SEN-107 Fundamental Data Structures
Lab 1 Practice Assignment
Taxi Queue

Instructions

General Instructions

Note: Labs will **not** be part of your final grade and submissions are **not** mandatory. If you decide to submit, we will provide you with feedback and also grade your submission in a similar manner as assessments. While the score will not be included as part of your final grade, we hope it will still be helpful for your learning experience.

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 ***airportTaxi.c***; as stated above, we have already provided a template for this file. The example input and output files are ***airportTaxi.in*** and ***airportTaxi.out***, respectively.

Problem Statement: Airport Taxi Service

Queues are often used in the real world to handle cases where there are multiple people who must wait for resources. For instance, in a bank, people will form a queue to get window service. At the airport, people who want a taxi will be added to the end of a queue. Each time a taxi arrives, the front party waiting in the queue will be assigned to the taxi.

Write a program that manages people waiting for taxis at the airport. 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 in the queue that arrived first. (If a taxi arrives when no one is waiting, then the taxi will leave without picking up anyone.)

You can create your own queue data structure, or you can modify one of the modules from the Demos for Week 1 and link your code to that structure. Note that if you use ***arrayQueue.c*** you will have to make some changes, because that demo can only handle a queue that holds 10 items, and the Suvannaphumi taxi queue can be much longer! (So you may want to create a linked list queue. Otherwise set the maximum capacity to at least 100, and be sure you check for overflow and underflow errors!)

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 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".

Example Input & Output

Input	**Output**
8 Taxi Party 2 Elon Party 1 Bill Party 3 Donald Taxi Party 2 JD Taxi Taxi	Empty Queue Pick up party Elon with 2 people Pick up party Bill with 1 people Pick up party Donald with 3 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 added into the queue. When the next three taxis arrive, these three parties are picked up in the same orders as they arrive. Note that at the end, the last party (JD) remains inside the queue.

Extra Challenges

Extra challenge 1: Before the program exits, print the following statistics:

– Total number of passenger groups served by taxis during this run
– Total number of passenger groups still waiting for taxis when the program exits

Extra challenge 2: The sample run above shows a case where a taxi arrives but there are no passengers in the queue. In the example, this taxi is then ignored; when a new passenger group arrives, they simply join the queue.

To make this more realistic, create another data structure that will keep track of excess taxis that have arrived, but that have not gotten any customers because no one was waiting. Then, when a new passenger group appears, check to see if there are any taxis waiting. If there are, assign this new group to the next taxi right away; do not add them to the passenger queue.

Basically, this will require you to create a second queue for taxis. You might want to add the arriving taxi's registration number to the input, so you can print which taxi is assigned to each customer group.