ASSIGNMENT 1

Problem 1

Stack & Queue

a) You have to implement a **Dynamic Stack** using classes that will be used in Problem 3.
b) You have to implement a **Dynamic Queue** using classes that will be used in Problem 2.

Stack: A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

Queue: A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed **enqueue** and **dequeue**. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item.

Difference between a stack and a queue?

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. For better understanding visit the following page: https://www.geeksforgeeks.org/difference-between-stack-and-queue-data-structures/

➔ **Function you have to implement for both stack and queue**

- **void push(int data); //Insert element in stack/queue**
- **int pop(); //Return and remove first/last element from stack/queue**
- **int peek(); //Return the first/last element from stack/queue**
- **bool isEmpty(); //Check if stack/queue is empty or not**

Problem 2

No Way Home?

Problem: Jack is running out of fuel and he need to refuel as soon as possible to make it to his destination; He has got a road map that might have or might not have one route that leads to the nearest petrol station. The major problem is that he can only reach the petrol station if he follows the right route from to the petrol station otherwise the petrol will run out. As a programmer you need to write a code that can tell jack whether there is a route to the nearest petrol station or not.

What is given: You are given a helper code that will load up the array with the data that is given in the file. So, you do not need to write the code for file reading. “arr” is a 2D dynamic array that contains data written in the file. So build your solution by using the file reading code given.

Explanation:

- File Structure: First line contains total number of rows in our data, second line contain number of columns in our data. After that we have got the data (road map) itself which contains binary number separated by comma followed by a space.

```
10
10
1, 1, 1, 1, 1, 1, 1, 1, 1, 0
1, 0, 0, 1, 0, 0, 0, 1, 0, 0
1, 0, 0, 1, 0, 0, 0, 1, 0, 1
1, 1, 1, 0, 0, 1, 1, 0, 0, 1
1, 0, 1, 1, 1, 0, 0, 0, 0, 1
1, 0, 0, 0, 1, 1, 1, 0, 0, 0
1, 0, 1, 0, 0, 0, 1, 0, 0, 1
1, 0, 1, 1, 1, 0, 1, 1, 1, 1
1, 1, 0, 0, 0, 0, 0, 0, 0, 0
0, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

- In our 2d array 1 represents a road and 0 represents a blockage. Initially jack is standing at the top left corner of our road map (array[0][0] in your case) and the petrol station is located on the bottom right of our road map (array[N-1][M-1] in your case). You need to find the path from array[0][0] to array[N-1][M-1].
- You are given a code that reads data in an array from the given file. Use this array to solve this connection. You have to make a **2d dynamic array** and populate it with the data given in the file. Moreover, you have to use a **Queue** in order to find the path from the source to the destination.
- **Output:** Print “Path exists” if there is a path from source to destination else print “Path does not exist”.

  Path in the above example (row, column) pairs: (0,0) -> (1,0) -> (2,0) -> (3,0) -> (4,0) -> (5,0) -> (6,0) -> (7,0) -> (8,0) -> (8,1) -> (9,1) -> (9,2) -> (9,3) -> (9,4) -> (9,5) -> (9,6) -> (9,7) -> (9,8) -> (9,9)

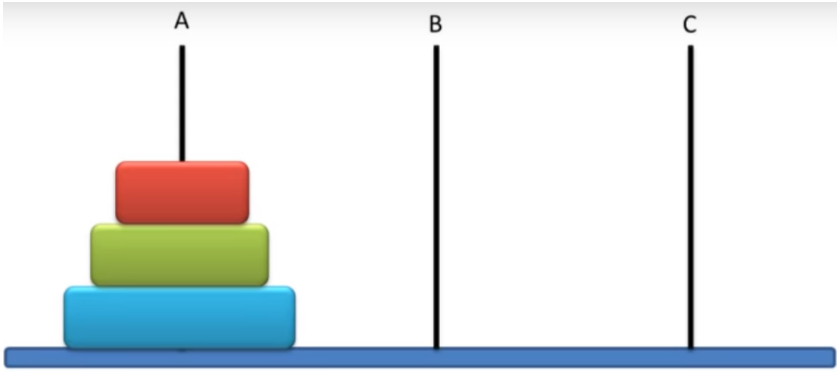
  Real Life Example: Imagine that you are driving and you have many friends with you and you don’t know the path to the destination. What would you do? You will send your friends in different directions one step at a time. After each step they will be divide and follow more and more paths until one person reaches the destination. If all the paths have been traversed and no one has reached the destination that means that destination is unreachable. Think of this in terms of queue and you will find a solution.

NOTE: Assume that there can be no more than 1 path from source to destination.

Problem 3

Stack Swapping

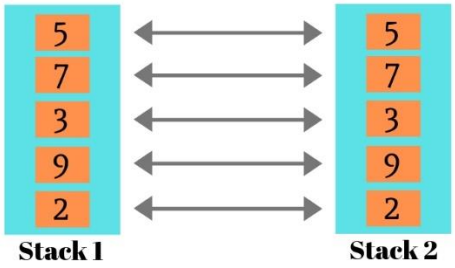
Problem: We have several distinct books stacked over each other. We need to place them at a different location but we are looking to maintain the order of the stacked books. Books are quite heavy and we can pick only one book at a time. Moreover, you can also get help from your friend in doing this but the problem is that he can also hold only one book at a time.



For example, in the above diagram we have 3 books stacker over each other at location A. We need to stack them in the same order at location B. We can lift only 1 book that is at the top of the pile at a time, we have also got a location C that can hold at most 1 book at a single time. Think of A & B as stacks and C as a integer variable.

Explanation:

- You need to get input from user. List of id's of books in the order in which they are stack on each other. For example, 1 2 3 4 5 contains 5 distinct books having id's from 1 to 5, 1 being at the top of the pile and 5 being at the bottom of the pile.
- After taking the input from the user you have to implement a stack and populate it with data taken from the user (1 at the top of stack and 5 at the bottom of the stack). This is going to be your source stack.
- You need to make another stack that is going to be your destination stack. This will be initially empty.
- You can only use two integer variables other than the two-stack implemented above.
  Hint: Variables can be used to temporarily store an element of stack.
- Your goal is to map source stack to the destination stack as shown in the diagram below.



- **Output:** Print both the source stack and the destination stack.

NOTE: You can only use two integer variables along with the source stack and the destination stack to solve this question. Loop variables are not included in these two integer variables.

Submission Details:

- **You have to Submit .cpp files for both questions.**
- **Name cpp file of question 1 as "YourRollNo_Q1.cpp", question 2 as "YourRollNo_Q1.cpp"**
- **Put both .cpp files in a single folder named after your RollNo and then zip the folder and then Submit!**
- **Do not copy code from anyone. Plagiarism with checked.**