NAMES: MUTABAZI Ntwari Guy Landry
REG N0: 224014868
DEPARTMENT: Business Information Technology
MODULE: DATA STRUCTURE AND LOGARITHM

Data structure Exercise

Part I- STACK

A. Basics

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

ANSWER: A stack is a collection that supports two principal operations: push (add an item to the top) and pop (remove the item from the top). Because the most recently pushed item is the first to be popped, stacks follow LIFO behavior. The MTN MoMo example (entering payment details step-by-step, pressing back removes the last step) is a direct analogy: the most recent input (last in) is removed first (first out when popping), demonstrating LIFO.

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

ANSWER: In UR Canvas during navigation of course modules, the web removes or undoes the most recent navigation state. This mirrors pop from a stack: the current state is on top and is the one that gets removed first. Because each back action undoes only the most recent change, the operation matches a stack pop.

B. Application

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

ANSWER: Stacks are commonly used to implement undo features. In BK Mobile Banking, suppose the app stores each edit to a form or each navigation step as a new stack entry; pressing undo pops the last action and reverts to the previous state. The stack's property ensures changes are undone in the reverse order they occurred, preventing inconsistent or partial undoing.

Q4: How can stacks ensure forms are correctly balanced?

ANSWER: A classical algorithmic use is balanced parentheses checking: push every opening bracket onto a stack; when a closing bracket appears, pop the stack and check whether the top matches. This ensures nested structures (e.g., form sections, fields) are

correctly opened and closed in proper order. For data-entry systems (e.g., Irembo registration forms), stacks can validate that nested inputs are properly balanced before submission.

C. Logical

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

ANSWER: The next task (top) is "Group assignment".

Q6: Which answers remain in the stack after undoing?

ANSWER: After three pops the stack will remain empty, as we're following LIFO rule, then all of three remained tasks will be removed by starting with last in.

D. Advanced Thinking

Q7: How does a stack enable this retracing process?

ANSWER: In a multi-step booking form (RwandAir example), each page or input state can be pushed onto a stack as the user progresses. If the passenger clicks "back", the system performs a pop to return to the previous state. Repeated pops retrace the user journey step-by-step.

Q8: Show how a stack algorithm reverses the proverb.

ANSWER: push("Umwana"), push("ni"), push("umutware")
Then pop repeatedly to output the words: pop→"umutware", pop→"ni", pop→"Umwana". The popped sequence yields the reversed sentence: "umutware ni Umwana". This demonstrates a simple stack-based reversal algorithm.

Q9: Why does a stack suit this case better than a queue?

ANSWER: Depth first search can be implemented with either recursion or an explicit stack. For searching deep shelves in Kigali Public Library (looking for items in the deepest branch before siblings), a stack is ideal is that, it allows the algorithm to follow a path deeply (push children as it goes) and backtrack by popping when dead ends are reached.

Q10: Suggest a feature using stacks for transaction navigation.

ANSWER: Implement a transaction state stack that stores the last states for transaction edits. Provide a one-tap "Undo last change" that pops the top state. Additionally, a "History breadcrumbs" view could show the stack contents (most recent first) and allow popping multiple entries to revert to any earlier state.

Part II- QUEUE

A. Basics

Q1: How does this show FIFO behavior?

ANSWER: A queue supports enqueue (add at the rear) and dequeue (remove from the front). This enforces FIFO, the first item added is the first removed. Real-life examples: customers lining up at a Kigali restaurant are served in the order they arrived

Q2: Why is this like a dequeue operation?

ANSWER: ; the first to enter the queue is the first to be served. The YouTube playlist that auto-plays the next video acts like a dequeue: the next item in sequence is taken for playback.

B. Application

Q3: How is this a real-life queue?

ANSWER: At RRA offices or MTN/Airtel service centers, people form queues when submitting jobs such as tax payments or SIM replacements. Queues ensure predictable and fair ordering each request is handled in the order received.

Q4: How do queues improve customer service?

ANSWER: Queue management also allows tracking and metrics (average waiting time, throughput), enabling better customer service.

C. Logical

Q5: Who is at the front now?

ANSWER: In-front there will be Eric.

Q6: Explain how a queue ensures fairness.

ANSWER: Queues guarantee fairness because each application or request is processed strictly by arrival order no later request overtakes an earlier one.

D. Advanced Thinking

Q7: Explain how each maps to real Rwandan life.

ANSWER: Linear queue: a single line where elements leave at the front — like guests taking plates once at a wedding buffet.

Circular queue: fixed-size buffer where rear wraps to front; models buses that loop (Nyabugogo buses) because positions are reused.

Deque (double-ended queue): insertion/removal at both ends — like boarding a bus from front and rear simultaneously.

Q8: How can queues model this process?

ANSWER: In a Kigali restaurant, each customer places an order, which is added to the system in the order it arrives. This is equivalent to an enqueue operation: the new order goes to the back of the queue. The kitchen then prepares meals in sequence. When the first order is ready, it is dequeued from the front of the queue, and the customer is called to collect it.

Q9: Why is this a priority queue, not a normal queue?

ANSWER: At CHUK hospital, emergencies are handled before routine cases. That is a priority queue: items have priorities and the highest-priority item is dequeued first, not necessarily the earliest arrival.

Q10: How would queues fairly match drivers and students?

In a moto/e-bike taxi app, both drivers and students can be managed using queues to ensure fairness. When a student requests a ride, their request is enqueued into a waiting line. Similarly, available drivers are enqueued into a separate queue. The system then performs dequeue operations from both queues simultaneously: it removes the student at the front of the passenger queue and the driver at the front of the driver queue, and matches them together.

Conclusion,
Stacks (LIFO) and queues (FIFO) are simple but essential data structures that model many real-world processes such as undo/backtracking and service lines, and power efficient algorithms like DFS and BFS. Choosing the right structure (and its variants, e.g., priority queues or deques) leads to fairer, clearer, and more reliable system and app designs.