Assignment 2 deliverables are worth 5% of your overall course mark. No group work is permitted on this assignment.

Objectives
The purpose of this assignment is to:
1. Practice the "private implementation" idiom to implement an immutable ADT and determine how that affects the implementation decisions.
2. Practice using exceptions to deal with object construction errors and to make code more robust. Entails modifying existing code.
3. Practice writing an interface specification, abstraction function and a representation invariant for an existing ADT.

Due dates
The questions are due on Thursday, June 8, 2017 at 11:59pm. They are to be submitted electronically through Marmoset.

Provided files
Some starter code, sample data, and basic test harnesses for questions 1 and 2 have been provided in the ZIP archive, providedFiles.zip. Also provided are sample executables so that you can verify that your program output and behaviour matches ours. Note that the test harnesses are designed as "throw-away" code, and are not intended to be robust. Thus, your testing should focus on your own code and not on the harnesses.

Marking rubric
Programming questions will be marked for design quality and programming style as well as correctness. See the marking rubric for details on how your programs' design and programming style will be marked.

Q1 [70 marks] Pimpl Idiom
You are to implement an immutable Matrix ADT whose data representation is hidden within a nested, private structure. This technique is known as the "private implementation" idiom, also known as the "pimpl" idiom.

Submission
Submit your answer to Marmoset in a ZIP archive named a2q1.zip. It must be an archive of the files, not an archive of the directory containing the files. The archive must include the files:
- Makefile that creates the executable matrix,
- Matrix.{h,cc},
- and your implementation files (.h, .cc) for the private class that Matrix uses to actually implement the matrix.

Marking
The test scripts and marking scheme are structured according to the following increments.
1. Commands c, <, d, e
2. Commands C, =, >, M, m
3. Commands s, S, +, -, *

We strongly recommend that you implement each increment to completion before progressing to the next increment. The output of your program with be checked automatically using Marmoset—it must match exactly the output of our solution. You can check this by using the UNIX command diff to compare the output of your program against the output of the provided sampleMatrix executable.

Q2 [70 marks] Exceptions
You are to re-implement your solution to question 3 of assignment 1 (EmailAddress, Account and its derived classes, User, Collection, and Graph ADTs and the provided test harness) to be robust with
respect to input data, but not user commands i.e. you may still assume that user commands are entered correctly in terms of syntax.

**Submission**
Submit your answer to Marmoset in a ZIP archive named a2q2.zip. It must be an archive of the files, not an archive of the directory containing the files. The archive must include the files:

- Makefile,
- EmailAddress.{h,cc},
- Collection.{h,cc},
- Graph.{h,cc},
- NetworkTestHarness.cc
- User.{h,cc},
- Account.{h,cc} and
- {Twitter,Reddit,Instagram,YouTube}Account.{h,cc}.
- util.{h,cc} should only be submitted if you have added your own helper routines. You may not change the existing code, though, since the test harness relies upon it.

You need to make sure that your code runs on the undergraduate environment (e.g., that your programming environment does not implicitly include libraries for you, or that your compiler is implemented differently than what exists in the student environment).

**Marking**
The test scripts and marking scheme are structured according to the same increments as in assignment 1.

1. Commands: g, i, p, a
2. Commands: +, -, f
3. Commands: e, d, k
4. Commands: c, C, =, <
5. Command: ?

We strongly recommend that you implement each increment to completion (for multiple types of each account) before progressing to the next increment. The output of your program with be checked automatically using Marmoset—it must match exactly the output of our solution. You can check this by using the UNIX command diff to compare the output of your program against the output of the provided sampleRobustNetwork executable.

**Q3 [30 marks] Interface Specification, Abstraction Function, Representation Invariant**
You are to provide an interface specification, abstraction function, and representation invariant for the Collection ADT from your answer to question Q2.

**Submission**
Submit your answer to Marmoset in a ZIP archive named a2q3.zip. It must be an archive of the files, not an archive of the directory containing the files. The archive must include:

- Collection.h, Collection.cc from your solution to question Q2, and
- Spec.txt with your solution to question Q3.