	**Carátula para entrega de prácticas**
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Karina García Morales

Asignatura: Fundamentos de programación

Grupo:

No. de Práctica: 9

Integrantes: Ever Ivan Rosales Gómez

No de Lista o brigade:

Semestre: 2023-2

Fecha de entrega: 16 de mayo de 2023

Observaciones:

Calificación

ARREGLOS UNIDIMENSIONALES

OBJETIVO

Elaborar programas en lenguaje C con ayuda de arreglos alineados en un vector o lista.

Para entender bien este apartado es importante mencionar. ¿Qué es un arreglo?. Es una serie de caracteres del mismo tipo y estos pueden ser enteros, caracteres, reales y booleanos.
Para identificar qué se habla de un arreglo este se puede observar porque está declarado con ayuda de corchetes [ ].
La manera de declarar un arreglo es de la siguiente manera

Tipo de dato	**Nombre del arreglo (puede ser el que el programador quiera)**	**Corchetes**	**;**
int float char	nombre	[ ]	;

Se pueden declarar de dos formas los arreglos:

1. Conociendo el tamaño del arreglo. Es importante mencionar que estos comienzan en 0 y terminan en n-1. Es decir, si se desea un arreglo de 6 elementos se debe poner 7 como tamaño.

Tipo de dato	**Nombre del arreglo**	**Corchetes**	**;**
int	numeros	[7]	;

En pseudocódigo quedaría int numeros[7];

2. La otra forma de declarar el arreglo es conociendo específicamente los valores que tendrá

Tipo de dato	Nombre del arreglo	Corchetes	=	Valores	;
int	numeros	[ ]	=	{1,2,3,4,5,6}	;

En pseudocódigo quedaría int numeros[ ]={1,2,3,4,5,6};

PROGRAMA 1

```
Bulgaria20:~ fp24alu35$ vi prog1.c
Bulgaria20:~ fp24alu35$ gcc prog1.c -o prog1.out
Bulgaria20:~ fp24alu35$ ./prog1.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Bulgaria20:~ fp24alu35$
```

```
#include <stdio.h>
int main ()
{
int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
int indice = 0; //AQUI SE ENCUENTRA UN CONTADOR
printf("\tLista\n");
while (indice < 5 ) // Acceso a cada elemento del arreglo unidimensional usando while
{
printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]); indice += 1; // Sentencia análoga a indice = indice + 1;
}
printf("\n");
return 0; }
~
~
```

Este programa se lleva a cabo con ayuda de arreglo. El arreglo es de tipo entero llamado lista que contiene 5 elementos.
El propósito del programa es arrojar las calificaciones de 5 alumnos con calificaciones ya definidas en el arreglo.
Con ayuda de un contador llamado índice ayuda a que el arreglo vaya cambiando de posición para los 5 alumnos.

PROGRAMA 2

```
#include <stdio.h>
int main () {
int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
int indice = 0;
printf("\tLista\n");
do // Acceso a cada elemento del arreglo unidimensional usando do-while
{
printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
indice += 1;
}
while (indice < 5 ); printf("\n");
return 0;
}
// Sentencia análoga a indice = indice + 1;
~
```

```
Bulgaria20:~ fp24alu35$ gcc prog2.c -o prog2.out
Bulgaria20:~ fp24alu35$ ./prog2.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Bulgaria20:~ fp24alu35$
```

Este es ejercicio tiene el mismo propósito que el pasado, pero ahora se maneja con una estructura DO WHILE, es decir que primero va hacer el bloque de instrucciones y posteriormente va a ver la condición.

PROGRAMA 1C

```
#include <stdio.h>
int main () {
int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
int indice=0;
printf("\tLista\n");
// Acceso a cada elemento del arreglo unidimensional usando for
for (indice = 0 ; indice < 5 ; indice++) {
printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]); }
printf("\n");
return 0; }
```

```
Bulgaria20:~ fp24alu35$ gcc prog3.c -o prog3.out
Bulgaria20:~ fp24alu35$ ./prog3.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Bulgaria20:~ fp24alu35$
```

Al igual que el anterior sigue siendo el mismo programa, pero ahora con ayuda de un ciclo FOR en donde la variable que irá dentro del ciclo for será el contador que en este caso se llama índice.

