



**Ingeniería en Desarrollo de Software**  
**Primer Semestre**

Programa de la asignatura:  
**Fundamentos de programación**

**Unidad 5. Estructuras de datos**

Clave:

**TSU    Licenciatura**

16141102 / 15141102

**Universidad Abierta y a Distancia de México**





### Índice

|   |    |
|---|----|
| Unidad 5: Estructuras de datos.....                 | 3  |
| Introducción .....                                  | 3  |
| Propósitos.....                                     | 4  |
| Competencia específica.....                         | 5  |
| 5.1. Arreglos .....                                 | 5  |
| 5.1.1. Definición y tipos de arreglos .....         | 5  |
| 5.1.2. Declaración e inicialización .....           | 8  |
| 5.1.3. Acceso a los elementos de un arreglo .....   | 11 |
| 5.1.4. Ciclos y arreglos .....                      | 12 |
| 5.1.5. Cadenas .....                                | 21 |
| 5.2. Estructuras.....                               | 25 |
| 5.2. Definición, declaración e inicialización ..... | 25 |
| 5.2.2. Acceso a sus elementos .....                 | 27 |
| Cierre de la Unidad.....                            | 34 |
| Fuentes de consulta .....                           | 35 |



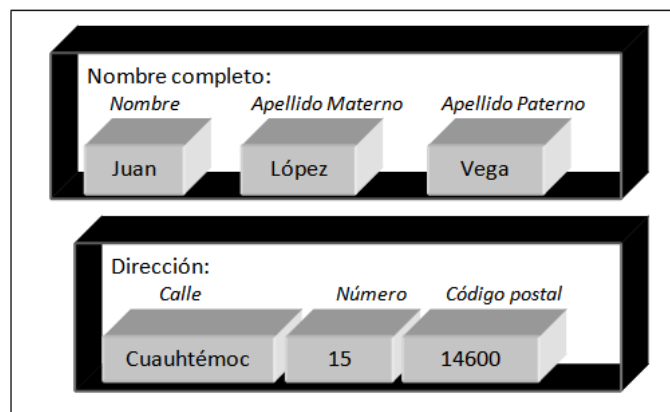
### Unidad 5: Estructuras de datos

#### Introducción



En muchas ocasiones nos vemos en la necesidad de procesar datos que están relacionados entre sí, a este tipo de datos se le conoce como estructurados, ya que están compuestos de un conjunto de datos básicos (recuerda la clasificación de datos presentada en la unidad 3 en la tabla 3.2 Tipo de datos).

Por ejemplo pensemos en el nombre completo de una persona, que está compuesto nombre, apellido paterno y apellido materno, o bien, en una dirección, formada por nombre de la calle, número y código postal, en este último caso no sólo está formada por varios datos simples sino que además podemos considerarlos de diferentes tipos (5.1).



5.1: Ejemplos de datos Estructurados

Con este tipo de datos será útil poder hacer referencia a ellos bajo un mismo *identificador*, y así tratarlos como una *unidad*. Una *estructura de datos* es un mecanismo de agrupación de datos que facilitan el manejo de datos estructurados y que se caracteriza por la forma en que se accede a sus elementos.

Pensemos en otro ejemplo en el cual se tienen datos relacionados, supongamos que nos enfrentamos al siguiente problema:



**Problema 5.1:** Se requiere un programa para llevar el registro de calificaciones de un grupo de diez estudiantes y generar reportes que incluyan datos como el promedio del grupo, la calificación máxima, el número de estudiantes que tienen una calificación superior al promedio del grupo, entre otros.

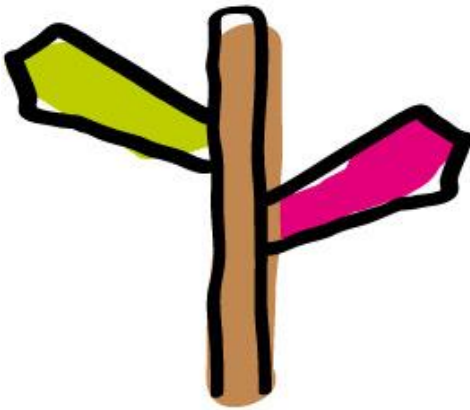
En este caso, a diferencia de los ejemplos anteriores, es claro que las calificaciones de cada estudiante se puede tratar como un dato simple e independiente de los otros, sin embargo las operaciones que se desean realizar serán las mismas para todo el conjunto de calificaciones, de tal forma que habría que escribir una serie de instrucciones secuenciales para ingresar cada dato y procesarlo. Por ejemplo, para ingresar los datos se requiere leer una por una cada calificación, para obtener el promedio se tendría que hacer la suma de todas y después dividir las entre 10, hasta aquí no se ha complicado mucho, pero imagina todas las comparaciones que debes hacer para identificar cuál es la calificación mayor.

Es claro que este método resulta de lo más ineficiente, y por supuesto si consideramos la posibilidad de modificar el programa para que sea capaz de procesar 60 o más calificaciones, el programa además de extenderse, implica reestructurarlo en su totalidad y que éste sea más complejo que la versión anterior. En cambio si consideramos a todos las calificaciones como un dato estructurado podemos hacer uso de una estructura de dato que nos facilite su manipulación.



Existen diferentes tipos de estructuras de datos, cada una caracterizada por la forma de acceso a sus elementos, y el tipo que estos pueden tener, así tenemos arreglos, listas, colas, tablas, pilas, entre otros. No obstante, para esta unidad nos centraremos sólo en las estructuras de datos que implementa el lenguaje C de forma directa: los arreglos y las estructuras.

### Propósitos



- Determinarás las estructuras de datos involucradas en la solución de un problema.
- Diseñarás soluciones empleando arreglos y estructuras (registros).
- Utilizarás arreglos y estructuras (registros) en programas escritos en lenguaje C.

### Competencia específica



Utilizar estructuras de datos para almacenar y manipular los datos de un programa por medio del desarrollo de programas en lenguaje C.

### 5.1. Arreglos

El uso de arreglos facilita y hace más eficiente la declaración y manipulación de una *colección de datos de un mismo tipo* que están relacionados entre sí, como es el caso de las calificaciones en el Problema1, ya que todas las calificaciones se pueden considerar como valores enteros.

#### 5.1.1. Definición y tipos de arreglos

“Un *arreglo* se define como una colección finita, homogénea y ordenada de elementos. Finita ya que para todo arreglo debe especificarse el número máximo de elementos que podrá contener; la homogeneidad se refiere a que todos los elementos deben ser del mismo tipo, y ordenada porque es posible determinar



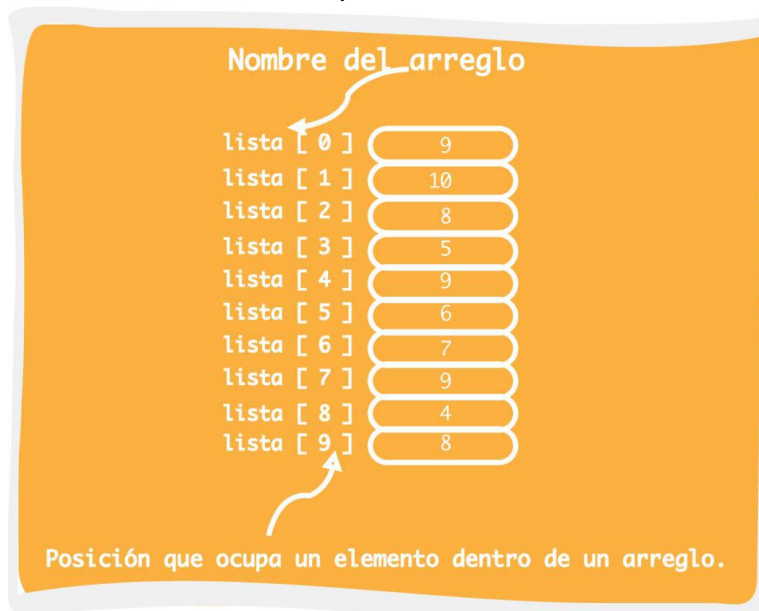


cuál es el primer elemento, cual el segundo, y así hasta el enésimo elemento”(Cairo Osvaldo, Guardati Buemo Silvia, 1993).

La posición que ocupa un elemento dentro de un arreglo se le denomina formalmente *índice* y siempre es un número entero. El *tamaño* o *longitud* de un arreglo se define como el número de elementos que lo constituyen. La *dimensión* de un arreglo está relacionada con el número de índices necesarios para especificar un elemento en particular.

