24K0600  
DS LAB 3:

Task 1:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int val) {

        data = val;

        next = NULL;

    }

};

class SLinkList {

public:

    Node\* head;

    Node\* tail;

    SLinkList() {

        head = NULL;

        tail = NULL;

    }

    void insertNode(int pos, int val) {

        Node\* n = new Node(val);

        if (pos == 0) {

            n->next = head;

            head = n;

            if (tail == NULL) tail = n;

            return;

        }

        Node\* curr = head;

        Node\* pre = NULL;

        for (int i = 0; i < pos; i++) {

            if (curr == NULL) {

                cout << "Position out of bounds\n";

                delete n;

                return;

            }

            pre = curr;

            curr = curr->next;

        }

        pre->next = n;

        n->next = curr;

        if (n->next == NULL) tail = n;

    }

    void insertAtTail(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            return;

        }

        tail->next = n;

        tail = n;

    }

    void insertAtStart(int val) {

        Node\* n = new Node(val);

        n->next = head;

        head = n;

        if (tail == NULL) tail = n;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void deleteAtEnd() {

        if (head == NULL) return;

        if (head == tail) {

            delete head;

            head = tail = NULL;

            return;

        }

        Node\* temp = head;

        while (temp->next != tail) temp = temp->next;

        delete tail;

        tail = temp;

        tail->next = NULL;

    }

    void deleteAtFront() {

        if (head == NULL) return;

        Node\* temp = head;

        head = head->next;

        if (head == NULL) tail = NULL;

        delete temp;

    }

    void deleteAfter(int pos) {

        if (head == NULL) {

            cout << "List is empty.\n";

            return;

        }

        Node\* curr = head;

        for (int i = 0; i < pos; i++) {

            if (curr == NULL || curr->next == NULL) {

                cout << "Position out of bounds.\n";

                return;

            }

            curr = curr->next;

        }

        Node\* toDelete = curr->next;

        if (toDelete == NULL) {

            cout << "No node to delete after the given position.\n";

            return;

        }

        curr->next = toDelete->next;

        if (toDelete == tail) tail = curr;

        delete toDelete;

    }

};

int main() {

    int arr[5] = {3, 1, 2, 5, 8};

    cout << "Array: ";

    for (int i = 0; i < 5; i++) cout << arr[i] << " ";

    cout << endl;

    SLinkList list;

    for (int i = 0; i < 5; i++) list.insertAtTail(arr[i]);

    cout << "Initial Linked List: ";

    list.display();

    list.insertAtTail(9);

    cout << "After adding 9 at the end: ";

    list.display();

    list.insertNode(3, 11);

    cout << "After adding 11 at position 3: ";

    list.display();

    list.insertAtStart(4);

    cout << "After adding 4 at the front: ";

    list.display();

    list.deleteAfter(1);

    cout << "After deleting 1: ";

    list.display();

    list.deleteAfter(1);

    cout << "After deleting 2: ";

    list.display();

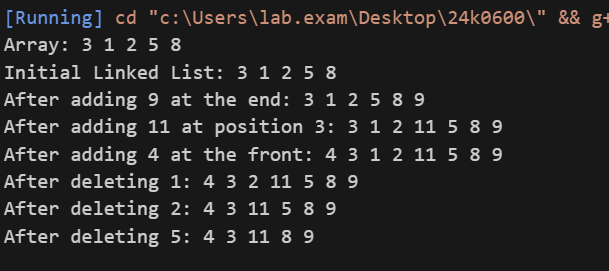
    list.deleteAfter(2);

    cout << "After deleting 5: ";

    list.display();

    return 0;

}



Task 2:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int val) {

        data = val;

        next = NULL;

    }

};

class SLinkList {

public:

    Node\* head;

    Node\* tail;

    SLinkList() {

        head = NULL;

        tail = NULL;

