Objetivo de la práctica:
- **Adquirir conocimietnos en en el uso de estructuras de datos basadas en series de datos del mismo tipo: *vectores y matrices en C/C++*.**
- **Uso de caracteres de texto en C/C++ .Concepto de *string***
- **Diseño y uso de Registros o Estructuras de datos generales en C/C++**

NOTA: Esta práctica requiere dos sesiones

PRIMERA PARTE

Conceptos básicos: VECTORES Y MATRICES

Un vector o array -arreglos en algunas traducciones- es una secuencia de objetos del mismo tipo almacenados consecutivamente en memoria. El tipo de objeto almacenado en el array puede ser cualquier tipo definido en C/C++.

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

INICIALIZACION DE VECTORES

Como toda variable, un vector puede tener valores iniciales después de su declaración. El formato general es:

tipo *identificador_variable* **[***tamaño***] =** *{ lista_de_valores };*

La lista de valores es una lista separada por coma " , ", de valores 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 se almacena en la primera posición del vector, la segunda constante de valor 1 en la segunda posición del vector, i 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 no asignados inicialmente. Por ejemplo:

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

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

También es posible al inicializar el vector, no declarar su tamaño. El compilador lo determina calculando el número de valores enumerados. Tenemos que la asignación siguiente:

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

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

Para el caso de poner valores iniciales a los vectors multidimensionales:

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

equivalente a:

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

En el ejemplo se declara una matriz de enteros de dos filas por cinco columnas.

Lo que sigue también es posible:

No obstante, es necesario ir con cuidado porque en las declaraciones incompletas las confusiones son fáciles, como muestra el ejemplo:

```
int taula1[2][4] = {1,2,3,4,5,6,7,8};

/* valors inicials complets, correspon a:

1 2 3 4            5 6 7 8  */

int taula2[2][4] = {1,2,3};

/* valors inicials incomplets, correspon a:

1 2 3 0            0 0 0 0 */

int taula3[2][4] = {{1},{2,3}};

/* valors inicials incomplets, correspon a:

1 0 0 0             2 3 0 0 */

/* valors inicials a un vector sense dimensions. La dimensió
indeterminada serà sempre la primera */

int taula4[][2] = {{1,2},{3,4},{5,6}};

/* valors inicials que correspon a:

1 2        3 4             5 6 */
```

EJERCICIO 1.
Hacer un programa que:

a) Lea una secuencia de 5 valores numéricos reales y los almacene en una vector.
b) Muestre por pantalla cuál es el valor mínimo, así como la posición que ocupa en el vector. En el caso de aparecer repetido el valor mínimo se mostrara el menor índice de los valores mínimos.

EJERCICIO 2

Compila y ejecuta el programa ***incògnita.cpp***. Qué hace el programa?.
Observa como se pueden introducir y leer los datos de la matriz.

EJERCICIO 3

Realizar un programa que permita la gestión de un vector de n enteros ordenado. El vector se creará vacío y se trabajará con un menú que permita las siguientes opciones:

Para insertar un elemento, pulsa 1
Para eliminar un elemento, pulsa 2
Para buscar un elemento, pulsa 3
Para salir, pulsa 4
NOTA: Utilizar el menú creado en la práctica 5.

Implementar una función para cada opción. Se deberá tener una variable que controle los elementos que contiene el vector, que será pasada por referencia a las funciones insertar y eliminar ya que en ellas se debe modificar. La dimensión del vector se definirá como una constante.

EJERCICIO 4

Calcular la matriz transpuesta de una matriz cuadrada. Una matriz cuadrada es aquella en qué el número de filas es igual al de columnas mientras que, una matriz transpuesta de una dada, es el resultado de cambiar en la matriz original el valor de las filas por el de las columnas.
Se definirá una función cuyo prototipo será:

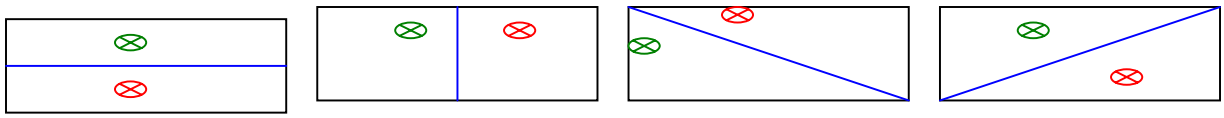
```
void transposada(double a[][MAX_FILES],int c); // MAX_FILES será una constante
```

Esta función recibirá la matriz original introducida por el teclado, y el número de sus columnas como un entero *c* y debe calcular su transpuesta. El programa principal debe mostrar por pantalla la matriz original y su transpuesta.

