Universidad Nacional de la Matanza

Elementos de Programación

UNIDAD 6. FUNCIONES

INDICE

1. PROGRAMACIÓN MODULAR 2

2. FUNCIONES 2

3. PARÁMETROS Y VALOR DE RETORNO 3

4. DECLARACION DE VARIABLES - GLOBALES Y LOCALES 6

5. CODIFICACIÓN DE LAS FUNCIONES 6

5.1 DECLARACIÓN DE UNA FUNCIÓN – PROTOTIPO 7

5.2 LLAMADA A UNA FUNCIÓN. 8

5.3 DESARROLLO DE UNA FUNCIÓN 8

6. METODOLOGÍAS PARA LA PROGRAMACIÓN MODULAR 9

UNIDAD 6.
Funciones

OBJETIVOS*: Realizar programas partiendo de pequeñas partes conocidas, denominadas "subprogramas", y que en C se conocen como "Funciones". Crear nuevas funciones. Comprender y aplicar los mecanismos para el intercambio de información entre las funciones.*

1. Programación modular

Para simplificar la resolución de los problemas, basado en la programación estructurada, es posible la división del problema PRINCIPAL en pequeños problemas o sub-problemas de fácil solución y mantenimiento. Cada sub-problema se transformará en una FUNCION.

La solución de los problemas se efectúan desarrollando un Algoritmo, el cual estará compuesto por un tronco o "Algoritmo principal", acompañado de una serie de sub-algoritmos, que unidos adecuadamente resuelven el problema.

El algoritmo principal genera el programa principal, que en C es una función llamada "main". Los sub-algoritmos serán funciones con nombres propios, definidos por el programador.

En general un programa, va a constar de un "tronco" ó programa principal y una serie de subprogramas vinculados con éste. Cuando se ejecuta el programa que contiene funciones, se transfiere en cada llamada, el control al subprograma, se ejecutan sus instrucciones y retorna el control al programa llamador, es como si estuviese intercalado dentro del programa principal.

Las funciones constituyen una herramienta esencial en la programación modular. Por un lado, permiten desarrollar sólo una vez procedimientos (rutinas o subrutinas) que son de utilización frecuente o que se utilizan en diferentes lugares de un mismo programa; por otro lado, permiten utilizar mecanismos de abstracción que facilitan la construcción de un programa concentrándose en los aspectos más relevantes y despreocupándose de detalles que pueden ser encarados en otra etapa del diseño.

Una función es, básicamente, un subprograma; por lo tanto, tiene una estructura similar a la de cualquier programa:

Entradas → Proceso → Salida

Sólo que interactúa (se comunica) con otra función o directamente con el programa principal; dicha comunicación se establece a través de argumentos (parámetros) que conforman los datos (entradas) y resultados (salidas). Como cualquier programa, utilizan variables de trabajo denominadas variables locales proveyendo mecanismos de protección que hacen a éstas inaccesibles desde cualquier otro ámbito fuera de la misma función.

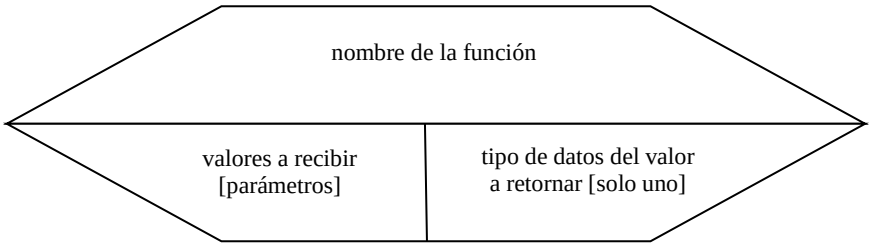
2. Funciones

Una función es una colección de declaraciones y sentencias que realizan una tarea específica, tiene cuatro componentes.

1. El nombre
2. La información que recibe o toma a través de los parámetros para realizar la tarea. (opcional)
3. Las sentencias que realizan la tarea
4. El valor que devuelve cuando termina su tarea (opcional).

Cada función se diagrama por separado, como si fuese otro programa. En la codificación serán incluidas a continuación de la función principal llamada main().

