	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Marco Antonio Martinez Quintana
Asignatura:	Fundamentos de Programación
Grupo:	03
No de Práctica(s):	Practica N°12
Integrante(s):	Torres Gracian Christian Ivan
No. de Equipo de cómputo empleado:	No Aplica
No. de Lista o Brigada:	55
Semestre:	Semestre N°1
Fecha de entrega:	
Observaciones:	

CALIFICACIÓN: _____

Guía práctica de estudio 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.

Licencia GPL de GNU

El software presente en esta práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
*/
```

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");
}
```

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);
}
```

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;
}
```

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);
}
```

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.

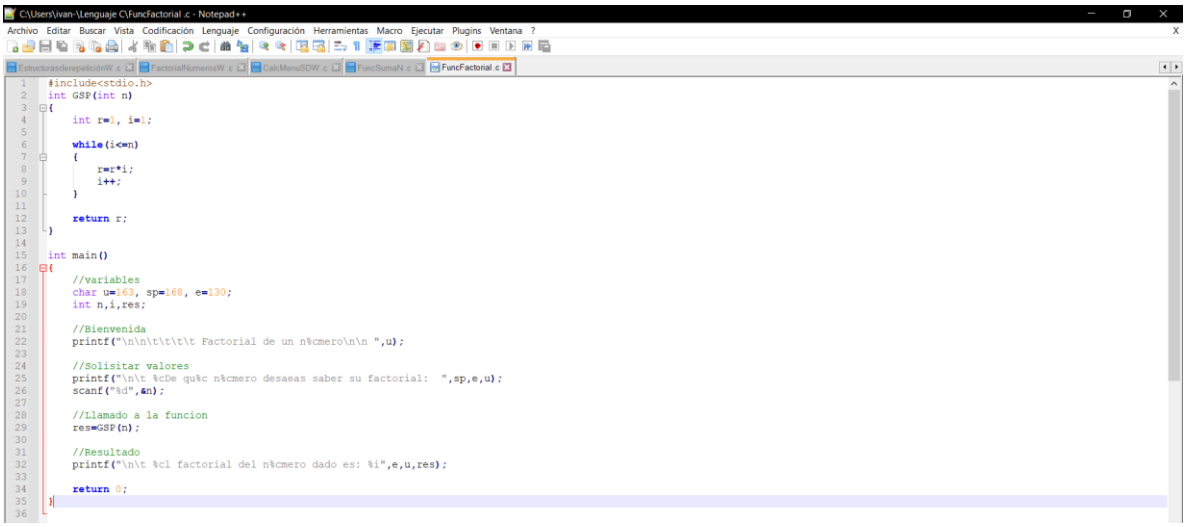
```
//################ funcEstatica.c ###############
#include <stdio.h>

/*
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);
```

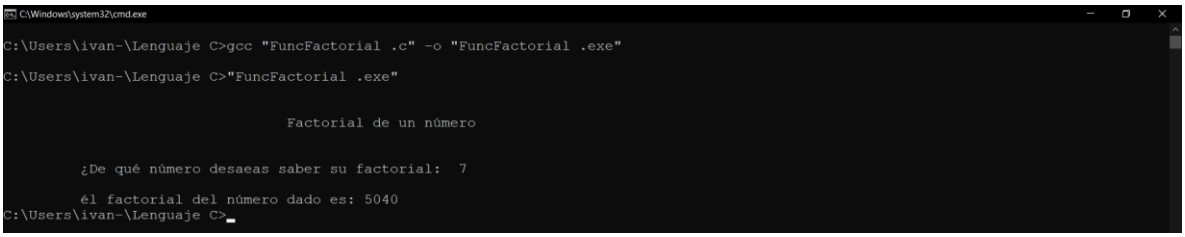
```
}

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));
}
```

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.

Resultados

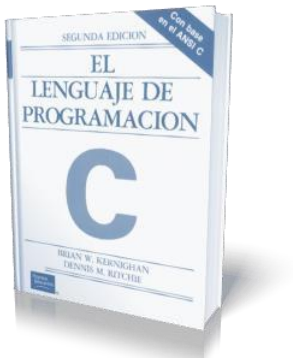

C:\Users\ivan-\Lenguaje C\FuncFactorial .c - Notepad++
Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?
FuncFactorial .c
#include<stdio.h>
int GSF(int n)
{
int r=1, i=1;
while(i<=n)
{
r=r*i;
i++;
}
return r;
}
int main()
{
//variables
char u=163, sp=168, e=130;
int n,i,res;
//Bienvenida
printf("\n\n\t\t\t\t Factorial de un n%cmero\n\n ",u);
//Solisitar valores
printf("\n\t %cDe qu%c n%cmero desaeas saber su factorial: ",sp,e,u);
scanf("%d",&n);
//Llamado a la funcion
res=GSF(n);
//Resultado
printf("\n\t %cl factorial del n%cmero dado es: %i",e,u,res);
return 0;
}


C:\Windows\system32\cmd.exe
C:\Users\ivan-\Lenguaje C>gcc "FuncFactorial .c" -o "FuncFactorial .exe"
C:\Users\ivan-\Lenguaje C>"FuncFactorial .exe"
Factorial de un número
¿De qué número desaeas saber su factorial: 7
él factorial del número dado es: 5040
C:\Users\ivan-\Lenguaje C>

```c
#include <stdio.h>
int Gauss(int n)
{
    int r=0,i=1;

    while(i<=n)
    {
        r=r+i;
        i++;
    }
    return r;

}

int main()
{
    //Variables
    char a=160;
    char u=163;
    char sp=168;
    int n,i,res;

    //Bienvenida
    printf("\n\n\t\t\t Suma de los primeros n n%cmeros \n\n",u);

    // Solisitar numeros
    printf("\t%cCu%cntos n%cmeros deseas  sumar?   ",sp,a,u);
    scanf("%i",&n);

    //Llamado a la funcion
    res=Gauss(n);

    //Resultado
    printf("\n\t La suma de los primeros %d n%cmeros es: %i",n,u,res);

```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.18363.1316]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ivan->cd "Lenguaje C"

C:\Users\ivan-\Lenguaje C>gcc "FuncSumaN .c" -o "FuncSumaN .exe"

C:\Users\ivan-\Lenguaje C>"FuncSumaN .exe"


                        Suma de los primeros n números

        ¿Cuántos números deseas  sumar?   100

         La suma de los primeros 100 números es: 5050
C:\Users\ivan-\Lenguaje C>
```

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.