CSE 4304-Data Structures Lab. Winter 2022-23
Batch: CSE 21
Date: August 28, 2023
Target Group: All
Topic: Queues

<u>**Instructions**</u>:
- Regardless you finish the tasks in the lab, you have to submit the solutions in the Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be at 11.59 PM of the day in which the lab has taken place.
- Task naming format: fullID_T01L02_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- Obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library.
- Modified sections will be marked with BLUE color.

Group	Tasks
2A	1 9 10 11
2B	1 9 10 12
1A	1 9 10 14
1B	1 9 10 13
Assignment	5,6, and the tasks that were not covered in your lab

Task-01: Implementing the basic operations of **Circular Queue.**

Queue is a linear data structure that follows the First In First Out (FIFO) principle. The first item to be inserted is the first one to be removed. The Insertion and Deletion of an element from a queue are defined as EnQueue() and DeQueue(). Furthermore, to ensure the reusability of space that becomes available when elements are dequeued, we use the concept of **circular queues.**

The first line contains *N* representing the size of a **Circular Queue**. The lines contain the 'function IDs' and the required parameter (if applicable). Function ID 1, 2, 3, 4, 5, and 6 correspond to EnQueue, DeQueue, isEmpty, isFull (assume the max size of Queue=5), size, and front. The return type of inEmpty and isFull is Boolean. Stop taking input once given -1.

Input	Output
5	
3	isEmpty: True
2	DeQueue: Underflow
1 10	EnQueue: 10
1 20	EnQueue: 10 20
5	Size: 2
1 30	EnQueue: 10 20 30
6	Front: 10
2	DeQueue: 20 30
1 40	EnQueue: 20 30 40
1 50	EnQueue: 20 30 40 50
4	isFull: False
1 60	EnQueue: 20 30 40 50 60
4	isFull: True
5	Size: 5
1 60	EnQueue: Overflow
5	Size: 5
2	DeQueue: 30 40 50 60
6	Front: 30
-1	Exit

Note:
You have to **implement** the circular queue operation functions **by yourself** for this task. Do **not** use the STL queue here.

Task 05: Time Needed to Buy Tickets

There are n people in a line queuing to buy tickets, where the 0-th person is at the front of the line and the (n-1)-th person is at the back of the line. You are given a 0-indexed integer array tickets of length n where the number of tickets that the i-th person would like to buy is tickets[i].

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

First line of input contains the info of the n-person in the line (ends with -1). Next line contains the value of k. Return the **time taken for the person at position k** (0-indexed) to finish buying tickets.

Input	Output
2 3 2 -1 2	6 Explanation: - First pass, everyone in the line buys a ticket and the line becomes [1, 2, 1]. - Second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0]. The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.
2 4 3 -1 2	8
5 1 1 1 -1 0	8 Explanation: - In the first pass, everyone in the line buys a ticket and the line becomes [4, 0, 0, 0]. - In the next 4 passes, only the person in position 0 is buying tickets. The person at position 0 has successfully bought 5 tickets and it took 4 + 1 + 1 + 1 + 1 = 8 seconds.
1 5 7 3 1 3 2 -1 3	15 States: 1 5 7 3 1 3 2 0 4 6 2 0 2 1 = 7 units 0 3 5 1 0 1 0 = 5 units 0 2 4 0 0 0 0 = 3 units

Note: Use a queue to solve this problem

Task 06: Number of Students Unable to Eat Lunch

The school cafeteria offers circular and square shaped sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwich.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:
- If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.
- Otherwise, s/he will leave it and go to the end of the queue.

This continues until none of the queue students want to take the top sandwich and are thus **unable to eat.**

First line of input will contain the number of students. Then you are given two integer arrays students and sandwiches where sandwiches[i] is the type of the i-th sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the j-th student in the initial queue (j=0 is the front of the queue). Return the number of students that are unable to eat.

Test case:

Input	Output
4 1 1 0 0 0 1 0 1	0 Explanation: - Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1]. - Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0]. - Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1]. - Front student leaves the top sandwich and returns to the end of the line making students = [0,1]. - Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1]. - Front student takes the top sandwich and leaves the line making students = [] and sandwiches = []. Hence all students are able to eat.
6 1 1 1 0 0 1 1 0 0 0 1 1	3
8 1 1 0 0 0 1 1 0	0

0 1 1 0 1 0 0 1	
8 1 0 0 0 1 0 0 1 0 1 1 0 1 0 0 1	1
8 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1	4

Task 9: **Implementing Queue using two Stacks**

Queue is an abstract data type that means the order in which elements were added to it, allowing the oldest element to be removed from the front and new elements to be added to the rear. This is called a First-In-First-Out (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting for the longest) is always the first one to be removed.

