Day 22: Singly Linked List

Gati Goyal

"A singly linked list is a powerful tool, one node at a time."
— Anonymous

---

1 Introduction

A **Singly Linked List** is a linear data structure in which elements are stored in nodes. Each node contains two parts:

- **Data:** Stores the actual value.
- **Next:** Points to the next node in the sequence (or NULL if it's the last node).

It is used in scenarios where elements need to be dynamically added or removed. Unlike arrays, the size of a linked list can grow or shrink at runtime, which makes it a highly efficient choice for dynamic memory allocation.

2 Operations on Singly Linked List

The most common operations performed on a singly linked list are:

- **Insertion:** Insert an element at the beginning, end, or at a specific position.
- **Deletion:** Remove an element from the beginning, end, or at a specific position.
- **Traversal:** Visit and print the elements of the list.

3 Code

```
#include <stdio.h>
#include <stdlib.h>

// Define a Node structure
struct Node {
    int data;
    struct Node* next;
};

```

```c
// Function to insert a node at the beginning
struct Node* insertAtBeginning(struct Node* head, int value) {
    // Allocate memory for new node
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
       Node));

    // Assign data and set the next pointer
    newNode->data = value;
    newNode->next = head; // Link the new node to the previous
       first node

    // Return the new head (new node)
    return newNode;
}

// Function to delete a node from the beginning
struct Node* deleteFromBeginning(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return head;
    }

    // Store the current head node
    struct Node* temp = head;

    // Move head to the next node
    head = head->next;

    // Free the memory of the old head node
    free(temp);

    printf("Deleted node from the beginning.\n");

    // Return the updated head
    return head;
}

// Function to print the list
void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

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

```c

int main() {
    struct Node* head = NULL; // Initialize an empty list (head
       points to NULL)
    int choice, value;

    // Ask user to insert a value
    printf("Enter the value to insert at the beginning: ");
    scanf("%d", &value);
    head = insertAtBeginning(head, value);

    // Ask user if they want to insert another value
    printf("Enter the value to insert at the beginning: ");
    scanf("%d", &value);
    head = insertAtBeginning(head, value);

    // Print the list after insertion
    printf("List after insertion: ");
    printList(head);

    // Ask user if they want to delete from the beginning
    printf("Do you want to delete a node from the beginning? (1
       for Yes, 0 for No): ");
    scanf("%d", &choice);
    if (choice == 1) {
        head = deleteFromBeginning(head);
    }

    // Print the list after deletion
    printf("List after deletion: ");
    printList(head);

    return 0;
}
```

4 Singly Linked List Operations output

```
PS C:\Users\gatig\AppData\Local\Temp> cd "C:\Users\gatig\AppData\Local\Temp\" ; if
rFile }
Enter the value to insert at the beginning: 5
Enter the value to insert at the beginning: 3
List after insertion: 3 -> 5 -> NULL
Do you want to delete a node from the beginning? (1 for Yes, 0 for No): 1
Deleted node from the beginning.
List after deletion: 5 -> NULL
PS C:\Users\gatig\AppData\Local\Temp>
```

Figure 1: Output

5 Conclusion

The singly linked list is a versatile and efficient data structure for dynamically managing data. It allows for flexible memory allocation and easy insertion and deletion operations. The operations demonstrated here, such as insertion and deletion at the beginning, are fundamental in many applications, such as implementing queues, stacks, and various algorithmic problems.