ESCUELA POLITÉCNICA SUPERIOR

INFORMÁTICA – CURSO 2023-24

PRÁCTICA 4: VECTORES, MATRICES Y CADENAS

HASTA AHORA...

En prácticas anteriores se ha aprendido:

- La estructura principal de un programa en C: la función `main` y las librerías.
- Variables y Constantes: con sus respectivos tipos, usos y funcionamiento.
- Funciones `printf/scanf`: muestra y obtención de información por pantalla/teclado.
- Condicionales `if/else`: ejecución de un programa en 'árbol'.
- Programación iterativa utilizando las instrucciones `while y for`.

OBJETIVOS

En esta práctica trabajaremos con conjuntos de variables agrupadas bajo un mismo nombre: *vectores*.

La idea es evitar crear multitud de variables del mismo tipo, y crear solo un elemento que agrupe todas las variables del mismo tipo.

PRERREQUISITOS

Para el correcto desarrollo de esta práctica, el alumno ANTES DE LA SESIÓN DE PRÁCTICA EN EL LABORATORIO, debe leer con atención y comprender el contenido de la parte teórica de dicha práctica, el apartado VECTORES de este documento.

Se aconseja el uso del material de apoyo de la asignatura. Así, para esta práctica sobre vectores el alumno puede consultar la siguiente bibliografía:

- Rodríguez Jódar, M.A. y otros, "Fundamentos de informática para ingeniería industrial":
  - Del *Capítulo 4, Fundamentos de programación*: apartado 4.6.1, *"Vector o tabla unidimensional"*[1] *y apartado 4.6.2 "Tabla bidimensional o matriz"*.

1.- VECTORES

Descripción

En ocasiones, tenemos que almacenar en nuestros programas un conjunto grande de valores, todos del mismo tipo, por lo que pude resultar tedioso y problemático crear tantas variables como necesitemos, sobre todo si ellas tienen alguna relación entre sí. Para esta tarea, podemos hacer uso de los vectores (*array* en inglés), un tipo de datos compuesto que nos va a permitir guardar un conjunto de valores de un mismo tipo de datos formando un conjunto indivisible.

Vectores (array).: conjunto de elementos del **mismo tipo** identificadas con el **mismo nombre** que se almacena en posiciones consecutivas y que pueden ser referenciados individualmente por un índice.

[1] L*os párrafos dedicados a cadenas de caracteres no son necesarios para esta práctica*

A. Declaración de vectores

A la hora de crear un vector, todos los elementos que lo forman han de ser de igual tipo. De esa forma la creación de un vector se realiza de la siguiente manera:

```
tipo nombre[TAMANYO];
```

- ***tipo:*** se corresponde con el tipo de todos los elementos que forman el vector
- ***nombre:*** utilizado para acceder a los elementos del mismo.
- ***TAMANYO:*** número de elementos que posee el vector.
- *Ejemplo:* `int v[10];`

Nota: los vectores creados de esta forma poseen un número FIJO de elementos, de manera que hay que especificar un número constante a la hora de crearlo (no puede ser una variable), y dicho tamaño NO puede ser modificado… aunque puede utilizarse un tamaño inferior si se contabiliza de forma independiente (haciendo uso de algún contador).

B. Inicialización de los valores en la creación del vector

Si a la hora de crear el vector queremos aportar unos valores iniciales a sus elementos, podríamos hacerlo de la siguiente manera:

```
tipo nombre[N] = {valor0, valor1, valor2, … , valorN-1};
```

Ejemplo:
```
int v[10] = {8,2,3,4,1,0,3,9,2,7};
```

C. Acceso a los elementos de un vector

Para acceder a los elementos de un vector hay que hacer uso del nombre con el que se ha creado e indicar, mediante un índice, a qué elemento queremos acceder.

De esta forma, para acceder al elemento i-ésimo del vector creado anteriormente, se haría:

```
v[i]
```

Nota: los elementos de un vector se identifican mediante la posición que ocupan dentro de él. Esta posición abarca valores entre el 0 y el N-1 (ambos inclusive), si hemos definido el vector con un tamaño de N (¡¡¡cuidado!!! ¡¡¡La posición N no existe!!!)

