Estructuras de Datos I

LABORATORIO

Práctica 0

Objetivos de la práctica:

- Que el alumno se inicie en la Programación Modular.
- Que el alumno descubra de forma práctica las ventajas del Diseño Modular.
- Que el alumno aprenda el concepto de proyecto.

Enunciado de la Práctica

Se le entrega el código de un programa en C++ que permite cargar, eliminar, buscar, listar y realizar operaciones aritméticas con matrices de números complejos o de números racionales (de longitud 5 x 5 elementos como máximo). El programa debe ser capaz de manejar un máximo de 20 matrices y constará de un intérprete de comandos que estará a la espera de órdenes dadas por el usuario. Las órdenes que el intérprete aceptará serán las siguientes:

1. Cargar matrices:
   Para cargar matrices con las que operar utilizaremos la orden **`cargar.`** Será la forma correcta de cargar más de una matriz a la vez. Los elementos de las matrices se leerán desde fichero, por lo que se proporciona el método de la clase matriz que realiza la carga y el fichero "numeros.dat" que contiene los elementos a leer.

   ```
   >cargar
   ```

   Una vez introducida la orden se pedirá al usuario el número de matrices que desea cargar y el programa debe chequear que no se supera el máximo de matrices que se pueden almacenar. Si esto ocurriera, se mostraría un mensaje de error, y en caso contrario se cargarán las matrices. El método cargar de la clase matriz que se proporciona permite la carga de una matriz, pide filas y columnas de la matriz y lee los elementos desde fichero. Habrá que hacer tantas llamadas al método cargar("numeros.dat"), como matrices se desee almacenar.

   ```
   >cargar
   Cuantas matrices desea almacenar?
   2
   Filas: 2
   Columnas: 2
   Carga de una matriz 2 x 2:
   [1,1]: 50/2
   [1,2]: 14/2
   [2,1]: 41/9
   [2,2]: 91/18
   Filas: 1
   Columnas: 1
   Carga de una matriz 1 x 1:
   [1,1]: 79/2
   >cargar
   Cuantas matrices desea almacenar?
   19
   Error: Se excede el numero de matrices con las que operar
   >
   ```

2. Listar matrices:
   Para mostrar por pantalla las matrices almacenadas utilizaremos la orden **listar**. En caso de no tener almacenada ninguna matriz mostrará un mensaje de error.

```
>listar
Error: No hay cargada ninguna matriz
```

Si por el contrario tenemos cargadas matrices, se mostrarán por pantalla.

```
>listar
Matriz 1:
50/2    14/2
41/9    91/18
Matriz 2:
79/2
>
```

3. Buscar una matriz:
   Para saber si una matriz introducida desde teclado ha sido almacenada previamente usaremos la orden **buscar**.

```
>buscar
```

A continuación se pedirán desde teclado las filas, columnas y elementos de la matriz que queremos buscar. Si la matriz no existe debe dar un mensaje de error. En caso contrario debe mostrar en pantalla la posición en la que se encuentra la matriz.

```
>buscar
Filas: 2
Columnas: 2
Carga de una matriz 2 x 2:
[1,1]: 50 2
[1,2]: 14 2
[2,1]: 41 9
[2,2]: 91 18
La matriz está en la posición 0
>buscar
Filas: 1
Columnas: 1
Carga de una matriz 1 x 1:
[1,1]: 12 3
Error: No se ha encontrado la matriz
>
```

4. Insertar una matriz:
   Para insertar una nueva matriz usaremos el comando **++** (incremento).

```
>++
```

A continuación se pedirá en que posición queremos insertar la nueva matriz. La posición podrá ser al principio (posición 0), al final (posición correspondiente al número de matrices que hay) o una posición intermedia, no pudiendo existir huecos. Si la posición introducida no es válida, se mostrará un mensaje de error, en caso contrario se desplazarán una posición hacia delante todas las matrices a partir de la posición introducida para dejar espacio para la nueva matriz, se pedirán los datos de la nueva matriz, se insertará la nueva matriz y se incrementará el número de matrices almacenadas.

```
>++
Posición? 4
Error: La posición no es valida
>++
Posición? 0
Filas: 1
Columnas: 1
Carga de una matriz 1 x 1:
[1,1]: 12 3
>listar
Matriz 1:
12/3
Matriz 2:
50/2    14/2
41/9    91/18
Matriz 3:
79/2
>
```

5. Eliminar una matriz:
   Para eliminar una matriz de las almacenadas usaremos el comando **--** (decremento).

```
>--
```