Podemos clasificar a los arreglos de acuerdo a su dimensión como *unidimensionales* o *multidimensionales*.

Los arreglos *unidimensionales*(también llamados *lineales*) reciben su nombre debido a que cualquier elemento es referenciado por un único índice, por ejemplo retomando el caso de las calificaciones del problema 5.1, éstas pueden ser almacenadas en un arreglo unidimensional como el que se muestra en la **¡Error! No se encuentra el origen de la referencia.**, en donde el nombre del arreglo es lista y los nombres de las variables donde se almacenan las calificaciones son: lista[0], lista[1], lista[2], lista[3], lista[4] ..., lista[9]. En este caso el nombre en común es lista y lo único que cambia para cada elemento es el número que le corresponde a cada variable según la posición que ocupa en la lista. Observa que un solo índice es suficiente para diferenciar a un elemento de otro.



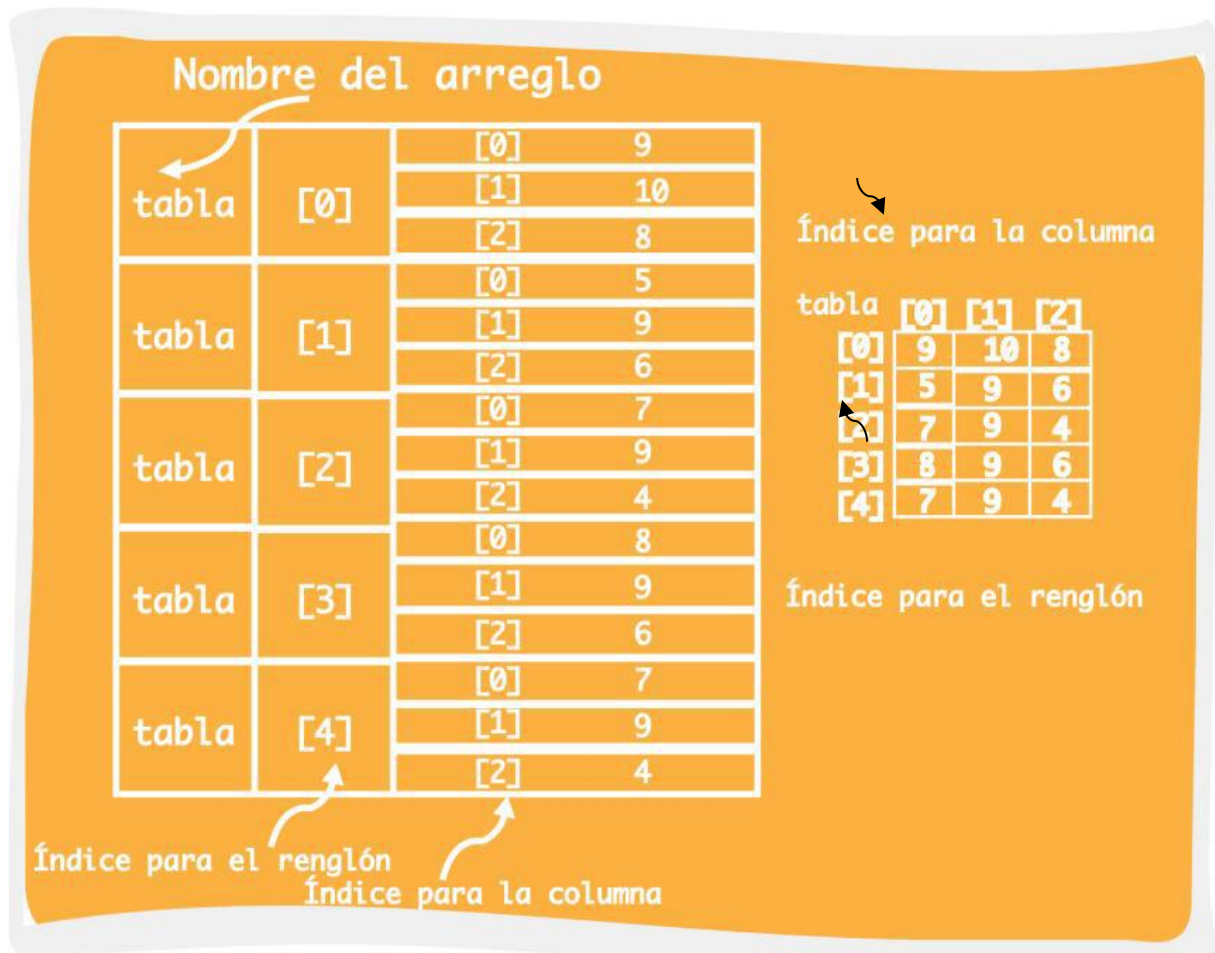
Representación gráfica de un arreglo unidimensional

Por otro lado los arreglos *multidimensionales* son aquellos para los cuales un solo índice no es suficiente para poder referenciar a un elemento individual, los arreglos *bidimensionales* son el caso más comúnmente utilizado de arreglos multidimensionales y por tanto los únicos que presentaremos.



“Un arreglo *bidimensional* es un conjunto de datos homogéneos, finito y ordenado, donde se hace referencia a cada elemento por medio de dos índices. El primero de los cuales generalmente se utiliza para indicar renglón y el segundo para indicar columna” (Cairo Osvaldo, Guardati Buemo Silvia, 1993).

Un arreglo bidimensional también puede verse como una tabla de valores, o bien como un arreglo de arreglos, de ahí la necesidad de dos índices, en la **¡Error! No se encuentra el rigen de la referencia.** siguiente se muestra un ejemplo gráfico de un arreglo bidimensional, en la cual del lado derecho podemos ver al arreglo como una tabla y del lado izquierdo representado como un arreglo de arreglos, Observa que cada renglón de la tabla es cada uno de los elementos del arreglo de arreglos. Es claro que con un solo índice no podríamos identificar a un único elemento ya que solo podríamos ubicar toda una columna o todo un renglón, en cambio la combinación de renglón-columna si nos identifica a un elemento en particular.



Representación gráfica de un arreglo bidimensional

### 5.1.2. Declaración e inicialización

En lenguaje C los índices de los arreglos siempre empiezan en cero, es decir, al primer elemento del arreglo le corresponde la posición 0, al segundo la posición 1, al tercero la posición 2 y así sucesivamente hasta llegar al elemento TAM-1, donde TAM corresponde al tamaño del arreglo.

La declaración de un arreglo consiste en reservar espacio de memoria suficiente para el conjunto de datos homogéneos. La declaración de *una variable de tipo arreglo* sigue las mismas reglas que para las variables simples; con la diferencia de que ahora será necesario especificar el tamaño del arreglo, esto se hace escribiendo el tamaño del arreglo encerrado entre corchetes [ TAM ], después del identificador.

La sintaxis para la declaración de un arreglo unidimensional en lenguaje C es la siguiente:  
<tipo><nombre>[<tamaño>];





Y para un arreglo bidimensional es:

`<tipo><nombre>[<tamaño1>][<tamaño2>];`

El tipo de dato para los arreglos puede ser cualquier tipo básico, es decir entero, flotante o carácter (en C int, float, double o char ). De todos ellos los arreglos de tipo carácter (char) tienen un tratamiento especial, ya que un arreglo de este tipo se considerara una *cadena*. Debido a la importancia que tienen las cadenas en la programación más adelante los trataremos de manera particular.

Al igual que las variables simples, un arreglo puede inicializarse al momento de ser declarado, para ello se utiliza el operador asignación "=", pero como un arreglo almacena a un conjunto de datos, es necesario inicializarlo con un conjunto de valores, los cuales se indican mediante llaves, separando por comas cada elemento del conjunto de valores iniciales, la sintaxis se muestra a continuación:

`<tipo><nombre>[<tamaño>]={<valor0>,<valor1>,...,<valorTAM-1>};`

La asignación de cada valor inicial se hace consecutivamente desde el elemento 0, por tanto no es posible asignar valores a elementos saltados.

Veamos como ejemplo la declaración del arreglo unidimensional lista (**¡Error! No se encuentra el origen de la referencia.**) planteado para las calificaciones del problema1. Inicializando sus elementos en la declaración queda como:

```
int lista[10] = {9,10,8,5,9,6,7,9,4,8};
```

En el caso de los arreglos bidimensionales la sintaxis es la siguiente:

```
<tipo><nombre>[<tamaño1>][<tamaño2>]={  
    {<valor00>,<valor01>,...,<valor0(TAM21)>},  
    {<valor10>,<valor11>,...,<valor1(TAM21-1)>},...,  
    {<valor(TAM1-1)0>,<valor(TAM2-1)1>,...,<elem(TAM1-1)(TAM2-1)>}  
};
```

Veamos ahora como queda la declaración del arreglo bidimensional tabla mostrado en la **¡Error! No se encuentra el origen de la referencia.** inicializando su valores:

```
int tabla[5][3]={9,10,8},{5,9,6},{7,9,4},{8,9,6},{7,9,4}};
```

Aunque también es posible declararlo de la siguiente forma:



```
int tabla[5][3]={9,10,8,5,9,6,7,9,4,8,9,6,7,9,4};
```

Esta es debido a que como ya se dijo antes un arreglo bidimensional se puede ver como un arreglo de arreglos.

