CSE 4304-Data Structures Lab. Winter 23-24
Batch: CSE 22
Date: Sept 18, 2024
Target Group: All
Topic: Queues

Instructions:

- Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
- Task naming format: fullID_T01L01_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the Google Classroom.
- If you find any tricky test cases that I didn't include but that others might forget to handle, please comment! I'll be happy to add them.
- Use appropriate comments in your code. This will help you to recall the solution in the future easily.
- 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.
- You are allowed to use the STL stack unless it's specifically mentioned to use manual functions.

Group	**Tasks**
2A	1 9 10 11 [Bonus: Task 15]
1B	1 11 9 10 [Bonus: Task 15]
1A	16 17 18 19 [Bonus: Task 14]
2B	16 17 18 19 [Bonus: Task 14]
Assignments (all groups)	12, 13, 14, 15, and the ones you weren't included in your respective 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 2 1 10 1 20 5 1 30 6 2 1 40 1 50 4 1 60 4 5 1 60 5 2 6 -1	 isEmpty: True DeQueue: Underflow EnQueue: 10 EnQueue: 10 20 Size: 2 EnQueue: 10 20 30 Front: 10 DeQueue: 20 30 EnQueue: 20 30 40 EnQueue: 20 30 40 50 isFull: False EnQueue: 20 30 40 50 60 isFull: True Size: 5 EnQueue: Overflow Size: 5 DeQueue: 30 40 50 60 Front: 30 Exit

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

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 two stacks and use them in such a way that you achieve the FIFO property from the stored data.

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 the Circular Queue concept.**

Task 11 — Bob's String Prediction

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, $X = X_1 X_2 \dots X_{|X|}$. Then, after one clockwise rotation, X changes to $X_{|X|} X_1 X_2 \dots X_{|X|-1}$. 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
- If Bob can't win : No
- If Bob wins : Yes. After x clockwise rotations.
  Here, 'x' represents the number of clockwise rotations required to convert S to T.

Input	**Output**
kyoto tokyo	Yes. After 2 clockwise rotations
abc arc	No
aaaaaaaaaaaaaaab aaaaaaaaaaaaaaab	Yes. Rotation not needed.
input putin	Yes. After 3 clockwise rotations

