ASSIGNMENT 3

Subject : Stack, Queue
TAs: Merve Ozdes
Programming Lanuage: C++
Due Date: 17.12.2020 23:59

1 Introduction

In this assignment, you will implement a cargo delivery simulation. According to this simulation, you are given several cities. In each city, there exists a cargo station which consists of cargo packages and cargo trucks. Your task consists of several missions. Each mission requires you to assemble a simulated cargo truck and cargo packages at the specified city and by following the given route, arrive at the destination city, leaving and taking cargo packages from the cities you visit. You are expected to create a program that reads the input about the cargo packages from the input files and after completing the missions, produces an output file exhibiting the final state of the stations at each city.

2 Background

2.1 Stack

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc. A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation. The following figure (Figure 1) depicts a stack and its operations A stack can be implemented by means of Array,

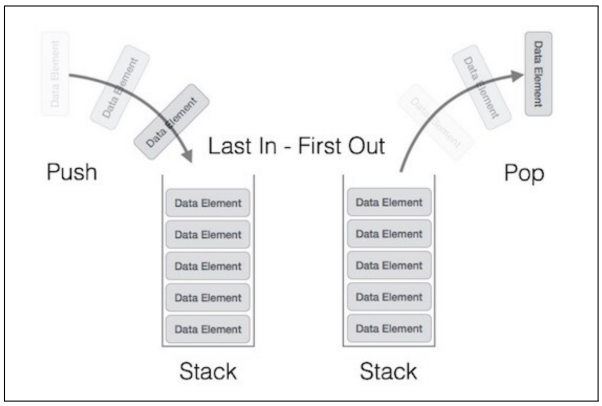

Figure 1: Stack and it's operation

Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using linked list.

2.2 Queue

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first. The following figure (Figure 2) depicts a queue and its operations.

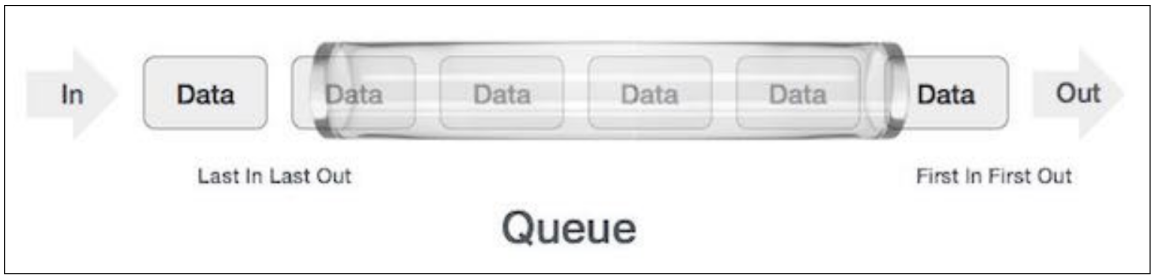

Figure 2: Queue representation

As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures. For this assignment, we are going to implement queues using linked lists.

Basic operations are as following:

- **insert / enqueue:**This function defines the operation for adding an element into queue.
- **remove / dequeue:** This function defines the operation for removing an element from queue.

For this assignment, stack ADT and queue ADT using linked list are provided to you.You are expected to develop generic classes as your stack and queue ADTs. In the Stack.h and Queue.h file given to you, there are incomplete classes that will be the ADTs you are expected to use in the project. The most of the structure for creating generic ADTs is given to you.You can also use additional class, function and class members.

3 Problem Definition

3.1 Scenario

You are expected to complete missions according to input files and produce an output file. The information about cities is in the text file which is dests.txt. In this file, you are given several cities as possible destination points for your simulations. In each city you are given, there exists a cargo station. In a station, there are cargo packages to be delivered to destinations and cargo trucks to transport these cargo packages. In our scenario, cargo packages are picked up and dropped off according to the stack principle which is last-in-first-out. The initial status of cargo packages in each city is given in packages.txt. A cargo truck garage keeps the cargo trucks in the station. In our scenario, cargo truck garages are modeled as queue principle which is first-in-first-out. When asked for a truck, they should retrieve the first truck inside the garage. The initial configuration of truck garages in each city is given in trucks.txt. The cargo truck and the packages it carries are modeled as a double-linked list. Keep in mind that cargo truck class extends packages class. Therefore, you will have doubly linked lists of packages as your carriage. The missions are given in mission.txt.

A misson has the following format: A-B-C-x-y-z_1,z_2,...

- A: Starting station
- B: Midway station

- C: Target station
- x: Number of cargo packages you take from the starting station
- y: Number of cargo packages you take from the midway station
- z: Indices of cargo packages you must leave at the midway station. Indices start from 0.

