STACK,

.Q1: In the MTN MoMo app, when you fill payment details step-by-step, pressing back removes the last step How does this show the LIFO nature of stacks

- MTN MoMo App

when you're making a payment in the MTN MoMo app.

You go through several steps like:

1. Choose service (e.g., Pay Bill)
2. Enter account number
3. Enter amount
4. Confirm details

Every step you take is like pushing (adding) that step onto a stack.

- How the MTN MoMo App Shows LIFO:

when you want to go back one step (by pressing the "Back" button):

- The app removes the last step you just did.
- For example, if you just entered the amount, pressing "Back" removes that and takes you to the account number screen again.

This means:

The **last step (amount)** is the **first to go** — exactly how a **stack** works (LIFO).

Q2: In UR Canvas, when you navigate course modules, pressing back undoes the last step Why is this action similar to popping from a stack

- **What is "Pop" in a Stack?**

Pop means:

- Remove the item from the **top** of the stack.
- Like removing the last thing you did.

- **UR Canvas and the Back Button**

When you're using **UR Canvas** (or any web navigation system), and you're moving through course modules or pages, the system keeps track of what you've clicked on, kind of like a **stack of pages**.

➢ Pressing "Back" = Pop

When you press the **Back** button:

- It **removes (pops)** the **last page** you visited (the one on top).
- Then it shows you the **previous page** (now on top of the stack).

➢ Why is this Similar to Popping from a Stack?

Because:

Stack Action	UR Canvas Navigation Example
You **push** an item	You open a new module or page
You **pop** an item	You press **Back** to remove the last page
LIFO behavior	Last page you opened is the first one you go back from

So, pressing **Back** in UR Canvas is just like **popping** the most recent action off a stack — you undo the last step and go to the one before it.

➢ Example:

1. You open **Module A** → Stack = [Module A]
2. You open **Module B** → Stack = [Module A, Module B]
3. You open **Module C** → Stack = [Module A, Module B, Module C]

Now you press **Back**:

- You **pop** Module C → Stack = [Module A, Module B]
- You're now back on **Module B**

Q3: In BK Mobile Banking, transactions are added to history How could a stack enable the undo function when correcting mistakes?

Solution,

How a Stack Helps with Undo:

Every time you do a transaction or action in the app (like send money, update info, etc.), it gets **pushed** (added) onto a **stack**.

If you make a mistake, you can **pop** (remove) the last action from the stack — this is the **undo**.

Q4: In Irembo registration forms, data entry fields must be correctly matched. How can stacks ensure forms are correctly balanced

Solution,

- ## How Does a Stack Help Keep Forms Balanced?

when the system is checking if all the fields and sections you open are properly closed.

Here's how a **stack** helps:

1. When you **open** a section (like starting a name field or selecting a country), the system **pushes** that onto the stack.
2. When you **close** that section or complete it, the system **pops** it off.
3. At the end, if the stack is **empty**, it means everything was opened and closed properly — the form is **balanced**.
4. If the stack still has items, it means **something was left open or unmatched** — the form is **not balanced**.

- ## Example:

Let's say you are filling this Irembo form:

- Start filling **Personal Info** → Stack = [Personal Info]
- Start filling **Address Info** → Stack = [Personal Info, Address Info]
- Finish **Address Info** → Pop → Stack = [Personal Info]
- Finish **Personal Info** → Pop → Stack = []

So, Stack is empty → All fields were matched and closed → Form is balanced!

How Stacks Help in Irembo Forms

Q5: A student records tasks in a stack: Push("CBE notes"), Push("Math revision"), Push("Debate"), Pop(), Push("Group assignment") Which task is next (top of stack)

"Group assignment" is the next task (top of the stack).

Q6: During ICT exams, a student undoes 3 recent actions. Which answers remain in the stack after undoing

Solution,

The answers that remain are those recorded before the last 3 actions.

Q7: In RwandAir booking, a passenger goes back step-by-step in the form. How does a stack enable this retracing process?

Solution,

Retracing steps means:

The passenger moves **backward**, one step at a time, through the booking form — like undoing each previous action.

How a stack helps:

1. Each step the passenger completes (e.g., selecting flight, entering info) is **pushed** onto the stack.
2. When the passenger clicks **"Back"**, the system **pops** the last step off the stack.
3. This takes them to the **previous step** — the one now on top of the stack

Because a **stack is Last In, First Out (LIFO)**:

- The **last step** done is the **first one removed**.
- This perfectly matches the idea of going **backward step-by-step**.

