UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA MEXICALI

Formato para Prácticas de Laboratorio

PROGRAMA EDUCATIVO	PLAN DE ESTUDIO	CLAVE DE UNIDAD DE APRENDIZAJE	NOMBRE DE LA UNIDAD DE APRENDIZAJE
LSC	2009-2	11294	Programación Estructurada

PRÁCTICA No.	LABORATORIO DE	Licenciados en Sistemas Computacionales	DURACIÓN (HORAS)
7	NOMBRE DE LA PRÁCTICA	Funciones con parámetros y que retornan valor utilizando Makefile	2

1. INTRODUCCIÓN

En la práctica anterior aplicamos el concepto de funciones en nuestros programas las cuales nos permiten realizar programas estructurados y modularizados de tal forma que evitamos la duplicidad de código realizando tareas que pueden ser llamadas en repetidas ocasiones cuantas veces sea necesario. En esta ocasión utilizaremos aquellas funciones que nos permiten recibir parámetros y retornan valor al programa que la llama con lo cual mejoraremos aún más nuestros programas ya que existen algunas situaciones problemáticas que requieren el uso de estas, para obtener una mayor eficiencia en los mismos.

2. OBJETIVO (COMPETENCIA)

Elaborar programas de cómputo aplicando las funciones que reciben parámetros y que regresan valor para optimizar su funcionamiento con una actitud analítica y responsable.

3. FUNDAMENTO

Funciones que reciben parámetros y retornan valor al programa que las llama.

Recordemos que una función es un módulo de un programa separado del cuerpo principal, que realiza una tarea específica y que puede regresar un valor a la parte principal del programa u otra función o procedimiento que la invoque.

Formuló	Revisó	Autorizó
LSC. Natalia Rodríguez Castellón	I.C. Josefina Mariscal Camacho	Dr. David Isaías Rosas Almeida
Nombre y Firma del Maestro	Nombre y Firma del Responsable de Programa Educativo	Nombre y Firma del Director / Representante de la Dirección

La forma general de una función es:

```
Tipodato Nomfun( tipoparam parametro)
{
cuerpo de instrucciones;
return [dato,var,expresion];
}
```

Donde :

tipodato : Especifica el tipo de dato que regresara la función.

Nomfun : Es el nombre que se le asigna a la función y debe ser relacionado con la tarea especifica que ejecuta.

tipoparam : Es el tipo del valor que la función recibirá para ser utilizado dentro de esta.
parámetro : Variable local de la función donde se recibe al valor que le fue enviado desde el programa principal u otra función.

Nota : Por cada parámetro que se reciba se debe indicar el tipo y el nombre de la variable donde se recibirá el valor, así mismo se separaran por comas.

La instrucción **return** es quien regresa un y solo un valor a la parte del programa que la este llamando o invocando, sin embargo return puede regresar un dato, una variable o una expresión algebraica(no ecuación o formula) como se muestra en los siguientes ejemplos:

a) return (3.1416);
b) return (area);
c) return (x + y);

La lista de parámetros formales es una lista de variables separadas por comas (,) que almacenaran los valores que reciba la función, estas variables actúan como locales dentro del cuerpo de la función.

Aunque no se ocupen parámetros los paréntesis son requeridos.

Llamada de función :
Cuando se llame una función de este tipo deberá haber una variable que reciba el valor que regresara la función, es decir generalmente se llama una función mediante una sentencia de asignación, por ejemplo

a) resultado=funcion(5, 3.1416);
b) suma=funcion(n1,n2);

El uso de funciones definidas por el programador permite dividir un programa grande en un cierto número de componentes más pequeños, cada uno de las cuales con un propósito único e identificable.

Uso de archivos make

La herramienta ***make*** se utiliza para compilar de manera condicional. Esto es, compila archivos fuente cuando éstos son mas nuevos que el archivo destino.
Es de especial importancia esta herramienta cuando se desarrollan aplicaciones grandes que constan de múltiples archivos con código fuente.
Al compilar, *make* es útil porque no solo nos ahorra el tener que escribir una larga línea indicando los archivos que deben compilarse, sino que también es útil debido a que solo compila los archivos que han cambiado desde la última compilación. Esto hace que el tiempo para compilar sea menor que si se compilaran todos los archivos cada vez que se hace un cambio pequeño en alguno de ellos.
Para utilizar **make** se debe tener un archivo llamado **Makefile** que le indique a **make** lo que debe hacer por medio de una serie de reglas. Una regla tiene la siguiente forma:

destino: requisitos . . .
instrucción
. . .
. . .

