CAPÍTULO 5
FUNCIONES

1) Implementación de funciones

a) Generalidades

Una función es un bloque de código de programa auto contenido, diseñado para realizar una tarea determinada [1].

El uso de funciones está directamente relacionado con el paradigma de la programación estructurada, también llamado programación modular. Este paradigma busca dividir al programa en módulos pequeños, en lugar de hacer un solo módulo grande. El escribir código usando funciones trae consigo varias ventajas; por ejemplo:

- Evita repetir líneas de código, pues, se escribe una sola vez la función apropiada y se le usa cuántas veces se desee dentro de un programa.
- La mantenibilidad mejora; ya que un cambio dentro del algoritmo implementado dentro de una función solo se hará una vez. Todos los segmentos de código dentro del programa principal donde se llame a la función, verán los cambios ya implementados.
- Será más sencillo el actualizar código dentro de módulos pequeños en lugar de hacerlo en un módulo grande que represente todo el programa.
- La legibilidad del programa o sistema aumentará, siempre y cuando los nombres usados para nombrar a las funciones sean descriptivos.

La estructura de un programa con uso de funciones sería:

Sin funciones	Usando funciones

int main ( ){
...
código 1;
...
código 2;
...
código 3;
...
return 0;
}
	int main ( ){
	función 1;
	función 2;
	función 3;
	return 0;
	}
	función 1 ( ){
	código 1;
	}
	función 2 ( ){
	código 2;
	}
	función 3 ( ){
	código 3;
	}

b) Estructura de una función.

La forma general de una función es:

```
tipo_retorno nombre_funcion (lista_parametros){
      /* cuerpo de la función */
      return expresión;
}
```

Donde:

`tipo_retorno` Especifica cualquier tipo de valor válido que devuelve la función, o la palabra reservada **`void`** si la función no devuelve ningún valor.

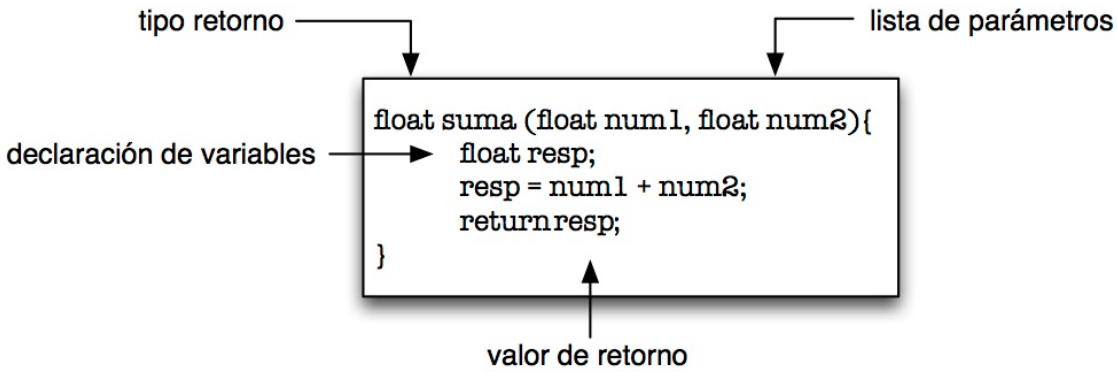
Aunque por defecto las funciones devuelvan un valor **int**, será mejor definir el valor de retorno acorde a las necesidades del programador.

`nombre_funcion` Es el nombre de la función (debe ser descriptivo). Un nombre de una función inicia con una letra o con un guion bajo (_) y pude contener tantas letras, números o subrayados como se desee (se deben respetar las normas para nombres en ANSI C).

`lista_parametros` Es la lista de nombre de variables, separados por coma, con sus tipos asociados. Una función puede requerir o no parámetros para ejecutar la acción definida en su bloque, los parámetros pueden ser de cualquier tipo válido de ANSI C.

