Q1. Write a program to check for palindrome using linked list

**CODE**

#include<iostream>

using namespace std;

class node{

public:

node\* next;

int data;

//constructor

node(int val)

{

    data=val;

    next=NULL;

}

};

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

//we are taking value of head by address because we want to modify our linked list

{

    node\* n = new node(val);

    //incase we are adding the first node to the linked list just use the below if statement

    if(head==NULL)

    {

        head=n;

        return;

    }

    node\* temp = head;

    while(temp->next!=NULL)

    {

        temp=temp->next;

    }

    //when temp reaches null, insert the node at the tail

    temp->next=n;

}

void display(node\* head)

//we are taking the value of head as an address

{

    node\* temp = head;

    while(temp!=NULL)

    {

        cout<<temp->data<<"->";

        temp = temp->next;

    }

cout<<"NULL"<<endl;

}

node\* reverse\_pointer(node\* &head)

{

    node\* prevptr = NULL;

    node\* currptr = head;

    node\* nextptr;

    while(currptr!=NULL)

    {

        nextptr=currptr->next;\

        currptr->next=prevptr;

        prevptr=currptr;

        currptr=nextptr;

    }

    return prevptr;//as we return the head but this head is the head of the reversed linked list

}

int count\_node(node\* head)

//as no changes are being made to the linked list we pass by variable and not the address

{

    node\* current = head;

    int count=1;

    while(current->next!=NULL)

    {

        current=current->next;

        count++;

    }

    return count;

}

node\* middle(node\* head)

{

    node\* slow = head;

    node\* fast = head;

    while(fast!=NULL&&fast->next!=NULL)

    {

        slow=slow->next;

        fast=fast->next->next;

    }

    return slow; //hence this returns the middle node of the linked list

}

bool check\_palindrome(node\* head)

{

    if(head==NULL)

    {return true;}

    node\* mid = middle(head);//hence we got the middle node and all we have to do is reverse it.

    node\* last = reverse\_pointer(mid->next);// hence the portion of the linked list starting from the node after the middle gets reversed

    //and in the head pointer of this reversed linked list is stored in last

    //but remember that the 'mid+1' node points to NULL

    node\* curr = head;

    while(last!=NULL)

    {

        if(last->data!=curr->data)

        {

            return false;

        }

        last=last->next;

        curr=curr->next;

    }

    return true;

}

int main()

{

    node\* head = NULL;

    insertAtTail(head,4);

    insertAtTail(head,2);

    insertAtTail(head,2);

    insertAtTail(head,4);

    display(head);

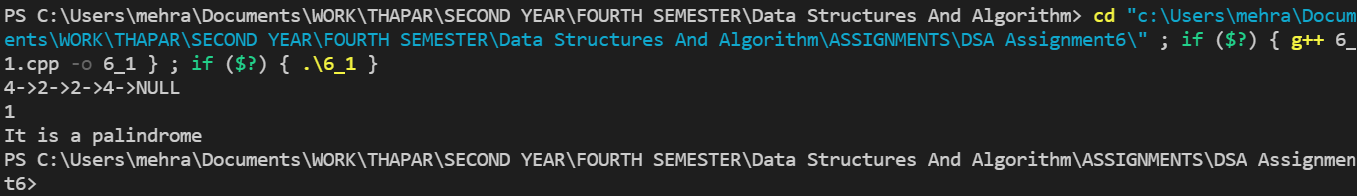
    cout<<check\_palindrome(head)<<endl;

    check\_palindrome(head)?cout<<"It is a palindrome"<<endl:cout<<"It is not a pallindrome"<<endl;

    return 0;

}

**OUTPUT**



Q2. Write a program using linked list to sum two numbers where digits of number are represented by the nodes of the linked list.

