Funciones.

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Pablo Gil*
Hector Farina

Revisores: *Ing. Ernesto Gigliotti*
Lic. Mauricio Dávila

Versión : 1

Índice de contenido

1 Funciones

Todos los programas desarrollados hasta el momento estaban formados por un cuerpo principal, llamado main, en el que se desarrollaba la totalidad del código. Dado que los programas eran relativamente sencillos no existía inconveniente en desarrollarlos de esa forma.
A medida que los programas crecen en extensión y complejidad la resolución se torna más complicada y su depuración y modificaciones futuras resultan casi imposibles.
Para resolver este tipo de problemas lo que se hace es dividir el programa en módulos más pequeños que cumplan una tarea simple y bien acotada. Con esto se logra:

1. El programa es más simple de comprender ya que cada módulo se dedica a realizar una tarea en particular.
2. La depuración queda acotada a cada módulo.
3. Las modificaciones al programa se reducen a modificar determinados módulos.
4. Cuando cada módulo está bien probado, se lo puede usar las veces que sea necesario sin volver a revisarlo.
5. Se obtiene una independencia del código en cada módulo.

Lo que llamamos en forma genérica como "módulo" en el lenguaje C son las funciones.
Una función es un programa que realiza una tarea determinada y bien acotada a la cual le pasamos datos y nos devuelve datos. Este programa se ejecuta cuando se lo llama (llamada a la función). Como ejemplo de funciones conocidas son:

- **getche**
- **scanf**
- **gets**
- **printf**
- **strcpy**

Estas funciones se usan continuamente cuando el usuario escribe su nombre, le pasa datos y la función retorna un resultado.
Por ejemplo la función *getche*. A esta función no recibe datos y devuelve el caracter que leyó del teclado.
Cuando la llama se escribe:

```
rta=getche();
```

El código que se ejecuta cuando se llama a la función *getche* está dentro de las bibliotecas que vienen con el compilador, el código no se conoce (independencia del código) pero se sabe qué es lo que hace y que lo hace bien.

De lo que se trata en este capítulo, es de cómo trabajan las funciones y cuál es el procedimiento a seguir para escribir una función.

1.1 Declaración de una función

La forma general de declarar una función es la siguiente

```
Tipo_devuelto   Nombre_de_funcion ( tipo variable_1, tipo variable_2 , ..... , tipo variable_N)
```

- Tipo_devuelto: Es el tipo de dato que devuelve la función luego de terminada su ejecución
- Nombre_de_función: Es el nombre de la función
- tipo variable_1: Es el tipo y nombre de la variable que se le pasa a la función.

Se observa que según la declaración de la función se devuelve solamente un valor.

Cuando trabajamos con funciones tenemos que tener en cuenta que la forma en la que escribimos un programa se modifica. Los programas desarrollados hasta el momento contaban solamente con la función *main* y tenían el siguiente formato

```
# include .....      Lugar donde se incluyen todos los archivos de cabecera necesarios
.............
# define.......      Lugar donde se definen las constantes y macros
.............

void main (void)     comienza el programa principal
{
.............        desarrollo del programa
.............
}
```

Al incluir el uso de las funciones , la forma general de escribir un programa es la siguiente:

```
# include .....      Lugar donde se incluyen todos los archivos de cabecera necesarios
.............
# define.......      Lugar donde se definen las constantes y macros
.............

declaración de
las funciones        Aquí se escriben todos los prototipos de las funciones a desarrollar

void main (void)     Comienza el programa principal
{
.............        Desarrollo del programa
.............
llamada a la
función              Desde aquí se puede realizar la llamada a las funciones
}

Desarrollo de
funciones            Terminado el programa principal comienza el desarrollo de las funciones.
```

Existen 3 instancias donde se hace mención a las funciones:

1-Declaración de las funciones: En este lugar se declaran todas aquellas funciones propias que se van a utilizar durante el programa. La forma de declararlas es como se explicó al comienzo, teniendo en cuenta que cada declaración de función termina con punto y coma (;) .A las declaraciones que se realizan en este punto se las llama prototipo de la función. Los prototipos son necesarios para que el compilador verifique que sean coherentes los tipos de datos declarados en el prototipo contra los que realmente se usan en la llamada a la función.

2-Llamada a la función: Cuando se llama a la función es para usarla, es decir para que realice su trabajo. Desde cualquier parte de *main* o desde otra función se puede hacer la llamada. En el momento en que se produce la llamada a la función, se produce un salto en el flujo del programa. En ese momento se pasa a ejecutar el código de la función que va a terminar de ejecutarse al encontrar la sentencia *return* o llegar a la llave que cierra la función. Cuando la función finaliza, se sigue ejecutando el código de la función que produjo la llamada.

3-Desarrollo de las funciones: Esta es la parte en la cuál se escribe el código de la función (tal cual se hacía con *main*).