Donde :

destino normalmente es el nombre del archivo que se generará por un programa; por ejemplo archivos ejecutables o archivos objeto. Un destino también puede ser el nombre de una acción a realizar por ejemplo *clean*.

Un ***requisito*** es un archivo que se utiliza como entrada para crear un *destino*. Un *destino* muchas veces depende de varios archivos.

Una ***instrucción*** es un archivo que se utiliza como entrada para crear un *destino*. Un *destino* muchas veces depende de varios archivos.
Una *instrucción* es la acción que *make* realiza. Una regla puede tener más de una instrucción, poniendo cada instrucción en su propia línea. Se debe poner el carácter de *tabulador* al inicio de cada línea de instrucción.

El siguiente listado, muestra un ***Makefile*** sencillo que describe la manera en que se crea el archivo ejecutable ***ejecuta.***Se puede ver que el ejecutable depende de 8 archivos de código objeto que a su vez dependen de 8 archivos de código fuente de C y de 3 archivos de encabezado

```
ejecuta : main.o kbd.o command.o display.o insert.o search.o file.o utils.o
        cc -o ejecuta main.o kdb.o command.o display.o insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
```

```
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean:
        rm ejecuta *.o
```

Para generar el archivo objeto *utils.o* se verifican los archivos *utils.c* y *utils.h* de los cuales depende. La instrucción para compilar contiene el parámetro *-c* que le indica al compilador que solo debe generar el código objeto y no llegar al ejecutable. (La extensión default de los archivos código objeto es *.o* bajo Linux.
La regla *clean* provocará que se borren todos los archivos objeto así como el ejecutable. A su vez, esto provoca que se recompilen todos los archivos con código fuente.
Para utilizar este makefile para crear el ejecutable *ejecuta*, solo se debe escribir **make**. Para utilizarlo para borrar el ejecutable y los objeto, se debe escribir **make clean**.
La herramienta *make* cuenta con muchas otras funciones que permiten que la creación de ejecutables sea sencilla y rápida.

Ejemplo:
El siguiente es un ejemplo de un programa que realiza la captura de números enteros en un vector mientras el usuario lo desee.
A continuación tenemos el ***Makefile*** del programa de ordenación del vector que crea el archivo ejecutable llamado **vector**. Este archivo depende de los archivos de código objeto que a su vez dependen de los de código fuente de C. El archivo **menu.c** llama a tres funciones que se hicieron en programas independientes a este: la función de **captura.c** que permite capturar los datos para el vector de números enteros, la función de **ordena.c**, en la que se realiza la ordenación de los datos del vector y la función de **muestra.c** que muestra en pantalla los datos del vector.

Nota: Recuerde que los programas que son de código fuente de C se tienen que hacer también los archivos de encabezado.

Contenido del Archivo menu.c

```
#include "captura.h"
#include "ordena.h"
#include "muestra.h"
#include<stdio.h>
int main()
{
int vector[100],op,cont=0;
do
  { printf("Ordenacion de Vector\n");
    printf("\n Captura del vector.....[1]\n");
    printf("\n Ordenacion del Vector.....[2]\n");
    printf("\n Mostrar Vector Resultante...[3]\n");
    printf("\n Salir del Sistema.....[4]\n");
```

```
     printf("\n Opcion a elegir.............[ ]\b\b");
     scanf("%d",&op);
     switch(op)
     {
      case 1 : cont=captura(vector,cont);
               break;
      case 2 : ordena(vector,cont);
               printf("Datos del vector ordenados\n\n");
               break;
      case 3 : muestra(vector,cont);
               break;
      case 4 : printf("Press any key to exit <<>>");
               break;
      default : printf("Invalid Option, Press any key to continue<<>>");
     }printf("\n");
  }while(op!=4);
return 0;
}
```

Contenido del Archivo captura.c

```
#include<stdio.h>
int captura(int vector[100], int c)
{ int opc;
   do
    {
     printf("Da el valor para la posicion [%d]: [  ]\b\b\b",c);
     scanf("%d",&vector[c]);
     c++;
     printf("Desea registrar otro numero s/n :");
     scanf("%c",&opc);
    }while(opc!='n');
return(c);
}
```

