24k-0918 **DSA** LAB-03 TASKS

TASK-01

#include <iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node(int *val*){

        data = *val*;

        next = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(*val*);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                newNode->next = head;

                head = newNode;

            }

        }

        void insertAtEnd(int *val*){

            Node\* newNode = new Node(*val*);

            if(tail == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                tail->next = newNode;

                newNode->next = nullptr;

                tail = newNode;

            }

        }

        void insertAtPos(int *val*, int *pos*){

            if(*pos* == 0){

                insertAtHead(*val*);

                return;

            }

            int count = 0;

            Node \*ptr = head;

            while(ptr != nullptr && count < *pos*-1){

                ptr = ptr->next;

                count++;

            }

            if(ptr == nullptr){

                cout<<"Invalid pos"<<endl;

                return;

            }

            Node\* newNode = new Node(*val*);

            newNode->next = ptr->next;

            ptr->next = newNode;

        }

        void deleteVal(int *val*){

            Node\* ptr = head;

            Node\* prev = nullptr;

            while(ptr != nullptr){

                if(ptr->data == *val*){

                    if(ptr==head){

                        Node\* temp = head;

                        head = head->next;

                        delete temp;

                        ptr = head;

                    }else{

                        prev->next = ptr->next;

                        if(ptr == tail) tail = prev;

                        Node\* temp = ptr;

                        ptr = prev->next;

                        delete temp;

                    }

                }

                else{

                    prev = ptr;

                    ptr = ptr->next;

                }

            }

        }

        void display(){

            Node\* temp = head;

            while(temp != nullptr){

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

                temp = temp->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList n;

    int arr[] = {3, 1, 2, 5, 8};

    cout<<"Displayin Array: ";

    for(int i=0; i<5; i++) {

        cout<<arr[i]<<" ";

*//inserting in linked list;*

        n.insertAtEnd(arr[i]);

    }

    cout<<endl;

    n.insertAtEnd(9);

    n.insertAtPos(11, 3);

    n.insertAtHead(4);

    n.deleteVal(1);

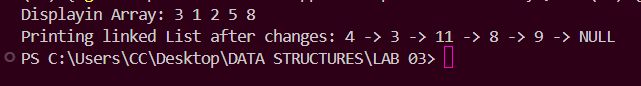
    n.deleteVal(2);

    n.deleteVal(5);

    cout<<"Printing linked List after changes: ";

    n.display();

}



TASK-02

#include <iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node(int *val*){

        data = val;

        next = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(val);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                newNode->next = head;

                head = newNode;

            }

        }

        void rotate(){

            if(head == nullptr || head->next == nullptr) return;

            Node\* temp = head;

            head = head->next;

            tail->next = temp;

            tail = temp;

            tail->next = nullptr;

        }

        void display(){

            Node\* temp = head;

            while(temp != nullptr){

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

                temp = temp->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList n;

    int arr[] = {2, 4, 6, 8, 1, 3, 5};

    int size = sizeof(arr)/sizeof(arr[0]);

    for(int i=0; i<size; i++) n.insertAtHead(arr[i]);

    cout<<"Linked List: ";

    n.display();

    int count;

    cout<<endl<<"Enter how many nodes to rotate: ";

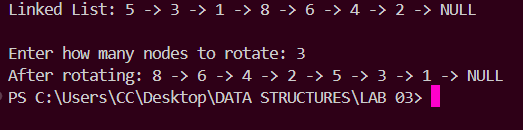
    cin>>count;

    for(int i=0; i<count; i++) n.rotate();

    cout<<"After rotating: ";

    n.display();

}



TASK-03

#include <iostream>

using namespace std;

struct Node {

    string name;

    Node \*next;

    Node(string *n*) : name(*n*), next(nullptr) {}

};

*// made with name Nodelist for my ease*

class NodeList{

    Node \*head;

    public:

        NodeList(){

            head = nullptr;

        }

        void reserveTicket(string *name*){

            Node \*newNode = new Node(*name*);

            if(head == nullptr || *name* < head->name){

                newNode->next = head;

                head = newNode;

                cout<<"Seat reserved"<<endl;

                return;

