Lab Task 5

Problem: 1 | Concatenating Lists

Write a program that concatenates two linked list objects of characters. The program should

include function concatenate, which takes references to both list objects as arguments and

concatenates the second list to the first list.

Code:

//Single Linklist implementation

#include<iostream>

using namespace std;

class node {

    public:

        int data;

        node \*next;

};

node \*head1= new node();

node \*head2= new node();

node \*curr1= new node();

node \*curr2= new node();

node \*List1= new node();

node \*List2= new node();

int length=0;

void GoToHead1() { // set curr pointer to head node;

    curr1= head1;

}

void GoToHead2(){

    curr2 = head2;

}

void insertNodeAtEnd1(int val) { // This function will insert new node at the end.

    GoToHead1();

    node \*t= new node();

    while(curr1->next!=NULL)

        curr1= curr1->next;

    t->data= val;

    t->next= NULL;

    curr1->next= t;

    length++;

}

void insertNodeAtEnd2(int val) { // This function will insert new node at the end.

    GoToHead2();

    node \*t= new node();

    while(curr2->next!=NULL)

        curr2= curr2->next;

    t->data= val;

    t->next= NULL;

    curr2->next= t;

    length++;

}

void AddNodeBeforeHead1( int val) { // This function will insert new node as a head.

    GoToHead1();

    node \*t= new node();

    t->data= val;

    t->next= curr1;

    head1= t;

    length++;

}

void AddNodeBeforeHead2( int val) { // This function will insert new node as a head.

    GoToHead2();

    node \*t= new node();

    t->data= val;

    t->next= curr2;

    head2= t;

    length++;

}

void InsertAfterSpecificKey1(int val, int key ) {

    node \*t= new node();

    GoToHead1();

    while (curr1!=NULL) {

        if (curr1->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr1->next;

            curr1->next= t;

            length++;

            break;

        }

        curr1= curr1->next;

    }

}

void InsertAfterSpecificKey2(int val, int key ) {

    node \*t= new node();

    GoToHead2();

    while (curr2!=NULL) {

        if (curr2->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr2->next;

            curr2->next= t;

            length++;

            break;

        }

        curr2= curr2->next;

    }

}

void InsertBeforeSpecificKey1(int val, int key ) {

    node \*ptr=NULL;

    GoToHead1();

    while (curr1!=NULL) {

        if (curr1->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr1;

        curr1= curr1->next;

    }

}

void InsertBeforeSpecificKey2(int val, int key ) {

    node \*ptr=NULL;

    GoToHead2();

    while (curr2!=NULL) {

        if (curr2->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr2;

        curr2= curr2->next;

    }

}

void printLinklist1() {

    GoToHead1();

    while(curr1!=NULL) {

        cout<<curr1->data<<"\t";

        curr1= curr1->next;

    }

}

void printLinklist2() {

    GoToHead2();

    while(curr2!=NULL) {

        cout<<curr2->data<<"\t";

        curr2= curr2->next;

    }

}

void DeleteNodeUsingKey1(int key) {

    GoToHead1();

    node \*prenode= new node();

    if(curr1->data== key) {

        head1= curr1->next;

        delete curr1;

        length--;

        return;

    } else

        while(curr1!=NULL) {

            if(curr1->data==key) {

                prenode->next= curr1->next;

                delete curr1;

                length--;

                break;

            }

            prenode= curr1;

            curr1=curr1->next;

        }

}

void DeleteNodeUsingKey2(int key) {

    GoToHead2();

    node \*prenode= new node();

    if(curr2->data== key) {

        head2= curr2->next;

        delete curr2;

        length--;

        return;

    } else

        while(curr2!=NULL) {

            if(curr2->data==key) {

                prenode->next= curr2->next;

                delete curr2;

                length--;

                break;

            }

            prenode= curr2;

            curr2=curr2->next;

        }

}

void DeleteNodeUsingPos1(int pos) {

    GoToHead1();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr1;

        head1= curr1->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr1;

            curr1= curr1->next;

        }

        prenode->next= curr1->next;

        delete curr1;

        length--;

    }

}

void DeleteNodeUsingPos2(int pos) {

    GoToHead2();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr2;

        head2= curr2->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr2;

            curr2= curr2->next;

