



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: 20

No. de práctica(s): No. 9 - Arreglos unidimensionales.

Integrante(s): López Olmedo Ernesto Yael

No. de lista o brigada: 27

Semestre: 2024-1

Fecha de entrega: 24 de Octubre 2023

Observaciones:

CALIFICACIÓN: _____

Introducción

Como desarrollo para el conocimiento al lenguaje de programación C y como continuación al desarrollo de las estructuras condicionales, en conjunto a la creación y uso de arreglos bidimensionales de una celda, ahora se presentarán los arreglos unidimensionales, con los cuales el alumno descubrirá como hacer la agrupación de datos, de un mismo tipo de variables (int, float, char), guardados en una sola variable y como hacer múltiples usos de estructuras condicionales, empezando por las simples, hasta las de control iterativo y como hacer el uso tanto por el sentido del contenido del arreglo, como la manipulación de cada uno de los elementos de este.

Objetivo

El alumno utilizará arreglos de una dimensión en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, alineados en un vector o lista.

Desarrollo

Arreglos unidimensionales

Laboratorio

Como actividades son las siguientes

Programa 1-a.c



```
1 #include <stdio.h>
2 int main ()
3 {
4     int lista[5] = {10, 8, 5, 8, 7}; // Se declara e inicializa el arreglo unidimensional
5     int indice = 0;
6     printf("\t\tLista\n");
7     while (indice < 5) // Acceso a cada elemento del arreglo unidimensional usando while
8     {
9         printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
10        indice += 1; // Sentencia análoga a indice = indice + 1;
11    }
12    printf("\n");
13    return 0;
14
15 }
```

Angola05:~ fp20alu27\$ Número de lista: 27.

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

Figura 1.1 - Programa 1.c

Modificación programa 1-a.c

Se hará que imprima el nombre del alumno usando el bucle while.

```
1 #include <stdio.h>
2 int main ()
3 {
4     char lista[4] = {'Y', 'a', 'e', 'l'}; // Se declara e inicializa el arreglo unidimensional
5     int indice = 0,s=0;
6     //printf("\tLista\n");
7     while (indice < 4 ) // Acceso a cada elemento del arreglo unidimensional usando while
8     {
9         //printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
10        printf(" %s ",lista);
11        indice += 1; // Sentencia análoga a indice = indice + 1;
12    }
13    printf("\n");
14    return 0;
15 }
```

Yael Yael Yael Yael

...Program finished with exit code 0
Press ENTER to exit console.

Figura 1.2 - Programa 1-a.c modificación 1

```
1 #include <stdio.h>
2 int main ()
3 {
4     char lista[4] = {'Y', 'a', 'e', 'l'}; // Se declara e inicializa el arreglo unidimensional
5     int indice = 0,s=0;
6     //printf("\tLista\n");
7     while (indice < 4 ) // Acceso a cada elemento del arreglo unidimensional usando while
8     {
9         //printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
10        printf(" %c \n",lista[indice]);
11        indice += 1; // Sentencia análoga a indice = indice + 1;
12    }
13    printf("\n");
14    return 0;
15 }
```

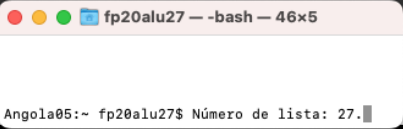
Y
a
e
l

Figura 1.3 - Programa 1-a.c modificación 2

Programa 1-b.c

Modificación:

```
1 #include <stdio.h>
2 int main ()
3 {
4     char lista[4] = {'Y', 'a', 'e', 'l'}; // Se declara e inicializa el arreglo unidimensional
5     int indice = 0,s=0;
6     //printf("\tLista\n");
7     do // Acceso a cada elemento del arreglo unidimensional usando do-while
8     {
9         printf("\n %c",lista[indice]);
10        indice += 1;
11    }while (indice < 5 );
12    printf("\n");
13    return 0;
14 }
```



Angola05:~ fp20alu27\$ Número de lista: 27.