Un ejemplo sencillo para ver cada una de las partes puede ser el siguiente:
Ingresar 2 números enteros y por medio de una función calcular la suma de los mismos

```
#include <stdio.h>

int suma(int a, int b);

void main(void)
{
	int x,y,z;

	printf("ingrese numero a sumar: ");
	scanf("%d",&x);
	printf("ingrese numero a sumar: ");
	scanf("%d",&y);
	z=suma(x,y);
	printf("La suma es %d",z);
}

int suma(int a, int b)
{
	int total;
	total=a+b;
	return total;
}
```

En el ejemplo de arriba aparecen los lugares donde interviene la función.
Se observa que el prototipo termina con ;. La función se llama suma, se le pasan 2 variables enteras como parámetros y devuelve un entero. A las variables que se le pasan a la función se la llaman **parámetros formales**.

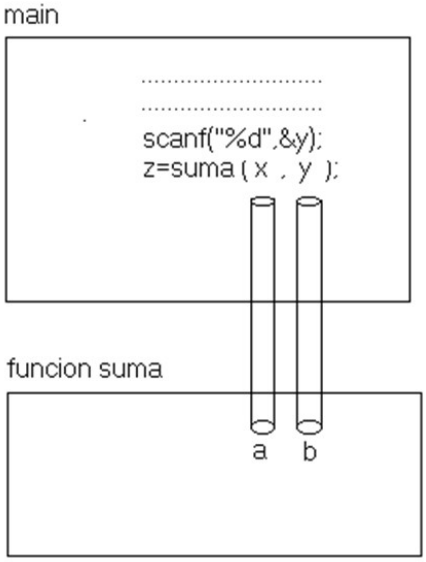
Cuando se llama a la función se le pasan las variables con las que tiene que trabajar , en este momento a las variables se las llama **parámetros actuales**.

En el desarrollo de la función se vuelve a escribir el prototipo pero ahora no se coloca el ";" al final de la línea. En este lugar se escribe el código de la función tal cual se escribe en el *main*.

La última línea de la función contiene una instrucción *return* que lo que hace es terminar la función y devolver el valor que se encuentra a continuación, en este caso devuelve el valor que tiene la variable "total".

NOTA: Las variables definidas como parámetros formales de la función son variables locales de la misma.

Gráficamente podemos imaginarnos a la función trabajando con el *main* de acuerdo al siguiente esquema



1.2 Visibilidad de variables locales y globales (scope)

Dependiendo en el lugar donde se define una variable, serán las características de visibilidad y ámbito de validez que ellas tienen.
Todas las variables que se encuentren definidas dentro de las llaves de una función (recordar que *main* también es una función) tienen validez dentro de dicha función y se llaman variables locales, al ámbito donde dichas variables son visibles, se lo conoce como *scope*.

Veamos el ejemplo anterior donde usamos la función *suma*. Las variables x, y, z que están definidas dentro de *main*, son locales de *main* y valen solamente mientras se esta dentro de *main*, es decir que cuando se esta ejecutando la función "suma" las variables mencionadas no existen y no es posible utilizarlas ya que en la función "suma" no existen.

Dentro de la función "suma" las variables que existen son a , b , total que son variables locales de suma y existen solamente mientras se esta ejecutando la función "suma".

Si una variable puede ser usada desde cualquier función y durante el transcurso de todo el programa, esa es una variable global. Las variables globales de definen fuera de cualquier función , normalmente debajo de los *include* que se colocan al comienzo del programa.

```
#include <stdio.h>

int var;
void carga(void);

void main(void)
{
	int x,y,z;
	var=5;
	carga();
	printf("%d",var);
}

void carga(void)
{
	var=3;
}
```

La variable *var* es una variable global. Esta variable puede ser usada desde el *main* o desde la función "carga".
Cuando el programa comienza a correr se le asigna a la variable *var* el valor 5, luego se llama a la función y esta le asigna el valor 3. Finalmente cuando se muestra el valor de la variable *var* aparecerá un 3 que es el último valor asignado.

Se debe tener en cuenta que al tener una variable definida en forma global se puede acceder a ella desde cualquier parte del programa ya sea para asignarle un nuevo valor o para leerla. Veamos la siguiente modificación al programa anterior.

```
#include <stdio.h>

int var;
void carga(void);

void main(void)
{
	int x,y,z;
	var=5;
	carga();
	printf("%d",var);
}

void carga(void)
{
	int var;
	var=3;
}
```

¿Cuál se supone que será el resultado mostrado en pantalla?
El resultado es 5.

Cuando comienza el programa, se le asigna a la variable var el valor 5, se llama a la función y esta le asigna un 3 a *var* para finalmente mostrar el valor 5. Si bien parece que hablamos siempre de la misma variable no es así, ya que a pesar que tienen el mismo nombre son distintas.

