



Universidad Nacional Autónoma de México

Facultad de Ingeniería



Fundamentos de Programación (1122)

Laboratorios de computación
salas A y B

Profesor: M.I. Marco Antonio Martínez Quintana
Semestre 2021-1

Practica No. 12

Funciones

Grupo: 1129

No. de Equipo de cómputo empleado: No aplica

No. de Lista o Brigada: No aplica

No. de Lista: 42

Nombre: Adolfo Román Jiménez

Objetivo:

Elaborar programas en C donde la solución del problema se divida en funciones. Distinguir lo que es el prototipo o firma de una función y la implementación de ella, así como manipular parámetros tanto en la función principal como en otras.

Introducción:

Un programa en C consiste en funciones, las cuales pueden estar definidas en la librería estándar de Co bien definidas por el propio usuario. Las funciones definidas por el usuario constan de, el tipo de dato de la función, su nombre, y los parámetros que tendrán de entrada, así como a final de esta, el valor de retorno que aportará al código principal.

La función es un fragmento de código que realiza una tarea bien definida. pueden ser utilizadas por el programador. Este tipo de funciones predefinidas son denominadas funciones de biblioteca. Sin embargo, cada programador puede definir sus propias funciones de acuerdo a sus necesidades. Las funciones que define el programador son conocidas como funciones de usuario.

Una función especial en C es la función “*main*” la cual es de tipo entera, como ya se sabe, todo programa en C que es ejecutado inicia en esta función. Las funciones pueden o no ser escritas dentro del mismo documento, en caso de estar escritas dentro de este, se llamarán con el nombre de esta dentro del archivo, por el contrario, si esta es declarada en otro archivo, además de llamarla, se deberán compilar los dos archivos.

La utilización de funciones nos permite dividir un programa extenso en pequeños segmentos que realizan tareas concretas. Probablemente, dentro de un mismo programa se realicen las mismas tareas varias veces, lo que se facilita mediante la utilización de funciones

Funciones

La sintaxis básica para definir una función es la siguiente:

```
valorRetorno nombre (parámetros){  
    // bloque de código de la función  
}
```

El nombre de la función se refiere al identificador con el cual se ejecutará la función; se debe seguir la notación de camello.

Una función puede recibir parámetros de entrada, los cuales son datos de entrada con los que trabajará la función, dichos parámetros se deben definir dentro de los paréntesis de la función, separados por comas e indicando su tipo de dato, de la siguiente forma:

(tipoDato nom1, tipoDato nom2, tipoDato nom3...)

El tipo de dato puede ser cualquiera de los vistos hasta el momento (entero, real, carácter o arreglo) y el nombre debe seguir la notación de camello. Los parámetros de una función son opcionales.

El valor de retorno de una función indica el tipo de dato que va a regresar la función al terminar el bloque de código de la misma. El valor de retorno puede ser cualquiera de los tipos de datos vistos hasta el momento (entero, real, carácter o arreglo), aunque también se puede regresar el elemento vacío (void).

El compilador C revisa que las funciones estén definidas o declaradas antes de ser invocadas. Por lo que una buena práctica es declarar todas las funciones al inicio del programa. Una declaración, prototipo o firma de una función tiene la siguiente sintaxis:

valorRetorno nombre (*parámetros*);

La firma de una función está compuesta por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de la función; finaliza con punto y coma (;). Los nombres de los parámetros no necesariamente deben ser iguales a los que se encuentran en la definición de la función. Las funciones definidas en el programa no necesariamente deberán ser declaradas; esto dependerá de su ubicación en el código.