            }

            Node \*ptr = head;

            while(ptr->next != nullptr && ptr->next->name < *name*) ptr = ptr->next;

            if(ptr->next != nullptr && ptr->next->name == *name*){

                cout<<"Ticket reserved already"<<endl;

                delete newNode;

                return;

            }

            newNode->next = ptr->next;

            ptr->next = newNode;

            cout<<"Seat reserved"<<endl;

        }

        void cancelTicket(string *name*) {

            if(head == nullptr){

                cout<<"Epmty"<<endl;

                return;

            }

            if(head->name == *name*){

                Node \*temp = head;

                head = head->next;

                delete temp;

                cout<<"Reservation cancelled"<<endl;

                return;

            }

            Node \*ptr = head;

            while(ptr->next != nullptr){

                if(ptr->next->name == *name*){

                    Node \*temp = ptr->next;

                    ptr->next = temp->next;

                    delete temp;

                    cout<<"Reservation cancelled"<<endl;

                    return;

                }

                ptr = ptr->next;

            }

            cout<<"No reservation exist"<<endl;

        }

        void checkTicket(string *name*){

            Node \*ptr = head;

            while(ptr != nullptr){

                if(ptr->name == *name*){

                    cout<<*name*<<" has reservation"<<endl;

                    return;

                }

                ptr = ptr->next;

            }

            cout<<*name*<<" does not have reservation"<<endl;

        }

        void display(){

            Node \*ptr = head;

            cout<<"Passenger List:"<<endl;

            while(ptr != nullptr){

                cout<<ptr->name<<" -> ";

                ptr = ptr->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList n;

    int choice;

    string name;

    cout<<"\n--- Airline Ticket Reservation ---\n";

    cout<<"1. Reserve a Ticket"<<endl;

    cout<<"2. Cancel a Reservation"<<endl;

    cout<<"3. Check a Reservation"<<endl;

    cout<<"4. Display Passengers"<<endl;

    cout<<"5. Exit"<<endl;

    do{

        cout<<"Enter Choice: ";

        cin>>choice;

        switch (choice)

        {

        case 1:

            cout<<"Enter name: ";

            cin>>name;

            n.reserveTicket(name);

            break;

        case 2:

            cout<<"Enter name: ";

            cin>>name;

            n.cancelTicket(name);

            break;

        case 3:

            cout<<"Enter name: ";

            cin>>name;

            n.checkTicket(name);

            break;

        case 4:

            n.display();

            break;

        case 5:

            cout<<"Exiting.. ";

            break;

        default:

            cout<<"Invalid choice";

        }

    }while (choice != 5);

}



TASK-04

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int *val*)

    {

        data = *val*;

        next = nullptr;

    }

};

class NodeList

{

    Node \*head;

    Node \*tail;

public:

    NodeList()

    {

        head = nullptr;

        tail = nullptr;

    }

    void insertAtEnd(int *val*)

    {

        Node \*newNode = new Node(*val*);

        if (tail == nullptr)

        {

            head = newNode;

            tail = newNode;

        }

        else

        {

            tail->next = newNode;

            newNode->next = nullptr;

            tail = newNode;

        }

    }

    void EvenOdd()

    {

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

            return;

*// detaching odd nodes only*

        Node \*curr = head;

        Node \*prev = nullptr;

        Node \*oddHead = nullptr;

        Node \*oddTail = nullptr;

        while (curr != nullptr)

        {

            if (curr->data % 2 != 0)

            {

                if (prev != nullptr)

                {

                    prev->next = curr->next;

                }

                else

                {

                    head = curr->next;

                }

*// add to odd sublist*

                if (oddHead == nullptr)

                {

                    oddHead = curr;

                    oddTail = curr;

                }

                else

                {

                    oddTail->next = curr;

                    oddTail = curr;

                }

                curr = curr->next;

                oddTail->next = nullptr;

            }

            else

            {

                prev = curr;

                curr = curr->next;

            }

        }

*// Append odd list at the end*

        if (oddHead != nullptr)

        {

            if (tail != nullptr)

            {

                tail->next = oddHead;

            }

            else

            {

                head = oddHead;

            }

            tail = oddTail;

        }

    }

    void display()

    {

        Node \*temp = head;

        while (temp != nullptr)

