UNIVERSITY OF RWANDA

NAMES: UGUSHAKAKWAWE ONESPHORE

REG N: 224013212

COLLEGE: BUSSINESS INFORMATION AND TECHNOLOGY

YEAR2

MODULE:DATA STRUCTURE AND ALGORITHM

PART I- STAC

A. BASICS

Q1. How does this show the LIFO nature of stacks?

Solution

 How the MTN MoMo App Demonstrates LIFO (Last In, First Out)

In computer science, a **stack** is a data structure that follows the **LIFO** principle — **Last In, First Out**. That means the last item added (pushed) is the first one removed (popped).

Now, in the **MTN MoMo app**, when you're entering payment details:

- You go through steps like: **Enter amount → Choose recipient → Confirm details**.
- Each step you complete is like **pushing** that step onto a stack.
- When you press**Back**, the app removes the **last step you completed** — just like a **pop** operation in a stack.
- Real-Life Stack Analogy in the App

```
push
pop
```

✓ Why This Is LIFO

- The **last screen you visited** (e.g., confirmation) is the **first one removed** when you press back.
- You **don't jump to the first screen** — you go **backward one step at a time**, removing the most recent.

This behavior perfectly mirrors how a stack works in programming. The app's navigation is built like a stack — and pressing back is like popping the top item off.

Q2. Why is this action similar to popping from a stack?

This action is similar to popping from a stack because both involve **removing the most recent item** added to a sequence.

In computer science, a **stack** is a data structure that follows the **Last In, First Out (LIFO)** principle. That means the last item added (or "pushed") to the stack is the first one removed (or "popped").

📚 In the context of UR Canvas:

- Each navigation step (e.g., opening a module or page) is like **pushing** a new item onto the stack.
- Pressing "Back" is like **popping** the top item off the stack — it undoes the most recent action and returns you to the previous state.

So, just like a stack, UR Canvas keeps track of your navigation history in order, and "Back" removes the latest step, mimicking the pop operation.

Great question! The concept of **popping from a stack** shows up all over real-world programming. Here are some key scenarios where it's essential:

 1. **Navigation History in Web Browsers**

- Every page you visit is pushed onto a stack.
- Pressing the "Back" button pops the last page, returning you to the previous one.
- This mirrors how UR Canvas handles module navigation.

 2. **Function Calls and Recursion**

- Programming languages use a **call stack** to manage function execution.
- When a function is called, it's pushed onto the stack.
- When it finishes, it's popped off, and control returns to the previous function.
- This is especially important in recursive functions like calculating factorials or traversing trees.

 3. **Undo/Redo Systems**

- Applications like Photoshop or Word use stacks to track changes.
- Each action is pushed onto an "undo" stack.
- Pressing "Undo" pops the last action and reverses it.
- A separate "redo" stack can be used to reapply actions.

 4. **Depth-First Search (DFS) Algorithms**

- DFS uses a stack to explore paths in graphs or trees.
- Nodes are pushed as they're visited, and popped when backtracking.
- This is common in maze-solving, puzzle games, and AI pathfinding.

 5. **Expression Evaluation (e.g., Calculators)**

- Stacks help evaluate mathematical expressions, especially in postfix (Reverse Polish) notation.
- Operands are pushed, and operators pop values to compute results.

Stacks are simple but powerful. They help programmers manage order, history, and control flow in a way that's both intuitive and efficient. Want to see a code example of one of these in action?

B. APPLICATION

Q3. : How could a stack enable the undo function when correcting mistakes?

Stacks are perfect for implementing **undo** functionality because they follow the **Last In, First Out (LIFO)** principle:

- Every action (e.g., typing, deleting, editing) is **pushed** onto the stack.
- When the user wants to undo, the system **pops** the most recent action from the stack and reverses it.
- This ensures that the **last change** is undone **first**, just like peeling back layers.

 In BK Mobile Banking, if a user makes a transaction and wants to undo it (e.g., cancel or reverse), the system could use a stack to track recent actions and allow step-by-step reversal.

 Q4: How can stacks ensure forms are correctly balanced?

Stacks are ideal for checking **balanced structures**, like parentheses or matched fields in forms:

- When a user opens a section (e.g., starts filling a form field), the system **pushes** that onto the stack.
- When the user completes or closes the section, the system **pops** it off.
- If at the end the stack is **empty**, it means all fields were properly matched and closed.
- If not, it signals an error — like a missing field or unmatched entry.

📝 In Irembo registration forms, this helps ensure that every required field is filled and closed properly, preventing submission errors due to incomplete or mismatched data.

C. LOGICAL

 Q5: Which task is next (top of stack)?

Sequence:

1. Push “CBE notes” → Stack: [CBE notes]
2. Push “Math revision” → Stack: [CBE notes, Math revision]
3. Push “Debate” → Stack: [CBE notes, Math revision, Debate]
4. Pop() → Removes “Debate” → Stack: [CBE notes, Math revision]
5. Push “Group assignment” → Stack: [CBE notes, Math revision, Group assignment]

✅ **Top of stack (next task): Group assignment**

 Q6: Which answers remain in the stack after undoing 3 recent actions?

Assuming the student initially performed these actions:

- Push “Answer 1”
- Push “Answer 2”
- Push “Answer 3”
- Push “Answer 4”
- Push “Answer 5”

Then they **undo 3 actions** (i.e., Pop 3 times):

- Pop “Answer 5”
- Pop “Answer 4”
- Pop “Answer 3”

Remaining in the stack:

- **Answer 1**
- **Answer 2**

D . ADVANCED THINKINGS

Q7: How does a stack enable this retracing process?