Y
a
e
l

Figura 1.4 - Programa 1-b.c modificación 1

Programa 1-C.c

```
1 #include <stdio.h>
2 int main ()
3 {
4     char lista[4] = {'Y', 'a', 'e', 'l'}; // Se declara e inicializa el arreglo unidimensional
5     int indice = 0,s=0;
6     //printf("\tLista\n");
7     for (indice = 0 ; indice < 5 ; indice++)
8     {
9         printf("\n %c",lista[indice]);
10    }
11    printf("\n");
12    return 0;
13 }
14 }
```



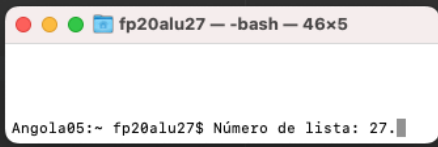
Angola05:~ fp20alu27\$ Número de lista: 27.

Y
a
e
l

Figura 1.5 - Programa 1-C.c modificación 1

Programa 2.c

```
1 #include <stdio.h>
2 int main () {
3     int lista[10]; // Se declara el arreglo unidimensional
4     int indice=0;
5     int numeroElementos=0;
6     printf("\nDa un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo\n");
7     scanf("%d",&numeroElementos);
8     if((numeroElementos>=1) && (numeroElementos<=10)) {
9         // Se almacena un número en cada elemento del arreglo unidimensional usando for
10        for (indice = 0 ; indice <= numeroElementos-1 ; indice++) {
11            printf("\nDar un número entero para el elemento %d del arreglo", indice );
12            scanf("%d",&lista[indice]);
13        }
14        printf("\nLos valores dados son: \n");
15        // Se muestra el número almacenado en cada elemento del arreglo unidimensional usando for
16        for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
17        {
18            printf("%d ", lista[indice] );
19        }
20    }
21    else printf("el valor dado no es válido"); printf("\n");
22    return 0;
23 }
```



```
Angola05:~ fp20alu27$ Número de lista: 27.
```

input

```
Da un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo
6

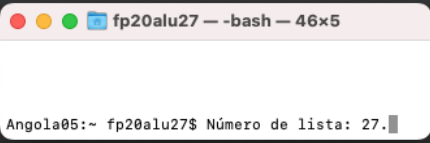
Dar un número entero para el elemento 0 del arreglo0
Dar un número entero para el elemento 1 del arreglo1
Dar un número entero para el elemento 2 del arreglo2
Dar un número entero para el elemento 3 del arreglo34
Dar un número entero para el elemento 4 del arreglo4
Dar un número entero para el elemento 5 del arreglo3

Los valores dados son:
0 1 2 34 4 3
```

Figura 1.6 - Programa 2.c

Programa 2.c modificado

```
1 #include <stdio.h>
2 int main () {
3     int lista[10]; // Se declara el arreglo unidimensional
4     int indice=0;
5     int numeroElementos=0;
6     printf("\nDa un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo\n");
7     scanf("%d",&numeroElementos);
8     if((numeroElementos>=1) && (numeroElementos<=10)) {
9         // Se almacena un número en cada elemento del arreglo unidimensional usando for
10        //for (indice = 0 ; indice <= numeroElementos-1 ; indice++) {
11        while (indice <= numeroElementos-1){
12            printf("\nDar un número entero para el elemento %d del arreglo", indice );
13            scanf("%d",&lista[indice]);
14            indice++;
15        }
16        printf("\nLos valores dados son: \n");
17        // Se muestra el número almacenado en cada elemento del arreglo unidimensional usando for
18        //for (indice = 0 ; indice <= numeroElementos-1 ; indice++)
19        while (indice <= numeroElementos-1){
20            printf("%d ", lista[indice] );
21            indice++;
22        }
23    }
24    else printf("el valor dado no es válido");
25    printf("\n");
26    return 0;
27 }
```



```
Angola05:~ fp20alu27$ Número de lista: 27.
```

input

```
Da un número entre 1 y 10 para indicar la cantidad de elementos que tiene el arreglo
2
Dar un número entero para el elemento 0 del arreglo 2
Dar un número entero para el elemento 1 del arreglo 3
```

Figura 1.7 - Programa 2.c modificado.