        {

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

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main()

{

    NodeList n;

    int arr[] = {17, 15, 8, 12, 10, 5, 4, 1, 7, 6};

    for (int x : arr)

        n.insertAtEnd(x);

    cout << "Original List:" << endl;

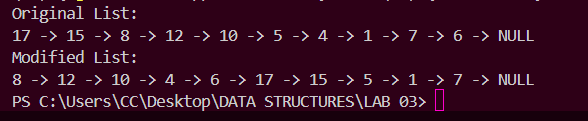
    n.display();

    n.EvenOdd();

    cout << "Modified List:" << endl;

    n.display();

}



TASK-05

#include <iostream>

using namespace std;

struct Node{

    char data;

    Node\* next;

    Node(int *val*){

        data = *val*;

        next = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(*val*);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                newNode->next = head;

                head = newNode;

            }

        }

        void insertAtEnd(int *val*){

            Node\* newNode = new Node(*val*);

            if(tail == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                tail->next = newNode;

                newNode->next = nullptr;

                tail = newNode;

            }

        }

        NodeList nodeCopy(){

            NodeList cpy;

            Node \*temp = head;

            while(temp != nullptr){

                cpy.insertAtEnd(temp->data);

                temp = temp->next;

            }

            return cpy;

        }

        void reverseList(){

            Node\* prev = NULL;

            Node\* curr = head;

            Node\* next = NULL;

            tail = head;

            while(curr != NULL){

                next = curr->next;

                curr->next = prev;

                prev = curr;

                curr = next;

            }

            head = prev;

        }

        bool isPalindrome(){

            NodeList cpy = nodeCopy();

            cpy.reverseList();

            Node \*p1 = head;

            Node \*p2 = cpy.head;

            while(p1 != nullptr && p2 != nullptr){

                if(p1->data != p2->data) return false;

                p1 = p1->next;

                p2 = p2->next;

            }

            return true;

        }

        void display(){

            Node\* temp = head;

            while(temp != nullptr){

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

                temp = temp->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList list1;

    int arr1[] = {1, 0, 2, 1};

    for (int x : arr1) list1.insertAtEnd(x + '0'); *// store as char*

    cout << "List 1: ";

    list1.display();

    cout << (list1.isPalindrome() ? "Linked List is a Palindrome\n"

                                  : "Linked List is NOT a Palindrome\n");

    NodeList list2;

    string word = "BORROWORROB";

    for (char c : word) list2.insertAtEnd(c);

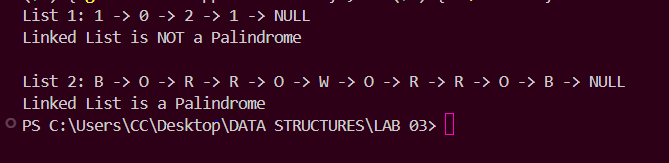
    cout << "\nList 2: ";

    list2.display();

    cout << (list2.isPalindrome() ? "Linked List is a Palindrome\n"

                                  : "Linked List is NOT a Palindrome\n");

}



TASK-06

#include <iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node(int *val*){

        data = *val*;

        next = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(*val*);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                newNode->next = head;

                head = newNode;

            }

        }

        void insertAtEnd(int *val*){

            Node\* newNode = new Node(*val*);

            if(tail == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                tail->next = newNode;

                newNode->next = nullptr;

                tail = newNode;

            }

        }

        bool deleteVal(int *val*){

            bool check = false;

            Node\* ptr = head;

            Node\* prev = nullptr;

            while(ptr != nullptr){

                if(ptr->data == *val*){

                    if(ptr==head){

                        Node\* temp = head;

                        head = head->next;

                        delete temp;

                        ptr = head;

                        check = true;

                    }else{

                        prev->next = ptr->next;

                        if(ptr == tail) tail = prev;

                        Node\* temp = ptr;

                        ptr = prev->next;

                        delete temp;

                        ptr = head;

                        check = true;

                    }

                }

                else{

                    prev = ptr;

                    ptr = ptr->next;

                }

            }

            return check;