A basic queue has the following operation:
- **Enqueue**: Add a new element to the end of the queue.
- **Dequeue**: remove the element from the front of the queue.

In this task, you have to *implement a linear Queue using two Stacks.* Then process *q* queries, where each query is one of the following two types:
- 1 x: Enqueue element *x* into the end of the Queue and print the Queue size along with printing all the elements.
- 2: Dequeue the element from the front and print the Queue size with all the elements.

Input Format
The first line contains two integers, **N** and *q*. N denotes the max size of the Queue and 'q' denotes the number of queries.
Each line '*i*' of the *q* subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query *type*, but only query **1** is followed by an additional space-separated value, *x,* denoting the value to be enqueued.

Output Format
For each query, perform the Enqueue/Dequeue & print the Queue elements on a new line.

Sample Input	Sample Output
5 10 1 42 2 1 14 1 25 1 33 2 1 10 1 22 1 99 1 75	 Size:1 Elements: 42 Size:0 Elements: Null Size:1 Elements: 14 Size:2 Elements: 14 25 Size:3 Elements: 14 25 33 Size:2 Elements: 25 33 Size:3 Elements: 25 33 10 Size:4 Elements: 25 33 10 22 Size:5 Elements: 25 33 10 22 99 Size:5 Elements: Overflow!

Hint: **Define manual functions for push() and pop() operations.** Then define two stacks and use them in such a way that you achieve the FIFO property from the stored data. **You can use STL stack for this task, however, the push() and pop() functions must be separately implemented.**

Task 10:

Implement the basic operations of a *'Deque'* data structure. Your program should offer the user the following options:
1. void **push_front**(int key): Insert an element at the beginning of the list.
2. void **push_back**(int key): Insert an element at the end of the list.
3. int **pop_front**():   Extracts the first element from the list.
4. int **pop_back**():   Extracts the last element from the list.
5. int **size**():   Returns the total number of items in the Deque.

Note:
- The maximum time complexity for any operation is **O(1)**.
- For option 3,4: the program shows an error message if the list is empty.

Input format:
- The program will offer the user the following operations (as long as the user doesn't use option 6):
    - Press 1 to push_front
    - Press 2 to push_back
    - Press 3 to pop_front
    - Press 4 to pop_back
    - Press 5 for size
    - Press 6 to exit.
- After the user chooses an operation, the program takes necessary actions (or asks for further values if required).

Output format:
- After each operation, the status of the list is printed.

Input	Output
1 10	10
1 20	20 10
2 30	20 10 30
5	3
2 40	20 10 30 40
3	10 30 40
1 50	50 10 30 40
4	50 10 30
5	3

Note: Do not use any built-in functions. **Solve this task using Circular Queue concept.**

Task 11 — Bob's String Prediction

Problem Statement

Alice and Bob are playing a guessing game. Alice has a string S consisting of lowercase English letters. Bob attempts to guess Alice's string and predicts another string T. Bob's guess will be correct if and only if his string T equals Alice's string S, after S undergoes a finite number of clockwise rotations.

Formally, we define a clockwise rotation of a string X as follows —
Let, $X = X_1 X_2 \ldots X_{|X|}$. Then, after one clockwise rotation, X changes to $X_{|X|} X_1 X_2 \ldots X_{|X|-1}$. Here, $|X|$ denotes the length of the string X.

Bob will win if his predicted string T is correct. Your task is to determine whether Bob wins or not.

Input

Each input consists of two strings S and T. The first line is Alice's string S and the second line is Bob's string T.

Output

Print "Yes" if Bob wins the game; otherwise, print "No".

Sample Test Case(s)

Input
```
kyoto
tokyo
```

Output
```
Yes
```

Input
```
abc
arc
```

Output
```
No
```

Input
```
aaaaaaaaaaaaaaab
aaaaaaaaaaaaaaab
```

Output
```
Yes
```

Explanation

In the first sample test case, the rotations look like: **kyoto → okyot → tokyo**.

Task 12 — Gamer Rage
Problem Statement
Ninja is an avid Fortnite player who is going through a rough patch lately. After losing 10 games in a row, he was consumed by rage and impulsively hit the right-side of his keyboard. His keyboard was not catastrophically damaged, but he did manage to damage the "**Home**" key and the "**End**" key. The problem was that sometimes the "**Home**" key or the "**End**" key gets automatically pressed (internally).