        }

        prenode->next= curr2->next;

        delete curr2;

        length--;

    }

}

void InsertNodeUsingKey1(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey1( val, key);

    else

        InsertAfterSpecificKey1( val, key);

}

void InsertNodeUsingKey2(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey2( val, key);

    else

        InsertAfterSpecificKey2( val, key);

}

void InsertNodeUsingPos1(int val, int pos, bool isBefore) {

    GoToHead1();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead1(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr1;

            curr1= curr1->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1->next;

            curr1->next= t;

        }

    }

}

void InsertNodeUsingPos2(int val, int pos, bool isBefore) {

    GoToHead2();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead2(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr2;

            curr2= curr2->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2->next;

            curr2->next= t;

        }

    }

}

void concatenate( node\*List1, node\*List2)

{

    printLinklist1();

    while(curr1!=NULL) {

        cout<<curr1->data<<"\t";

        curr1= curr1->next;

    }

    curr1->next=List2;

}

int main () {

    head1->data= 1;

    head2->data=1;

    head1->next=NULL;

    head2->next=NULL;

    insertNodeAtEnd1(2);

    insertNodeAtEnd1(3);

    insertNodeAtEnd1(4);

    printLinklist1();

    cout<<endl;

    insertNodeAtEnd2(2);

    insertNodeAtEnd2(3);

    insertNodeAtEnd2(4);

    printLinklist2();

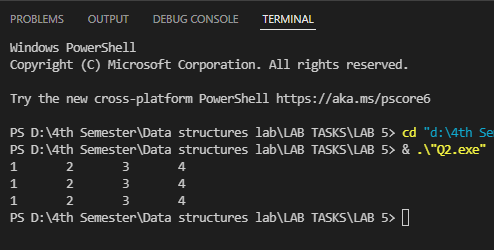
    cout<<endl;

    concatenate(List1,List2);

    return 0;

}

Output:



Problem: 2 | Merging Ordered Lists

Write a program that merges two ordered list objects of integers into a single ordered list

object of integers. Function merge should receive references to each of the list objects to be

merged and reference to a list object into which the merged elements will be placed.

CODE:

#include<iostream>

using namespace std;

class node {

    public:

        int data;

        node \*next;

};

node \*head1= new node();

node \*head2= new node();

node \*curr1= new node();

node \*curr2= new node();

node \*List1= new node();

node \*List2= new node();

int length=0;

void GoToHead1() { // set curr pointer to head node;

    curr1= head1;

}

void GoToHead2(){

    curr2 = head2;

}

void insertNodeAtEnd1(int val) { // This function will insert new node at the end.

    GoToHead1();

    node \*t= new node();

    while(curr1->next!=NULL)

        curr1= curr1->next;

    t->data= val;

    t->next= NULL;

    curr1->next= t;

    length++;

}

void insertNodeAtEnd2(int val) { // This function will insert new node at the end.

    GoToHead2();

    node \*t= new node();

    while(curr2->next!=NULL)

        curr2= curr2->next;

    t->data= val;

    t->next= NULL;

    curr2->next= t;

    length++;

}

void AddNodeBeforeHead1( int val) { // This function will insert new node as a head.

    GoToHead1();

    node \*t= new node();

    t->data= val;

    t->next= curr1;

    head1= t;

    length++;

}

void AddNodeBeforeHead2( int val) { // This function will insert new node as a head.

    GoToHead2();

    node \*t= new node();

    t->data= val;

    t->next= curr2;

    head2= t;

    length++;

}

void InsertAfterSpecificKey1(int val, int key ) {

    node \*t= new node();

    GoToHead1();

    while (curr1!=NULL) {

        if (curr1->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr1->next;

            curr1->next= t;

            length++;

            break;

        }

        curr1= curr1->next;

    }

}

void InsertAfterSpecificKey2(int val, int key ) {

    node \*t= new node();

    GoToHead2();

    while (curr2!=NULL) {

        if (curr2->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr2->next;

            curr2->next= t;

            length++;

            break;

        }

        curr2= curr2->next;

    }

}

void InsertBeforeSpecificKey1(int val, int key ) {

    node \*ptr=NULL;

    GoToHead1();

    while (curr1!=NULL) {

        if (curr1->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr1;

        curr1= curr1->next;