Programa 4.c

Normal:

```
1  #include<stdio.h>
2  int main () {
3      int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
4      int *apEnt;
5      apEnt = &a;
6      printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
7      printf("apEnt = &a\n");
8      /*A la variable b se le asigna el contenido de la variable a
9      la que apunta apEnt*/
10     b = *apEnt;
11     printf("b = *apEnt \t-> b = %i\n", b);
12     /*A la variable b se le asigna el contenido de la variable a
13     la que apunta apEnt y se le suma uno*/
14     b = *apEnt +1;
15     printf("b = *apEnt + 1 \t-> b = %i\n", b);
16     //La variable a la que apunta apEnt se le asigna el valor cero
17     *apEnt = 0;
18     printf("*apEnt = 0 \t-> a = %i\n", a);
19     /*A apEnt se le asigna la dirección de memoria que tiene el elemento 0
20     del arreglo c*/
21     apEnt = &c[0];
22     printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt); return 0;
23 }
```

input

```
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

...Program finished with exit code 0
Press ENTER to exit console.
```

fp20alu27 — -bash — 46x5

Angola05:~ fp20alu27\$ Número de lista: 27.

Figura 1.9 - Programa 4.c

Programa 4.c

Modificado:

```
1  #include<stdio.h>
2  int main () {
3      int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
4      int *apEnt;
5      apEnt = &a;
6      printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
7      printf("apEnt = &a\n");
8      /*A la variable b se le asigna el contenido de la variable a
9      la que apunta apEnt*/
10     b = *apEnt;
11     printf("b = *apEnt \t-> b = %i\n", b);
12     /*A la variable b se le asigna el contenido de la variable a
13     la que apunta apEnt y se le suma uno*/
14     b = *apEnt + 1;
15     printf("b = *apEnt + 1 \t-> b = %i\n", b);
16     //La variable a la que apunta apEnt se le asigna el valor cero
17     *apEnt = 0;
18     printf("*apEnt = 0 \t-> a = %i\n", a);
19     /*A apEnt se le asigna la dirección de memoria que tiene el elemento 0
20     del arreglo c*/
21     apEnt = &c[9];
22     printf("apEnt = &c[9] \t-> apEnt = %i\n", *apEnt); return 0;
23 }
```

input

```
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[9]   -> apEnt = 0

...Program finished with exit code 0
Press ENTER to exit console.
```

fp20alu27 — -bash — 46x5

Angola05:~ fp20alu27\$ Número de lista: 27.

Figura 1.10 - Programa 4.c modificado.

Programa 5.c

```
1  #include <stdio.h>
2  int main () {
3      int arr[] = {5, 4, 3, 2, 1};
4      int *apArr; //Se declara el apuntador apArr
5      int x;
6      apArr = arr;
7      printf("int arr[] = {5, 4, 3, 2, 1};\n");
8      printf("apArr = &arr[0]\n");
9      x = *apArr; /*A la variable x se le asigna el contenido del arreglo arr en
10 su elemento 0*/
11      printf("x = *apArr \t -> x = %d\n", x);
12      x = *(apArr+1) +2; /*A la variable x se le asigna el contenido del arreglo arr
13 en su elemento 1*/
14      printf("x = *(apArr+1) \t -> x = %d\n", x);
15      x = *(apArr+2); /*A la variable x se le asigna el contenido del arreglo arr
16 en su elemento 2*/
17      printf("x = *(apArr+2) \t -> x = %d\n", x);
18      x = *(apArr+3); /*A la variable x se le asigna el contenido del arreglo arr
19 en su elemento 3*/
20      printf("x = *(apArr+3) \t -> x = %d\n", x);
21      x = *(apArr+4); /*A la variable x se le asigna el contenido del arreglo arr
22 en su elemento 4*/
23      printf("x = *(apArr+4) \t -> x = %d\n", x); return 0;
24 }
```

input

```
int arr[] = {5, 4, 3, 2, 1};
apArr = &arr[0]
x = *apArr      -> x = 5
x = *(apArr+1)  -> x = 6
x = *(apArr+2)  -> x = 3
x = *(apArr+3)  -> x = 2
x = *(apArr+4)  -> x = 1
```

