DATASTRUCTURES(STACK & QUEUE) APPLICATION EXAMPLES

II. STACK

Q1.Stack Analogy in MTN MoMo App

When you fill in payment details step-by-step, each screen or input is like a "push" operation onto a stack:

*Step 1: You enter the amount → pushed onto the stack, Step 2: You choose the recipient → pushed onto the stack, Step 3: You confirm the payment → pushed onto the stack.

Now, when you press Back, the app performs a "pop" operation:

- It removes the last screen you interacted with (e.g., confirmation).

- Pressing Back again removes the previous one (e.g., recipient).

- You're unwinding the steps in reverse order—exactly how a stack works.

Why This Is LIFO

- Last In: The confirmation screen was the last one added.

- First Out: It's the first one removed when you press Back.

- Each Back press pops the most recent layer off the stack, just like how a stack data structure behaves.

Q2.Stack Behavior in UR Canvas Navigation

Each time you open a new module or page in UR Canvas:

- That action is like a push onto a navigation stack.

- The app keeps track of your steps in the order they were taken.

When you press Back:

- It performs a pop—removing the last visited page from the stack.

- You're returned to the previous state, just like popping the top item off a stack.

Why It's a Stack (LIFO)

- Last In: The last module or screen you opened.

- First Out: It's the first one undone when you press Back.

- Earlier steps remain untouched until you pop them one by one.

Q3: Stack-Based Undo in BK Mobile Banking

In BK Mobile Banking, each transaction or user action (like entering an amount, selecting a recipient, confirming payment) can be pushed onto a stack:

- Push: Every step is stored in order—latest on top.

- Undo (Pop): If a mistake is made (e.g., wrong amount), the system can pop the last action to revert it.

Why This Works:

- LIFO logic: The most recent action (likely the mistake) is removed first.

- Controlled rollback: You can undo multiple steps in reverse order, preserving earlier correct actions.

- Example: If you accidentally confirm a payment, pressing "Undo" pops that confirmation, letting you re-edit before final submission.

This mirrors how text editors or design tools implement undo—each change is a stack entry, and undo pops the last one.

Q4: Balanced Parentheses in Irembo Forms

In Irembo registration, forms often have nested fields or conditional sections (e.g., if you select "Business," new fields appear). Ensuring these are correctly matched is like checking for balanced parentheses.

Stack-Based Validation:

- Push: When an opening field group is started (e.g., a new section), it's pushed onto the stack.

- Pop: When the section is completed or closed, it's popped off.

- Validation: At the end, the stack should be empty—meaning all opened sections were properly closed.

Why This Matters:

- Prevents mismatched or incomplete form entries.

- Ensures logical nesting—like not closing a "Business Details" section before finishing its subfields.

- Helps validate dynamic forms where visibility depends on previous inputs.

Stack Sequence Breakdown

*Push("CBE notes") → Stack: ["CBE notes"], Push("Math revision") → Stack: ["CBE notes", "Math revision"], Push("Debate") → Stack: ["CBE notes", "Math revision", "Debate"], Pop() → Removes "Debate" → Stack: ["CBE notes", "Math revision"], Push("Group assignment") → Stack: ["CBE notes", "Math revision", "Group assignment"]

Top of Stack (Next Task)

The top of the stack is "Group assignment" — it was the last task added and hasn't been removed.

Stack Before Undo

Assume the student performed these actions during the ICT exam:

*Push("Answer A"), Push("Answer B"), Push("Answer C"),Push("Answer D"),Push("Answer E")

Stack now looks like: ["Answer A", "Answer B", "Answer C", "Answer D", "Answer E"]

Undo with 3 Pops

Undoing 3 actions means popping the top 3 items:

- Pop() → "Answer E" removed, Pop() → "Answer D" removed, Pop() → "Answer C" removed

Remaining Answers in Stack

After undoing, the stack contains: ["Answer A", "Answer B"]

Q7: Pop to Backtrack in RwandAir Booking

In RwandAir's booking form, each step (e.g., selecting destination, choosing dates, entering passenger info) is pushed onto a stack as the user progresses.

- When the passenger presses Back, the app pops the last step.

- This allows retracing in reverse order, just like stack unwinding.

Why it works:

- Preserves the exact sequence of actions.

- Enables precise backtracking without affecting earlier steps.

- Mimics how function calls are unwound in programming.

Q8: Reversing "Umwana ni umutware" with a Stack

Let's reverse the proverb using a stack:

Step-by-step:

*Push each word: Stack: ["Umwana", "ni", "umutware"]

*Pop each word: Pop → "umutware", Pop → "ni", Pop → "Umwana"

Reversed Output: "umutware ni Umwana"

This shows how stacks reverse sequences by popping in LIFO order.

Q9: DFS in Kigali Public Library — Why Stack Wins

When a student performs a deep search (DFS) through shelves:

- A stack helps explore one path fully before backtracking.

- It pushes each shelf or section visited.

- When a dead end is reached, it pops back to the previous point.

Why stack suits DFS better than queue:

- Queue (FIFO) explores broadly (like BFS), not deeply.