    }

}

void InsertBeforeSpecificKey2(int val, int key ) {

    node \*ptr=NULL;

    GoToHead2();

    while (curr2!=NULL) {

        if (curr2->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr2;

        curr2= curr2->next;

    }

}

// void printLinklist1() {

//  GoToHead1();

//  while(curr1!=NULL) {

//      cout<<curr1->data<<"\t";

//      curr1= curr1->next;

//  }

// }

// void printLinklist2() {

//  GoToHead2();

//  while(curr2!=NULL) {

//      cout<<curr2->data<<"\t";

//      curr2= curr2->next;

//  }

// }

void DeleteNodeUsingKey1(int key) {

    GoToHead1();

    node \*prenode= new node();

    if(curr1->data== key) {

        head1= curr1->next;

        delete curr1;

        length--;

        return;

    } else

        while(curr1!=NULL) {

            if(curr1->data==key) {

                prenode->next= curr1->next;

                delete curr1;

                length--;

                break;

            }

            prenode= curr1;

            curr1=curr1->next;

        }

}

void DeleteNodeUsingKey2(int key) {

    GoToHead2();

    node \*prenode= new node();

    if(curr2->data== key) {

        head2= curr2->next;

        delete curr2;

        length--;

        return;

    } else

        while(curr2!=NULL) {

            if(curr2->data==key) {

                prenode->next= curr2->next;

                delete curr2;

                length--;

                break;

            }

            prenode= curr2;

            curr2=curr2->next;

        }

}

void DeleteNodeUsingPos1(int pos) {

    GoToHead1();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr1;

        head1= curr1->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr1;

            curr1= curr1->next;

        }

        prenode->next= curr1->next;

        delete curr1;

        length--;

    }

}

void DeleteNodeUsingPos2(int pos) {

    GoToHead2();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr2;

        head2= curr2->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr2;

            curr2= curr2->next;

        }

        prenode->next= curr2->next;

        delete curr2;

        length--;

    }

}

void InsertNodeUsingKey1(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey1( val, key);

    else

        InsertAfterSpecificKey1( val, key);

}

void InsertNodeUsingKey2(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey2( val, key);

    else

        InsertAfterSpecificKey2( val, key);

}

void InsertNodeUsingPos1(int val, int pos, bool isBefore) {

    GoToHead1();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead1(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr1;

            curr1= curr1->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr1->next;

            curr1->next= t;

        }

    }

}

void InsertNodeUsingPos2(int val, int pos, bool isBefore) {

    GoToHead2();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead2(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr2;

            curr2= curr2->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr2->next;

            curr2->next= t;

        }

    }

}

void Merge( node\*List1, node\*List2)

{

    //printLinklist1();

    while(curr1!=NULL) {

        cout<<curr1->data<<"\t";

        curr1= curr1->next;

    }

    curr1->next=List2;

}

void merge(node \*first, node \*\*second)

{

    node \*firstRef = first;

    // finding the lat node of first linked list

    while (firstRef->next != NULL)

    {

         firstRef = firstRef->next;

    }

    firstRef->next = \*second;

}

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

{

    node\* new\_node = new node();

    new\_node->data = new\_data;

    new\_node->next = (\*head\_ref);

    (\*head\_ref) = new\_node;

}

void Print(node \*head)

{

    node \*temp = head;

    while (temp != NULL)

    {

        cout<<temp->data<<" ";

        temp = temp->next;

    }

    cout<<endl;

}

int main () {

    head1->data= 1;

    head2->data=1;

    head1->next=NULL;

    head2->next=NULL;

    // insertNodeAtEnd1(2);

    // insertNodeAtEnd1(3);

    // insertNodeAtEnd1(4);

    // printLinklist1();

    cout<<endl;

    // InsertAfterSpecificKey1(99, 2);

    // printLinklist1();

    // cout<<endl;

    // DeleteNodeUsingKey1(99);

    // printLinklist1();

    // cout<<endl;

    // InsertBeforeSpecificKey1(99, 2);

    // printLinklist1();

    // cout<<endl;

    // InsertNodeUsingPos1(88,1,true);

    // printLinklist1();

    // cout<<endl;

