UNIVERSITY of RWANDA

COLLEGE OF BUSINESS AND ECONOMICS

SCHOOL OF BUSINESS

DEPARTMENT OF BUSINESS AND INFORMATION TECHNOLOGY(BIT)

LECTURER NAMES: RUKUNDO PRINCE

| | |
|---|---|
| NAMES | NTIRANDEKURA Pacifique |
| REG NO | 224011312 |

# BIT - Data structure Exercise - number 2

## Part I- STACK

## A. Basics

1. Operation: Push/Pop (LIFO) — In a stack, the last item added is the first removed. In the MTN MoMo app, when you fill payment details step-by-step, pressing back removes the last step
   **Q1: How does this show the LIFO nature of stacks?**

This example from the MTN MoMo app illustrates the **LIFO (Last In, First Out)** principle of stacks by showing how actions are reversed in the exact opposite order they were performed.

**Here's how the analogy works:**

- In a **stack**, each new item is placed on top of the previous ones. When you remove an item (using the "pop" operation), you always take the last one added**.**
- In the MTN MoMo app, as you fill in payment details step-by-step, each step is like a "push" onto the stack.
- When you press **back**, the app removes the most recent step—the last one you completed—just like a "pop" from a stack.

2. Operation: Pop (Undo) — Pop removes the top item. In UR Canvas, when you navigate course modules, pressing back undoes the last step. **Q2: Why is this action similar to popping from a stack?**

This action in UR Canvas is similar to **popping from a stack** because it follows the same logic of **LIFO (Last In, First Out)** behavior described in the surrounding content.

- In a **stack**, the **pop** operation removes the **top item**, which is the most recent one added.
- In **UR Canvas**, as you navigate through course modules, each step you take (e.g., opening a module or clicking into a section) is like adding an item to the stack.
- When you press **back**, the app undoes the last navigation step—just **B. Application**

3. Operation: Push (Add to stack) — New actions are added to the stack top. In BK Mobile Banking, transactions are added to history.

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

**Solution**

A stack could enable the undo function in **BK Mobile Banking** by keeping track of each transaction as a push operation—adding it to the top of the stack. Here's how this works based on the surrounding page content:

- Each time a user performs a transaction (like sending money or paying a bill), that action is pushed onto the stack, meaning it's stored at the top of the transaction history.

- If the user realizes they made a mistake and wants to undo it, the system can perform **a pop** operation—removing the most recent transaction from the stack.
- This ensures that only the latest action is undone first, which matches the **LIFO (Last In, First Out)** principle of stacks.

**4. Operation: Balanced Parentheses Check (Stack-based matching) —** Push opening bracket, pop when matching closing bracket is found.

In **Irembo registration forms,** data entry fields must be correctly matched.

**Q4:** How can stacks ensure forms are correctly balanced?

Stacks ensure forms are correctly balanced—like in Irembo registration—by following the same logic used in **balanced parentheses checking**, as described in the surrounding page content.

Here's how it works:

- When a user begins filling a form, each **opening field** (like starting a section or entering a value) is treated like an **opening bracket** and is **pushed** onto the stack.
- As the user completes or closes each field, the system checks for a **matching closing action**—like submitting or validating the field—and **pops** the corresponding item from the stack.
- If every opening field has a matching closing action and the stack is empty at the end, the form is **balanced**.

## C. Logical

5. **Operation: Push and Pop sequence.**

A student records tasks in a stack: Push("CBE notes"), Push("Math revision"), Push("Debate"), Pop(), Push("Group assignment")

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

**Solution**

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 the stack (next task): "Group assignment"**

This matches the LIFO behavior described earlier: the last item pushed is the first one accessed.

6. **Operation: Undo with multiple Pops**.
   During ICT exams, a student undoes 3 recent actions.
   **Q6**: Which answers remain in the stack after undoing?

This question illustrates how a **stack** supports the **undo operation** using multiple **pop actions**, following the **LIFO (Last In, First Out)** principle.

**Here's how it works:**

- Each action the student performs during the ICT exam is **pushed** onto the stack—added to the top.
- When the student undoes 3 recent actions, the system performs **three pops**, removing the **last three actions** one by one from the top.
- The **remaining answers** in the stack are those that were added **before** the last three actions.

## D. Advanced Thinking

**7. Operation: Pop to backtrack**.

In **RwandAir booking**, a passenger goes back step-by-step in the form.

**Q7:** How does a stack enable this retracing process?

**Solution**

This retracing process in the RwandAir booking system reflects the Pop to backtrack operation described in the surrounding page content, which is rooted in the LIFO (Last In, First Out) behavior of stacks.

