BIT – Data Structure Exercise – Number 2

Part I – STACK

A. Basics

Q1: How does this show the LIFO nature of stacks?
In the MTN MoMo app, when filling payment details, the process is step-by-step: for example, Step 1: Enter phone number, Step 2: Enter amount, Step 3: Confirm. If you press *back*, the system removes the last action first (Step 3). This is exactly how a stack works using **Last in, First Out (LIFO)**.

- Last step (Confirm) = first to be removed.
- Second last step (Enter amount) = removed after the confirm step.
  Thus, the app demonstrates the LIFO principle of stacks.

Q2: Why is this action similar to popping from a stack?
In UR Canvas, while navigating modules, pressing *back* undoes the **most recent action**. This resembles the **Pop() operation**, because:

- A stack stores steps like Module 1 → Module 2 → Module 3.
- When you press *back*, Module 3 (top of stack) is removed.
- You are then returned to Module 2.
  Therefore, "back" is equivalent to popping from a stack.

B. Application

Q3: How could a stack enable the undo function when correcting mistakes?
When performing actions in BK Mobile Banking, every step (like entering PIN, choosing beneficiary, confirming amount) can be pushed onto a stack. If you make an error, the undo function simply pops the last action, taking you back to the previous step.
Example:

- Push("Enter PIN")
- Push("Enter Amount")
- Push("Confirm")
  If you undo, Pop() removes "Confirm". Another Pop() removes "Enter Amount". You are safely back at "Enter PIN".
  This way, stacks allow correcting mistakes without disturbing earlier steps.

Q4: How can stacks ensure forms are correctly balanced?
Stacks are useful in checking balance of fields, brackets, or tags. For example, in Irembo registration:

- When you open a field (like "Full Names [ ]"), push it into the stack.
- When you close or finish that field, pop it out.
  If all pushes and pops match perfectly, then the form is complete and balanced.
  If the stack is not empty at the end, it means some fields were left unmatched. This prevents errors in data entry.

C. Logical

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

- 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]

☞ The top of the stack = **Group assignment**.

Q6: Which answers remain in the stack after undoing?
Suppose a student answered 5 questions in ICT exam. If they undo 3 recent actions, those 3 answers are popped out. The stack will now contain only the first 2 answers.
Example:

- Stack before undo = [Q1 answer, Q2 answer, Q3 answer, Q4 answer, Q5 answer]
- Undo (pop) × 3 = removes Q5, Q4, Q3
- Remaining = [Q1 answer, Q2 answer]

Thus, earlier work remains safe, while the recent mistakes are undone.

D. Advanced Thinking

Q7: How does a stack enable this retracing process?
In RwandAir booking, each step (Choose destination → Select date → Enter passenger info →

Confirm) is pushed onto the stack. When a passenger decides to go back, the system pops the last step and returns to the previous one.
This way, stacks allow **step-by-step retracing** without jumping randomly.

Q8: Show how a stack algorithm reverses the proverb.
Proverb = "Umwana ni umutware"
Steps:

- Push("Umwana")
- Push("ni")
- Push("umutware")
  Stack = [Umwana, ni, umutware]
  Now Pop one by one:
- Pop → "umutware"
- Pop → "ni"
- Pop → "Umwana"
  Reversed proverb = **"umutware ni Umwana"**

Stacks are powerful for reversing sentences, numbers, and even data sequences.

Q9: Why does a stack suit this case better than a queue?
In the Kigali Public Library, when searching books using DFS, the student goes deep into one shelf before backtracking.

- Stack fits because it stores the most recent exploration and backtracks in reverse order (LIFO).
- A queue, on the other hand, would search level by level (BFS), which is not depth-first.

Thus, stacks are better for DFS because of their backtracking ability.

Q10: Suggest a feature using stacks for transaction navigation.
In BK Mobile app, stacks can be used for **Transaction History Navigation**:

- Every time a user views a transaction, it is pushed onto a stack.
- The user can press "Back" to pop the last viewed transaction and return to the earlier one.
- Another feature could be "Redo", which re-pushes popped items.

This makes navigation smooth and user-friendly.

Part II – QUEUE

A. Basics

Q1: How does this show FIFO behavior?
In a Kigali restaurant, the first customer who entered the line is served first, while the last customer waits. This is the **First In, First Out (FIFO)** principle: service is done in the exact order of arrival.

Q2: Why is this like a dequeue operation?
In YouTube playlists, when one video finishes, the next video (which was first in the queue) automatically starts. That is exactly what a **Dequeue()** does – removes the front element.

B. Application

Q3: How is this a real-life queue?
At RRA offices, people line up to pay taxes. The first person to arrive is served first. Others wait until the person in front is done. This is a clear real-life application of queue structures.

Q4: How do queues improve customer service?
Queues ensure fairness: customers are served in the order they arrived. This prevents arguments and confusion, reduces waiting stress, and improves transparency in MTN/Airtel service centers.

C. Logical

Q5: Who is at the front now?
Operations:

- Enqueue(Alice) → [Alice]
- Enqueue(Eric) → [Alice, Eric]
- Enqueue(Chantal) → [Alice, Eric, Chantal]
- Dequeue() removes Alice → [Eric, Chantal]
- Enqueue(Jean) → [Eric, Chantal, Jean]

☞ Front = **Eric**.

Q6: Explain how a queue ensures fairness.
In RSSB pension applications, the first applicant is the first to be served. A queue ensures that no one jumps ahead, so fairness and trust are maintained. Everyone is treated equally according to arrival time.

D. Advanced Thinking

Q7: Explain how each maps to real Rwandan life.

- **Linear queue**: At a wedding buffet, guests line up once, and when served, they leave the queue.
- **Circular queue**: Buses at Nyabugogo return after a trip and rejoin the queue for another round.
- **Deque**: In boarding buses, passengers can enter from either the front or rear doors, so insertion/removal is from both ends.

Q8: How can queues model this process?
In a Kigali restaurant, customers place orders which are enqueued. The kitchen processes orders one by one. Once food is ready, the order is dequeued, and the customer is served. This models order serving perfectly.

Q9: Why is this a priority queue, not a normal queue?
At CHUK hospital, emergencies are given higher priority than normal cases. Even if someone arrived earlier, a critical patient will be served first. This breaks the FIFO rule and follows **priority queue logic**.

Q10: How would queues fairly match drivers and students?
In moto/e-bike apps, drivers are queued based on availability. When a student requests a ride, the first driver in the queue is matched (dequeued). This ensures fairness and prevents idle drivers from being skipped.

References:

• Goodrich, M. T., Tamassia, R., C Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.

- Wirth, N. (1976). Algorithms + Data Structures = Programs. Prentice-Hall.

- GeeksforGeeks. (n.d.). Stack Data Structure. Retrieved from **https://www.geeksforgeeks.org/stack-data-structure/**

- GeeksforGeeks. (n.d.). Queue Data Structure. Retrieved from **https://www.geeksforgeeks.org/queue-data-structure/**

- TutorialsPoint. (n.d.). Data Structures Tutorial. Retrieved from **https://www.tutorialspoint.com/data_structures_algorithms/index.htm**

- MTN Rwanda. (n.d.). MTN MoMo App. Retrieved from https://mtn.co.rw

- Bank of Kigali. (n.d.). BK Mobile Banking App. Retrieved from https://bk.rw

- https://irembo.gov.rwIrembo Rwanda. (n.d.). IremboGov Services. Retrieved from **https://irembo.gov.rw**