    }

    void insertNode(int pos, int val) {

        Node\* n = new Node(val);

        if (pos == 0) {

            n->next = head;

            head = n;

            if (tail == NULL) tail = n;

            return;

        }

        Node\* curr = head;

        Node\* pre = NULL;

        for (int i = 0; i < pos; i++) {

            if (curr == NULL) {

                cout << "Position out of bounds\n";

                delete n;

                return;

            }

            pre = curr;

            curr = curr->next;

        }

        pre->next = n;

        n->next = curr;

        if (n->next == NULL) tail = n;

    }

    void insertAtTail(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            return;

        }

        tail->next = n;

        tail = n;

    }

    void insertAtStart(int val) {

        Node\* n = new Node(val);

        n->next = head;

        head = n;

        if (tail == NULL) tail = n;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void deleteAtEnd() {

        if (head == NULL) return;

        if (head == tail) {

            delete head;

            head = tail = NULL;

            return;

        }

        Node\* temp = head;

        while (temp->next != tail) temp = temp->next;

        delete tail;

        tail = temp;

        tail->next = NULL;

    }

    void deleteAtFront() {

        if (head == NULL) return;

        Node\* temp = head;

        head = head->next;

        if (head == NULL) tail = NULL;

        delete temp;

    }

    void deleteAfter(int pos) {

        if (head == NULL) {

            cout << "List is empty.\n";

            return;

        }

        Node\* curr = head;

        for (int i = 0; i < pos; i++) {

            if (curr == NULL || curr->next == NULL) {

                cout << "Position out of bounds.\n";

                return;

            }

            curr = curr->next;

        }

        Node\* toDelete = curr->next;

        if (toDelete == NULL) {

            cout << "No node to delete after the given position.\n";

            return;

        }

        curr->next = toDelete->next;

        if (toDelete == tail) tail = curr;

        delete toDelete;

    }

};

int main() {

    int arr[7] = {5, 3, 1, 8, 6, 4, 2};

    cout << "Array: ";

    for (int i = 0; i < 7; i++) cout << arr[i] << " ";

    cout << endl;

    SLinkList list;

    for (int i = 0; i < 7; i++) list.insertAtTail(arr[i]);

    cout << "Initial Linked List: ";

    list.display();

    int n;

    cout<<"Enter no:";

    cin>>n;

    for(int i = 0; i < n; i++) list.insertAtTail(arr[i]);

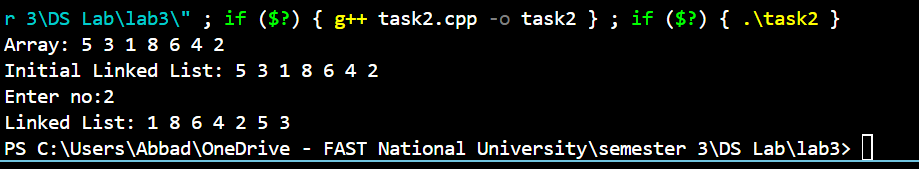
    for(int i = 0; i < n; i++) list.deleteAtFront();

    cout << "Linked List: ";

    list.display();

    return 0;

}



Task 3:

#include <iostream>

#include <string>

using namespace std;

class Node {

public:

    string data;

    Node\* next;

    Node(string val) {

        data = val;

        next = NULL;

    }

};

class SLinkList {

public:

    Node\* head;

    Node\* tail;

    SLinkList() {

        head = NULL;

        tail = NULL;

    }

    void reserveTicket(string val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = tail = n;

            cout << "Ticket reserved for " << val << endl;

            return;

        }

        if (val < head->data) {

            n->next = head;

            head = n;

            cout << "Ticket reserved for " << val << endl;

            return;

        }

        Node\* curr = head;

        Node\* pre = NULL;

        while (curr != NULL && curr->data < val) {

            pre = curr;

            curr = curr->next;

        }

        if (curr != NULL && curr->data == val) {

            cout << "Passenger " << val << " already has a reservation.\n";

            delete n;

            return;

