Description:

The goal of this assignment was to implement a data structure that allows us to store data in a two-dimensional array called a matrix. This program declares and defines a class Matrix. The program defines how an object of Matrix is declared, defined, and is what operations are permitted to the object and its data it holds. In addition to the Matrix class we also had to make a templated version of the class to store any data type. The purpose of this programming assignment was to reinforce and recall most of the programing concepts we learned from CSCE 121 class. Moreover, this assignment tested our knowledge of classes, pointers, dynamic memory allocation, deallocation, constructors, overloaded functions and operators.

Data Structures and Algorithms:

A dynamic two-dimensional array is the main data structure used to define a matrix of integers or any data type for the templated version of the matrix class. The data structure had four private member variables: two integers pertaining to the number of rows and columns, a double pointer to the integer type or any data type (for templated class), and finally a function which dynamically allocates memory for matrix elements. These four characteristics were essential to implementation of the data structure. I implemented the structure by first getting the number of rows and columns then dynamically allocating memory for every element that would be contained in the matrix. All of these are operations are done by the defined constructors and helper functions.

Matrix operations were defined by overloaded operators and within each operator function I used algorithms to compute the necessary multiplication and addition operations for matrices. For the addition operator the algorithm first checked if both input matrices had the same number of rows

and columns if they did then it would go into a nested ‘for’ loop to compute every single element for the resulting matrix. If the number of rows and columns weren’t same for the two input matrices, then it would just throw an exception of incompatibility. After analyzing the algorithm, the time complexity is O(n*m), where n is the number of rows and m is the number of columns. The other algorithm I used was for multiplication. Here the function checks if the number of columns in the first matrix input is equal to the number of rows of the second matrix input, if it is then it goes into a three nested ‘for’ loop. If it is not, then it throws an exception. After analyzing the multiplication algorithm, the time complexity is $O(n^3)$. Considering the assignment requirements, these algorithms are sufficient in that the inputs won’t be very large matrices.

Organization and Implementation:

Implementation of the data structure was done by defining the class Matrix which is basically a dynamic two-dimensional array that stores data types. The matrix class has a default constructor, parameter constructor, destructor, copy constructor, move constructor, copy assignment, move assignment and other overloaded operator functions that are necessary to define operations for an object of the matrix class.

First the default constructor, initializes the number of rows and columns to zero and sets the double pointer to a nullptr. This is useful in declaring an object. Next, the parameter constructor initializes the number of rows and columns to their respective inputs and allocates memory for each element on the heap. The destructor then is used for deallocating object of the class matrix. The copy constructor makes a deep copy of the input matrix. Move constructor basically ‘steals’ the properties of the input matrix. Copy assignment, copies correctly, elements of the input matrix to a new matrix with the help of the copy constructor and deallocates memory for the old matrix to avoid memory leak and returns **this** object. Move assignment is like move constructor

in that it ‘steals’ representation of the input matrix and returns **this** object. Implementation of this constructor involve manipulating the pointers and making sure no memory leak occurred. All these constructors were necessary for the class to properly perform the intended actions when using overloaded operators.

Overloaded operators and helper functions (including setters and getters) used in this program were: output <<, input >>, set rows, set columns, call allocate memory, number of rows, number of columns, indexing (), and elem(). I implemented the indexing operators by using the fact that the double pointer, ‘ptr’, can access specific rows and elements by using the [] (bracket) operator already included in the std library. So, the I just returned the specified element or row. For the output operator I just formatted the out put and used a nested for loop to cycle through every element and printed it out. The setters would just set the matrix rows and columns to their respective input and the getters would only return the magnitude of rows and columns. Call allocate memory just called the private member to allocate new memory. For the input operator I first read in the matrix dimensions and then read each number after that I stored them in a vector to check against the number of elements that should be in the input file. If the number of elements was correct, then it would go back to the top of the file and read in the matrix element into the desired matrix, if the number of elements expected was wrong then it throws the invalid input exception.

One OOP concept I used when constructing the matrix class was overloading operators and functions. The concept of templates was also used in the implementation of the templated class matrix, giving the data structure the ability to hold all data types. Other than that, not much abstraction, encapsulation, or inheritance were used in the making of this program.