Se puede hacer uso de estos accesos para leer o escribir en una posición concreta del vector. Una vez que se acceda a dicha posición, hay que tratar el elemento como una variable del tipo base a todos los efectos.

Ejemplos:
```
v[0] = 23;
a = v[8] + v[5];
printf("El primer elemento del vector es %d\n", v[0]);
```

D. Recorrido de vectores

Se ha hablado del uso de los vectores como almacenes indivisibles de variables del mismo tipo, pero ¿tienen alguna otra ventaja?

Al encontrarse almacenados en posiciones consecutivas y hacer uso de un índice numérico de tipo entero para acceder a cada uno de los elementos, si queremos realizar la misma operación sobre todos los elementos del vector, podríamos utilizar variables enteras para los índices y recorrer los elementos del vector con bucles que actualicen la variable que actúa de índice. Veamos, a continuación, algunos ejemplos.

a) Inicialización de un vector de 100 posiciones:

Versión sin buble: Inicialización específica de cada elemento del vector. (Requiere 100 instrucciones)	Versión con bucle: Inicialización mediante un bucle que modifica 'i' que actúa de índice en el vector. (Requiere sólo 2 instrucciones)

`int v[100];`
`v[0] = 0;`
`v[1] = 0;`
`v[2] = 0;`
`v[3] = 0;`
`…`
`v[98] = 0;`
`v[99] = 0;`
	`int i, v[100];`
	`for (i=0 ; i<10 ; i++)`
	`{`
	`  v[i]=0;`
	`}`

b) Ejemplo de utilización de un vector que almacena el valor de un conjunto de sendores:

```
#include <stdio.h>
main() {
     int i;
     float sensores[100];
     printf("Valores erróneos en los sensores:\n");
     for(i=0;i<100;i++)
     {
          if(sensores[i]<0 || sensores[i]>1000)
          {
               printf("sensor %d => %f\n", i, sensores[i]);
          }
     }
}
```

Como puede comprobarse al ejecutar el código, algunos valores de sensores que se muestran por pantalla son extraños. Esto ocurre porque, al igual que en las variables simples, las posiciones de memoria en la que se ubican las variables pueden contener valores desconocidos (denominados basura) lo que exige inicializar las variables antes de usarlas.
A continuación, se muestra el mismo ejemplo, pero con la inicialización de sus variables:

```
#include <stdio.h>
main()
{
     int i;
     float sensores[100];
     for(i=0;i<100;i++)
     {
          printf("Introduzca valor de posicion %d:  ", i);
          scanf("%f", &sensores[i]);
     }
     printf("Valores erróneos en los sensores:\n");
     for(i=0;i<100;i++)
     {
          if(sensores[i]<0 || sensores[i]>1000)
          {
                    printf("sensor %d => %d\n", i, sensores[i]);
          }
     }
}
```

EJERCICIOS DE VECTORES:

Ejercicio V1: Analice el siguiente programa e indique, sin utilizar ZinjaI, lo que se mostrará por pantalla:

```
int main()
{   int v[10],i;
    for (i=0; i<10; i++)
    {
        v[i] = i*2;
    }
    for (i=9; i>=0; i--)
    {
        printf("%d ", v[i]);
    }
    return 0;
}
```

Ejercicio V2: Analice el siguiente código, sin utilizar ZinjaI, y especifique qué es lo que calcula:

```
int main()
{   int i, cnt=0,  v[10]={12,23,14,45,56,47,78,69,90,100};
    for (i=1; i<10; i++)
    {
        if (v[i]<v[i-1])
        {
           cnt++;
        }
    }
    printf("%d ", v[i]);
    return 0;
}
¿Por qué el bucle comienza en i=1 en vez de i=0?
```

Ejercicio V3: Utilizando el siguiente programa como base, donde se declara un vector de tamaño 10 y se recogen por teclado cada uno de sus elementos, añada el código necesario para que realice las siguientes tareas:

```
int main()
{   int i, v[10];
    for (i=0; i<10; i++)
    {
        scanf("%d", &v[i]);
    }
    // Nuevas tareas...
    return 0;
}
```

a) Mostrar todos los elementos del vector separados por comas.
b) Mostrar cuántos elementos son positivos y cuántos negativos.
c) Mostrar el valor máximo y mínimo de los elementos del vector.
d) Calcular el sumatorio y mostrarlo por pantalla.