Dentro de *main* cuando se ejecuta var=5 se esta cargando la variable global ya que el compilador se fija primero si esta definida como local y después busca la global. Al no encontrar la local se la asigna a la global.

Dentro de la función "carga" esta definida una variable local con el nombre *var*, por lo tanto se hace la asignación de dicha variable que cuando termina la función y por ser una variable local ya no tiene más validez.

Entonces de acuerdo a esto podemos establecer algunas reglas:

1-La variable local tiene validez solo dentro de la función en la que fue definida, fuera de ella no existe.

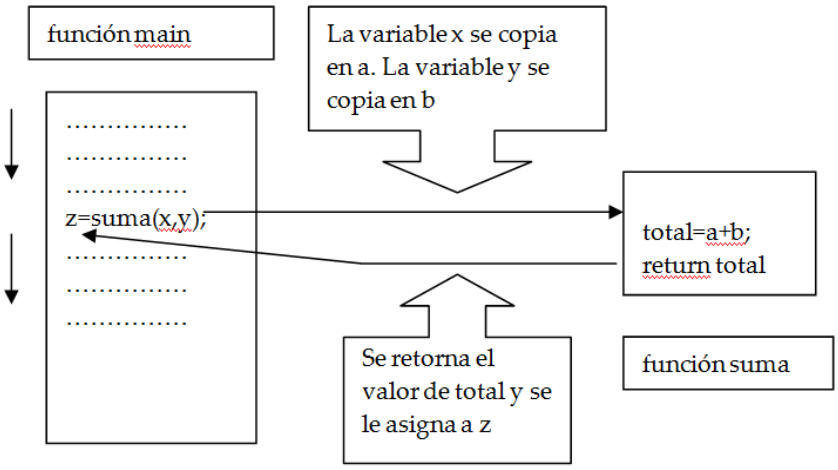
2-Si existe una variable global y otra local con el mismo nombre tiene validez la variable local dentro de esa función

3-Si en la función "a" tenemos la variable "nombre" y en la función "b" existe la variable "nombre" no existe problema de ningún tipo ya que ambas son variables locales de funciones distintas (aunque tengan el mismo nombre)

4-Es conveniente usar variables con distinto nombre para evitar confusiones.

1.3 Diagrama de flujo de un programa que usa funciones

Vamos a analizar cómo es la ejecución de un programa que utiliza funciones como por ejemplo el de la función suma.



Un programa siempre comienza por ***main***, se generan las variables y se empiezan a ejecutar las líneas del programa.

Cuando se llega al llamado de la función se produce un salto en el flujo del programa para ejecutar el código de la función. Antes de comenzar a ejecutarse el código se deben inicializar las variables de la función "suma", es decir se copia el valor de la variable "x" en la variable "a" y el de la variable "y" en "b" (este proceso se realiza automáticamente, no se debe agregar nada en el programa para hacerlo). Hecha la inicialización, se ejecuta el programa y al aparecer la instrucción ***return*** se devuelve el valor de total que se asigna a la variable "z" de *main*. A partir de allí se sigue la ejecución normal de *main*.

1.4 Pasaje de parámetros por valor y por referencia

Se llama pasaje por valor cuando a la función se le pasa como parámetro actual el valor de la variable.

Se llama pasaje por referencia cuando a la función se le pasa como parámetro actual la dirección de memoria de una variable. Tenga en cuenta que en este caso la variable que recibe deber ser un **puntero**, ya que lo que esta pasando es una dirección de memoria (Este tema se abordará más adelante, por lo que haremos referencia solo al pasaje por valor)

1.4.1 Pasaje por valor

```
#include <stdio.h>

void muestra(int x,int y);

void main(void)
{
	int x,y;

	printf("Ingrese un numero entero");
	 scanf("%d",&x);
	printf("Ingrese un numero entero");
	scanf("%d",&y);
	muestra(x,y);
	printf("\n-----valores dentro de main----");
	printf("\nx vale %d \ny vale %d",x,y);
}

void muestra(int x,int y)
{
	x=y;
	printf("\n-----valores dentro de la funcion----");
	printf("\nx vale %d \ny vale %d",x,y);
}
```

Cuando se ejecuta la función “muestra”, debido a la asignación que se encuentra en la primera línea, las variables x e y tienen el mismo valor. Por lo tanto cuando se ejecute el *printf* se muestra el valor de las 2 variables (que son variables locales de la función). Ahora cuando se ejecute el *printf* del *main* aparecen los valores que se cargaron en el primer momento sin ninguna modificación.

Esto ocurre por que aunque las variables tengan el mismo nombre, son distintas ya que son locales de cada función y como sabemos tienen validez solamente dentro de la función en la cual están definidas.