Guía 2
Funciones

Concepto de función, funciones sin parámetros y sin retorno

Hasta ahora hemos trabajado con una metodología de programación lineal. Todas las instrucciones de nuestro programa se ejecutan en forma secuencial de principio a fin.

Esta forma de organizar un programa solo puede ser llevado a cabo si el mismo es muy pequeño (decenas de líneas)

Cuando los problemas a resolver tienden a ser más grandes la metodología de programación lineal se vuelve ineficiente y compleja.

El paradigma de programación que propone el lenguaje C es la programación estructurada.

La programación estructurada busca dividir o descomponer un problema complejo en pequeños problemas. La solución de cada uno de esos pequeños problemas nos trae la solución del problema complejo.

En el lenguaje C el planteo de esas pequeñas soluciones al problema complejo se hace dividiendo el programa en funciones.

Una función es un conjunto de instrucciones que resuelven un problema específico. En el lenguaje C es obligatorio el planteo de una función como mínimo llamada main:

```
int main()
{
    ....
}
```

El lenguaje C tiene librerías que proveen algunas funcionalidades básicas. Algunas de ellas ya las utilizamos en conceptos anteriores como son las funciones: scanf, printf, strlen, strcpy etc.

Veamos ahora como crear nuestras propias funciones.

El tema de funciones en un principio puede presentar dificultades para entenderlo y ver sus ventajas ante la metodología de programación lineal que veníamos trabajando todo en la función main.

Los primeros problemas que presentaremos nos puede parecer que sea más conveniente trabajar todo en la función main en vez de dividir el problema en varias funciones.

A medida que avancemos veremos que si un programa empieza a ser más complejo (cientos de líneas, miles de líneas o más) la división en pequeñas funciones nos permitirá tener un programa más ordenado y fácil de entender y por lo tanto en mantener.

Estructura de una función.

Una función tiene un nombre, puede tener parámetros y encerrada entre llaves le sigue su algoritmo:

```
[valor que retorna]   [nombre de la función] ([parámetros de la función])
{
   [altoritmo]
}
```

Problema 1:

Confeccionar un programa que muestre una presentación en pantalla del programa. Solicite la carga de dos valores y nos muestre la suma. Mostrar finalmente un mensaje de despedida del programa.

Implementar estas actividades en tres funciones.

Programa: programa85.c

```
#include<stdio.h>


void presentacion()
{
   printf("Programa que permite cargar dos valores por teclado.\n");
   printf("Efectua la suma de los valores\n");
   printf("Muestra el resultado de la suma\n");
   printf("******************************\n");
}
```

```
void cargaSuma()
{
   int valor1,valor2,suma;
   printf("Ingrese el primer valor:");
   scanf("%i",&valor1);
   printf("Ingrese el segundo valor:");
   scanf("%i",&valor2);
   suma=valor1+valor2;
   printf("La suma de los dos valores es: %i",suma);
}
void finalizacion()
{
   printf("\n*******************************\n");
   printf("Gracias por utilizar este programa");
}
int main()
{
   presentacion();
   cargaSuma();
   finalizacion();
   getch();
   return 0;
}
```

Como podemos observar hemos dispuesto previo a la función main otras tres funciones.

Lo importante es tener en cuenta que todo programa en C comienza a ejecutarse siempre a partir de la primer línea contenida en la función main, en nuestro ejemplo sería:

```
int main()

{

   presentacion();
```

Lo que estamos indicando en esta línea es llamar a la función presentacion que debe estar codificada en líneas anteriores en nuestro archivo.

La codificación de la función presentacion es:

```
void presentacion()

{

   printf("Programa que permite cargar dos valores por teclado.\n");

   printf("Efectua la suma de los valores\n");

   printf("Muestra el resultado de la suma\n");

   printf("********************************\n");

}
```

Veamos un poco la sintaxis necesaria para crear una función en el lenguaje C.