**Here's how a stack enables this:**

- As the passenger fills out the booking form, each step—like selecting a destination, choosing a date, entering passenger details—is pushed onto the stack.
- When the passenger decides to go back, the system performs a pop operation, removing the most recent step from the stack.
- This allows the passenger to backtrack step-by-step, undoing actions in reverse order—starting from the last one taken.

7. **Operation: Push words, then Pop to reverse.**
8. To reverse "Umwana ni umutware", push each word and then pop.
   **Q8:** Show how a stack algorithm reverses the proverb.

To reverse the proverb **"Umwana ni umutware"** using a stack algorithm, we apply the **Push and Pop** operations described in the surrounding page content, which follow the **LIFO (Last In, First Out)** principle.

**Step-by-step Stack Algorithm:**

1. **Push each word** onto the stack:
   - Push("Umwana") → Stack: ["Umwana"]
   - Push("ni") → Stack: ["Umwana", "ni"]
   - Push("umutware") → Stack: ["Umwana", "ni", "umutware"]
2. Pop each word to reverse the order:

- Pop() → "umutware"
- Pop() → "ni"
- Pop() → "Umwana"

Here's a clear and contextual explanation of each question based on the surrounding page content:

---

**Q9: Why does a stack suit this case better than a queue?**
In a **Depth-First Search (DFS)**, like a student searching shelves in Kigali Public Library, the goal is to explore one path deeply before backtracking. A **stack** suits this because it stores the most recent shelf visited at the top. When the student needs to backtrack, they **pop** the last shelf and continue from the previous one. This **LIFO** behavior ensures deep exploration, unlike a queue which would explore shelves in a broad, shallow manner.

**Q10: Suggest a feature using stacks for transaction navigation.**
In the **BK Mobile app**, a useful feature could be a **"Backtrack History"** button. Each transaction viewed is **pushed** onto a stack. When the user wants to revisit previous transactions, pressing back would **pop** the last viewed transaction, showing the one before it. This allows intuitive, step-by-step navigation through transaction history, just like retracing steps in a stack.

---

**Part II- QUEUE**

**A. Basics**

**Q1: How does this show FIFO behavior?**
At a restaurant in Kigali, customers are served in the order they arrive. This is **FIFO (First In, First Out)**: the **first customer to e n queue** is the **first to be served**, just like removing from the front of a queue.

**Q2: Why is this like a de queue operation?**
In a YouTube playlist, the **next video plays automatically**, removing it from the front of the list. This mirrors a **de queue** operation, where the **first item added** is the **first one removed**.

**B. Application**

**Q3: How is this a real-life queue?**
At RRA offices, people waiting to pay taxes form a line. Each person joins at the rear (e n queue) **and is** served from the front (de queue)**. This is a classic** queue system**,** ensuring orderly service.

**Q4: How do queues improve customer service?**
In MTN/Airtel service centers, SIM replacement requests are handled in order. A **queue system** prevents jumping ahead, reduces confusion, and ensures **fair, predictable service**, improving customer satisfaction.

**C. Logical**

**Q5: Who is at the front now?**
Sequence:

- En queue(“Alice”), En queue(“Eric”), En queue(“Chantal”) → Queue: Alice, Eric, Chantal
- De queue() → Removes Alice
- En queue(“Jean”) → Queue: Eric, Chantal, Jean
  **Front of queue: Eric**

**Q6: Explain how a queue ensures fairness.**
At RSSB, pension applications are processed by arrival time. A **queue** ensures that **no one skips ahead**, and everyone is served in the **order they arrived**, promoting fairness and transparency.

**Q7: Explain how each maps to real Rwandan life.**

- **Linear queue**: People at a wedding buffet line up and are served in order.
- **Circular queue**: Buses at Nyabugogo loop through stops and return to the start.
- **Deque**: Boarding a bus from either front or rear door allows flexibility—like a **double-ended queue**.

**Q8: How can queues model this process?**
At a Kigali restaurant, customers **enqueue** their orders. When food is ready, the kitchen **dequeues** the order and calls the customer. This models a **service queue**, where requests are processed in order.

**Q9: Why is this a priority queue, not a normal queue?**
At CHUK hospital, emergencies jump the line. Unlike a normal queue, a **priority queue** allows urgent cases to be handled before others**,** based on importance—not arrival time.

**Q10: How would queues fairly match drivers and students?**
In a moto/e-bike taxi app, **students** enqueue ride requests. Drivers **are** matched in order, ensuring that **the** first student to request is the **first to be served**, maintaining fairness and efficiency.

---