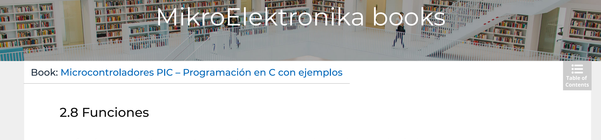

2.8 Funciones

Una función es una subrutina que contiene una lista de sentencias a realizar. La idea principal es dividir un programa en varias partes utilizando estas funciones para resolver el problema inicial con más facilidad. Además, las funciones nos permiten utilizar las destrezas y el conocimiento de otros programadores. Una función se ejecuta cada vez que se llame dentro de otra función. En C, un programa contiene como mínimo una función, la función main(), aunque el número de funciones es normalmente mayor. Al utilizar funciones el código se hace más corto ya que es posible llamar una función tantas veces como se necesite. En C, el código normalmente consiste en muchas funciones. No obstante, en caso de que su programa sea muy corto y simple, puede escribir todas las sentencias dentro de la función principal.

FUNCIÓN PRINCIPAL

La función principal main() es una función particular puesto que es la que se ejecuta al iniciar el programa. Además, el programa termina una vez completada la ejecución de esta función. El compilador reconoce automáticamente esta función y no es posible llamarla por otra función. La sintaxis de esta función es la siguiente:

```
void main (void) {

/* el primer 'void' significa que main no devuelve ningún valor. El segundo
'void' significa que no recibe ningún valor. Note que el compilador
también admite la siguiente sintaxis: 'main()' o 'void main()' o
'main(void)' */


..

/* --- Introduzca su programa aquí --- */

.
};
```

Esto significa que f es una función que recibe un número real x como parámetro y devuelve 2*x-y. La misma función en C se parece a lo siguiente:

```
float f (float x, float y) // variables flotantes x y y se pueden utilizar en f
{
  float r;    // declarar r para almacenar el resultado
  r = 2*x - y; // almacenar el resultado del cálculo en r
  return r;   // devolver el valor de r
}
```

Cada función debe ser declarada apropiadamente para poder interpretarla correctamente durante el proceso de compilación. La declaración contiene los siguientes elementos:

- Tipo de resultado (valor devuelto): tipo de dato del valor devuelto
- Nombre de función: es un identificador que hace posible llamar a una función.
- Declaración de parámetros se parece a la declaración de variable regular (por ejemplo: float x). Cada parámetro consiste en una variable, constante, puntero o matriz, precedidos por la etiqueta de tipo de dato. Se utilizan para pasar la información a la función al llamarla. Los parámetros diferentes están delimitados por comas.
- Cuerpo de función: bloque de sentencias dentro de llaves

Una función se parece a lo siguiente:

```
tipo_de_resultado nombre_de_función (tipo argumento1, tipo argumento2,...)
{
  Sentencia;
  Sentencia;
  ...
  return ...
}
```

Note que una función no necesita parámetros (función main() por ejemplo), pero debe estar entre paréntesis. En caso contrario, el compilador malinterpretaría la función. Para hacerlo más claro, puede sustituir el espacio en blanco encerrado entre paréntesis por la palabra clave void: main (void).

VALOR DEVUELTO

Una función puede devolver un valor (esto no es obligatorio) por medio de la palabra clave return. Al llegar a return, la función evalúa un valor (puede ser una expresión) y lo devuelve a la línea de programa desde la que fue llamada.

```
return r;       // Devolver el valor contenido en r
return (2*x - y); // Devolver el valor de la expresión 2*x-y
```

Una función no puede devolver más de un valor, pero puede devolver un puntero o una estructura. Tenga cuidado al utilizar matrices y punteros. El siguiente ejemplo es un error típico:

```
int *reverse(int *tab) // Esta función debe devolver una matriz r
{                      // cuyo contenido está en orden inverso con
                       // respecto a la matriz tab
  int r[DIM]; // Declaración de una nueva matriz denominada r
  int i;
  for(i=0;i<DIM;i++)     // Bucle que copia el contenido de tab en r
    r[i] = tab[DIM-1-i]; // al invertir el orden

  return r; // Devolver el valor r
}
```

