COLLEGE OF BUSINESS

BIT. LEVEL 2

MODULE: DATA BASE MANAGEMENT SYSTEM

NAMES: NIYIGENA Francois

REG NO:224007241

ASSIGNMENT-2
BIT-1 .DATA STRUCTURE

Part I – STACK

1.A stack is a data structure that follows LIFO (Last In, First Out) order:

Push = add something on top.

Pop = remove the most recent thing you added.

Example with the MTN MOMO app

When you're filling in payment details:

Step 1 – Choose recipient.

Step 2 – Enter amount.

Step 3 – Confirm payment.

These steps are placed like items in a stack:

Push Step 1 → Stack = [Step 1]

Push Step 2 → Stack = [Step 1, Step 2]

Push Step 3 → Stack = [Step 1, Step 2, Step 3]

Now, when you press Back:

The last step (Step 3) is the first one removed (popped).

If you press back again, Step 2 is removed next.

You only return to Step 1 after removing Step 3 and Step 2.

for

2.When you use ''UR Canvas'' and move through modules step by step, each action you take (like opening a page or going deeper into a module) gets placed "on top" of your history

just like adding an item on top of a stack.

When you press ''Back'', the system removes the ''last thing you did'' (the top of the stack) and takes you back to the previous step.

This is just like a ''Pop'' operation on a stack:

''Pop'' always removes the topmost item.

''Back'' always undoes your most recent action.

So, pressing ''Back'' in UR Canvas is the same idea as popping the last item from a stack , you go back in the exact reverse order of how you moved forward.

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

- Each new action is pushed onto the stack. If a mistake happens, the system can pop the last action and undo it. This way, only the most recent action is removed, while earlier work stays safe.

Q4: How can stacks ensure forms are correctly balanced?

- For every opening field or bracket, the system pushes it on the stack. When a closing match appears, the system pops the top. If all items are popped correctly and the stack is empty at the end, the form is balanced. If not, something is unmatched.

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

Steps:

Push “CBE notes” → [CBE notes]

Push “Math revision” → [CBE notes, Math revision]

Push “Debate” → [CBE notes, Math revision, Debate]

Pop() removes “Debate” → [CBE notes, Math revision]

Push “Group assignment” → [CBE notes, Math revision, Group assignment]

- Answer: Group assignment is on top.

Q6: Which answers remain in the stack after undoing?

- If a student undoes 3 recent actions, the stack pops 3 items off the top. Only the earlier answers (those before the last 3) remain in the stack.

Q7: How does a stack enable this retracing process?

- Each booking step is pushed onto the stack. When the passenger presses back, the app pops the last step and returns to the previous one. This allows retracing step-by-step in reverse order.

Q8: Show how a stack algorithm reverses the proverb “Umwana ni umutware”.

Steps:

Push words: [Umwana, ni, umutware]

Pop → “umutware”

Pop → “ni”

Pop → “Umwana”

Reversed: “umutware ni Umwana”

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

- A stack follows deep search (DFS): it goes deeper into one shelf before backtracking. A queue would do BFS (breadth-first), checking all shelves level by level, which is slower for deep searching. That’s why a stack is better.

Q10: Suggest a feature using stacks for transaction navigation.

- A “Back to Previous Transaction” feature could use a stack. Each viewed transaction is pushed, and pressing back pops the last one, taking the user step-by-step through history.

Part II – QUEUE

Q1: How does this show FIFO behavior?

- In a restaurant, the first customer to arrive is the first to be served, while the last customer waits. That is exactly FIFO (First In, First Out).

Q2: Why is this like a dequeue operation?

- In a YouTube playlist, the next video at the front of the queue automatically plays first (dequeues). When it ends, the next one in line plays, just like removing items from the front of a queue.

Q3: How is this a real-life queue?

- At RRA offices, people line up in the order they arrive. The first person in line pays first, while new arrivals join at the back. That’s exactly how enqueue and dequeue work.

Q4: How do queues improve customer service?

- Queues make service fair and orderly, so no one jumps ahead. Customers are served in the sequence they arrived, reducing confusion and conflict.

Q5: Who is at the front now?

Steps:

Enqueue(Alice, Eric, Chantal) → [Alice, Eric, Chantal]

Dequeue() removes Alice → [Eric, Chantal]

Enqueue(Jean) → [Eric, Chantal, Jean]

- Answer: Eric is at the front.

Q6: Explain how a queue ensures fairness.

- In RSSB, applications are handled in arrival order. A queue ensures that no one skips ahead; everyone waits their turn, which guarantees fairness.

Q7: Explain how each maps to real Rwandan life.

Linear queue: People at a wedding buffet — first person serves food first.

Circular queue: Buses at Nyabugogo loop around and rejoin the line after trips.

Deque: Boarding a bus from both front and rear doors — people can enter from either end.

Q8: How can queues model this process?

- Each food order is enqueued when placed. The kitchen prepares them in the same order, and when food is ready, it is dequeued and delivered to the right customer.

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

- At CHUK, emergencies are treated first, even if they arrived later. This breaks normal FIFO order. That's why it is a priority queue.

Q10: How would queues fairly match drivers and students?

- Each driver and student is placed in a queue. The app dequeues the next available driver and matches them to the next waiting student. This ensures no one is skipped — it's fair and orderly.