# Day 23: Reverse a Linked List

**Gati Goyal**

---

*"In a linked list, reversing brings you back to the beginning with a different perspective."*
— Anonymous

---

## 1 Introduction

A **Singly Linked List** is a linear data structure where each element (node) contains two parts:

- The data element.
- A pointer to the next node in the list.

In this problem, we are asked to **reverse the singly linked list**. This means that the head node will be transformed into the last node, and each node will point to the previous one.

We can reverse a singly linked list using the **three-pointer technique**. This method uses three pointers:

- **Previous Pointer:** Points to the previous node.
- **Current Pointer:** Points to the current node.
- **Next Pointer:** Points to the next node.

## 2 Steps to Reverse a Linked List

The process involves the following steps:

1. Initialize three pointers: `prev` (NULL), `current` (head), and `next` (NULL).
2. Traverse the list, and for each node:
   (a) Set `next` to `current->next`.
   (b) Change `current->next` to `prev`.
   (c) Move `prev` to `current` and `current` to `next`.
3. Repeat this until `current` becomes NULL.
4. The `prev` pointer will be pointing to the new head of the reversed list.

## 3 Applications of Reversing a Linked List

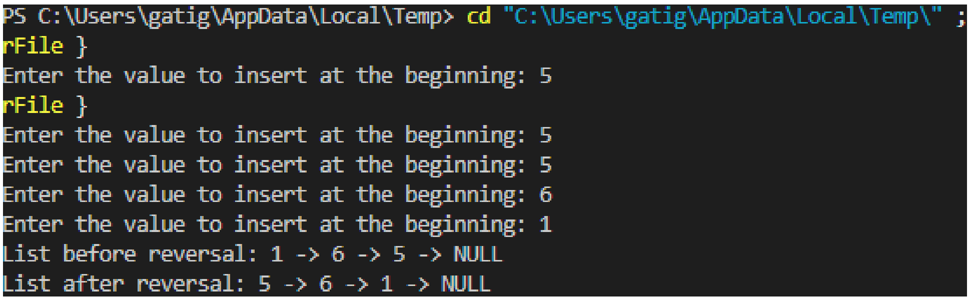
Reversing a linked list can be useful in several scenarios, including:

- Reversing a list of nodes for printing or traversing.
- Implementing undo operations (reversing actions).
- Reversing a stack of elements, where the linked list can serve as an auxiliary data structure.

## 4 Code Implementation

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

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

// 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 reverse the linked list
struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    // Traverse the list and reverse the links
    while (current != NULL) {
        next = current->next;     // Store the next node
        current->next = prev;     // Reverse the current node's pointer
        prev = current;           // Move prev and current one step forward
        current = next;
    }

    // The new head is the previous node at the end of the list
    return prev;
}

// 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");
}

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

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

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

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

    // Print the list before reversal
    printf("List before reversal: ");
    printList(head);

    // Reverse the list
    head = reverseList(head);

    // Print the list after reversal
    printf("List after reversal: ");
    printList(head);
    return 0;
}
```

# 5 Reversal of Linked List



```
PS C:\Users\gatig\AppData\Local\Temp> cd "C:\Users\gatig\AppData\Local\Temp\" ;
rFile }
Enter the value to insert at the beginning: 5
rFile }
Enter the value to insert at the beginning: 5
Enter the value to insert at the beginning: 5
Enter the value to insert at the beginning: 6
Enter the value to insert at the beginning: 1
List before reversal: 1 -> 6 -> 5 -> NULL
List after reversal: 5 -> 6 -> 1 -> NULL
```

Figure 1: Linked List's Reversal

# 6 Conclusion

Reversing a singly linked list is a fundamental operation that can be useful in various algorithms, such as undo operations and traversals. The three-pointer technique is an efficient way to reverse a list in place, without requiring additional space, making the operation both time and space-efficient.