NAME: Sonia UWIMANA

REG no: 224010032

ASSIGNMENT 2: DATA STRUCTURE

Part 1: STACK

INTRODUCTION

What is a Stack?

- A stack is a linear (or sequential) data structure that follows the LIFO principle: *Last In, First Out*. That means the *last* item you insert into the stack is the *first* one you remove.

There are 2 main operations on stack:

1. Push (): Add an element x to the *top* of the stack.
2. Pop (): Remove and return the element at the *top* of the stack.

A. Basic

1.**MTN MoMo app — filling payment details step-by-step; pressing back removes the last step**

How this shows LIFO nature of stacks:

- When you perform each step in filling out payment details (e.g. entering amount → recipient → confirm → etc.), the app presumably "records" each step (or state) in order.
- The *last* step you did is the one that gets removed first when you press "back." For example, if you input "confirm" last, pressing back would undo that "confirm" step, before undoing earlier ones.
- That is exactly LIFO: the *most recent* "push" (i.e. the last input / step) is the *first* to be "popped" (i.e. removed when going back).

So, the "back" action works like pop: it removes the top/most recent step. That matches stack behavior

2. **UR Canvas — navigating course modules, pressing back undoes the last step**

Why is this similar to popping from a stack:

- Each navigation (e.g. clicking into Module 1, then Module 1.1, then Module 1.1.2, etc.) can be thought of like pushing that module onto a navigation history stack.

- Pressing back undoes the last navigation: it takes you back from Module 1.1.2 to Module 1.1, etc. That is "pop" in action: remove the most recent item (the current module) to return to the previous.
- So, pressing back corresponds to "pop the top" of the stack of navigation states / pages, which matches the pop operation in stacks.

B. Application

Q3: How could a stack enable the undo function when correcting mistakes (in BK Mobile Banking or similar)?

- The idea is that every time a user does something (types of an entry, submits a field, etc.), you **push** that action onto an *undo stack*.
- If the user wants to undo, you **pop** the top action — this is the most recent one — and reverse its effect.
- Thus, you always undo the most recent change first (because of LIFO).

So, a stack lets you store history of actions, so that undoing always removes the latest, giving correct behavior.

Q4: How can stacks ensure forms are correctly balanced (balanced parentheses check) in something like Irembo registration forms?

If there are fields or inputs that require pair-matching (e.g. opening bracket "(", closing ")"; perhaps tags, nested fields, or matching "if / endif" blocks, etc.), you can use a stack to verify correctness.

In forms, that ensures that every opening has a matching closing, order is correct (you can't close something before what's most recently opened), etc. This is exactly how "balanced parentheses" problems are solved.

C. Logical

Q5: A student records tasks in a stack:

Push ("CBE notes"),

Push ("Math revision"),

Push("Debate"),

Pop (),

Push ("Group assignment")

Answer: *Group assignment* is the next/top task.

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

- Suppose the stack before undoing has some sequence of answers/actions. For example, imagine stack is:

[Answer1, Answer2, Answer3, Answer4, Answer5] (with “Answer5” being the top, most recent)

- Undoing 3 pops means remove top three:
    - Pop → remove Answer5
    - Pop → remove Answer4
    - Pop → remove Answer3
- The stack then has remaining: [Answer1, Answer2] with *Answer2* now top.

D. Advanced Thinking

Q7: Operation: Pop to backtrack (in RwandAir booking, going back step-by-step in the form). How does a stack enable this retracing process?

- Each time the user moves forward in the form (selecting flight, entering passenger info, seat selection, payment, etc.), you push the current state (or page / step) onto a stack.
- If user presses “back”, you pop the top state; the system returns to that previous page/state.

Q8: Operation: Push words, then Pop to reverse. To reverse “Umwana ni umutware”, push each word then pop. Show how a stack algorithm reverses the proverb.

- Given the proverb: **“Umwana ni umutware”**
- Step by step:
    - Push “Umwana” → stack: [Umwana]
    - Push “ni” → stack: [Umwana, ni]
    - Push “umutware” → stack: [Umwana, ni, umutware] (top is “umutware”)
    - Pop → remove “umutware” → output: “umutware”
    - Pop → remove “ni” → output: “ni”
    - Pop → remove “Umwana” → output: “Umwana”
- Reversed proverb becomes: **“umutware ni Umwana”**

Q9: Operation: DFS using a stack. A student searches shelves in Kigali Public Library (deep search). Why does a stack suit this case better than a queue?

- DFS (Depth First Search) aims to go as deep as possible along one branch/path before backtracking.

- A stack naturally supports that: you push next nodes to explore, always pop the most recently added, so you move deeper first.
- If you used a queue instead (like BFS), you'd explore all shelves at one “distance” (or “level”) before going further, which might not be efficient if you want to reach some deep shelf quickly.

Q10: Operation: Push/Pop for navigation. In BK Mobile app, moving through transaction history uses push and pop. Suggest a feature using stacks for transaction navigation.

Here are some possible features:

1. **“Jump back / Forward” navigation**
    - Each transaction viewed is pushed onto a stack.
    - A “back” button pops the current transaction and returns to the previous.
    - Also maintain a “redo” stack: when you go back, the popped transaction gets pushed to a redo stack; pressing a “forward” button pops from redo to return.
2. **Bookmarks with undo**
    - User marks certain transactions as “important” → push into a bookmark stack.
    - If they unbook mark, pop from that stack.
3. **Search path history**
    - The app could store the sequence of filters and search results the user applies (dates, amount filters, categories), pushing each filter state.
    - A “back to previous filter” action would pop the last filter state, restoring previous view.
4. **Transaction editing preview**
    - Suppose user edits a transaction: change amount, recipients etc., but before final confirmation. Each edit step can be pushed.
    - If user changes their mind, they can undo the last edit(s) by popping.