Código (funciones)

```
#include <stdio.h>
#include <string.h>

/*
    Este programa contiene dos funciones: la función main y la función
    imprimir. La función main manda llamar a la función imprimir. La función
    imprimir recibe como parámetro un arreglo de caracteres y lo recorre de fin a
    inicio imprimiendo cada carácter del arreglo.
*/

// Prototipo o firma de las funciones del programa
void imprimir(char[]);

// Definición o implementación de la función main
int main (){
    char nombre[] = "Facultad de Ingeniería";
    imprimir(nombre);
}

// Implementación de las funciones del programa
void imprimir(char s[]){
    int tam;
    for ( tam=strlen(s)-1 ; tam>=0 ; tam-- )
        printf("%c", s[tam]);
    printf("\n");
}
```

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /*
5  Este programa contiene dos funciones: la función main y la función imprimir.
6  La función main manda llamar a la función imprimir. La función imprimir
7  recibe como parámetro un arreglo de caracteres y lo recorre de fin a inicio
8  imprimiendo cada carácter del arreglo.
9  */
10
11 // Prototipo o firma de las funciones del programa
12 void imprimir(char[]);
13
14 // Definición o implementación de la función main
15 int main ()
16 {
17     char nombre[] = "Facultad de Ingeniería";
18     imprimir(nombre);
19 }
20
21 // Implementación de las funciones del programa
22 void imprimir(char s[])
23 {
24     int tam;
25     for (tam = strlen(s)-1 ; tam>=0 ; tam-- )
26         printf("%c", s[tam]);
27     printf("\n");
28 }
```

```
~/ $ clang -o test practica12.c
~/ $ ./test
aeinegnI ed datlucaF
~/ $
```

NOTA: *strlen* es una función que recibe como parámetro un arreglo de caracteres y regresa como valor de retorno un entero que indica la longitud de la cadena. La función se encuentra dentro de la biblioteca *string.h*, por eso se incluye ésta al principio del programa.

Ámbito o alcance de las variables

Las variables declaradas dentro de un programa tienen un tiempo de vida que depende de la posición donde se declaren. En C existen dos tipos de variables con base en el lugar donde se declaren: variables locales y variables globales.

Como ya se vio, un programa en C puede contener varias funciones. Las variables que se declaren dentro de cada función se conocen como variables locales (a cada función). Estas variables existen al momento de que la función es llamada y desaparecen cuando la función llega a su fin.

```
void sumar() {  
    int x;  
    // ámbito de la variable x  
}
```

Las variables que se declaran fuera de cualquier función se llaman variables globales. Las variables globales existen durante la ejecución de todo el programa y pueden ser utilizadas por cualquier función.

```
#include <stdio.h>  
  
int resultado;  
  
void multiplicar() {  
    resultado = 5 * 4;  
}
```

Código (Ámbito de las variables)

```
#include <stdio.h>

/*
    Este programa contiene dos funciones: la función main y la función incremento. La
    función main manda llamar a la función incremento dentro de un ciclo for. La función
    incremento aumenta el valor de la variable enteraGlobal cada vez que es invocada.
*/

void incremento();

// La variable enteraGlobal es vista por todas
// las funciones (main e incremento)
int enteraGlobal = 0;

int main(){
    // La variable cont es local a la función main
    for (int cont=0 ; cont<5 ; cont++){
        incremento();
    }

    return 999;
}

void incremento(){
    // La variable enteraLocal es local a la función incremento
    int enteraLocal = 5;
    enteraGlobal += 2;
    printf("global(%i) + local(%i) = %d\n", enteraGlobal, enteraLocal,
    enteraGlobal+enteraLocal);
}
```

```

1  #include <stdio.h>
2
3  /*
4  Este programa contiene dos funciones: la función main y la función
5  incremento. La función main manda llamar a la función incremento
6  dentro de un ciclo for. La función incremento aumenta el valor de
7  la variable enteraGlobal cada vez que es invocada.
8  */
9
10 void incremento();
11
12 // La variable enteraGlobal es vista por todas
13 // las funciones (main e incremento)
14
15 int enteraGlobal = 0;
16
17 int main()
18 {
19     // La variable cont es local a la función main
20     for (int cont=0 ; cont<5 ; cont++)
21     {
22         incremento();
23     }
24
25     return 999;
26 }
27
28 void incremento()
29 {
30     // La variable enteraLocal es local a la función incremento
31     int enteraLocal = 5;
32     enteraGlobal += 2;
33     printf("global(%i) + local(%i) = %d\n", enteraGlobal, enteraLocal, enteraGlobal+enteraLocal);
34 }
35
36

```

