Q2 [70 marks] Exceptions
You are to revise your ADTs from assignment 1, question 3 to be robust i.e. to recover from bad input values. You should not assume that any error checking performed in the provided test harness will be performed by the client code or in our Marmoset tests. Thus, in this assignment question:

• The EmailAddress ADT ensures that the string follows the form local-part@domain\(^1\). To keep things simple, we'll restrict the email address to a total of 255 characters, including the '@' symbol, and the strings to alphanumeric characters plus the period ('.') and the hyphen ('-'). There are no restrictions on the quantity of periods or hyphens that may occur in either string, and the only restriction on placement of the period or hyphen is that it may not be the first character i.e. the first character of both the local-part and the domain must be alphanumeric. Neither the local-part nor the domain may consist of an empty string. If any of these conditions are not met, it throws an exception.

• The Collection ADT represents a set of unique User accounts in program i.e. the EmailAddress used as a key for each user must be unique in the Collection. There may be more than one collection, and their contents may overlap. Any attempt to add or remove a (User*) that is a nullptr raises an exception. Deleting a non-existent User also raises an exception.

• The Graph ADT ensures that new graph nodes refer only to users in the given collection i.e. (User*) must not be nullptr, that new graph edges connect existing nodes, and that paths-to-be-computed are between existing nodes; otherwise, it throws an exception. The Graph ADT also throws an exception if asked to add a node or edge that already exists, to add an edge with an unknown/bad connector type, or to remove a node or edge that does not exist.

Objective
The objective is to practice throwing, catching and recovering from exceptions. You are to modify part of your solution to assignment 1, question 3 to:

• revise your implementations of the ADTs to throw exceptions when given bad input data (the validity of input data should be checked only by the ADT that uses the data);
• revise the test harness to catch thrown exceptions and to recover from them; only the client code (test harness) should recover from thrown exceptions—if an ADT catches an exception, it should only be to clean up resources (and re-throw the exception) or to throw a more informative exception.

Think carefully about how and where to define and structure your exceptions and the handler(s). You may find it helpful to overload the Graph add and remove edge methods to take two additional string parameters so that the email address information is available for the exception. You shouldn't need to change anything else in the public interface.

The expectation is that you will simply extend your solution to assignment 1, question 3. However, there is no restriction on the use of STL containers or smart pointers in your solution to this question, so you are free to reimplement some or all of your code from assignment 1.

IMPORTANT: The test cases will focus mainly on invalid input data, but there will be some tests to check that your program continues to work after your modifications. During our tests, we will be using different test harnesses to perform unit testing on your ADTs in addition to testing your system as a whole.

\(^1\) Described by the \texttt{addr-spec} field defined in \texttt{RFC 5322} if you want to know the details.
\(^2\) Alphanumeric characters consist of: \texttt{a-z, A-Z, and 0-9}. 
Provided files
You are free to use the test harness code that we provided in assignment 1 as the starting point for the test harness in your solution to this question. We are not providing any additional code for this question.

Execution
There are 15 valid commands that the test harness recognizes. They are the same commands with the same syntax used in assignment 1. Input to your program will adhere to the correct syntax in terms of command types, number of parameters, and parameter types. When a command includes expected input, such as adding a node to the graph for a user, your program should behave as specified in assignment 1. When an error is detected and an exception is raised, the processing of the command must be abandoned and the specified error message must be output to the console/standard output. We will thus specify how the program is to behave under invalid input, where applicable.

When adding an edge, if there are multiple errors, only one message is produced. First priority is a message about reference to an unknown user, second priority is a message about an edge from a node to itself, third priority is a non-existent connector type, and last priority is a message about an already existing edge.