Ejercicio V4: Se pretende gestionar las calificaciones de los alumnos de una determinada clase. Para ello, se pide una serie de utilidades que irán completando, poco a poco, el problema:

- **a)** Declare un vector de 10 posiciones de tipo *float* (a este vector lo llamaremos "clase"). Este vector almacenará las notas finales de cada alumno en dicha clase, de tal forma que un alumno vendrá identificado numéricamente por la posición que ocupa en el vector.
- **b)** Pida que se introduzcan las notas teóricas y prácticas de cada alumno y almacene la nota final realizando un porcentaje del 40% de la nota teórica y del 60% de la nota práctica.
- **c)** Muestre el porcentaje de alumnos suspensos (nota<5), el porcentaje de aprobados (5≥nota<7), el porcentaje de notables (7≥nota<9) y el porcentaje de sobresalientes (nota≥9).
- **d)** Indique la posición del vector que contienen la mejor nota.
- **e)** Indique la posición del vector que contienen la peor nota.

Ejercicio V5: Realice un programa que contabilice el número de apariciones de un determinado valor dentro de un vector. Para ello, realice los siguientes pasos:

- **a)** Defina 2 variables. Una de tipo entero cuyo valor se indicará por teclado y otra que será un vector de 5 elementos de tipo entero inicializado estáticamente (esto es, sus valores se inicializan en la misma declaración del vector) con valores aleatorios.
- **b)** Realice un recorrido por el vector para que contabilice el número de veces que se repite en el vector el valor introducido por teclado y muestre el resultado por pantalla.

Ejercicio V6: Sin hacer uso del ordenador, analice el siguiente código e indique qué se mostraría por pantalla al finalizar la ejecución:

```
int i, v[10];
for(i=0; i<10; i++)
{
    if(i%2 == 0 && i < 5)
    {
        v[i]=i*2 + 1;
    }
    else if (i%2 != 0)
    {
        v[i] = 2 + i;
    }
    else
    {
        v[i] = 0;
    }
    printf("%d,", v[i]);
}
```

Ejercicio V7: Partiendo del siguiente programa, realice los siguientes apartados:

```
int main()
{   int i, m;
    float v[10]={12.3, -23.4, 34.5, 4.56,-5.67, 6.78, 7.89,    0, -9.01, 0.12};
    float w[11]={-3.21, 43.1, 5.43,-6.54,    0, 8.76, 9.87, 3.73,     0, 2.10};
    for (i=0; i<10; i++)
    {
        if (v[i]<=0)
        {
           v[i] = -v[i];
        }
        if (v[i]>w[i])
        {
            m = v[i];
            v[i]=w[i];
            w[i]=m;
        }
        printf("%f->%f ", v[i], w[i]);
        // Nuevas tareas...
    }
    return 0;
}
```

a) Analice el código e indique las operaciones que realiza con los vectores.
b) Calcule y muestre la media aritmética de los elementos del vector *v* hasta encontrar un elemento cuyo valor sea 0 (el elemento con valor 0 y los elementos posteriores no formarán parte de la media).
c) Calcule y muestre la media aritmética de los elementos del vector *w* que no sean 0.
d) Actualice cada elemento del vector v con la suma de todos los elementos que están en posiciones anteriores a éste incluyendo al propio elemento. Siendo i el elemento a actualizar, podría expresarse como:

$$v[i] = \sum_{n=0}^{i} v[n]$$

Ej. Si el vector fuese v={3, 2, 4, 1, 5 } el resultado sería v={3, 5, 9, 10, 15 }

Ejercicio V8: Desarrolle un programa que copie los elementos de un vector en otro del mismo tamaño, para ello, será necesario realizar la copia elemento a elemento.

