CS211 – Programming Practicum Fall 2020

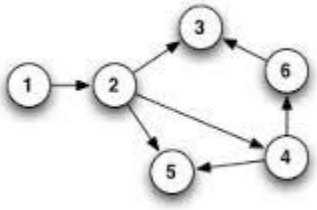
Programming Project 5

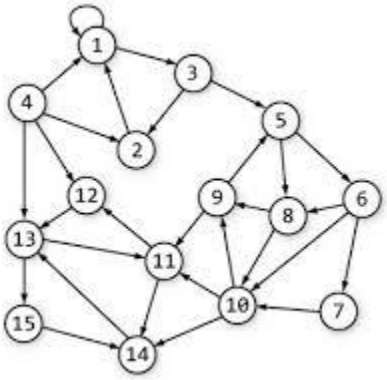
Due: Friday, 11/13/20 at 11:59 pm

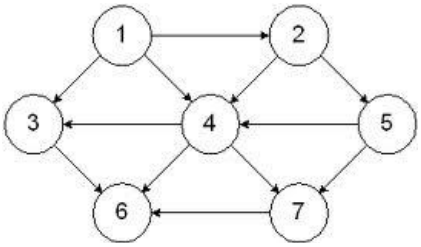
Archipelago Expedition Planning

For this program, you will write a **C++ Program** to represent an archipelago expedition. This expedition will use an array of linked lists as its primary storage structure. This type of storage structure is typically called an "adjacency list".

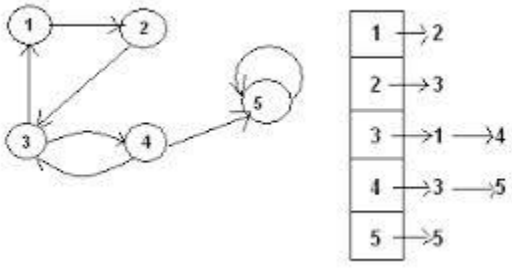
Welcome to the paradise! You have traveled to a beautiful archipelago in the Pacific Ocean and you are planning your great expedition. These group of remote islands are accessible with ferry rides. Each island has a unique number. If there's a ferry ride from Island X to Island Y, the network depicting the archipelago will have an "edge" from X to Y. Below are a number of drawings representing this idea. Think about this drawing as maps available in the tavern of each island that will inform all tourists how to move between islands and plan their expedition. The islands are represented by a circle and the edges are represented by arrows. Consider the first drawing, this archipelago consists of 6 islands. It has a ferry ride between island 1 to island 2. Island 3 has three ferry rides going to island 3, island 4, and island 5. There are no ferry rides out of island 3 or island 5 so you have to be careful, you can get stuck there. Legend has it, the islands are so beautiful no one wants to leave after arriving (or maybe there's a monster eating newly arrived tourists, I leave this to your imagination). There are two ferry rides leaving island 4 for island 5 and island 6. Finally, there's one ferry ride leaving island 6 for island 3.







In an adjacency list, each location/island needs a list of those locations/islands that one can get to in one move/ferry ride. In this program, we need a list for each island. If the archipelago has N islands, the array will have N linked lists, one for each island. If island X has three ferry rides to three different islands, the linked list for island X would have 3 nodes. Consider the following image showing an archipelago network and an adjacency list:



There are 5 islands, so we have an array of 5 linked lists. Since Island 3 has ferry rides to two Islands, namely Island 1 and Island 4, the linked list for Island 3 has two nodes: one node containing the value 1; another node containing the value 4.

Program Input and Commands

The input for the operations will come from standard input and from files. The input will initially come from standard input. If the user specified the f command, your program will then read input from a file. See the description below for more details. The commands are to follow the descriptions given below. Note: that the form <int> could be any integer number and it will NOT be enclosed in angle brackets. <int> is just a notation to specify and integer value. The integer value is to be input on the same line as the command character. If the first character on the line is not one of the following characters, print an error message and ignore the rest of the information on that line.

q - quit the program immediately.

? - display a list of the commands the user can enter for the program.