Para la primera línea de la "definición de la función" se utiliza un gráfico especial, que es el siguiente



Para las llamadas a las funciones se utiliza el gráfico que corresponde a una instrucción donde se invocará a la función por su nombre y se le enviarán los datos para utilizarla si es que los requiere.

Tanto los parámetros como el valor de retorno son opcionales, por lo tanto, podemos tener las siguientes combinaciones:

1. Funciones que NO reciben parámetros y NO retornan ningún valor: por ejemplo, una función que muestra un mensaje fijo, como un encabezado o título del programa.
2. Funciones que reciben parámetros y NO retornan ningún valor: por ejemplo, alguna función que realice algún cálculo y muestre el resultado dentro de la función. También se utiliza este tipo de funciones para los siguientes temas como arrays y archivos.
3. Funciones que NO reciben parámetros y retornan un valor: por ejemplo, para solicitar el ingreso de un dato, validarlo y retornar el número ingresado al programa principal.
4. Funciones que reciben parámetros y retornan un valor: por ejemplo, para funciones que realicen cálculos pero no lo muestren sino que retorne el resultado para ser utilizado en el programa principal.

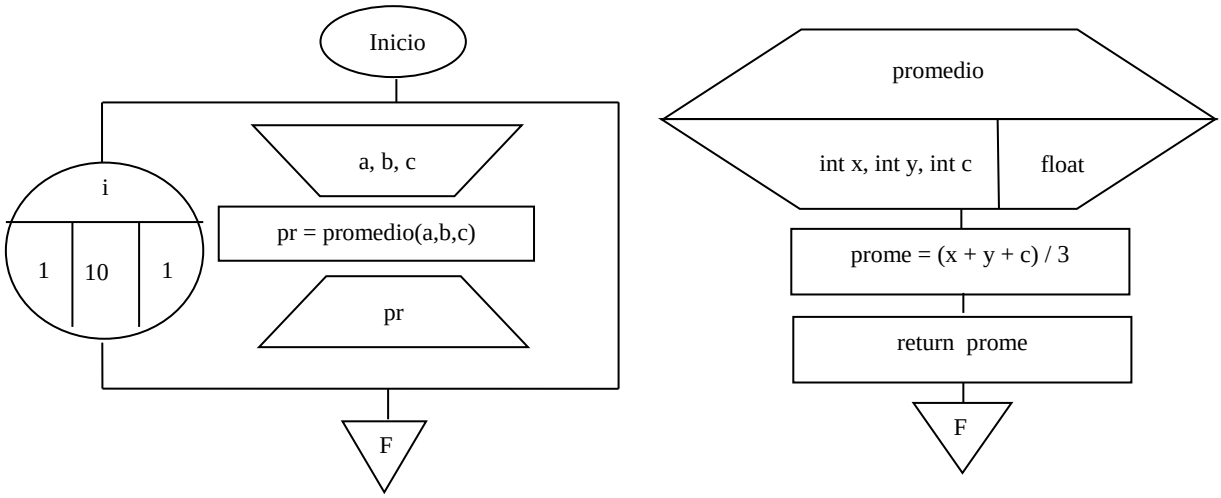
VENTAJAS DEL USO DE FUNCIONES

- Facilita la programación porque divide el problema en subproblemas.
- Simplifica la puesta a punto, ya que puedo corregir o cambiar una función sin tocar el resto del programa. Idem los futuros cambios.
- Permite trabajar simultáneamente a varios programadores.
- Permite la reutilización de estas funciones sin volver a programarlas, solo se utilizan.
- Ahorra memoria, reduce el tamaño ocupado por el programa cuando uso la misma función varias veces.

3. Parámetros y valor de retorno

A una función opcionalmente se le pueden enviar datos, esos datos son recibidos en la función para ser utilizados dentro de ella. Al especificar la función se detallan esos datos que recibe con su tipo de dato y un identificador que será el nombre de la variable LOCAL a la función donde se guardará el dato recibido. Estos datos que la función recibe se denominan parámetros formales. Veamos el siguiente ejemplo:

Ejemplo: Confeccionar un programa que solicite el ingreso de 10 ternas de valores reales y para cada una de las ternas calcule e informe el promedio de sus valores. Para el cálculo del promedio confeccionar y utilizar una función.