En realidad, el compilador reserva memoria para el almacenamiento de variables de la función reverse sólo durante su ejecución. Una vez completada la ejecución de reverse, la localidad de memoria para la variable i o para la matriz r ya no está reservada. Esto significa que la dirección que contiene los valores de i o r[] está libre para introducir datos nuevos. Concretamente, la función devuelve sólo el valor &r[0], así que sólo el primer elemento de la matriz tab será almacenado en la memoria. Las demás localidades de memoria, tales como &tab[1], &tab[2], etc. serán consideradas por el compilador como espacios en blanco, o sea, estarán listas para recibir los nuevos valores. Para escribir esta función es necesario pasar la matriz r [] como parámetro (vea la subsección Pasar los parámetros). La función puede contener más de una sentencia return. En este caso, al ejecutar la primera sentencia return, la función devuelve el valor correspondiente y se detiene la ejecución de la función.

```
float abs (float x, float y) // Devolver el valor absoluto de 2*x-y
{
  if ((2*x - y) >= 0)
    return (2*x - y);
  else
    return (-2*x + y);
}
```

Si la función no devuelve ningún valor, la palabra void debe ser utilizada como un tipo de resultado en la declaración. En este caso, la sentencia return no debe ser seguida por ninguna expresión. Puede ser omitida como en el siguiente ejemplo:

```
void wait_1 (unsigned int a)
{
  cnt ++;       // Incremento de una variable global cnt
  Delay_ms(a) ; // Ejecución de la función Delay_ms
}              // Note que Delay_ms no devuelve nada
```

DECLARAR PROTOTIPOS DE FUNCIONES

Para utilizar una función, el compilador debe ser consciente de su presencia en el programa. En la programación en C, los programadores normalmente primero escriben la función main() y luego las funciones adicionales. Para avisar al compilador de la presencia de las funciones adicionales, se requiere declarar los prototipos de funciones en el principio de programa antes de la función main(). Un prototipo de función está compuesto por:

- tipo de resultado
- nombre de función
- tipos de parámetros
- un punto y coma (;)

El prototipo de la función main no necesita ser declarado.

```
float f (float, float);

/* no es obligatorio escribir los nombres de los parámetros. Este prototipo
informa al compilador: en el programa se utilizará la función f,
que utiliza dos parámetros de tipo float y devuelve el resultado del tipo
float. */
```

LLAMAR UNA FUNCIÓN

Mientras una función es definida y su prototipo declarado, se puede utilizar en culquier parte de programa. Sin embargo, como la función main es 'raíz' del programa, no puede ser llamada de ninguna parte de programa. Para ejecutar una función, es necesario escribir su nombre y los parámetros asociados. Vea los siguientes ejemplos:

```
float resultado,a,b; // resultado,a,b,time deben coincidir con los tipos
                     // definidos
int time = 100;      // en la declaración de las funciones f y wait_1
a = 10.54;
b = 5.2;
resultado = f(a,b);  // Ejecutar la función f por medio de los parámetros a y b

                     // El valor devuelto se le asigna a la variable resultado
pausa_1(tiempo);     // Ejecutar la función pausa_1 por medio de la variable tiempo
función1();          // Ejecutar la función función1 (sin parámetros)
```

Cuando se llama una función, el programa salta a la función llamada, la ejecuta, después vuelve a la línea desde la que fue llamada.