**CODE**

#include <bits/stdc++.h>

using namespace std;

/\* Linked list node \*/

class node {

public:

    int data;

    node\* next;

};

/\* Function to create a

new node with given data \*/

node\* newNode(int data)

{

    node\* new\_node = new node();

    new\_node->data = data;

    new\_node->next = NULL;

    return new\_node;

}

/\* Function to insert a node  \*/

void insert(node\*\* head\_ref, int new\_data)

{

    /\* allocate node \*/

    node\* new\_node = newNode(new\_data);

    /\* link the old list off the new node \*/

    new\_node->next = (\*head\_ref);

    /\* move the head to point to the new node \*/

    (\*head\_ref) = new\_node;

}

/\* Adds contents of two linked lists and

return the head node of resultant list \*/

node\* addTwoLists(node\* f, node\* s)

{

    // res is head node of the resultant list

    node\* res = NULL;

    node \*temp, \*prev = NULL;

    int carry = 0, sum;

    // while both lists exist

    while (f != NULL

        || s != NULL) {

        // Calculate value of next

        // digit in resultant list.

        sum = carry + (f ? f->data : 0)

            + (s ? s->data : 0);

        // update carry for next calculation

        carry = (sum >= 10) ? 1 : 0;

        // update sum if it is greater than 10

        sum = sum % 10;

        // Create a new node with sum as data

        temp = newNode(sum);

        // if this is the f node then

        // set it as head of the resultant list

        if (res == NULL)

            res = temp;

        // If this is not the f

        // node then connect it to the rest.

        else

            prev->next = temp;

        // Set prev for next insertion

        prev = temp;

        // Move f and s

        // pointers to next nodes

        if (f)

            f = f->next;

        if (s)

            s = s->next;

    }

    if (carry > 0)

        temp->next = newNode(carry);

    // return head of the resultant list

    return res;

}

node\* reverse\_ll(node\* head)

{

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

        return head;

    node\* rest = reverse\_ll(head->next);

    head->next->next = head;

    head->next = NULL;

    /\* fix the head pointer \*/

    return rest;

}

// A utility function to print a linked list

void printList(node\* node)

{

    while (node != NULL) {

        cout << node->data << " ";

        node = node->next;

    }

    cout << endl;

}

/\* Driver code \*/

int main(void)

{

    node\* res = NULL;

    node\* f = NULL;

    node\* s = NULL;

    // create f list 7->5->9->4->6

    insert(&f, 6);

    insert(&f, 4);

    insert(&f, 9);

    insert(&f, 5);

    insert(&f, 7);

    printf("First List is ");

    printList(f);

    // create s list 8->4

    insert(&s, 4);

    insert(&s, 8);

    cout << "s List is ";

    printList(s);

    // reverse\_ll both the lists

    f = reverse\_ll(f);

    s = reverse\_ll(s);

    // Add the two lists

    res = addTwoLists(f, s);

    // reverse\_ll the res to get the sum

    res = reverse\_ll(res);

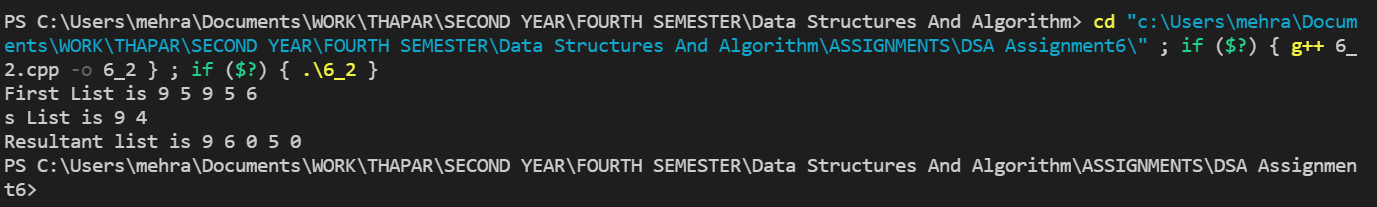
    cout << "Resultant list is ";

    printList(res);

    return 0;

}

**OUTPUT**



Q3. Write a program for Bubble sort using linked list and also do analysis of bubble sort with this data structure.

