Answer Sheet: Data Structures Worksheet - Array-Based Stack & Circular Queue

Part 1: Array-Based Stack

Conceptual Questions

1. **LIFO (Last-In-First-Out)** means that the last element added to the stack is the first one to be removed. This is useful in scenarios like function calls, undo operations, and backtracking algorithms.

2. **Stack Operations:**

   - `push(value)` - Adds an element to the top of the stack. Complexity: $O(1)$.
   - `pop()` - Removes the top element from the stack. Complexity: $O(1)$.
   - `peek()` - Returns the top element without removing it. Complexity: $O(1)$.

3. If a stack is full, attempting to push another element results in an **overflow**. To handle this, we can either dynamically resize the array (if possible) or return an error message.

4. **Stack State After Operations:**

Operation	Stack State (Top $\rightarrow$ Bottom)
push(5)	[5]
push(10)	[10, 5]
push(15)	[15, 10, 5]
pop()	[10, 5]
push(20)	[20, 10, 5]
push(25)	[25, 20, 10, 5]
push(30)	Stack Overflow (No space left)

5. Stacks manage function calls by storing return addresses and local variables in a call stack. When a function is called, its details are pushed onto the stack. When it returns, the details are popped.

6. **Postfix Expression Evaluation:** Using a stack:

   - Push 5
   - Push 3
   - Encounter '+': Pop 3 and 5, compute $5 + 3 = 8$, push 8
   - Push 8

- Encounter ‘*‘: Pop 8 and 8, compute $8 \times 8 = 64$, push 64
- Push 2
- Encounter ‘-‘: Pop 2 and 64, compute $64 - 2 = 62$, push 62

Final result: **62**.

Part 2: Array-Based Circular Queue

Conceptual Questions

1. A circular queue uses a fixed-size array where the rear pointer wraps around to the beginning when it reaches the array's end, unlike a standard queue which shifts elements.

2. Modular arithmetic is used to calculate the next available index efficiently: new position = (current position + 1) mod capacity.

3. A circular queue is full when (rear + 1) mod size = front. An empty queue has front and rear both set to -1 or equal.

4. **Circular Queue State:**

Operation	Queue State (Front → Rear)	Front Index	Rear Index
enqueue(1)	[1]	0	0
enqueue(2)	[1, 2]	0	1
enqueue(3)	[1, 2, 3]	0	2
enqueue(4)	[1, 2, 3, 4]	0	3
enqueue(5)	[1, 2, 3, 4, 5]	0	4
dequeue()	[,2, 3, 4, 5]	1	4
dequeue()	[,,3, 4, 5]	2	4
enqueue(6)	[6, ,3, 4, 5]	2	0
enqueue(7)	[6, 7, 3, 4, 5]	2	1
dequeue()	[6, 7, ,4, 5]	3	1
enqueue(8)	[6, 7, 8, 4, 5]	3	2
enqueue(9)	Queue Overflow	-	-

5. **Queue State for Size 6:**

   - enqueue(10), enqueue(20), enqueue(30) → [10, 20, 30, ,,]$(front = 0, rear = 2) dequeue()$ß[,20, 30,, ,]$(front = 1, rear = 2)$

- enqueue(40), enqueue(50), enqueue(60), enqueue(70) → [,20, 30, 40, 50, 60]$(front = 1, rear = 5) dequeue(), dequeue()$ß[,,30, 40, 50, 60]$(front = 3, rear = 5)$

  - enqueue(80), enqueue(90) → [90, ,30, 40, 50, 60]$(front = 3, rear = 0) enqueue(100) resultsinqueue$

6. Circular queues are used in scheduling algorithms (e.g., round-robin CPU scheduling) where processes are assigned in a cyclic order.