        }

        void display(){

            Node\* temp = head;

            while(temp != nullptr){

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

                temp = temp->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList n;

    int arr[] = {2, 4, 6, 8, 1, 3, 5, 6};

    int size = sizeof(arr)/sizeof(arr[0]);

    for(int i=0; i<size; i++) n.insertAtHead(arr[i]);

    cout<<"Linked List before deleting: ";

    n.display();

    int count;

    cout<<"enter how many values you want to delete: ";

    cin>>count;

    for(int i=0; i<count; i++){

        int val;

        cout<<"Enter value "<<i+1<<": ";

        cin>>val;

        if(n.deleteVal(val)){

            cout<<"After deleting "<<val<<": ";

            n.display();

        }else{

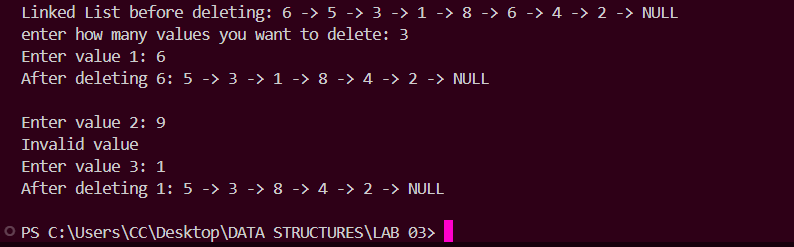
            cout<<"Invalid value";

        }

        cout<<endl;

    }

}



TASK-07

#include <iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node(int *val*){

        data = *val*;

        next = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(*val*);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

                newNode->next = head;

            }else{

                newNode->next = head;

                head = newNode;

                tail->next = head;

            }

        }

        void insertAtEnd(int *val*){

            Node\* newNode = new Node(*val*);

            if(tail == nullptr){

                head = newNode;

                tail = newNode;

                newNode->next = head;

            }else{

                tail->next = newNode;

                newNode->next = head;

                tail = newNode;

            }

        }

        void insertAtPos(int *val*, int *pos*){

            if(*pos* < 0) return;

            if(*pos* == 0){

                insertAtHead(*val*);

                return;

            }

            int count = 0;

            Node \*ptr = head;

            while(ptr->next != head && count < *pos*-1){

                ptr = ptr->next;

                count++;

            }

            if(count != *pos*-1){

                cout<<"Invalid pos"<<endl;

                return;

            }

            Node\* newNode = new Node(*val*);

            newNode->next = ptr->next;

            ptr->next = newNode;

        }

        void deleteVal(int *val*){

            if(head == nullptr) return;

            Node\* ptr = head;

            Node\* prev = nullptr;

            do{

                if(ptr->data == *val*){

                    if(ptr==head){

                        if(head == tail){

                            delete head;

                            head = nullptr;

                            tail = nullptr;

                            return;

                        }

                        head = head->next;

                        tail->next = head;

                        delete ptr;

                        return;

                    }else{

                        prev->next = ptr->next;

                        if(ptr == tail) tail = prev;

                        delete ptr;

                        return;

                    }

                }

                else{

                    prev = ptr;

                    ptr = ptr->next;

                }

            }while(ptr != head);

        }

        void display(){

            Node\* temp = head;

            if(head == nullptr) {

                cout<<"Empty List"<<endl;

                return;

            }

            do{

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

                temp = temp->next;

            }while(temp != head);

        }

};

int main(){

    NodeList n;

    int arr[] = {3, 1, 2, 5, 8};

    for(int i=0; i<5; i++) {

        n.insertAtEnd(arr[i]);

    }

    cout<<"Displaying Linked List before changes: ";

    n.display();

    n.insertAtEnd(9);

    n.insertAtPos(11, 3);

    n.insertAtHead(4);

    n.deleteVal(1);

    n.deleteVal(2);

    n.deleteVal(5);

    cout<<endl<<"Displaying Linked List after changes: ";

    n.display();}



TASK-08

#include <iostream>

using namespace std;

struct Node{

    int data;

    Node\* next;

    Node\* prev;

    Node(int *val*){

        data = *val*;

        next = nullptr;

        prev = nullptr;

    }

};

class NodeList{

    Node\* head;

    Node\* tail;

    public:

        NodeList(){

            head = nullptr;

            tail = nullptr;