How to compile and run my program:

You will find that inside the tar file I have three folders, one for the report, one for part 1, and one for part 2. Note, that in part 1 and part 2 folders there are make files, and input files one of each in each folder. **To compile** simply go in the folder you want to run and type: "**make all**" then to run use command "**./main**".

That's all to run and compile. You should find that's its nicely formatted sectioned off by tests.

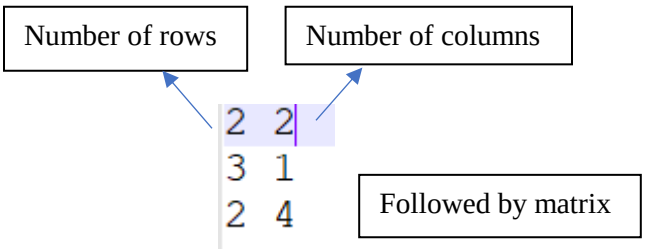
Input file:

Each folder should have an input file:

Part 1: "matrix.txt"

Part 2: "doubletypematrix.txt."

Format:



Possible crash inputs:

As any program, my program can crash due to erratic input. Such as wrong input file name, file didn't open, formatting of input file is wrong, and strings when expecting numbers.

Exceptions used:

Exceptions used for the class were 'invalid_input', 'incompatible_matrices', and 'out_of_range'. These exceptions were used to check input, check matrices for addition and multiplication operations and checking element indexing for returning a certain element. These exceptions are

used through out my program to prevent compilation errors and runtime errors. In the main.cpp I have commented with try blocks will throw an exception however, then program should run to completion unless the input files were not open so some reason.

Testing:

Part 1 testing:

Copy Constructor/Copy Assignment:

```
My_matrix m1(2,2);//Createc
m1(0,0)=3;//Initializing al
m1(0,1)=1;
m1(1,0)=4;
m1(1,1)=10;
cout<<"m1:\n";
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment1/Part1MatrixClass> (19:05:03 02/04/19)
:: ./main
Phase 1: Test 1
m1:
3 1
4 10

------------------------------------------------------------
```

```
My_matrix m3(m1);
My_matrix m4 = m1;
cout<<"Matrix m3:\n";
cout<<m3;
cout<<"Matrix m4:\n";
cout<<m4;
```

```
------------------------------------------------------------
Phase 1: Test 3
Matrix m3:
3 1
4 10
Matrix m4:
3 1
4 10
```

Input operator:

Valid

```
3 3
5 6 4
7 9 40
55 8 4
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment1/Part1MatrixClass> (18:08:43 02/04/19)
:: ./main
Phase 1: Test 2
Look inside the matrix.txt file and output file to check correctness
m2:
5 6 4
7 9 40
55 8 4
```

output1.txt

```
5 6 4
7 9 40
55 8 4
```

Invalid

```
3 1
q
q
q
```

```
[adriangamez]@linux2 ~/CSCE221/Assignment1/Part1MatrixClass> (18:11:23 02/04/19)
:: ./main
Phase 1: Test 2
Look inside the matrix.txt file and output file to check correctness
Matrix.txt file has different amount of elements than expected.
Error: Invalid matrix input
```

Some Matrices for testing

```
My_matrix m5(3,3);
m5(0,0)=1;
m5(0,1)=2;
m5(0,2)=3;
m5(1,0)=4;
m5(1,1)=5;
m5(1,2)=6;
m5(2,0)=7;
m5(2,1)=8;
m5(2,2)=9;
My_matrix m6(3,1);
m6(0,0)=1;
m6(1,0)=2;
m6(2,0)=3;
My_matrix m7(3,1);
m7(0,0)=4;
m7(1,0)=5;
m7(2,0)=6;
My_matrix m8(2,2);
m8(0,0)=7;
m8(0,1)=8;
m8(1,0)=9;
m8(1,1)=10;
My_matrix m9(2,2);
m9(0,0)=2;
m9(0,1)=4;
m9(1,0)=8;
m9(1,1)=16;
```

```
------------------------------------------------------------
Here are some Matrices to test Addition and Multiplication
Feel free to change them in the main.cpp file.
Matrix m5:
1 2 3
4 5 6
7 8 9
Matrix m6:
1
2
3
Matrix m7:
4
5
6
Matrix m8:
7 8
9 10
Matrix m9:
2 4
8 16
```