Por otro lado, en lenguaje C siempre es necesario especificar el tamaño del arreglo al momento de declararlo, sin embargo esto se puede hacer de forma explícita o implícita.

- Explícitamente es cuando se especifica el tamaño dentro de los corchetes que siguen al identificador, como en los ejemplos anteriores.
- De forma implícita se hace cuando el arreglo es inicializado con un conjunto de valores, y se omite el tamaño dentro de los corchetes, entonces el compilador asume el tamaño del arreglo igual al tamaño del conjunto de valores iniciales, de tal forma que la declaración del arreglo lista puede quedar como:

```
int lista[] = {9,10,8,5,9,6,7,9,4,8};
```

Observa que en este caso no se escribe el tamaño dentro de los corchetes, pero como hay 10 elementos en el conjunto de valores iniciales, el compilador de C asume un tamaño 10 para el arreglo. Para los arreglos bidimensionales, sólo es posible especificar una dimensión de forma implícita, el tamaño de renglones siempre debe hacerse de forma explícita.

La asignación de un conjunto de valores al arreglo, en una sola operación de asignación, únicamente es posible en su declaración, si se intenta realizar en otro momento el compilador marcará un error, ya que en cualquier otra parte del programa sólo se podrán asignar valores simples a cada uno de los elementos por separado. Es importante señalar que cuando se desea inicializar el arreglo al declararlo, es posible inicializar sólo algunos de sus elementos, pero en este caso se tendría que especificar explícitamente el tamaño, además se debe recordar que la asignación de valores iniciales es consecutiva desde el elemento 0. Los elementos para los cuales no se indique un valor inicial, automáticamente se inicializan en cero. Por ejemplo la declaración:

```
int lista[10] = {5};
```

Reservará espacio en memoria para los 10 elementos del arreglo de los cuales al primer elemento se le asignará un 5 y al resto se les asignará un cero. En el caso de los arreglos bidimensionales es posible declarar sólo algunos elementos por renglón, siempre y cuando los elementos sean consecutivos, como en el caso de los unidimensionales. Por ejemplo la siguiente declaración para el arreglo tabla:

```
int tabla[5][3]={{9,10},{5},{7,9,4},{8,9,}};
```



Darí­a como resultado la siguiente asignación de valores iniciales

|     | [0] | [1] | [2] |
|-----|-----|-----|-----|
| [0] | 9   | 10  | 0   |
| [1] | 5   | 0   | 0   |
| [2] | 7   | 9   | 4   |
| [3] | 8   | 9   | 0   |
| [4] | 0   | 0   | 0   |

En el caso de que la declaración fuera:

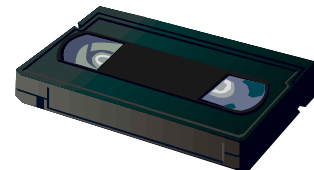
```
int tabla[5][3]={9,10,5,7,9,4,8,9,};
```

Entonces la asignación de valores iniciales se harí­a de la siguiente forma

|     | [0] | [1] | [2] |
|-----|-----|-----|-----|
| [0] | 9   | 10  | 5   |
| [1] | 7   | 9   | 4   |
| [2] | 8   | 9   | 0   |
| [3] | 0   | 0   | 0   |
| [4] | 0   | 0   | 0   |

Para consultar un ejemplo de estructura de datos consulta el recurso de video.

Para ingresar al video haz clic en la imagen siguiente:



### 5.1.3. Acceso a los elementos de un arreglo

Para referirse a un elemento del arreglo es necesario indicar el nombre del arreglo seguido del índice o índices correspondientes al elemento que deseamos acceder. Para ello se debe seguir la siguiente sintaxis.



Elementos de un arreglo unidimensional:  
<nombre del arreglo> [<índice>];

Elementos de un arreglo bidimensional:  
<nombre del arreglo> [<índice de renglón>]  
[<índice de columna>];

Observa que para cada índice se utilizan corchetes separados. Cada elemento del arreglo se puede tratar igual que a cualquier otra variable, es decir, podemos asignarle un valor, incluir en una expresión algebraica o lógica, imprimir en pantalla su valor, asignarle desde el teclado un valor, etc.

Veamos algunos ejemplos:

| Instrucción                                 | Descripción   |
|---|---|
| <code>tabla[0][2] = 8;</code>               | Asignar el valor de 8 al tercer elemento del primer renglón de arreglo tabla      |
| <code>printf("%d", lista[4]);</code>        | Imprimir en pantalla el quinto elemento del arreglo lista                         |
| <code>scanf("%d", &amp;tabla[0][0]);</code> | Lee un entero desde teclado y asignarlo en la primera posición del arreglo tabla. |
| <code>lista[1]++;</code>                    | Incrementar en uno el valor del segundo elemento del arreglo lista                |

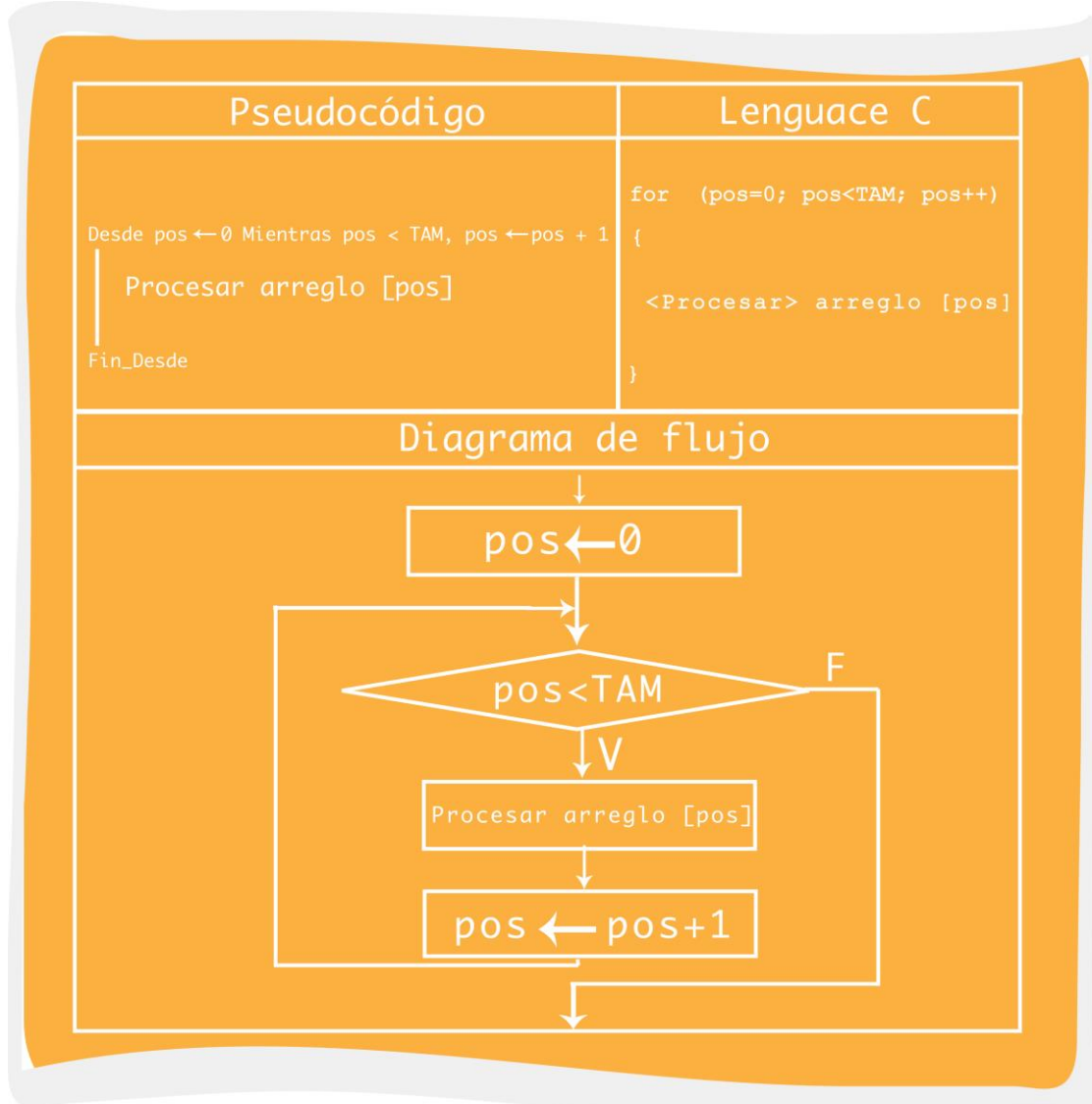
### 5.1.4. Ciclos y arreglos

Los arreglos y los ciclos están estrechamente relacionados, podríamos afirmar que en cualquier programa que se emplee un arreglo éste será manipulado por medio de una estructura repetitiva. Anteriormente mencionamos que un arreglo se utiliza cuando queremos almacenar y manipular datos relacionados entre sí y, generalmente, cuando tenemos un arreglo debemos ejecutar el mismo conjunto de operaciones sobre cada uno de sus elementos.

