CSE 4304-Data Structures Lab. Winter 2024-25
Date: 26 November 2025
Target Group: All groups
Topic: Basic tasks, Arrays, recursion, logic-based problems.

<u>**Instructions**</u>:

- 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 recall the solution in the future easily.
- The 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**
1A/1B/2A/2B	1 2 3 4
Assignment	5 6 7 8

Task 1: Implementing the basic operations of a Circular Queue.

A 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 the 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, size, and front (assume the max size of the Queue is 5). 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. Don’t use Vector.

Task 2: Implement the basic operations of a Deque.

In this task, you need to 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 options 3 and 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 3: Implementing a Queue using two Stacks

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 maximum 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	Size:1 Elements: 42
2	Size:0 Elements: Null
1 14	Size:1 Elements: 14
1 25	Size:2 Elements: 14 25
1 33	Size:3 Elements: 14 25 33
2	Size:2 Elements: 25 33
1 10	Size:3 Elements: 25 33 10
1 22	Size:4 Elements: 25 33 10 22
1 99	Size:5 Elements: 25 33 10 22 99
1 75	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. You can use STL functions to implement Stacks.

Task 4: 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 **undergoing 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**	**Explanation**
kyoto tokyo	Yes. After 2 clockwise rotations	kyoto → okyot → tokyo (2 rotations)
input putin	Yes. After 3 clockwise rotations	
abcd bcda	Yes. After 3 clockwise rotations	abcd → dabc → cdab → bcda (3 rotations) This could be done in 1 rotation by abcd → abcd, but that would be anti-clockwise, which is not allowed in this scenario.
abc arc	No	
aaaaaaaaaaaaaaab aaaaaaaaaaaaaaab	Yes. Rotation not needed.	
abab baba	Yes. After 1 clockwise rotations	
aaaa aaaaa	No	
abca acab	No	Two strings containing the same letters in different order (an anagram), but they can't be equal even after N rotations.
a a	Yes. Rotation not needed.	

Input	**Output**	**Explanation**
a b	No	
abc ab	No	

Task 5: How many are unable to eat?

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.
The second line will start with the word student followed by N integers (0 or 1) representing the preferences of the students in the queue.
The third line will start with the word dishes followed by N integers (0 or 1) representing the order in which the dishes are stacked.
The first number after student represents the preference of the student at the front of the queue, and the first number after dishes represents the dish on top of the stack.

Output Format
Return the number of students that are unable to eat.

Input	**Output**
4 student 1 1 0 0 dishes 0 1 0 1	0
3 student 1 1 0 dishes 1 0 1	0
9 student 0 1 1 1 0 1 0 1 0 dishes 1 1 0 0 1 1 0 0 0	1
4 student 1 0 0 1 dishes 1 0 0 0	1
8 student 1 0 0 1 0 0 1 1 dishes 1 0 0 1 1 1 1 1	2
6 student 1 1 1 0 0 1 dishes 1 0 0 0 1 1	3
5 student 0 1 1 0 0 dishes 1 1 1 0 1	3
7 student 1 0 0 1 1 0 1 dishes 0 0 1 0 0 1 0	3

Note: Only the student at the front of the queue may be rotated to the back if their desired dish does not match the dish at the front of the dish list. The dishes cannot be rearranged, they can only be removed when the dish at the front matches the preference of the student at the front of the queue.
Red - Match (The student standing in front and the dish in front gets removed)
Blue - No Match (Student standing in front of the queue goes to the back)

Test Case 6 :

```
Dishes =     [1, 0, 0, 0, 1, 1]
Students =   [1, 1, 1, 0, 0, 1]          (Match - Dequeue from front)

Dishes =     [0, 0, 0, 1, 1]
Students =   [1, 1, 0, 0, 1]              (No Match - Dequeue from front, enqueue at
the back)

Dishes =     [0, 0, 0, 1, 1]
Students =   [1, 0, 0, 1 ,1]              (No Match - Dequeue from front, enqueue at
the back)

Dishes =     [0, 0, 0, 1, 1]
Students =   [0, 0, 1 ,1 ,1]              (Match - Dequeue from front)

Dishes =     [0, 0, 1, 1]
Students =   [0, 1 ,1 ,1]          (Match - Dequeue from front)

Dishes =     [0, 1, 1]
Students =   [1 ,1 ,1]             (No further matches)


# of students remaining = 3
```

