Q3 [100 marks] Special Member Functions & Pointers
You are to complete the implementation of an ADT for dynamically-sized graphs. A Graph object is a set of nodes containing Users, and connections (edges) between them where an edge is annotated by how the users know about each other i.e. through which social network site. In addition to the expected operations for adding/removing/finding nodes and the five special member functions, you need to implement an operation for finding simple paths in a graph. This will let us determine whether or not two users are connected to each other through their social network accounts, and which ones.

You will need to make two changes to your code from question 2 to implement this question, so place a copy of your Q2 account code into your directory for Q3.
1. Replace the string email address from the constructor of the Account class and its derived classes with a (const User*).
2. Add the method virtual Graph::Connector accountType() const; to each of your account classes. See the provided Graph.h file for a description of the internal enumerated type, Graph::Connector, which is used to specify the type of social network site through which the connection is made.

Objective
We have specified the public interface for the Graph ADT, which you will complete and implement. You are to also define and implement ADTs for EmailAddress, User, and Collection as described in this assignment's question 1 specification and in the provided code. You may add private/protected information and the necessary accessors/mutators; however, you may not change the provided public methods or their signatures since the test harness relies upon their presence. The only allowed friends to your classes are the overloaded output operators for the Graph and its implementation classes. You may use STL containers for Collection, but your Graph must be solely implemented using pointers (no arrays, no STL containers) to ensure you remain practiced with dynamic memory manipulation. The Graph ADT must provide appropriate definitions for:
- default constructor
- copy constructor
- move constructor
- destructor
- assignment operator
- assignment move operator
- equality operator

Provided Files
1. The test harness that handles simple user I/O has been provided in the file NetworkTestHarness.cc. It will be used to create various types of account objects and perform simple operations upon them. It is intentionally not robust since it has been designed as "throwaway" code. Invalid commands may (or may not) cause the program to terminate or misbehave. You are not expected to test it with invalid input. The test harness also uses the Boost Gregorian date class and the provided util.{h,cc} files. Do not submit with your solution.
2. Some common, useful code has been provided in util.{h,cc}. You may not remove or change the existing code. If you don't change anything, don't submit these files with your solution. You may add helper routines of your own to the file if they're used across multiple classes, and not appropriate to include in one of the classes you're implementing. If you add anything to util.{h,cc}, then you must submit your version with your question solution.
3. You have been provided with a skeleton of the User, EmailAddress, and Collection classes in the files User.h, EmailAddress.h and Collection.h respectively. All implementation code must be placed in the appropriate .cc file.

4. A Makefile has been provided that will link in the Boost and JSON C++ libraries. Feel free to modify it or use your own. The only restriction is that it must create the executable network when the command make is issued. Your solution must include a Makefile that functions as described.

5. You have also been provided with sample JSON files for the user information as well as a sample Graph initialization file.

6. A version of our solution set, compiled in the linux.student.cs.uwaterloo.ca environment (ubuntu1604-NNN), sampleNetwork, has been provided so that you can compare your program's output to ours.

**Execution**

When the test harness initially starts running, it creates an initially empty dictionary as a Collection, five initially empty Graph objects (identified as 0 to 4), sets the current graph pointer to nullptr, prints a welcome message and asks for the first command from the user. Unless otherwise noted, all test harness commands apply to the graph specified by the most recent g command as described below. **For this assignment, you may assume that all input is valid and correctly formed.** There will be no users with duplicate email addresses. Edges are added to existing graph nodes. Only existing edges, nodes and users are removed. There will be zero or one connection between each pair of users. (Assignment 2 will add appropriate error checking and handling via exceptions.)