**CODE**

//bubble ort using linked list

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val)

{

    data = val;

    next = NULL;

}

};

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

//we are taking value of head by address because we want to modify our linked list

{

    node\* n = new node(val);

    //incase we are adding the first node to the linked list just use the below if statement

    if(head==NULL)

    {

        head=n;

        return;

    }

    node\* temp = head;

    while(temp->next!=NULL)

    {

        temp=temp->next;

    }

    //when temp reaches null, insert the node at the tail

    temp->next=n;

}

void display(node\* head)

//we are taking the value of head as an address

{

    node\* temp = head;

    while(temp!=NULL)

    {

        cout<<temp->data<<"->";

        temp = temp->next;

    }

cout<<"NULL"<<endl;

}

void bubbleSort(node\* &head)

{ //The bubble sort code was simply changed by using pointers instead of variables

    node\* current = head;

    for(node\* i = current;i->next->next!=NULL;i=i->next)

        { //i<n-1      ANALOGOUS TO THIS

            for(node\* j=current;j->next!=NULL;j=j->next)

            { //j<n    ANALOGOUS TO THIS

                if(j->data>j->next->data)

                {

                    int temp;

                    temp=j->data;

                    j->data=j->next->data;

                    j->next->data=temp;

                }

            }

        }

}

int main()

{

    node\* head = NULL;

    insertAtTail(head,34);

    insertAtTail(head,5);

    insertAtTail(head,12);

    insertAtTail(head,89);

    insertAtTail(head,0);

    insertAtTail(head,46);

    display(head);

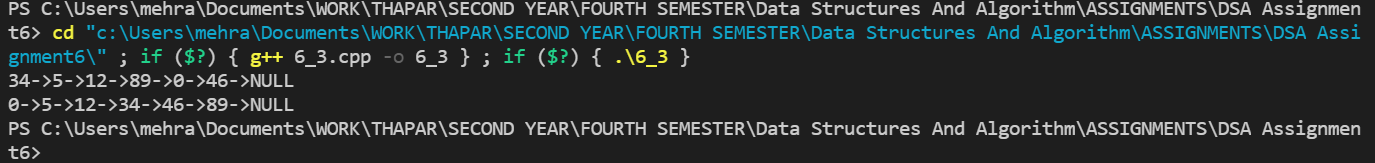
    bubbleSort(head);

    display(head);

    return 0;

}

**OUTPUT**



Q4. Check whether the linked list is having loop or not.

**CODE**

#include<iostream>

using namespace std;

class node{

public:

node\* next;

int data;

//constructor

node(int val)

{

    data=val;

    next=NULL;

}

};

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

{

    node\* n = new node(val);

    n->next = head; //this is the node that is to be inserted at the beginning.

    //Currently head is pointing to the "2nd element". So this step points n to the 2nd element as n is supposed to be the 1st element

    head=n; //using this we change head to point to n. and n becomes the first element.

}

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

//we are taking value of head by address because we want to modify our linked list

{

    node\* n = new node(val);

    //incase we are adding the first node to the linked list just use the below if statement

    if(head==NULL)

    {

        head=n;

        return;

    }

    node\* temp = head;

    while(temp->next!=NULL)

    {

        temp=temp->next;

    }

    //when temp reaches null, insert the node at the tail

    temp->next=n;

}

void display(node\* head)

//we are taking the value of head as an address

{

    node\* temp = head;

    while(temp!=NULL)

    {

        cout<<temp->data<<"->";

        temp = temp->next;

    }

cout<<"NULL"<<endl;

}

bool detectCycle(node\* &head)

{

    node\* slow = head;

    node\* fast = head;

    while(fast!=NULL&&fast->next!=NULL)

    {

        slow=slow->next;// moves 1 step at a time

        fast = fast->next->next;//moves 2 steps at a time

        //in case cycle is present

        if(slow==fast)

        {

            return true;

        }

    }

    //in case cycle is not present in the loop

    return false;

}

void makeCycle(node\* &head, int pos)

{

    //pos tells the position from which we want the loop to start

    node\* temp = head;

    node\* startNode;

    int count=1;

    while(temp->next!=NULL)

    {

        if(count==pos)

        {

            startNode = temp;//assigns that position to new pointer from where the loop is to start

        }

        temp = temp->next;

        count++;

    }

    //at the end of the loop, temp points to the last node of the linked list

    temp->next=startNode;

    //this connects the end to this loop

}

int main()

{

    node\* head = NULL;

    insertAtTail(head,1);

    insertAtTail(head,2);

    insertAtTail(head,3);

    insertAtHead(head,4);

    display(head);

    makeCycle(head,3);

    //display(head);

    detectCycle(head)?cout<<"Cycle is present":cout<<"Cycle is absent";

    return 0;

}

**OUTPUT**