Ejercicio V9: Realice un programa que copie los elementos de un vector en otro del mismo tamaño, pero en orden inverso. Para ello deberá declarar 2 vectores del mismo tamaño e inicializar el que va a actuar como vector origen. Finalmente, mostrará el resultado por pantalla.

Ej. origen = {1,2,3,4,5} → destino = { 5,4,3,2,1}

Nota: puede utilizar alguna regla matemática para calcular los índices o varios contadores.

Ejercicio V10: Complete el programa para que rellene un vector con los primeros 40 números de la sucesión de Fibonacci. En dicha sucesión, el primer número es el 0 y el segundo es el 1. El resto de números se obtiene de manera recurrente mediante la suma de los 2 anteriores. De esta forma, el comienzo de la sucesión es el siguiente: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…

Finalmente, muestre por pantalla los elementos del vector para comprobar que el comienzo de la serie se corresponde con el que se ha indicado.

```
int main()
{   int i, fibo[40];
    fibo[0]=___;
    fibo[1]=___;
    for (i=____; i<40; i++)
    {
        fibo[i]=fibo[____] + fibo[____];
    }
    // Probar
    for (i=0; i<40; i++)
    {
        printf("%d ", fibo_____);
    }
    return 0;
}
```

Ejercicio V11: Realice un programa que verifique si un número es capicúa. Supongamos que el número en cuestión está almacenado en un vector de enteros de forma que cada elemento del vector contiene una cifra del número.

Ejemplo: Para comprobar si el número 1331 es capicúa, inicialice un vector de 4 posiciones con los valores:

1	3	3	1

Ejercicio V12: Teniendo un número almacenado de igual forma que en el ejercicio anterior, realice un programa que verifique si dicho número es múltiplo de 3 aplicando la propiedad de que un número es múltiplo de 3 si la suma de todas sus cifras es múltiplo de 3.

Ejercicio V13: Realice un programa que ordene de forma **ascendente** los elementos de un vector utilizando el **método de la burbuja** cuyo algoritmo se describe a continuación. Los valores del vector podrán ser inicializados estáticamente en la propia declaración o mediante valores introducidos por teclado.

Algoritmo de ordenación basado en el método de la burbuja:
Este algoritmo se basa en la comparación de los elementos 2 a 2 determinando cual es el mayor o menor según se ordene ascendente o descendentemente. En caso de que 2 elementos no estén ordenados, se intercambian sus posiciones. En cada iteración (bucle exterior) se compara un elemento con todos los que se encuentran en una posición posterior a éste realizando en cada caso los intercambios de posiciones pertinentes, de esta forma, en la primera iteración se compara el elemento de la posición 0 con los que van desde la posición 1 hasta la última, en la segunda iteración se compara el elemento de la posición 1 con los que van desde la posición 2 a la última y así sucesivamente hasta llegar al penúltimo. A continuación, se muestra en lenguaje pseudocódigo el comportamiento del algoritmo.

```
Desde i=primero hasta i=último-1
      Desde j=i+1 hasta j=último
            si vector[i] < vector[j]
                  aux = vector[i]
                  vector[i] = vector[j]
                  vector[j] = aux
            fin si
      fin desde
fin desde
```

Nota 1: las instrucciones situadas en el cuerpo de la instrucción condicional intercambian los elementos situados en las dos posiciones comparadas.
Nota 2: Este algoritmo realiza una ordenación ascendente. Para realizar una ordenación descendente, es necesario modificar la condición del 'si' para que intercambie los elementos cuando el elemento de la posición i es mayor que el elemento de la posición j..

Ejercicio V14: Modifique el programa del ejercicio anterior para que la ordenación sea descendente.