PROGRAMA 2

```c
#include <stdio.h>
int main () {
int lista[10]; // Se declara el arreglo unidimensional
int indice=0;
int numeroElementos=0;
printf("\nDa un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo\n");
scanf("%d",&numeroElementos);
if((numeroElementos>=1) && (numeroElementos<=10)) {
        // Se almacena un número en cada elemento del arreglo unidimensional usando for
for (indice = 0 ; indice <= numeroElementos-1 ; indice++) {
            printf("\nDar un número entero para el elemento %d del arreglo", indice );
            scanf("%d",&lista[indice]);
}
printf("\nLos valores dados son: \n");
// Se muestra el número almacenado en cada elemento del arreglo unidimensional usando for
for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
          {
            printf("%d  ", lista[indice] );
} }
else printf("el valor dado no es válido"); printf("\n");
return 0;
}
```

```
Da un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo
4

Dar un número entero para el elemento 0 del arreglo1

Dar un número entero para el elemento 1 del arreglo2

Dar un número entero para el elemento 2 del arreglo3

Dar un número entero para el elemento 3 del arreglo4

Los valores dados son:
1  2  3  4
```

```
Bulgaria20:~ fp24alu35$ gcc prog1.c -o prog1.out
Bulgaria20:~ fp24alu35$ ./prog1.out

Da un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo
3

Dar un número entero para el elemento 0 del arreglo: 1

Dar un número entero para el elemento 1 del arreglo: 2

Dar un número entero para el elemento 2 del arreglo: 3

Los valores dados son:
1  2  3
```

```c
#include <stdio.h>
int main () {
int lista[10]; // Se declara el arreglo unidimensional
int indice=0;
int numeroElementos=0;
printf("\nDa un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo\n");
scanf("%d",&numeroElementos);
if((numeroElementos>=1) && (numeroElementos<=10)) {
        // Se almacena un número en cada elemento del arreglo unidimensional usando for
//for (indice = 0 ; indice <= numeroElementos-1 ; indice++) {
       do{
        printf("\nDar un número entero para el elemento %d del arreglo: ", indice );
        scanf("%d",&lista[indice]);
        indice++;
}while(indice<=numeroElementos-1);
//}
printf("\nLos valores dados son: \n");
// Se muestra el número almacenado en cada elemento del arreglo unidimensional usando for
for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
          {
            printf("%d  ", lista[indice] );
} }
else printf("el valor dado no es válido"); printf("\n");
return 0;
}
```

En este ejercicio se modificó el ciclo **for** por un ciclo DO WHILE que como ya se sabe primero ejecuta el bloque de instrucciones y después

revisa la condición que tiene el while. Es importante mencionar que cuando se usa DO WHILE, el while finaliza con **;.**

APUNTADORES
¿Qué es un apuntador?
Es una variable que contiene la dirección de memoria de una variables. Es decir que encuentra el lugar en que se guarda una variable que podemos encontrar.
Para declarar una variable se lleva a cabo lo siguiente:

TipoDeDatos *apuntador,variable; El * señala que se habla de un apuntador

```
#include <stdio.h>
int main ()
{
char *ap, c = 'a'; // Se declara el apuntador ap de tipo alfanumérico
ap = &c; //Se le asigna al apuntador la dirección de memoria de la variable c
printf("Carácter: %c\n",*ap); /* Se imprime el contenido de la variable a la
que apunta el apuntador ap */
printf("Código ASCII: %d\n",*ap); /*Se imprime el código ASCII del carácter'a' */
printf("Dirección de memoria: %s\n",ap);/*Se imprime la dirección de memoria que almacena el apuntador*/
return 0;
}
~
~
```

```
[Bulgaria20:~ fp24alu35$ gcc prog3.c -o prog3.out
[Bulgaria20:~ fp24alu35$ ./prog3.out
Carácter: a
Código ASCII: 97
Dirección de memoria: ao?4??
Bulgaria20:~ fp24alu35$
```

