	<b>Carátula para entrega de prácticas</b>	
Facultad de Ingeniería	Laboratorio de docencia	

# Laboratorios de computación salas A y B

<i>Profesor:</i>	Alejandro Esteban Pimentel Alarcón	
<i>Asignatura:</i>	Fundamentos de Programación	
<i>Grupo:</i>	135	
<i>Práctica(s):</i>	09	
<i>Intearante(s):</i>	Vanessa Bazaldúa Morales Torres Alcántara Alan Eliezer	
<i>No. de Equipo de cómputo</i>	Somalia 42	Se los advertí
<i>No. de Lista o</i>	9032 y #6; 8166	
<i>Semestre:</i>	2020 - 1	Solo evaluaré este
<i>Fecha de</i>	28/10/2019	
<i>Observaciones:</i>	Muy bien	

**CALIFICACIÓN:** 10

## ARREGLOS UNIDIMENSIONALES Y MULTIDIMENSIONALES

### Objetivo:

Reconocer la importancia y utilidad de los arreglos, en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, así como trabajar con arreglos tanto unidimensionales como multidimensionales.

### ¿Qué es un arreglo unidimensional?

Un arreglo unidimensional es un tipo de datos estructurado que está formado por una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales. Los datos que se guarden en los arreglos todos deben ser del mismo tipo.

A cada elemento (dato) del arreglo se le asocia una posición particular, el cual se requiere indicar para acceder a un elemento en específico, esto se logra a través del uso de índices.

El tipo de acceso a los arreglos unidimensionales es el acceso directo, es decir, podemos acceder a cualquier elemento del arreglo sin tener que consultar a elementos anteriores o posteriores, esto mediante el uso de un índice para cada elemento del arreglo que nos da su posición relativa. Los arreglos pueden ser unidimensionales o multidimensionales y se utilizan para hacer más eficiente el código de un programa C.

Para implementar arreglos unidimensionales se debe reservar espacio en memoria.

Los arreglos nos permiten hacer un conjunto de operaciones para manipular los datos guardados en ellos, estas operaciones son: ordenar, buscar, insertar, eliminar, modificar entre otras.

### Arreglos unidimensionales



La primera localidad del arreglo corresponde al índice 0 y la última corresponde al índice n-1, donde n es el tamaño del arreglo.

La sintaxis para definir un arreglo en lenguaje C es la siguiente:

**tipoDeDato nombre[tamaño]**

Donde nombre se refiere al identificador del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo. Un arreglo puede ser de los tipos de dato entero, real, carácter o estructura.

Un apuntador es una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

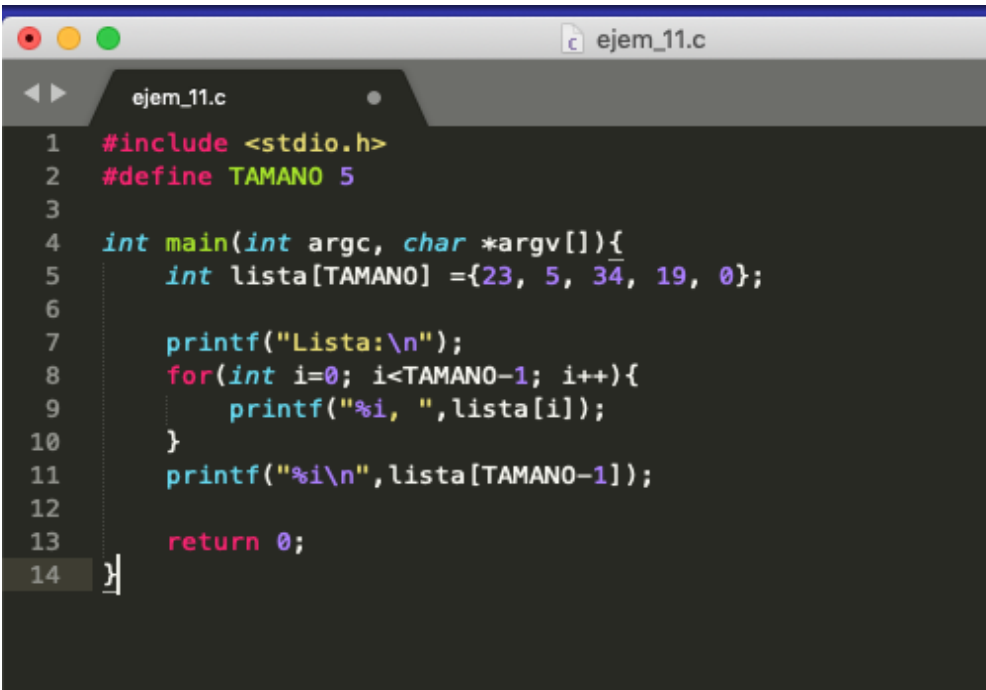
La sintaxis para declarar un apuntador y para asignarle la dirección de memoria de otra variable es, respectivamente:

**TipoDeDato \*apuntador, variable; apuntador = &variable;**

La declaración de una variable apuntador inicia con el carácter \*. Cuando a una variable le antecede un ampersand, lo que se hace es acceder a la dirección de memoria de la misma (es lo que pasa cuando se lee un dato con scanf).

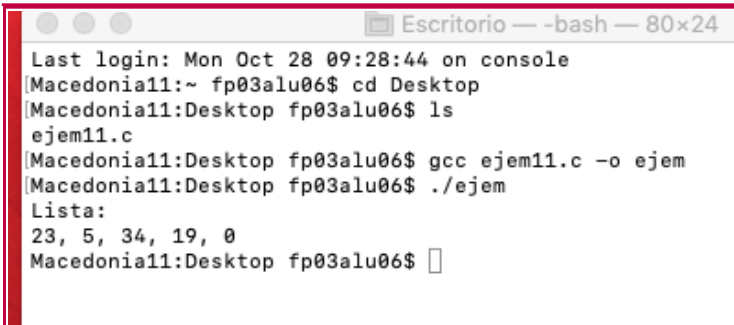
Para empezar, iniciamos con este arreglo unidimensional, el cual ya tiene una lista declarada y lo que hace imprimir la lista declarada en pantalla, solo que en el proceso se agregan unos detalles para que lea completa la lista y muestre el último número sin sobre escribirlo.

### ARREGLOS UNIDIMENSIONALES

A screenshot of a code editor window titled 'ejem\_11.c'. The code is as follows:

```
1  #include <stdio.h>
2  #define TAMANO 5
3
4  int main(int argc, char *argv[]){
5      int lista[TAMANO] = {23, 5, 34, 19, 0};
6
7      printf("Lista:\n");
8      for(int i=0; i<TAMANO-1; i++){
9          printf("%i, ", lista[i]);
10     }
11     printf("%i\n", lista[TAMANO-1]);
12
13     return 0;
14 }
```

Al momento de correrlo en la terminal nos muestra este resultado:

A screenshot of a terminal window titled 'Escritorio — -bash — 80x24'. The terminal output is as follows:

```
Last login: Mon Oct 28 09:28:44 on console
[Macedonia11:~ fp03alu06$ cd Desktop
[Macedonia11:Desktop fp03alu06$ ls
ejem11.c
[Macedonia11:Desktop fp03alu06$ gcc ejem11.c -o ejem
[Macedonia11:Desktop fp03alu06$ ./ejem
Lista:
23, 5, 34, 19, 0
Macedonia11:Desktop fp03alu06$
```

Lenguaje C permite crear arreglos de varias dimensiones con la siguiente sintaxis:

**tipoDato nombre[ tamaño ][ tamaño ]...[tamaño];**

Donde nombre se refiere al identificador del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo por dimensión (el número de dimensiones está determinado por el número de corchetes).

Los tipos de dato que puede tolerar un arreglo multidimensional son: entero, real, carácter o estructura. De manera práctica se puede considerar que la primera dimensión corresponde a los renglones, la segunda a las columnas, la tercera al plano, y así sucesivamente. Sin embargo, en la memoria cada elemento del arreglo se guarda de forma contigua, por lo tanto, se puede recorrer un arreglo multidimensional con apuntadores.