PASAR LOS PARÁMETROS

Al llamar una función, se le pasan los parámetros. En C existen dos formas diferentes para pasar parámetros a una función. El primer método, denominado 'paso por valor', es el más fácil. En este caso, los parámetros se pueden considerar como variables locales de la función. Cuando se llama una función, el valor de cada parámetro se copia a un nuevo espacio de memoria reservado durante la ejecución de la función. Como los parámetros se consideran como variables locales por el compilador, sus valores pueden ser modificados dentro de la función, pero sus modificaciones no se quedan en la memoria una vez completada la ejecución de la función. Tenga en cuenta de que la función devuelve un valor, y no una variable. Además, se crean copias de los valores de los parámetros, por lo que sus nombres en la función f pueden ser diferentes de los parámetros utilizados en la main(). La mayor desventaja del 'paso por el valor' es que la única interacción que una función tiene con el resto del programa es el valor devuelto de un solo resultado (o la modificación de las variables globales). El otro método, denominado 'paso por dirección' le permite sobrepasar este problema. En vez de enviar el valor de una variable al llamar a función, se debe enviar la dirección de memoria del valor. Entonces, la función llamada será capaz de modificar el contenido de esta localidad de memoria.

```
// Función 'sort'ordena los miembros de la matriz por valor ascendente
// y devuelve el miembro con máximo valor

int sort(int *); // Prototipo de función
const SIZE = 5; // Número de miembros a ordenar

void main() {
  int maximum, input[SIZE] = {5,10,3,12,8}; // Declaración de variables en la matriz
  maximum = sort(input);                    // Llamar a función y asignarle el máximo
  // valor a la variable maximum
}

int sort(int *sequence) {
  int i, temp, permut;    // Declaración de variables
  permut = 1;             // Bandera de bit indica que se ha hecho una permutación

  while(permut!=0) {      // Quedarse en el bucle hasta reiniciar la bandera
    permut = 0;           // Bandera reiniciada
    for(i=0;i<SIZE-1;i++) {  // Comparar y ordenar los miembros de la
                             // matriz (dos a dos)
      if(sequence [i] > sequence[i+1]){
        temp = sequence [i];
        sequence[i] = sequence[i+1];
        sequence[i+1] = temp;
        permut = 1; // Se ha hecho una permutación, bandera de bit
                    //se pone a uno
 }
    }
  }

  return sequence[SIZE-1]; // Devolver el valor del último miembro

} // que es al mismo tiempo el miembro con el máximo valor
```

En este ejemplo, por medio de una función se realizan dos operaciones: ordena los miembros de la matriz por valor asdendente y devuelve el máximo valor. Para utilizar una matriz en una función es necesario asignar la dirección a la matriz (o a su primer miembro). Vea el siguiente ejemplo:

```
float método_1(int[]);          // Declaración de prototipo de la función Método_1
float método_2(int*);           // Declaración de prototipo de la función Método_2

const NÚMERO_DE_MEDICIONES = 7; // Número de los miembros de la matriz

void main()
{
  double promedio1, promedio2; // Declaración de las variables promedio1
                               // y promedio2
  int voltaje [NÚMERO_DE_MEDICIONES] = {7,8,3,5,6,1,9}; // Declaración de la
                                                        // matriz voltaje
  promedio1 = método_1(&voltaje[0]); // Parámetro de la la función es la dirección
                                     // del primer miembro
  promedio2 = método_2(voltaje); // Parámetro de la función es la dirección de
                                 // la matriz
}

//*****************************************************************
float método_1(int voltaje[])       // Inicio de la función método_1
{
  int i, suma;                      // Declaración de las variables locales i y suma

  for(i=0;i<NÚMERO_DE_MEDICIONES;i++) // Cálculo del valor promedio de voltaje
    suma += voltaje[i];               // Es posible utilizar *(voltaje+i)en vez de voltaje[i]

  return(suma/NÚMERO_DE_MEDICIONES);
}

//*****************************************************************
float método_2 (int *voltaje) //Inicio de la función método_2
{
  int i, suma;                  // Declaración de las variables locales i y suma

  for(i=0;i<NÚMERO_DE_MEDICIONES;i++) // Cálculo del valor promedio de voltaje
    suma += *(voltaje+i);             // Es posible utilizar voltaje[i] en vez de *(voltaje+i)

  return(suma/NÚMERO_DE_MEDICIONES);
}
```

Las funciones 'método_1' y 'método_2' son completamente equivalentes. Las dos devuelven el valor promedio de la matriz 'voltaje[]'. Después de declararla, la dirección del primer miembro se puede escribir como 'voltaje' o '&voltaje[0]'.