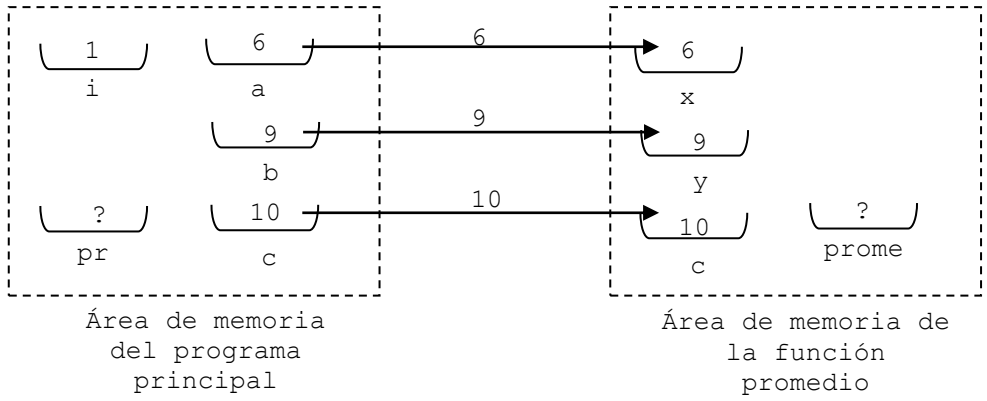


Como se mencionó anteriormente se realizan dos diagramas separados, un diagrama para el programa principal y otro diagrama para el desarrollo de la función. En este ejemplo se ingresan 3 números reales en el programa principal que se almacenan en las variables a, b y c. Luego, se invoca la función promedio enviándole como parámetro los tres valores recién leídos. Los parámetros trabajan por posición, es decir, que el primer valor que se indique en la llamada a la función va a ser guardado en una variable llamada x dentro de la memoria de la función promedio, el segundo la variable y , y el tercero en la variable c.

Las funciones trabajan con direcciones de memoria totalmente separadas a las del programa principal, al invocar a una función lo que se envía es un valor, es decir, el contenido de una variable y no la variable. Este valor es almacenado en un área distinta de memoria propia de la función. Cada vez que se invoca a una función se reserva un área de memoria separada distinta a la que se utilizó en otras llamadas. Además, al ser un área distinta de memoria es posible utilizar el mismo identificador que en el programa principal pero teniendo en cuenta que aún así es una variable diferente.

Suponga que el usuario ingresa los valores 6, 9 y 10 en el programa principal en las variables a, b y c respectivamente. Al invocar la función lo que se envían son esos números que se COPIAN en los parámetros de la función promedio, por cada parámetro se crea automáticamente una variable local a la función por lo que no deben volver a definirse. En la figura 1 puede ver un esquema de las áreas de memoria del programa principal y de la función promedio.

Figura 1: Esquema de memoria y envío de parámetros del programa principal a una función



Cada casillero representa un espacio de memoria con un identificador que es el nombre de la variable definida. El programa principal tiene 5 variables: k, utilizada como índice del ciclo for, pr para guardar el resultado del promedio y los tres valores ingresados por el usuario a, b y c. La primera vez que se ejecute el ciclo for k toma el valor 1 y a, b y c tomarán los valores ingresados por el usuario mediante el teclado. La variable pr, no tiene valor asignado aún por lo que no se sabe qué valor tiene, tiene un valor aleatorio que estaba ya en la memoria, pero no se sabe cuál es. Como no se inicializa la variable se dice que trae "basura" de memoria porque es un valor anterior que no tiene ningún sentido para el programa.

La función promedio tiene 4 variables ya que se crea automáticamente una variable para almacenar cada uno de los parámetros formales y además se declara dentro de la función una variable local llamada prome. Puede notar que la función promedio tiene una variable c al igual que el programa principal, pero son variables distintas porque están en áreas de memoria separadas. Dentro de la función solo se puede acceder a sus variables locales, por lo tanto, no puede acceder a las variables definidas en el main o en otras funciones. Cada función trabaja con sus propios datos lo que facilita que pueda reutilizarse.