Addtion operator:

```
cout<<"Phase 1: Test 4 (multiplication)\n";
cout<<"m5*m6:\n";
My_matrix m10=m5*m6;
cout<<m10;
cout<<"Number of rows: "<<m10.number_of_rows()<<" ";
cout<<"Number of columns: "<<m10.number_of_columns();
cout<<endl;
cout<<"m8*m9:\n";
My_matrix m11=m8*m9;
cout<<m11;
cout<<"Number of rows: " <<m11.number_of_rows()<<" ";
cout<<"Number of columns: "<<m11.number_of_columns();
cout<<endl;
```

Valid

```
------------------------------------------------------------
Phase 1: Test 4 (multiplication)
m5*m6:
14
32
50
Number of rows: 3 Number of columns: 1
m8*m9:
78 156
98 196
Number of rows: 2 Number of columns: 2
```

```
try
{
//This is to test multiplication functionality with incompatible
//its supposed to throw an exception
cout<< "This is testing the multiplication functionality, this s
cout<<"m5*m8:\n";
cout<<m5*m8;//should throw exception becuase m5 is a (3x3) matri
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
```

Invalid

```
This is testing the multiplication functionality, this should throw exception
m5*m8:
Error: Incompatible matrices
------------------------------------------------
```

Multiplication operator:

```
cout<<"Phase 1: Test 5 (addition)\n";
cout<<"m6+m7:\n";
My_matrix m12= m6+m7;
cout<<m12;
cout<<"Number of rows: "<<m12.number_of_rows()<<" ";
cout<<"Number of columns: "<<m12.number_of_columns();
cout<<endl;
cout<<"m8+m9:\n";
My_matrix m13= m8+m9;
cout<<m13;
cout<<"Number of rows: "<<m13.number_of_rows()<<" ";
cout<<"Number of columns: "<<m13.number_of_columns();
cout<<endl;
```

Valid

```
-----------------------------------------------
Phase 1: Test 5 (addition)
m6+m7:
5
7
9
Number of rows: 3 Number of columns: 1
m8+m9:
9 12
17 26
Number of rows: 2 Number of columns: 2
```

```
try
{
//This try block is to test addition functionality with incompatibl
//This is supposed to throw an exception
cout<< "This is testing the addition functionality, this should thr
cout<<"m5+m8:\n";
cout<<m5+m8;
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
```

Invalid

```
This is testing the addition functionality, this should throw exception
m5+m8:
Error: Incompatible matrices
```

Part 2 testing:

Matrix declaration of different types:

```
cout<<"Phase 2: Test 1 with different data types\n";
cout<<endl;
TemplatedMy_matrix<double> m1_doubles(2,2);//Created object of My_matrix called m1;
m1_doubles(0,0)=2.1;//Initializing all possible elements
m1_doubles(0,1)=7.6;
m1_doubles(1,0)=3.9;
m1_doubles(1,1)=8.4;
cout<<"m1 with doubles:\n";
cout<<m1_doubles;
cout<<endl;
TemplatedMy_matrix<float> m1_float(2,2);/
m1_float(0,0)=1.134;//Initializing all po
m1_float(0,1)=73.623;
m1_float(1,0)=65.964;
m1_float(1,1)=3.445;
cout<<"m1 with floating point :\n";
cout<<m1_float;
cout<<endl;
TemplatedMy_matrix<char> m1_char(2,2);//C
m1_char(0,0)='d';//Initializing all possi
m1_char(0,1)='e';
m1_char(1,0)='q';
m1_char(1,1)='r';
cout<<"m1 with characters:(just to demons
cout<<m1_char;
```