A complete mission requires you to complete the following procedure:

1. Assemble a cargo truck at the starting city, A. Since a cargo truck is modeled as a doubly linked list of cargo packages, you need to start with the first truck you have in the truck garage and insert it into the head of your list. Afterwards, you need to insert x packages at city A to this list.
2. Append y packages from the midway station, B, to your list.
3. Leave packages z_1, z_2... at midway station.
4. When disbanding the cargo packages, put the truck into the truck garage and packages at C.

During a mission you need to pay attention to the order of insertions and removals. An example run is given in below.

3.2 Experiment

Your system will take the input files as parameters which are dests.txt, packages.txt, trucks.txt and missions.txt. The initial configuration of the stations can be obtain by reading the input from these files. After reading the mission, the cargo truck is selected and the cargo packages specified in the mission are distributed. The status of each station should be written into the output file after the mission is completed.

3.3 Inputs

There will be four input files that your program will take as parameters. These are *dests.txt*, *packages.txt*, *trucks* and *missions.txt* respectively. Details of the files are given below.

dests.txt	**packages.txt**	**trucks.txt**	**missions.txt**
Istanbul	P1 Istanbul	T1 Ankara 4.8	Istanbul-Ankara-Diyarbakir-3-2-1,2
Ankara	P2 Istanbul	T2 Istanbul 7.2	
Diyarbakir	P3 Ankara	T3 Istanbul 2.8	
	P4 Istanbul	T4 Ankara 4.3	
	P5 Ankara		
	P6 Diyarbakir		
	P7 Diyarbakir		
	P8 Ankara		
	P9 Istanbul		
	P10 Ankara		

3.4 Outputs

You are expected to produce a output. In this output, status of each station should be printed. The format of this file is as follows:

Istanbul
Packages:
P1
Trucks:
T3
———-
Ankara
Packages:
P2
P4
P5
P3
Trucks:
T1
T4
———-
Diyarbakir
Packages:
P8
P10
P9
P7
P6
Trucks:
T2
———

3.5 Example Run

Assume that you are given the files in Section 3.3.
Reading the input from these files, the initial configuration of the stations as shown in Figure 3:

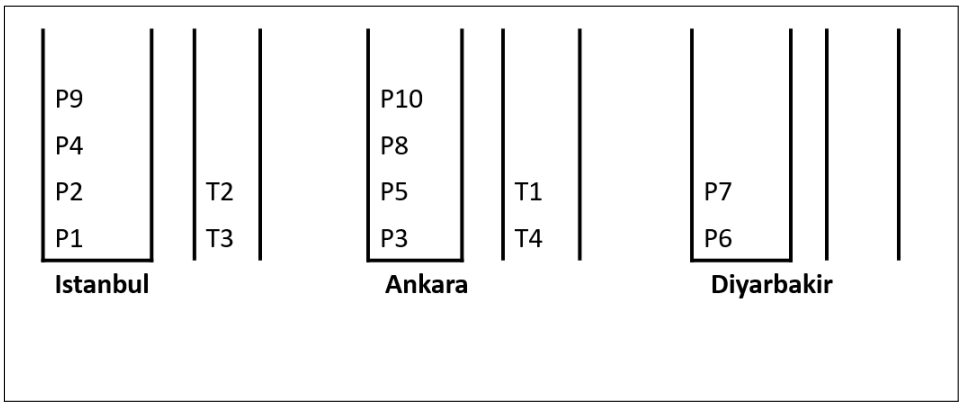

Figure 3: The initial configuration of the stations

After reading the mission, assemble the following cargo truck, leaving the station in Istanbul as shown in Figure 4. You need to start with the first truck that is the first item of queue you have in the cargo truck garage and insert it into the head of your list. Afterwards, you need to insert 3 packages of the cargo packages at city A to this list.

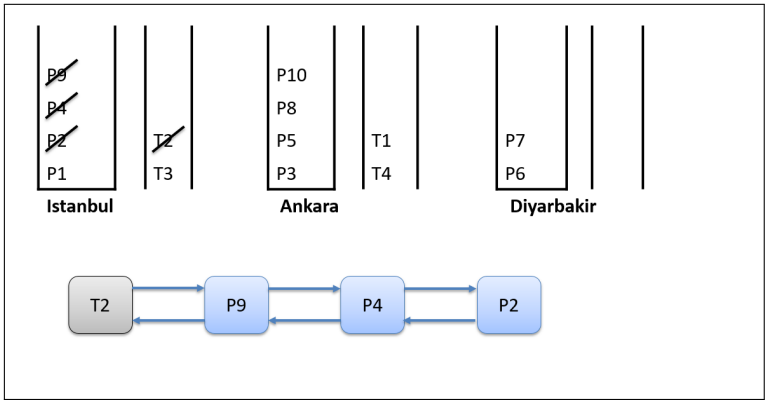

Figure 4: After reading the mission configuration of the stations

The cargo truck takes 2 packages from Ankara and leaves the first and second package after the truck at Ankara (Truck is not counted, packages indices start from 0). Pay attention to the order. This situation is shown in Figure 5.