There are 15 valid commands that the test harness recognizes:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g id</td>
<td>Sets the current graph to point to one of the existing graphs, with id 0-4.</td>
</tr>
<tr>
<td>i file-name</td>
<td>Reads the JSON data from the specified file to create a series of 0 or more User and {Twitter,Reddit,YouTube,Instagram}Account objects. Users are added to the dictionary if not already present. For the specified followers, not only adds them as friends/followers to the appropriate accounts, but adds the appropriate edge between the associated Users to the current graph, after adding the user to the graph if it was not already in the graph. This implies the command can be used to either initialize or add to the current graph. Relies upon the validateJSON method being implemented for the User class and each derived Account class. Also requires that the EmailAddress::validateEmailAddress method be implemented, though for now it can simply return true.</td>
</tr>
<tr>
<td>+ id1 id2 &lt;conn&gt;</td>
<td>Adds the social network site account of type conn, i.e. one of Instagram/Reddit/Twitter/YouTube, identified by the id id1 as a follower of the account identified by id1, and the account identified by id2 is added as a friend of the account identified by id id1. Both ids must exist. id1 must not equal id2. Also adds the appropriate edge between the associated users in the current graph. There is no output associated with this command.</td>
</tr>
<tr>
<td>- id1 id2 &lt;conn&gt;</td>
<td>Removes the social network site account of type conn, i.e. one of Instagram/Reddit/Twitter/YouTube, identified by the id id1 as a follower of the account identified by id1, and the account identified by id2 is removed as a friend of the account identified by id id1. Both ids must exist. id1 must not equal id2. If the ids are not unique, it uses the first one found. This also removes the existing edge between the two users in the current graph. There is no output associated with this command.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>a id</td>
<td>Adds a node to the current graph for the user whose <strong>EmailAddress</strong> matches the string <code>id</code>. There is no output associated with this command.</td>
</tr>
<tr>
<td>f id</td>
<td>Finds a user within the current graph (not within the collection) whose <strong>EmailAddress</strong> matches the string <code>id</code>. If the user is found, the test harness prints the user information in the following format: &lt;email address&gt;\t&lt;user name&gt;\n i.e. the email address, a tab character ('\t'), the user name, and then a newline character ('\n'). (Note that the angle brackets don't actually appear in the output, they're just a notational convenience.) If the user is not found, an error message is printed (don't worry about its format).</td>
</tr>
<tr>
<td>p</td>
<td>Prints the current graph to the screen using the overloaded output operator. The output has the following format: &lt;email address&gt;\t&lt;user name&gt;\n \tConnects to: &lt;id1&gt; (connector), ...\n \n &lt;email address&gt;\t&lt;user name&gt;\n \tConnects to: &lt;id1&gt; (connector), ...\n \n ...</td>
</tr>
<tr>
<td>d id</td>
<td>Finds and deletes the user within the current graph (but does not modify the collection) whose <strong>EmailAddress</strong> matches the string <code>id</code>. If there is a node in the graph for that user, then it and all adjacent edges are removed. There is no output associated with the command.</td>
</tr>
<tr>
<td>k id</td>
<td>Removes the user whose <strong>EmailAddress</strong> matches the string <code>id</code> from the collection. All nodes in all of the graphs that contain the user, and all of their associated edges, are removed. There is no output associated with the command.</td>
</tr>
<tr>
<td>c id1 id2</td>
<td>Uses the <strong>copy constructor</strong> to initialize the graph with <code>id1</code> to the contents of the graph with <code>id2</code>. If the graph with <code>id1</code> previously was initialized, its contents are deleted beforehand. Prints the contents of the graph with <code>id1</code> using the same format as the p command.</td>
</tr>
</tbody>
</table>
| C id1 id2 | Uses the **move constructor** to initialize the graph with `id1` to the contents of the graph `id2`. Graph `id2` will be left empty afterwards. If the graph with `id1`
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`previously was initialized, its contents were deleted beforehand. Prints the contents of the graph with id1 using the same format as the p command.</td>
<td></td>
</tr>
<tr>
<td><code>=</code> id1 id2</td>
<td>Uses the assignment copy operator to replace the contents of the graph with id1 with a copy of the contents of the graph with id2. Prints the contents of the graph with id1 using the same format as the p command.</td>
</tr>
<tr>
<td><code>&lt;</code> id1 id2</td>
<td>Uses the assignment move operator to exchange the contents of the graph with id1 with the contents of the graph with id2. There is no output associated with the command.</td>
</tr>
<tr>
<td><code>e</code> id1 id2</td>
<td>Uses the equality operator to compare the contents of the graph with id1 with the contents of the graph with id2. The test harness prints the appropriate message based upon the result. (You don't need to worry about the format.)</td>
</tr>
<tr>
<td><code>?</code> id1 id2 &lt;all&gt;</td>
<td>Searches the current graph to see if there is a simple path (no cycles) between the two users identified by the email addresses id1 and id2. The path is printed if found. The boolean value all is used to determine if either a single path (if false) or all paths (if true—not a requirement, see below) are output. The output has the following format:</td>
</tr>
</tbody>
</table>
| Paths from id1 to id2 are: | \tid1 (<connector>) -> <id> (<connector>) -> … -> id2
|
| You do not have to worry about the format of the header, which is output by the test harness. A path consists of a tab character (\t) followed by the email address of the start of the path, id1, followed by a sequence of zero or more <connector>s, email addresses, and arrows ("-\") that correspond to the sequence of edges that make up the found path, ending with the email address of the end of the path, id2. If id1 and id2 are the same, the printed “path” is simply the starting email address. The output ends with a newline character, \n. |
| For extra credit, you could implement a search that finds all simple paths from the graph node associated with id1 to the graph node associated with id2. If all has value true, then the paths are printed one per line. The output ends with a blank line. |