Ninja was not aware of this issue, and he decided to write something in the in-game chat. He was focusing on typing the text while looking at the keyboard and forgot to look at the monitor. After he finished typing, he looked at the monitor and saw a text on the screen that looked different from the text he was focused on typing. Let's call this a **broken** version of the text.

You are given the string that Ninja types using his keyboard, along with the internal "**Home**" and "**End**" button presses. Your task is to print the **broken** string that Ninja would see if he looked at the monitor.

Input
There are several test cases. Each test case is a single line containing letters, underscores and two special characters '[' and ']'. '[' means the "**Home**" key is pressed internally, and ']' means the "**End**" key is pressed internally. **The input is terminated by end-of-file (EOF).**

Output
For each test case, print the **broken** text on the screen.

Sample Test Case(s)
Input
```
This_is_a_[Broken]_text
[[]][][]Happy_Birthday_to_you
gg[wp]
i[h[g]j]k]l]m]n]]o[f]p]q[e]r[d]s]t]u[c[b[a
```

Output
```
BrokenThis_is_a__text
Happy_Birthday_to_you
wpgg
Abcdefghijklmnopqrstu
```

Task 13 — Burn 'em
Problem Statement
Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. In card game terminology, there is a move called **burning**. A card is said to **burned** when the following operation is performed as long as there are at least two cards in the deck:

> *Throw away (burn) the top card and move*
> *the card that is now on the top of the*
> *deck to the bottom of the deck.*

Your task is to find the sequence of discarded cards and the last, remaining card.

Input
Each line of input (except the last) contains a number n. The last line contains '0' and this line should not be processed.

Output
For each number from the input, produce two lines of output. The first line presents the sequence of discarded cards, the second line reports the last remaining card. No line will have leading or trailing spaces. See the sample for the expected format.

Sample Test Case(s)
Input
```
7
19
10
6
0
```

Output
```
Discarded cards: 1, 3, 5, 7, 4, 2
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8, 12, 16, 2, 10, 18, 14
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 2, 6, 10, 8
Remaining card: 4
Discarded cards: 1, 3, 5, 2, 6
Remaining card: 4
```

Task 14 — Unethical Queue
Problem Statement
It is considered a bad manner to cut in front of a line of people because it is unfair for those at the back. However, we see such unethical behavior very regularly in our daily lives. Let's call such a queue an *Unethical Queue*.

In an unethical queue, each element belongs to a friend circle. If an element (person) enters the queue, he first searches the queue from head to tail to check if some of his friends (elements of the same friend circle) are already in the queue. If yes, he enters the queue right behind them. If not, he enters the queue at the tail and becomes the new last element. Dequeuing is done like in normal queues— elements are processed from head to tail in the order they appear in the unethical queue.

Your task is to write a program that simulates such an unethical queue.

Input
The input will contain one or more test cases. Each test case begins with the number of friend circles 't'. Then t friend circle descriptions follow, each one consisting of the number of elements belonging to the group and the elements themselves. A friend group may contain many people/elements.

Finally, a list of commands follows. There are three different kinds of commands:
1. **ENQUEUE** x - enter a person x into the unethical queue
2. **DEQUEUE** - Process the first element and remove it from the queue
3. **STOP** - end of test case

The input will be terminated by a value of 0 for t.
Note: The implementation of **the unethical queue should be efficient** — both enqueuing and dequeuing of an element should only take **constant time**.

Output
For each test case, first print a line saying "Scenario #k", where k is the number of the test case. Then, for each DEQUEUE command, print the dequeued element on a single line. Print a blank line after each test case, even after the last one.

Sample Test Case(s)
Input

Input	Output
2 3 101 102 103 3 201 202 203 ENQUEUE 101 ENQUEUE 201 ENQUEUE 102 ENQUEUE 202 ENQUEUE 103 ENQUEUE 203 DEQUEUE DEQUEUE DEQUEUE DEQUEUE DEQUEUE DEQUEUE STOP	Scenario #1 101 102 103 201 202 203
2 5 259001 259002 259003 259004 259005 6 260001 260002 260003 260004 260005 260006 ENQUEUE 259001 ENQUEUE 260001 ENQUEUE 259002 ENQUEUE 259003 ENQUEUE 259004 ENQUEUE 259005 DEQUEUE DEQUEUE ENQUEUE 260002 ENQUEUE 260003 DEQUEUE DEQUEUE DEQUEUE DEQUEUE STOP 0	Scenario #2 259001 259002 259003 259004 259005 260001