        }

        void insertAtHead(int *val*){

            Node \*newNode = new Node(*val*);

            if(head == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                newNode->next = head;

                head->prev = newNode;

                head = newNode;

                head->prev = nullptr;

            }

        }

        void insertAtEnd(int *val*){

            Node\* newNode = new Node(*val*);

            if(tail == nullptr){

                head = newNode;

                tail = newNode;

            }else{

                tail->next = newNode;

                newNode->next = nullptr;

                newNode->prev = tail;

                tail = newNode;

            }

        }

        static NodeList concatenate(NodeList &*n1*, NodeList &*n2*){

            NodeList n;

            if (*n1*.head == nullptr) return *n2*;

            if (*n2*.head == nullptr) return *n1*;

*n1*.tail->next = *n2*.head;

*n2*.head->prev = *n1*.tail;

            n.head = *n1*.head;

            n.tail = *n2*.tail;

            return n;

        }

        void display(){

            Node\* temp = head;

            while(temp != nullptr){

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

                temp = temp->next;

            }

            cout<<"NULL"<<endl;

        }

};

int main(){

    NodeList n1;

    int arr1[] = {3, 1, 2, 5, 8};

    for(int x : arr1)  n1.insertAtEnd(x);

    cout<<"Linked List 1: ";

    n1.display();

    NodeList n2;

    int arr2[] = {2, 4, 6, 8, 1, 3, 5};

    for(int x : arr2) n2.insertAtEnd(x);

    cout<<endl<<"Linked List 2: ";

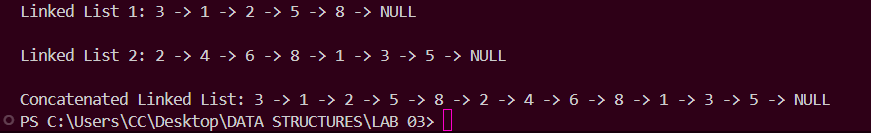
    n2.display();

    NodeList final = NodeList::concatenate(n1, n2);

    cout<<endl<<"Concatenated Linked List: ";

    final.display();

}



TASK-09

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node(int *val*)

    {

        data = *val*;

        next = nullptr;

    }

};

class NodeList

{

    Node \*head;

    Node \*tail;

public:

    NodeList()

    {

        head = nullptr;

        tail = nullptr;

    }

    void insertAtEnd(int *val*)

    {

        Node \*newNode = new Node(*val*);

        if (tail == nullptr)

        {

            head = newNode;

            tail = newNode;

        }

        else

        {

            tail->next = newNode;

            newNode->next = nullptr;

            tail = newNode;

        }

    }

    void performFunc()

    {

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

            return;

*// 1- extracting alternative nodes*

        Node \*ptr = head;

        Node \*tempHead = nullptr;

        Node \*tempTail = nullptr;

        while (ptr != nullptr && ptr->next != nullptr)

        {

            Node \*temp = ptr->next;

            ptr->next = temp->next;

            if (tempHead == nullptr)

            {

                tempHead = temp;

                tempTail = temp;

            }

            else

            {

                tempTail->next = temp;

                tempTail = temp;

            }

            tempTail->next = nullptr;

            if (ptr->next == nullptr)

            {

                tail = ptr;

            }

            ptr = ptr->next;

        }

*// 2- reversing alternative nodes*

        Node \*prev = nullptr;

        Node \*currAlt = tempHead;

        while (currAlt)

        {

            Node \*next = currAlt->next;

            currAlt->next = prev;

            prev = currAlt;

            currAlt = next;

        }

        tempHead = prev;

*// 3- appending reversed list*

        tail->next = tempHead;

        if (tempHead != nullptr)

        {

            Node \*newTail = tempHead;

            while (newTail->next != nullptr)

            {

                newTail = newTail->next;

            }

            tail = newTail;

        }

    }

    void display()

    {

        Node \*temp = head;

        while (temp != nullptr)

        {

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

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

};

int main()

{

    NodeList n;

    int arr[] = {10, 4, 9, 1, 3, 5, 9, 4};

    for (int x : arr)

    {

        n.insertAtEnd(x);

    }

    n.display();

    n.performFunc();

    cout << endl;

    n.display();

}