En lenguaje C el índice del elemento se puede especificar mediante una expresión, es decir una variable, una constante o como el resultado de una operación, lo que hace posible ir cambiando el índice de un elemento dentro de un ciclo sin tener que escribir una serie de instrucciones secuenciales para realizar la misma operación sobre cada uno de los elementos del arreglo. La forma general de procesar un arreglo



unidimensional por medio de un ciclo se muestra en la **¡Error! No se encuentra el origen de la referencia.** observa cómo la variable contador **pos** del ciclo, también se utiliza como índice para el elemento del arreglo, de tal forma que en cada iteración se haga referencia a un elemento diferente del arreglo.



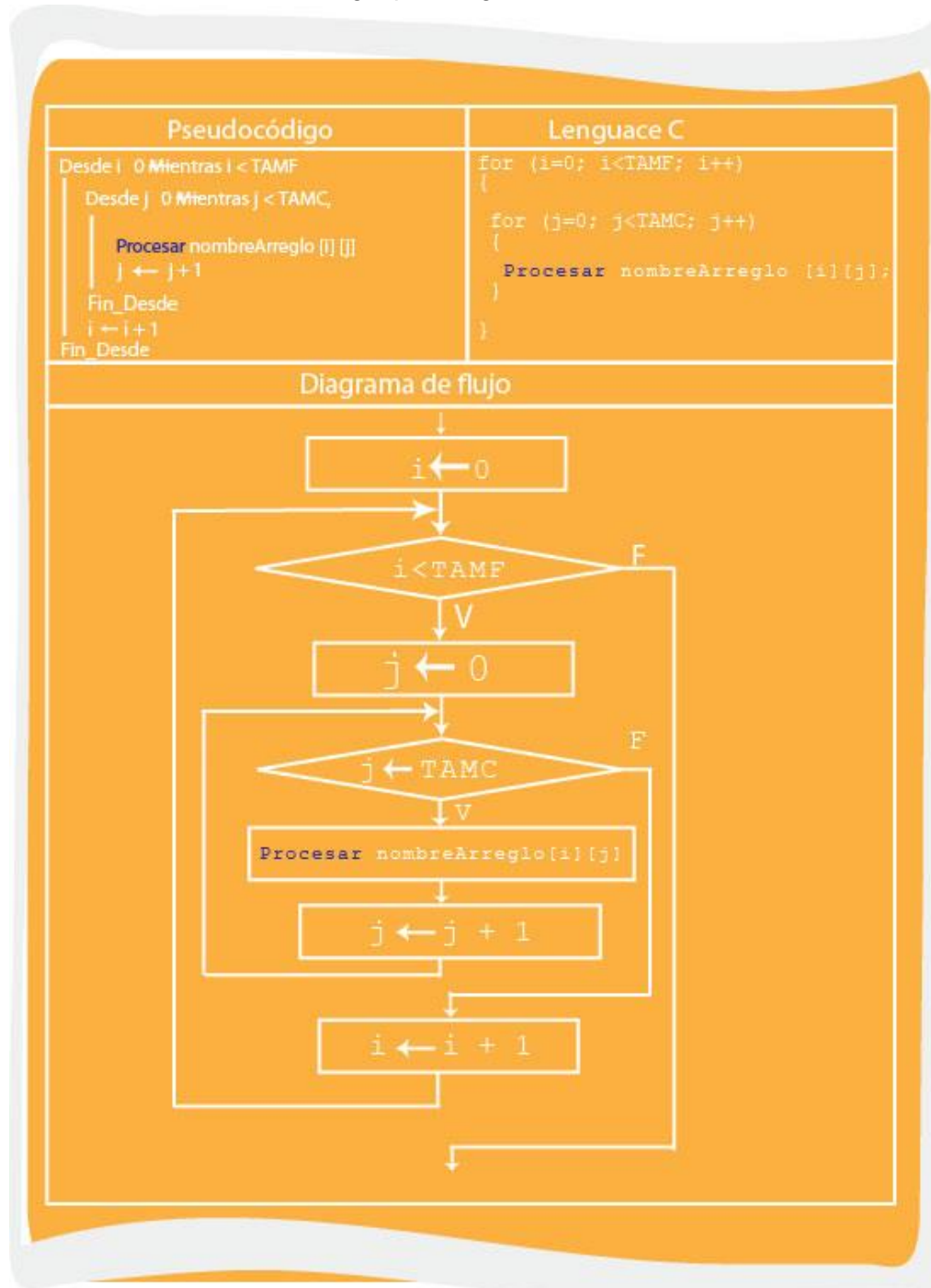
Procesamiento de arreglos unidimensionales mediante ciclos

No es difícil intuir que para el caso de los arreglos bidimensionales será necesario no solo un ciclo sino un par de ciclos anidados, ya que tenemos dos dimensiones, cada uno de los ciclos se encargará de variar a un índice, la estructura general para estos ciclos se muestra en la **¡Error! No se encuentra el origen de la referencia.** Dos ciclos están anidados cuando está uno dentro de otro, de tal forma que por cada iteración del externo, el interno completa todo un ciclo. Considerando esto, observa que en la **¡Error! No se**





encuentra el origen de la referencia. siguiente se toma como contador a la variable  $i$  para el ciclo externo, la cual también se utiliza como índice de renglón. Asimismo, se utiliza a la variable  $j$  como contador para el ciclo interno, además de índice de columna, de esta manera se está recorriendo el arreglo por renglón.



Procesamiento de arreglos bidimensionales mediante ciclos anidados



Si analizamos el diagrama de flujo podemos observar que para la primera iteración del ciclo externo, la variable  $i$  tiene el valor de 0 mientras que  $j$  toma los valores desde 0 hasta TAMC-1 (TAMC es el número total de columnas) de tal forma que el renglón se mantiene fijo mientras nos movemos en todas las columnas, en la siguiente iteración cambia el renglón ya que  $i$  toma el valor de 1 y recorremos nuevamente todas las columnas, este proceso se repite hasta el valor final de  $i$  que es TAMF-1 (TAMF es el número de renglones).

**Ejemplo 5.1:** Veamos cómo utilizar arreglos en la solución de un problema, resolviendo como ejemplo parte del problema 1, enfocándonos únicamente en la lectura de las 10 calificaciones y el cálculo del promedio.

Para almacenar las calificaciones se puede utilizar un arreglo unidimensional de tipo entero, será necesario pedir al usuario que ingrese las 10 calificaciones para poder realizar las operaciones necesarias para el cálculo del promedio, es decir, la suma de las mismas y la división entre el total de calificaciones, para finalmente imprimir en pantalla el promedio, adicionalmente se imprimirá también la lista de calificaciones ingresadas. No existe ninguna restricción que sea necesaria considerar.

