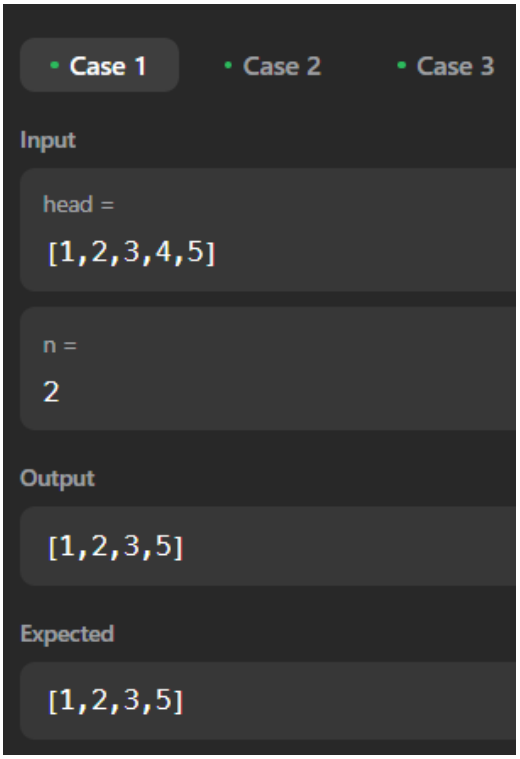
Week 4: Singly Linked List

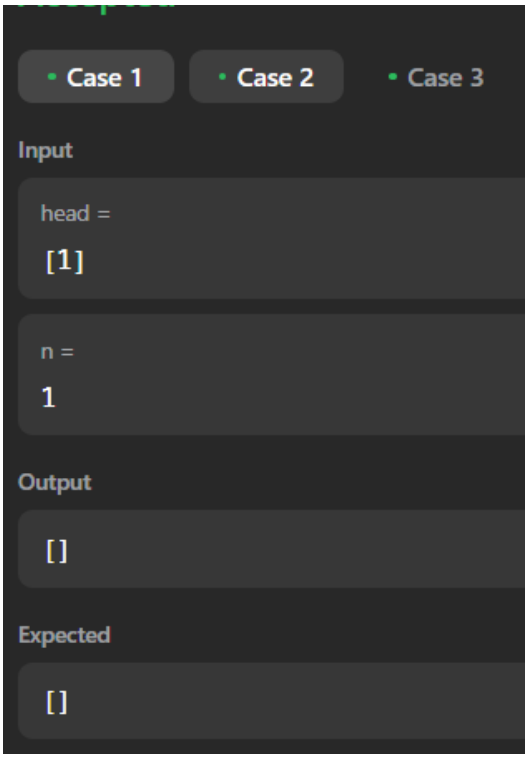
Aim: Given the head of a linked list, remove the nth node from the end of the list and return its head.