        }

        pre->next = n;

        n->next = curr;

        if (curr == NULL) tail = n;

        cout << "Ticket reserved for " << val << endl;

    }

    void cancelReservation(string val) {

        if (head == NULL) {

            cout << "No reservations to cancel.\n";

            return;

        }

        if (head->data == val) {

            Node\* temp = head;

            head = head->next;

            if (head == NULL) tail = NULL;

            delete temp;

            cout << "Reservation cancelled for " << val << endl;

            return;

        }

        Node\* curr = head;

        Node\* pre = NULL;

        while (curr != NULL && curr->data != val) {

            pre = curr;

            curr = curr->next;

        }

        if (curr == NULL) {

            cout << "No reservation found for " << val << endl;

            return;

        }

        pre->next = curr->next;

        if (curr == tail) tail = pre;

        delete curr;

        cout << "Reservation cancelled for " << val << endl;

    }

    void checkReservation(string val) {

        Node\* temp = head;

        while (temp != NULL) {

            if (temp->data == val) {

                cout << "Yes, " << val << " has a reservation.\n";

                return;

            }

            temp = temp->next;

        }

        cout << "No reservation found for " << val << endl;

    }

    void displayPassengers() {

        if (head == NULL) {

            cout << "No passengers reserved.\n";

            return;

        }

        cout << "Passenger List: ";

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

};

int main() {

    SLinkList flight;

    int choice;

    string name;

    do {

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

        cout << "1. Reserve a ticket\n";

        cout << "2. Cancel a reservation\n";

        cout << "3. Check reservation\n";

        cout << "4. Display all passengers\n";

        cout << "5. Exit\n";

        cout << "Enter choice: ";

        if (!(cin >> choice)) {

            cin.clear();

            cin.ignore(10000, '\n');

            cout << "Invalid input! Please enter a number.\n";

            continue;

        }

        cin.ignore();

        switch (choice) {

        case 1:

            cout << "Enter passenger name: ";

            getline(cin, name);

            flight.reserveTicket(name);

            break;

        case 2:

            cout << "Enter passenger name: ";

            getline(cin, name);

            flight.cancelReservation(name);

            break;

        case 3:

            cout << "Enter passenger name: ";

            getline(cin, name);

            flight.checkReservation(name);

            break;

        case 4:

            flight.displayPassengers();

            break;

        case 5:

            cout << "Exiting program.\n";

            break;

        default:

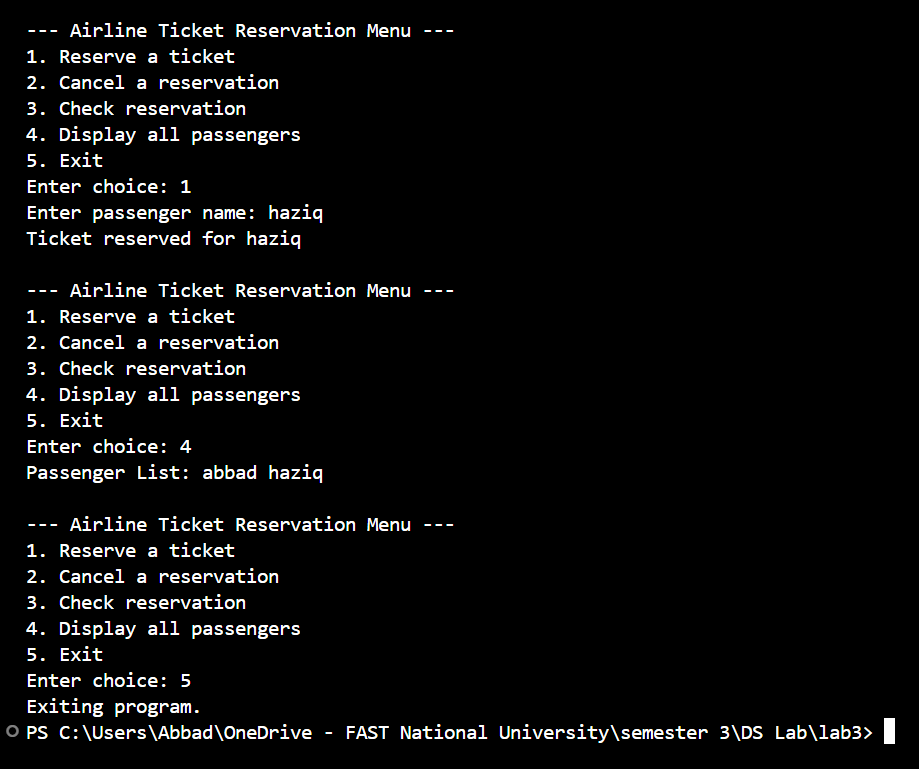
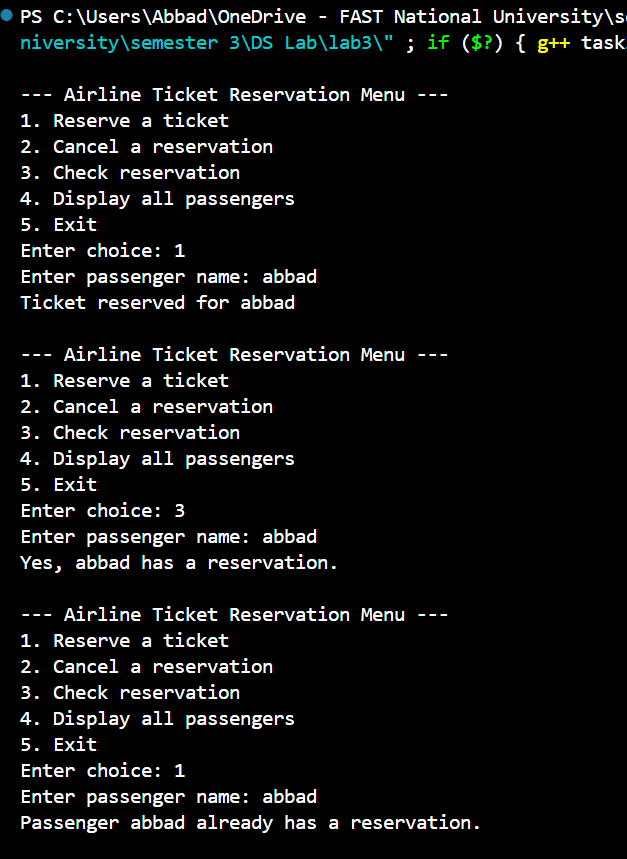
            cout << "Invalid choice.\n";

        }

    } while (choice != 5);

    return 0;

}



Task4:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int val) {

        data = val;

        next = NULL;

    }

};

class SLinkList {

public:

    Node\* head;

    Node\* tail;

    SLinkList() {

        head = NULL;

        tail = NULL;

    }

    void insertNode(int pos, int val) {

        Node\* n = new Node(val);

        if (pos == 0) {

            n->next = head;

            head = n;

            if (tail == NULL) tail = n;

            return;

        }

        Node\* curr = head;

        Node\* pre = NULL;

        for (int i = 0; i < pos; i++) {

            if (curr == NULL) {

                cout << "Position out of bounds\n";

                delete n;

                return;

            }

            pre = curr;

            curr = curr->next;

        }

        pre->next = n;

        n->next = curr;

        if (n->next == NULL) tail = n;

    }

    void insertAtTail(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            return;

        }

        tail->next = n;

        tail = n;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void sortEvenODD(){

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

        Node\* evenHead = NULL;

        Node\* evenTail = NULL;

        Node\* oddHead = NULL;

        Node\* oddTail = NULL;