Al invocar a la función desde el programa principal los valores de las variables a, b y c viajan hacían la función, se copian. Esos valores se denominan argumentos de la función y son recibidos y almacenados en las variables definidas automáticamente por los parámetros formales. Desde el programa principal como argumentos de una función se pueden enviar:

- Variables (se enviará el contenido de la misma)
- Constantes
- Operaciones (se resuelve la operación y se envía el resultado)
- Funciones que retornen un valor (al invocar a una función esta si retorna un valor, puede ser utilizado como parámetro de otra función y lo que se envía es el valor retornado por dicha función)

Dentro de la función se calcula el promedio y el resultado se guarda en la variable local de la función prome. Luego se utiliza un comando que permite retornar un valor al programa principal, el comando return. Este comando retorna un valor cuyo tipo de dato debe coincidir con el especificado como retorno en la función. En este caso la función debe retornar un valor float, por lo tanto, la variable local prome debe ser también float para poder ser retornada. El esquema de la figura 2 muestra el estado de la memoria al retornar el valor.

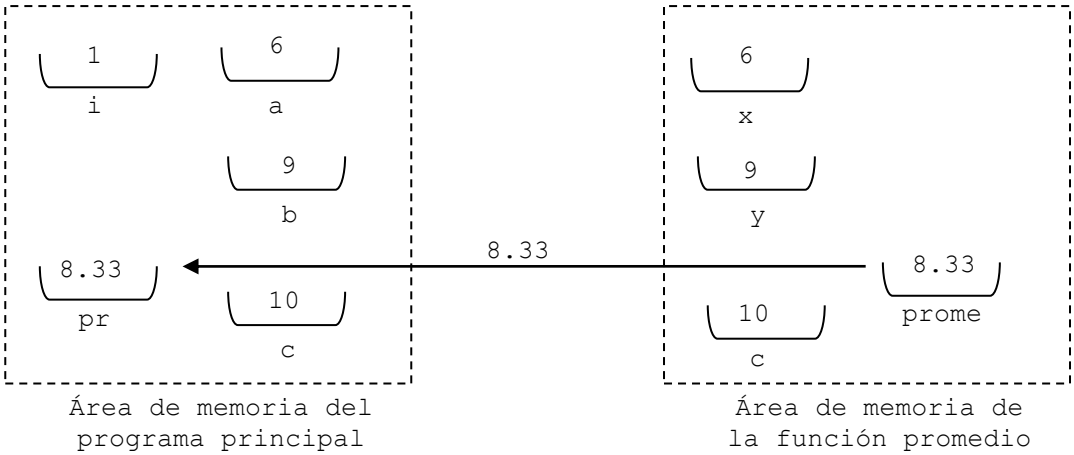


Figura 2: Esquema de memoria y envío del valor de retorno desde la función al programa principal

El valor de la variable prome retorna al programa principal como resultado de la función. Dicho resultado es almacenado en la variable pr. Si no se le asigna el resultado de la función a la variable con la instrucción pr = promedio(a,b,c) el valor de retorno se pierde, por lo tanto si se desea guardar el resultado de una función dicho resultado debe asignarse a una variable local desde donde se la invoca. Luego de retornar el valor la función finaliza, es decir, que siempre al llegar a

una instrucción return se sale de la función por lo que es recomendable usar solo una instrucción de este tipo al final de la misma para evitar confusiones. Al salir toda la memoria asignada a la función se pierde, ya no es accesible, por lo tanto, todo lo declarado y calculado dentro de la función se pierde.

4. Declaracion de variables - globales y locales

Existen 2 lugares en el programa donde se pueden definir variables:

a) Antes del bloque principal, éstas serán variables GLOBALES. Su ámbito de incumbencia es todo el programa, incluso todas las funciones que éste contenga.