## Arreglos multidimensionales

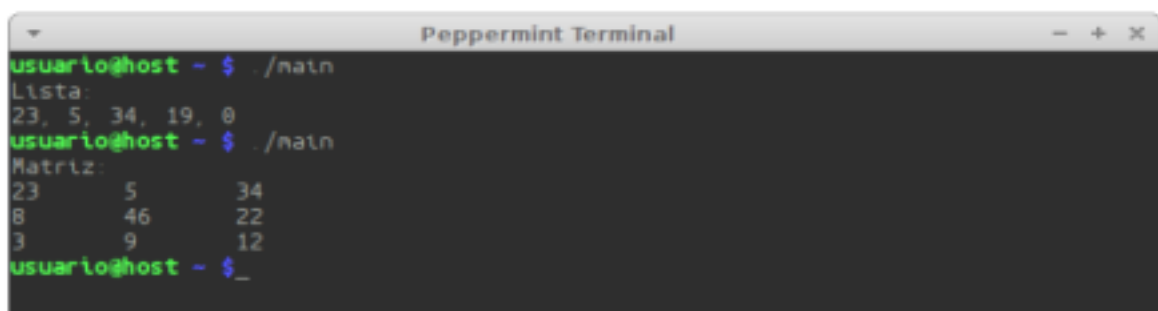
```
#include <stdio.h>
#define DIM 3

int main(int argc, char *argv[])
{
    int matriz[DIM][DIM] = {{23, 5, 34},
                             { 8, 46, 22},
                             { 3, 9, 12}};

    printf("Matriz:\n");
    for(int i=0; i < DIM; i++){
        for(int j=0; j < DIM ; j++){
            printf("%i\t",matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

## Arreglos multidimensionales



```
Peppermint Terminal
usuario@host ~$ ./main
Lista:
23 5 34 19 0
usuario@host ~$ ./main
Matriz:
23      5      34
8       46     22
3       9      12
usuario@host ~$ _
```

### Actividad 1

Hacer un programa que pida:

- ★ Pida al usuario un número.
- ★ Genere un arreglo de esa longitud .
- ★ Pida al usuario los números suficientes para llenar el arreglo.
- ★ Muestre al usuario el número menor y el mayor de dicho arreglo.

```

/media/vanessa/KINGSTON/DC
File Edit Selection Find View Goto Tools Project Preferences
tablas.c promedio.c 1.c
1 #include<stdio.h>
2
3 int main(){
4     int t;
5     printf("Ingresa el tamaño de la lista\n");
6     scanf("%i",&t);
7
8     int lista[t];
9
10    for(int a=0;a<t;a++){
11
12        printf("lista[%i]=\n",a+1);gcc
13        scanf("%i",&lista[a]);
14    }
15
16    int menor;
17    menor=lista[0];
18
19    for(int a=0;a<t;a++){
20        if (lista[a]<menor){
21            menor=lista[a];
22        }
23    }
24
25    int mayor;
26    mayor=lista[0];
27
28    for(int a=0;a<t;a++){
29        if (lista[a]>mayor){
30            mayor=lista[a];
31        }
32    }
33    printf("El número menor es %i\n",menor);
34    printf("El número mayor es %i\n",mayor);
35
36    return 0;
37

```

En este programa con el primer *for* llenamos la lista. Luego declaramos nuestra variable *menor*. Por consiguiente, le asignamos un valor a *menor*, suponiendo que el primer elemento de la lista es el número menor.

El segundo *for* es para obtener el número menor.

Y ahora con el *if* vamos a comparar el siguiente elemento de la lista (porque *i=1*) con el número menor relativo que fue el primero (que la primera vez es *lista[0]*).

Una vez que ya realizamos eso si el siguiente elemento es menor éste pasará a ser el nuevo número menor relativo.

Y seguirá así hasta que se comparen todos los elementos de la lista.

Por último mostramos en pantalla el resultado del número mayor y menor obtenidos.

Ya que realizamos todo lo anterior y comprobamos que funcionara, éstos fueron nuestros resultados:

