EC4070: DATA STRUCTURES & ALGORITHMS

PRELAB 2

NAME : RENUJAN J.

REG NO : 2022/E/065

1. Define a stack and a queue. Explain how they differ in terms of their data access policies.

Stack

- LIFO (Last-In-First-Out): Elements are added and removed from the same end, known as the "top" of the stack.
- Operations:
    - Push: Adds an element to the top of the stack.
    - Pop: Removes and returns the top element from the stack.
    - Peek: Returns the top element without removing it.
- Real-world analogy: A stack of plates where you add and remove plates from the top.
- Use cases:
    - Function call management
    - Expression evaluation
    - Backtracking algorithms

Stacks are like a pile of books: We can only access the topmost book.

Queue

- FIFO (First-In-First-Out): Elements are added at one end (rear) and removed from the other end (front).
- Operations:
    - Enqueue: Adds an element to the rear of the queue.
    - Dequeue: Removes and returns the front element from the queue.
    - Peek: Returns the front element without removing it.
- Real-world analogy: A queue of people waiting for a service.
- Use cases:
    - Task scheduling
    - Breadth-first search algorithms
    - Print queues

Queues are like a line of people: The first person in line is the first to be served.

2. What is a linked list, and how does it differ from an array in terms of memory usage and performance for insertion and deletion?

Linked List

A linked list is a linear data structure where elements are not stored in contiguous memory locations. Instead, each element, called a node, contains two parts:

i. Data: The actual value stored in the node.
ii. Pointer: A reference to the next node in the sequence.

This structure allows for dynamic allocation of memory, meaning we don't need to specify the size of the list beforehand.

Differences from Arrays

- Memory Usage:
    - Linked Lists: More flexible in memory usage as they allocate memory for each node individually.
    - Arrays: Less flexible, as they require a fixed amount of contiguous memory allocation.
- Insertion and Deletion:

    - Linked Lists: More efficient for insertion and deletion, especially at the beginning or middle, as only pointers need to be updated.
    - Arrays: Less efficient for insertion and deletion, especially in the middle, as elements need to be shifted to accommodate the new element or fill the gap left by the deleted element.
- Random Access:
    - Linked Lists: Less efficient for random access, as you need to traverse the list from the beginning to reach a specific element.
    - Arrays: More efficient for random access, as you can directly access any element using its index.

3. **Explain how to detect an overflow and underflow condition in a queue implemented using an array**.

Overflow:

- Condition: Occurs when the queue is full and there's no more space to add new elements.
- Detection:
    - Simple Implementation:
        - Maintain two pointers: front and rear.
        - If rear reaches the end of the array, the queue is full.
    - Circular Queue Implementation:
        - Use modulo operation to wrap around the array.
        - Overflow occurs when: (rear + 1) % capacity == front

Underflow:

- Condition: Occurs when the queue is empty and there are no elements to remove.
- Detection:
    - Simple Implementation:
        - If front and rear are both -1, the queue is empty.
    - Circular Queue Implementation:
        - Underflow occurs when front == rear.

4. **Implement a menu-driven program to simulate a stack, where the user can:**
    - **Push an element.**
    - **Pop an element.**
    - **Display all elements.**

5. **Implement a queue using two stacks and explain the logic behind enqueue and dequeue operations**