Objetivo de la práctica:

- Aprender los conocimientos necesarios sobre las estructuras de información basadas en series de datos del mismo tipo: *vectores y matrices en C/C++*.

CONCEPTOS BÁSICOS: VECTORES Y MATRICES

Un vector o array es una secuencia de objetos del mismo tipo que se almacenan de forma contigua en memoria. El tipo de objeto de objeto almacenado en el array puede ser cualquier tipo definido en C/C++.

Se adopta el convenio de usar *typedef* para crear tipos que serán los vectores y las matrices. Así:

```
typedef   int vector20[20];   //define un vector de 20 enteros
vector20  vector_mio;    //declara un vector de 20 enteros
```

El acceso a los elementos de un vector se hace mediante un índice. El índice para acceder a los elementos del vector empieza en 0 y acaba en el número de elementos menos 1.

```
Ej. vector_mio[3]=45; //asigna el valor 45 a la cuarta componente del vector
```

De forma análoga pasa con las matrices o arrays bidimensionales:

```
typedef int matriz10x4[10][4];
matriz10x4 matriz;
matriz_mia[1][3]=32;
```

Los vectores y matrices se pasan siempre por referencia como argumentos de una función. La única diferencia con los tipos simples es que no se usa el '&' por lo que simplemente se pondrá el nombre del vector o matriz.

INICIALIZAR VECTORES

Como toda variable, un vector puede tomar valores iniciales después de ser declarado. La forma general es:

tipo identificador_variable[tamaño] ={ lista_de_valores};

La lista de valores es una lista separada por comas "," de constantes que son del mismo tipo que el tipo base del vector.
Ejemplo:

```
int num[5] = {0,1,2,3,4};
```

En este ejemplo la primera constante de valor 0 la almacenará en la primera posición del vector, la segunda constante de valor 1 en la segunda posición del vector, y así sucesivamente hasta completar las cinco constantes. El compilador las situará en posiciones contiguas de memoria.

No es necesario poner el valor inicial a todo el vector, en este caso, el compilador pondrá ceros a todos aquellos elementos a los que no les hayamos asignado valor. Por ejemplo:

```
int num[5] = {0,1,2};
```

En este ejemplo, el compilador asignará ceros a los dos últimos elementos del vector.

También es posible, al poner valores iniciales al vector, no declarar el tamaño. El compilador lo determinará calculando el número de valores enumerados.

Así, la asignación:

```
int num[] = {0,1,2,3,4};
```

Hace que la dimensión de la variable *num* sea 5.

Para poner valores iniciales a los vectores multidimensionales lo haremos de una forma parecida a como lo hemos hecho en vectores unidimensionales.
Las siguientes declaraciones de asignación, son ejemplos de cómo se ponen valores iniciales a los vectores multidimensionales:

```
int tab[2][5] = {{0,1,2,3,4},{5,6,7,8,9}};
```

Es equivalente a:

```
int tab[2][5] = {0,1,2,3,4,5,6,7,8,9};
```

Y también equivale a:

```
int tab[2][5] ={
                  {0,1,2,3,4},
                  {5,6,7,8,9}
                };
```

En este ejemplo hemos declarado una matriz de enteros de dos filas y cinco columnas. De esta forma, al igual que sucedía en los vectores unidimensionales en que el valor inicial se podía poner en forma parcial, ahora también es posible con la única condición de que se debe comenzar por el principio de cada índice del vector.

No obstante, hay que ir con cuidado porque en las declaraciones incompletas las confusiones son fáciles, tal y como se muestra en el siguiente ejemplo:

```
int tabla1[2][4] = {1,2,3,4,5,6,7,8};
/* valores iniciales completos, corresponden a:
1 2 3 4
5 6 7 8
*/

int tabla2[2][4] = {1,2,3};
/* valores iniciales incompletos, corresponden a:
1 2 3 0
0 0 0 0
*/

int tabla3[2][4] = {{1},{2,3}};
/* valors inicials incomplets, correspon a:
1 0 0 0
2 3 0 0
*/
/* valores iniciales a un vector sin dimensiones. La dimensión
indeterminada será siempre la primera */

int tabla4[][2] = {{1,2},{3,4},{5,6}};
/* valores iniciales que corresponden a:
1 2
3 4
5 6
*/
```