Contenido del Archivo captura.h

```
#ifndef CAPTURA_H
#define CAPTURA_H
int captura(int [100],int);
#endif
```

Contenido del Archivo ordena.c

```
void ordena(int vector[100], int c)
{
  int x,y,temp;

  for(x=0;x<c;x++)
   for(y=x+1;y<c;y++)
    {
     if(vector[x]>vector[y])
      { temp=vector[x];
        vector[x]=vector[y];
```

```
          vector[y]=temp;
        }
      }
  }
```

Contenido del Archivo ordena.h

```
#ifndef ORDENA_H
#define ORDENA_H
void ordena(int [100],int);
#endif
```

Contenido del Archivo muestra.c

```
#include<stdio.h>
void muestra(int vector[100],int c)
{ int x;

  for(x=0;x<c;x++)
   printf("%d\t",vector[x]);
}
```

Contenido del Archivo muestra.h

```
#ifndef MUESTRA_H
#define MUESTRA_H
void muestra(int [100],int);
#endif
```

Contenido del Archivo Makefile

```
vector : menu.o captura.o ordena.o muestra.o
      cc -o vector menu.o captura.o ordena.o muestra.o
menu.o : menu.c
      cc -c -g menu.c
captura.o : captura.c captura.h
      cc -c -g captura.c
ordena.o : ordena.c ordena.h
      cc -c -g ordena.c
muestra.o : muestra.c muestra.h
      cc -c -g muestra.c
clean:
      rm *.o *~ vector
```

4.

A) EQUIPO NECESARIO	**MATERIAL DE APOYO**
Computadoras con Linux instalado	Práctica impresa y apuntes de clase

B) DESARROLLO DE LA PRÁCTICA

El alumno realizara el programa correspondiente al día de la clase.

Martes

La Sra. Dora, vende joyería casa por casa, excepto los lunes ya que ese día es su día de descanso, y le interesa tener un programa que le permita llevar el control de ventas de la misma por día, por lo que se necesita que tenga las siguientes opciones:

1) Registrar Ventas Joyería Fina (Deberá solicitar el monto de ventas realizadas diariamente por semana, de martes a domingo)
2) Registrar Ventas Bisutería(Deberá solicitar el monto de ventas realizadas diariamente por semana, de martes a domingo)
3) Joyería más Vendida (Deberá mostrar cuál de los 2 tipos de joyería es mayormente aceptada por las clientas.
4) Venta Total(Deberá mostrar el monto total de ventas en la semana, Nota: este dato deberá regresarlo y mostrarlo en el programa principal)
5) Salir

Jueves

La librería "El que lee más sabe más" desea un sistema que le permita controlar la cantidad de libros vendidos durante una semana, para lo cual requiere que se muestren las siguientes opciones:
a)Capturar Ventas Diarias (Deberá solicitar las cantidades de libros vendidos diariamente durante una semana la cual estará compuesta de lunes a sábado)
*b)Mostrar Ventas(*Mostrar las cantidades de ventas diarias así como el nombre del día)
c)Consultar Préstamo(Solicitar el nombre del día, mostrar la cantidad vendida de dicho día)
d)Venta Mayor(Mostrar la cantidad mayor de ventas realizados en un día)Nota: este dato deberá regresarlo y mostrarlo en el programa principal.
e)Salir

Viernes

La cafetería "Coffee Club" requiere de un sistema que le permita saber si es factible la venta de café por lo que requiere el programa con las siguientes opciones.

1) Registrar Café Tradicional (Registrar la cantidad de cafés tradicionales vendidos diariamente durante una semana)
2) Registrar Frapuchinos (Registrar la cantidad De cafés frapuchinos vendidos diariamente durante una semana)
3) Mejor Aceptado (Mostrar cuál de los 2 cafés es mayormente aceptado por la clientela)
4) Día mayor venta(Mostrar que día de la semana se realizó más venta incluyendo ambos cafés)
5) Venta Total (Deberá mostrar el monto total de ventas en la semana, Nota: este dato deberá regresarlo y mostrarlo en el programa principal
6) Salir

C)

Se revisarán los programas haciendo pruebas 1 o más veces.

5. RESULTADOS Y CONCLUSIONES

El alumno será capaz de elaborar programas haciendo uso de funciones con parámetros y que retornan valor, por medio de Makefile.

6. ANEXOS

Las prácticas se realizaran dependiendo del día que corresponda la clase.

7. REFERENCIAS