Toda función tiene un nombre que no puede tener espacios en blanco, comenzar con un número ni tener caracteres especiales. Previo al nombre de la función indicamos el dato que devuelve la función, con la palabra clave void indicamos que la función no retorna valor (no requiere que tenga la palabra clave return como hemos dispuesto siempre en la función main)

Luego del nombre de la función indicamos entre paréntesis los parámetros, si no los tiene disponemos los paréntesis abiertos y cerrados y no disponemos un punto y coma al final:

```
void presentacion()
```

Luego del nombre de la función debe ir encerrado entre llaves el algoritmo propio de la función, en nuestro ejemplo disponemos una serie de llamadas a la función printf para mostrar la presentación del programa:

```
{

   printf("Programa que permite cargar dos valores por teclado.\n");

   printf("Efectua la suma de los valores\n");

   printf("Muestra el resultado de la suma\n");

   printf("********************************\n");
```

```
}
```

Cuando se ejecuta esta función al llegar a la llave de cerrado vuelve a la función main desde donde se la llamó.

De regreso en la función main ahora se llama a la función cargaSuma:

```
int main()

{

   presentacion();

   cargaSuma();
```

El programa ahora nuevamente sale de la función main y pasa a ejecutarse la función cargaSuma:

```
void cargaSuma()

{

   int valor1,valor2,suma;

   printf("Ingrese el primer valor:");

   scanf("%i",&valor1);

   printf("Ingrese el segundo valor:");

   scanf("%i",&valor2);

   suma=valor1+valor2;

   printf("La suma de los dos valores es: %i",suma);

}
```

La función cargaSuma no tiene parámetros y no devuelve datos (void)

Dentro del algoritmo de la función cargaSuma definimos tres variables llamadas locales a la función cargaSuma. Estas variables solo existen mientras se ejecuta la función cargaSuma y no pueden ser utilizadas por las otras funciones de nuestro programa.

El algoritmo de cargar dos valores por teclado, sumarlos y mostrar la suma por pantalla no varía con lo visto anteriormente, pero si varía la ubicación de dicho algoritmo dentro de una función creada por nosotros.

Cuando finaliza de ejecutarse cada una de las instrucciones de la función cargaSuma regresa a la función main donde se procederá a llamar a la tercera función que creamos nosotros llamada finalizacion:

```
int main()
{
   presentacion();
   cargaSuma();
   finalizacion();
```

Como vemos desde la main debemos indicar el nombre exacto de la función que llamamos (recordar que el lenguaje C es sensible a mayúsculas y minúsculas)

La función finalizacion tiene por objetivo mostrar una serie de mensajes por pantalla para que el operador sepa que el programa está finalizando:

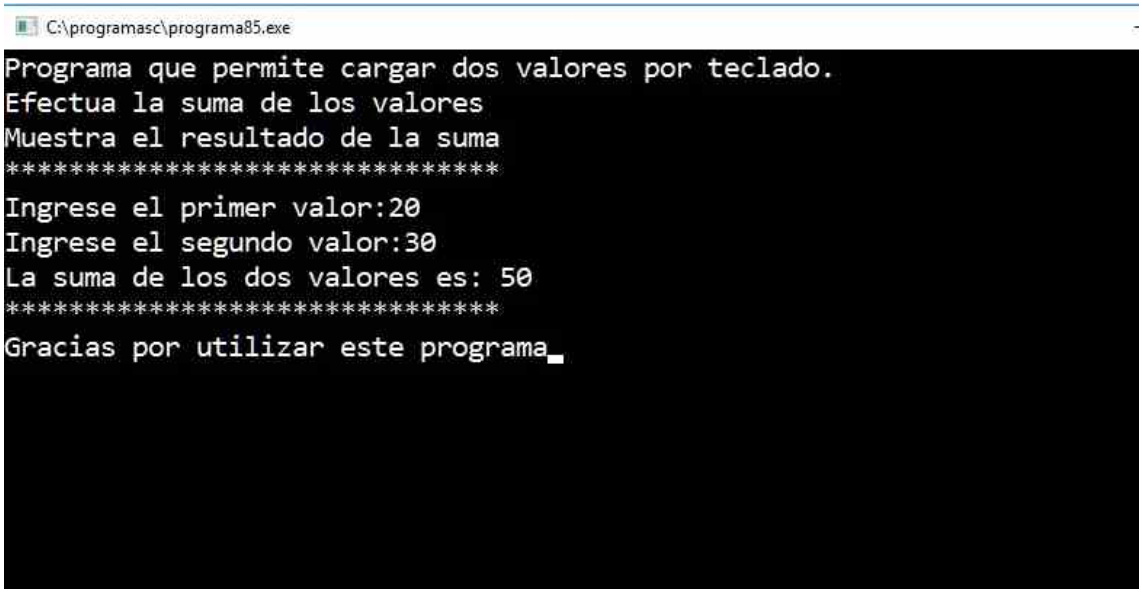
```
void finalizacion()
{
   printf("\n*****************************\n");
   printf("Gracias por utilizar este programa");
}
```