2.- MATRICES

Descripción

Hasta ahora hemos visto que los vectores permiten guardan un conjunto de valores del mismo tipo ordenados mediante un índice que determina la posición dentro del vector. Bajo esta visión, los elementos están ubicados linealmente en una sola dimensión. Pero en ocasiones, resulta interesante organizar ese conjunto de valores en 2 dimensiones representando una tabla, una distribución de valores en un plano, una matriz matemática… o incluso utilizando 3 o más dimensiones en función de la naturaleza de los datos. En programación, los vectores con varias dimensiones se denominan matriz o array multidimensional.

Según lo anterior, un conjunto de datos podría organizarse utilizando una o varias dimensiones según resulte más cómodo en función de cómo interpretemos la información que queremos almacenar.

Por ejemplo: para guardar la temperatura máxima diaria del último año, podría utilizarse un vector de 365 posiciones en el que cada posición corresponde a un día del año, pero podría ser más cómodo utilizar una matriz bidimensional de 53x7 para guardar la temperatura de los 7 días de la semana de las 53 semanas del año o una matriz de 12x31 para identificar el día del año mediante el número del mes (0 sería enero) y el día del mes.

A continuación, vamos a extender los conceptos que hemos visto en vectores a matrices (vectores dimensiones)

A. Declaración de matrices

A la hora de crear una matriz, la nomenclatura es idéntica a un vector, pero indicando dos dimensiones diferentes. Cada una de ellas deberá estar contenida entre corchetes:

```
tipo nombre[FILAS][COLUMNAS];
```

- ***tipo:*** se corresponde con el tipo de todos los elementos que forman la matriz
- ***nombre:*** utilizado para acceder a los elementos de la matriz.
- ***FILAS:*** número de elementos que conforman la primera dimensión de la matriz.
- ***COLUMNAS:*** número de elementos que conforman la segunda dimensión de la matriz.
- *Ejemplo:* `int m[3][2];`

Nota: Al igual que en vectores, el tamaño de cada dimensión es constante y tendrá que ser indicada en la declaración de la matriz.

B. Inicialización de valores en la creación

Si a la hora de crear el vector queremos aportar unos valores iniciales a sus elementos, podríamos hacerlo de la siguiente manera:

```
tipo nombre[N][M]={{valor0,0,…, valor0,M-1},…,{valorN-1,0,…, valorN-1,M-1}};
```

Ejemplo: `int m[3][2]={{8,2},{3,4},{1,0}};`

C. Acceso a los elementos de una matriz

Para acceder a los elementos de una matriz hay que hacer uso del nombre de la matriz e indicar, mediante dos índices, a qué elemento queremos acceder (fila y columna dentro de la matriz).

De esta forma, para acceder al elemento de la fila i-ésima y columna j-ésima de la matriz creada anteriormente, se haría:

```
m[i][j]
```

Nota: los elementos de una matriz se identifican por la posición que ocupan dentro de ella en ambas dimensiones. Esta posición abarca valores del 0 y el N-1 para filas y entre 0 y M-1 para columnas, al igual que sucedía con un vector, pero extrapolado a las dos dimensiones.

Se puede hacer uso de estos accesos para leer o escribir en una posición concreta. Una vez que se acceda a dicha posición, hay que tratar el elemento como una variable del tipo base a todos los efectos.

Ejemplos:
```
m[0][1] = 23;
a = m[2][0] + m[0][1];
printf("El primer elemento de la matriz es %d\n", m[0][0]);
```

D. Recorrido de matrices

La misma ventaja de la que se disponía con los vectores a la hora de acceder a ellos puede ser tenida en cuenta con las matrices. En este caso, al disponer de dos dimensiones, se necesitarán dos bucles para poder recorrer la matriz al completo: uno para filas y otro para columnas.