En este programa se cuenta con un apuntador (*ap) de tipo alfanumérico y su propósito es ubicar la dirección de memoria de la variable c.

```
#include<stdio.h>
int main () {
int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
int *apEnt;
apEnt = &a;
printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n"); printf("apEnt = &a\n");
    /*A la variable b se le asigna el contenido de la variable a la que
apunta apEnt*/
b = *apEnt;
printf("b = *apEnt \t-> b = %i\n", b);
/*A la variable b se le asigna el contenido de la variable a la que
      apunta apEnt y se le suma uno*/
b = *apEnt +1;
printf("b = *apEnt + 1 \t-> b = %i\n", b);
//La variable a la que apunta apEnt se le asigna el valor cero
*apEnt = 0;
printf("*apEnt = 0 \t-> a = %i\n", a);
/*A apEnt se le asigna la dirección de memoria que tiene el elemento 0
      del arreglo c*/
apEnt = &c[0];
printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt); return 0;
}
~
```

```
Bulgaria20:~ fp24alu35$ vi prog4.c
Bulgaria20:~ fp24alu35$ gcc prog4.c -o prog4.out
Bulgaria20:~ fp24alu35$ ./prog4.c
-bash: ./prog4.c: Permission denied
Bulgaria20:~ fp24alu35$ ./prog4.out
a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}
apEnt = &a
b = *apEnt      -> b = 5
b = *apEnt + 1  -> b = 6
*apEnt = 0      -> a = 0
apEnt = &c[0]   -> apEnt = 5
Bulgaria20:~ fp24alu35$
```

A continuación en el en el apENT se cambiará el valor 0 por otro número del arreglo de a

```
Bulgaria20:~ fp24alu35$ gcc prog4.c -o prog4.out
Bulgaria20:~ fp24alu35$ ./prog4.out
a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}
apEnt = &a
b = *apEnt      -> b = 5
b = *apEnt + 1  -> b = 6
*apEnt = 0      -> a = 0
apEnt = &c[5]   -> apEnt = 9
Bulgaria20:~ fp24alu35$
```

Como se puede observar en la imagen el valor es diferente de 5.

PROGRAMA 6

```c
#include <stdio.h>
int main () {
int arr[] = {50, 40, 30, 20, 10};
int *apArr;//Se declara el apuntador apArr
int x;
apArr = arr;
printf("int arr[] = {5, 4, 3, 2, 1};\n");
printf("apArr = &arr[0]\n");
x = *apArr; /*A la variable x se le asigna el contenido del arreglo arr en
su elemento 0*/
printf("x = *apArr \t -> x = %d\n", x);
x = *(apArr+1); /*A la variable x se le asigna el contenido del arreglo arr
                        en su elemento 1*/
printf("x = *(apArr+1) \t -> x = %d\n", x);
x = *(apArr+2); /*A la variable x se le asigna el contenido del arreglo arr
                        en su elemento 2*/
printf("x = *(apArr+2) \t -> x = %d\n", x);
x = *(apArr+3); /*A la variable x se le asigna el contenido del arreglo arr
                        en su elemento 3*/
printf("x = *(apArr+3) \t -> x = %d\n", x);
x = *(apArr+4); /*A la variable x se le asigna el contenido del arreglo arr
                        en su elemento 4*/
printf("x = *(apArr+2) \t -> x = %d\n", x); return 0;
}
```

```
Bulgaria20:~ fp24alu35$ gcc prog5.c -o prog5.out
Bulgaria20:~ fp24alu35$ ./prog5.out
int arr[] = {5, 4, 3, 2, 1};
apArr = &arr[0]
x = *apArr        -> x = 50
x = *(apArr+1)    -> x = 40
x = *(apArr+2)    -> x = 30
x = *(apArr+3)    -> x = 20
x = *(apArr+2)    -> x = 10
Bulgaria20:~ fp24alu35$
```

Al cambiar los valores del arreglo los resultados cambian como se puede mostrar en pantalla con ayuda del printf

PROGRAMA 7

```
Bulgaria20:~ fp24alu35$ gcc prog6.c -o prog6.out
Bulgaria20:~ fp24alu35$ ./prog6.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Bulgaria20:~ fp24alu35$
```

```c
#include <stdio.h>
int main () {
int lista[5] = {10, 8, 5, 8, 7};
int *ap = lista; //Se declara el apuntador ap
int indice;
printf("\tLista\n");
//Se accede a cada elemento del arreglo haciendo uso del ciclo for
for (indice = 0 ; indice < 5 ; indice++)
{
printf("\nCalificación del alumno %d es %d", indice+1,
*(ap+indice)); }
printf("\n");
return 0; }
~
```

