



Carátula para entrega de prácticas

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(s): #12

Integrante(s): Cuevas Antunez Samantha

*No. de Equipo de
cómputo empleado:* No aplica

No. de Lista o Brigada: 12

Semestre: Primer semestre

Fecha de entrega: 11/01/2021

Observaciones:

CALIFICACIÓN: _____



OBJETIVOS

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

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.

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.

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. Tipos de variables con base en el lugar donde se declaren:

❖ Variables locales: variables que se declaren dentro de cada. 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  
}
```

- ❖ Variables globales: variables que se declaran fuera de cualquier función, 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;  
}
```

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 cadenas se guarda en el segundo parámetro (argument vector). Para enviar parámetros, el programa se debe ejecutar de la siguiente manera:

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.

Estático

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.

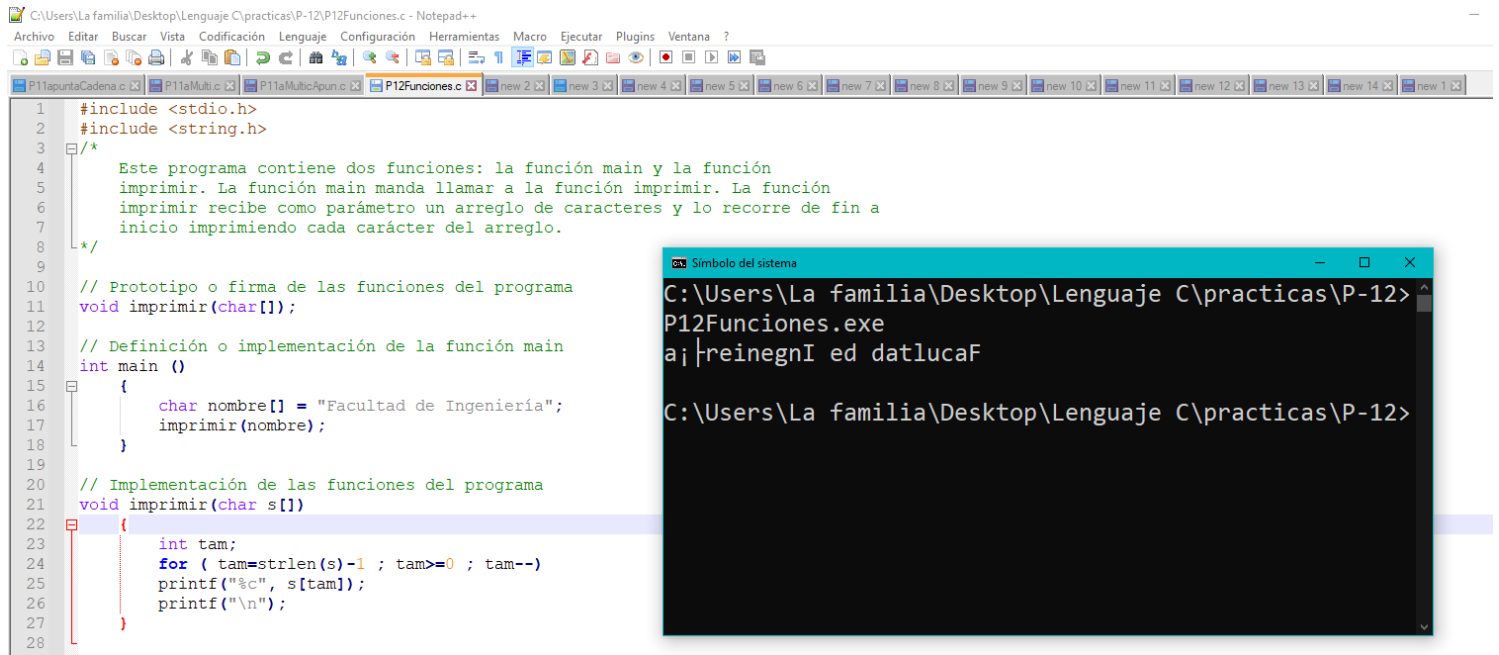
El atributo static en una variable hace que ésta permanezca en memoria desde su creación y durante toda la ejecución del programa, lo que quiere decir que su valor se mantendrá hasta que el programa llegue a su fin.