Cada uno de los procesos que se van a realizar sobre las calificaciones, es decir leerlas, sumarlas e imprimirlas en pantalla se pueden implementar mediante ciclos, en vez de tener que escribir 10 veces la misma expresión para cada una de las 10 calificaciones. La lectura y la suma de las calificaciones se pueden implementar dentro del mismo ciclo. De esta manera podemos resumir el análisis del problema de la siguiente forma:

**Datos de entrada:** Calificaciones de los 10 estudiantes (calif [ ])

**Salida:** Promedio de calificaciones (prom)

**Método:**

$$\text{prom} = \frac{\sum_{i=0}^9 \text{calif}[i]}{10}$$

La solución del problema representada en pseudocódigo se muestra en el siguiente algoritmo.



```
Inicio
    suma ← 0
    Desde i ← 0 mientras i<10, i ← i+1
        Imprimir "Ingresa la calificación" i
        Leer calif[i]
        suma← suma+calif[i]
    Fin Desde
    prom ← suma/10
    Imprimir "Las calificaciones ingresadas fueron:"
    Desde i ← 0 mientras i<10, i ← i+1
        Imprimir "Calificación" i ":" calif[i]
    Fin Desde
    Imprimir "Calificación promedio = " prom
Fin
```

**Algoritmo 5.1.** Promedio de calificaciones

La codificación del algoritmo anterior es la siguiente:

---

```
/*Directivas de preprocesador*/
#include <stdio.h>
#include <stdlib.h>
/* Definimos como constante simbólica el tamaño del arreglo*/
#define TAM 10
/* Definición de función principal */
main( )
{
    /*Declaración del arreglo calificaciones*/
    int calif[TAM];
    double prom = 0;
    int i;

    printf("*****\n");
    printf("* El siguiente programa calcula el promedio de *\n");
    printf("* un grupo de diez estudiantes *\n");
    printf("*****\n");

    /*Lectura y suma de las calificaciones*/
    for(i=0; i < TAM; i++)
    {
        printf("Proporciona la calificación %d: ",i+1);
        scanf("%d", &calif[i]);
        prom = prom + calif[i];
    }
}
```

---



```
}
/*Cálculo e impresión del promedio*/
prom = prom/TAM;
/*Impresión de las calificaciones*/
printf("\nLas calificaciones ingresadas fueron: \n");
for(i=0; i < TAM; i++)
    printf("\nCalificacion %d: %d", i+1, calif[i]);

printf("\n\n\tPromedio = %.2f\n\n", prom);
system("pause");
}
```

**Programa 5.1:** promCalificaciones.c

En la siguiente figura se muestra una ejecución del programa.

```
C:\Users\Diana\Documents\CursosSEP\Programacion\ejemploArreglos.exe
*****
* El siguiente programa calcula el promedio de *
* un grupo de diez estudiantes *
*****
Proporciona la calificación 1: 9
Proporciona la calificación 2: 3
Proporciona la calificación 3: 2
Proporciona la calificación 4: 5
Proporciona la calificación 5: 6
Proporciona la calificación 6: 8
Proporciona la calificación 7: 8
Proporciona la calificación 8: 9
Proporciona la calificación 9: 10
Proporciona la calificación 10: 9

Las calificaciones ingresadas fueron:

Calificacion 1: 9
Calificacion 2: 3
Calificacion 3: 2
Calificacion 4: 5
Calificacion 5: 6
Calificacion 6: 8
Calificacion 7: 8
Calificacion 8: 9
Calificacion 9: 10
Calificacion 10: 9

        Promedio = 6.90

Presione una tecla para continuar . . .
```

**5.2:** Ejecución del programa promCalificaciones.c

Observa que el tamaño del arreglo se especifica por medio de una constante simbólica, utilizando la directiva `#define`, esto facilita el cambiar el tamaño del arreglo sin tener que hacer cambios en todo el código.

A continuación se presenta otro ejemplo para ilustrar el uso de arreglos bidimensionales.



**Ejemplo 5.2:** Se requiere un programa que calcule el determinante de una matriz de 2x2. Considerando la siguiente información.

Dada la siguiente matriz:

$$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$$

Su determinante se define como:  $\Delta = a_{00}a_{11} - a_{01}a_{10}$

**Análisis del problema:** Para este problema se puede utilizar un arreglo bidimensional de 2 renglones y 2 columnas para almacenar los valores de la matriz, los cuales se pueden solicitar al usuario utilizando la estructura de ciclos anidados presentada en la **¡Error! No se encuentra el origen de la referencia.**, nuestro dato de salida será el valor del determinante y adicionalmente también se mostrará en pantalla la matriz ingresada.

**Datos de entrada:** Elementos de la matriz (A[ ][ ])

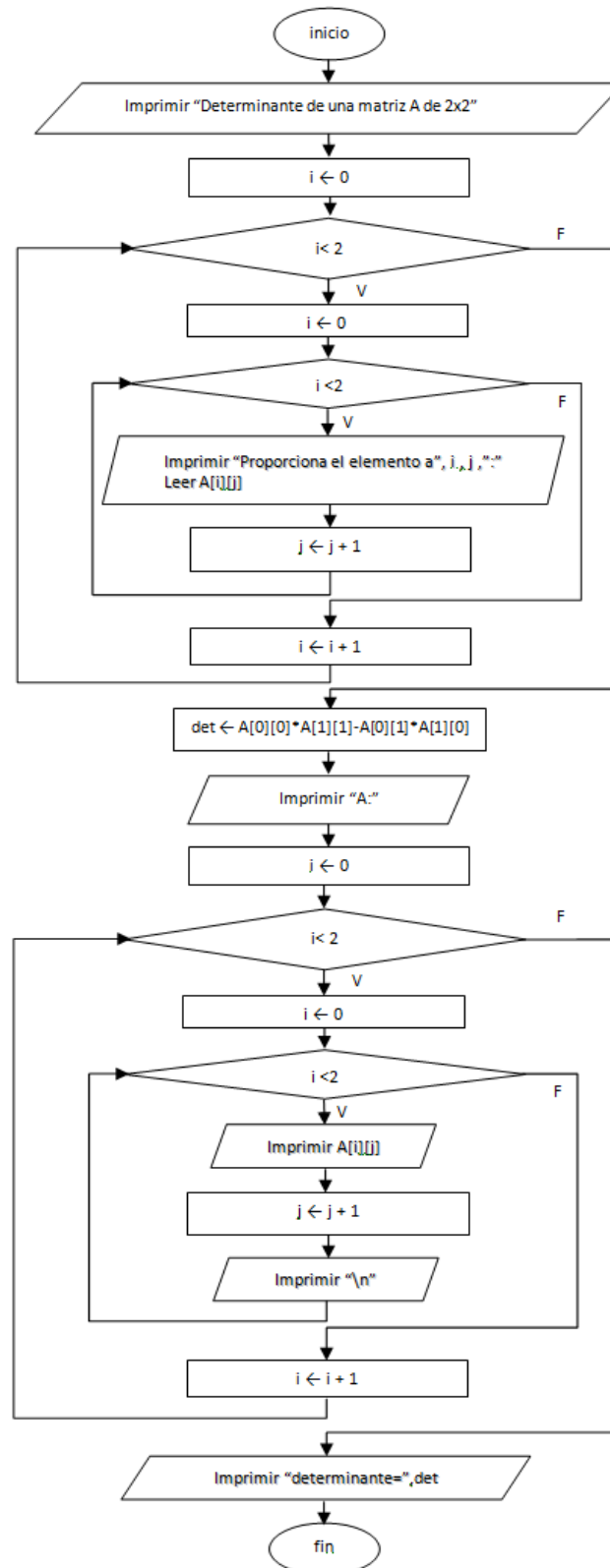
**Salida:** Valor del determinante (det)

**Método:**

$$det = A[0][0] * A[1][1] - A[0][1] * A[1][0]$$

La solución del problema se da en el siguiente diagrama de flujo.





