Part 1 - Understanding Linked List

Definition of Linked List

A Linked List is a linear data structure consisting of elements called nodes. Each node has two main parts:

- **Data** – stores the value of the element.
- **Pointer/Reference** – points to the next node (or the previous node in a Doubly Linked List).

Differences Between Linked List and Array

Aspect	Array	Linked List
Storage	Elements are stored contiguously in memory.	Nodes are stored in non-contiguous memory and linked by pointers.
Element Access	Direct access using an index, complexity O(1).	Access requires traversal from the start, complexity O(n).
Size	Static, size determined at declaration.	Dynamic, can grow or shrink as needed.
Insert/Delete	Expensive due to element shifting, complexity O(n).	More efficient, only pointer adjustments needed, complexity O(1) if position is known.

Benefits of Linked List

Dynamic Size

- No need to define size in advance like an array.
- Can grow or shrink according to program needs.

Efficiency in Insert/Delete Operations

- Adding or removing elements does not require shifting data.
- Complexity can be O(1) if the pointer is known.

More Flexible Memory Usage

- Nodes can be stored in scattered memory locations.
- Useful in systems with fragmented memory.

Support for Complex Data Structures

- Forms the basis for other data structures such as Stack, Queue, Deque, and Graph.
- Enables creation of variants like Circular Linked List and Doubly Linked List.

Two-Way Traversal (in Doubly Linked List)

- Makes navigation forward and backward easier.

Applications of Linked Lists

Dynamic Memory Allocation

- Used to keep track of free and allocated memory blocks.

Undo/Redo Operations in Text Editors

- Stores changes and navigates through editing history.

Graph Representation

- Efficiently implements adjacency lists in graph structures.

Implementation of Other Data Structures

- Serves as a base for stacks, queues, and deques.

Advantages of Linked List

- Efficient insertion and deletion since shifting of elements is not required as in arrays.
- Dynamic size allows growth or shrinkage at runtime.
- Memory utilization is more efficient; no pre-allocation reduces wasted space.
- Suitable for applications with large or frequently changing datasets.
- Uses non-contiguous memory blocks, useful in memory-limited systems.

Disadvantages of Linked List

- Direct access to an element is not possible; traversal from the start is required.
- Each node requires extra memory to store a pointer.
- More complex to implement and manage compared to arrays.
- Pointer mismanagement can cause bugs, memory leaks, or segmentation faults.

Short Code Example

```
#include <iostream>
using namespace std;
```

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

void pushFront(Node*& head, int value) {
    Node* newNode = new Node(); // Buat node baru
    newNode->data = value;      // Isi data
    newNode->next = head;       // Hubungkan ke head lama
    head = newNode;             // Jadikan node baru sebagai head
}

void printList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " -> ";
        current = current->next;
    }
    cout << "NULL\n";
}

int main() {
    Node* head = nullptr;  // List kosong

    pushFront(head, 10);
    pushFront(head, 20);
    pushFront(head, 30);

    printList(head); // Output: 30 -> 20 -> 10 -> NULL

    return 0;
}
```

Code Explanation:

- `Node` is the basic structure of a Linked List.
- `pushFront` adds a new node at the beginning of the list.
- `printList` traverses the list and displays all elements.

Revision #8
Created 2025-09-02 08:25:22 UTC by BH
Updated 2025-09-04 00:54:44 UTC by BH