CSC 402/502 HW6 Instructor: Jeff Ward
Covers: C++ templates and operator overloading 25 points

Setup Stage 1: Installing a C++ environment

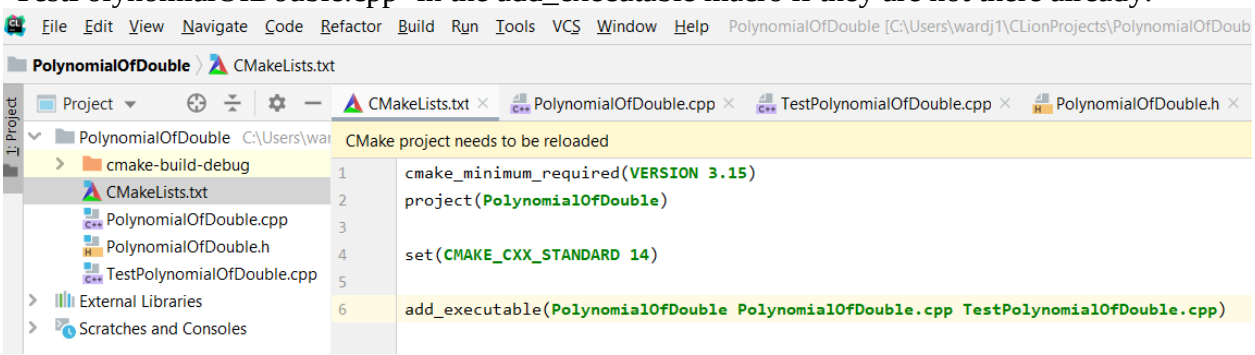
Among your choices for a C++ IDE are JetBrains CLion (similar to JetBrains IntelliJ and JetBrains Rider) or Microsoft Visual Studio (which some of you may have used in CSC 362 or CSC 501). I've test both options and either works fine for our purposes.

To install JetBrains Rider I followed the first 7 minutes of this video: https://www.youtube.com/watch?v=HSf-GiJr1Bs. The CLion download page is https://www.jetbrains.com/clion/download/ and the video will step you through the setup. You will have to select which compiler you want to use: I suggest choosing MinGW. The process of installing the compiler is integrated into the process of setting up the IDE, as you will see in the video.

Setup Stage 2: Running the examples provided on Canvas

Download from Canvas the following files: PolynomialOfDouble.h, PolynomialOfDouble.cpp, TestPolynomialOfDouble.cpp, Polynomial.h, TestPolynomial.cpp, MatrixOfDouble.h, MatrixOfDouble.cpp, TestMatrixOfDouble.cpp, and TestMatrix.cpp.

Create a project named PolynomialOfDouble. With CLion you would do this via File->'New Project'. Select 'C++ Executable'. Change the 'untitled' part of the location to 'PolynomialOfDouble'. The default 'Language standard' that CLion selects is C++14, which will work fine for us. CLion will create a main.cpp file with code to print a "Hello, World!" message. You can select Build->'Build PolynomialOfDouble' to build the project and Run->'Run PolynomialOfDouble' to run it. Once that works, copy each of the PolynomialOfDouble files mentioned above and paste them into the PolynomialOfDouble folder in the project explorer. Right-click on the main.cpp file and 'Delete' it. Double-click the CMakeLists.txt file so that it opens up for editing. Enter 'PolynomialOfDouble', 'PolynomialOfDouble.cpp', and 'TestPolynomialOfDouble.cpp' in the add_executable macro if they are not there already:



Select Build->'Rebuild Project' and run the project. The code is discussed in one of the videos provided with this assignment.

Creating a project with these files using Microsoft Visual Studio is similar, if you are using that IDE.

Likewise create and run the provided code for a 'Polynomial' project, which demonstrates a class template that can be used to represent polynomials over types of values other than doubles only.

Also create and run the provided code for a 'MatrixOfDouble' class.

Besides the video on the 'PolynomialOfDouble' project, there are also videos on 'Polynomial' and 'MatrixOfDouble'.

Similar to the way in which the Polynomial class template generalizes the PolynomialOfDouble class, your task in this assignment is to write a Matrix class template that generalizes MatrixOfDouble...

