Stack and Queue Operations: Challenges and Reflections

Stack Operations

Practical Examples

UR Assignment Submission:

python

```
stack = ["Assignment1", "Assignment2", "Assignment3"]
stack.pop()  # Removes Assignment3
stack.pop()  # Removes Assignment2
stack.pop()  # Removes Assignment1
# Result: Empty stack - nothing remains
```

Irembo Form Processing:

python

```
stack = ["Form1", "Form2", "Form3"]
stack.pop()  # Removes Form3
# Remaining: ["Form1", "Form2"]
```

Challenge: Reverse "KIGALI" Using Stack

python

```
def reverse_string(text):
    stack = []
    # Push all characters onto stack
    for char in text:
        stack.append(char)

    # Pop all characters to reverse
    reversed_text = ""
    while stack:
        reversed_text += stack.pop()

    return reversed_text
```

```
original = "KIGALI"
reversed_result = reverse_string(original)
print(f"Original: {original}")
print(f"Reversed: {reversed_result}")
# Output: "ILAGIK"
```

Reflection: Why Stack Models Undo for Exams

Connection to Practicals: Like the UR assignment submission where we pop assignments in reverse order, the undo functionality follows Last-In-First-Out (LIFO) principle.

Key Insights:

- **Sequential Actions:** Each exam action (typing, deleting, formatting) can be pushed onto a stack
- **Reverse Recovery:** Undo pops the most recent action first, mirroring how we remove assignments from the top
- **Error Correction:** Allows students to revert mistakes in the reverse order they were made
- **Memory Efficiency:** Only the action sequence is stored, not multiple versions of the entire document

Queue Operations

Practical Examples

RSSB Applicant Processing:

python

```
from collections import deque

queue = deque(["Applicant1", "Applicant2", "Applicant3", "Applicant4", "Applica
nt5", "Applicant6"])
queue.popleft()  # Serve Applicant1
queue.popleft()  # Serve Applicant2
# Front now: Applicant3
```

Airtel Client Service:

```python
queue = deque(["Client1", "Client2", "Client3", "Client4", "Client5", "Client6", "Client7"])
served = [queue.popleft(), queue.popleft(), queue.popleft()]
# Third served: Client3
```

Challenge: Queue vs Stack for MoMo Transactions

Queue Approach (FIFO):

- Transactions processed in arrival order
- First transaction submitted is first processed
- Fair for all users regardless of transaction size

Stack Approach (LIFO):

- Latest transactions processed first
- Could cause starvation for early transactions
- Unfair for users who submitted first

Recommendation: Queue is fairer because:

- Prevents transaction starvation
- Maintains chronological fairness
- Reduces disputes about processing order

Reflection: Why FIFO Reduces Disputes in Services

Connection to Practicals: As demonstrated in RSSB and Airtel examples, FIFO ensures predictable, fair processing.

Key Benefits:

1. **Transparent Ordering:** Clear "first come, first served" principle
2. **Predictable Waiting:** Users can estimate service time based on queue position
3. **Equal Treatment:** No preference given to later arrivals
4. **Reduced Conflicts:** Eliminates arguments about who should be served next
5. **Social Fairness:** Aligns with natural expectations of fairness in public services

Comparative Analysis

Aspect	Stack (LIFO)	Queue (FIFO)
Fairness	Favors recent entries	Favors early entries
Use Cases	Undo functions, recursion	Service systems, processing
Conflict Potential	High (can seem unfair)	Low (naturally fair)
Practical Example	UR assignment submission	RSSB applicant processing

Conclusion

The choice between stack and queue structures depends on the specific requirements of fairness, order processing, and user expectations. While stacks excel in scenarios requiring reverse-order operations like undo functionality, queues provide the fairness and predictability essential for customer service environments.