**Algoritmo 5.2:** Determinante de una matriz 2x2



La codificación del algoritmo se deja como ejercicio al lector, sin embargo, en la siguiente figura se muestra un ejemplo de ejecución del programa.

```
/* Directivas al preprocesador */
#include <stdlib.h>
#include <stdio.h>

/* Constantes con el tamaño de la matriz */
#define TAM 2

/* Función principal */
main()
{
    int i, j;
    float det;
    float A[TAM][TAM]; /*declaración de la matriz*/

    /* Mensaje de bienvenida */
    printf("*****\n");
    printf("* Determinante de una matriz A de 2x2 *\n");
    printf("*****\n");

    /* Lectura de la matriz A */
    for(i=0; i<TAM; i++)
        for(j=0; j<TAM; j++){
            printf("\nProporciona el elemento A[%d][%d]: ", i,j);
            scanf("%f", &A[i][j]);
        }
    det = A[0][0]*A[1][1] - A[0][1]*A[1][0];

    printf("\nA: \n\t");

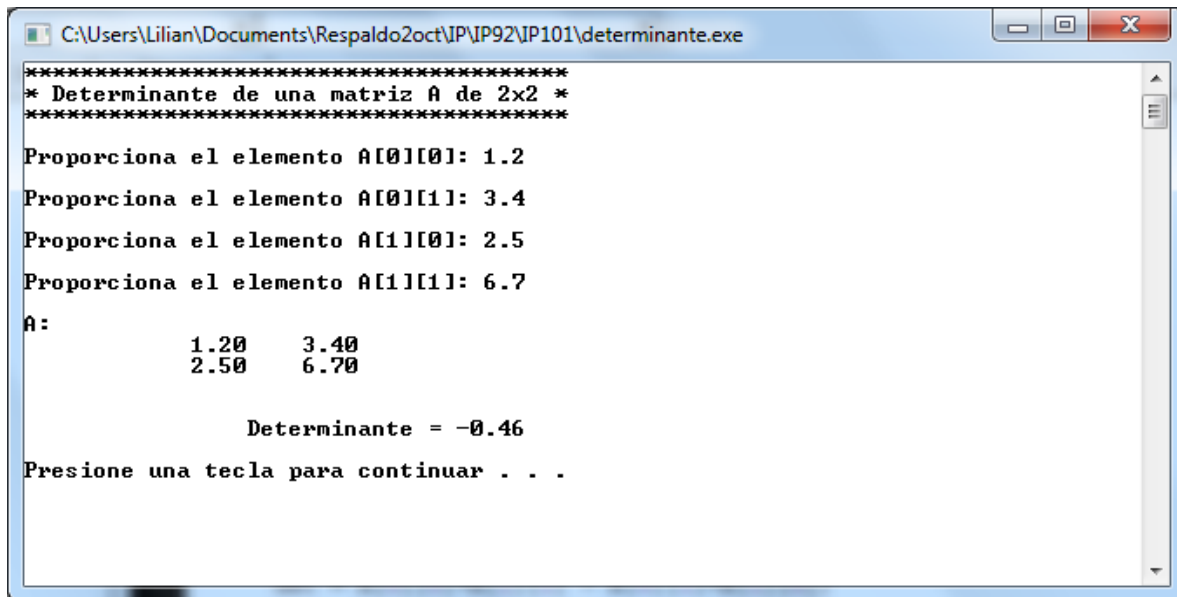
    /* Impresión de la matriz A */
    for(i=0; i<TAM; i++){
        for(j=0; j<TAM; j++){
            printf("%8.2f", A[i][j]);
            printf("\n\t");
        }

        printf("\n\n\t\tDeterminante = %.2f\n\n", det);
        system("pause");
    }
}
```

**Programa 5.2:** determinantes.c



En la siguiente figura se muestra una ejecución del programa.



```
C:\Users\Lilian\Documents\Respaldo2oct\IP\IP92\IP101\determinante.exe
*****
* Determinante de una matriz A de 2x2 *
*****
Proporciona el elemento A[0][0]: 1.2
Proporciona el elemento A[0][1]: 3.4
Proporciona el elemento A[1][0]: 2.5
Proporciona el elemento A[1][1]: 6.7
A:
      1.20    3.40
      2.50    6.70

      Determinante = -0.46
Presione una tecla para continuar . . .
```

Ejecución del programa determinante.c

### 5.1.5. Cadenas

Una *cadena* es una serie de caracteres tratados como una sola unidad. Una cadena puede incluir letras, dígitos y varios caracteres especiales(Deitel H. M., Deitel P. J., 1995).

En algunos lenguajes de programación existe un tipo de dato específico para definir a las cadenas, sin embargo, en lenguaje C las cadenas se implementan por medio de arreglos de tipo carácter ya que no existe un tipo específico para ellas, pero existe todo un conjunto de funciones estándar definidas en la biblioteca string.h mediante las cuales es posible realizar las operaciones más comunes, por ejemplo: copiarlas, concatenarlas, compararlas, entre otras. Además es posible imprimirlas y leerlas de forma similar que un dato simple.

En lenguaje C toda constaten de tipo cadena se indica entre comillas dobles, por ejemplo:

“Calle 2 #135”

Una cadena en C termina siempre con el *carácter nulo* ‘\0’ (cuyo valor ascii es cero) que representa el fin de cadena.



Al declarar arreglos de tipo char que sean una cadena se pueden inicializar directamente con una constante cadena de la siguiente forma:

```
char cad[50]="saludo";
```

Al inicializar una variable cadena de esta manera, se agrega automáticamente el símbolo de carácter nulo para indicar el fin de cadena, es decir, en la posición 6 del arreglo cad se encuentra almacenado el fin de cadena '\0'. De forma general, es importante señalar que en un arreglo de tamaño N es posible almacenar correctamente una cadena de máximo N-1 caracteres. De tal forma que en el arreglo cad se puede almacenar una cadena de máximo 49 caracteres.

Las cadenas en C pueden ser asignadas a un arreglo de tipo char utilizando la función scanf mediante el especificador de formato %s, por ejemplo la línea de código:

```
scanf("%s",cad);
```

De esta manera se lee desde el teclado una cadena y se guarda en el arreglo cad, sólo que en este caso *no se incluye el operador &* antes del nombre del arreglo, pues el identificador del arreglo almacena la dirección del primer elemento del mismo.

La función gets() también nos permite leer del teclado una cadena y asignarla a un arreglo de tipo char, por ejemplo la instrucción:

```
gets(cad);
```

Lee una cadena desde el teclado y la almacena en el arreglo cad.

Una diferencia importante entre usar scanf y gets es que con la primera la lectura de la cadena se da por terminada cuando el usuario presiona [Enter] o Espacio, mientras que la segunda termina la lectura de la cadena únicamente cuando el usuario presiona [Enter], tal como se muestra en los siguientes ejemplos:

| Cadena ingresada | Instrucción          | Contenido del arreglo cad[]  |   |   |   |   |     |   |   |     |   |     |   |   |     |   |   |   |  |   |  |  |  |  |  |  |  |   |  |   |
|------------------|----------------------|--|---|---|---|---|-----|---|---|-----|---|-----|---|---|-----|---|---|---|--|---|--|--|--|--|--|--|--|---|--|---|
| “Calle 2 #135”   | scanf(“%s”,c<br>ad); | cad[]:<br><table><tr><td>C</td><td>a</td><td>l</td><td>l</td><td>e</td><td>/</td><td>/</td><td>...</td><td>/</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td>0</td></tr></table>  | C | a | l | l | e   | / | / | ... | / |     |   |   |     |   | 0 | 0 |  | 0 |  |  |  |  |  |  |  |   |  |   |
| C                | a                    | l  | l | e | / | / | ... | / |   |     |   |     |   |   |     |   |   |   |  |   |  |  |  |  |  |  |  |   |  |   |
|                  |                      |  |   |   | 0 | 0 |     | 0 |   |     |   |     |   |   |     |   |   |   |  |   |  |  |  |  |  |  |  |   |  |   |
| “Calle 2 #135”   | gets(cad);           | cad[]:<br><table><tr><td>C</td><td>a</td><td>l</td><td>l</td><td>e</td><td></td><td>2</td><td>#</td><td>1</td><td>3</td><td>5</td><td>/</td><td>...</td><td>/</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td></td><td>0</td></tr></table> | C | a | l | l | e   |   | 2 | #   | 1 | 3   | 5 | / | ... | / |   |   |  |   |  |  |  |  |  |  |  | 0 |  | 0 |
| C                | a                    | l  | l | e |   | 2 | #   | 1 | 3 | 5   | / | ... | / |   |     |   |   |   |  |   |  |  |  |  |  |  |  |   |  |   |
|                  |                      |  |   |   |   |   |     |   |   |     | 0 |     | 0 |   |     |   |   |   |  |   |  |  |  |  |  |  |  |   |  |   |

Similarmente, para imprimir en pantalla una cadena se puede utilizar la función printf con el especificador de formato %s, o bien, la función puts, nuevamente ambas funciones



tienen un comportamiento similar con la única diferencia de que puts incluye siempre un salto de línea al final, esto se ilustra a continuación.

| Código c   | Ejecución |
|--|-----------|
| <b>Impresión de cadena con printf</b>  |           |
| <pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; main(){ char mensaje[30]="Mar profundo "; printf("%s",mensaje); system("pause"); }</pre> |           |
| <b>Impresión de cadena con puts</b>  |           |
| <pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; main(){ char mensaje[30]="Mar profundo "; puts(mensaje); system("pause"); }</pre>        |           |

Las funciones que nos permiten el manejo de cadenas se encuentran en la biblioteca estándar string.h, para ilustrar algunas se muestra el siguiente programa en C.

**Ejemplo 5.3:** El programa siguiente verifica si una clave (password) de 8 caracteres alfanuméricos ingresado por el usuario es correcta.