```

~/ $ clang -o test practica12.c
~/ $ ./test
global(2) + local(5) = 7
global(4) + local(5) = 9
global(6) + local(5) = 11
global(8) + local(5) = 13
global(10) + local(5) = 15
~/ $

```

Argumentos para la función main

Como se mencionó anteriormente, la firma de una función está compuesta por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de la función.

La función main también puede recibir parámetros. Debido a que la función main es la primera que se ejecuta en un programa, los parámetros de la función hay que enviarlos al ejecutar el programa. La firma completa de la función main es:

```
int main (int argc, char ** argv);
```

La función main puede recibir como parámetro de entrada un arreglo de cadenas al ejecutar el programa. La longitud del arreglo se guarda en el primer parámetro (argument counter) y el arreglo de cadenas se guarda en el segundo parámetro (argument vector). Para enviar parámetros, el programa se debe ejecutar de la siguiente manera:

- En plataforma Linux/Unix
./nombrePrograma arg1 arg2 arg3 ...
- En plataforma Windows
nombrePrograma.exe arg1 arg2 arg3 ...

Esto es, el nombre del programa seguido de los argumentos de entrada. Estos argumentos son leídos como cadenas de caracteres dentro del *argument vector*, donde en la posición 0 se encuentra el nombre del programa, en la posición 1 el primer argumento, en la posición 2 el segundo argumento y así sucesivamente.

Código (argumentos función main)

```
#include <stdio.h>
#include <string.h>

/*
 Este programa permite manejar los argumentos enviados al ejecutarlo.
 */

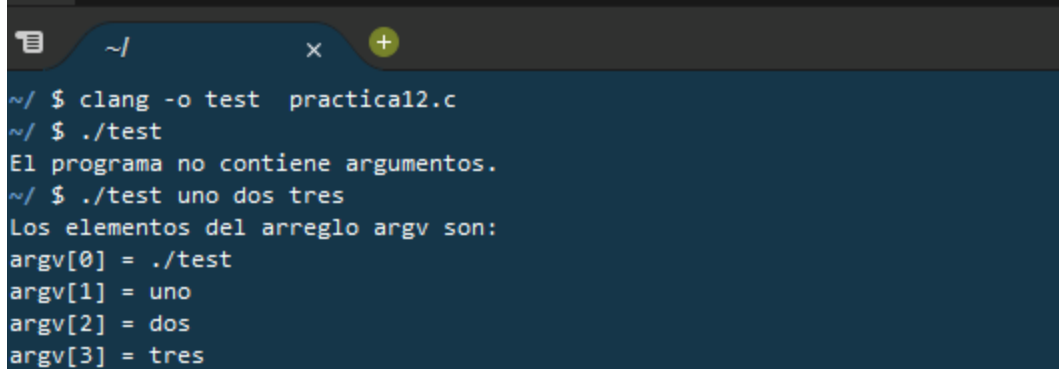
int main (int argc, char** argv){
    if (argc == 1){
        printf("El programa no contiene argumentos.\n");
        return 88;
    }

    printf("Los elementos del arreglo argv son:\n");
    for (int cont = 0 ; cont < argc ; cont++){
        printf("argv[%d] = %s\n", cont, argv[cont]);
    }

    return 88;
}
```



```
1 #include <stdio.h>
2 #include <string.h>
3
4 /*
5 Este programa permite manejar los argumentos enviados al ejecutarlo.
6 */
7
8 int main (int argc, char** argv)
9 {
10     if (argc == 1)
11     {
12         printf("El programa no contiene argumentos.\n");
13         return 88;
14     }
15
16     printf("Los elementos del arreglo argv son:\n");
17
18     for (int cont = 0 ; cont < argc ; cont++ )
19     {
20         printf("argv[%d] = %s\n", cont, argv[cont]);
21     }
22
23     return 88;
24 }
25
```



```
~/ $ clang -o test practica12.c
~/ $ ./test
El programa no contiene argumentos.
~/ $ ./test uno dos tres
Los elementos del arreglo argv son:
argv[0] = ./test
argv[1] = uno
argv[2] = dos
argv[3] = tres
```

Estático

Lenguaje C permite definir elementos estáticos. La sintaxis para declarar elementos estáticos es la siguiente:

```
static tipoDato nombre;
static valorRetorno nombre(parámetros);
```

Es decir, tanto a la declaración de una variable como a la firma de una función solo se le agrega la palabra reservada static al inicio de las mismas.

El atributo static en una variable hace que ésta permanezca en memoria desde su creación y durante toda la ejecución del programa, lo que quiere decir que su valor se mantendrá hasta que el programa llegue a su fin.

El atributo static en una función hace que esa función sea accesible solo dentro del mismo archivo, lo que impide que fuera de la unidad de compilación se pueda acceder a la función.

Código (variable estática)

```
#include <stdio.h>

/*
    Este programa contiene dos funciones: la función main y la función
    llamarFuncion. La función main manda llamar a la función llamarFuncion dentro
    de un ciclo for. La función llamarFuncion crea una variable estática e imprime
    su valor.
*/

void llamarFuncion();

int main (){
    for (int j=0 ; j < 5 ; j++){
        llamarFuncion();
    }
}

void llamarFuncion(){
    static int numVeces = 0;
    printf("Esta función se ha llamado %d veces.\n",++numVeces);
}
```

```

1  #include <stdio.h>
2
3  /*
4  Este programa contiene dos funciones: la
5  función main y la función llamarFuncion.
6  La función main manda llamar a la función
7  llamarFuncion dentro de un ciclo for.
8  La función llamarFuncion crea una variable
9  estática e imprime su valor.
10 */
11
12 void llamarFuncion();
13
14 int main ()
15 {
16     for (int j=0 ; j < 5 ; j++)
17     {
18         llamarFuncion();
19     }
20 }
21
22 void llamarFuncion()
23 {
24     static int numVeces = 0;
25     printf("Esta función se ha llamado %d veces.\n",++numVeces);
26 }
27

```

Código ejecutado:

```

~/ $ clang -o test practica12.c
~/ $ ./test
Esta función se ha llamado 1 veces.
Esta función se ha llamado 2 veces.
Esta función se ha llamado 3 veces.
Esta función se ha llamado 4 veces.
Esta función se ha llamado 5 veces.
~/ $

```

Una vez declarada una variable estática, esta permanece en memoria a lo largo de la ejecución del programa, por lo tanto, la segunda vez que se llama a la función ya no se vuelve a crear la variable, si no que se utiliza la que está en la memoria y por eso conserva su valor.

Código (función estática)

Este ejemplo consta de dos archivos: funcEstatica.c y calculadora.c.

```
//##### funcEstatica.c #####
#include <stdio.h>

/*
Este programa contiene las funciones de una calculadora básica: suma, resta, producto y
cociente.
*/

int suma(int,int);
static int resta(int,int);

int producto(int,int);
static int cociente (int,int);

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

static int resta (int a, int b){
    return a - b;
}

int producto (int a, int b){
    return (int)(a*b);
}
```

```
}  
  
static int cociente (int a, int b){  
    return (int)(a/b);  
}
```

```
//##### calculadora.c #####  
#include <stdio.h>  
  
/*  
Este programa contiene el método principal, el cual invoca a las funciones  
del archivo funcEstatica.c.  
*/  
  
int suma(int,int);  
//static int resta(int,int);  
int producto(int,int);  
//static int cociente (int,int);  
  
int main(){  
    printf("5 + 7 = %i\n",suma(5,7));  
    //printf("9 - 77 = %d\n",resta(9,77));  
    printf("6 * 8 = %i\n",producto(6,8));  
    //printf("7 / 2 = %d\n",cociente(7,2));  
}
```