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.

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 está en la memoria y por eso conserva su valor.

RESULTADOS

Funciones

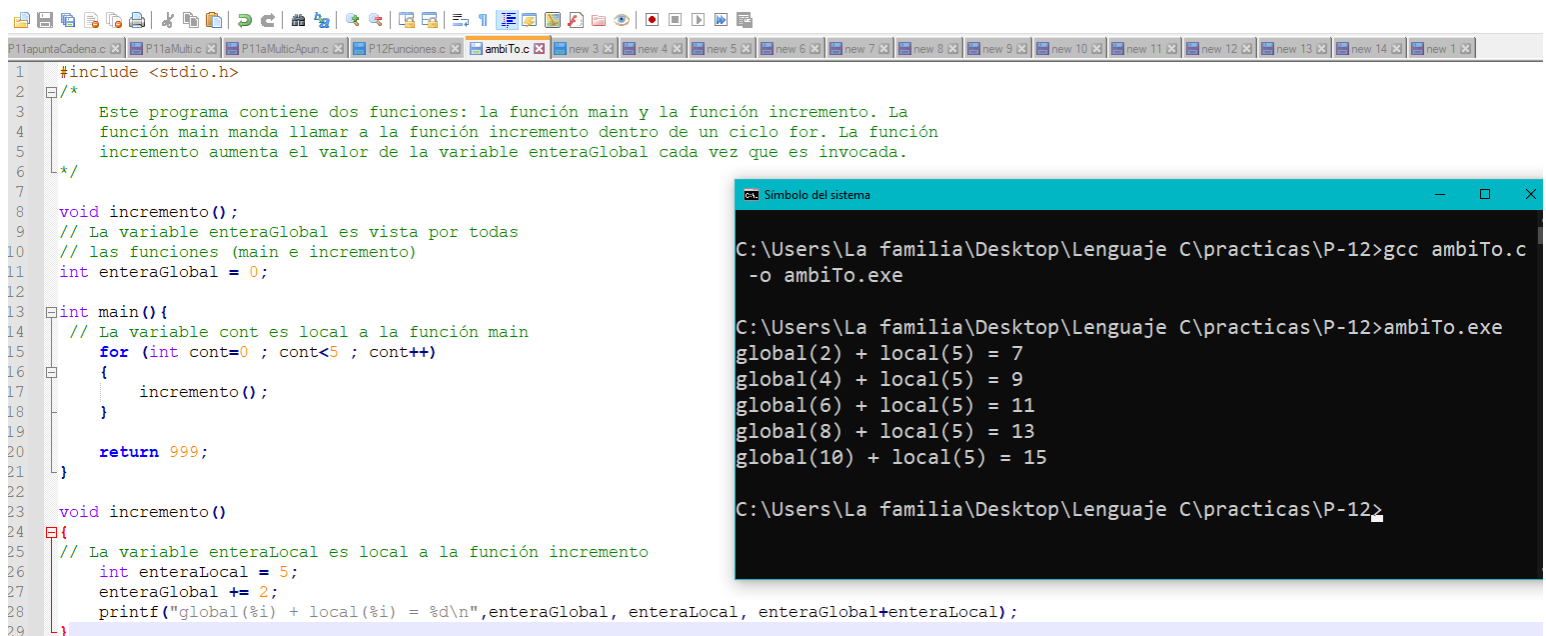


The screenshot shows a C program in Notepad++ and its execution in a Windows command prompt. The program defines a function `imprimir` that prints a string character by character. The `main` function calls `imprimir` with the string "Facultad de Ingeniería".

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

The command prompt shows the execution of `P12Funciones.exe`, which outputs the string "a|reinegnI ed datlucaF" (the string "Facultad de Ingeniería" printed in reverse).

Ámbito de las variables



The screenshot shows a C program in Notepad++ and its execution in a Windows command prompt. The program demonstrates variable scope by using a global variable `enteraGlobal` and a local variable `enteraLocal` within the `incremento` function. The `main` function calls `incremento` five times, and the output shows the cumulative value of `enteraGlobal` and the local value of `enteraLocal` at each step.

