C++ Programming
Lab 5

- You may optionally work on lab 5 with one other person. If you are working alone, follow normal directions for uploading your files on your own. If you are working in a group of 2, please write BOTH names on the report and in the comments of the code. Only one student needs to upload code.

- Define a template class Matrix that behaves like a two-dimensional array. It is outlined here for you, but you'll have to do all the C++ specific implementation. Define each template class in a header file (.h file) with the same name as the class. Do not define a C++ file (.cpp file) for template classes. Define the methods inside the class declaration (not separately, but down in the .cpp file as we have been doing for non-template classes).

- (3 points) Convert the following class Array into a template where the element type is a template parameter. Define it in a file Array.h with the method definitions inside the class declaration. Be sure you know what all the iomanip objects do in the print function (and know how to use them). There may be a final exam question that depends on your understanding and knowing them.

```
#include <cassert>
class Array
{
private:
  int len;
  int * buf;
public:
  Array(int newLen)
    : len(newLen), buf(new int[newLen])
  {
  }
  Array(const Array & l)
    : len(l.len), buf(new int[l.len])
  {
    for (int i = 0; i < l.len; i++) {
      buf[i] = l.buf[i]; // note this is incorrect for pointers
    }
  }
  int length()
  {
    return len;
  }
  int & operator [] (int i)
  {
    assert(0 <= i && i < len);
    return buf[i];
  }
  void print(ostream & out)
  {
```

```
    for (int i = 0; i < len; i++) {
      out << setw(8) << setprecision(2) << fixed << right <<
    }
buf[i]; // #include <iomanip>
  }
  friend ostream & operator << (ostream & out, Array & a)
  {
    a.print(out);
    return out;
  }
  friend ostream & operator << (ostream & out, Array * ap)
  {
    ap->print(out);
    return out;
  }
  // note the overloading of operator << on a pointer as
well
};
```

- (3 points) Write the following class Matrix as a template where the element type is a template parameter. Put it entirely in Matrix.h similar to how you did for template class Array.

```
class Matrix
{
private:
  int rows, cols;
  // define m as an Array of Array pointers using the
  // template you defined above
public:
  Matrix(int newRows, int newCols)
    : rows(newRows), cols(newCols), m(rows)
  {
    for (int i = 0; i < rows; i++)
      m[i] = new Array <Element>(cols);
  }
  int numRows()
  {
    return rows;
  }
  int numCols()
  {
    return cols;
  }
  Array <Element> & operator [] (int row)
  {
    return * m[row];
  }
```

```
  void print(ostream & out)
  {
    // You can write this one too, but use the
Array::operator<<
  }
  friend ostream & operator << (ostream & out, Matrix & m)
  {
    m.print(out);
    return out;
  }
};
```

- (3 points) Get your Matrix to work with the following main (put it in main.cpp which includes Matrix.h). Test your matrix for at least two different element types (e.g., int and double). This code was not compiled, so there may be some errors. It will be good practice for you to eliminate them.

```
template
  <typename T>
void fillMatrix(Matrix <T> & m)
{
  int i, j;
  for (i = 0; i < m.numRows(); i++) {
    m[i][0] = T();
  }
  for (j = 0; j < m.numCols(); j++) {
    m[0][j] = T();
  }
  for (i = 1; i < m.numRows(); i++) {
    for (j = 1; j < m.numCols(); j++)
    {
      m[i][j] = T(i * j);
    }
  }
}
void test_int_matrix()
{ // here is a start, but make it better
    Matrix <int> m(10,5);
    fillMatrix(m);
    cout << m;
    // PUT YOUR TRY/CATCH HERE AND TEST EXCEPTIONS
}
void test_double_matrix()
{ // here is a start, but make it better
    Matrix <double> M(8,10);
    fillMatrix(M);
    cout << M;
    // PUT YOUR TRY/CATCH HERE AND TEST EXCEPTIONS
```

```
}
int main()
{
   test_int_matrix();
   test_double_matrix();
   return 0;
}
```

- (1 points) Define an exception, IndexOutOfBoundsException, and have your Array and Matrix class throw it if any index is out of bounds. Have your main testing functions (e.g., test_int_matrix() and test_double_matrix()) catch this exception and do something appropriate. Modify your main, by adding some illegal index after the last statement above, to cause this exception to be thrown.