Linked List vs. Arrays:

Feature	Linked List	Arrays
Size	Dynamic; can grow/shrink as needed	Fixed size; allocated at compile time
Memory Allocation	Non-contiguous memory allocation	Contiguous memory allocation
Insertion/Deletion	Easier and more efficient	Requires shifting elements
Access Time	Sequential access (slower)	Random access (faster)
Memory Wastage	No wastage (dynamically allocated)	Wastage possible due to fixed size

1. Linked List as an ADT:

A **Linked List** is a linear data structure where each element (called a node) contains data and a reference (or link) to the next node in the sequence. Unlike arrays, linked lists don't have a fixed size and can dynamically grow or shrink during program execution.

- **Operations:**
  - **Insert**: Add a new node.
  - **Delete**: Remove an existing node.
  - **Traverse**: Visit each node to access or print its data.
  - **isEmpty**: Checks whether the linked list is empty
  - **isFu**ll: hecks whether the linked list has run out of memory.

Dynamic Memory Allocation

Dynamic memory allocation refers to the process of allocating memory during runtime instead of compile-time, allowing more flexibility in the management of memory. This process is essential when we don't know in advance how much memory is required, which is a common scenario in data structures like **linked lists**.

Key Concepts:

- **Memory Allocation:** Memory is allocated at runtime, and the program can request memory for a variable or object as needed.
- **Memory Deallocation:** After the memory is no longer needed, it is deallocated (freed) to prevent memory leaks and optimize resource use.

Functions in C/C++:

1. **malloc(size_t size):**
   Allocates a block of memory of the specified size and returns a pointer to the first byte of that block.

   int* ptr = (int*)malloc(sizeof(int)); // Allocates memory for one integer.

2. **calloc(size_t num, size_t size):**
   Allocates memory for an array of num elements of size bytes each, and initializes all bytes to 0.

   int* arr = (int*)calloc(5, sizeof(int)); // Allocates memory for 5 integers.

3. *realloc(void ptr, size_t size):**
   Resizes previously allocated memory block. If the new size is larger, it may also move the memory to a new location.

   ptr = (int*)realloc(ptr, 10 * sizeof(int)); // Resizes the memory block for 10 integers.

4. *free(void ptr):**
   Frees previously allocated memory, making it available for reuse.

   free(ptr); // Frees the allocated memory.

Benefits:

- Allows the creation of dynamic structures like linked lists, where the size can change during execution.
- Efficient use of memory by allocating space only when needed.

Types

INKE DIAGRAM MUST HE <compulsory>

1. Singly Linked List (SLL):

- **Definition:**
  - A **Singly Linked List** consists of nodes where each node has two parts:
    - **Data**: Stores the value.
    - **Next pointer**: A reference or pointer to the next node in the list.
- **Structure:**
  - The list is linear, and each node points to the next node.
  - The last node's next pointer is NULL, indicating the end of the list.
- **Advantages:**
  - Dynamic memory allocation.
  - Can grow and shrink in size as needed.
- **Disadvantages:**
  - Accessing an element requires traversing from the head node.
  - Searching for a specific node can take linear time.

---

2. Doubly Linked List (DLL):

- **Definition:**
  - A **Doubly Linked List** is an extension of the singly linked list where each node has three parts:
    - **Data**: Stores the value.
    - **Next pointer**: Points to the next node in the list.
    - **Previous pointer**: Points to the previous node in the list.
- **Structure:**
  - Each node has two pointers (next and previous), allowing traversal in both directions.
  - The first node's previous pointer and the last node's next pointer are NULL.
- **Advantages:**

    - Easier to traverse in both directions.
    - More efficient insertion and deletion from both ends.
- **Disadvantages:**
    - Requires more memory for the previous pointer.
    - More complex to implement than a singly linked list.

---

3. Circular Linked List (Singly Circular Linked List):

- **Definition:**
    - In a **Singly Circular Linked List**, the last node's next pointer points back to the first node, making the list circular.
    - Unlike a singly linked list, there is no NULL to mark the end of the list; the list loops back to the first node.
- **Structure:**
    - There is no NULL value at the end of the list, so traversal can continue indefinitely from the last node back to the first node.
    - The first node is the head, and the last node points to the head.
- **Advantages:**
    - Efficient for circular queues and round-robin scheduling.
    - Continuous traversal without checking for NULL.
- **Disadvantages:**
    - Requires careful handling to avoid infinite loops during traversal.
    - More complex than a singly linked list.

---

4. Doubly Circular Linked List:

- **Definition:**
    - A **Doubly Circular Linked List** combines the features of both a doubly linked list and a circular linked list.
    - The last node's next pointer points to the first node, and the first node's previous pointer points to the last node.
- **Structure:**
    - Each node has three parts: **data**, **next**, and **previous**.
    - The first node's previous pointer points to the last node, and the last node's next pointer points to the first node, making the list circular.
- **Advantages:**
    - Allows for bidirectional traversal and circular traversal.

    - More flexible for operations that require circular navigation.
- **Disadvantages:**
    - Requires more memory due to two pointers (next and previous) in each node.
    - More complex to implement than singly circular linked lists.

To learn In Short Types of Linked Lists:

1. **Singly Linked List**: One pointer (next) per node, linear structure.
2. **Doubly Linked List**: Two pointers (next, previous) per node, allows bidirectional traversal.
3. **Singly Circular Linked List**: Circular structure where the last node points to the first.
4. **Doubly Circular Linked List**: Circular structure with two pointers per node, allowing bidirectional and circular traversal.