Object-Oriented Programming in C++
Matrix: Assignment 1 (10%)

Objective

The objective of this assignment is to get practice in programming classes that use internal dynamic memory allocation, but yet behave well (e.g., are exception safe and use appropriately defined constructors, copy constructor, assignment operator and destructor). Also, you will get practice in overloading operators and working with files. You can work on the assignment in a group of two people (strongly recommended to work in a group rather than alone!).

Description

A header file *matrix.h* is provided. It shows the interface to the Matrix class as well as its internal data member declarations. You are not allowed to change this file. The matrix class provides common arithmetic operations on matrices as well as methods for input and output.

An example use of the matrix class is shown below:

```
//     1 2 3    1
//     4 5 6    2
//              3
Matrix m(2, 3), n(3,1);

m(0,0) = 1; m(0,1) = 2; m(0,2) = 3;
m(1,0) = 4; m(1,1) = 5; m(1,2) = 6;

n(0,0) = 1;
n(1,0) = 2;
n(2,0) = 3;

Matrix tm = n.transpose();  // becomes [ 1 2 3 ]
Matrix mm = 2 * (m + m);

cout << n(1,0) << endl;
cout << mm << endl;
```

output:

```
2
 4  8  12
 16  20  24
```

Part I

In a file *matrix.cpp*, implement the missing functionality of the Matrix class. Where appropriate ensure that the matrix dimensions are legitimate for the operation and, if not, throw the exception *matrix_dimensions_wrong* (e.g. matrix addition requires the two matrices to be of the same size). Also throw an *out_of_range* exception if accessing a matrix with an invalid index. Test your implementation thoroughly before proceeding to part II.

Part II

Internet based search engines that look for flight information between two or more destinations are now becoming increasingly common place. A user can typically specify in a search query whether he or she is interested in direct flights

only or also connection flights, in which case the allowed maximum number of connections is specified (e.g., 1 or 2).

One can formulate such a query as a search in a *directed graph*, where the nodes indicate airports and an edge between two nodes indicates that there is a flight from the source to the destination. One way to represent a directed graph is with an *adjacency matrix*. If we have a graph with n nodes its adjacency matrix will be of size $n \times n$, where a matrix entry at location (i, j) will be 1 if there is a directed edge from node i to node j, but otherwise 0.

One can then compute *reachability* in the graph --- that is, whether one can reach a node in the graph from another node in some specified number of steps --- by using matrix multiplication on the graph's adjacency matrix. For example, in an adjacency matrix A each entry indicates whether a length one path (direct flight route) exists between i and j; in A^2 ($A \times A$) each entry specifies the number of different paths of length two; in A^3 ($A \times A \times A$) each entry specifies the number of paths of length three, etc. For example, if one wants to find the number of routes with *at most* two connections (three flights) one would compute $A + A^2 + A^3$. [*]

Write a program *main.cpp* that takes as a command line argument the name of a file containing information about direct flights (see file format below). A second optional command-line argument gives the name of an output file. If it is provided the program: computes the full reachability of the graph, that is, the number of paths between any two airports consisting of up to *n-1* connections; writes the reachability matrix to a file (named as indicated by the second command-line argument); and then quit. On the other hand, if only one command-line argument is provided, the program prompts the user (*cin*) for a name of a source and a destination airport and the maximum number of connections allowed and then displays how many flights paths there are between those two airports requiring no more than the indicated number of connections (for simplicity, there is no need not list the routes, just how many they are). The user will be prompted repeatedly until the input `quit` is entered.

The input (and out ouput) file is formatted such that on the first line is a list of the names of the airports (space separated). The number of listed airports, n, will determine the size of the adjacency matrix ($n \times n$), which is listed next one row at a time. A 1 entry means there is a (one-way) direct flight between the two airports and 0 entry that there isn't. An example entry is shown below:

```
KEF OSL BGO
0 1 0
1 0 1
1 1 0
```

shows direct flight connections between three airports (Keflavik, Oslo, and Bergen). There are direct flights from: KEF → OSL, OSL → KEF, OSL → BGO, BGO → OSL and BGO → KEF (note, the flights connections need not necessarily be symmetrical).

Submit via MySchool your *matrix.cpp* file (from part I) and your *main.cpp* (from part II). The submission deadline is Monday May 1st, 11 pm.

Good luck! ☺

---

[*] For further explanation, see e.g.:
http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/L16-ConnectedGraphs.htm