- Stack (LIFO) dives into one branch, ideal for deep, focused search.

- Efficient for finding specific books buried in nested categories.

Q10: Stack-Based Feature for BK Mobile Transaction Navigation

Suggested Feature: "Smart Undo Trail"

- Every transaction view (e.g., details, edits, confirmations) is pushed onto a stack.

- User can pop back through their navigation trail.

- Add a "Jump to Last Viewed" button that pops multiple steps at once to return to a key transaction.

Benefits:

- Intuitive rollback of navigation.

- Preserves user flow without reloading entire history.

- Could include "Undo Last Action" for edits or reversals.

Q9: DFS in Kigali Public Library — Why Stack Wins

When a student performs a deep search (DFS) through shelves:

- A stack helps explore one path fully before backtracking.

- It pushes each shelf or section visited.

- When a dead end is reached, it pops back to the previous point.

Why stack suits DFS better than queue:

- Queue (FIFO) explores broadly (like BFS), not deeply.

- Stack (LIFO) dives into one branch, ideal for deep, focused search.

- Efficient for finding specific books buried in nested categories.

Q10: Stack-Based Feature for BK Mobile Transaction Navigation

Suggested Feature: "Smart Undo Trail"

- Every transaction view (e.g., details, edits, confirmations) is pushed onto a stack.

- User can pop back through their navigation trail.

- Add a "Jump to Last Viewed" button that pops multiple steps at once to return to a key transaction.

Benefits:

- Intuitive rollback of navigation.

- Preserves user flow without reloading entire history.

- Could include "Undo Last Action" for edits or reversals.

II. QUEUE

Q1: FIFO at a Restaurant in Kigali

When customers arrive at a restaurant:

- They're enqueued at the end of the line.

- The first customer to arrive is the first to be served.

This is classic FIFO behavior:

- First In: The earliest customer.

- First Out: That same customer is served before others.

- Later arrivals must wait their turn, just like items in a queue.

Q2: FIFO in a YouTube Playlist

In a playlist:

- Videos are queued in the order they were added.
- The next video (at the front of the queue) plays automatically.

This mirrors a dequeue operation:

- The video at the front is removed and played.
- The rest shift forward, maintaining order.

Q3: Real-Life Queue at RRA Offices

At RRA (Rwanda Revenue Authority):

- Taxpayers arrive and enqueue by taking a number or standing in line.
- They're served in the order of arrival.

This is a real-life queue:

- Prevents jumping ahead.
- Ensures fairness and structure.
- Matches FIFO logic: first come, first served.

Q4: Queue Management at MTN/Airtel Service Centers

SIM replacement requests are handled in sequence:

- Each request is enqueued upon arrival.
- Staff dequeues and processes them one by one.

How queues improve service:

- Avoids chaos and confusion.
- Reduces wait-time anxiety by showing progress.
- Enables tracking and prioritization if needed.

Who's at the Front in Equity Bank?

Operations:

* Enqueue("Alice") → Queue: ["Alice"], Enqueue("Eric") → Queue: ["Alice", "Eric"], Enqueue("Chantal") → Queue: ["Alice", "Eric", "Chantal"], Dequeue() → Removes "Alice" → Queue: ["Eric", "Chantal"], Enqueue("Jean") → Queue: ["Eric", "Chantal", "Jean"]

Front of the queue: Eric

Q6: Fairness in RSSB Pension Applications

RSSB processes applications in arrival order:

- Each application is enqueued when received.

- They're dequeued one by one, starting with the earliest.

How this ensures fairness:

- No skipping or favoritism.

- Everyone waits their turn.

- Predictable and transparent processing.

Q7: Real-Life Queue Types in Rwanda

Let's map each type:

- Linear Queue (Wedding Buffet):

Guests line up, move forward, and leave after serving.

One-way flow, no reuse of space.

- Circular Queue (Nyabugogo Bus Loop):

Buses rotate through stops and return to the start.

Efficient reuse of queue positions—ideal for transport loops.

- Deque (Bus Boarding Front/Rear):

Passengers can enter or exit from either end.

Flexible access—useful in crowded or fast-paced boarding.

Q8: Kigali Restaurant Orders

Customers enqueue their food orders:

- Orders are placed in sequence.

- When ready, each is dequeued and served.

Queue models:

- Tracks who ordered first.
- Ensures food is served in order.
- Can be extended with timestamps or priority (e.g., VIP orders).

Q9: CHUK Hospital — Priority Queue

In emergencies:

- Critical patients jump the line regardless of arrival time.
- Less urgent cases wait.

Why it's a priority queue:

- Items (patients) are served based on urgency, not order.
- Each has a priority level—emergencies > regular checkups.
- Ensures life-saving care is not delayed.

Q10: Fair Matching in Moto/E-bike Taxi App

Drivers and students are enqueued as they become available:

- Passengers are matched to the next available driver.
- If multiple drivers are idle, the one who's waited longest is matched first.

Queue logic ensures:

- Fairness: No driver is skipped.
- Efficiency: Reduces idle time.
- Can be enhanced with location-based priority or rating filters.