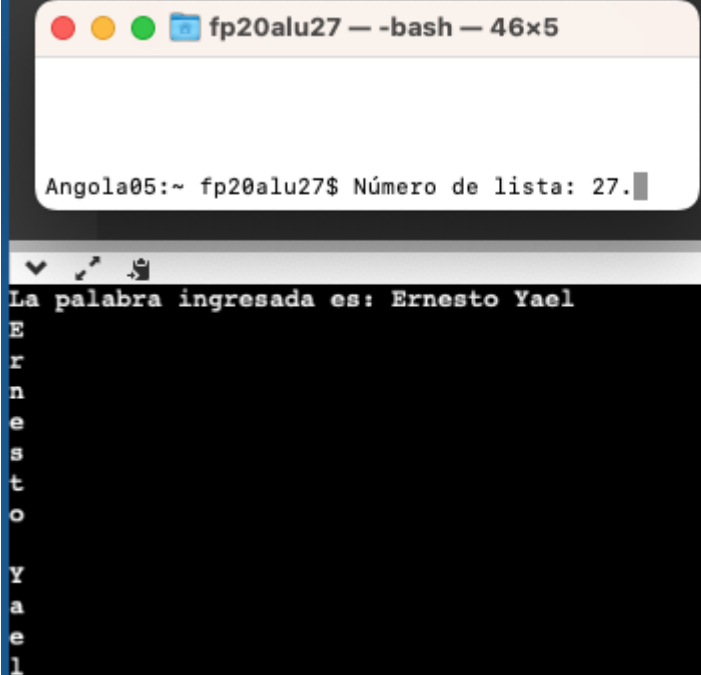
fp20alu27 — -bash — 46x5

Angola05:~ fp20alu27\$ Número de lista: 27.

Figura 1.11 - Programa 5.c modificado.

Programa 7.c

```
1 #include <stdio.h>
2 int main() {
3     char palabra[20];
4     int i=0;
5     printf("Ingrese una palabra: ");
6     //scanf("%s", palabra);
7     gets(palabra);
8     printf("La palabra ingresada es: %s\n", palabra);
9     for (i = 0 ; i < 20 ; i++)
10     {
11         printf("%c\n", palabra[i]);
12     }
13     return 0;
14 }
```



Angola05:~ fp20alu27\$ Número de lista: 27.

La palabra ingresada es: Ernesto Yael
E
r
n
e
s
t
o

Y
a
e
l

Figura 1.12 - Programa 7.c modificado.

Tarea

1.-Indica que realiza el siguiente programa:

```
#include <stdio.h>

int arreglo[] = {20,19,12,17,16,36,14,13,23,31}; // Declarado antes del main

int i, j, n, aux; // Declarado antes del main

main() {

n = 10;

for(i=1; i< i++) { // Error en el for, falta de condición y “;”

    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< i++) { // Error en el for, falta de condición y “;”

    printf("Elemento [%d]: %d\n", i, arreglo[i]);

}

return 0;

}
```

Como el programa se encuentra no se ejecutaría ya que tiene múltiples errores de escritura, empezando por el método main que está declarado después de las variables y el arreglo, lo cual no entraría en lectura y fallaría; al mismo tiempo el for le falta terminar la condición y un punto y coma. Posterior a este, están otros errores de declaración de variables (arreglo) y lo mismo en el segundo for. (Los errores están marcados en el código).

Haciendo la correcciones ya mencionadas el código termina:

```
#include <stdio.h>

int main() {

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

    int i, j, n, aux;

    n = 10;

    for(i=1; i<n; 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<n; i++) {

        printf("Elemento [%d]: %d\n", i, arreglo[i]);

    }

    return 0;

}
```

La funcionalidad del programa consiste en actuar sobre la posición del arreglo, este programa no trata de hacer operaciones o de alterar los valores del arreglo, sino que actúa en la posición de cada elemento, a través del while le da posición y jerarquía entre ellos, acomodandolos de menor a mayor.

```
1  #include <stdio.h>
2  int main() {
3      int arreglo[] = {20,19,12,17,16,36,14,13,23,31};
4      int i, j, n, aux;
5
6      n = 10;
7      for(i=1; i<n; i++){
8          j = i;
9          aux = arreglo[i];
10         while(j>0 && aux<arreglo[j-1]){
11             arreglo[j] = arreglo[j-1];
12             j=j-1;
13         }
14         arreglo[j] = aux;
15     }
16     printf("\n\nLos elementos obtenidos del arreglo son: \n");
17     for(i=0; i<n; i++) {
18         printf("Elemento [%d]: %d\n", i, arreglo[i]);
19     }
20     return 0;
21 }
```

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

Figura 1.13 - Programa tarea 1 modificado.

2.- Genera un programa que le solicite una cadena de letras y números al usuario(emplear arreglo) e imprima solo letras.

-Análisis:

DE: Imprimir solo las letras de un arreglo con una combinación de letras y números.