```

1 //##### funcEstatica.c #####
2 #include <stdio.h>
3
4 /*
5 Este programa contiene las funciones de una calculadora básica: suma, resta, producto y cociente.
6 */
7
8 int suma(int,int);
9 static int resta(int,int);
10 int producto(int,int);
11 static int cociente (int,int);
12
13
14 int suma (int a, int b)
15 {
16     return a + b;
17 }
18
19 static int resta (int a, int b)
20 {
21     return a - b;
22 }
23
24 int producto (int a, int b)
25 {
26     return (int)(a*b);
27 }
28
29 static int cociente (int a, int b)
30 {
31     return (int)(a/b);
32 }
33
34

```

C:\Users\angel\OneDrive\Escritorio\Lenguaje C\Practica 12\funcEstatica.c - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?

funcEstatica.c calculadora.c

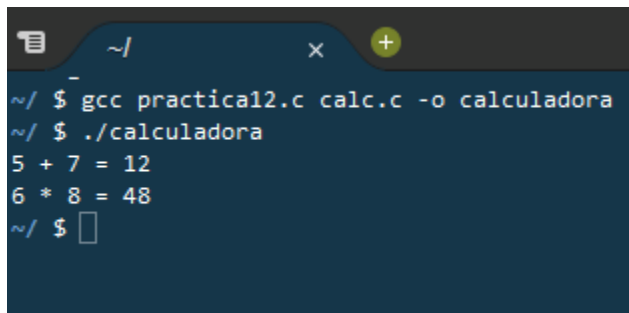
```

1 //##### funcEstatica.c #####
2 #include <stdio.h>
3 /*
4 Este programa contiene las funciones de una calculadora básica: suma, resta, producto y
5 cociente.
6 */
7 int suma(int,int);
8 static int resta(int,int);
9
10 int producto(int,int);
11 static int cociente (int,int);
12
13
14 int suma (int a, int b){
15     return a + b;
16 }
17 static int resta (int a, int b){
18     return a - b;
19 }
20 int producto (int a, int b){
21     return (int) (a*b);
22 }
23 static int cociente (int a, int b){
24     return (int) (a/b);
25 }

```

- Calculadora.c

```
2 #include <stdio.h>
3
4 /*
5 Este programa contiene el método principal,
6 el cual invoca a las funciones del archivo funcEstatica.c.
7 */
8
9 int suma(int,int);
10 //static
11 int resta(int,int);
12 int producto(int,int);
13 //static
14 int cociente (int,int);
15
16 int main()
17 {
18     printf("5 + 7 = %i\n",suma(5,7));
19     //printf("9 - 77 = %d\n",resta(9,77));
20     printf("6 * 8 = %i\n",producto(6,8));
21     //printf("7 / 2 = %d\n",cociente(7,2));
22 }
23
```



```
~/ $ gcc practica12.c calc.c -o calculadora
~/ $ ./calculadora
5 + 7 = 12
6 * 8 = 48
~/ $
```

Cuando se compilan los dos archivos al mismo tiempo (gcc funcEstatica.c calculadora.c -o exe), las funciones suma y producto son accesibles desde el archivo calculadora y, por tanto, se genera el código ejecutable. Si se quitan los comentarios y se intenta compilar los archivos se enviará un error, debido a que las funciones son estáticas y no pueden ser accedidas fuera del archivo funcEstaticas.c.

Scrabble.c

El siguiente código es un juego de scrabble en el cual a través de una función se evalúa el puntaje de las palabras que cada usuario ingresa a el y cuyas puntos están ordenados en un arreglo cuyo cada lugar del abecedario corresponde al numero de puntos que se le da a esa letra. La función que se declara antes de que el programa inicie se encuentra desglosada abajo y otorga puntos dependiendo de las letras usada en cada palabra. Y consecuentemente, declara un ganador o un empate.

```

1 #include <ctype.h>
2 #include <cs50.h>
3 #include <stdio.h>
4 #include <string.h>
5
6 // Points assigned to each letter of the alphabet
7 int POINTS[] = {1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10};
8
9
10 //Declaration of function contained inside the main function
11 int compute_score(string word);
12
13 int main(void)
14 {
15     // Get input words from both players
16     string word1 = get_string("Player 1: ");
17     string word2 = get_string("Player 2: ");
18
19     // Score both words
20     int score1 = compute_score(word1);
21     int score2 = compute_score(word2);
22
23     // TODO: Print the winner
24     if (score1 > score2)
25     {
26         printf("Player 1 wins!\n");
27     }
28     else if (score1 < score2)
29     {
30         printf("Player 2 wins!\n");
31     }
32     else
33     {
34         printf("Tie!\n");
35     }
36 }
37

```