EJERCICIO 5 **(Opcional: este ejercicio servirá para subir nota)**

En una matriz cuadrada (nxn) de enteros se definen cuatro ejes de simetría que aparecen en la figura. Para un elemento dado (i,j) tendremos, por cada eje, un elemento de la matriz (i*,j*) que ocupa una posición simétrica con el anterior respecto a dicho eje.
Se pide un función que para un elemento (i,j) de la matriz, determine cuáles de sus cuatro posibles simétricos tienen además su mismo valor. El resultado se devolverá en un vector de cuatro elementos, uno por simetría, que indicará con el valor 1, o *true,* que ambos simétricos són iguales o con el valor 0, o *false,* que son distintos.

SIMETRÍA 0 ***SIMETRIA1*** ***SIMETRIA2*** ***SIMETRIA3***



Elemento simétrico :

Elemento (i,j):

AYUDA: Por ejemplo dada la siguiente matriz:

1,1	*1,2*	**1,3**	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	**3,4**
4,1	**4,2**	4,3	4,4

Los elementos simétricos del elemento (1,2) –en subrayado- serian los cuatro elementos en negrita.

1. Encontrar una relación entre los índices de los elementos para determinar las simetrías. Por ejemplo, para una simetría :

   (1,2)→(1,3).....i=i ; j*=n-j+1 // ojo , en C/C++ los indices comienzan por 0*

2. El prototipo de la función será:

   void simetries(int b[3],int i, int j);// b[0]=1 si hi ha simètria0…

Realizar un programa que pida por pantalla la introducción de una matriz cuadrada - n=4 elementos, como el ejemplo s-, el elemento (i,j) y se obtenga el vector b y un mensaje señalando los elementos simétricos iguales.
Se intentará realizar el programa con las máximas funciones posibles, por ejemplo, la introducción de la matriz, la obtención de resultados y el calculo de las simetrías serían las tres funciones a realizar.

SEGUNDA PARTE

Conceptos básicos: tipo string

Las cadenas de caracteres (strings) permiten la manipulación de textos. Generalmente, se considera que un string no es más que un array de caracteres. Por esa razón, se suele relacionar el concepto de string con el de array. En C++, existe el tipo (clase, en la nomenclatura de los lenguajes orientados a objetos) string. Para su uso es preciso utilizar **#include <string>** , no <string.h> .

Operaciones básicas definidas para string:
• **Creación de variables**:
string paraula, frase;

• **Asignación:**
frase = paraula; *frase = "hola";*
• **Acceso a los caracteres (como arrays):**
paraula[0]
• **Comparación lexicográfica (== , != , < , >):**
frase == paraula ; frase > paraula;
• **Lectura/escritura:**
cin >> paraula;
getline (cin, frase); //lee frase de cin hasta encontrar el fin de línea
cout << frase << endl;

• **Manipulación de textos (suponer unsigned int i;):**

// nº de caracteres de *paraula*
i = paraula.length();
// inserta *paraula* en la posición 3 de frase
frase.insert(3, paraula);
// concatena (une) *paraula* y "hola" y almacena el resultado en *frase*
frase = paraula + "hola";
// concatena (añade al final) *paraula* a *frase*
frase += paraula;
// borra 7 caracteres de *frase* desde la posición 3
frase.erase (3,7);
// sustituye (reemplaza) 6 caracteres de *frase*, empezando en la posición 1, por la cadena *paraula*
frase.replace (1, 6, paraula);
//busca *paraula* como una subcadena dentro de frase, devuelve la posición donde la encuentra
i = frase.find(paraula);
//devuelve la subcadena formado por 3 caracteres desde la posición 5 de *frase*
paraula=frase.substr(5,3);

NOTA : Un string puede considerarse también como una vector de caracteres:
char frase[5]="hola" // se guardara como un vector formado por

'h' 'o' 'l' 'a' '\0'

Observar que se añade el carácter ***'\0'*** al final del vector.

EJERCICIOS

6. Escribir un programa que convierta a mayúsculas todas las letras de una frase dada por teclado y muestre por pantalla la conversión.