---

```
/*Directivas de preprocesador*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h> /* se incluye la biblioteca de cadenas */

main( )
{
    /* Declaración de variables */
    char pwscorrecto[9]="jk278la0"; /* Clave correcta */
    char pwsingresado[9]; /* para leer la clave que
```

---





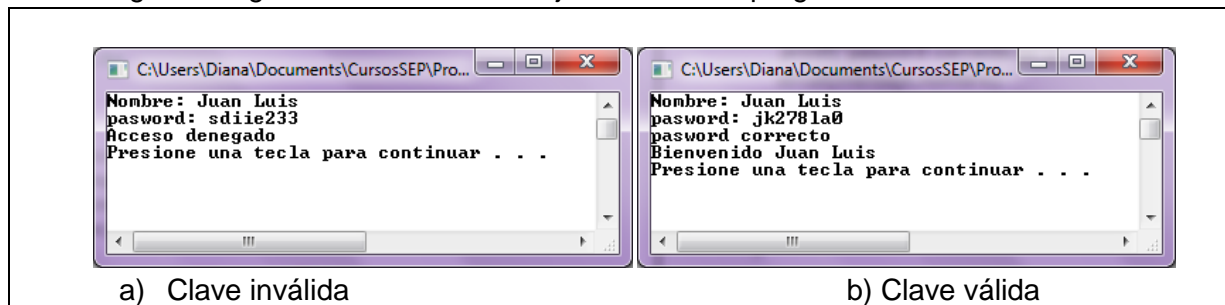
```
        ingrese el usuario */
    char nombre[10]; /* Para leer el nombre del usuario */
    char mensaje[50]="Bienvenido ";
    /* Lectura de datos */
    printf("Nombre: ");
    gets(nombre); /* Lectura de una cadena con espacios */
    printf("password: ");
    scanf("%s",pwsingresado); /* Lectura de una cadena sin
                               espacios*/

    if (!strcmp(pwscorrecto,pwsingresado)){ /* comparación
        de claves, si la función strcmp regresa 0 son i
        iguales */
        printf("password correcto \n");
        strcat(mensaje,nombre); /* pega al final de
            mensaje el nombre del usuario*/
        puts(mensaje); /* impresión de la cadena con salto
            de línea*/
    }
    else {
        strcpy(mensaje, "Acceso denegado"); /* copia la
            cadena acceso denegado al mensaje */
        puts(mensaje); /* imprime la cadena*/
    }

    system("pause");
}
```

Programa 5.3: password.c

En la siguiente figura se muestran dos ejecuciones del programa



Ejecución del programa password.c (Fuente: elaboración propia utilizando DevC++)



Con esto se concluye la sección de arreglos, recuerda que un arreglo es un conjunto de dato del mismo tipo. En la siguiente sección verás cómo puedes agrupar datos de diferente tipo.

### 5.2. Estructuras

Las estructuras en C al igual que los arreglos nos permiten tratar a un conjunto de datos bajo un mismo identificador pero a diferencia de los arreglos las estructuras son conjuntos de datos contiguos en memoria *homogéneos* de tal forma que una estructura puede estar formada por datos de diferentes tipos.

Una de las aplicaciones para las estructuras es para la manipulación de registros que se recuperan o se almacenan en una base de datos.

### 5.2. Definición, declaración e inicialización

“Una *estructura* es una colección de una o más variables, de tipos posiblemente diferentes, agrupadas bajo un nombre para manejo conveniente (en algunos lenguajes también se conocen como registros). Las estructuras permiten tratar a un grupo de variables relacionadas como una unidad, en vez de que se traten en forma separada (Kernighan & Ritchie, 1991, pág. 141)”.

La definición de una estructura en C inicia con la palabra reservada **struct** seguida del identificador para el tipo de estructura y de un bloque de definiciones de variable que constituyen el conjunto de elementos que forman parte de ella, la sintaxis para definir la estructura es la siguiente:

```
struct<identificadorEstructura>{  
    <tipo1><identificadorE1>;  
    <tipo2><identificadorE2>;  
    ...  
    <tipoN><identificadorEN>;  
}
```

Observa que se utiliza la palabra reservada **struct** para iniciar la definición y que las declaraciones para los elementos de la estructura se encierran entre llaves. Una estructura puede contener a N elementos de diferentes tipos, de cualquiera de los tipos básicos, o incluso un arreglo, veamos un ejemplo:

```
struct paciente {
```



```
int nss; /* número de seguro social */
char apellido[50];
char nombre[20];
int edad;
float estatura;
char sexo;
}
```

En este ejemplo se está definiendo la estructura paciente que tiene seis elementos: dos enteros (nss y edad), dos cadenas (apellido y nombre), un flotante (estatura) y un carácter (sexo). Sin embargo la definición anterior no reserva espacio en memoria para la estructura, más bien define un tipo de dato, por lo tanto para poder utilizar la estructura, es necesario declarar una variable de este tipo, es aquí cuando se reserva espacio en memoria. La sintaxis para hacer esta declaración es la siguiente:

**struct**<identificadorEstructura><identificador\_var>;

Por ejemplo la declaración:

```
struct paciente paciente1, paciente2;
```

Declara a las variables paciente1 y paciente2, las cuales son del tipo paciente y por tanto para cada una de ellas se reserva espacio en memoria suficiente para cada uno de sus seis elementos.

Otra forma válida de hacer la declaración es haciéndola seguida a la definición de la estructura, para el ejemplo anterior puede escribirse como sigue:

```
struct paciente {
    int nss;
    char apellido[50];
    char nombre[20];
    int edad;
    float estatura;
    char sexo;
} paciente1, paciente2;
```

En este caso el identificador para el tipo de estructura puede omitirse, pero entonces la única forma de declarar variables para ese tipo de estructura es en su definición, y no en una declaración por separado, de tal forma que nuestro ejemplo puede quedar como sigue:



```
struct {  
    int nss;  
    char apellido[50];  
    char nombre[20];  
    int edad;  
    float estatura;  
    char sexo;  
} paciente1, paciente2;
```

Por otro lado, al igual que los arreglos, también se pueden inicializar los elementos de una estructura en el momento de la declaración de una variable del tipo de la estructura en cuestión, éstos deben estar encerrados entre llaves y separados por comas. La sintaxis general es la siguiente:

```
struct<identificadorEstructura><identificador_var> =  
    { <valorE1>, <valorE2>, ..., <valorEN> };
```

Por ejemplo :

```
struct paciente paciente1 = {1240, "PicaPiedra", "Pedro", 45, 1.80, 'M'};
```

Sólo en el momento de la declaración es posible asignar todos los valores de una estructura (al igual que con los arreglos), así que si después se quiere modificar tendrán que hacerse las modificaciones de forma separada en cada uno de sus elementos, como se muestra en el siguiente subtema.

### 5.2.2. Acceso a sus elementos

Para referenciar un elemento de la estructura se utiliza el operador punto "." con la siguiente sintaxis:

```
<identificador_var>.<identificadorEi>
```

Este operador seguido del identificador del elemento, referencia el elemento indicado, de tal forma que la sentencia para asignar al paciente1 un nss de 23442145 será:

```
paciente1.nss = 23442145;
```

Ahora, si se quiere asignar a la misma variable el nombre "Pablo", la instrucción sería:

```
paciente1.nombre = "Pablo";
```



Del mismo modo se realiza la impresión o lectura del elemento de la estructura, sin perder de vista su tipo. Para ilustrar esto se propone el siguiente ejemplo.

**Ejemplo 5.4:** En el siguiente programa se declara una estructura de tipo perro, que tiene los siguientes elementos:

| Elemento | Tipo   |
|----------|--------|
| Raza     | char[] |
| Edad     | int    |
| Peso     | float  |

Posteriormente se declaran dos variables de éste tipo, una se inicializa en la declaración y a la otra se le asignan valores desde el teclado, ambos se muestran al final en pantalla.