A stack enables retracing in RwandAir booking by storing each completed step.
Pressing "Back" pops the most recent step off the stack, returning the passenger to the previous step.

Q8: To reverse “Umwana ni umutware”, push each word and then pop. Show how a stack algorithm reverses the proverb

Step 1: Split the sentence into words

```
["Umwana", "ni", "umutware"]
```

Step 2: Push each word onto the stack

Action	Stack (Top at right)
Push "Umwana"	["Umwana"]
Push "ni"	["Umwana", "ni"]
Push "umutware"	["Umwana", "ni", "umutware"]

Step 3: Pop each word from the stack to reverse the sentence

Action	Stack (Top at right)	Output
Pop	["Umwana", "ni"]	"umutware"
Pop	["Umwana"]	"umutware ni"
Pop	[]	"umutware ni Umwana"

Final Reversed Sentence:

```
umutware ni Umwana
```

Q9: A student searches shelves in Kigali Public Library (deep search). Why does a stack suit this case better than a queue?

Solution,

A **stack** (Last In, First Out) is better for deep search because it explores **one path deeply** before backtracking. When a student searches shelves in the Kigali Public Library deeply, they go down one topic or section fully. A **stack** supports this by:

- Allowing you to go deep and backtrack easily
- Using less memory in deep searches
- Reaching deep items faster

Q10: In BK Mobile app, moving through transaction history uses push and pop Suggest a feature using stacks for transaction navigation

Solution,

- **Back Stack (push):**
  Each time a user views a transaction or opens a detail page, it's **pushed** onto the **Back Stack**.
- **Forward Stack (pop/push):**
  If the user presses "Back", the current page is **popped** from the Back Stack and **pushed** to the Forward Stack.
  If they press "Forward", it's popped from the Forward Stack and returned to the Back Stack.

Example:

1. User views:
   → Transaction A → Transaction B → Transaction C
   (Each pushed to **Back Stack**)

2. User taps **Back**:
   Sees Transaction B (Transaction C moves to **Forward Stack**)
3. User taps **Forward**:
   Goes back to Transaction C

QUEUE

Q1: At a restaurant in Kigali, customers are served in order How does this show FIFO behavior?

❖ At a restaurant, customers are typically served **in the order they arrive** — the **first** customer to arrive is the **first** to be served.

This is FIFO behavior:

- **FIFO** stands for **First In, First Out**
- It means the **first item added** is the **first one removed**
- In the restaurant:
  - The **first customer** to enter the queue (or sit down) is the **first** to get their order
  - The **next customer** waits until the first one is served — just like in a queue

Arrival Order	**Serving Order**
1. Alice	1. Alice
2. Ben	2. Ben
3. Chantal	3. Chantal

This matches **FIFO: First In → First Served**

Q2: In a YouTube playlist, the next video plays automatically. Why is this like a dequeue operation

Solution,

A **queue** is a **First-In-First-Out (FIFO)** data structure. It works like a line — the first item added is the first one to be removed.

- When you play a YouTube playlist:
  - Videos are **lined up** in order.
  - The **first video** is played (dequeued).
  - Once it's done, the **next video** in the queue is automatically played.
  - The one that just played is **removed** (or considered "done").

- **How this resembles a dequeue (specifically `dequeue from front`):**

- The **playlist** is like a **queue**.
- When a video plays, it's like **removing** the video from the **front** of the queue.

- Then the next video becomes the new "front".

Q3: At RRA offices, people waiting to pay taxes form a line. How is this a real-life queue?

Solution,

Real-Life Queue at RRA:

- **People arrive** and stand at the **end of the line**.
- The **first person** who arrived is served **first**.
- As each person is served, they **leave the line** (dequeue).
- New arrivals go to the **back** of the line.

Why it's a Queue:

Feature	Real-Life Example (RRA Line)	Computer Queue Equivalent
Ordered line	People stand in sequence	Elements are stored in order
First served = first in	First person to arrive is served first	FIFO behavior
Add at the end	New person joins the back	`enqueue()`
Remove from front	Next person is called to the counter	`dequeue()`

Q4: In MTN/Airtel service centers, SIM replacement requests are processed in order How do queues improve customer service

Solution,

1. Fairness and Order

2. Efficiency

- Reduces chaos, especially during peak hours.

3. Predictability

- Queues give customers a **clear idea of their wait time**, which helps set expectations.

4. Better Resource Management

5. Improved Customer Experience

- A smooth and predictable process reduces stress and frustration.