7. Realizar un programa en C++ que pida una frase y nos indique si es o no palíndroma (se lee igual de izquierda a derecha que de derecha a izquierda). Por ejemplo:
Entrada: dabale arroz a la zorra el abad
Salida: cierto

8. Escribir un programa en C++ que dada una cadena de caracteres permita sustituir todas las ocurrencias de una subcadena. Ejemplo:
Cadena: ***Aquest problema és més interesant que el problema anterior***
Vieja Subcadena: ***problema***
Nueva Subcadena: ***programa***
Cadena Resultado: ***Aquest programa és més interesant que el programa anterior***

9.(Opcional: este ejercicio servirá para subir nota)

9.a. Escribir un programa en C++ que calcule el número de vocales que contiene una frase introducida por teclado y nos diga cuantas vocales de cada clase hay (cuantas 'a', cuantas 'e', ...)
9.b. Escribir un programa en C++ que calcule el número de letras de cada clase que contiene una frase introducida por teclado y nos diga cuantas hay (cuantas 'a', cuantas 'b', ...)

TERCERA PARTE

CONCEPTOS BÁSICOS

El tipo de dato registro

Los registros, también llamados a veces por vicio del lenguaje "estructuras", son tipos compuestos de datos heterogéneos, es decir, que agrupan datos que pueden ser de diferentes tipos.
Consideremos por ejemplo una ficha de venta de un libro para una librería. En ella se guarda la siguiente información:
-Titulo del libro
-Autor
-Editorial
-número de páginas
-precio

Toda esta información puede guardarse en una estructura de datos de tipo registro. Para definirla se utiliza la palabra reservada `struct` y tiene la siguiente sintaxis:

```
struct ficha{
   string titulo;
   string autor;
   string editorial;
   int num_pag;
   float precio;
};
```

Esta declaración debe entenderse como una declaración de tipo (de ahí el punto y coma final) y no como un subprograma **ni como una declaración de variables**.

Para declarar una variable de tipo ficha deberemos hacerlo dentro de un subprograma de la manera habitual:

```
int main(){
ficha venta;
....
return 0;
}
```

Ahora la variable `venta` es de tipo `ficha`.

A cada uno de los componentes de un registro se denomina **campo**. Así, los datos `titulo, autor, editorial, num_pag, precio`, son los campos del registro.

Para acceder a cada uno de los campos de un registro se utiliza el operador punto '`.`'

```
venta.titulo = "El conde de Montecristo";
```

Inicialización de un registro

Un registro puede inicializarse en el momento de su declaración como variable:

```
struct dimensiones_libro{
   float alto;
   float ancho;
};
int main(){
   dimensiones_libro dim={12.4, 20.8};
....
return 0;
}
```

Como caso aparte, si algún campo del registro es de tipo `string`, no puede inicializarse de la anterior manera.
La alternativa es la inicialización campo a campo que podemos hacer en cualquier punto del programa:

```
int main(){
   ficha libro;
...
   libro.titulo = "El conde de Montecristo";
   libro.autor = "Alejandro Dumas";
   libro.editorial = "Aguilar";
   libro.num_pag = 324;
   libro.precio = 12.35;
...
   return 0;
}
```

Entrada y salida de registros

Tanto la entrada como la salida de un registro debe hacerse campo a campo, siempre que el campo sea de un tipo de dato que tenga salida directa con `cout` o entrada directa con `cin` (tipo de dato simple o tipo string).

Asignación entre registros y paso como parámetro a un subprograma

En C++ está permitida la asignación entre registros:

```
dimensiones_libro libroA, libroB;
libroA.alto = 13.56;
libroA.ancho = 12.56;

libroB = libroA; //ahora libroB tiene el mismo contenido que libroA
```

Como consecuencia de esta propiedad, **un registro puede pasarse por valor o por referencia como parámero a una función. Igualmente una función puede devolver un registro**.

```
dimensiones_libro carga_informacion(ficha & mi_libro){
  dimensiones_libro dim;

  cout << "Escribe titulo \n";
  cin >> mi_libro.titulo;
  cout << "Escribe autor \n";
  cin >> mi_libro.autor;
```

```
  cout << "Escribe editorial \n";
  cin >> mi_libro.editorial;
  cout << "Escribe numero de paginas \n";
  cin >> mi_libro.num_pag;
  cout << "Escribeprecio \n";
  cin >> mi_libro.precio;

  cout << "Escribe centimetros de alto\n";
  cin >> dim.alto;
  cout << "Escribe centimetros de ancho \n";
  cin >> dim.ancho;

  return(dim);
}

int main(){
   ficha libroA;
   dimensiones_libro dimA;

   dimA = carga_informacion(libroA);
...
   return 0;
}
```