Uno de esos bucles deberá ser interior al otro, ¿por qué? Si se pretende hacer un recorrido completo sobre la matriz, para cada valor indicado para filas se deberá hacer un barrido completo por todas las columnas. Eso, a nivel de código, se traduce en que para cada valor del índice del bucle (dentro del bucle) se deberá hacer un recorrido sobre todos los posibles valores de columna. En la práctica, basta con hacer un doble bucle para recorrerla al completo:

```
int m[25][50];
int i, j;
for(i=0; i<25; i++)
{
      for(j=0; j<50; j++)
      {
            m[i][j]=0;
      }
}
```

EJERCICIOS DE MATRICES:

Ejercicio M1: Realice un programa que contabilice el número de veces que se repite un valor especificado por teclado en una matriz previamente inicializada. Para el desarrollo del programa, realice los siguientes pasos:

a) Declarar una matriz de M filas y N columnas siendo M y N constantes declaradas en el programa.
b) Inicializar sus elementos con valores introducidos por teclado.
c) Solicitar por teclado el valor a buscar en la matriz.
d) Contabilizar el número de apariciones de dicho valor en la matriz.

Ejercicio M2: Realice un programa que guarde en una matriz, la nota de teoría, de prácticas y final de todos los alumnos (suponga sólo 5 alumnos). Para cada alumno, solicitará la nota de teoría y prácticas, calculará la nota final mediante la media aritmética de las anteriores y guardará las 3 notas en la matriz.

Ejercicio M3: Realice un programa que inicialice 2 matrices de dimensión MxN y compruebe si ambas matrices son iguales. Ambas matrices son iguales si todos sus elementos son iguales.

Ejercicio M4: Escriba un programa que declare e inicialice en el código una matriz bidimensional a_{ik} (de dimensión $M \times N$) y un vector b_k (de dimensión N) y que calcule el vector producto de ambos c_i (de dimensión M). Los valores de las dimensiones deben definirse como constantes en el programa. Recuerde que los elementos del vector producto se calculan según la expresión:

$$c_i = \sum_{k=1}^{n} a_{ik} b_k$$

Ejercicio M5: Realice un programa que gestione las reservas de asientos de un avión en el que los asientos están distribuidos en 25 filas y 4 columnas. Mediante un menú, el programa deberá poder reservar un asiento, cancelarlo y mostrar una lista con los asientos libres. Para ello declare una matriz que represente la disponibilidad de cada asiento del avión, considerando que cada elemento con valor a 0 se interpreta como asiento libre y con valor a 1 indica que está ocupado. Para ello siga los siguientes pasos:

a) Declare la matriz e inicialícela a 0 (todos los asientos vacíos inicialmente).
b) Incluya un menú con opciones como el siguiente:

```
int opcion=1;
…
while(opcion!=0)
{
    printf("\n\n\n1.Reservar\n2.Cancelar\n3.Listar\n0.Salir\nSeleccione opción:");
    scanf("%d", &opcion);
    if (opcion==1)
    {   printf("\nReservar asiento");
        //...
    }
    else if (opcion==2)
    {   printf("\nCancelar reserva");
        //...
    }
    else if (opcion==3)
    {   printf("\nLista asientos disponibles");
        //...
    }
}
```

c) Realice las opciones reservar y cancelar. Para ello deberá solicitar la fila y el asiento, y reservar o cancelar según corresponda.
d) Finalmente realice la opción listar. Esta funcionalidad deberá recorrer todos los asientos y mostrar la (fila, columna) de todos los asientos disponibles.

3.- CADENAS DE CARACTERES

Las cadenas de caracteres se refieren al conjunto de caracteres que definen un texto, como podría ser un mensaje, un nombre, etc.

En el lenguaje C estándar, el almacenamiento de las cadenas de caracteres no utilizan un tipo de datos propio, sino que se apoyan en los **vectores** (*arrays*) de elementos **de tipo char**, de forma que cada elemento del vector represente una letra del texto que almacena.

Según estudiamos en el tema 1 de teoría, para representar los caracteres en un computador podría utilizarse la codificación ASCII, la cual codifica cada carácter (que incluyen el alfabeto más una serie de símbolos) utilizando un código de 8 bits. De esta manera, cada uno de los 256 caracteres que forma la codificación ASCII posee un código numérico de 8 bits que lo identifica.

Precisamente, las variables de tipo *char* antes mencionadas pueden almacenar un valor numérico de 8 bits, por lo que son utilizadas para almacenar el código de 8 bits de un carácter ASCII. De ahí que este tipo numérico sea denominado *char* (de *character*).