| Ctrl-d | Terminates the program by simulating end-of-file. |

**Sample execution**

Below is a sample partial execution. User input is shown in **bold** font.

Enter command: g 0
Enter command: i data/q3-users.json
Enter command: a kameron.hurley@fake.email.com
Enter command: p
Graph 0
kameron.hurley@fake.email.com Kameron Hurley

Enter command: g 1
Enter command: a wfish@gmail.com
Enter command: a cheshire.cat@yahoo.com
Enter command: a kameron.hurley@fake.email.com
Enter command: a jspratt@gmail.com
Enter command: a tnext@gmail.com
Enter command: + xCd123 23uj1e Reddit
Enter command: + 1671492 9294396 Instagram
Enter command: + 1671492 9404238 Instagram
Enter command: + 9294396 9404238 Instagram
Enter command: p

Graph 1
cheshire.cat@yahoo.com Unitary Authority of Warrington Cat
         Connects to: jspratt@gmail.com (Instagram), tnext@gmail.com (Instagram), wfish@gmail.com (Reddit)

jspratt@gmail.com Jack Spratt
         Connects to: tnext@gmail.com (Instagram), cheshire.cat@yahoo.com (Instagram)

kameron.hurley@fake.email.com Kameron Hurley
         Connects to: tnext@gmail.com (Instagram), cheshire.cat@yahoo.com (Instagram)
wfish@gmail.com Wanda Fish
         Connects to: cheshire.cat@yahoo.com (Reddit)

Enter command: g 2
Enter command: p

Graph 2
Enter command: e 0 1
Graphs 0 and 1 are NOT equal.
         Enter command: < 0 2
         Enter command: g 0
         Enter command: p

Graph 0
Enter command: g 2
Enter command: p

Graph 2
kameron.hurley@fake.email.com Kameron Hurley

Enter command: = 0 2
Graph 0
kameron.hurley@fake.email.com Kameron Hurley

Enter command: g 2
Enter command: a alt-potus45@fake.email.com
Enter command: + 817269572998860800 27869564 Twitter
Enter command: p

Graph 2
alt-potus45@fake.email.com Alt-POTUS 45
         Connects to: kameron.hurley@fake.email.com (Twitter)

kameron.hurley@fake.email.com Kameron Hurley
         Connects to: alt-potus45@fake.email.com (Twitter)

Enter command: ^D