- ignore this line of input. Treat the line of input as a comment

t <int1> <int2> - display a message stating whether a person can go from island **<int1>** to island **<int2>** with one or more ferry rides.

r <int> - remove all values from the archipelago expedition and resize the array to contain the number of islands as indicated by the given integer value. The value of the integer must be greater than zero. **The islands will be numbered from 1 to the given integer value.** Be sure to deallocate all unused memory as part of this command.

i <int1> <int2> - insert the edge to indicate a ferry ride from island **<int1>** to island **<int2>**.

d <int1> <int2> - delete the edge that indicates a ferry ride from island**<int1>** to island **<int2>**.

l - list all the items contained in the archipelago expedition. First display all of the islands (if any) that can be reached from the island #1 with one ferry ride (that have an edge in the archipelago), followed by all the islands (if any) that can be reached from island #2 with one ferry ride, etc.

f <filename> - open the file indicated by the **<filename>** (assume it is in the current directory) and read commands from this file. When the end of the file is reached, continue reading commands from previous input source. This must be handled using recursion. Beware of a possible case of an infinite recursive loop, the **f** command is may not call a file that is currently in use.

Initially your program should have the array to hold 10 islands (numbered 1 to 10). If a command specifies an island outside of the current valid range, print an error message and ignore the command.

The code given in proj5Base.cpp should provide the basics on reading input. You will need to do some cutting and pasting of code to read in input for all commands. The input is properly read in for the t and f commands. It is assumed that you can determine the code for the rest of the commands by looking at the code those these two commands.

Expedition Algorithm and the Island Object

To determine if a person can go from island X to island Y by taking one or multiple ferries, a recursive depth-first-search algorithm must be used. For this algorithm to work, we will need to be able to mark each island as visited. Setting up an Island class is required. This object will contain all of the data that one island knows (the head of the linked list for the island's adjacency list and the object will also contain a Boolean value to determine if an island has been visited or not) and the methods to use that data. The Archipelago Expedition **MUST** be a dynamic array of these Island objects. The adjacency list will also need a Node class/object to store the linked list information.

The pseudo code for this algorithm is as shown below. Note it is valid to ask, can I to go from Island X to Island X in one or more ferry rides. It really asks, "If I leave island X, can I return to it?" This algorithm is recursive and you MUST use this recursive algorithm in your program.

```
void depthFirstSearchHelper (int x, int y)
{
  mark all islands as unvisited;
  if ( dfs (x, y) = = TRUE)
     print ("You can get from island " + x + " to island " + y + " in one or more ferry rides");
  else
     print ("You can NOT get from island " + x + " to island " + y + " in one or more ferry
rides");
}

 bool dfs (int a, int b)

{
  for (each island c that can be reached from a in one ferry ride)
      {
        if (c = = b)
            return TRUE;
        if ( island c is unvisited )
           {
             mark island c as visited;
             if ( dfs (c, b) = = TRUE )
                  return TRUE;
           }
       }
  return FALSE;
}
```

The FILE Command: f

The f command may seem difficult to implement at first, but it has a creative solution that you are to use. The code in the file Proj5Base.cpp is intended to give you an idea on how this solution is to be implemented.

First note that main( ), is extremely short. It just creates an instance of the ArchipelagoExpedition class and calls the processCommandLoop( ) method with an value of FILE* that reads from standard input.

The method processCommandLoop( ) reads from the input source specified by the parameter and determines the which command is being invoked.

When the f command is invoked, it is to open the file specified by the command, create a new instance of FILE* that reads from this file. Then make a recursive call to processCommandLoop( ) with this new instance of the FILE* so the next line of input comes from the specified file instead of where the previous command came from. When the end of a file is reached, the program is to revert back to the previous input source that contained the f command. This previous input source could be standard input or a file. By making these calls recursively, reverting back to the previous input source is a complete no-brainer.

