LINKED LIST

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*next;

    // constructor

    Node(int val)

    {

        data = val;

        next = NULL;

    }

};

// FUNCTIONS FOR INSERT

void insertAtTail(Node \*&head, int val)

{

    Node \*n = new Node(val);

    if (head == NULL)

    {

        head = n;

        return;

    }

    Node \*temp = head;

    while (temp->next != NULL)

    {

        temp = temp->next;

    }

    temp->next = n;

}

void insertAtHead(Node \*&head, int val)

{

    Node \*n = new Node(val);

    n->next = head;

    head = n;

}

// FUNCTION TO DISPLAY

void display(Node \*head)

{

    Node \*temp = head;

    while (temp != NULL)

    {                               // temp pointing to the NULL variable and it should not be temp->next!=NuLL

        cout << temp->data << "->"; // after reaching the last node temp->next==Null and does not enter the loop

        temp = temp->next;          // and the last node does not prints

    }

    cout << "NULL" << endl;

}

// FUNCTION TO SEARCH

int search(Node \*head, int key)

{

    Node \*temp = head;

    while (temp != NULL)

    {

        if (temp->data == key)

        {

            return key;

        }

        temp = temp->next;

    }

    return -1;

}

// to delete first node we cannot have n-1th node

void deleteAtHead(Node \*&head)

{

    Node \*todelete = head;

    head = head->next;

    delete todelete;

}

// FUNCTION TO DELETE

void deleteNode(Node \*&head, int val)

{

    // cornered case1

    // where no node is present in the linkedlist

    if (head == NULL)

    {

        return; // cannot do anything

    }

    // cornered case2

    // where only 1 node is present we cannot access n+1th node

    if (head->next == NULL)

    {

        deleteAtHead(head);

        return;

    }

    Node \*temp = head;

    while (temp->next->data != val)

    { // we are on the n-1th node

        temp = temp->next;

    }

    Node \*todelete = temp->next;   // nth node to delete

    temp->next = temp->next->next; // make link between n-1th node to point to n+1th node

    delete todelete;

}

// reverse a linkedlist

Node \*reverse(Node \*head)

{ // will return a node address and we only have one pointer i.e head

    // initialize 3 pointers

    Node \*prev = NULL;

    Node \*curr = head;

    Node \*next;

    while (curr != NULL)

    {

        // initialize next with pointing to current ka next

        next = curr->next;

        // change the link to reverse

        // curr should point to previous

        // we still do not miss the rest of the linkedlist as next points to the list

        curr->next = prev;

        // make all the 3 pointer to move ahead

        prev = curr;

        curr = next;

        // do not increment next pointer as in the first line of the loop it is incremented

    }

    return prev; // new head

}

// reverse a linkedlist using recursion

Node \*reverseRecursive(Node \*&head)

{

    // base case

    if (head == NULL || head->next == NULL)

    {

        return head;

    }

    // keep the head and change from head ka next it will give us the rest of the list in reverse

    Node \*newhead1 = reverseRecursive(head->next); // 1->2<-3<-4

    head->next->next = head;                       // 1<-2<-3<-4 point 2 to 1

    head->next = NULL;                             // this becomes the last node

    return newhead1;

}

int main()

{

    Node \*head = NULL;

    insertAtTail(head, 3);

    insertAtTail(head, 36);

    insertAtTail(head, 2);

    insertAtTail(head, 10);

    insertAtTail(head, 9);

    // display(head);

    insertAtHead(head, 111);

    insertAtHead(head, 777);

    insertAtHead(head, 26);

    cout << "New LinkedList" << endl;

    display(head);

    // cout<<"Element: "<<search(head,36)<<endl;

    // deleteNode(head,111);

    // cout<<"After delete operation"<<endl;

    // cout<<"--------------------------"<<endl;

    // deleteAtHead(head);

    // display(head);

    Node \*newHead = reverseRecursive(head);

    cout << "Reverse LinkedList using recursion" << endl;

    display(newHead);

    // Node\* newHead2=reverseRecursive(head);

    //   cout<<"Reverse LinkedList using recursion"<<endl;

    // display(newHead2);

    return 0;

}

output

New LinkedList

26->777->111->3->36->2->10->9->NULL

Reverse LinkedList using recursion

9->10->2->36->3->111->777->26->NULL