    // DeleteNodeUsingPos1(1);

    // DeleteNodeUsingPos1(2);

    // printLinklist1();

    // cout<<endl;

    // insertNodeAtEnd2(52);

    // insertNodeAtEnd2(37);

    // insertNodeAtEnd2(48);

    // printLinklist2();

    // cout<<endl;

    // InsertAfterSpecificKey2(99, 2);

    // printLinklist2();

    // cout<<endl;

    // DeleteNodeUsingKey2(99);

    // printLinklist2();

    // cout<<endl;

    // InsertBeforeSpecificKey2(99, 2);

    // printLinklist2();

    // cout<<endl;

    // InsertNodeUsingPos2(88,1,true);

    // printLinklist2();

    // cout<<endl;

    // DeleteNodeUsingPos2(1);

    // DeleteNodeUsingPos2(2);

    // printLinklist2();

    // cout<<endl;

    //Merge(List1,List2);

    node \*first = NULL, \*second = NULL;

    //inserting 1, 2 and 3 in first linked list

    insert(&first, 5);

    insert(&first, 4);

    insert(&first, 3);

    insert(&first, 2);

    insert(&first, 1);

    cout<<"First Linked List:\n";

    Print(first);

    //inserting 4, 5, 6, 7, 8 in secnd linked list

    insert(&second, 10);

    insert(&second, 9);

    insert(&second, 8);

    insert(&second, 7);

    insert(&second, 6);

    cout<<"\nSecond Linked List:\n";

    Print(second);

    merge(first, &second);

    //firstrinting merged linked list

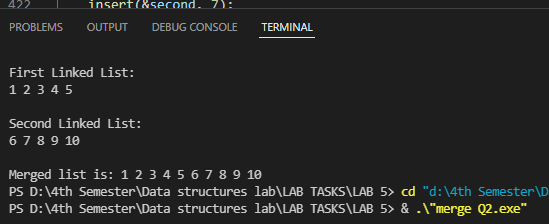
    cout<<"\nMerged list is: ";

    Print(first);

    return 0;

}

Output:



Problem: 3 | Copying a List in Reverse Order

Write a program that creates a linked list object of 10 characters and creates a second list

object containing a copy of the first list, but in reverse order.

Code”

#include<iostream>

using namespace std;

class node {

    public:

        int data;

        node \*next;

};

node \*head= new node();

node \*curr= new node();

int length=0;

void GoToHead() { // set curr pointer to head node;

    curr= head;

}

void insertNodeAtEnd(int val) { // This function will insert new node at the end.

    GoToHead();

    node \*t= new node();

    while(curr->next!=NULL)

        curr= curr->next;

    t->data= val;

    t->next= NULL;

    curr->next= t;

    length++;

}

void AddNodeBeforeHead( int val) { // This function will insert new node as a head.

    GoToHead();

    node \*t= new node();

    t->data= val;

    t->next= curr;

    head= t;

    length++;

}

void InsertAfterSpecificKey(int val, int key ) {

    node \*t= new node();

    GoToHead();

    while (curr!=NULL) {

        if (curr->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr->next;

            curr->next= t;

            length++;

            break;

        }

        curr= curr->next;

    }

}

void InsertBeforeSpecificKey(int val, int key ) {

    node \*ptr=NULL;

    GoToHead();

    while (curr!=NULL) {

        if (curr->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr;

        curr= curr->next;

    }

}

void printLinklist() {

    GoToHead();

    while(curr!=NULL) {

        cout<<curr->data<<"\t";

        curr= curr->next;

    }

}

void DeleteNodeUsingKey(int key) {

    GoToHead();

    node \*prenode= new node();

    if(curr->data== key) {

        head= curr->next;

        delete curr;

        length--;

        return;

    } else

        while(curr!=NULL) {

            if(curr->data==key) {

                prenode->next= curr->next;

                delete curr;

                length--;

                break;

            }

            prenode= curr;

            curr=curr->next;

        }

}

void DeleteNodeUsingPos(int pos) {

    GoToHead();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr;

        head= curr->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr;

            curr= curr->next;

        }

        prenode->next= curr->next;

        delete curr;

        length--;

    }

}

void InsertNodeUsingKey(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey( val, key);

    else

        InsertAfterSpecificKey( val, key);