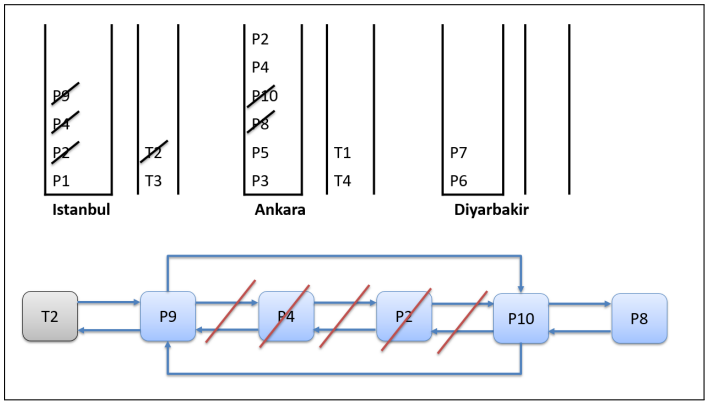

Figure 5: Status of each station after leaving Ankara

After the cargo truck reaches to Diyarbakir, it is disbanded at this city. Packages and the truck are inserted to the corresponding place. Again, pay attention to the order. Status of each station after reaching to Diyarbakir is shown in Figure 6.

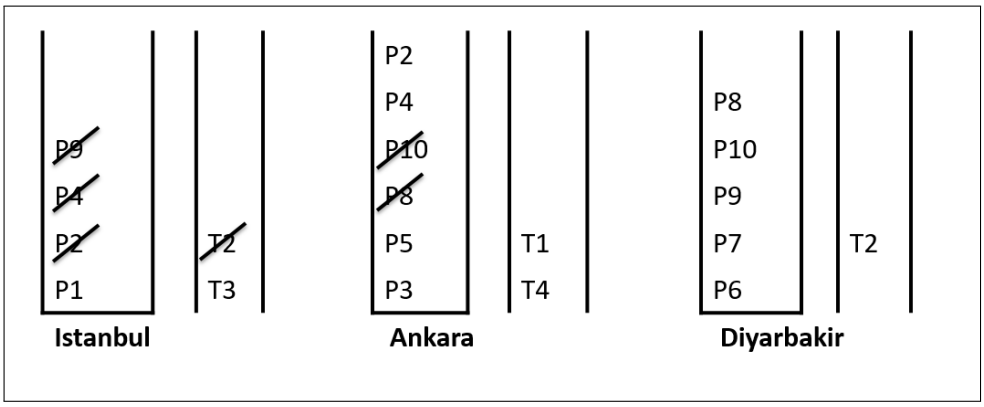

Figure 6: Status of each station after reaching to Diyarbakir

4 Grading and Evaluation

- Your work will be graded over a maximum of 100 points.
- Your total score will be partial according to the grading policy stated below.

Reading the file contents and building the initial status of stations	20p
Initial cargo truck at the starting city	15p
Picking up and dropping off of cargo packages according to the stack principle	25p
Correct reporting the status of each station after mission is completed	25p
Code design, clean and readable code, algorithmic perspective, comments	15p

- Your code will be tested with different inputs. And one output file will be expected as output file.
- You should submit only source files (*.cpp and *.h); not the compiled executable.
- Late submissions will be graded over 85 for the first 24 hours and over 70 for submissions between 24 -48 hours late.

Usage example:
> g++ -std=c++11 Main.cpp -o Main
>./Main dests.txt packages.txt trucks.txt missions.txt result.txt

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- Try to put as small code as possible inside the main method. Create other methods or classes, preferably, and call them in you main method. Main method should be quite comprehensible.
- Write READABLE SOURCE CODE block
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/fall/bbm201) and you are supposed to be aware of everything discussed in Piazza. You will be held responsible for any announcement and explanation made by the advisors.
- You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ No other submission method (email or etc.) will be accepted. Do not submit any file via e-mail related with this assignment.
- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system). You must submit your work with the file hierarchy stated below:

<student_id.zip>/(Required)
→src/(Required)
→Main.cpp(Required)
→Stack.h(required)
→Queue.h(required)
→*.cpp(optional)
→*.h(optional)

Policy

All work on assignments must be done **individually** unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an **abstract** way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) **will not be tolerated**. In short, turning in someone else's work (from internet), in whole or in part, as your own will be considered **as a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.