Con ayuda del apuntador en este caso ap buscará la posición del arreglo, iniciando desde 0 y aumentando 1. En el primer caso, lugar 0 el

valor es 10 para el alumno, el programa se vuelve a ejecutar ahora el valor 1 será 8 para el alumno 2, así sucesivamente.

```
#include <stdio.h>
int main () {
int lista[5] = {10, 8, 5, 8, 7};
int *ap = lista; //Se declara el apuntador ap
int indice = 0;
printf("\tLista\n");
//Se accede a cada elemento del arreglo haciendo uso del ciclo while
while (indice < 5) {
printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
indice++; }
printf("\n");
return 0; }
~
```

Este ejercicio tenía dos errores ya que el texto entre // estaba agregado el int que está debajo, además es importante recordar que cuando se usa únicamente while va con llaves en caso de ser largo el bloque de código o puede ir sin llaves en caso de ser corto, en el programa venía agregado ;

```
#include <stdio.h>
int main () {
int lista[5] = {10, 8, 5, 8, 7};
int *ap = lista; //Se declara el apuntador ap
int indice = 0;
printf("\tLista\n");
//Se accede a cada elemento del arreglo haciendo uso del ciclo do-while
do{
printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
indice++;
}while (indice < 5); //el error estaba aquí ya que no tenía ;
printf("\n");
return 0;
}
~
~
```

Al igual que el ejercicio anterior estaba mal la sintaxis ya que en este caso si estamos usando un DO WHILE, el programa no tenía ; y justo ese era el error.

PROGRAMA 7

```c
#include <stdio.h>
int main() {
char palabra[20];
int i=0;
printf("Ingrese una palabra: ");
scanf("%s", palabra); /* Se omite & porque el propio nombre del arreglo de tipo cadena apunta, es decir, es equivalente a la dirección de comienzo del propio arreglo*/
printf("La palabra ingresada es: %s\n", palabra);
for (i = 0 ; i < 20 ; i++)
{
printf("%c\n", palabra[i]);
} return 0;
}
```

```
Bulgaria20:~ fp24alu35$ gcc prog7.c -o prog7.out
Bulgaria20:~ fp24alu35$ ./prog7.out
Ingrese una palabra: everisto
La palabra ingresada es: everisto
e
v
e
r
i
s
t
o
```