However, this can cause an infinite loop if you try to access a file that your program is already reading from. Consider this scenario. Assume the user enters a command from standard input to start reading from file A. However; file A tells you to read from file B, file B tells you to read from file C, and file C tells you to read from file A. Since you always start reading from the top of the file, when file C eventually tells the program to read from file A, the program will reprocess the command to read from file B, which will reprocess the command to read from file C, which will reprocess the command to read from file A, which will reprocess the command to read from file B, which will reprocess the command to read from file C, which will reprocess the command to read from file A, which will reprocess…

In order to stop this, you are required to maintain a linked list of file names. Before the f command attempts to create a new instance of FILE* that read from file X, the f command is to check if the linked list of file names already contains the name of X.

- If the name X already exists in the linked list, the f command will NOT create a new instance of FILE* and it will NOT make the recursive call to processCommandLoop( ).
- If the name X does not exist in the linked list, the f command will add the name X to the linked list before making the recursive call to processCommandLoop( ) and it must remove the name X from the linked list after the call to processCommandLoop( ) returns.

You are responsible to write the code for this linked list yourself. Note that this will most likely be a linked list of Strings, while each island's adjacency list will most likely be a linked list of integers.

Print Statements for autograder

You can use cout instead of printf.

For inserting:

```
printf ("Performing the Insert Command for %d\n" );
```

For deleting:

```
printf ("Performing the Delete Command for %d\n" );
```

For doList( ):

```
printf ("Displaying the adjacency list:\n" );
```

Example of adjacency list in doList( ):

```
1 -->  2
2 -->  3
3 -->  1  4
4 -->  3  5
5 -->  5
```

This part may get updated as we tune the autograder on Gradescope.
Majority of print statement are already given in the base code and this pdf file. Any extra print statement will be added here in red.
Updates for print statement:

1- File not found:
```
printf("Cannot open the file\n");
```

2- File in use
```
printf("File is already in use\n");
```

3- Invalid island number in i, d, t commands
```
printf("Invalid value for island\n");
```

4- Existing island for i command
```
printf("Ferry already added\n");
```

5- Not existing island for d command
```
printf("Ferry ride does not exist\n");
```

Classes You Must Write

First, you are to write a class called Island. This class is to contain everything that is known about a specific Island. This MUST include the island's adjacency list and the "visited" status for the Depth First Search.

Next, you must write a list class. You are encouraged to name the class MyList as the name "List" is already used in the C++ Standard Template Libraries. Please note that your list class will need another class for the nodes in the list. Thus, you must write a class called MyNode. For your list class, in addition to the normal insertValue( ) and deleteValue( ) operations, you may want to write operations such as getNumberOfCurrentsValues( ) and getNthValue( ).
In order to pass Gradescope tests please insert new nodes at the end of the linked list.

The ArchipelagoExpedition class has been started for you. This class must contain the array of Islands for the specific map and the linked list of filenames for the "in-use" files needed for the f command.

All data members (or "instance variables") must be made private. Failure to do this will result is a severe reductions of points for the project.

You are not allowed to use any of the classes from the C++ Standard Template Libraries in this program. These classes include ArrayList, Vector, LinkedList, List, Set, Stack, HashMap, etc. **If you need such a class, you are to write it yourself.** These are sometimes called the C++ Standard Container Library. A full listing of the C++ Standard Template Libraries can be found at:
http://www.cplusplus.com/reference/stl/

Multiple Source Code Files

Your program is to be written using at least two source code files. One of the source file files is to contain the main function of the program named in a file using your Program name, like:

Proj5main.cpp

The other source code file must contain your Island class in a file named:

Island.cpp

You may use additional source code files if you wish (i.e. MyList.cpp), but these two are required. The above implies that you will need to write any appropriate .h file(s) and a makefile.

Coding Style

Don't forget to use good coding style when writing your program. Good coding style makes your program easier to be read by other people as the compiler ignores these parts/differences in your code. Elements of good code style include (but may not be limited to):

- Meaningful variable names
- Use of functions/methods
- Proper indentation
- Use of blank lines between code sections
- In-line comments
- Function/method header comments
- File header comments

The Code Review Checklist also hints at other elements of good coding style.

Program Submission

You are to submit the files for this project as a single zip file on **Gradescope.**
In your makefile, please rename the final executable program to "Project5".
Create a directory called "Project5", then put all of your source code files, your header file and your makefile into this directory. Zip this directory and submit the zip file.

Note that the folder name and file name are case sensitive.