Data Structures: Stack and Queue Exercises

In computer science, data structures are ways of organizing and storing data so that they can be accessed and modified efficiently. Two of the most fundamental data structures are Stack and Queue. Both have unique ways of handling data, and they are widely used in everyday applications such as banking apps, navigation systems, and even messaging platforms.

Part I: Stack

A stack is a linear data structure that follows the Last-In, First-Out (LIFO) principle. This means the last item added (pushed) to the stack is the first one to be removed (popped). Imagine a pile of plates: the last plate placed on top is the first one you take off.

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

In the MTN MoMo app, when filling payment details step by step, pressing the back button removes the most recent step. This is LIFO because the last entered detail is the first to be undone.

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

In UR Canvas, pressing back undoes the last navigation. This is like popping the top item from a stack, removing the most recent action first.

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

Each action is pushed onto the stack. When undo is pressed, the last action (on top) is popped. This ensures mistakes can be corrected in reverse order of execution.

Q4: How can stacks ensure forms are correctly balanced?

Stacks help track opening and closing brackets (or form fields). Each opening is pushed, and each closing pops the stack. If all match correctly, the form is balanced.

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

After the sequence Push(CBE notes), Push(Math revision), Push(Debate), Pop(), Push(Group assignment), the top of the stack is 'Group assignment'.

Q6: Which answers remain in the stack after undoing?

If a student undoes 3 actions, the last three pushed answers are popped. Only the earlier answers remain in the stack.

Q7: How does a stack enable this retracing process?

In RwandAir booking, each step is pushed on the stack. When going back, steps are popped one by one, allowing retracing.

Q8: Show how a stack algorithm reverses the proverb.

Push each word of 'Umwana ni umutware' onto the stack: [Umwana, ni, umutware]. Then pop them out: 'umutware ni Umwana'.

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

DFS uses a stack because it explores deeply by following one branch completely before backtracking. A queue would follow breadth-first, not depth.

Q10: Suggest a feature using stacks for transaction navigation.

A stack can be used in BK Mobile app so users can move forward/backward in transaction history, just like navigating steps in order.

Python Example: Stack

Below is a simple implementation of a stack in Python using lists:

```
stack = []
stack.append('CBE notes')
stack.append('Math revision')
stack.append('Debate')
print('Stack after pushes:', stack)
stack.pop()
print('After popping:', stack)
```

```
stack.append('Group assignment')
print('Final Stack:', stack)
```

Part II: Queue

A queue is another linear data structure but follows the First-In, First-Out (FIFO) principle. This means the first item added to the queue is the first one removed. Think of people standing in line at a bank: the first person to arrive is served first.

Q1: How does this show FIFO behavior?

At a restaurant in Kigali, customers are served in order. The first to join the line is the first served.

Q2: Why is this like a dequeue operation?

In a YouTube playlist, the next video plays automatically. This is like dequeuing the first element.

Q3: How is this a real-life queue?

At RRA offices, people form a line to pay taxes. The first in line is the first processed, like enqueue/dequeue.

Q4: How do queues improve customer service?

Queues maintain order so customers are served fairly and efficiently, avoiding chaos.

Q5: Who is at the front now?

After the sequence Enqueue(Alice), Enqueue(Eric), Enqueue(Chantal), Dequeue(), Enqueue(Jean), the front is 'Eric'.

Q6: Explain how a queue ensures fairness.

Queues process tasks in arrival order. This ensures no one skips ahead, creating fairness.

Q7: Explain how each maps to real Rwandan life.

Linear queue: people waiting at a buffet. Circular queue: buses looping at Nyabugogo. Deque: boarding a bus from both front and rear.

Q8: How can queues model this process?

At a restaurant, orders are enqueued as placed. Once food is ready, it is dequeued and served.

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

At CHUK hospital, emergencies are handled first regardless of arrival order. This is priority, not FIFO.

Q10: How would queues fairly match drivers and students?

In moto/e-bike apps, students are enqueued as requests, drivers are dequeued in order. This ensures fair matching of passengers and drivers.

Python Example: Queue

Below is a simple implementation of a queue in Python using collections.deque:

```
from collections import deque
queue = deque()
queue.append('Alice')
queue.append('Eric')
queue.append('Chantal')
print('Queue after enqueues:', queue)
queue.popleft()
print('After dequeue:', queue)
queue.append('Jean')
print('Final Queue:', queue)
```