	Funciones
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M. I. Marco Antonio Martínez Quintana

Asignatura: Fundamentos de Programación

Grupo: 3

No. de Práctica 12

Integrante(s): Sánchez García Rocío

No. de Equipo de cómputo empleado: No aplica

No. de Lista: 47

Semestre: 2021-1

Fecha de entrega: Domingo 17 de enero de 2020

Observaciones:

CALIFICACIÓN: _____

Práctica 12

Funciones

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.

Actividades

- Implementar en un programa en C la solución de un problema dividido en funciones.
- Elaborar un programa en C que maneje argumentos en la función principal.
- En un programa en C, manejar variables y funciones estáticas.

Introducción

Como ya se mencionó, un programa en lenguaje C consiste en una o más funciones. C permite tener dentro de un archivo fuente varias funciones, esto con el fin de dividir las tareas y que sea más fácil la depuración, la mejora y el entendimiento del código.

En lenguaje C la función principal se llama *main*. Cuando se ordena la ejecución del programa, se inicia con la ejecución de las instrucciones que se encuentran dentro de la función *main*, y ésta puede llamar a ejecutar otras funciones, que a su vez éstas pueden llamar a ejecutar a otras funciones, y así sucesivamente.

Desarrollo

Funciones

La sintaxis 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 trabajara la función, dichos parámetros se deben definir dentro de 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 real, entero, 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.

El compilador C revisa que las funciones estén definidas o declaradas antes de ser invocadas, 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)

```c
#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");
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc funciones.c -o funciones.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>funciones.exe
a¡reinegnI ed datlucaF

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

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 variable globales.

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.

```c
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.

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

Código (ámbito de las variables)

```c
#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);
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc variableAmbito.c -o variableAmbito.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>variableAmbito.exe
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

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Argumentos para la función *main*

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 cadena se guarda en el sungo parámetro (*argument vector*).

Para enviar parámetros, el programa se debe ejecutar de la siguiente manera:

- La 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)

```c
#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;
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc argumentos.c -o argumentos.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>argumentos.exe
El programa no contiene argumentos.

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Estático

La sintaxis para declarar elementos estáticos es la siguiente:

```
static tipoDato nombre;

static valorRetorno nombre (parámetros);
```

Tanto a la declaración de una variable como a la firma de una función solo se agrega la palabra reservada *static* al inicio de las mismas.

El atributo *static* es una variable que hace que esta 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)

```c
#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 funci%cn se ha llamado %d veces.\n",162,++numVeces);
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc variableEstatica.c -o variableEstatica.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>variableEstatica.exe
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.

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

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, sino que se utiliza la que esta en la memoria y por eso conserva su valor.

Código (variable estática)

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

```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));
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc funcionEsca.c -o funcionEsca.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>funcionEsca.exe
5 + 7 = 12
6 * 8 = 48

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Cuando se compilan los dos archivos al mismo tiempo, 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 intentan compilar los archivos se enuncia un error, debido a que las funciones son estáticas y no pueden ser accedidas fuera del archivo.

➢ Actividades asignadas por el profesor

Funciones

Es un conjunto de código que se quiere utilizar en varias ocasiones y por lo general tiene entradas y salidas.

Sintaxis:

```
{
    //Instrucciones
    return valorRetorno;
}
```

Implementación:

Programar una función utilizando el código de la sumatoria de los primeros n números.

```c
#include<stdio.h>
int gauss(int n)
{
    int r=0;
    for(int i=1;i<=n;i++)
    {
        r = r+i;
    }
    return r;
}
int main()
{
    //Declarar las variables
    char au = 163,sp = 168,aa = 160;
    int n,res;
    //Mensaje de bienvenida
    printf("\n\tSuma de los primeros n n%cmeros\n\n",au);
    //Solicitar el número de elementos a solicitar
    printf("\t%cCu%cntos n%cmeros deseas sumar? ",sp,aa,au);
    scanf("%d",&n);
    //Mandamos llamar la función gauss
    res = gauss(n);

    //Mostrar el resultado
    printf("\n\tLa suma de los primeros %d n%Cmeros es: %d \n",n,au,res);
    return 0;
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc funciongauss.c -o funciongauss.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>funciongauss.exe

        Suma de los primeros n números

        ¿Cuántos números deseas sumar? 100

        La suma de los primeros 100 nmeros es: 5050

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>
```

