**Subject Name: Data Structures and Algorithms (DSA)**

**1. Runtime Memory Organization**

This refers to how memory is organized and managed during the execution of a program.

**🔹 Key Components:**

* **Code/Text Segment**: Contains the compiled program code (machine instructions).
* **Data Segment**: Stores global and static variables.
  + *Initialized Data Segment*: Holds global variables initialized by the programmer.
  + *Uninitialized Data Segment (BSS)*: Holds uninitialized global and static variables.
* **Heap**: Used for dynamic memory allocation during runtime (e.g., malloc in C, new in C++).
* **Stack**: Stores function calls, local variables, and control data.

**📌 Importance:**

* Helps in understanding memory leaks, buffer overflows, and efficient memory usage.
* Critical for writing optimized and bug-free code.

**2. Runtime Stack Layout**

The runtime stack is a key component of memory that manages function calls and execution context.

**🔹 Stack Frame (Activation Record):**

Each function call creates a new stack frame that contains:

* **Return address**: Where to return after function completes.
* **Function parameters**: Passed to the function.
* **Local variables**: Declared inside the function.
* **Saved registers**: For restoring CPU state.

**🔹 Stack Behavior:**

* **LIFO**: Last In, First Out.
* **Push and Pop**: Functions and local variables are *pushed* when a function is called and *popped* when it returns.

**📌 Example:**

If a function foo() calls bar(), the stack will first store foo()'s frame and then bar()'s frame on top.

**3. Queue ADT (Abstract Data Type)**

A **queue** is a linear data structure that follows the **FIFO (First In, First Out)** principle.

**🔹 Operations:**

* **Enqueue(x)**: Insert element x at the rear.
* **Dequeue()**: Remove element from the front.
* **Peek()/Front()**: View the front element.
* **IsEmpty()**: Check if the queue is empty.
* **IsFull()**: Check if the queue is full (in array-based implementation).

**🔹 Types of Queues:**

* **Simple Queue**
* **Circular Queue**
* **Priority Queue**
* **Double-Ended Queue (Deque)**

**📌 Applications:**

* CPU scheduling
* I/O Buffers
* Breadth-First Search (BFS) in graphs
* Print queue management

**4. Implementing Queue ADT Using:**

**a. Linked List**

A dynamic implementation where each node contains:

* **Data**
* **Pointer to next node**

**🔹 Advantages:**

* No size limitation
* Memory efficient for unknown queue sizes

**🔹 Implementation Details:**

* Use two pointers: **front** and **rear**
* Enqueue: Add at rear
* Dequeue: Remove from front

**b. Circular Array**

A static implementation where the end of the array wraps around to the beginning, forming a circle.

**🔹 Key Concept:**

* Use modulo operation to wrap indices:

ini

CopyEdit

rear = (rear + 1) % size

**🔹 Advantages:**

* Utilizes space efficiently (compared to simple array implementation)
* Fixed size

**🔹 Variables Used:**

* front, rear, and size
* count to track number of elements (optional but useful)

**📌 When to Choose:**

* Use **linked list** when queue size is dynamic
* Use **circular array** when queue size is fixed and memory needs to be compact