Linked Lists: The Dynamic Data Structure

Department – Computer Science And Engineering

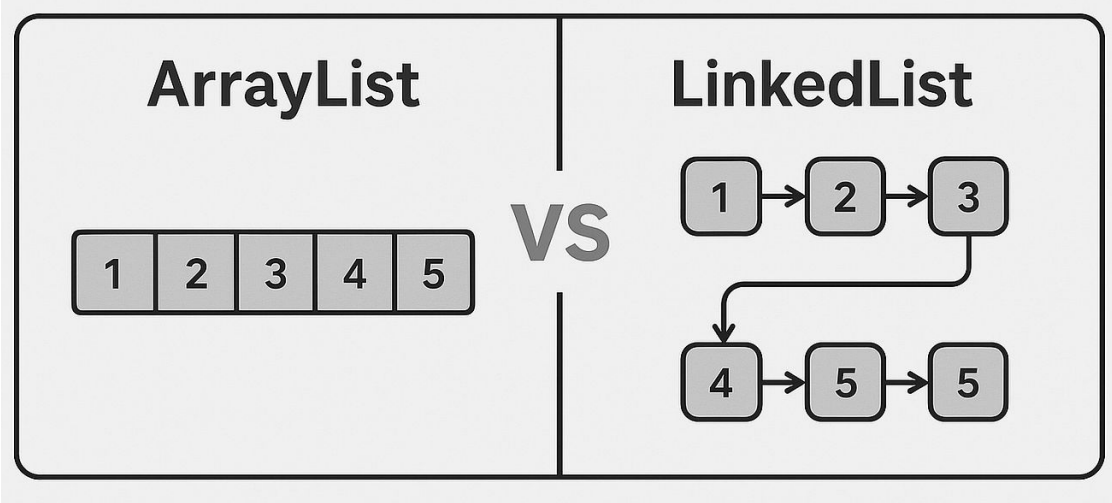
Date of presentation – 05/08/2025

Team – 1) Mohd. Adnan Malik (2022a1r092)
2) Tanvir Singh (2022a1r077)
3) Akshit Thapa (2022a1r078)
4) Vishwa Bandhu (2022a1r098)
3) Aman Manhas (2022a1r086)

WHAT IS A LINKED LIST

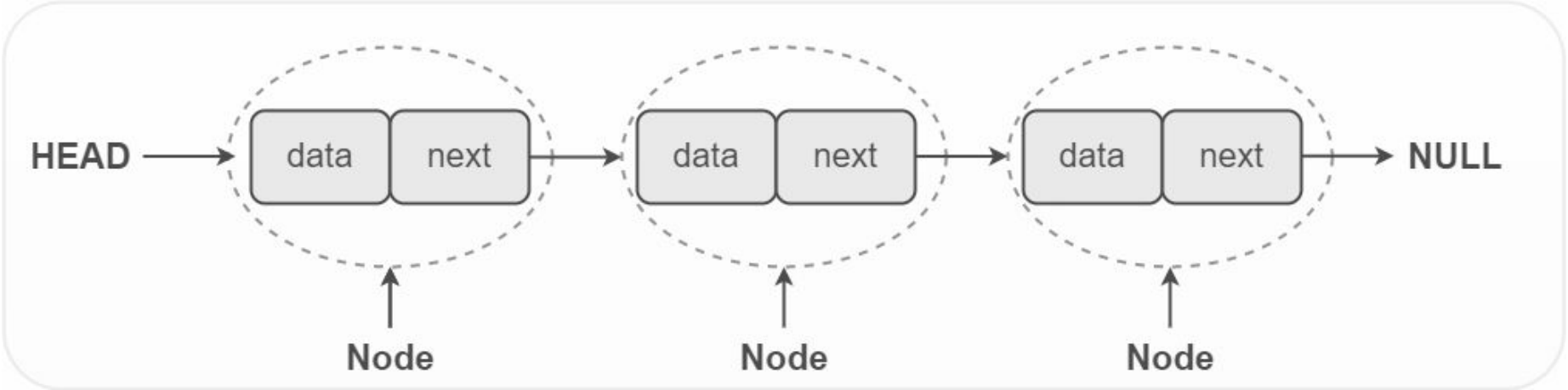
- A Linear Data Structure where elements are stored at non contiguous memory locations.
- Elements are connected using pointers or links.
- How is it different from Arrays:

Arrays	**Linked Lists**
Fixed Size	Dynamic Size
Contiguous Memory	Non Contiguous Memory



Core Components of a Linked List

- Node: The fundamental block of a linked list.
  - Data: The value is stored in the node.
  - Pointer: A reference to the next node in the list.
- Head: A pointer to the first node in the linked list.
- Tail: A pointer to the last node in the linked list.



Types of Linked Lists

Properties	Singly Linked List	Doubly Linked List	Circular Linked List	Circular Doubly Linked List
Traversal	Unidirectional	Bi-Directional	Unidirectional but looping	Bidirectional and Looping
Reference	Reference to next node	Reference to both next and previous node	Reference to next node & last node references the first node	Reference to both next and previous node & last node references the first node

Implementation in C++

SINGLY LINKED LIST

CODE:

```
struct Node{
    int data;
    Node* next;
}
```

1. A structure or a class may be used for a blueprint for the linked list
2. The struct is a self referential structure, i.e. It's member includes a reference to itself.
3. Only one member is a pointer that points to the next node.

DOUBLY LINKED LIST

CODE:

```
struct Node{
    Node* prev;
    int data;
    Node* next;
}
```

1. A structure or a class may be used for a blueprint for the linked list
2. The struct also uses self referential structure.
3. Two of the members are pointers that point to next and previous node.

Time & Space Complexity Analysis

Operation	Linked List	Array
Access By Index	O(n)	O(1)
Search	O(n)	O(n) unsorted, O(logn) sorted
Insertion / Deletion	O(n) at any position	O(n)
Space Complexity	O(n)	O(n)

- Accessing an element in a linked list requires traversing from the head.
- Insertion at the head is O(1) because we only change a few pointers.
- Insertion in an array requires shifting all subsequent elements, which is a slow O(n) operation.

Memory Management in C++

- Linked lists use **dynamic memory allocation**. This means you must explicitly manage memory yourself.
- **new:** Used to allocate memory for a new node on the heap.
- **delete:** Used to free the memory of a node when it is no longer needed.
- **The Danger: Memory Leaks:** If you create a node with new but forget to delete it, the memory remains allocated but is inaccessible. This is a **memory leak**.

Solution: Destructors

- The C++ Linked List class should have a destructor (~LinkedList()).
- The destructor is automatically called when a LinkedList object goes out of scope.
- **Its job is to delete every single node in the list** to free up all dynamically allocated memory.

Real World Application

- **Music Playlists:** Songs in a playlist can be nodes, with next and previous pointers for skipping tracks.
- **Image Viewers:** Browsing through photos uses a linked list structure to move between images.
- **Web Browsers:** The history functionality can be implemented as a doubly linked list to allow for "back" and "forward" navigation.
- **Undo/Redo Functionality:** A sequence of actions can be stored as a doubly linked list.

Summary

- Linked lists are a powerful, dynamic data structure.
- They excel at insertions and deletions, unlike static arrays.
- In C++, implementing them requires careful **dynamic memory management** using **new** and **delete**.
- A well-designed **destructor** is essential to prevent memory leaks.


Thank You