Q5: In Equity Bank, operations are: Enqueue("Alice"), Enqueue("Eric"), Enqueue("Chantal"), Dequeue(), Enqueue("Jean") Who is at the front now?

Solution,

1. **Enqueue("Alice")** → Queue: `Alice`
2. **Enqueue("Eric")** → Queue: `Alice, Eric`
3. **Enqueue("Chantal")** → Queue: `Alice, Eric, Chantal`
4. **Dequeue()** → Removes **Alice** (the first in queue)
   → Queue becomes: `Eric, Chantal`
5. **Enqueue("Jean")** → Queue becomes: `Eric, Chantal, Jean`

Final Queue:

```
Eric (front), Chantal, Jean
```

Q6: RSSB pension applications are handled by arrival order Explain how a queue ensures fairness

Solution,

1. Serves People in Order of Arrival

- Everyone is served **based on when they applied**, so no one skips ahead.
- This is fair because early applicants get attended to earlier.

2. Equal Treatment

- All applications are processed using **the same rules**, without favoritism.
- Everyone waits their turn, regardless of status or background.

Q7: Explain how each maps to real Rwandan life.

Operation: Different queue types.

Examples: • Linear queue = people at a wedding buffet.

• Circular queue = buses looping at Nyabugogo.

• Deque = boarding a bus from front/rear

Solution,

1. Linear Queue

A queue where elements are added at the rear and removed from the front — **First-In-First-Out (FIFO)**.

Real-life Example in Rwanda:

- **People queuing at a wedding buffet**

  - At traditional Rwandan weddings (*ubukwe*), guests line up to serve food.

- The first person in line gets served first, and others follow in order.
- No one jumps the queue — it's a simple, straight line.

2. Circular Queue

Definition: A queue where the last position connects back to the first, forming a circle — used to manage continuous processes.

Real-life Example in Rwanda:

- **Buses looping at Nyabugogo Bus Park**

- Buses that operate in a loop (e.g., city buses or inter-district routes) leave Nyabugogo, go to their destinations, and return to the start.
- Instead of parking permanently, they **reuse the same slots**, like a circular queue reusing memory locations.
- As one bus leaves, another comes in — a continuous cycle.

3. Deque (Double-Ended Queue)

Definition: A queue where elements can be added or removed from both front and rear.

Real-life Example in Rwanda:

- **Boarding a minibus (*coaster* or *twegerane*) from front or rear**

- In busy areas like Remera or Kimironko, passengers may enter a minibus from the front (door near the driver) or from the back, especially when it's crowded.
- Similarly, they may exit from either end.
- This reflects **insertion and removal from both ends**, just like a deque.

Q8: At a Kigali restaurant, customers order food and are called when ready. How can queues model this process

Solution,

At a Kigali restaurant:

1. Customers arrive and place their orders at the counter.
2. Orders are processed **in the order they were received**.
3. When food is ready, customers are called to collect it.

How Queues Model This Process**:**

1. Linear Queue (FIFO – First In, First Out)

- **What it models**: The order in which food is prepared and delivered.
- **Why**:

    - The first customer to place an order is the **first** to receive their food.
    - Orders are handled **one after another**, in the order they arrived.
- **Example**:
    - If Alice orders first, then Bosco, then Chantal — the kitchen prepares **Alice's order first**, then **Bosco's**, then **Chantal's**.

This matches the behavior of a **linear queue**, where:

- Orders are **enqueued** when placed.
- Orders are **dequeued** when ready and called out.

Q9: At CHUK hospital, emergencies jump the line. Why is this a priority queue, not a normal queue

Solution,

Normal Queue (FIFO – First In, First Out):

- Patients would be treated **in the order they arrived**, no matter how serious their condition is.
- Example: If Jean came before Aline, Jean is treated first, even if Aline is in critical condition.
- **This would be dangerous** in a hospital setting.

Priority Queue:

- Patients are treated based on the **urgency of their condition**, **not** when they arrived.
- Emergency cases get **higher priority** and are moved ahead in the queue.
- Less serious cases **wait longer**, even if they arrived earlier.

CHUK uses a **priority queue** system because **saving lives** is more important than serving in order of arrival.
In healthcare, **urgency matters more than time**.

Q10: In a moto/e-bike taxi app, riders wait for passengers How would queues fairly match drivers and students

Solution,

Two Queues:

- **Passenger Queue**: Students waiting for rides.
- **Driver Queue**: Moto/e-bike drivers waiting for passengers.

How Matching Works:

- **FIFO (First-In, First-Out)**:
    - First student → matched with first available driver.