Task1-4

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val){

data = val;

next = NULL;

}

};

class LinkedList{

public:

node\* head;

LinkedList(){

head=NULL;

}

// Adding the node at the end of the list

void inserAtEnd(int val){

node\* newnode = new node(val);

if(head==NULL){

head=newnode;

return;

}

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newnode;

}

// Adding the node at the starting position

void insertAtHead(int val){

node\* newnode = new node(val);

newnode->next=head;

head=newnode;

}

// Inserting the node at the given value

void insertAtPos(int pos, int val){

if (pos==0){

insertAtHead(val);

return;

}

node\* newnode = new node(val);

node\* temp= head;

int currpos = 0;

while(currpos!=pos-1){

temp= temp->next;

currpos++;

}

newnode->next = temp->next;

temp->next=newnode;

}

//Displaying the list

void display(){

node\* temp = head;

while(temp!=NULL){

cout<<temp->data<<"->";

temp=temp->next;

}

cout<<"NULL";

}

};

int main(){

LinkedList newlist;

cout<<"Inseting at the end of the list"<<endl;

newlist.inserAtEnd(3);

newlist.inserAtEnd(4);

newlist.inserAtEnd(5);

newlist.inserAtEnd(10);

newlist.display();

cout <<endl<<"Inserting at the Start of the list"<<endl;

newlist.insertAtHead(55);

newlist.insertAtHead(60);

newlist.display();

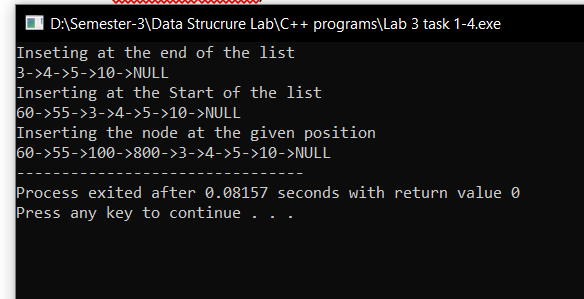
cout<<endl<<"Inserting the node at the given position"<<endl;

newlist.insertAtPos(2,100);

newlist.insertAtPos(3,800);

newlist.display();

}



Task 5

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val){

data=val;

next=NULL;

}

};

class LinkedList{

public:

node\* head;

LinkedList(){

head=NULL;

}

//Adding the node at the start

void insertAtHead(int val){

node\* newnode = new node(val);

newnode->next=head;

head= newnode;

}

//Display

void display(){

node\* temp = head;

while(temp!=NULL){

cout<<temp->data<<"->";

temp= temp->next;

}

cout<<"NULL";

}

//Deleting From end;

void del(){

node\* second\_last= head;

while(second\_last->next->next!=NULL){

second\_last= second\_last->next;

}

node\* temp = second\_last->next;

second\_last->next=NULL;

temp=NULL;

}

void delAtPos(int pos){

int currpos =0;

node\* prev = head;

while(currpos!=pos-2){

prev=prev->next;

currpos++;

}

node\* temp = prev->next;

prev->next= prev->next->next;

temp=NULL;

}

};

int main(){

LinkedList newlist;

newlist.insertAtHead(7);

newlist.insertAtHead(897);

newlist.insertAtHead(667);

newlist.insertAtHead(10);

newlist.insertAtHead(15);

newlist.insertAtHead(18);

newlist.insertAtHead(1300);

newlist.insertAtHead(177);

newlist.display();

cout<<endl<<"Deleting from the end"<<endl;

newlist.del();

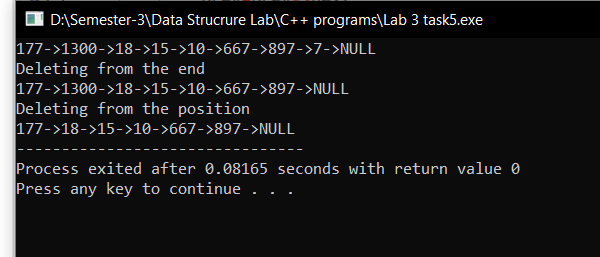
newlist.display();

cout<<endl<<"Deleting from the position"<<endl;

newlist.delAtPos(2);

newlist.display();

}



Task 6

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val){

data=val;

next=NULL;

}

};

class LinkedList{

public:

node\* head;

LinkedList(){

head=NULL;

}

// Adding the node at the end of the list

void insertAtEnd(int val){

node\* newnode = new node(val);

if (head == NULL) {

head = newnode;

return;

}

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newnode;

}

// Display fun

void display() {

node\* temp = head;

while (temp != NULL) {

cout << temp->data << "->";

temp = temp->next;

}

cout << "NULL" << endl;

}

//Updating the list

void update(int pv, int nv){

node\* temp = head;

while(temp->data!=pv){

temp= temp->next;

}

temp->data=nv;

}

};

int main(){

LinkedList newlist;

newlist.insertAtEnd(4);

newlist.insertAtEnd(1);

newlist.insertAtEnd(5);

newlist.insertAtEnd(7);

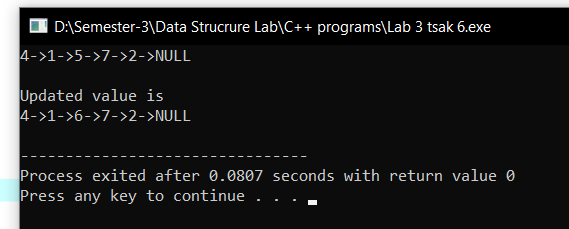
newlist.insertAtEnd(2);

newlist.display();

cout<<endl<<"Updated value is "<<endl;

newlist.update(5,6);

newlist.display();