Ejercicio

Crear una función con el código del factorial y probarla en la calculadora

```c
#include <stdio.h>
int factorial(int n)
{
    int f=1,i;
    i=1;
    while(i<=n)
    {
        f = f*i;
        i++;
    }
    return f;
}
int main()
{
    //Declarar las variables a utilizar
    int n1,n2,n,res,i,fact=1;
    char aa = 160,ao =162,au = 163,mod =37,op = '\0',sp =168;
    //Mensaje de bienvenida
    printf("\n\n\t\t\tCalculadora\n\n");
    do
    {
        //Mostrar menú
        printf("\n\t1) Suma\n\t2) Resta\n\t3) Multiplicaci%cn\n\t4) Divisi%cn y M%cdulo\n\t5) Factorial\n\t6) Suma de los primeros n n%cmeros\n\t7) Salir\n",ao,ao,ao,au);
        //Solicitar la opción
        printf("\n\tElige una operaci%cn: ",ao);
        scanf("%d",&op);
        switch(op)
        {
            case 1:
                //Solicitar las variables
                printf("\n\tIngresa dos n%cmeros enteros separados por coma: ",au);
                scanf("%i,%i",&n1,&n2);
                printf("\n\t%d + %d = %d\n ",n1,n2,n1+n2);
                break;
            case 2:
                //Solicitar las variables
                printf("\n\tIngresa dos n%cmeros enteros separados por coma: ",au);
                scanf("%i,%i",&n1,&n2);
                printf("\n\t%d - %d = %d\n",n1,n2,n1-n2);
                break;
            case 3:
                //Solicitar las variables
                printf("\n\tIngresa dos n%cmeros enteros separados por coma: ",au);
                scanf("%i,%i",&n1,&n2);
                printf("\n\t%d * %d = %d\n",n1,n2,n1*n2);
                break;
            case 4:
                //Solicitar las variables
                printf("\n\tIngresa dos n%cmeros enteros separados por coma: ",au);
                scanf("%i,%i",&n1,&n2);
                if (n1 > n2)
                {
                    if (n2==0)
                    {
                        printf("\n\tIndeterminaci%cn\n",ao);
                        printf("\n\tIngrese otro valor\n");
                    }
                    else
                    {
                        printf("\n\t%d / %d = %d \n",n1,n2,n1/n2);
                        printf("\n\t%d %c %d = %d \n",n1,mod,n2,n1%n2);
                    }

                }
                else
                {
                    if (n2 > n1)
                    {
                        if (n1==0)
                        {
                            printf("\n\tIndeterminaci%cn\n",ao);
                            printf("\n\tIngrese otro valor\n");
                        }
                        else
                        {
                            printf("\n\t%d / %d = %d \n",n2,n1,n2/n1);
                            printf("\n\t%d %c %d = %d \n",n2,mod,n1,n2%n1);
                        }
                    }
                }
                break;
            case 5:
                //Solicitar un número
                printf("\n\tDigita el n%cmero: ",au);
                scanf("%d",&n);
                //Multiplicar los números
                fact = factorial(n);
                //Mostrar el resultado
                printf("\n\tEl factorial del n%cmero %d es: %d \n",au,n,fact);
                break;
            case 6:
                //Solicitar el número de elementos a solicitar
                printf("\n\t%cCu%cntos n%cmeros deseas sumar? ",sp,aa,au);
                scanf("%d",&n);
                //Sumar los n números
                res = 0;
                i = 1;
                while(i<=n)
                {
                    res = res+i;
                    i++;
                }
                //Mostrar el resultado
                printf("\n\tLa suma de los primeros %d n%Cmeros es: %d \n",n,au,res);
                break;
            case 7:
                printf("\n\tHas salido de la calculadora\n");
                break;
            default:
                printf("\n\tOpci%cn no v%clida.\n",ao,aa);
                break;
        }
    }while(op!=7);
    return 0;
}
```

```
C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>gcc funcionFactorial.c -o funcionFactorial.exe

C:\Users\rocio\OneDrive\Escritorio\Lenguaje c\Ejemplos>funcionFactorial.exe


                    Calculadora


       1) Suma
       2) Resta
       3) Multiplicación
       4) División y Módulo
       5) Factorial
       6) Suma de los primeros n números
       7) Salir

       Elige una operación: 5

       Digita el número: 5

       El factorial del número 5 es: 120
```

Conclusiones

- Una función realizara las tareas que nosotros le asignemos en un inicio.
- Al hacer uso de estas funciones contamos con la oportunidad de volver retomarla sin necesidad de escribirla en el código una y otra vez, lo cual tiende a ahorrar una gran cantidad de espacio en caso de que esta tarea tenga que repetirse mas de una vez durante la ejecución del programa.

Bibliografía

Manual de prácticas del Laboratorio de Fundamentos de Programación, Facultad de ingeniería UNAM, recuperada el 11 de enero, en http://lcp02.fi-b.unam.mx/