Luego de ejecutar los dos printf de la función finalizacion el programa regresa a la main:

```
int main()
{
   presentacion();
   cargaSuma();
   finalizacion();
   getch();
   return 0;
}
```

En la main solo queda la llamada a la función getch y la instrucción return que finaliza el programa.

Si ejecutamos el programa podemos ver una salida por pantalla similar a esta:



Recordar que cuando se llama a una función desde la main cuando la misma finaliza regresa a la siguiente línea de la main:

```c
#include<stdio.h>

void presentacion()
{
    printf("Programa que permite cargar dos valores por teclado.\n");
    printf("Efectua la suma de los valores\n");
    printf("Muestra el resultado de la suma\n");
    printf("**********\n");
}

void cargaSuma()
{
    int valor1,valor2,suma;
    printf("Ingrese el primer valor:");
    scanf("%i",&valor1);
    printf("Ingrese el segundo valor:");
    scanf("%i",&valor2);
    suma=valor1+valor2;
    printf("La suma de los dos valores es: %i",suma);
}

void finalizacion()
{
    printf("\n**********\n");
    printf("Gracias por utilizar este programa");
}


int main()
{
    presentacion();
    cargaSuma();
    finalizacion();
    getch();
    return 0;
}
```

Si no hacemos las llamadas a las funciones los algoritmos que implementan las funciones previas a la main nunca se ejecutarán:

```
int main()
{
   getch();
   return 0;
}
```

El orden que llamamos a las funciones desde la main también es muy importante, si implementamos la main con este orden de llamadas a las funciones el resultado será muy distinto:

```
int main()
{
   finalizacion();
   cargaSuma();
   presentacion();
   getch();
   return 0;
}
```

Funciones con parámetro

Una función puede recibir uno o más parámetros para que sea más flexible y se adapte a varias situaciones.

Veremos primero funciones que reciban uno o más parámetros de tipo int, float o char.

La sintaxis cuando una función recibe parámetros es:

```
[valor que retorna]   [nombre de la función] ([parámetros de la función])
{
   [altoritmo]
}
```

Los parámetros se disponen luego del nombre de la función encerrado entre paréntesis y separados por coma, algunos ejemplos:

```
void funcion1(int v1)
{
   ....
}
void funcion2(int v1,int v2)
{
   ....
}
void funcion3(float f1,int v1)
{
   ....
}
void funcion4(char letra)
{
   ....
}
```

Arriba vemos como declarar funciones con uno o más parámetros de tipos int, float y char.

Problema 1:

Confeccionar una función que reciba dos enteros e imprima el mayor de ellos. Llamar a la función desde la main cargando previamente dos valores por teclado.

