■ Linked List Notes

1. Introduction to Linked Lists

Definition

A Linked List is a linear data structure where data elements (called nodes) are connected using pointers. Each node has two parts: Data and Pointer to the next node.

Characteristics

- Dynamic memory allocation
- Non-contiguous memory allocation
- Efficient insertion and deletion
- Sequential access
- Extra memory for pointers

Linked List vs Array

Array vs Linked List Comparison:
• Array: Contiguous memory, fixed size, direct indexing O(1)
• Linked List: Non-contiguous memory, dynamic size, sequential traversal O(n)

Real-life Examples

• Music playlist
• Train coaches
• Web browsing history

2. Types of Linked Lists

2.1 Singly Linked List

Each node contains data and a pointer to the next node. The last node points to NULL.

```
Diagram:
Head → [10 | next] → [20 | next] → [30 | NULL]
C Example:
#include <stdio.h>
#include <stdlib.h>

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

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10; head->next = second;
    second->data = 20; second->next = third;
    third->data = 30; third->next = NULL;

    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->data);
```

```
        temp = temp->next;
    }
    printf("NULL\n");
    return 0;
}
```

2.2 Doubly Linked List

Each node has three parts: previous pointer, data, and next pointer. Allows traversal in both directions.

```
Diagram:
NULL ← [prev | 10 | next] ↔ [prev | 20 | next] ↔ [prev | 30 | next] → NULL
C Example:
#include <stdio.h>
#include <stdlib.h>

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

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10; head->prev = NULL; head->next = second;
    second->data = 20; second->prev = head; second->next = third;
    third->data = 30; third->prev = second; third->next = NULL;

    struct Node* temp = head;
    printf("Forward Traversal: ");
    while(temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
    return 0;
}
```

2.3 Circular Linked List

In a circular linked list, the last node points back to the first node. Traversal is circular.

```
Diagram:
Head → [10 | next] → [20 | next] → [30 | next]
       ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
C Example:
#include <stdio.h>
#include <stdlib.h>

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

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10; head->next = second;
    second->data = 20; second->next = third;
```

```
    third->data = 30; third->next = head;

    struct Node* temp = head;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while(temp != head);
    printf("(Back to Head)\n");
    return 0;
}
```