}

void InsertNodeUsingPos(int val, int pos, bool isBefore) {

    GoToHead();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr;

            curr= curr->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr->next;

            curr->next= t;

        }

    }

}

void reverse()

    {

        // Initialize current, previous and

        // next pointers

        node\* current = head;

        node \*prev = NULL, \*next = NULL;

        while (current != NULL) {

            // Store next

            next = current->next;

            // Reverse current node's pointer

            current->next = prev;

            // Move pointers one position ahead.

            prev = current;

            current = next;

        }

        head = prev;

    }

int main () {

    head->data= 1;

    head->next=NULL;

    cout << "Given linked list\n";

    insertNodeAtEnd(2);

    insertNodeAtEnd(3);

    insertNodeAtEnd(4);

    insertNodeAtEnd(6);

    insertNodeAtEnd(8);

    insertNodeAtEnd(1);

    insertNodeAtEnd(3);

    insertNodeAtEnd(5);

    insertNodeAtEnd(9);

    insertNodeAtEnd(12);

    printLinklist();

    cout<<endl;

    // InsertAfterSpecificKey(99, 2);

    // printLinklist();

    // cout<<endl;

    // DeleteNodeUsingKey(99);

    // printLinklist();

    // cout<<endl;

    // InsertBeforeSpecificKey(99, 2);

    // printLinklist();

    // cout<<endl;

    // InsertNodeUsingPos(88,1,true);

    // printLinklist();

    // cout<<endl;

    // DeleteNodeUsingPos(1);

    // DeleteNodeUsingPos(2);

    // printLinklist();

    // cout<<endl;

    reverse();

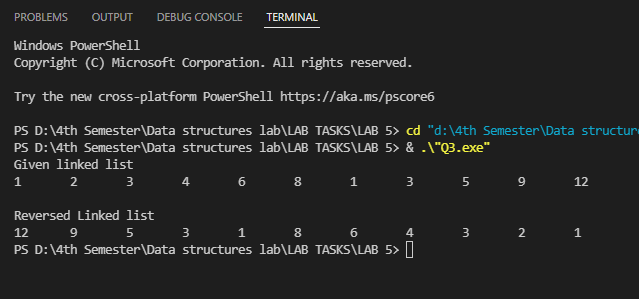
    cout << "\nReversed Linked list \n";

    printLinklist();

    return 0;

}

Output:

Problem: 4 |

Write a C++ program to move the last node to the front of a Singly Linked List.

Example:

Given: 1->2->3->4->NULL

Return: 4->1->2->3->NULL

Code:

#include<iostream>

#include <bits/stdc++.h>

using namespace std;

class node {

    public:

        int data;

        node \*next;

};

node \*head= new node();

node \*curr= new node();

int length=0;

void GoToHead() { // set curr pointer to head node;

    curr= head;

}

void insertNodeAtEnd(int val) { // This function will insert new node at the end.

    GoToHead();

    node \*t= new node();

    while(curr->next!=NULL)

        curr= curr->next;

    t->data= val;

    t->next= NULL;

    curr->next= t;

    length++;

}

void AddNodeBeforeHead( int val) { // This function will insert new node as a head.

    GoToHead();

    node \*t= new node();

    t->data= val;

    t->next= curr;

    head= t;

    length++;

}

void InsertAfterSpecificKey(int val, int key ) {

    node \*t= new node();

    GoToHead();

    while (curr!=NULL) {

        if (curr->data==key) {

            t->data= val;

            t->next= NULL;

            t->next= curr->next;

            curr->next= t;

            length++;

            break;

        }

        curr= curr->next;

    }

}

void InsertBeforeSpecificKey(int val, int key ) {

    node \*ptr=NULL;

    GoToHead();

    while (curr!=NULL) {

        if (curr->data==key) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr;

            ptr->next= t;

            length++;

            break;

        }

        ptr= curr;

        curr= curr->next;

    }

}

// void printLinklist(node\*node) {

//  GoToHead();

//  while(curr!=NULL) {

//      cout<<curr->data<<"\t";

//      curr= curr->next;

//  }

// }

void DeleteNodeUsingKey(int key) {

    GoToHead();

    node \*prenode= new node();

    if(curr->data== key) {

        head= curr->next;

        delete curr;

        length--;

        return;