<table>
<thead>
<tr>
<th>ADT</th>
<th>Type of error</th>
<th>Description</th>
</tr>
</thead>
</table>
| EmailAddress | Invalid email address | If an email address does not have a valid format, the program outputs an error message to the console/standard output that explains all of the ways in which it must be changed to comply with the required format, in order:  
• have a length greater than 0 and less than 255 characters,  
• contain an '@' symbol,  
• contain only one '@' symbol,  
• apart from the '@' symbol, hyphens ('-'), and periods ('.'), contains only alphanumeric characters,  
• length of the local-part must be greater than 0 characters,  
• length of the domain part must be greater than 0 characters,  
• each of the local part and the domain must start with an alphanumeric character  
Some of the conditions are mutually exclusive to each other, so not all messages are possible in combination.

```
\nERROR: Email address "<address>" has an invalid format.\n  \t- 0 < length < 255\n  \t- must have an '@'\n  \t- must have only one '@'\n  \t- contains invalid characters\n  \t- local part must not be empty\n  \t- domain part must not be empty\n  \t- local part must start with an alphanumeric\n  \t- domain part must start with an alphanumeric\n```

3 If there is more than one '@' symbol, then the first one is taken as the delimiter of the local part, and the rest of the string, excluding the first '@' symbol, is taken as the domain part.
<table>
<thead>
<tr>
<th>Collection</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-unique email addresses.</td>
<td>\nERROR: Email address &quot;&lt;address&gt;&quot; is already in use. \n</td>
</tr>
<tr>
<td>Remove non-existent user.</td>
<td>\nERROR: There is no user with email address &quot;&lt;address&gt;&quot; in the collection. \n</td>
</tr>
<tr>
<td>User parameter is a nullptr.</td>
<td>\nERROR: User is a nullptr. \n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>One or both users specified when adding /removing an edge, or finding a path, don't exist as nodes in the current graph.</td>
<td>If there is no node in the graph for both users, then the error message reports only that there is no node for id1. Otherwise, it reports the appropriate id. \nERROR: There is no user &quot;&lt;id1&gt;&quot; in the graph. \nThis command results in no changes to graph&lt;num&gt;. \n</td>
</tr>
<tr>
<td>Tried to add an edge between the same user i.e. id1 == id2.</td>
<td>\nERROR: Cannot connect node &quot;&lt;id1&gt;&quot; to itself. \nThis command results in no changes to graph&lt;num&gt;. \n</td>
</tr>
<tr>
<td>Edge already exists between two users when tries to add an edge, even if different connector type.</td>
<td>\nERROR: There is already an edge between &quot;&lt;id1&gt;&quot; and &quot;&lt;id2&gt;&quot; in the graph. \nThis command results in no changes to graph&lt;num&gt;. \n</td>
</tr>
<tr>
<td>Tried to remove an edge of specified type but no edge exists.</td>
<td>\nERROR: There is no edge between &quot;&lt;id1&gt;&quot; and &quot;&lt;id2&gt;&quot; in the graph to be removed. \nThis command results in no changes to graph&lt;num&gt;. \n</td>
</tr>
<tr>
<td>Tries to add a node for the user with the specified id, but the current graph already contains a node for the user.</td>
<td>\nERROR: There is already a node for user with email address &quot;&lt;id&gt;&quot; in the graph. \nThis command results in no changes to the graph&lt;num&gt;. \n</td>
</tr>
</tbody>
</table>

**Sample execution**

Below is a sample partial execution. User input is shown in **bold** font. Error messages are shown in red.

```
Enter command: g 0
Enter command: i data/q3.json
Enter command: ? scamperbeasts@fake.email.com nnedi.okorafor@fake.email.com false
Paths from scamperbeasts@fake.email.com to nnedi.okorafor@fake.email.com are: scamperbeasts@fake.email.com (Twitter) -> scalzi@fake.email.com (Twitter) -> kameron.hurley@fake.email.com (Twitter) -> nnedi.okorafor@fake.email.com

Enter command: d scalzi@fake.email.com
Enter command: + 14202817 4625687418 Twitter
ERROR: There is no user "scalzi@fake.email.com" in the graph. This command results in no changes to graph0.

Enter command: - 14202817 4625687418 Twitter
ERROR: There is no user "scalzi@fake.email.com" in the graph. This command results in no changes to graph0.

Enter command: ? scalzi@fake.email.com kylo.r3n@fake.email.com false
```
Paths from scalzi@fake.email.com to kylo.r3n@fake.email.com are:

ERROR: There is no user "scalzi@fake.email.com" in the graph.
This command results in no changes to graph0.
Enter command: + 14202817 14202817 Twitter

ERROR: There is no user "scalzi@fake.email.com" in the graph.
ERROR: Cannot connect node "scalzi@fake.email.com" to itself.
This command results in no changes to graph0.
Enter command: ^D