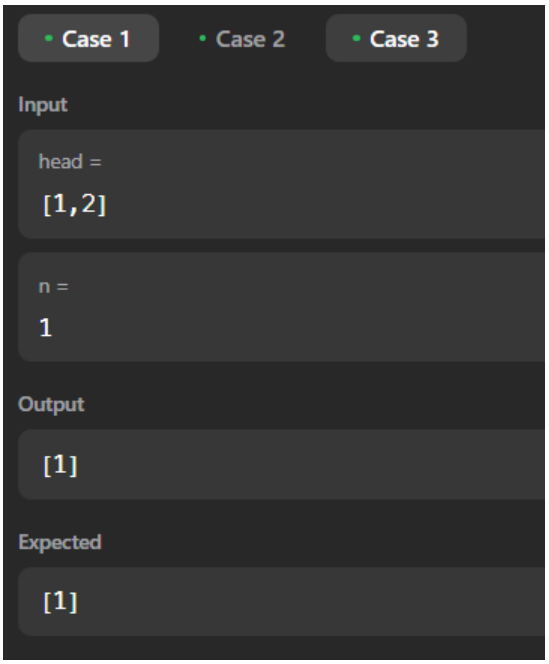
Program:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
   ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* temp  = head;
        int cnt = 0;
        while(temp)
        {
            cnt++;
            temp = temp ->next;
        }
        n = cnt -n-1;
        if(n == -1)
        {
            head = head ->next;
            return head;
        }
        temp = head;
        for(int i =0;i<n;i++)
        {
            temp = temp ->next;
        }
        // cout<<temp->val;
        if(temp ->next)
            temp ->next = temp->next->next;
        return head;
    }
};
```

Input & Output:







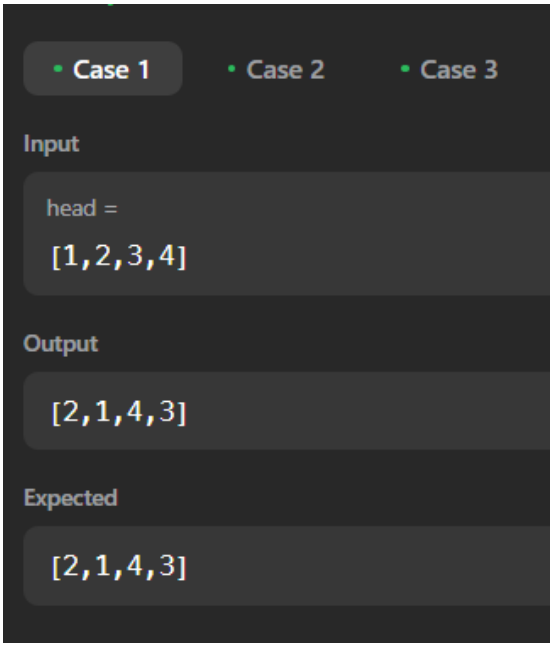
Conclusion: From the above program I have learned to traverse the linked list and delete the node of the linkedlist

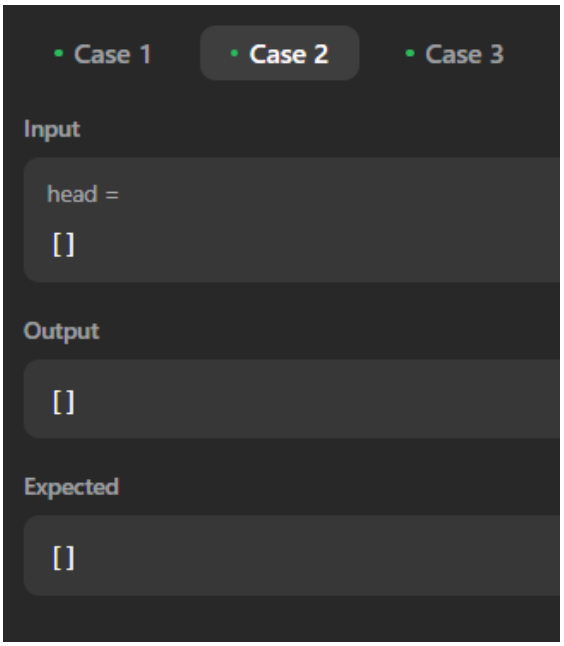
Aim: Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

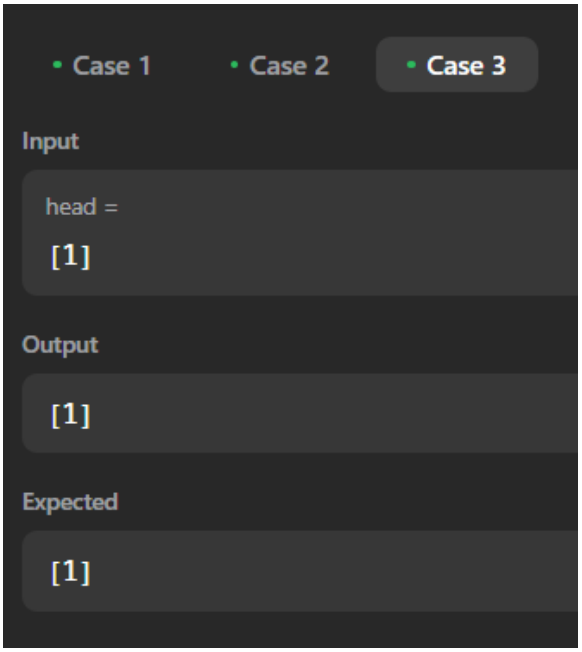
Program:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if(head == NULL)
            return  NULL;
        if(head ->next == NULL)
            return head;
        ListNode *cur = head;
        ListNode *forw = head ->next;
        ListNode *nf = forw ->next;
        forw->next = cur;
        cur ->next  = swapPairs(nf);
        return forw;
    }
};
```