Test Case 8:

```
Dishes =   [0, 0, 1, 0, 0, 1, 0]
Students = [1, 0, 0, 1, 1, 0, 1] (No Match - Dequeue from front, enqueue at the back)

Dishes =   [0, 0, 1, 0, 0, 1, 0]
Students = [0, 0, 1, 1, 0, 1, 1] (Match - Dequeue from front)

Dishes =   [0, 1, 0, 0, 1, 0]
Students = [0, 1, 1, 0, 1, 1] (Match - Dequeue from front)

Dishes =   [1, 0, 0, 1, 0]
Students = [1, 1, 0, 1, 1] (Match - Dequeue from front)

Dishes =   [0, 0, 1, 0]
Students = [1, 0, 1, 1] (No Match - Dequeue from front, enqueue at the back)


Dishes =   [0, 0, 1, 0]
Students = [0, 1, 1, 1] (Match - Dequeue from front)

Dishes =   [0, 1, 0]
Students = [1, 1, 1] (No Match - Dequeue from front, enqueue at the back)
```

```
Dishes =   [0, 1, 0]
Students = [1, 1, 1] (No further matches)

# of students remaining = 3
```

Task 6 – 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	Output
This_is_a_[Broken]_text	BrokenThis_is_a__text
[[]][][]Happy_Birthday_to_you	Happy_Birthday_to_you
gg[wp]	wpgg

Note: Although the task can be solved in multiple ways, we want you to use stack/queue/deque to solve this.

Note: Pressing the Home key moves the cursor to the beginning, while pressing the End key moves the cursor to the end.

Task 7: Unethical Queue

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

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, print the final queue status.

Note: Although the task can be solved in multiple ways, we want you to use stack/queue/deque to solve this.

Sample Test Case(s)
Input

Input	**Output**
3 4 204 238 201 207 2 214 213 4 117 104 138 220 ENQUEUE 214 ENQUEUE 201 ENQUEUE 207 ENQUEUE 220 ENQUEUE 204 ENQUEUE 117 ENQUEUE 104 ENQUEUE 238 ENQUEUE 138 ENQUEUE 213 STOP	214, 213, 201, 207, 204, 238, 220, 117, 104, 138
3 3 101 102 103 2 201 202 4 301 302 303 304 ENQUEUE 101 ENQUEUE 201 DEQUEUE ENQUEUE 103 ENQUEUE 302 ENQUEUE 102 DEQUEUE ENQUEUE 202 ENQUEUE 301 STOP	103, 102, 302, 301, 202

Test Case 1 Explained

```
Initially:    Queue = []
(Empty)

ENQUEUE 214    Queue = [214]
(Gets into the front of empty queue)

ENQUEUE 201    Queue = [214, 201]
(None of his friends is on the queue, joins the back)

ENQUEUE 207    Queue = [214, 201, 207]
(His friend 201 is on the queue, stands behind him)

ENQUEUE 220    Queue = [214, 201, 207, 220]
(None of his friends is on the queue, joins the back)

ENQUEUE 204    Queue = [214, 201, 207, 204, 220]
(Friends 201 and 207 are on the queue, stands behind the last friend 207)

ENQUEUE 117    Queue = [214, 201, 207, 204, 220, 117]
(Friend 220 is on the queue, stands behind him)

ENQUEUE 104    Queue = [214, 201, 207, 204, 220, 117, 104]
(Friends 220 and 117 are on the queue, stands behind the last friend 117)

ENQUEUE 238    Queue = [214, 201, 207, 204, 238, 220, 117, 104]
(Friends 201, 207, and 204 are on the queue, stands behind the last friend 204)

ENQUEUE 138    Queue = [214, 201, 207, 204, 238, 220, 117, 104, 138]
(Friends 220, 117, and 104 are on the queue, stands behind the last friend 104)

ENQUEUE 213    Queue = [214, 213, 201, 207, 204, 238, 220, 117, 104, 138]
(Friend 214 is on the queue, stands behind him)
```

