C++ Programming
Homework 6

- Define a template class Matrix that behaves like a two dimmensional array. I'll outline it here for you, but you'll have to do all the C++ specific implementation.
- (35 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.

```
class Array
{
private:
  int len;
  int * buf;
public:
  Array( int newLen )
    : len( newLen ), buf( new int[newLen] )
  {
  }
  Array( Array & l )
    : len( l.len ), buf( new int[l.len] )
  {
    for ( int i = 0; i < l.len; i++ )
      buf[i] = l.buf[i];
  }
  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(5) << buf[i];
  }
  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
};
```

- (35 points) Write the following class Matrix as a template where the element type and the dimmensions are template parameters. Put it in Matrix.h similar to how you did for template class Array.

```
template
    < class Element >
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;
```

```
  }
};
```

- (20 points) Get your Matrix to work with the following main. Test your matrix for at least two different element types (such as int and double), but feel free to try something else. I wrote all this off the top of my head (without compiling), so there may be some errors, but it will be good practice for you to eliminate them.

```
template
  < class T >
void fillMatrix( const 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);
    }
}
int main()
{
  Matrix < int > m(10,5);
  fillMatrix( m );
  cout << m;
  Matrix < double > M(8,10);
  fillMatrix( M );
  cout << M;
}
```

- (10 points) Define an exception, IndexOutOfBoundsException, and have your Array and Matrix class throw it if any index is out of bounds. Have your main 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.