   A continuación se pedirá que posición tiene la matriz que queremos eliminar. La posición podrá ser al principio (posición 0), al final (posición que tenga la última matriz almacenada) o una posición intermedia. Si la posición introducida no es válida, se mostrará un mensaje de error, en caso contrario se desplazarán una posición hacía atrás todas las matrices a partir de la posición introducida para ocupar el espacio que deja la matriz eliminada y se decrementará el número de matrices almacenadas.

```
>--
Posición? 4
Error: La posición no es valida
>--
Posición? 1
>listar
Matriz 1:
12/3
Matriz 2:
79/2
>
```

6. Consultar matriz:
   Para consultar una matriz en una determinada posición usaremos el comando **?** (interrogación).

```
>?
```

   A continuación se pedirá la posición que tiene la matiz que queremos consultar. Si la posición introducida no es válida, no hay ninguna matriz en esa posición, se mostrará un mensaje de error, en caso contrario se mostrará por pantalla la matriz consultada.

```
>?
Posición? 4
Error: La posición no es valida
>?
Posición? 1
79/2
>
```

7. Sumar matrices:
   El comando para sumar dos matrices será el signo **+**.

```
>+
```

   A continuación se pedirán las dos posiciones que ocupan las matrices que queremos sumar. El programa debe chequear que las matrices a sumar existen, las posiciones son válidas, así como que dichas matrices pueden sumarse (tienen el mismo número de filas y de columnas). En el caso que la suma se pueda realizar se mostrará en pantalla el resultado. En caso contrario, mostrará un mensaje de error.

```
>+
Posiciones? 0 2
Error: Posiciones no validas
>+
Posiciones? 0 1
Matriz resultante:
261/6
>++
Posición? 0
Filas: 2
Columnas: 2
Carga de una matriz 2 x 2:
[1,1]: 50 2
[1,2]: 14 2
[2,1]: 41 9
[2,2]: 91 18
>+
Posiciones? 0 1
Error: Las matrices no se pueden sumar
>
```

8. Restar matrices:
   El comando para restar dos matrices será el signo **-**.

```
>-
```

   A continuación se pedirán las dos posiciones que ocupan las matrices que queremos restar. El programa debe chequear que las matrices a restar existen, las posiciones son válidas, así como que dichas matrices pueden restarse (tienen el mismo número de filas y de columnas). En el caso que la resta se pueda realizar se mostrará en pantalla el resultado. En caso contrario, mostrará un mensaje de error.

```
>-
Posiciones? 0 4
```

```
Error: Posiciones no validas
>-
Posiciones? 0 2
Matriz resultante:
-213/6
>-
Posiciones? 0 1
Error: Las matrices no se pueden restar
>
```

9. Multiplicar matrices:
   El comando para multiplicar matrices será el signo *****.

```
>*
```

   A continuación se pedirán las dos posiciones que ocupan las matrices que queremos multiplicar. Al igual que ocurre con la suma y resta, el programa debe chequear que las matrices a multiplicar existen, así como que dichas matrices pueden multiplicarse (el número de columnas de la primera es igual al número de filas de la segunda). En caso que el producto de las dos matrices se realice mostrará en pantalla el resultado del mismo, en caso contrario, mensaje de error.

```
>*
Posiciones? 0 4
Error: Posiciones no validas
>*
Posiciones? 0 2
Matriz resultante:
948/6
>*
Posiciones? 0 1
Error: Las matrices no se pueden multiplicar
>
```

10. Salir de la aplicación:
    Para abandonar el intérprete de comandos introduciremos la orden salir.

```
>salir
```

Cualquier otro comando introducido que sea distinto a los anteriores o que no siga estrictamente la sintaxis indicada antes debe dar un mensaje de error:

```
>exit
Error: comando o orden incorrecta
>¿
Error: comando o orden incorrecta
>sumar
Error: comando o orden incorrecta
```

Estructuras, clases, atributos y métodos a usar

Para el desarrollo de la práctica se han implementado y utilizado las siguientes clases:

- Una clase **complejo** para definir las operaciones con números complejos
- Una clase **quebrado** para definir las operaciones con números racionales
- Una clase **matriz** para operar con matrices de números complejos o racionales