Valid

```
[adriangamez]@linux2 ~/CSCE221/Assignment1/Part2TemplatedMatrixClass> (19:38:30 02/04/19)
:: ./main
Phase 2: Test 1 with different data types

m1 with doubles:
2.1 7.6
3.9 8.4

m1 with floating point :
1.134 73.623
65.964 3.445

m1 with characters:(just to demonstrate it works for all data types)
d e
q r
------------------------------------------------------------
```

Input Operator:

```
  cout<<"Phase 2: Test 2\n";
  cout<<"Look inside the matrix.txt file and output file to check correctness\n";
  TemplatedMy_matrix<double> m2;
try{
  const char * input_file= "doubletypematrix.txt";
  ifstream is(input_file);//opened input file
  if(!is)
  {
      cout<<"Unable to open file: "<<input_file<<endl;
      return 1;
  }
  ofstream outf("output2.txt");//opened output file
  if(!outf)
  {
      cout<<"Unable to open file: "<<outf<<endl;
      return 1;
  }
  is>>m2;//read in matrix
  cout<<"m2:\n";
  cout<<m2;//print to screen
  outf<<m2;//print to output file
  }
  catch(exception &error){
  cerr << "Error: " << error.what() << endl;
```

Valid

```
1  2 4
2  1.2 12.4 345.3 23.5
3  23.3 6.7 54.5 45.6
```

output2.txt

```
1  1.2 12.4 345.3 23.5
2  23.3 6.7 54.5 45.6
3
```

```
------------------------------------------------------------
Phase 2: Test 2
Look inside the matrix.txt file and output file to check correctness
m2:
1.2 12.4 345.3 23.5
23.3 6.7 54.5 45.6
------------------------------------------------------------
```

Invalid

```
1  1 3
2  1.2 12.4 345.3 23.5
```

```
------------------------------------------------------------
Phase 2: Test 2
Look inside the matrix.txt file and output file to check correctness
Matrix.txt file has different amount of elements than expected.
Error: Invalid matrix input
------------------------------------------------------------
```

Copy Constructor/Copy Assignment: Look back at m1 to compare:

```
cout<<"Phase 2: Test 3\n";
TemplatedMy_matrix<double> m3_doubles(m1_doubles);
TemplatedMy_matrix<float> m3_float(m1_float);
TemplatedMy_matrix<char> m3_char(m1_char);
TemplatedMy_matrix<double> m4_doubles=m1_doubles;
TemplatedMy_matrix<float> m4_float=m1_float;
TemplatedMy_matrix<char> m4_char=m1_char;
cout<<"Using copy constructor:\n";
cout<<"Matrix m3 (doubles):\n";
cout<<m3_doubles;
cout<<"Matrix m3 (floating point):\n";
cout<<m3_float;
cout<<"Matrix m3 (characters):\n";
cout<<m3_char;
cout<<endl;
cout<<"Using copy assignment:\n";
cout<<"Matrix m4(doubles):\n";
cout<<m4_doubles;
cout<<"Matrix m4(floating point):\n";
cout<<m4_float;
cout<<"Matrix m4(characters):\n";
cout<<m4_char;
cout<<"-----------------------------------------------
```

Valid

```
Phase 2: Test 3
Using copy constructor:
Matrix m3 (doubles):
2.1 7.6
100 8.4
Matrix m3 (floating point):
1.134 73.623
65.964 3.445
Matrix m3 (characters):
d e
q r

Using copy assignment:
Matrix m4(doubles):
2.1 7.6
100 8.4
Matrix m4(floating point):
1.134 73.623
65.964 3.445
Matrix m4(characters):
d e
q r
--------------------------------
```