`return expresión` Expresión que debe dar un valor como resultado que debe ser del mismo tipo de retorno de la función. Una función puede tener cualquier número de sentencias **return**; pero, tan pronto como encuentre la primera de ellas devolverá el valor y terminará su ejecución.
Si el tipo de retorno es void, la sentencia return se puede escribir como "**`return ;`**" sin ninguna expresión, o bien, de modo alternativo se puede omitir la sentencia **return**.

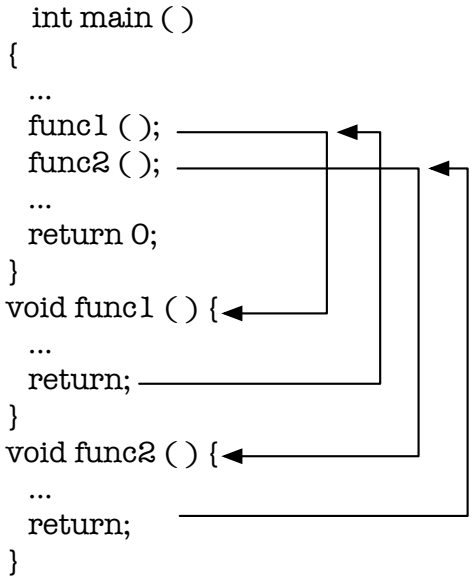
Ejemplo: Función que recibe dos números flotantes y devuelve su suma:



Nota: La función *return* termina inmediatamente la función [2]. Esto es, se sale inmediatamente de la función y no se ejecutan el resto de instrucciones.

c) Llamada a una función.

Las funciones para poder ser ejecutadas, deben ser llamadas o invocadas desde la función principal o desde otra función.
La función llamada recibe el control del programa, se ejecuta desde el principio y cuando termina (alcanza la sentencia "`return`" o "`}`", lo que ocurra primero), el control del programa vuelve a la función que la llamó [3].



La sintaxis para llamada a funciones es, en general:

```
nombre_funcion(parametros_funcion);
```

Si la función devuelve valor, éste puede ser asignado a una variable, en la forma

```
variable = nombre_funcion(parametros_funcion);
```

`parametros_funcion` debe coincidir en: número, tipo y orden con **`lista_parametros`** de la función a la que se llama. Si la función llamada no recibe argumentos se dejará el espacio en blanco.

El siguiente ejemplo calcula la media aritmética de dos número reales. Se ha creado una función llamada `media` (recibe como argumentos dos valores de tipo **`double`** y retorna un valor **`double`**) la cual es llamada desde la función principal.

```
include <stdio .h>
double media(double x1, double x2)
{
      return(x1 + x2)/2;
}

int main()
{
      double num1, num2, med;
      printf (" Introducir dos números reales:");
      scanf("%lf %lf",&num1, &num2);
      med = media(numl , num2);  //llamada a la func. media
      printf("El valor medio es %.4lf \n", med);
      return 0;
}
```

d) Prototipos.

La declaración de una función se denomina prototipo. Los prototipos de una función contienen su cabecera, no el detalle de implementación, y terminan con un punto y coma [3].

La declaración de la función se realiza en la cabecera del programa y le permite conocer al compilador: el tipo de dato que retorna la función, el número y los tipos de los argumentos [4]. Esto se conocerá antes de saber cómo está implementada la función, y le sirve al compilador para ir reconociendo y revisando las llamadas a las funciones.

Nota de programación: C recomienda que se declara una función antes de que sea definida o implementada [3].