```

vanessa@Titan:~/Escritorio$ gcc 1.c -o 1
vanessa@Titan:~/Escritorio$ ./1
Ingresa el tamaño de la lista
3
lista[1]=
1
lista[2]=
2
lista[3]=
3
El número menor es 1
El número mayor es 3
vanessa@Titan:~/Escritorio$ ./1
Ingresa el tamaño de la lista
6
lista[1]=
5
lista[2]=
9
lista[3]=
7
lista[4]=
3
lista[5]=
1
lista[6]=
4
El número menor es 1
El número mayor es 9
vanessa@Titan:~/Escritorio$

```

## Actividad 2

Hacer un programa que:

- 🔴 Pida al usuario dos números N y M.
- 🔴 Genere dos matrices de N x M.
- 🔴 Pida al usuario números suficientes para llenar ambas matrices.
- 🔴 Muestre al usuario la matriz resultado de sumar las dos de entrada.

```

tablas.c  promedio.c  a2.c
#include<stdio.h>

int main(){
    int x,y;
    printf("Ingrese columnas de la matriz\n");
    scanf("%i",&x);
    printf("Ingrese filas de la matriz\n");
    scanf("%i",&y);

    int m1[x][y];
    int m2[x][y];
    int m3[x][y];
    printf("Llenar la Matriz 1\n");
    printf("\n");
    for(int a=0;a<x;a++){
        for (int b=0;b<y;b++){
            printf("lugar[%i][%i]\n",a+1,b+1);
            scanf("%i",&m1[a][b]);
        }
    }
    printf("\n");

    printf("Llenar la Matriz 2\n");

    for (int a=0;a<x;a++){
        for (int b=0;b<y;b++){
            printf("lugar[%i][%i]\n",a+1,b+1);
            scanf("%i",&m2[a][b]);
        }
    }
}

```

```

34
35
36     }
37     printf("\n");
38
39     printf("La suma de las dos matrices es\n");
40
41     for (int a=0;a<x;a++){
42         for (int b=0;b<y;b++){
43             m3[a][b]= m1[a][b]+ m2[a][b];
44             printf("%i\t",m3[a][b]);
45         }
46         printf("\n");
47     }
48     return 0;
49 }
50
51
52
53
54
55
56

```

En este programa lo que hacemos de inicio, es declarar las variables de matrices: 1, 2 y resultado, luego le indicamos al usuario que ingrese los números para rellenar el arreglo numero 1 y eso es posible gracias al primer *for* el cual tiene dentro otro *for* que de igual manera nos ayudará a rellenar el segundo arreglo y el tercer *for* tiene como función sumar e imprimir el resultado de la suma.

Una vez que realizamos los detalles para que el programa corriera, estos fueron nuestros resultados:

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
vanessa@Titan:~/Escritorio$ ./a2
Ingrese columnas de la matriz
2
Ingrese filas de la matriz
3
Llenar la Matriz 1
lugar[1][1]
1
lugar[1][2]
2
lugar[1][3]
3
lugar[2][1]
6
lugar[2][2]
5
lugar[2][3]
48
Llenar la Matriz 2
lugar[1][1]
7
lugar[1][2]
9
lugar[1][3]
8
lugar[2][1]
68
lugar[2][2]
8
lugar[2][3]
8
La suma de las dos matrices es
8      11      11
74     13     56

```

### Conclusión:

Para concluir, hemos observado que los *for* tienen una gran utilidad en este tipo de arreglos, o al menos son una forma de hacerlos más eficientes o más prácticos ya que precisamente, como lo habíamos visto en actividades pasadas los *for* nos ayudan a crear arreglos o listas.

En este caso los arreglos son más complicados, ya que estamos trabando con matrices, pero solo fue necesario aclarar bien las ideas, pensar que teníamos que hacer y generar una buena estructura de cada programa sin complicarnos tanto.