RE: Imprimir sólo letras sin números.

DS: Obtener la impresión de las letras.

Solución:

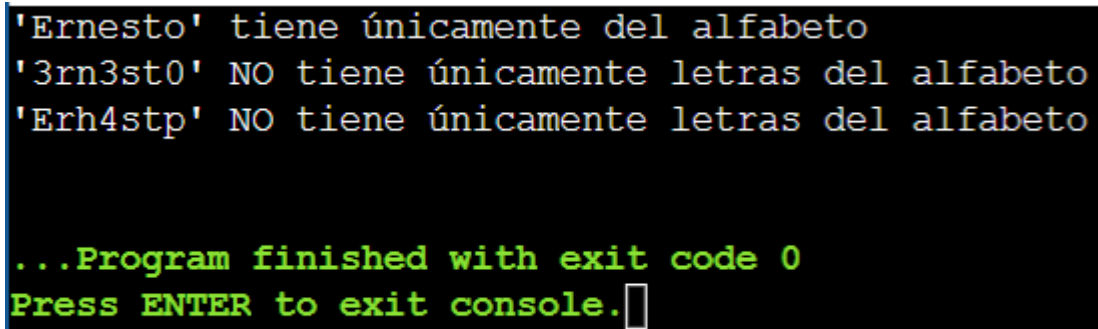
```
#include <stdio.h> // Para printf
#include <ctype.h> // Para isalpha

int Alfabeto(char cadena[]) {
    int i = 0;
    while (cadena[i]) {
        // Si no es del alfabeto y no es un espacio regresamos falso
        if (
            !isalpha(cadena[i])
            && cadena[i] != ' ' // El espacio cuenta como válido, si no, quita la condición
        )
            return 0;
        i++;
    }
    // Si terminamos de recorrer la cadena y no encontramos errores, regresamos verdadero
    return 1;
}

int main() {
    // Esta tiene únicamente letras del alfabeto
    char cadena1[] = "Ernesto";
    if (Alfabeto(cadena1)) {
        printf("'%s' tiene únicamente del alfabeto\n", cadena1);
    } else {
        printf("'%s' NO tiene únicamente letras del alfabeto\n", cadena1);
    }
    // Esta no, porque tiene números
    char cadena2[] = "3rn3st0";
    if (Alfabeto(cadena2)) {
        printf("'%s' tiene únicamente del alfabeto\n", cadena2);
    } else {
        printf("'%s' NO tiene únicamente letras del alfabeto\n", cadena2);
    }
    // Extra
    char cadena3[] = "Erh4stp";
    // Recuerda que puedes almacenar el resultado
    int resultado = Alfabeto(cadena3);
    if (resultado) {
        printf("'%s' tiene únicamente letras del alfabeto\n", cadena3);
    }
}
```

```
} else {  
    printf("'" %s' NO tiene únicamente letras del alfabeto\n", cadena3);  
}  
return 0;  
}
```

Salida terminal

A screenshot of a terminal window with a black background and green text. The output shows three lines of program execution: the first line is a string 'Ernesto' followed by 'tiene únicamente del alfabeto'; the second line is '3rn3st0' followed by 'NO tiene únicamente letras del alfabeto'; the third line is 'Erh4stp' followed by 'NO tiene únicamente letras del alfabeto'. Below these, a message states '...Program finished with exit code 0' and 'Press ENTER to exit console.' followed by a cursor icon.

```
'Ernesto' tiene únicamente del alfabeto  
'3rn3st0' NO tiene únicamente letras del alfabeto  
'Erh4stp' NO tiene únicamente letras del alfabeto  
  
...Program finished with exit code 0  
Press ENTER to exit console. █
```

Figura 1.14 - Salida programa tarea 2.3

Conclusión

El uso de arreglos unidimensionales permiten el almacenamiento y manipulación de información con la cual se pueden efectuar procesos sin la necesidad de una entrada por terminal, al mismo tiempo se analizarán otros comandos los cuales facilitan la obtención de datos por parte del usuario y el cómo almacenarlos, traducirlos a diversos tipos de datos. Con lo que queda cumplido el objetivo de la práctica ya que al ser analizados los programas de la práctica y al modificarlos, se comprobó la facilidad que le permite al desarrollador ahorrar código y precisar de una forma más extensiva que es lo que quiere y no en su programa.