Test Case 2 Explained

```
Initially:    Queue = []
(Empty)

ENQUEUE 101    Queue = [101]
(Queue empty, joins the back)

ENQUEUE 201    Queue = [101, 201]
(None of his friends is on the queue, joins the back)

DEQUEUE    Queue = [201]
(First element 101 is processed and removed from the queue)

ENQUEUE 103    Queue = [201, 103]
(Friend 101 was dequeued, no friends in queue → joins the back)

ENQUEUE 302    Queue = [201, 103, 302]
(None of his friends is on the queue, joins the back)

ENQUEUE 102    Queue = [201, 103, 102, 302]
(Friend 103 is in queue, stands behind him)
```

```
DEQUEUE    Queue = [103, 102, 302]
(First element 201 is processed and removed from the queue)

ENQUEUE 202    Queue = [103, 102, 302, 202]
(Friend 201 is gone, no friends in queue → joins the back)

ENQUEUE 301    Queue = [103, 102, 302, 301, 202]
(His friend 302 is on the queue, stands behind him)
```

Note: Members belonging to a group are of the same color.

Task 8

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 take their turns alternatively.

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 unique integers, representing the **number written** on each stone. (The leftmost number denotes the stone on top of the pile)
The second line contains the player going first.

Output Format
For each test case, print one line for each move. On each line, print the player whose turn it is, followed by the contents of the pile after their move.

# 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	**Output**
5 40 10 50 30 20 Ishraq	Ishraq 30 20 40 10 Daiyan 40 10 30 Ishraq 40 10 Daiyan 40
6 18 3 21 7 12 9 Daiyan	Daiyan 21 7 12 9 18 Ishraq 9 18 21 7 Daiyan 21 7 9 Ishraq 21 7 Daiyan 21
7 55 10 35 25 45 5 15 Ishraq	Ishraq 25 45 5 15 55 10 Daiyan 5 15 55 10 25 Ishraq 10 25 5 15 Daiyan 5 15 10 Ishraq 5 15 Daiyan 5
4 9 2 6 12 Daiyan	Daiyan 6 12 9 Ishraq 6 12 Daiyan 6

6 53 71 209 116 6 9	Ishraq 116 6 9 53 71 Daiyan 9 53 71 116 Ishraq 116 9 53 Daiyan 53 116 Ishraq 116

Note: Although the task can be solved in multiple ways, we want you to use stack/queue/deque to solve this.

Initial Pile: [40, 10, 50, 30, 20]

- Top = 40
- Bottom = 20
- First player = Ishraq

Turn 1: Ishraq

- **Move1:** remove top (40) → put at bottom → [10, 50, 30, 20, 40]
- **Move1:** remove top (10) → put at bottom → [50, 30, 20, 40, 10]
- **Move2:** remove top (50) → discard → [30, 20, 40, 10]

Pile after Ishraq: [30, 20, 40, 10]

Turn 2: Daiyan

- **Move1:** remove top (30) → put at bottom → [20, 40, 10, 30]
- **Move2:** remove top (20) → discard → [40, 10, 30]

Pile after Daiyan: [40, 10, 30]

Turn 3: Ishraq

- **Move1:** remove top (40) → put at bottom → [10, 30, 40]
- **Move1:** remove top (10) → put at bottom → [30, 40, 10]
- **Move2:** remove top (30) → discard → [40, 10]

Pile after Ishraq: [40, 10]

Turn 4: Daiyan

- **Move1:** remove top (40) → put at bottom → [10, 40]
- **Move2:** remove top (10) → discard → [40]

Pile after Daiyan: [40]