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

The command prompt shows the compilation of `ambiTo.c` into `ambiTo.exe` and its execution. The output shows the cumulative value of `enteraGlobal` and the local value of `enteraLocal` at each step:

```
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 función main

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

```
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>gcc argumentosFuncionMain.c -o argumentosFuncionMain.exe
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>argumentosFuncionMain.exe
El programa no contiene argumentos.
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>
```

Variable estática

```
1 #include <stdio.h>
2 /*
3  * Este programa contiene dos funciones: la función main y la función
4  * llamarFuncion. La función main manda llamar a la función llamarFuncion dentro
5  * de un ciclo for. La función llamarFuncion crea una variable estática e imprime
6  * su valor.
7  */
8
9 void llamarFuncion();
10 char aa=160, ae=130, ai=161, ao=162, au=163, sp=168;
11 int main ()
12 {
13     for (int j=0 ; j < 5 ; j++)
14     {
15         llamarFuncion();
16     }
17 }
18
19 void llamarFuncion()
20 {
21     static int numVeces = 0;
22     printf("Esta función se ha llamado %d veces.\n",ao,++numVeces);
23 }
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>gcc VariableEstatica.c -o VariableEstatica.exe
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>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\La familia\Desktop\Lenguaje C\practicas\P-12>
```

Calculadora

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

```
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>gcc FuncionEstatica.c -o FuncionEstatica.exe
c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../../mingw32/bin/ld.exe: c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../../libmingw32.a(main.o):(.text.startup+0xc0): undefined reference to `WinMain@16'
collect2.exe: error: ld returned 1 exit status
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>
```

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

```
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>gcc calculadora.c -o calculadora.exe
c:/mingw/bin/./lib/gcc/mingw32/6.3.0/../../../../mingw32/bin/ld.exe: C:\Users\LAFAMI~1\AppData\Local\Temp\ccXz9x3q.o:calculadora.c:(.text+0x1e): undefined reference to `suma'
c:/mingw/bin/./lib/gcc/mingw32/6.3.0/../../../../mingw32/bin/ld.exe: C:\Users\LAFAMI~1\AppData\Local\Temp\ccXz9x3q.o:calculadora.c:(.text+0x42): undefined reference to `producto'
collect2.exe: error: ld returned 1 exit status

C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>
```

Como vimos al tratar de compilar cada programa por separado nos marca error. 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.

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

```
9 archivos      135,743 bytes
2 dirs  735,904,485,376 bytes libres

C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>gcc FuncionEstatica.c calculadora.c -o exe
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>
```

```
C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>exe.exe
5 + 7 = 12
6 * 8 = 48

C:\Users\La familia\Desktop\Lenguaje C\practicas\P-12>
```


CONCLUSIONES

Como se abordó en la práctica las funciones realizan tareas concretas, la función que hemos utilizado es la función main pero además de esta podemos crear otras funciones independientes al código e irlas llamando dentro de nuestra función principal cuando las necesitemos, debido a que cuando hay problemas grandes es más sencillo dividir la resolución en distintas funciones.

Las funciones contras de 3 partes principales: el nombre, los parámetros y el valor de retorno de dicha función. También se trataron las variables locales y globales, las primeras son las variables que se declaren dentro de cada función, mientras las globales se declaran fuera de cualquier función, existen durante la ejecución de todo el programa y pueden ser utilizadas por cualquier función.

El atributo static en una función hace que esa función sea accesible solo dentro del mismo archivo, restringe su ámbito al archivo donde se definen. Por lo cual dichas funciones son muy utilizadas cuando se desarrollan aplicaciones de gran tamaño, se suelen establecer políticas para el uso de prototipos.

REFERENCIAS

Facultad de Ingeniería. (2018, 6 abril). Manual de prácticas de fundamentos de programación MAD0-17_FP. Laboratorio de Computación Salas A y B. <http://lcp02.fi-b.unam.mx/>

“Programación en C: Metodología, algoritmos y estructura de datos”, Luis Joyanes Aguilar, Ignacio Zahonero Martínez; McGrawHill, ed. 2, 2005