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]
```

Output

```
BrokenThis_is_a__text
Happy_Birthday_to_you
wpgg
```

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:
It is considered bad manners 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

Task 15

Daiyan and Ishraq are exploring a new game involving a pile of stones. There are total N stones in the pile, each marked with an integer. They can make two distinct types of moves with the pile:

1. **remove** a stone from the **top** of the pile and **put it back** on the **bottom** of the pile.
2. **remove** a stone from the top of the pile and **throw** it away.

Daiyan in his turn will perform **move 1** **once** and then **move 2** **once**.
Ishraq in his turn will perform **move 1** **twice** and then **move 2** **once**.
They will **stop** making moves when there is only **1 stone left** in the pile. Both of them alternate with **Daiyan** going **first**. Find the **person** performing the **last move** and the **number** written on the **last stone** left in the pile.

Input Format
The first line of each test case contains an integer **N**, representing the **number of stones** in the pile.
The second line of each test case contains N integers, representing the **number written** on each stone. (The leftmost number denotes the stone on top of the pile)

Output Format
For each test case, print integers representing the person making the last move and the number written on the stone remaining.

Input	**Output**
3 -5 0 5	Ishraq -5
4 -1 -3 2 4	Daiyan 2
6 -100000 0 0 100000 -1000000 1000000	Daiyan 0
11 21 11 10 98 5 124 105 76 62 59 83	Ishraq 76
1 217	Ishraq 217
6 53 71 209 116 6 9	Daiyan 209

Task-16:
A **queue** is a **linear data structure** that operates on the **First In First Out (FIFO)** principle, where the first element inserted is the first one to be removed. In this structure, adding an element is called **EnQueue()**, while removing one is known as **DeQueue()**. To optimize the use of space freed up by dequeued elements, **circular queues** are utilized.

Input Format
The first line contains N, which indicates the **size** of the circular queue.
The following lines contains a series of instructions–

a) **E x** (Enqueue x) : push x into the queue (back).
b) **D** (Dequeue) : remove the element from the front of the queue.

Stop taking input after encountering "**-1**".

Output Format
Each instruction will be followed by an output displaying the queue status in the following format.
Current Size : M
Full : No/Yes
Empty : No/Yes
Front Element : A
Front Index : a
Rear Element : Z
Rear Index : z
Queue Elements : A B C D….

Input	**Output**
3	
E 3	Current Size : 1 Full? : No Empty? : No Front Element : 3 Front Index : 0 Rear Element : 3 Rear Index : 0 Queue Elements : 3
E 7	Current Size : 2 Full? : No Empty? : No Front Element : 3 Front Index : 0 Rear Element : 7 Rear Index : 1 Queue Elements : 3 7
E 5	Current Size : 3 Full? : Yes

	Empty? : No Front Element : 3 Front Index : 0 Rear Element : 5 Rear Index : 2 Queue Elements : 3 7 5
E 100	Overflow!! Current Size : 3 Full? : Yes Empty? : No Front Element : 3 Front Index : 0 Rear Element : 5 Rear Index : 2 Queue Elements : 3 7 5
D	Current Size : 2 Full? : No Empty? : No Front Element : 7 Front Index : 1 Rear Element : 5 Rear Index : 2 Queue Elements : 7 5
D	Current Size : 1 Full? : No Empty? : No Front Element : 5 Front Index : 2 Rear Element : 5 Rear Index : 2 Queue Elements : 5
E 2	Current Size : 2 Full? : No Empty? : No Front Element : 5 Front Index : 2 Rear Element : 2 Rear Index : 0 Queue Elements : 5 2
D	Current Size : 1 Full? : No Empty? : No Front Element : 2` Front Index : 0

	Rear Element : 2 Rear Index : 0 Queue Elements : 2
D	Current Size : 0 Full? : No Empty? : Yes Front Element : - Front Index : 0 Rear Element : - Rear Index : 0 Queue Elements : -
D	Underflow!! Current Size : 0 Full? : No Empty? : Yes Front Element : - Front Index : 0 Rear Element : - Rear Index : 0 Queue Elements : -
-1	

Note:
You have to **implement** the circular queue operation functions **by yourself** from scratch for this task. You are **not allowed** to use **STL Library Queue** or **Vector** for this task.

Task-17:
Given a **queue** data structure that supports standard operations such as enqueue() and dequeue(), the objective is to **implement a stack** data structure using two queues. The stack functions that you have to implement using **queues** are -

a) push(x) : pushes x in the top of stack
b) pop() : pops the topmost element in the stack
c) isFull() : returns true if the stack is full, otherwise returns false.
d) isEmpty() : returns true if the stack is empty, otherwise returns false.
e) top() : returns the topmost element in the stack.
f) size() : returns the size of the stack.

Input Format
The first line contains N, which indicates the **size** of the stack.
The following lines contains a series of instructions–

a) **push x** : pushes x into the stack.
b) **pop** : removes the topmost stack element..

Stop taking input after encountering "**-1**".

Output Format
Print the stack elements.

Input	Output
3	
push 5	5
push 7	5 7
push 1	5 7 1
push 100	Overflow!! 5 7 1
pop	5 7
pop	5
pop	
pop	Underflow!!

Note : Use isFull() and isEmpty() to check overflow and underflow conditions respectively.

Task-18:
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 first line denotes the size of the deque.
The program will offer the user the following operations (as long as the user doesn't use last option):

- **PF** to push_front
- **PB** to push_back
- **DF** to pop_front
- **DB** to pop_back
- **S** for size
- **F** for front index
- **R** for rear index
- **STAT** for Full,Empty or none
- **E** to exit.

- After the user chooses an operation, the program takes necessary actions (or asks for further values if required).

Output format:
Print the deque elements for first 4 operations, otherwise return the asked element/value

Input	Output
3	
PF 1	1
PB 3	1 3
PB 4	1 3 4
STAT	FULL
DF	3 4
PB 5	3 4 5
F	1

S	3
R	0
DB	3 4
DB	3
DB	
DB	Underflow
E	

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

Task-19:
IUT cafeteria offers **Kalabhuna** and **Mutton Curry** at lunch break on Fridays, referred to by numbers **0** and **1** respectively. All students stand in a queue. Each student either prefers **Kalabhuna** or **Mutton Curry**.
The total number of these two dishes in the cafeteria is equal to the number of students. The dishes are placed in a stack. At each step:

1) If the student at the front of the queue prefers the dish on the top of the stack, they will take it and leave the queue.
2) Otherwise, they will leave it and go to the queue's end.

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

Input Format

In the first line you'll be provided with the total number of students, N. Followed by two lines having N integers (0 or 1).
The second line will contain the choice of the students in the queue and the third line will contain the order in which the dishes are stacked.
(First number of the second line denote the preference of the student standing in front of the queue and the first number in the third line denotes the dish on top of the stack)

Output Format

Return the number of students that are unable to eat.

Input	Output
4 students = [1, 1, 0, 0] dishes = [0, 1, 0, 1]	0
3 Students = [1, 1, 0] Dishes = [1, 0, 1]	0
9 Students = [0, 1, 1, 1, 0, 1, 0, 1, 0] Dishes = [1, 1, 0, 0, 1, 1, 0, 0, 0]	1
4 Students = [1, 0, 0, 1] Dishes = [1, 0, 0, 0]	1
8 Students = [1, 0, 0, 1, 0, 0, 1, 1] Dishes = [1, 0, 0, 1, 1, 1, 1, 1]	2
6 Students = [1, 1, 1, 0, 0, 1] Dishes = [1, 0, 0, 0, 1, 1]	3