Input & Output:







Conclusion: From the above program I have learned to traverse the linked list using recursion and swap the node of the linked list

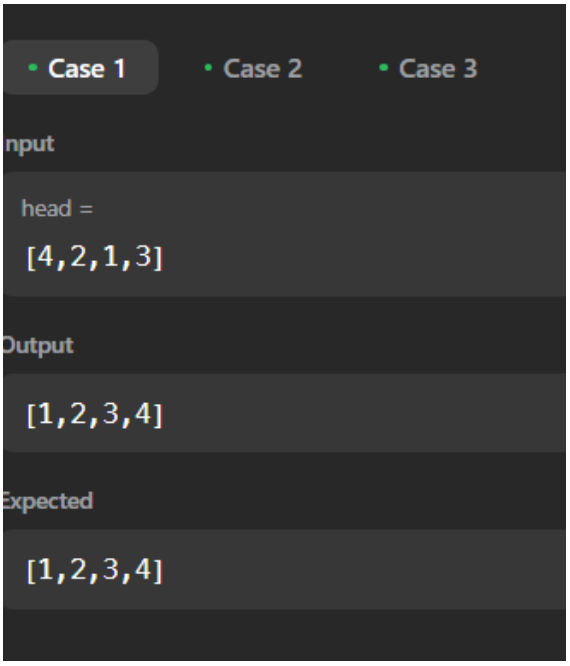
Aim: Given the head of a linked list, return the list after sorting it in ascending order.

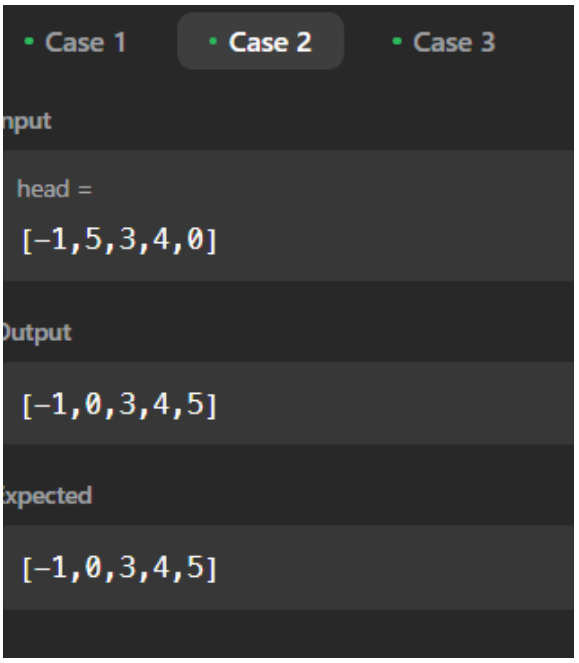
Program:

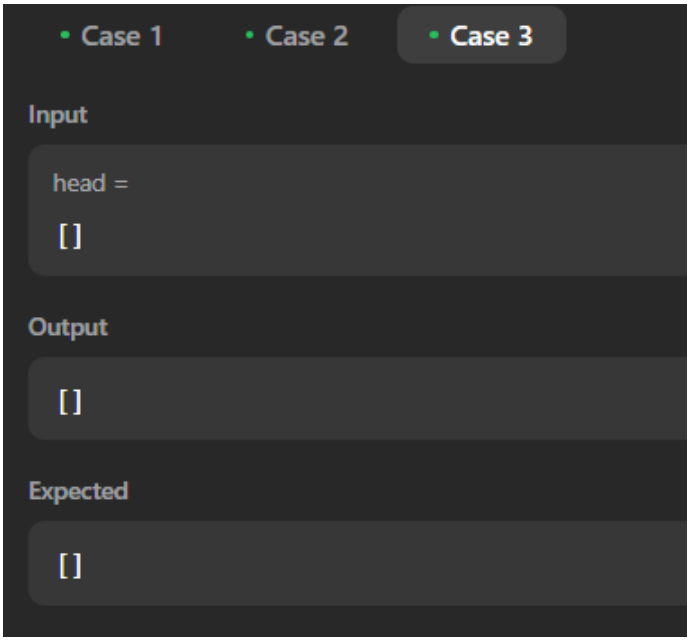
```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode * getMid(ListNode* head)
    {
        ListNode*noob = head;
        ListNode *pro = head->next;
        while(pro != NULL && pro->next != NULL)
        {
            pro = pro ->next->next;
            noob = noob -> next;
        }
        return noob;
    }
   ListNode* merge(ListNode* list1, ListNode* list2) {
        ListNode* sorted_head = new ListNode(-1);
        ListNode* sorted_tail = sorted_head;

        ListNode *cur1 = list1;
        ListNode *cur2 = list2;

        while(cur1 != NULL && cur2 != NULL)
        {
            if(cur1 -> val <= cur2 ->val)
            {
                sorted_tail -> next = cur1;
                sorted_tail = cur1;
                cur1 = cur1 -> next;
            }
            else
            {
                sorted_tail -> next = cur2;
                sorted_tail = cur2;
                cur2 = cur2 -> next;
```

```
            }
        }
        while(cur1 != NULL)
        {
            sorted_tail -> next = cur1;
                sorted_tail = cur1;
                cur1 = cur1 -> next;
        }
        while(cur2 != NULL)
        {
            sorted_tail -> next = cur2;
                sorted_tail = cur2;
                cur2 = cur2 -> next;
        }
        return sorted_head ->next;
    }
    ListNode* sortList(ListNode* head) {
        if(head == NULL || head ->next == NULL)
            return head;
        ListNode *mid = getMid(head);
        ListNode *left = head;
        ListNode *right = mid->next;
        mid ->next = NULL;
        left = sortList(left);
        right =sortList(right);

        ListNode* result = merge(left,right);
        return result;
    }
};
```

Input & Output:







Conclusion: From the above program I have learned to sort the Linked List using merge sort.

Aim: Delete front and end node of a singly linked list.

Input Format

- Input lines contain choices for insertion at end, deletion from front and end. For example, 1. insert 2. delete front 3. delete last, 4.display and 0 for exit.
- For insertion functions, value must be given along with choice 1.
- 0 must be given at end of input.