```

37
38 int compute_score(string word)
39 {
40     // TODO: Compute and return score for string
41
42     //Declares addition of score
43     int count = 0;
44
45     //Iterates inside the letters of each word
46     for (int i = 0, n = strlen(word); i < n; i++)
47     {
48         // Makes every letter capital
49         char letter = toupper(word[i]);
50
51         //Assigns points to each letter referencig the POINTS array @ the beginning
52         if (letter >= 'A' && letter <= 'Z')
53         {
54             count += POINTS[letter - 65]; //Capital letters begin from 65 in ASCII so we just subtract that for score in POINTS
55         }
56         else
57         {
58             count += 0;
59         }
60     }
61
62     return count; // Returns result of score
63 }
64

```



```
~/ $ ./practical13
Player 1: fundamentos
Player 2: programacion
Player 2 wins!
~/ $ ./practical13
Player 1: science
Player 2: computer
Player 2 wins!
~/ $ ./practical13
Player 1: people
Player 2: people
Tie!
~/ $
```

Funcion de Gauss para factorial

gauss.c — C:\Users\adolf\Desktop — Atom

File Edit View Selection Find Packages Help

	Welcome Guide	Welcome	scrabble.c	hello.c
1	#include <stdio.h>			
2	int gauss(int n)			
3	{			
4	int r = 0;			
5	for(int i = 1; i<=n; i++)			
6	{			
7	r=r+i;			
8	}			
9	return r;			
10	}			
11	int main()			
12	{			
13	//Declarar variables			
14	char au = 163, sp=168, aa=160;			
15	int n,res;			
16	//Mensaje de bienvenida			
17	printf("\n\n\t\t\tSuma de los primeros n n%cmeros\n\n",au);			
18				
19	//Solicitar el ndmero de elementos a sumar			
20	printf("%cCu%cntos n%cmeros deseas sumar? " ,sp,aa,au) ;			
21	scanf("%d",&n);			
22				
23	//Mandamos llamar nuestra función gauss res-gauss (n);			
24	//Mostrar el resultado			
25	printf("La suma de los primeros %d n%cmeros es: %d \n",n,au,res);			
26				
27	return 0;			
28	}			
29				

```
C:\MinGW\bin> gcc -o gauss.exe ../../Users/adolfo/Desktop/ gauss.c
```

```
C:\MinGW\bin>gauss.exe
```

```
Suma de los primeros n números
```

```
¿Cuántos números deseas sumar? 5 4
```

```
La suma de los primeros 5 números es: 3371008
```

```
C:\MinGW\bin>gauss.exe
```

```
Suma de los primeros n números
```

```
¿Cuántos números deseas sumar? 10
```

```
La suma de los primeros 10 números es: 2404352
```

```
C:\MinGW\bin>
```

Ejercicio 6

Crear una función con su código del factorial y probarla en su calculadora.

```
1  #include<stdio.h>
2  int main ()
3  {
4      //Declarar variables
5      int op,res,n1,n2,div,mod,fact,n,i;
6      char aa=160, ae=130, ai=161, ao=162, au=163, sp=168, cr=175;
7
8      //Mensaje de bienvenida
9      printf("\n\n\t\t\tCalculadora\n\n",au);
10
11     do
12     {
13         //Mostrar el menú
14         printf("\n1) Suma \n2) Resta \n3) Multiplicaci%cn \n4) Divisi%cn \n5) M%cdulo\n",au);
15
16         //Solicitar la opción
17         printf("Elige una opci%cn: ",ao);
18         scanf("%d",&op);
19
20         switch(op)
21         {
22             case 1:
23                 printf("Dame 2 n%cmoros separados por coma: ",au);
24                 scanf("%i,%i",&n1,&n2);
25
26                 res=n1+n2;
27                 printf("La suma de %d y %d es: %d \n",n1,n2,res);
28                 break;
29             case 2:
30                 printf("Dame 2 n%cmoros separados por coma: ",au);
31                 scanf("%i,%i",&n1,&n2);
32
33                 res=n1-n2;
34                 printf("La resta de %d y %d es: %d \n",n1,n2,res);
35                 break;
36             case 3:
37                 printf("Dame 2 n%cmoros separados por coma: ",au);
38                 scanf("%i,%i",&n1,&n2);
39
40                 res=n1*n2;
41                 printf("La multiplicaci%cn de %d y %d es: %d \n",n1,n2,res);
42                 break;
43             case 4:
44                 printf("Dame 2 n%cmoros separados por coma: ",au);
45                 scanf("%i,%i",&n1,&n2);
46
47                 div=n1/n2;
48                 printf("La divisi%cn de %d y %d es: %d \n",n1,n2,div);
49                 break;
50             case 5:
51                 printf("Dame 2 n%cmoros separados por coma: ",au);
52                 scanf("%i,%i",&n1,&n2);
53
54                 mod=n1%n2;
55                 printf("El M%cdulo de %d y %d es: %d \n",n1,n2,mod);
56                 break;
57             default:
58                 printf("Opci%cn no v%clida\n",au);
59         }
60     } while (op != 0);
61 }
```