PART II_ QUEUE

INTRODUCTION

What Is a Queue?

- A *queue* is a linear (sequential) data structure or Abstract Data Type (ADT) in which items are added at one end (called the rear or tail) and removed from the other end (called the front or head). It follows the **FIFO** principle: *First In, First Out*. That means the first element you insert is the first one you remove.

There are 2 main operations of queue:

1. enqueue(x): Add element **x** at the **rear** of the queue.
2. dequeue (): Remove and return the element at the **front** of the queue.

A. Basics

Q1: How does "At a restaurant in Kigali, customers are served in order" show FIFO behavior?

- Because the first customer to arrive is the first to be served. Later arrivals wait behind, so the arrival order determines service order. That's exactly the FIFO rule: enqueue when someone arrives at the rear, dequeue when someone is served from the front.

Q2: Why is "In a YouTube playlist, the next video plays automatically" like a dequeue operation?

- Because once you've watched (i.e. processed) a video at the front of the playlist, it is removed (or considered done), and the next video (which was queued) becomes the front and then plays. That mimics the *dequeue* operation (remove from front). The playlist is like a queue of videos.

B. Application

Q3: How is "At RRA offices, people waiting to pay taxes form a line" a real-life queue?

- People arriving and joining at the end of the line → Enqueue.

- The person at the front gets served (pays taxes) → Dequeue.
- That ensures fairness (first-come, first-served), avoids chaos, and uses order of arrival to decide who is served first.

Q4: How do queues improve customer service at MTN/Airtel SIM replacement centers (processing requests in order)?

- By ensuring each customer is treated according to when they arrive.
- It prevents someone who comes later from being served earlier without due reason.
- It makes wait times predictable, avoids frustration, and helps organize staff's work.
- Overall, it reduces disputes or perceptions of unfairness.

C. Logical

Q5: In the sequence Enqueue("Alice"), Enqueue("Eric"), Enqueue("Chantal"), Dequeue (), Enqueue("Jean") — who is at the front now?

Let's track:

1. Enqueue("Alice") → queue: [Alice]
2. Enqueue("Eric") → [Alice, Eric]
3. Enqueue("Chantal") → [Alice, Eric, Chantal]
4. Dequeue () → removes "Alice" → [Eric, Chantal]
5. Enqueue("Jean") → [Eric, Chantal, Jean]

So, **Eric** is now at the front. Alice has been removed.

Q6: Explain how a queue ensures fairness when "RSSB pension applications are handled by arrival order."

- Because first-submitted applications are processed first.
- No one who applied after someone else can "jump the queue" under normal queue rules.
- Everyone waits their turn in order of arrival, so priority is not given based on any other criteria (unless there is a priority scheme).
- Ensures equality among applicants with respect to time.

D. Advanced Thinking

Q7: Operation: Different queue types. Examples:

- Linear queue = people at a wedding buffet.
- Circular queue = buses looping at Nyabugogo.
- Deque = boarding a bus from front/rear.

Explain how each maps to real Rwandan life.

- **Linear queue**: People line up once at the buffet — start of line is front, end is rear; you serve in arrival order, no wrapping around. Everyone goes through once.
- **Circular queue**: Buses that go in a loop: when one bus finishes a route, it returns to the start and picks more passengers. The set of buses operating in a loop is analogous to a circular buffer: capacity is reused continuously. Or consider taxis/motos cycling in a fixed area.
- **Deque (double-ended queue)**: You can enter or leave from both ends. In Rwandan life, boarding a bus from front or rear is an example: maybe passengers enter front, or sometimes allowed from back, or for goods at back, etc. Also, for example goods being loaded into a van from back, offloaded from the front, etc. It gives more flexibility.

Q8: Operation: Enqueue orders, Dequeue when ready. At a Kigali restaurant, customers order food and are called when ready. How can queues model this process?

- Enqueue: as customers' orders are placed, they are added to a service queue.
- Dequeue: when an order is completed, the customer whose order was placed earliest (front of queue) is called / served.
- There might be multiple queues if different food types or priority customers, but in simple case one queue works.
- This also means that even if someone orders after, their food will be served later (unless priority rules apply) to preserve order.

Q9: Operation: Priority queue. At CHUK hospital, emergencies jump the line. Why is this a priority queue, not a normal queue?

- Because not all items (patients) are treated equally: those with **higher urgency** (higher priority) get served before others, regardless of when they arrived.

- Emergency patients **jump ahead** of non-emergency ones. That breaks normal FIFO order.
- But among non-emergency patients or among equally-urgent, FIFO may apply.

Q10: Operation: Enqueue/Dequeue matching system. In a moto/e-bike taxi app, riders wait for passengers. How would queues fairly match drivers and students (passengers)?

Here's how a queue system could be used:

- **Drivers** (or riders) who are "available" are enqueued in a waiting list (based on time when they sign in or become free).
- A passenger request comes in: the system dequeues the front driver (the one who has waited longest) and assigns the ride.
- Then that driver is removed from queue, works, then after finishing can be enqueued again.
- This ensures fairness: drivers who have waited long get rides first, avoiding bias.
- If there's priority (e.g. rating, distance, or special status), then priority queue concepts might be added.