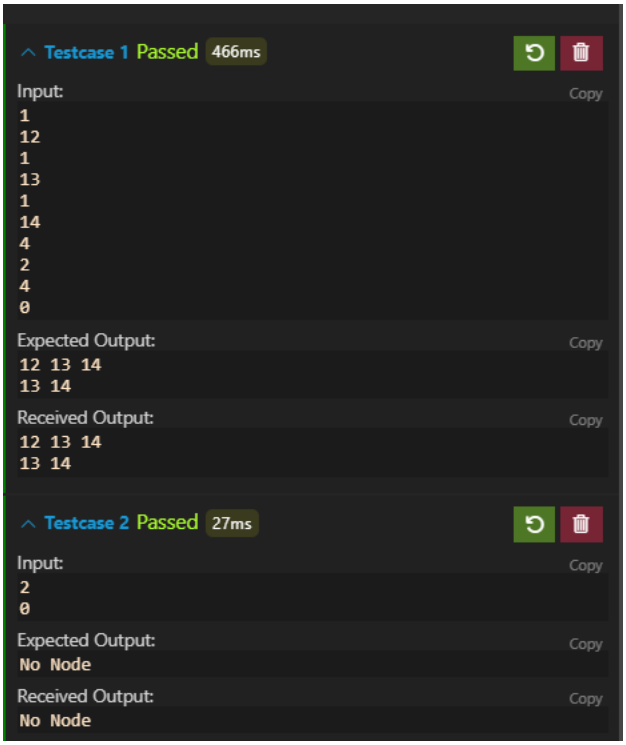
Program:

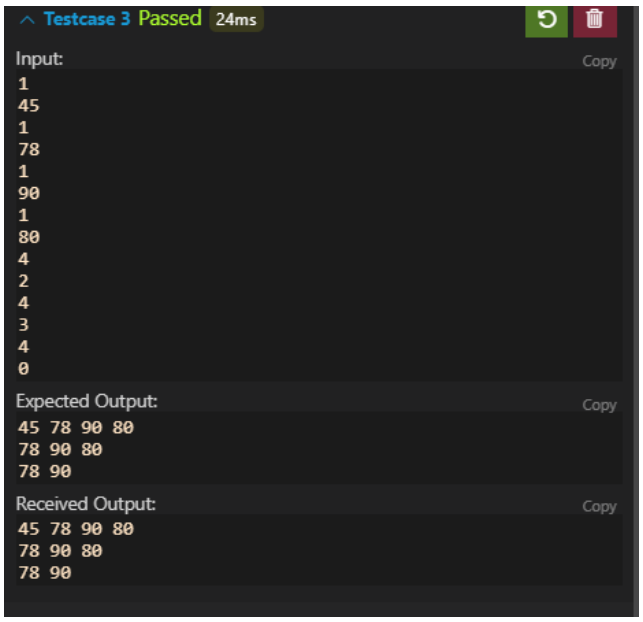
```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node()
    {
        data = 0;
        next = NULL;
    }
    Node(int d)
    {
        data = d;
        next = NULL;
    }
};
void insert_end(Node *&head, Node *&tail, int data)
{
    if (head == NULL)
    {
        Node *temp = new Node(data);
        head = temp;
        tail = temp;
        return;
    }
    Node *temp = new Node(data);
```

```
    tail->next = temp;
    tail = temp;
}
void delete_front(Node *&head)
{
    if(head == NULL)
    {
        cout<<"No Node";
        return;
    }
    head = head->next;
}
void delete_end(Node *&head, Node *&tail)
{
    if(head == NULL)
    {
        cout<<"No Node";
        return;
    }
    Node *cur = head;
    while (cur->next != tail)
    {
        cur = cur->next;
    }
    cur->next = tail->next;
    cur = tail;
}
void print(Node *head)
{
    if(head == NULL)
    {
        cout<<"No Node";
        return;
    }
    Node *cur = head;

    while (cur != NULL)
    {
        cout << cur->data << " ";
        cur = cur->next;
    }
    cout << endl;
}
int main()
{
```

```
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    Node *head = NULL;
    Node *tail = head;
    int t;
    cin >> t;
    while (t != 0)
    {
        if (t == 1)
        {
            int n;
            cin >> n;

            insert_end(head, tail, n);
        }
        else if (t == 2)
        {
            delete_front(head);
        }
        else if (t == 3)
        {

            delete_end(head, tail);
        }
        else if (t == 4)
        {
            print(head);
        }
        else if (t == 0)
            break;
        cin >> t;
    }
    return 0;
}
```

Input & Output:





Conclusion: From the above program I have learned to create the Linked List from scratch and to implement basic function of Linked List