```

37     printf("Dame 2 nmeros separados por coma: ",au);
38     scanf("%i,%i",&n1,&n2);
39
40     res=n1*n2;
41     printf("La multiplicacin de %d y %d es: %d \n",ao,n1,n2,res);
42     break;
43     case 4:
44         printf("Dame 2 nmeros separados por coma: ",au);
45         scanf("%i,%i",&n1,&n2);
46         if(n2==0)
47         {
48             printf("La divisin de %d y %d es: indefinida\n",ao,n1,n2);
49         }
50         else
51         {
52             div=n1/n2;
53             printf("La divisin de %d y %d es: %d \n",ao,n1,n2,div);
54         }
55         break;
56     case 5:
57         printf("Dame 2 nmeros separados por coma: ",au);
58         scanf("%i,%i",&n1,&n2);
59         if(n2==0)
60         {
61             printf("El mdulo de %d y %d es: indefinida \n",ao,n1,n2);
62         }
63         else
64         {
65             mod=n1%n2;
66             printf("El mdulo de %d y %d es: %d \n",ao,n1,n2,mod);
67         }
68         break;
69     case 6:
70     printf("Introduce el nmero positivo del cual deseas calcular el factorial y que sea diferente de 0 \n%c",au,cr);
71     scanf("%d",&n);
72

```

```

70 printf("Introduce el n mero positivo del cual deseas calcular el factorial y que sea diferente de 0 \n%c",au,cr);
71 scanf("%d",&n);
72
73 fact=1;
74 i=1;
75 while(i <= n)
76 {
77     fact = fact * i;
78     i++;
79 }
80
81 printf("\nEl factorial de %d es: %d \n",n,fact);
82 break;
83 case 7:
84     printf("%cCu ntos n meros deseas sumar? ",sp,aa,au);
85     scanf("%d",&n);
86
87     res=0;
88     i=1;
89     while(i<=n)
90     {
91         res=res+i;
92         i++;
93     }
94
95     printf("La suma de los primeros %d n meros es: %d \n",n,au,res);
96     break;
97 case 8:
98     printf("Elegiste Salir\n\n");
99     break;
100 default:
101     printf("Opci n no v lida!!!\n\n",ao,aa);
102 }
103
104 }
105 while(op!=8);
106 printf("Gracias por usar nuestro programa :) \n");
107 return 0;
108

```

```
C:\MinGW\bin>calc.exe

                          Calculadora

1) Suma
2) Resta
3) Multiplicación
4) División
5) Módulo
6) Factorial
7) Suma de los primeros n números
8) Salir
Elige una opción: 6
Introduce el número positivo del cual deseas calcular el factorial y que sea diferente de 0
»5

El factorial de 5 es: 120

1) Suma
2) Resta
3) Multiplicación
4) División
5) Módulo
6) Factorial
7) Suma de los primeros n números
8) Salir
Elige una opción:
```

Conclusiones:

En esta practica pudimos aprender como elaborar funciones, declararlas y agregarlas a los códigos para que puedan ejecutarse al momento de evaluar condiciones que necesitamos para que los programas se ejecuten correctamente.

Referencias:

- Facultad de ingeniería - UNAM. Manual de prácticas del laboratorio de Fundamentos de programación.: http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf