Práctica 6: Vectores y Matrices

Objetivos.

- Aprender a diseñar e implementar tipos de usuarios definidos por el programador.
- Usar las estructuras de datos basadas en colecciones del mismo tipo, los tipos de datos estructurados homogéneos. En concreto: los vectores y las matrices.

Conceptos Básicos.

Un vector (o *array unidimensional*) es una colección de objetos del mismo tipo que se almacenan de forma contigua en memoria. Puede ser de cualquier tipo definido en C++. Se adopta el convenio de usar *typedef* para crear los tipos de datos vectores y matrices. Esto permite aumentar la legibilidad y mantenimiento de los programas. Por ejemplo:

```
typedef int Vector[20]; //define un tipo de datos: vector de
                        20 enteros
Vector v;  //declara una variable v de tipo vector de 20 enteros
```

El acceso a los elementos de un vector se realiza mediante un índice. Los elementos están ordenados y el índice comienza desde 0.

```
v[3] = 45; //asigna el valor 45 a la cuarta componente
```

De forma análoga, el tipo de dato matriz (o *array* bidimensional) se define como:

```
typedef int Matriz[10][4];
Matriz m;

m[1][3] = -5; //asigna el valor -5 al elemento 2ª fila, 4ª col.
```

Los vectores y matrices se pasan por defecto por referencia como parámetros de una función, para evitar duplicar la memoria y evitar el coste temporal de copiar. No se usa el símbolo '**&**'. Simplemente, se indica el tipo de dato y el identificador de la variable.

Incialización de un Vector.

Un vector puede tomar valores iniciales al ser declarado. La sintaxis es:

```
tipo identificador [tamaño] = { lista_de_valores };
```

La lista de valores es una lista separada por comas de constantes numéricas que son del mismo tipo que el tipo base del vector. Por ejemplo:

```
int v[5] = {20, 10, 5, 13, 1};
```

La primera constante de valor 20 se almacena en la primera posición del vector, la segunda constante de valor 10 en la segunda posición del vector, y así sucesivamente.

No es necesario poner el valor inicial a todo el vector, en este caso, el compilador inicializa a ceros a aquellos elementos a los que no les hayamos asignado valor. En este ejemplo, el compilador asignará ceros a los dos últimos elementos del vector.

```
int v[5] = {20,10,25};
```

También es posible no declarar el tamaño. El compilador lo determina calculando el número de valores enumerados. Así, la sentencia siguiente es correcta:

```
int v[] = {0,-1, 8, -3, 2};
```

El tamaño de la variable *v* es 5.

Incialización de una Matriz.

La inicialización de una matriz se realiza de forma parecida. Por ejemplo:

```
int m[2][4] = {{0, -1, 6, -3},{-5, 8, 1, 0}};
```

Es equivalente a:

```
int m[2][4] = {0, -1, 6, -3, -5, 8, 1, 0};
```

Se ha declarado y dado valores iniciales a una matriz de enteros de dos filas y cuatro columnas.

Las matrices también se pueden inicializar de forma parcial con la única condición de que se debe comenzar por el principio de cada dimensión.

```
int m[2][4] = {1,2,3};
/* 1 2 3 0
   0 0 0 0 */

int m[2][4] = {{1},{2,3}};
/* 1 0 0 0
   2 3 0 0 */
```

Si damos valores iniciales a un vector sin dimensiones, la dimensión indeterminada es siempre la primera:

```
int m[][2] = {{1,2},{3,4}};
/* 1 2
   3 4 */
```

BLOQUE DE EJERCICIOS

Ejercicio 1 (ejercicio1.cpp): El siguiente programa debe leer los elementos de un vector y mostrar la suma de ellos. Modifícalo hasta que funcione correctamente.

```
#include <iostream>

using namespace std;

const unsigned int TAM = 4;
typedef float VectorF[TAM];

void leeVector ( );       // Línea a modificar
float sumaVector ( const VectorF );

int main()
{
  VectorF v;
  float suma;

  leeVector ();       // Línea a modificar
  sumaVector (v);     // Línea a modificar

  cout << "Los elementos del vector suman " << suma << endl;

  return 0;
}

void leeVector ( )    // Línea a modificar
{
  cout << "Introduce los elementos del vector:" << endl;

  for (unsigned int i = 0; i < TAM; i ++ )
  {
    cout << "elemento(" << i << ")? ";
    cin >> w[i];
  }

  return;
}

float sumaVector ( const VectorF )              // Línea a modificar
{
  float res = 0.0;

  for (unsigned int i = 1; i <= TAM; i ++ )    // Línea a modificar
    res = res + u[i];

  return res;
}
```

Ejercicio 2 (ejercicio2.cpp): Este ejercicio muestra algunos de los problemas que pueden ocurrir al manipular vectores de forma inadecuada y corromper la memoria. Abre el fichero ejercicio2.cpp. Añade puntos de ruptura en las líneas 60 y 65. **No modifiques el valor de la constante TAM**. Compílalo y ejecútalo.

Introduce el valor 3 para la cantidad de elementos del vector y rellena la siguiente tabla usando el comando `print` de GDB. Justifica si es correcta o no la información e interpreta los resultados.
https://www.calculadoraconversor.com/hexadecimal-a-decimal/

3 elem.	v[0]	&v[0]	v[1]	&v[1]	v[2]	&v[2]	v[3]	&v[3]	v[4]	&v[4]
Línea 60										
Línea 65										

Indica ahora el valor 5 para la cantidad de elementos del vector y rellena la tabla.

5 elem.	v[0]	&v[0]	v[1]	&v[1]	v[2]	&v[2]	v[3]	&v[3]	v[4]	&v[4]
Línea 60										
Línea 65										

Finalmente, usa el valor 15 para la cantidad de elementos del vector y rellena la tabla.

	info locals
Línea 60	
Línea 65	

¿La función `test()` termina correctamente? ¿Qué puede significar el mensaje “Cannot find bounds of current function”? ¿Qué le ha podido ocurrir a la pila de llamadas a funciones?
Pulsa “Next Instruction” ¿Qué significa el mensaje “Program received signal SIGSEGV, segmentation fault”?
https://docs.microsoft.com/en-us/cpp/c-runtime-library/signal-constants?view=msvc-170

Ejercicio 3 (ejercicio3.cpp): Implementa un programa en C++ que lea los elementos de una matriz de enteros de tamaño 4×3 y genere un vector de tamaño 4 en el que cada elemento almacene el valor de la suma de los elementos de una fila de la matriz. El programa debe mostrar la matriz original y el vector en el formato que se indica en el ejemplo:

```
0 [ 27 2 10 ]  ->  39
1 [ 5  0 -1 ]  ->   4
2 [ 3  2  0 ]  ->   5
3 [ 1 -2 12 ]  ->  11
```

Define los tipos de datos necesarios e implementa las siguientes funciones:

- Pide al usuario los elementos de la matriz y la rellena.
  `void leerMatriz(Matriz);`

- Calcula la suma de los elementos de una fila determinada.
  `int sumarFila(const Matriz, unsigned int);`

- Calcula los valores de la suma de cada fila y se guardan en un vector. Tiene como parámetro de entrada la matriz y devuelve como parámetro de salida el vector rellenado.
  `void rellenarVector(const Matriz, Vector);`

- Muestra el resultado tal y como se indica en el ejemplo.
  `void imprimirMatrizYVector(const Matriz, const Vector);`

Ejercicio 4 (ejercicio4.cpp): Implementa en C++ el juego de *Luces Fuera*. Es un juego de lógica que consiste en un tablero de 5x5 celdas que pueden estar encendidas (carácter 'X') o apagadas (carácter '0') y el objetivo es apagarlas todas. En cada turno, el jugador selecciona una celda del tablero. La celda seleccionada, y todas sus vecinas no diagonales, cambian su estado (de encendidas a apagadas y viceversa).

Puedes entrenarte en:

https://www.logicgamesonline.com/lightsout/

Y conocer la estrategia ("cazando las luces") para resolver el juego en:

`https://www.logicgamesonline.com/lightsout/tutorial.html`

Por ejemplo, si partimos del siguiente tablero:

```
X 0 0 0 0
0 0 0 0 0
0 0 X X 0
0 0 0 0 0
0 X 0 0 0
```

```
X 0 0 0 0
0 0 X 0 0
0 X 0 0 0
0 0 X 0 0
0 X 0 0 0
```

Cambios al elegir la fila y col. 3

Define las siguientes funciones:

- **Inicializar** el tablero. Establece aleatoriamente para cada elemento de la matriz su valor encendido o apagado.
- **Mostrar** el contenido del tablero.
- **Pedir** por teclado el número de fila y de columna de la celda para la jugada. Se debe comprobar que los valores introducidos son correctos, de lo contrario se solicitaran valores nuevos hasta que lo sean.
- **Actualizar** el estado de una celda y de sus vecinas no diagonales (de encendido a apagado y viceversa) a partir del número de fila y de columna que se pasan como parámetro. ¡Cuidado con el efecto de borde! ¡No te salgas del tablero!
- **Comprobar** el número de luces encendidas y devolver este valor.

El programa principal es:

```
const unsigned int TAM = 5;
typedef char Matriz[TAM][TAM];

int main()
{
  Matriz t; //tablero
  unsigned int fil, col, luces, njugadas = 0;

  inicializar (t);
  mostrar (t);

  do
  {
    pedir (fil, col);
    actualizar (t, fil, col);
    mostrar (t);
    luces = comprobar (t);
   cout << “Cantidad de luces encendidas: “ << luces << endl;
    njugadas++;
  }
  while (luces > 0);

  cout << “Has ganado en ” << njugadas << “ jugadas. “ << endl;
  cout << “Enhorabuena!” << endl;

  return 0;
}
```

Ayuda:

```
Para generar números aleatorios usa rand(). Incluye <cstdlib> y <ctime>.
srand(time(NULL)); //inicializa el generador de números aleatorios
val = rand()%N; //var. aleatoria uniformemente distribuida en [0,N-1]
```

https://www.cplusplus.com/reference/cstdlib/rand/
https://www.cplusplus.com/reference/cstdlib/srand/

Ejercicio 5 (ejercicio5.cpp): Escribe un programa en C++ que permita manipular matrices de números enteros positivos con un tamaño indicado por el usuario. El tamaño de la matriz será **como mucho** 64x64 elementos (arrays parcialmente llenos). El programa debe permitir al usuario las siguientes operaciones:

A. Rellenar una matriz.
B. Mostrar en pantalla la matriz original.
C. Umbralizar la matriz.
D. Mostrar en pantalla la matriz umbralizada.
E. Acabar.

Implementa las funciones:

- **`randomMatriz`**. Tiene como parámetro de entrada el tamaño (número de filas y columnas realmente utilizadas) y devuelve como parámetro de salida la matriz rellenada con números aleatorios comprendidos entre el 0 y 255.
- **`umbralizarMatriz`**. Pone a 0 los valores de la matriz iguales o inferiores al umbral y a 1 los valores superiores al umbral. La función tiene como parámetros de entrada: la matriz original, el tamaño y el umbral a aplicar. La función tiene como parámetro de salida la matriz donde guarda el resultado. *Ayuda*: boletín de prerrequisitos.
- **`imprimirMatriz`**. Tiene como parámetro de entrada la matriz a imprimir y el tamaño.
- **`menú`**. Imprime el menú y devuelve la opción elegida por el usuario.

```
const unsigned short MAX = 64;
typedef unsigned short Matriz[MAX][MAX];

void randomMatriz (Matriz, unsigned short, unsigned short);

void umbralizarMatriz (const Matriz, Matriz, unsigned short, unsigned
short, unsigned short);

void imprimirMatriz (const Matriz, unsigned short, unsigned short);


char menu (void);
```

IMPORTANTE:

- Todos los programas deben seguir la Guía de Estilo e incluir los comandos de Doxygen para generar la documentación.
- Subir al Aula Virtual los archivos .cpp de uno en uno (enunciado y adicionales). Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.