Pop to Backtrack — Rwanda Air Booking

In a multi-step form (like Rwanda Air’s booking), each completed step is **pushed** onto a stack. When a passenger clicks “Back”:

- The system **pops** the last step (e.g., payment info).
- It then shows the previous step (e.g., seat selection).
- This allows **step-by-step retracing**, just like peeling layers off a stack.

✅ **Stack enables backward navigation** by removing the most recent action and restoring the previous state.

Q8: Show how a stack algorithm reverses the proverb.

Reverse “Umwana ni umutware” Using Stack

Algorithm:

1. Split the sentence into words: `["Umwana", "ni", "umutware"]`
2. Push each word onto the stack:

   Stack: [Umwana] Stack: [Umwana, ni] Stack: [Umwana, ni, umutware]

3. Pop each word to reverse:

Pop → "umutware" Pop → "ni" Pop → "Umwana"

✅ **Reversed Proverb:** “umutware ni Umwana”

📚 Q9: DFS in Kigali Public Library — Why Stack Beats Queue

Depth-First Search (DFS) explores as deep as possible before backtracking. A **stack** supports this by:

- Pushing new shelf sections (nodes) as they’re discovered.
- Popping to backtrack when reaching a dead end.
- This suits **deep exploration**, like searching inside nested shelves or categories.

In contrast, a **queue** (used in BFS) explores broadly, which is slower for deep targets.

✅ **Stacks prioritize depth**, making them ideal for focused, deep searches like in a library.

📲 Q10: Stack-Based Navigation in BK Mobile App

Suggested Feature: “Smart Backtrack”

- Every transaction view (e.g., details, receipt, confirmation) is **pushed** onto a stack.
- Users can tap “Back” to **pop** and return to the previous screen.
- Add a **“Jump to Last Viewed”** button that pops multiple steps to return to the last major transaction.

✅ This stack-based navigation makes history browsing intuitive and efficient.

PARTII –QUEUE

A. Basics

Q1: Restaurant in Kigali — FIFO Behavior

At the restaurant:

- Customers arrive and **enqueue** at the back of the line.
- The first customer to arrive is **served first** — **dequeue** from the front.

✅ This is **First-In, First-Out (FIFO)**: the earliest entry gets processed first, just like a queue in programming.

Q2: YouTube Playlist — Dequeue Operation

In a playlist:

- Videos are arranged in order.
- The **first video** plays automatically, then the next, and so on.
- Each played video is **dequeued** from the front.

✅ This mimics a queue where items are removed in the order they were added — perfect example of **automatic dequeue**.

🏢 B. Application

B. APPLICATION

Q3: RRA Offices — Real-Life Queue

At Rwanda Revenue Authority:

- People arrive and **enqueue** to pay taxes.
- Each person is served in turn, based on arrival time.

✅ This is a **real-life queue**: orderly, fair, and predictable — just like a queue data structure.

Q4: MTN/Airtel Service Centers — Queue Management

For SIM replacement:

- Requests are **queued** and processed one by one.
- No skipping or confusion — each customer knows their turn.

✅ Queues improve customer service by:

- Reducing wait-time anxiety.
- Ensuring fairness.
- Allowing staff to manage workload efficiently.

 C.LOGICAL

Q5: Equity Bank — Who's at the front?

Operations:

1. Enqueue("Alice") → Queue: [Alice]
2. Enqueue("Eric") → Queue: [Alice, Eric]
3. Enqueue("Chantal") → Queue: [Alice, Eric, Chantal]
4. Dequeue() → Removes "Alice" → Queue: [Eric, Chantal]
5. Enqueue("Jean") → Queue: [Eric, Chantal, Jean]

✅ **Front of the queue: Eric**

Q6: RSSB Pension — How FIFO Ensures Fairness

In a **FIFO (First-In, First-Out)** system:

- Applications are processed in the order they arrive.
- No skipping or favoritism.
- Everyone gets served based on **arrival time**, not status or influence.

✅This builds **trust and transparency**, especially in public services like RSSB.

D. Advanced Thinking

Q7: Queue Types in Rwandan Life

- **Linear Queue** → *Wedding Buffet Line*
  Guests line up, get served one by one, and leave — classic FIFO.
- **Circular Queue** → *Nyabugogo Bus Loop*
  Buses rotate through stops in a loop. When full circle is reached, they restart — no overflow, just wrap-around.
- **Deque (Double-Ended Queue)** → *Bus Boarding from Front/Rear*
  Passengers can enter or exit from either end. Useful for flexible access and faster flow.

✅ Each queue type reflects different **movement patterns** in daily life.

Q8: Kigali Restaurant — Enqueue Orders, Dequeue When Ready

- Customers place orders → **Enqueue**
- Kitchen prepares food in order
- When ready, orders are **Dequeued** and served

✅ This models a **task queue**: fair, organized, and efficient — just like a print queue or job scheduler.

Q9: CHUK Hospital — Why It's a Priority Queue

- Normal queue: first come, first served
- **Priority queue**: emergencies (e.g., trauma cases) jump ahead of routine checkups

✅ Medical urgency overrides arrival time — **life-saving logic** that ensures critical cases are handled first.

Q10: Moto/E-bike App — Fair Matching System

- Riders **Enqueue** when available
- Students **Enqueue** when requesting a ride
- System **Dequeues** both to match nearest rider with waiting passenger

✅ This ensures:

- **Fairness** (first available rider gets matched)
- **Efficiency** (minimizes wait time)
- **Transparency** (no hidden prioritization)