Programa: programa89.c

```
#include<stdio.h>


void imprimirMayor(int v1,int v2)
```

```
{
   if (v1>v2)
   {
       printf("El mayor es %i",v1);
   }
   else
   {
       printf("El mayor es %i",v2);
   }
}
int main()
{
   int valor1,valor2;
   printf("Ingrese primer valor:");
   scanf("%i",&valor1);
   printf("Ingrese segundo valor:");
   scanf("%i",&valor2);
   imprimirMayor(valor1,valor2);
   getch();
   return 0;
}
```

En este problema se requiere implementar una función que reciba dos enteros y muestre el mayor de ellos. Los parámetros van después del nombre de la función entre paréntesis y separados por coma:

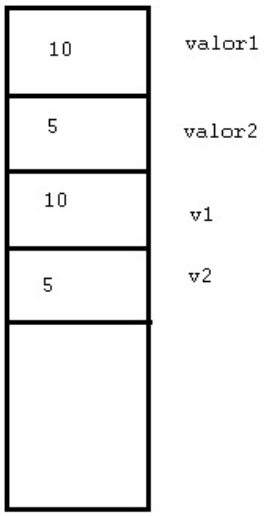
```
void imprimirMayor(int v1,int v2)
```

Los parámetros de la función solo se pueden utilizar dentro de dicha función y luego que finaliza de ejecutarse dicha función los mismos se borran de la memoria.

Cuando llamamos a la función desde la main le debemos pasar dos variables enteras, las variables que le pasamos deben estar cargadas con información:

```
imprimirMayor(valor1,valor2);
```

En el momento que se llama la función el contenido de valor1 se copia en el parámetro v1 y el contenido de valor2 se copia en el parámetro v2:



Si por algún motivo modificamos v1 o v2 dentro de la función las variables definidas en la main no cambian.

Los parámetros son la forma de comunicarse entre una función y otra.

Podemos inclusive desde la main llamar a la función inprimirMayor pasando directamente valores enteros:

```
imprimirMayor(70, 5);
```

Tengamos en cuenta que en este caso v1 recibe el número 70 y v2 recibe el número 5.

Problema 2:

Confeccionar un programa que solicite el pago por hora de un empleado y la cantidad de horas trabajadas dentro de una estructura repetitiva en la función main. Elaborar una función que reciba como parámetro el valor de la hora y la cantidad de horas trabajadas y nos muestre el total a pagar.

Programa: programa90.c

```
#include<stdio.h>


void calcularSueldo(float costoHora, int cantidadHoras)
```

```
{
   float sueldo=costoHora * cantidadHoras;
   printf("El sueldo total a pagar es %0.2f\n",sueldo);
}
int main()
{
   float costoHora;
   int cantidadHoras;
   char opcion;
   do {
      printf("Ingrese cuanto se le paga por hora:");
      scanf("%f",&costoHora);
      printf("Cuantas horas trabajo:");
      scanf("%i",&cantidadHoras);
      calcularSueldo(costoHora, cantidadHoras);
      printf("Desea calcular los datos de otro empleado[s/n]:");
      scanf(" %c",&opcion);
   } while (opcion=='s');
   getch();
   return 0;
}
```

Para resolver este problema planteamos una función que reciba como parámetros un float y un int, dentro del bloque de la función definimos una variable local llamada sueldo y procedemos a guardar el producto de los dos parámetros. Mostramos finalmente el sueldo total a pagar (damos formato al valor float para mostrar dos decimales):

```
void calcularSueldo(float costoHora, int cantidadHoras)
{
```

```
   float sueldo=costoHora * cantidadHoras;

   printf("El sueldo total a pagar es %0.2f\n",sueldo);

}
```

En la función main procedemos a llamar a la función calcularSueldo y le pasamos a los parámetros dos variables que definimos en la main:

```
      calcularSueldo(costoHora, cantidadHoras);
```

No hay ningún problema que las variables que definimos en la función main se llamen igual que los parámetros definidos en la otra función:

```
int main()

{

   float costoHora;

   int cantidadHoras;
```

Y los parámetros de la otra función:

```
void calcularSueldo(float costoHora, int cantidadHoras)
```

Pero recordar que en la memoria de la computadora se almacenan en lugares distintos las variables de la función main y los parámetros de la función calcularSueldo.