---

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

main(){
/* Declaración de la estructura perro*/
struct perro{
char raza[20];
int edad;
float peso;
} fido, pluto = {"labrador", 7, 20}; /* inicialización de la
                                     variable pluto*/

printf("*****");
printf("\n* Comparando perros          *");
printf("\n*****");

printf("\n\nIngresa la raza de fido:");
scanf("%s",&fido.raza); /* lectura de un elemento */

printf("Ingresa la edad de fido en años: ", 164);
scanf("%d",&fido.edad); /* lectura de un elemento */

printf("Ingresa el peso de fido en kilos de fido:");
scanf("%f",&fido.peso); /* lectura de un elemento */

/* impresión de los elementos de las estructuras */
printf("\nFido es de raza %s, tiene %d años y pesa %.2f
kilos\n",fido.raza,fido.edad,164,fido.peso);
printf("\nPluto es de raza %s, tiene %d años y pesa %.2f
```

---





---

```
kilos\n",pluto.raza,pluto.edad,164,pluto.peso);
```

```
/* comparación de los nombres que son cadenas */
```

```
if(!strcmp(pluto.raza,fido.raza))
printf("\nPluto Y Fido son de la misma raza \n");
else
printf("\nFido y Pluto son de razas distintas\n");
```

```
/* comparación de elementos de tipo numérico */
```

```
if(pluto.peso > fido.peso)
printf("Pluto es m%cs pesado que Fido\n",160);
else if(pluto.peso < fido.peso)
printf("Fido es m%cs pesado que Pluto\n",160);
else
printf("Fido y Pluto pesan lo mismo\n");
```

```
if(pluto.edad > fido.edad)
printf("Pluto es mas viejo que Fido\n");
else if(pluto.edad < fido.edad)
printf("Fido es mas pesado que Pluto\n");
else
printf("Fido y Pluto tienen la misma edad \n");
```

```
getch();
}
```

---

#### 5.4: perros.c

Nota: Observa que en este caso, para poder imprimir la letra ñ se utilizó su código Ascii (164) para lo cual en la cadena de control del printf se escribió %c en donde se desea imprimir la ñ. Este mismo truco se hizo para acentuar la letra a.

En la siguiente figura se muestra una ejecución del programa anterior.



5.5. Ejecución del programa perros.c

Para concluir con este capítulo veamos el siguiente ejemplo donde se utilizan estructuras y arreglos:

**Ejemplo 5.5:** Se requiere un programa que permita registrar los datos de los perros que ingresan a refugio para perros, y poder desplegar los datos de alguno de los perros, a cada perro se le asigna una clave numérica consecutiva. Los datos que se registran del perro son:

- la fecha de ingreso (cadena)
- nombre (cadena)
- raza (cadena)
- color (cadena)
- edad (entero)
- peso (flotante)

El refugio tiene capacidad máxima para 100 perros.

Para la solución del problema se puede plantear un menú que permita registrar a un perro, o bien, desplegar los datos del perro solicitados. En este caso tenemos como datos de entrada la opción del menú que elija el usuario. En el caso de que sea un registro tendremos también se tienen como datos de entrada los datos del perro. Para la opción de despliegue se tendrá que dar como datos de entrada la clave del perro y la salida serán los datos correspondientes a la clave.

Para desplegar y repetir el menú se requiere un ciclo y una estructura condicional que maneje las opciones del menú. Para almacenar los datos del perro, se puede utilizar una estructura similar al ejemplo anterior. Mediante un arreglo de estas estructuras estaremos en capacidad para manipular los datos de varios perros, el tamaño de este arreglo tendrá



que ser igual a la capacidad del refugio (100 perros), de tal forma que el índice del arreglo que toca a cada perro corresponderá con su clave.

Una restricción que hay que tomar en cuenta es que se debe verificar que no se sobre pase la capacidad de los 100 peros, tanto al ingresar datos como al recuperarlos, para ello se llevará un contador (c) que actualice el número de perros ingresados.

```
Inicio
c ← 0
Hacer
  Imprimir "Refugio para perros -Ladrado Feliz- "
  Imprimir "1) Registrar un perro "
  Imprimir "2) Buscar un perro "
  Imprimir "3) Salir "
  Imprimir "Elige una opción: "
  Leer op
  Casos para op
    Caso 1: Si c ≥ 100 entonces
      Imprimir "El refugio está lleno"
    Si no
      Imprimir "Ingresa los datos del perro:"
      Imprimir "Clave:" c
      Imprimir "fecha de ingreso[dd/mm/aa]: "
      Leer perros[c].fecha
      Imprimir "color: "
      Leer perros[c].color
      Imprimir "nombre: "
      Leer perros[c].nombre
      Imprimir "raza: "
      Leer perros[c].raza
      Imprimir "edad: "
      Leer perros[c].edad
      Imprimir "peso: "
      Leer perros[c].peso
      c ← c+1
    Fin si-si no
  Caso2: Imprimir "Clave: "
  Leer clave
  Mientras clave ≥ 100 v clave < 0 hacer
    Imprimir "La calve no es válida, ingresa nuevamente la clave:"
    Leer clave
  Fin mientras
```



```
Imprimir "nombre:", perros[clave].nombre
Imprimir "fecha de ingreso:", perros[clave].fecha
Imprimir "color: ", perros[clave].color
Imprimir "raza: ", perros[clave].raza
Imprimir "edad: ", perros[clave].edad
Imprimir "peso: ", perros[clave].peso
```

Caso 3:

Otro caso: Imprimir "Opción no válida"

Fin casos

Mientras op≠3

Fin Hacer-mientras

Fin

**Algoritmo 5.3.** Registro perros

La prueba de escritorio se deja como ejercicio al lector. En cuanto a la codificación del algoritmo se muestra a continuación.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

main(){
/* Declaración del arreglo de tipo estructura perro */
struct perro{
char fecha[10];
char raza[30];
char color[50];
char nombre[30];
int edad;
float peso;
} perros[100];

int c=0, op, clave;

do{ /* Inicio del ciclo que imprime el menú*/
printf( "\n-----\n");
printf( "\nRefugio para perros -Ladrado Feliz- \n" );
printf( "\n-----\n");
printf( "1) Registrar un perro \n" );
printf( "2) Buscar un perro \n" );
printf( "3) Salir \n" );
printf( "Elige una opci%cn:",162 );
```



```
scanf("%d",&op);

switch (op){
case 1: /*Opción Registrar perro */
    printf( "\n-----\n");
    if(c>=100) /* Verifica si hay espacio */
        printf("El refugio esta lleno\n");
    else{
        /*Si hay espacio pide los datos del perro y
        Y los guarda en el registro c del arreglo */
        printf( "Ingresa los datos del perro:");
        printf( "Clave:%.3d\n", c);
        printf( "fecha de ingreso[dd/mm/aa]: ");
        scanf( "%s", perros[c].fecha);
        printf( "nombre: ");
        fflush(stdin);
        gets( perros[c].nombre);
        printf( "color: ");
        gets( perros[c].color);
        printf( "raza: ");
        gets( perros[c].raza);
        printf( "edad: ");
        scanf("%d" ,&perros[c].edad);
        printf( "peso: ");
        scanf("%f" ,&perros[c].peso);
        c++;
    }
    break;
case 2: /* Opción buscar perro */
    printf( "\n-----\n");
    printf( "Clave: ");
    scanf("%d",&clave);
    /* verifica que la clave sea válida */
    while(clave>=100 || clave <0){
        printf("La calve no es válida, ingresa
        nuevamente la clave:");
        scanf("%d",&clave);
    }
    /* Imprime los datos del perro correspondiente
    a la clave */
    printf("nombre:%s\n",perros[clave].nombre);
```





---

```
printf( "fecha de ingreso: %s\n",
        perros[clave].fecha);
printf( "color: %s\n", perros[clave].color);
printf( "raza: %s\n", perros[clave].raza);
printf( "edad: %d a%cos\n",
        perros[clave].edad,164);
printf( "peso: %.2f kilos\n",
        perros[clave].peso);
break;
case 3: /* Caso salir, no hace nada */
    break;
default: /* Caso opción inválida */
printf( "Opcion no valida\n");
}
}while (op!=3); /* El ciclo do-while se repite mientras la
                opción no sea salir (3) */
}
```

---

Programa 5.5: registroPerros.c

### Cierre de la Unidad

¡Felicidades!, aquí concluye el estudio de la Unidad 5. **Estructuras de datos**. Esperamos que los contenidos y actividades a lo largo de la unidad hayan permitido que utilices las estructuras de datos para almacenar y manipular los datos de un programa por medio del desarrollo de programas en lenguaje C. Si requieres profundizar o resolver alguna duda sobre el tema no dudes en consultar a tu docente en línea.



### Fuentes de consulta

- Cairo Osval
- do, Guardati Buemo Silvia. (1993). *Estructura de Datos*. México: McGraw-Hill.
- Deitel H. M., Deitel P. J. (1995). *Cómo programar en C/C++*. México: Prentice Hall.
- Joyanes, L., & Zohanero, I. (2005). *Programación en C. Metodología, algoritmos y estructuras de datos*. aspaño: Mc Graw Hill.
- Kernighan, B., & Ritchie, D. (1991). *El lenguaje de programación C*. México: Prentice-Hall Hispanoamericana.
- López, L. (2005). *Programación estructurada en lenguaje C*. México: Alfaomega.