En el siguiente ejemplo se escribe la función **area**( ) de rectángulo. En la función **main** ( ) se llama a la función entrada ( ) para pedir la base y la altura; a continuación se llama a la función **area**( ). Note como en este caso se hace una declaración de prototipos en la cabecera del programa.

```
#include <stdio.h>
/*prototipo de la función area*/
float area_rectangulo( float b, float a);
/*prototipo de la función entrada*/
float entrada(void);

int main()
{
      float b, h;
      printf("\nBase del rectangulo: ");
      b = entrada ( );
      printf("\nAltura del rectangulo: ");
      h = entrada( );
      printf("\nArea del rectangulo: %.2f",area_rectangulo(b,h));
      return 0;
}
/* devuelve número positivo */
float entrada ( )
{
      float m;
      do{
            scanf("%f", &m);
      } while (m <= 0.0);
      return m;
}

/* calcula el area de un rectángulo */
float area_rectangulo(float b, float a)
{
      return (b*a);
}
```

Nota de programación: Las funciones en C no se pueden anidar.

---

2) Funciones Recursivas.

Una función recursiva es aquella que se llama a si misma de forma directa o indirecta. Una *recursión directa* es cuando en el código de una función `f( )`, se encuentra una sentencia en la que se invoca a la misma función `f( )`. En la recursión indirecta en cambio, una función `f( )`, contienen un sentencia que invoca a un función *g*( ), y ésta función *g*( ) invoca a otras funciones o a la función `f( )` nuevamente.
Un requisito de la función recursiva es, que, debe ser ***controlada***; es decir, debe existir una forma segura de terminar las llamadas sobre si misma, de manera que no se genere una secuencia infinita de llamadas. El número de repeticiones de la recursión, se controla tomando decisiones acerca de si la función se llama a si misma o no. Por lo tanto, la definición recursiva de una función `f(n)` debe contener obligatoriamente una ***condición de salida*** en la que `f(n)` se defina directamente (no se llame a si misma), para uno o más valores de `n`.

Un ejemplo estándar de recursión controlada es la función `factorial`. La función factorial de un número `n` se define como el producto de todos los números enteros desde `1` hasta `n`. Además, el factorial de `0` es `1`.

Mire el siguiente desarrollo de la función factorial:

`factorial(0) = 1`
`factorial(1) = 1`
`factorial(2) = 1 * 2 = 2`
`factorial(3) = 1 * 2 * 3 = 6`
`factorial(4) = 1 * 2 * 3 * 4 = 24`

Ahora, fíjese en la siguiente representación:

`factorial(0) = 1`
`factorial(1) = 1 * factorial (0) = 1`
`factorial(2) = 2 * factorial (1) = 2`
`factorial(3) = 3 * factorial (2) = 6`
`factorial(4) = 4 * factorial (3) = 24`

En base al ejemplo anterior podemos notar dos aspectos:
Para n > 0, su factorial puede ser calculado así:

`factorial(n) = n * factorial(n-1)`

Para n igual a 0:

`factorial(0) = 1`

La primera sentencia es recursiva (define al `factorial(n)` en función de `factorial(n-1)`). Mientras que la segunda sentencia sería la condición de salida.

Éste es el código que implementaría la función factorial.

```
#include <stdio.h>
int factorial(int n);
int main ( )
{
   printf ("%d\n", factorial(3));
   return 0;
```

```
}
int factorial (int n)
{
   if (n == 0)
         return 1;
   else
         return (n * factorial (n-1));
}
```

Bibliografía.

[1] E. Granizo Montalvo, Lenguaje C, Teoría y Ejercicios, 2da ed. Quito - Ecuador; ESPE, 1999.

[2] L. Joyanes Aguilar y I. Zahonero, Programación en C: metodología, algoritmos y estructuras de datos. McGraw-Hill, 2005.

[3] L. Joyanes Aguilar y I. Zahonero, Programación en C: metodología, algoritmos y estructuras de datos. McGraw-Hill, 2005.

[4] E. Granizo Montalvo, Lenguaje C, Teoría y Ejercicios, 2da. ed. Quito - Ecuador: ESPE, 1999.

Lectura recomendada.

Para mayor detalle refiérase al Capítulo 7 del libro “Programación en C: metodología, algoritmos y estructuras de datos” de Luis Joyanes.