Problema 3:

Desarrollar dos funciones que reciban como parámetro el valor del lado de un cuadrado. La primera debe calcular y mostrar la superficie y la segunda calcular y mostrar el perímetro.

En la main llamar a las funciones pasando los valores enteros comprendidos entre 10 y 20

Programa: programa91.c

```
#include<stdio.h>


void calcularSuperficie(int lado)

{

   int superficie=lado*lado;

   printf("La superficie de un cuadrado de lado %i es de %i\n",lado,superficie);
```

```
}
void calcularPerimetro(int lado)
{
   int perimetro=lado*4;
   printf("El perimetro de un cuadrado de lado %i es de %i\n",lado,perimetro);
}
int main()
{
   int x;
   for(x=10;x<=20;x++)
   {
      calcularSuperficie(x);
      calcularPerimetro(x);
   }
   getch();
   return 0;
}
```

Hemos planteado dos funciones que reciben un parámetro de tipo int y cada una implementa el algoritmo:

```
void calcularSuperficie(int lado)
{
   int superficie=lado*lado;
   printf("La superficie de un cuadrado de lado %i es de %i\n",lado,superficie);
}


void calcularPerimetro(int lado)
{
```

```
    int perimetro=lado*4;
    printf("El perimetro de un cuadrado de lado %i es de %i\n",lado,perimetro);
}
```

Desde la función main llamamos a dichas funciones 11 veces a cada una pasando en la primer llamada el valor 10, ya que x vale 10:

```
    for(x=10;x<=20;x++)
    {
        calcularSuperficie(x);
        calcularPerimetro(x);
    }
```

Problema 4:

Desarrollar una función que reciba como parámetro un caracter. Si llega una 'h' mostrar por pantalla el mensaje "hombre", si llega una 'm' mostrar el mensaje "mujer".

Llamar desde la función main pasando una vez una 'h' y otra vez una 'm'.

Programa: programa92.c

```
#include<stdio.h>


void mostrarGenero(char tipo)
{
    if (tipo=='h')
    {
        printf("Hombre\n");
    }
    if(tipo=='m')
    {
        printf("Mujer\n");
    }
```

```
}
```

```
int main()
{
   mostrarGenero('h');
   mostrarGenero('m');
   getch();
   return 0;
}
```

La función mostrarGenero recibe un solo parámetro de tipo char. Dentro del bloque de las llaves según el valor que llega muestra el mensaje "Hombre", "Mujer" según corresponda:

```
void mostrarGenero(char tipo)
{
   if (tipo=='h')
   {
      printf("Hombre\n");
   }
   if(tipo=='m')
   {
      printf("Mujer\n");
   }
}
```

Desde la función main llamamos a la función mostrarGenero y le pasamos directamente el caracter 'h' en la primer llamada y el caracter 'm' en la segunda llamada:

```
int main()
```

```
{
   mostrarGenero('h');
   mostrarGenero('m');
```

Recordar que cuando llamamos a la función no es obligatorio pasar variables sino podemos pasar un valor del mismo tipo que tiene el parámetro.

Funciones con retorno

Estamos viendo en los últimos conceptos la metodología de programación estructurada. Buscamos dividir un problema en subproblemas y plantear funciones que los resuelvan.

Hemos visto que una función la definimos mediante un nombre y que puede recibir datos por medio de sus parámetros.