Por tanto, un valor de una variable de tipo *char* podría interpretarse tanto como un valor numérico como un carácter. Para justificar esta ambivalencia véase el siguiente ejemplo:

```
char crt = 97;
printf("El carácter ASCII %c corresponde al número %d", crt, crt);
```

Al ejecutar este ejemplo se obtiene el mensaje "El carácter ASCII **a** corresponde al número 97" puesto que el especificador de formato *%c* de *printf* interpreta *cnt* como carácter y *%d* como número entero.

Por tanto, **una cadena de caracteres es una secuencia de números de tipo *char* que se interpretan como caracteres**. Veamos un ejemplo.

Carácter	H	o	l	a		m	u	n	d	o	\0
ASCII	72	111	108	97	32	109	117	110	100	111	0

No obstante, ha de tenerse en cuenta que todo mensaje contenido en un vector de *char* debe terminar con el carácter especial '\0' y que corresponde al código 0 de la tabla ASCII que actúa como *marca de fin de texto*. No confundir con el carácter '0', cuyo código ASCII es el 48.

Este carácter especial es necesario para determinar en qué posición del vector finaliza el texto, ya que el tamaño de un vector es fijo pero el texto que contiene no tiene por qué serlo. Como ejemplo, supongamos la variable *char nombre[10]* que almacena el nombre de una persona. El contenido de esta variable podría ser:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
S	u	S	a	n	a	**\0**			
A	n	T	o	n	i	o	**\0**		
J	u	A	n		M	a	n	u	**\0**

Como puede apreciarse en el ejemplo con un vector de 10 elementos, el nombre más largo que puede almacenarse es de 9 caracteres pues hay que reservar espacio para la marca '\0', por tanto, el tamaño del vector que almacena un texto debe ser del tamaño del mensaje más largo que pudiese ser almacenado más 1 para especificar la marca '\0' de fin de texto.

o CREACIÓN E INICIALIZACIÓN

Como se ha indicado anteriormente, se utiliza un **vector de tipo char**, cuyo tamaño será de, al menos, la longitud de la cadena que queramos guardar incluyendo el carácter '\0'. Ejemplos:

```
char una_cadena[30];
char otra[]="No decimos la longitud, pero le asignamos valor";
```

Como se puede apreciar, al igual que con un vector, se puede inicializar la cadena en el momento de su definición. En este caso, se escribiría entre comillas dobles el texto que se almacenaría en la cadena de caracteres creada: cada carácter que conforma dicho texto entrecomillado se almacenaría en una posición diferente del vector de caracteres (cabe destacar que los espacios y demás signos de puntuación son caracteres en sí mismos).

o *USO DE PRINTF Y SCANF*

Al igual que sucede con todos los tipos básicos que se han utilizado hasta ahora, se puede hacer uso de las funciones *printf* y *scanf* para mostrar y almacenar el contenido de una cadena de caracteres, respectivamente. En este caso, existe un par de peculiaridades al respecto:

a) El especificador utilizado para indicar que se trabaja con una cadena de caracteres en las funciones *printf* y *scanf* es "%s", que proviene de "string" (cadena en inglés).

b) Cuando utilicemos el especificador de formato "%s" en la función *scanf*, es muy importante no añadir & delante de la variable que se especifica en el segundo parámetro, de lo contrario el comportamiento del programa podría ser impredecible.

```
char nombre[10];
scanf ("%s", &nombre);    →  scanf ("%s", nombre);
printf("tu nombre es %s", nombre);
```

o *MANIPULACIÓN DE CADENAS COMO VECTORES*

Puesto que las cadenas de caracteres son vectores de tipo *char*, todo el tratamiento estudiado a los vectores en la presente práctica es aplicable a las cadenas de caracteres.

Ejemplo:

VECTORES: modifique todas las apariciones del valor '5' por el valor '3'.	*CADENAS:* modifique todas las apariciones del carácter 'a' por el carácter 'e'.