   Esta definición debe evitarse debido a que puede complicar el programa ya que si se altera por error una variable global dentro de una función, dicho cambio se reflejará en todo el programa. Además, complica el uso de funciones realizadas por otras personas, ya que debería analizar los nombres y uso de todas las variables.

b) Dentro del main() (es una función más) ó dentro de cualquier función, estas variables se llaman LOCALES, porque su ámbito de incumbencia queda reducido solo a la función donde se la define.

   Las variables LOCALES ocupan memoria solo cuando se utiliza la función, por lo cual no es posible conservar datos en una variable local, entre 2 ejecuciones sucesivas de la misma función. El uso de las variables LOCALES, aparte de ahorrar memoria permite escribir FUNCIONES AUTOSUFICIENTES, que pueden usarse en otros programas sin más que copiarlas, evitando errores en el uso de las variables. Como ejemplo puede ver el caso de las Funciones de Biblioteca.

Algunas consideraciones:

- Pueden utilizarse variables con igual nombre en distintas funciones: esto puede realizarse, ya que la función busca primero a la variable a usar dentro de las variables definidas como locales, si no la encuentra pasa a buscarla dentro de las variables globales definidas en el programa llamador. Esta es la causa por la cual puedo usar el mismo nombre para una variable local de una función que para una global, o una definida en el main().
- En general, si se usan los mismos nombres se debe tener cuidado en no confundirse.
- La llamada a una función se puede realizar, en general, desde cualquier sentencia.
- La utilización más eficiente de las funciones es haciendo uso de los PARAMETROS, que son realmente quienes establecen la comunicación entre el programa principal (main) y las funciones, usando variables locales.

5. Codificación de las funciones

La estructura de una función es similar a la de la función principal main(), salvo que comienza su cabecera con "su nombre" en lugar de la palabra "main".

En “tres” lugares del programa se debe mencionar a cada función que se utiliza:

a) Declaración del prototipo (solo la primer línea, con los parámetros y sus tipos de datos, ó solo los tipos de datos), al comienzo del programa, luego de las directivas al preprocesador y antes de la línea main().
b) Uso de la función (llamada a la misma) dentro del programa, en el lugar adecuado. Puede usarse en el programa principal o dentro de otra función.
c) Desarrollo de la función completa, con su cabecera y sentencias que la componen. Este desarrollo se realiza debajo de la función main().

```
            #directivas al preprocesador (include y define)
            declaración de variables "globales";
(a) prototipo →{tipo/void}  nombre-función ([parámetros(solo el tipo) ] ) ;
            void main()
            {
               -   declaración de variables locales del main;
               -  sentencias;
(b) uso →      [var =] nombre-función ([argumentos]);
            }
(c) desarrollo→{tipo /void} nombre-función ([parámetros (tipo y nombre)])
            {
               -   declaración variables locales de la función;
               -   sentencias;
               [return expresión;]
            }
```

A continuación, se muestra la codificación del ejemplo presentado anteriormente para calcular el promedio de 10 ternas de valores ingresadas por teclado mediante una función.

```
#include <stdio.h>
float promedio (float,float ,float );//prototipo de la función
int main()
{
       float a, b, c, pr;
       int k;
       for (k=1;k<=10;k++)
       {
              printf("\nIngrese una terna de valores : ");
              scanf("%f%f%f",&a,&b,&c);
              pr = promedio (a,b,c); //llamada a la función
              printf ("\n Promedio = %6.2f", pr);
       }
       return 0;
}

float promedio(float x, float y, float c) //desarrollo de la función
{
       float prome;
       prome = (x+y+c)/3;
       return (prome);
}
```

5.1 Declaración de una función – Prototipo

Esta declaración, que comprende solo la cabecera de la función y que se hace previa al comienzo del main(), le permite al compilador conocer el nombre de la misma, conocer los parámetros y sus tipos de datos y luego poder chequear a éstos con los de la llamada y también el tipo del resultado. No se define el cuerpo de la función.

Una función no puede ser llamada si no se escribió su prototipo, salvo que la función sea declarada completa antes del main(), en dicho caso no es necesario el prototipo.