ACCESO A LOS ELEMENTOS DE LOS VECTORES

Para acceder a los elementos del vector será necesario indicar el (o los) índices que lo determinan. Ejemplo de como mostrar por pantalla el contenido de los vectores:

```
int num[5] = {0,1,2};
int tabla1[2][4] = {1,2,3,4,5,6,7,8};

for(i=0 ; i < 5 ; i++) //la dimension del vector es 5
      cout << "num[" << i << "]=" << num[i] << endl;

for(i=0 ; i < 2 ; i++) //tiene 2 filas
      for(j=0 ; i < 4 ; j++) //tiene 4 columnas
            cout<< "tabla1["<< i <<"]["<< j <<"]="<< tabla1[i][j] <<endl;
```

BLOQUE DE EJERCICIOS

1. Realizar el siguiente programa que pida diez valores por teclado y los guarde en un vector. A continuación se pasará el vector a una función que devolverá el valor máximo, el mínimo y el valor medio de los valores guardados en el vector:

```
#include <iostream>

using namespace std;


const int TAM=10;
void llenar ( int [TAM] );
void max_min_med ( int [TAM], int &, int &, int & );

void llenar ( int vector [TAM] )
{
   ...
}

void max_min_med ( int vector [TAM], int &maximo, int &minimo, int &media )
{
   ...
}

int main ( void )
{
   int vect [TAM];
   int vect, max, min, med;

   llenar ( vect );
   max_min_med ( vect, max, min, med );
   cout << "Max: " << max << "Min: " << min << "Med: " << med << endl;
   return 0;
}
```

2. Realizar un programa que lea una cantidad determinada (que se pedirá al principio del programa) de números enteros de una sola cifra, almacene dichos números en un vector y compruebe si el número formado por cada uno de los elementos del vector es capicúa o no. Tomar como tamaño máximo del vector, 100 elementos.

3. Realiza un programa que vaya pidiendo números enteros mientras que no se introduzca el cero y rellene dos vectores, uno con los números pares, y el otro con los número impares. Al final, se debe mostrar por pantalla tanto el vector de números pares y como el de impares indicando la posición del vector y el valor que ha sido almacenado.

4. Calcular la matriz transpuesta de una matriz dada. Una matriz transpuesta de otra dada es la que resulta de cambiar los valores de las filas de la matriz original por el de las columnas.

Definirse un prototipo de la forma:

```
const int MAX_F = 100;
const int MAX_C = 100;
typedef matriz double matriz[MAX_F][MAX_C];

void transpuesta(matriz a, int f, int c);
```

La función "transpuesta" recibe la matriz original introducida por teclado y el número de filas y columnas (parámetros f y c respectivamente) y tiene que calcular su transpuesta. El programa principal deberá mostrar por pantalla, en formato de matriz (filas x columnas) la matriz original y su transpuesta. Nota: Hacer también una función para leer matrices y para mostrarlas.

5. Realizar un programa que pida dos matrices (utilizar para ello la función de lectura de matices del ejercicio 4) y las sume (siempre que sea posible) o avise de que no se pueden sumar.

Nota:

La suma de dos matrices A y B es otra matriz C teniendo en cuenta que, para poder sumar matrices, ambas deben tener la misma dimensión, es decir:

A←m x n (A es una matriz de m filas y n columnas)
B←p x q (B es una matriz de p filas y q columnas)

Si *m=p* y *n=q* entonces:

C←m x n (C será la matriz suma que tendrá m filas y n columnas) y se obtendrá como:

$$C_{i,j} = A_{i,j} + B_{i,j}$$

6. Realizar un programa que pida dos matrices (utilizar para ello una función de lectura de matices del ejercicio 4) y las multiplique (comprobando que esta operación se puede realizar).

Nota:

Producto de las matrices A y B es C

A←m x p (A es una matriz de m filas y p columnas)
B←p x n (B es una matriz de p filas y n columnas)
C←m x n (C será una matriz que tendrá m filas y n columnas)

Donde $C_{i,j} = \sum_{k=1}^{p} A_{i,k} * B_{k,j}$

El programa principal deberá de tener la siguiente estructura:

```
#include <iostream>

/* Prototipos de funciones */
...

/* Desarrollo de las funciones necesarias */
...

/* Funcion principal */
int main ( void )
{
   int mat1[TAM][TAM], mat2[TAM][TAM], resultado[TAM][TAM];

….
/* si se puede realizar la operación entonces… */

   llenar_matriz ( mat1 , m , p);
   llenar_matriz ( mat2 , p , n);
   multiplicar ( mat1, mat2, resultado , m , p);
   mostrar_matriz ( resultado );
…
/* si no se puede avisar de ello */
   return 0;
}
```