SEN-107 Fundamental Data Structures
Assessment Activity SEN-107:00010
Creating and Using Lists (Stacks and Queues)

Last update: 8 January 2025

Instructions

General Instructions

Write a program that solves the problem described below. Your program should read from the standard input and write to the standard output. We have provided example input and output files. When running your program on the example input file, it should produce the same output as in the example output file. While this example is a good starting point, you should test your program thoroughly, with multiple different inputs too. Your program should correctly handle all possible cases specified below. Correctness will be part of your assessment score. Please also follow the coding standards in ***C_CodingStandards.pdf***. Violating the coding standards will lower your assessment score.

Submit your ***C source code*** only. Your source code must be a single plain text file. Do not submit code as a doc file, pdf file, IDE project, or anything else. We must be able to compile and run your code on my computer, without any warnings or errors.

For convenience, we have provided a template for you, which already contains the skeleton code that handles the input-output. You may implement your solution using any data structure you learn from lectures (or beyond). However, you are **not** allowed to use other external libraries, such as C++ Standard Template Library (STL).

Assignment-Specific Instructions

For this assessment, you should submit a single file called ***vipTaxi.c***; as stated above, we have already provided a template for this file. The example input and output files are ***vipTaxi.in*** and ***vipTaxi.out***, respectively.

Problem Statement: VIP Taxis - A stack of queues

In Lab 1, you have already implemented a queue for taxis. However, in Thailand, resources are sometimes controlled by a party's status. For instance, a VIP group might get a taxi before an ordinary person, even when that ordinary person is first in the queue. Basically, this "queue" are multiple queues, for different levels of status. The higher the level of status, the higher the priority.

In fact, this behavior is like a stack. When a higher status party arrives, looking for a taxi, a new queue will be created on top of the stack. Only when all the parties waiting in that queue have gotten taxis will the stack be popped, allowing lower status parties to get taxis.

For this assessment, you must write a program called ***vipTaxi.c*** that implements this system.

- There are five levels of status. From lowest to highest these are (1) Ordinary Person, (2) MP, (3) Governor, (4) Cabinet Minister and (5) Prime Minister. If a party with a higher status arrives, parties of lower statuses have to wait until that party gets a taxi.
- At each timestep, two events can occur: A taxi arrives, or a party arrives.

- When a taxi arrives, it will be given to a party waiting with highest status; if there are multiple parties with the same highest status waiting, then it will be given to such a party that arrived first. (If a taxi arrives when no one is waiting, then the taxi will leave without picking up anyone.)
- If a party arrives that has strictly lower status than at least one of the parties waiting, that party will be told to go away because there are already VIPs waiting. (Otherwise, that party can join the waiting queue.)

Input Description

The first line of the input contains a number T denoting the number of timesteps.
Each of the next T lines denotes one of the two events.

- If the line is "Taxi", it indicates that a taxi has arrived.
- If the line starts with "Party", it indicates that a party has arrived. It will be followed by the party's status (which is a number from 1 to 5), the number of people in the party (which is also a number from 1 to 5), and the party's name (which is a string of length at most 20).

Output Description

For each timestep, the program should print the following in a new line:

- If a taxi arrives and picks up a party, the program should print "Pick up party x with y people" where x, y denote the name and the number of people in the party.
- If a taxi arrives and does not pick up anyone, then output "Empty Queue".
- If a party arrives and is accepted to the queue, then output "Accepted".
- If a party arrives and is told to go away, then output "Rejected".

Example Input & Output

`Input`	`Output`

`7`
`Taxi`
`Party 3 2 Elon`
`Party 3 1 Bill`
`Party 5 3 Donald`
`Party 4 2 JD`
`Taxi`
`Taxi`
	`Empty Queue`
	`Accepted`
	`Accepted`
	`Accepted`
	`Rejected`
	`Pick up party Donald with 3 people`
	`Pick up party Elon with 2 people`

Explanation: The first taxi arrives but the queue is still empty so it does not pick anyone up. In the next three timesteps, each party is accepted into the queue. In the fifth timestep, party JD is rejected because Donald's party—which has higher status (5) compared to JD's (4) —is already in the queue. When a taxi arrives next, Donald's party—which has the highest priority—gets picked up. Finally, when the last taxi arrives, only Elon's and Bill's parties remain in the queue; since Elon arrives first, his party gets the taxi.

Hints

One solution to this problem is to implement a stack of queues.