Si la función no recibe parámetros se pueden dejar los paréntesis vacios o escribir la palabra void, que es el tipo de dato vació, es decir, indica que no recibe nada. En el valor de retorno siempre hay que especificar un tipo de dato, si la función no retorna nada se DEBE especificar la palabra void. Si se olvida escribir un tipo de dato por defecto el compilador asume que la función retornará un int (entero). Es recomendable escribir siempre el tipo de dato de retorno para mayor claridad. Si se asigna un identificador a los parámetros en el prototipo los mismos serán ignorados.

```
[void ó tipo] nombre ([parámetros formales con tipos ó solo el tipo de datos] );
```

Ejemplos :

```
int suma (int, int, int);
void CalcularYMostrarPromedio(float, int);
char IngresarVocal();
int IngresaNumeroPar(void);
void MostraTitulo();
```

Orden de los prototipos: Si hay varias funciones y una función llama a otra debe ponerse primero el prototipo de la función más interna. Es decir, que: si la función "a" llama a la "b", debe definirse previamente la "b". Pero si a su vez la función “b” llama a una función “c” entonces el primer prototipo que se debe escribir es el de la función “c”, luego el de la “b”, y por último, el de la “a”. El desarrollo de las funciones puede hacerse en cualquier orden si los prototipos fueron escritos en la parte superior correctamente.

5.2 Llamada a una función.

Es similar al llamado a una función de biblioteca. La llamada significa la ejecución de la función.

Tiene la forma general:

```
[variable = ]  nombre  ( [ argumentos] ) ;
```

Por ejemplo, en esta llamada a la función potencia:

```
raiz = potencia (3.5 , a);
```

La variable raiz representa el lugar donde se guardará el resultado traído de la función; potencia es el nombre asignado a la función y los valores 3.5 y a, son los argumentos y son los valores que se envían a la función para ser asignados a sus parámetros formales. Ambas listas de parámetros deben coincidir en cantidad y tipo.

Ejemplos:

```
sum = suma (n1, n2, n3);
sum = suma (n1, 10, n3);
CalcularYMostrarPromedio(suma, 5);
CalcularYMostrarPromedio(suma, cant);
CalcularYMostrarPromedio(100, 5);
vocal = IngresarVocal();
num= IngresaNumeroPar();
MostraTitulo();
```

5.3 Desarrollo de una función

```
[void ó tipo] nombre ([parámetros formales con tipos de datos])
{
      [ declaración de variables locales ]
      sentencias;
      [ return  expresión ; ]
}
```

Tipo: si no se indica la palabra void significa que la función retorna UN VALOR, que puede ser de cualquier tipo, menos array (esto se verá en unidades posteriores). Por omisión es int

Nombre: es un identificador del nombre de la función

Parámetros formales: Los parámetros formales son las variables que reciben los valores pasados en la llamada a la función. Si no se reciben valores se deja vacía o se coloca la palabra void. Por cada parámetro formal se debe especificar el tipo de dato y el nombre.

Declaraciones: si se declaran variables, éstas serán LOCALES de la función.

Sentencias: Estas son las instrucciones que forman el cuerpo de la función

Valor retornado por una función: Cada función puede retornar un solo valor cuyo tipo se indica en la cabecera de la función. Este valor a retornar se debe indicar en la sentencia return puede ser una constante una variable o una expresión.

Ejemplos:

```
return 1;     //retorna siempre el valor 1
return a+b;    //retorna la suma del contenido de las variables a y b
return suma;  //retorna el contenido de la variable suma
```

6. Metodologías para la programación modular

La programación modular permite separar el problema en sub-problemas menores. Es decir, que parte de la lógica se puede delegar para que sea resuelta por una función lo que permite encarar el diseño de un algoritmo de dos formas distintas:

Botton-Up (ascendente)

Esta metodología se basa en comenzar a diseñar el algoritmo desarrollando primero las funciones o viendo que funciones se tienen para luego con esas funciones armar el diseño del programa principal. Es decir, que parte de lo particular para luego generar una solución al problema.

Top-Down (descendente)

Se realiza un diseño general del sistema centrándose en la lógica del programa principal, delegando algunas tareas en funciones que serán desarrolladas luego. Es decir, que primero se hace un diseño general y luego se va a lo particular especificando las partes de la solución que falten.