    } else

        while(curr!=NULL) {

            if(curr->data==key) {

                prenode->next= curr->next;

                delete curr;

                length--;

                break;

            }

            prenode= curr;

            curr=curr->next;

        }

}

void DeleteNodeUsingPos(int pos) {

    GoToHead();

    node \*prenode= new node();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 ) { // if we want to delet head node

        prenode= curr;

        head= curr->next;

        delete prenode;

        length--;

    } else {

        for (int x=1; x<pos; x++) {

            prenode= curr;

            curr= curr->next;

        }

        prenode->next= curr->next;

        delete curr;

        length--;

    }

}

void InsertNodeUsingKey(int val, int key, bool isBefore) {

    if (isBefore)

        InsertBeforeSpecificKey( val, key);

    else

        InsertAfterSpecificKey( val, key);

}

void InsertNodeUsingPos(int val, int pos, bool isBefore) {

    GoToHead();

    if(pos>length) {

        cout<<"This Position dosenot exist"<<endl;

        return;

    } else if (pos==1 && isBefore ) { // if we want to insert before head

        AddNodeBeforeHead(val);

    } else {

        node \*prenode= new node();

        for (int x=1; x<pos; x++) {

            prenode= curr;

            curr= curr->next;

        }

        if (isBefore) {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr;

            prenode->next= t;

        } else {

            node \*t= new node();

            t->data= val;

            t->next= NULL;

            t->next= curr->next;

            curr->next= t;

        }

    }

}

void moveToFront(node \*\*head)

{

    /\* If linked list is empty, or

    it contains only one node,

    then nothing needs to be done,

    simply return \*/

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

        return;

    /\* Initialize second last

    and last pointers \*/

    node \*secLast = NULL;

    node \*last = \*head;

    /\*After this loop secLast contains

    address of second last node and

    last contains address of last node in Linked List \*/

    while (last->next != NULL)

    {

        secLast = last;

        last = last->next;

    }

    /\* Set the next of second last as NULL \*/

    secLast->next = NULL;

    /\* Set next of last as head node \*/

    last->next = \*head;

    /\* Change the head pointer

    to point to last node now \*/

    \*head = last;

}

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

{

    /\* allocate node \*/

    node\* new\_node = new node();

    /\* put in the data \*/

    new\_node->data = new\_data;

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

    new\_node->next = (\*head);

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

    (\*head) = new\_node;

}

void printLinklist(node \*node)

{

    while(node != NULL)

    {

        cout << node->data << " ";

        node = node->next;

    }

}

int main () {

    // head->data= 1;

    // head->next=NULL;

    node \*start = NULL;

    // insertNodeAtEnd(2);

    // insertNodeAtEnd(3);

    // insertNodeAtEnd(4);

    // //printLinklist(start);

    // cout<<endl;

    // InsertAfterSpecificKey(99, 2);

    // printLinklist();

    // cout<<endl;

    // DeleteNodeUsingKey(99);

    // printLinklist();

    // cout<<endl;

    // InsertBeforeSpecificKey(99, 2);

    // printLinklist();

    // cout<<endl;

    // InsertNodeUsingPos(88,1,true);

    // printLinklist();

    // cout<<endl;

    // DeleteNodeUsingPos(1);

    // DeleteNodeUsingPos(2);

    // printLinklist();

    // cout<<endl;

    insert(&start, 5);

    insert(&start, 4);

    insert(&start, 3);

    insert(&start, 2);

    insert(&start, 1);

    cout<<"Linked list before moving last to front\n";

    printLinklist(start);

    //moveToFront(&start);

    cout<<"\nLinked list after removing last to front\n";

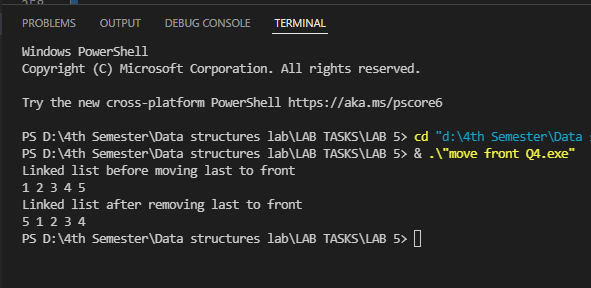
    moveToFront(&start);

    printLinklist(start);

    return 0;

}

Output:



Problem: 5 | Linked List the Palindrome

Write a function to check whether the given Singly Linked List is Palindrome or not.