`int i;`
`int v[10]= {1,2,3,4,5,6,7,8,9,10};`
`for(i=0; i<10; i++)`
`{`
`  if(v[i]==5)`
`  {`
`  v[i]=3;`
`  }`
`}`
	`int i;`
	`char c[10]= "hola hola";`
	`for(i=0; i<10; i++)`
	`{`
	`  if(c[i]=='a')`
	`  {`
	`  c[i]='e';`
	`  }`
	`}`

Nota: *Podemos apreciar cómo un carácter puede ser identificado o asignado haciendo uso de las comillas simples cuyo resultado sería equivalente a utilizar el valor numérico correspondiente de la tabla ASCII (* `c[i]=='a'` *es lo mismo que* `c[i]==97`*, de hecho 'a'+1=97+1=98='b'*).

EJERCICIOS DE CADENAS:

Ejercicio C1: Desarrolle un programa que muestre por pantalla la longitud de un texto introducido por teclado. Para ello deberá tener en cuenta la marca de fin de texto.

Ejercicio C2: Cree un programa que pida por teclado una serie de cadenas de caracteres hasta introducir la cadena "fin". El programa mostrará por pantalla el número de cadenas introducidas antes de finalizar.

Ejercicio C3: Realice un programa que solicite por teclado un texto y que muestre por pantalla el mismo texto en mayúsculas.

> **Nota**: para determinar la longitud del texto, utilice el código del ejercicio C1. Tenga en cuenta además que las letras del alfabeto en minúsculas están ordenadas en la tabla ASCII a partir del 97 y las letras mayúsculas a partir del 97, por lo que toda letra minúscula se convierte en mayúsculas restándole 'a'-'A'=97-65=32.

Ejercicio C4: Escriba un programa que introduzca por teclado un mensaje de texto y una letra (carácter). El programa devolverá el número de veces que aparece dicho carácter en el mensaje.

Ejercicio C5: Implemente un programa que verifique si la palabra introducida por teclado es un palíndromo. Un palíndromo es una palabra que se lee igual de izquierda a derecha, o de derecha a izquierda (versión alfabética de un número capicúa). Para determinar la longitud de texto, utilice el código desarrollado en el ejercicio C1.

> Ejemplo de palíndromos: ANA, ANILINA…

Ejercicio C6: Realice un programa que pida al usuario un texto y realice las siguientes tareas:

- **a)** Imprimir por pantalla el número de veces que aparece cada vocal en dicho texto.
- **b)** Imprimir por pantalla el número de palabras de una letra, de dos letras, de tres y de cuatro letras que aparecen en el texto.

***ANEXO*: Patrones de algoritmos**

BÚSQUEDA DE VALORES EN UN VECTOR:

1. Búsqueda de máximo (equivalente para mínimo variando la condición)

```
máximo = vector[0]
posición = 0
desde i=segundo hasta i=último
          si vector[i] > máximo
              máximo = vector[i]
              posición = i
          fin si
fin desde
```

Nota: Se supone inicialmente que el valor máximo es el que se encuentra en la primera posición del vector. A partir de ahí, haciendo un recorrido desde el segundo en adelante, se comparan uno a uno los valores de las sucesivas posiciones del vector, comprobando si son mayores al máximo que tenemos hasta ahora. En el pseudocódigo, se guarda el valor y la posición que ocupa.

2. Búsqueda simple de valor concreto

```
posición = -1
desde i=primero hasta i=último
          si vector[i] == valor
              posición = i
          fin si
fin desde
```

3. Búsqueda valor concreto con escape

```
posición = -1
i = 0
mientras i < n  Y  posición == -1
          si vector[i] == valor
              posición = i
          fin si
fin desde
```

ORDENACIÓN:

1. Algoritmo de la Burbuja

(Véase la descripción en el ejercicio V13)

```
Desde i=primero hasta i=último-1
     Desde j=i+1 hasta j=último
          sí vector[i] < vector[j]
               aux = vector[i]
               vector[i] = vector[j]
               vector[j] = aux
          fin si
     fin desde
fin desde
```