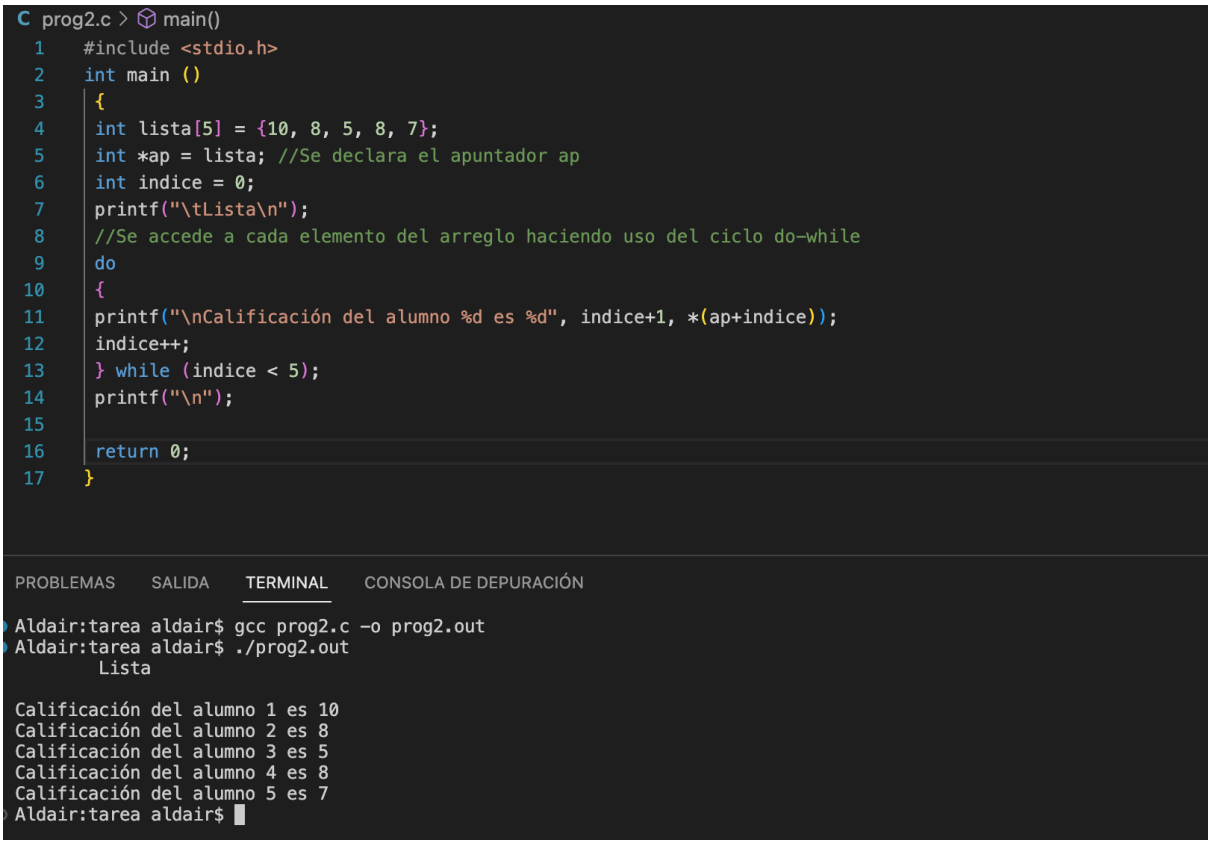
EJERCICIOS DE TAREA

1. **Ejecutar los últimos 3 ejercicios de clase.**
   Para ejecutar los 3 programas utilicé el mismo nombre para ambos.

```c
C prog2.c > main()
#include <stdio.h>
int main ()
{
int lista[5] = {10, 8, 5, 8, 7};
int *ap = lista; //Se declara el apuntador ap
int indice = 0;

printf("\tLista\n");
//Se accede a cada elemento del arreglo haciendo uso del ciclo while
while (indice < 5) // es importante mencionar que el while no lleva ;
{
printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
indice++;
}
printf("\n");
return 0;
}
```

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

Aldair:tarea aldair$ gcc prog2.c -o prog2.out
Aldair:tarea aldair$ ./prog2.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Aldair:tarea aldair$
```

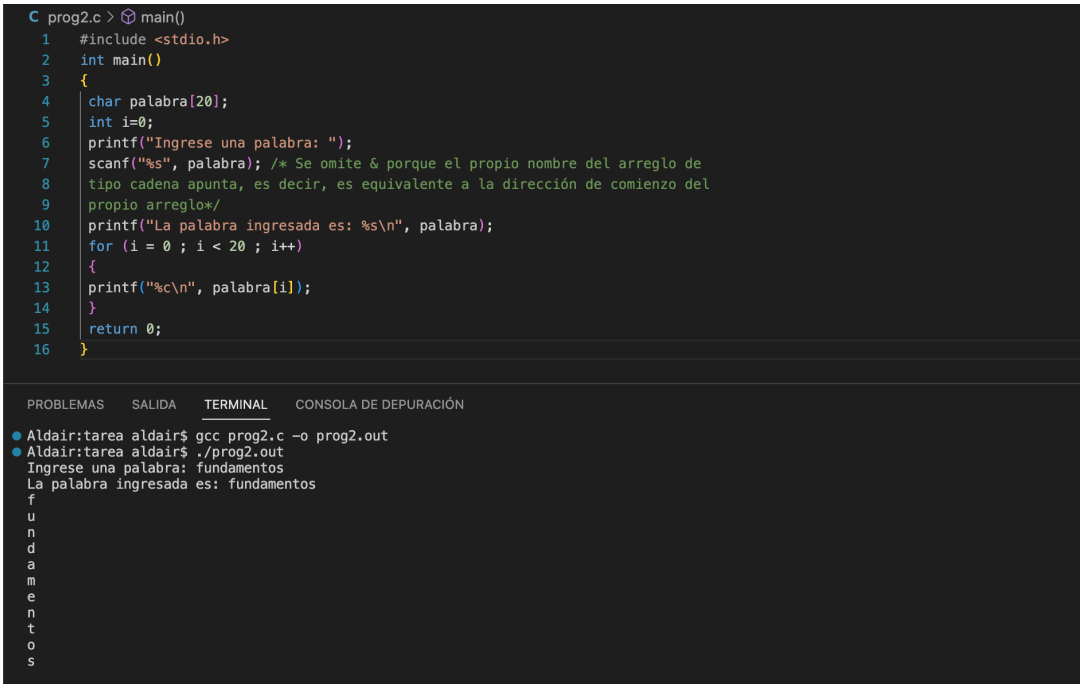
El programa tenía punto y coma en el while. Es importante mencionar que únicamente se pone punto y coma cuando se está usando un ciclo do-while.