Some matrices of doubles to test addition and multiplication:

```
cout<<"Here are some Matrices to test Addition and Multiplication (using double type):\n";
cout<<"Feel free to change them in the main.cpp file.\n";
//Created multiple matrices for testing addition and multiplication
TemplatedMy_matrix<double> m5(3,3);
m5(0,0)=1.5;
m5(0,1)=2.6;
m5(0,2)=3.7;
m5(1,0)=4.8;
m5(1,1)=5.9;
m5(1,2)=6.11;
m5(2,0)=7.1;
m5(2,1)=8.3;
m5(2,2)=9.4;
TemplatedMy_matrix<double> m6(3,1);
m6(0,0)=1.2;
m6(1,0)=2.3;
m6(2,0)=3.4;
TemplatedMy_matrix<double> m7(3,1);
m7(0,0)=4.3;
m7(1,0)=5.4;
m7(2,0)=6.5;
TemplatedMy_matrix<double> m8(2,2);
m8(0,0)=7.4;
m8(0,1)=8.6;
m8(1,0)=9.7;
m8(1,1)=10.12;
TemplatedMy_matrix<double> m9(2,2);
m9(0,0)=2.55;
m9(0,1)=4.67;
m9(1,0)=8.66;
m9(1,1)=16.3;
```

```
------------------------------------------------------------
Here are some Matrices to test Addition and Multiplication (using double type):
Feel free to change them in the main.cpp file.
Matrix m5:
1.5 2.6 3.7
4.8 5.9 6.11
7.1 8.3 9.4
Matrix m6:
1.2
2.3
3.4
Matrix m7:
4.3
5.4
6.5
Matrix m8:
7.4 8.6
9.7 10.12
Matrix m9:
2.55 4.67
8.66 16.3
------------------------------------------------------------
```

Multiplication:

```
cout<<"Phase 2: Test 4 (multiplication) using double matrices\n";
cout<<"m5*m6:\n";
TemplatedMy_matrix<double> m10=m5*m6;
cout<<m10;
cout<<"Number of rows: "<<m10.number_of_rows()<<" ";;
cout<<"Number of columns: "<<m10.number_of_columns();
cout<<endl;
cout<<endl;
cout<<"m8*m9:\n";
TemplatedMy_matrix<double> m11=m8*m9;
cout<<m11;
cout<<"Number of rows: " <<m11.number_of_rows()<<" ";
cout<<"Number of columns: "<<m11.number_of_columns();
cout<<endl;
```

Valid

```
Phase 2: Test 4 (multiplication) using double matrices
m5*m6:
20.36
40.104
59.57
Number of rows: 3 Number of columns: 1

m8*m9:
93.346 174.738
112.374 210.255
Number of rows: 2 Number of columns: 2
```

```
try
{
//This is to test multiplication functionality with incompatible r
//its supposed to throw an exception
cout<< "This is testing the multiplication functionality, this sh
cout<<"m5*m8:\n";
cout<<m5*m8;//should throw exception becuase m5 is a (3x3) matrix
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
```

Invalid

```
This is testing the multiplication functionality, this should throw exception
m5*m8:
Error: Incompatible matrices
------------------------------------------------
```

Addtion:

```
cout<<"Phase 2: Test 5 (addition, using double data type)\n";
cout<<"m6+m7:\n";
TemplatedMy_matrix<double> m12= m6+m7;
cout<<m12;
cout<<"Number of rows: "<<m12.number_of_rows()<<" ";
cout<<"Number of columns: "<<m12.number_of_columns();
cout<<endl;
cout<<endl;
cout<<"m8+m9:\n";
TemplatedMy_matrix<double> m13= m8+m9;
cout<<m13;
cout<<"Number of rows: "<<m13.number_of_rows()<<" ";
cout<<"Number of columns: "<<m13.number_of_columns();
cout<<endl;
```

valid

```
Phase 2: Test 5 (addition, using double data type)
m6+m7:
5.5
7.7
9.9
Number of rows: 3 Number of columns: 1

m8+m9:
9.95 13.27
18.36 26.42
Number of rows: 2 Number of columns: 2
```

```
try
{
//This try block is to test addition functionality with incompatib
//This is supposed to throw an exception
cout<< "This is testing the addition functionality, this should th
cout<<"m5+m8:\n";
cout<<m5+m8;
}
catch(exception &error){cerr<<"Error: "<<error.what()<<endl;}
cout<<"End of Part 2\n";
```

Invalid

```
This is testing the addition functionality, this should throw exception
m5+m8:
Error: Incompatible matrices
```