Uso de registros en estructuras de datos complejas

Registros con campos de registros

Un registro puede tener como campo otro registro. Para ello el registro que hace de campo debe declararse antes que en el momento de su uso como tipo de dato de un campo de otro registro.

```
struct dimensiones_libro{
   float alto;
   float ancho;
};

struct ficha_modif{
   string titulo;
   string autor;
   string editorial;
   int num_pag;
   float precio;
   dimensiones_libro tamanyo;
};
```

El acceso a los campos del registro "tamanyo" se hace usando dos veces el operador punto '.'

```
ficha_modif libro;

   cout << "El libro mide " << libro.tamanyo.alto  << "cm de alto \n";
```

El campo registro tiene todas las propiedades de los registros (asignación,...).

Vectores de registros

Esta estructura de datos es muy útil para hacer colecciones de datos . Por ejemplo para tener los datos de todos los libros en venta de una librería, podemos guardar las fichas de los libros en un vector.

```
const int MAX_TAM= 500;

ficha inventario[MAX_TAM];
```

La variable `inventario` guardaría hasta un máximo de 500 fichas de libros en venta.

Registros con vectores

Igualmente un campo de un registro puede ser un vector o una matriz.
En nuestro ejemplo sería interesante llevar la cuenta de la ocupación útil de nuestro vector de fichas.
Para ello me creo un registro más complejo que será:

```
struct titulos_en_venta{
  ficha stock[MAX_TAM];
  int cuantos;
};
```

El campo `cuantos` de este registro, indica el número de fichas rellenas del vector `stock`.

Para listar todos los títulos disponibles podemos hacer un bucle:

```
titulos_en_venta oferta;
//Inicializamos oferta

for(i=0;i<oferta.cuantos;i++)
   cout << oferta.stock[i].titulo;
```

EJERCICIOS:

10. Escribe en un fichero texto la definición de las estructuras de datos necesarias para almacenar la siguiente información:

a) Una hipoteca bancaria donde quede reflejado el capital de la hipoteca, el porcentaje de interés cobrado, el número de años de duración de la hipoteca y el nombre del beneficiario de la misma.
b) La información escrita que hay en el anverso de un DNI (Nombre, Apellido1, apellido2, fecha creación, fecha caducidad,numero dni, letra nif)
c) Una imagen de 64x64 pixeles, donde cada pixel es definido por tres números entre 0 y 255, en el que cada número hace referencia al grado de participación de un color (Rojo,verde,azul) en la composición de un color (por ej r=255,g=255, b= 255 es el color blanco y r=255, g=0, b=0 es el color rojo).
d) El juego de la Oca. Donde cada casilla (hay 60) tiene la información del número de casilla, si está o no ocupada por una ficha, el color de la ficha que la ocupa y un campo que indica el tipo de casilla que es (hay 8 tipos: normal, oca, puente, dados, muerte, laberinto, cárcel, Final).
e) Un tablero de ajedrez. Donde en cada casilla puede haber una ficha de un determinado tipo y color

11. Crea una estructura de datos que almacene la información de un vector en el espacio $\Re^3$. Deberás guardar sus tres componentes y el nombre del vector.

Escribe una función que sume dos vectores en $\Re^3$ y devuelva el resultado. Llámala desde el programa principal que es el que debe de inicializar los vectores pidiendo los datos al usuario.

12. Repetir el ejercicio 3, pero ahora en lugar de ser el vector de tipo entero, se va a definir un registro alumno con los siguientes campos:

nombre, apellidos, nif, tfno, nota febrero, nota junio, nota final.

El vector estará ordenado por el campo apellidos, si hay igualdad se usa el nombre para el desempate.

introducir una nueva opción al menú que es calcular la nota media de la clase.

(Opcional: este ejercicio servirá para subir nota)

13. Crea una estructura de datos que almacene información acerca de componentes mecánicos de automóvil.
Cada ficha de un componente debe llevar la siguiente información:
-Nombre de la pieza
-unidades disponibles
-precio de la pieza

La estructura de datos debe ser capaz de almacenar un máximo de **10** piezas.
Haz un programa, que mediante un menú realice las siguientes acciones:
-Introducir una nueva pieza, tras comprobar que queda espacio
-Eliminar una pieza corriendo una posición todas las posteriores. Debemos llevar un contador para saber el número de piezas almacenadas en la estructura de datos
-Buscar por el nombre y listar por pantalla las características de una pieza dada.

Cada opción del menú debe ser realizada como un subprograma diferente pasando los parámetros que consideres necesarios.