Code:

#include<bits/stdc++.h>

using namespace std;

class node {

public:

        int data;

        node(int d){

            data = d;

        }

        node \*ptr;

};

// Function to check if the linked list

// is palindrome or not

bool isPalin(node\* head){

        // Temp pointer

        node\* slow= head;

        // Declare a stack

        stack <int> s;

        // Push all elements of the list

        // to the stack

        while(slow != NULL){

                s.push(slow->data);

                // Move ahead

                slow = slow->ptr;

        }

        // Iterate in the list again and

        // check by popping from the stack

        while(head != NULL ){

            // Get the top most element

            int i=s.top();

            // Pop the element

            s.pop();

            // Check if data is not

            // same as popped element

            if(head -> data != i){

                return false;

            }

            // Move ahead

        head=head->ptr;

        }

return true;

}

int main(){

    // Addition of linked list

    node one = node(1);

    node two = node(2);

    node three = node(3);

    node four = node(2);

    node five = node(1);

    // Initialize the next pointer

    // of every current pointer

    five.ptr = NULL;

    one.ptr = &two;

    two.ptr = &three;

    three.ptr = &four;

    four.ptr = &five;

    node\* temp = &one;

    // Call function to check palindrome or not

    int result = isPalin(&one);

    if(result == 1)

            cout<<"isPalindrome is true\n";

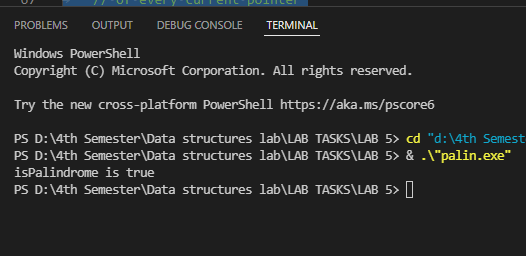
    else

        cout<<"isPalindrome is true\n";

return 0;

}

Output:



Problem: 6 | Remove Duplicates

Write a RemoveDuplicates() function which takes a list sorted in increasing order and deletes

any duplicate nodes from the list. Ideally, the list should only be traversed once.

Code:

#include<iostream>

using namespace std;

class node {

    public:

        int data;

        node \*next;

};

node \*head= new node();

node \*curr= new node();

int length=0;

void GoToHead() {

    curr= head;

}

void insertNodeAtEndHead(int val) {

    GoToHead();

    node \*p= new node();

    while(curr->next!=NULL)

        curr= curr->next;

    p->data= val;

    p->next= NULL;

    curr->next= p;

    length++;

}

void printLinklist() {

    GoToHead();

    while(curr!=NULL) {

        cout<<curr->data<<"\t";

        curr= curr->next;

    }

}

void removeDuplicate(node \*h)

{

    GoToHead();

    node\* prev;

    while(curr->next!=NULL)

    {

        prev=curr;

        curr=curr->next;

        if(prev->data == curr->data)

        {

            prev->next=curr->next;

            node\* temp=curr;

            curr=curr->next;

            delete temp;

            length--;

        }

    }

}

int main()

{

    head->data=1;

    head->next=NULL;

    insertNodeAtEndHead(4);

    insertNodeAtEndHead(4);

    insertNodeAtEndHead(6);

    insertNodeAtEndHead(7);

    insertNodeAtEndHead(9);

    insertNodeAtEndHead(9);

    insertNodeAtEndHead(15);

    printLinklist();

    cout<<endl;

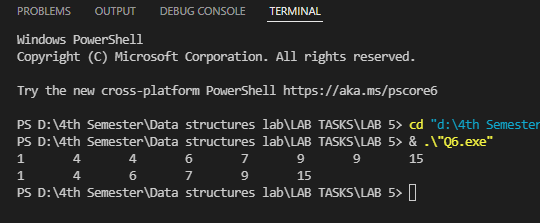
    removeDuplicate(head);

    printLinklist();

    return 0;

}

Output:

\

The end……………….