipo. La definición y el prototipo de la función pueden estar presentes en el mismo fichero. El siguiente ejemplo ilustra esta situación:

```
int suma(int, int);
int main()
{
    int i, j;
    i = 10;
    j = 20;
    /* Invocación de la función */
    i += suma(i, j);
}
/* Definición de la función */
int suma(int a, int b)
{
    return (a + b);
}
```