}

Task 7

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val){

data=val;

next=NULL;

}

};

class LinkedList{

public:

node\* head;

LinkedList(){

head=NULL;

}

//Inserting the value at the head

void insertAtHead(int val){

node\* newnode = new node(val);

newnode->next=head;

head=newnode;

}

//Inserting the value at the end

void insertAtEnd(int val){

node\* newnode= new node(val);

if(head==NULL){

head = newnode;

return;

}

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next= newnode;

}

//Display

void display(){

node\* temp = head;

while(temp!=NULL){

cout<<temp->data<<"->";

temp=temp->next;

}

cout<<"NULL";

}

//Operation

void operation(){

LinkedList update;

int evcnt=0;

int odcnt=0;

node\* temp = head;

do{

// cout<<endl<<temp->data<<"->";

if(temp->data%2 ==0){

update.insertAtEnd(temp->data);

// cout<<endl<<"inserted"<<temp->data<<endl;

}

temp=temp->next;

}while(temp!=NULL);

//cout<<endl<<"This is second do-while"<<endl;

temp=head;

do{

// cout<<endl<<temp->data<<"->";

if(temp->data%2 ==1){

update.insertAtEnd(temp->data);

// cout<<endl<<"inserted"<<temp->data<<endl;

}

temp=temp->next;

}while(temp!=NULL);

cout<<endl<<"NEW LIST IS"<<endl;

update.display();

}

};

int main(){

LinkedList newlist;

newlist.insertAtHead(6);

newlist.insertAtHead(7);

newlist.insertAtHead(1);

newlist.insertAtHead(4);

newlist.insertAtHead(5);

newlist.insertAtHead(10);

newlist.insertAtHead(12);

newlist.insertAtHead(8);

newlist.insertAtHead(15);

newlist.insertAtHead(17);

newlist.display();

// newlist.insertAtEnd(300);

newlist.operation();

cout<<endl;

LinkedList evenList;

evenList.insertAtEnd(8);

evenList.insertAtEnd(12);

evenList.insertAtEnd(10);

evenList.display();

evenList.operation();

LinkedList oddList;

oddList.insertAtEnd(1);

oddList.insertAtEnd(3);

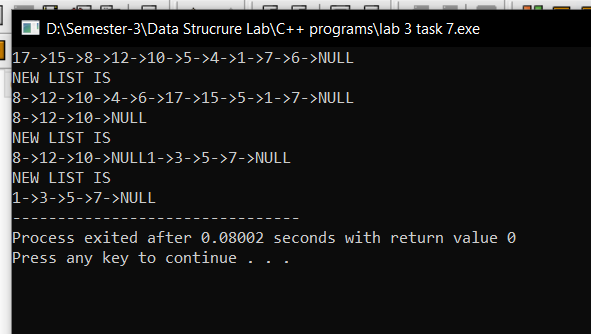
oddList.insertAtEnd(5);

oddList.insertAtEnd(7);

oddList.display();

oddList.operation();

}



Task 8

#include<iostream>

using namespace std;

class node{

public:

int data;

node\* next;

node(int val){

data=val;

next=NULL;

}

};

class LinkedList{

public:

node\* head;

LinkedList(){

head=NULL;

}

// Inserting the value at the end of List

void insertAtTail(int val){

node\* newnode = new node(val);

if (head == NULL) {

head = newnode;

return;

}

node\* temp = head;

while(temp->next!=NULL){

temp = temp->next;

}

temp->next = newnode;

}

// Displaying the linked List

void display() {

node\* temp = head;

while (temp != NULL) {

cout << temp->data << "->";

temp = temp->next;

}

cout << "NULL" << endl;

}

//Palindrome

void isPalindrome() {

node\* temp = head;

int c = 0;

do {

c++;

temp = temp->next;

} while (temp != NULL);

// cout << c;

temp = head;

node\* check = head;

if (c % 2 == 1) {

int t = (c / 2) + 1;

// cout << endl << "t= " << t;

// cout << endl << " c is " << c;

int counter=0;

int k = 0;

do {

check = head;

for (int i = 0; i < c - 1 - k; i++) {

check = check->next;

}

if (temp->data == check->data) {

// cout << endl << "Passed";

counter++;

}

// cout << endl << "check data is " << check->data << endl;

// cout << endl << "temp data is " << temp->data << endl;

temp = temp->next;

k++;

// cout << endl << "t is " << t << "k is " << k;

} while (t != k);

if(counter==t){

cout<<"Palindrome";

}else{

cout<<"Not Palindrome";

}

}else{

int t = (c / 2) + 1;

// cout << endl << "t= " << t;

// cout << endl << " c is " << c;

int counter=0;

int k = 0;

do {

check = head;

for (int i = 0; i < c - 1 - k; i++) {

check = check->next;

}

if (temp->data == check->data) {

// cout << endl << "Passed";

counter++;

}

// cout << endl << "check data is " << check->data << endl;

// cout << endl << "temp data is " << temp->data << endl;

temp = temp->next;

k++;

// cout << endl << "t is " << t << "k is " << k;

} while (t != k);

if(counter==t){

cout<<"Palindrome";

}else{

cout<<"Not Palindrome";

}

}

}

};

int main(){

LinkedList newlist;

newlist.insertAtTail(1);

newlist.insertAtTail(0);

newlist.insertAtTail(2);

newlist.insertAtTail(0);

newlist.insertAtTail(1);

newlist.display();

newlist.isPalindrome();

}