```c
#include <stdio.h>
int main ()
{
int lista[5] = {10, 8, 5, 8, 7};
int *ap = lista; //Se declara el apuntador ap
int indice = 0;
printf("\tLista\n");
//Se accede a cada elemento del arreglo haciendo uso del ciclo do-while
do
{
printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
indice++;
} while (indice < 5);
printf("\n");

return 0;
}
```

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

Aldair:tarea aldair$ gcc prog2.c -o prog2.out
Aldair:tarea aldair$ ./prog2.out
        Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
Aldair:tarea aldair$
```

Como se mencionó en el ejercicio anterior, cuando se usa una estructura do while el while lleva punto y coma. el ejercicio no lleva eso por lo que marcaba error.



```c
#include <stdio.h>
int main()
{
char palabra[20];
int i=0;
printf("Ingrese una palabra: ");
scanf("%s", palabra); /* Se omite & porque el propio nombre del arreglo de
tipo cadena apunta, es decir, es equivalente a la dirección de comienzo del
propio arreglo*/
printf("La palabra ingresada es: %s\n", palabra);
for (i = 0 ; i < 20 ; i++)
{
printf("%c\n", palabra[i]);
}
return 0;
}
```

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

Aldair:tarea aldair$ gcc prog2.c -o prog2.out
Aldair:tarea aldair$ ./prog2.out
Ingrese una palabra: fundamentos
La palabra ingresada es: fundamentos
f
u
n
d
a
m
e
n
t
o
s
```

2. **Indica que realiza en el siguiente programa**

Antes de comenzar es importante mencionar que al programa le hace falta una condición en los ciclos, hasta dónde llegará la variable i.
El programa va a imprimir el número al que corresponde cada elemento del arreglo como se muestra en pantalla.

```c
C prog1.c > main()
#include <stdio.h>

int arreglo[] = {20,19,12,17,16,36,14,13,23,31};

int i, j, n, aux;

int main() {
n = 10;
for(i=1; i<10 ;i++) {
                j = i;
                aux = arreglo[i];
               while(j>0 && aux<arreglo[j-1]){

                                arreglo[j] = arreglo[j-1];

                                j=j-1;
                }
                arreglo[j] = aux;

}

printf("\n\nLos elementos obtenidos del arreglo son: \n");

for(i=0; i<10 ;i++) {
                printf("Elemento [%d]: %d\n", i, arreglo[i]);
}
return 0;
}
```

```
Los elementos obtenidos del arreglo son:
Elemento [0]: 12
Elemento [1]: 13
Elemento [2]: 14
Elemento [3]: 16
Elemento [4]: 17
Elemento [5]: 19
Elemento [6]: 20
Elemento [7]: 23
Elemento [8]: 31
Elemento [9]: 36
```

CONCLUSIÓN

Con ayuda de arreglos es más fácil llevar a cabo un programa que requiere la presencia de varios conjunto de datos y como bien lo dice la definición de arreglos deben ser del mismo tipo.
Es importante tener una idea de cómo llevar a cabo la estructura del lenguaje, a medida que se vaya formando el programa se dará cuenta que se pueden ahorrar una serie de pasos y que se pueden sustituir algunas estructuras de repetición por otras en caso de que el programa lo necesite. Por último, como bien lo comenta la profesora es importante

llevar un orden y comentar algunas cosas en caso de que el programa lo tenga que usar otra persona, pensar que el programa puede ayudar a muchas personas.

REFERENCIAS

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