Los parámetros son la forma para que una función reciba datos para ser procesados. Ahora veremos otra característica de las funciones que es la de devolver un dato a quien invocó la función (recordemos que una función la podemos llamar desde la función main o inclusive desde otras funciones de nuestro programa)

Problema 1:

Confeccionar una función que le enviemos como parámetro el valor del lado de un cuadrado y nos retorne su superficie.

Programa: programa96.c

```
#include<stdio.h>


int retornarSuperficie(int lado)
{
   int superficie=lado*lado;
   return superficie;
}
int main()
{
   int valor;
   int sup;
```

```
    printf("Ingrese el valor del lado del cuadrado:");
    scanf("%i",&valor);
    sup=retornarSuperficie(valor);
    printf("La superficie del cuadrado es %i",sup);
    getch();
    return 0;
}
```

Cuando una función retorna un valor indicamos previo al nombre de la función el tipo de dato que retorna (puede ser alguno de los tipos de datos que conocemos: char, int, float):

```
int retornarSuperficie(int lado)
```

En nuestro ejemplo la función retornarSuperficie devuelve un int.

La función retornarSuperficie recibe un parámetro llamado lado, definimos una variable local llamada superficie donde almacenamos el producto del parámetro lado por sí mismo.

La variable local superficie es la que retorna la función mediante la palabra clave return:

```
int retornarSuperficie(int lado)
{
    int superficie=lado*lado;
    return superficie;
}
```

Hay que tener en cuenta que las variables locales (en este caso superficie) solo se pueden consultar y modificar dentro de la función donde se las define, no se tienen acceso a las mismas en la función main del programa o dentro de otra función.

Hay un cambio importante cuando llamamos o invocamos a una función que devuelve un dato:

```
    sup=retornarSuperficie(valor);
```

Es decir el valor devuelto por la función retornarSuperficie se almacena en la variable sup.

Es un error lógico llamar a la función retornarSuperficie y no asignar el valor a una variable:

```
retornarSuperficie(valor);
```

El dato devuelto por la función(en nuestro caso la superficie del cuadrado) no se almacena y se pierde sin que podamos imprimirlo.

Si podemos utilizar el valor devuelto para pasarlo a otra función como por ejemplo a printf:

```
printf("La superficie del cuadrado es %i",retornarSuperficie(valor));
```

La función retornarSuperficie devuelve un entero y se lo pasamos a la función printf para que lo muestre.

Ahora podemos entender porque la función main tiene obligatoriamente la palabra clave return al final devolviendo un 0. Es obligatorio que la función main retorne un int. El dato devuelto podrá ser capturado si nuestro programa es llamado desde otro programa.

Problema 2:

Confeccionar una función que defina dos parámetros enteros y nos retorne el mayor.

Programa: programa97.c

```
#include<stdio.h>


int retornarMayor(int v1,int v2)
{
   int mayor;
   if (v1>v2)
   {
      mayor=v1;
   }
   else
   {
```

```
        mayor=v2;
    }
    return mayor;
}
int main()
{
    int valor1,valor2;
    printf("Ingrese el primer valor:");
    scanf("%i",&valor1);
    printf("Ingrese el segundo valor:");
    scanf("%i",&valor2);
    printf("El valor mayor es %i",retornarMayor(valor1,valor2));
    getch();
    return 0;
}
```

Estamos definiendo una función que recibe dos parámetros y retorna el mayor de ellos:

```
int retornarMayor(int v1,int v2)
{
    int mayor;
    if (v1>v2)
    {
        mayor=v1;
    }
    else
    {
```

```
        mayor=v2;
    }
    return mayor;
}
```

Si queremos podemos hacerla más sintética a esta función sin tener que guardar en una variable local el valor a retornar:

```
int retornarMayor(int v1,int v2)
{
    if (v1>v2)
    {
        return v1;
    }
    else
    {
        return v2;
    }
}
```

Cuando una función encuentra la palabra return no sigue ejecutando el resto de la función sino que sale a la línea del programa desde donde llamamos a dicha función.