Definición de la clase quebrado

```
class quebrado {
      int numerador, denominador;
public:
      quebrado() { numerador=0; denominador=1; }
      quebrado(int n, int d=1);
      int getnumerador() { return numerador; }
      int getdenominador() {return denominador; }
      void valor(int n, int d=1);
      quebrado operator+(quebrado q);
      quebrado operator-(quebrado q);
      quebrado operator*(quebrado q);
      quebrado operator/(quebrado q);
      quebrado operator-();
      int operator==(quebrado q);
      void mostrar();
};
// prototipo de funciones genéricas usadas en los métodos
int mcd(int a, int b);
int mcm(int a, int b);
```

Definición de la clase complejo

```
class complejo {
      double real, imag;
public:
      complejo(void) { real = 0.0; imag = 0.0; }
      complejo(double re, double im=0.0) { real = re; imag = im;}
      complejo(const complejo& r) { real = r.real;imag = r.imag;}
      void SetReal(double re) { real = re; }
      void SetImag(double im) { imag = im; }
      double GetReal(void) {return real; }
      double GetImag(void) {return imag; }
      complejo operator+(int e);
      complejo operator+(complejo c);     //por valor
      complejo operator-(const complejo& c); //por ref
      int operator== (complejo c);
      complejo operator=(complejo c);
      complejo operator*(complejo c);
      complejo operator++(); // ++obj;
      complejo operator++(int notused); //obj++
      complejo operator-();
```

```
      void mostrar();
};
```

Definición de la clase matriz

```
#define M 5
typedef quebrado tipoelemento;
class matriz {
      tipoelemento celda[M][M];
      int fila,col;
public:
      matriz(int f=0, int c=0);
      int getfila() { return fila; }
      int getcol() { return col; }
      bool operator==(matriz m);
      matriz operator*(matriz m);
      matriz operator+(matriz m);
      matriz operator-(matriz m);
      matriz operator-();
      void mostrar();
      void cargar(); //carga una matriz desde teclado
      void cargar(char cad[]);//carga una matriz desde fichero
};
```

Como se puede observar en la línea `typedef quebrado tipoelemento;` se define el tipo de los elementos de los objetos de la clase matriz que se definan. El programa principal que se pide debe de permitir operar con matrices de números complejos y con matrices de números racionales. Esto se debe conseguir con sólo cambiar esta línea de definición de la clase matriz por `typedef complejo tipoelemento;`

Los ejemplos que se han dado del funcionamiento de los comandos del interprete de comandos que se pide, son de operaciones con matrices de números quebrados, para que estas operaciones se realicen con números complejos sólo habrá que cambiar el tipo tipoelemento y los complejos se visualizarán de la forma parte_real+parte_imaginaria**i**.

Se proporciona la implementación del método de la clase matriz cargar, que permite la carga de los elementos de una matriz desde fichero y el fichero desde él que leer los elementos de la matriz llamado “numeros.dat”.

```
void matriz::cargar(char cad[]) {
 int x, y;
 FILE *fin=NULL;
 do {
  cout << "Filas: "; cin >> fila;
  cout << "Columnas: "; cin >> col;
 } while (fila>M || fila<1 || col>M || col<1);
 cout << "Carga de una matriz " << fila << " x " << col << ":\n";
 fin = fopen(cad, "rb");
 if (fin != NULL) {
  for (int i=0; i<fila; i++)
   for (int j=0; j<col; j++) {
            int n= rand()%80+1;
            fseek(fin,sizeof(int)*n,SEEK_SET);
            fread(&x,sizeof(int),1,fin);
            fread(&y,sizeof(int),1,fin);
            tipoelemento A(x,y);
            celda[i][j]=A;
```

```
            cout << "[" << i+1 << "," << j+1 << "]: ";
            A.mostrar();
            cout<<'\n';
  }
    fclose(fin);
    fin = NULL;
  }
  else {
    cout << "Error al abrir el fichero de numeros\n";
  }
}
```

¿Qué tenemos que hacer?

Nos proporcionan un programa escrito y dispuesto en un único fichero y no diseñado de forma modular. Las clases complejo, matriz y quebrados son susceptibles de ser usadas en otros posibles programas y aconsejan que el proyecto hubiera sido desarrollado de forma modular. Nuestro propósito es éste. Practicar a partir de este programa y transformarlo en un proyecto diseñado de forma modular (como debería haber sido de un principio diseñado)

Para poner en práctica el diseño modular, cada una de estas clases se deben definir e implementar como un módulos, con su fichero de definición o cabecera (.h) y su fichero de implementación (.cpp). El proyecto en Codeblocks unirá estos módulos con el programa principal que hace uso de ellos.