Funciones en el Lenguaje C

Una función es una rutina o conjunto de sentencias que realiza una determinada labor. En C todas las funciones devuelven un valor, que por defecto es un entero. Las funciones admiten argumentos, que son datos que le pasan a la función las sentencias que la llaman.

1.- Definición de una función.

La sintaxis habitual en la definición de una función es:

```
tipo identificador(lista_de_argumentos)

{

    /* bloque de código */

}
```

Donde:

- **tipo** es el tipo de datos devuelto por la función
- **identificador** es el nombre de la función. Debe ser un identificador valido.
- **lista_de_argumentos** es una lista de variables, separadas por comas, que conforman los datos que le pasamos a la función.

El tipo y la lista de argumentos son opcionales. Si omitimos el tipo, la función por defecto devolver un entero. Muchas veces el valor devuelto por la función es ignorado en el programa.

La lista de argumentos es también opcional. Un ejemplo es la función main(), que en principio no tiene argumentos. Podemos escribir como ejemplo:

```
hola()

{

    printf("hola\n");

}
```

que simplemente es una función que cuando es llamada imprime en pantalla un mensaje de saludo.

Cuando el programa al ejecutarse alcanza la llave de cierre '}' de la función, esta finaliza y devuelve el control al punto del programa que la llamó.

2.- Retorno de valores

Cuando la función finaliza hemos dicho que se devuelve un valor. Este valor en principio no está definido, es decir, puede devolver cualquier cosa.

Para obligar a la función a retornar un determinado valor se utiliza la sentencia return, seguida del valor a retornar. Como todas las sentencias en C se debe acabar con un ';'. Por ejemplo:

```
lista()

{

        return 1;

}
```

devuelve el entero 1 cada vez que es llamada. En C podemos devolver cualquier tipo de datos de los llamados escalares. Los tipos de datos escalares son los punteros, tipos numéricos y el tipo carácter. En C no se pueden devolver matrices ni estructuras.

3.- Paso de parámetros a una función

Utilizando la lista de argumentos podemos pasar parámetros a una función.

En la lista de parámetros se suele colocar un conjunto de identificadores, separados por comas, que representarán cada uno de ellos a uno de los parámetros de la función (parámetros formales). Observar que el orden de los parámetros es importante. Para llamar a la función habrá que colocar los parámetros en el orden en que la función los espera (Parámetros reales).

Cada parámetro puede tener un tipo diferente.

Asi:

```
imprime(int numero, char letra)

{

printf("%d, %c\n", numero, letra);

}
```

es una función que admite dos variables, una entera y otra de tipo carácter.

Un ejemplo de llamada a dicha función dería:

Imprime(5, 'a');

4.- Paso de parámetros por valor y por referencia

En los lenguajes de programación estructurada hay dos formas de pasar variables a una función: por referencia o por valor. Cuando la variable se pasa por referencia, la función puede acceder a la variable original. Este enfoque es habitual en lenguajes como el Pascal.

En C sin embargo todos los parámetros se pasan por valor. La función recibe una copia de los parámetros y variables, y no puede acceder a las variables originales. Cualquier modificación que efectuemos sobre un parámetro no se reflejará en la variable original. Esto hace que no podamos alterar el valor de la variable por equivocación.

Sin embargo, en determinadas ocasiones necesitaremos alterar el valor de la variable que le pasamos a una función. Para ello en el C se emplea el mecanismo de los punteros.

Un puntero es una variable que guarda la dirección de memoria donde esta guardada otra variable. Por lo tanto para crear parámetros de entrada-salida (paso de parámetros por referencia) es necesario declarar la función utilizando punteros como parámetros.

Ejemplo: Vamos a declarar un procedimiento (función de tipo void) que intercambia dos valores.

Void intercambiar(int *a,int *b)

El asterisco nos permite declarar un puntero. Si *a* es un puntero, hay que tener en cuenta que su valor es una dirección de memoria. Por lo tanto para ontener el contenido de la variable a la que apunta hay que utilizar *a.

La llamada a esta función sera como sigue:

Int a;

Int b;

a=2;

b=4;

inctercambiar(&a,&b);

…

Si recordamos la función scanf (que también recibe una dirección de memoria) tenemos que utilizar el operador &, que indica dirección de memoria de la variable indicada.

5.- Declaración y comprobación de tipos

Al igual que para las variables, cuando una función se va a usar en un programa antes del lugar donde se define, o cuando una función se define en otro fichero (funciones externas), la función se debe declarar.

La declaración de una función consiste en especificar el tipo de datos que va a retornar la función. Esto es obligatorio cuando vamos a usar una función que no devuelve un entero. Además en la declaración se puede especificar el número de argumentos y su tipo. Una declaración típica de función es:

tipo identificador(lista_de_argumentos_con_tipo);

Esto avisa al compilador de que la función ya existe, o que la vamos a definir después.

La lista de argumentos con tipo difiere de la lista de argumentos antes presentada en que el tipo de cada argumento se coloca dentro de la lista, antes de su correspondiente identificador, como hacíamos en la definición de variables. Por ejemplo:

char print(int numero, int letra);

declara una función que devuelve un carácter y tiene dos parámetros, un entero y un carácter.

La lista de argumentos permite al compilador hacer comprobación de tipos, ya que el tipo y número de argumentos debe coincidir en la declaración, definición y llamada a una función.