Your assignment: A Class Template for Matrices

Create a project called Matrix. Then, in a file called Matrix.h implement a Matrix class template that takes three template parameters:

1. the type, T, of the entries
2. the number of rows in each matrix
3. the number of columns in each matrix

```
template<typename T, size_t rows, size_t cols>
class Matrix {
```

The advantage of having the dimensions of a matrix as part of its type is that it allows the *compiler* to verify that the Matrix operations are applied only to objects with appropriate dimensions. So, we have two advantages with this approach: (1) Matrix can be used with a wide variety of element types, and (2) all dimension checking can be done by the compiler – we will never have an exception thrown at runtime due to an attempt to perform an operation on matrices with incompatible dimensions.

The Matrix class template should provide the following operators:

1. a constructor that takes an optional value and initializes all of the matrix elements to that value; if the optional value is omitted from the call then the value should be the default value for T
   ```
   Matrix(const T & value = T()) {
   ```
2. a [] operator that takes an index i and returns a reference to i-th row of the matrix
3. a [] operator like the one above, but declared const and returning a reference to const
4. a unary + operator
5. a unary - operator
6. a += operator
7. a -= operator
8. a binary + operator
9. a binary - operator
10. a binary matrix multiplication operator template
    ```
    template<size_t otherCols>
    Matrix<T, rows, otherCols> operator*(const Matrix<T, cols, otherCols> & other) const {
    ```

11. a print operator

You should also provide a global stream output operator:

```
template<typename T, size_t rows, size_t cols>
ostream & operator<<(ostream & out, const Matrix<T, rows, cols> & m) {
```

("Global means that it is not declared as a member of the class.)

As mentioned above, the number of rows and number of columns are now *compile-time constants*. Hence, you do not need the numRows() and numCols() methods as members of the class. You can remove calls to those from the other methods and replace them with the compile-time constants *rows* and *cols*, respectively.

The file MatrixTest.cpp has been provided on Canvas to test your operations. The correct output from main.cpp is given below:

```
Testing Matrix<int,_,_>
m0:
 0 0 0 0 0
 0 0 0 0 0
m1:
 5 5
 5 5
m2:
 3 3
 3 3
m3:
 30 30
 30 30
m4:
 2 2
 2 2
 2 2
m4[1][1]: 2
m5[0][0]: 10
m5:
 10 1
 1 1
m6[0][0]: 22
m6:
 22 4
 22 4
 22 4
m4 + m6:
 24 6
 24 6
 24 6
m6 += m4:
 24 6
 24 6
```

```
 24 6
m6 -= m4:
 22 4
 22 4
 22 4
+m4:
 2 2
 2 2
 2 2
-m4:
 -2 -2
 -2 -2
 -2 -2

Testing Matrix<double,_,_>
mA:
 0.7 0.7 0.7
 0.7 0.7 0.7
 0.7 0.7 0.7
 0.7 0.7 0.7
mB:
 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8
 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8
 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8
mC:
 1.68 1.68 1.68 1.68 1.68 1.68 1.68 1.68
 1.68 1.68 1.68 1.68 1.68 1.68 1.68 1.68
 1.68 1.68 1.68 1.68 1.68 1.68 1.68 1.68
 1.68 1.68 1.68 1.68 1.68 1.68 1.68 1.68

Testing Matrix<complex,_,_>
mD:
 (1.1,-2.1) (1.1,-2.1) (1.1,-2.1) (1.1,-2.1)
 (1.1,-2.1) (1.1,-2.1) (1.1,-2.1) (1.1,-2.1)
 (1.1,-2.1) (1.1,-2.1) (1.1,-2.1) (1.1,-2.1)
mE:
 (0.2,3.5) (0.2,3.5)
 (0.2,3.5) (0.2,3.5)
 (0.2,3.5) (0.2,3.5)
 (0.2,3.5) (0.2,3.5)
mD * mE:
 (30.28,13.72) (30.28,13.72)
 (30.28,13.72) (30.28,13.72)
 (30.28,13.72) (30.28,13.72)

Testing Matrix<string,_,_>
m7:
    Hi Hello Hello Hello
 Hello    Hi Hello Hello
 Hello Hello    Hi Hello
```

How to submit your code for automated grading:
On Canvas, visit “Assignments”->”HW6: C++ template” and click the “Load HW6: C++ template” button at the bottom of the page. This will take you to Mimir. Upload your Matrix.h file and have Mimir test it. You can revise your program and resubmit if necessary.