        Node\* curr = head;

        while (curr != NULL) {

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

                if (evenHead == NULL) {

                    evenHead = curr;

                    evenTail = curr;

                } else {

                    evenTail->next = curr;

                    evenTail = curr;

                }

            } else {

                if (oddHead == NULL) {

                    oddHead = curr;

                    oddTail = curr;

                } else {

                    oddTail->next = curr;

                    oddTail = curr;

                }

            }

            curr = curr->next;

        }

        if (evenHead == NULL || oddHead == NULL) return;

        evenTail->next = oddHead;

        oddTail->next = NULL;

        head = evenHead;

        tail = oddTail;

    }

};

int main() {

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

    cout << "Array: ";

    for (int i = 0; i < 10; i++) cout << arr[i] << " ";

    cout << endl;

    SLinkList list1;

    for (int i = 0; i < 10; i++) list1.insertAtTail(arr[i]);

    cout << "Initial Linked List: ";

    list1.display();

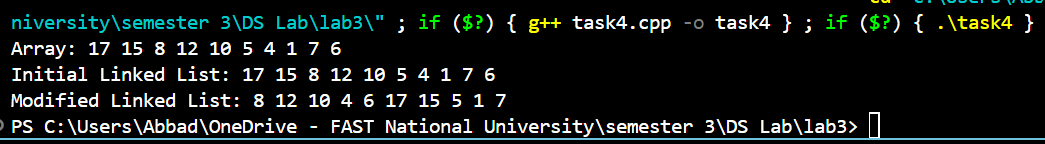
    list1.sortEvenODD();

    cout << "Modified Linked List: ";

    list1.display();

    return 0;

}



Task5:

#include <iostream>

using namespace std;

class Node {

public:

    char data;

    Node\* next;

    Node(char val) {

        data = val;

        next = NULL;

    }

};

class LinkedList {

public:

    Node\* head;

    LinkedList() {

        head = NULL;

    }

    void insert(char val) {

        Node\* newNode = new Node(val);

        if (!head) {

            head = newNode;

            return;

        }

        Node\* temp = head;

        while (temp->next) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    Node\* reverse(Node\* head) {

        Node\* prev = NULL;

        Node\* curr = head;

        Node\* next = NULL;

        while (curr) {

            next = curr->next;

            curr->next = prev;

            prev = curr;

            curr = next;

        }

        return prev;

    }

    bool isPalindrome() {

        if (!head || !head->next) return true;

        Node\* slow = head;

        Node\* fast = head;

        while (fast->next && fast->next->next) {

            slow = slow->next;

            fast = fast->next->next;

        }

        Node\* secondHalf = reverse(slow->next);

        Node\* firstHalf = head;

        Node\* checkHalf = secondHalf;

        bool palindrome = true;

        while (checkHalf) {

            if (firstHalf->data != checkHalf->data) {

                palindrome = false;

                break;

            }

            firstHalf = firstHalf->next;

            checkHalf = checkHalf->next;

        }

        slow->next = reverse(secondHalf);

        return palindrome;

    }

    void display() {

        Node\* temp = head;

        while (temp) {

            cout << temp->data;

            if (temp->next) cout << " -> ";

            temp = temp->next;

        }

        cout << " -> NULL" << endl;

    }

};

int main() {

    LinkedList list1;

    list1.insert('1');

    list1.insert('0');

    list1.insert('2');

    list1.insert('0');

    list1.insert('1');

    cout << "List: ";

    list1.display();

    if (list1.isPalindrome()) {

        cout << "Palindrome\n";

    } else {

        cout << "NOT Palindrome\n";

    }

    LinkedList list2;

    char word[] = {'B','O','R','R','O','W','O','R','R','O','B'};

    for (char c : word) {

        list2.insert(c);

    }

    cout << "List: ";

    list2.display();

    if (list2.isPalindrome()) {

        cout << "Linked List is a Palindrome\n";

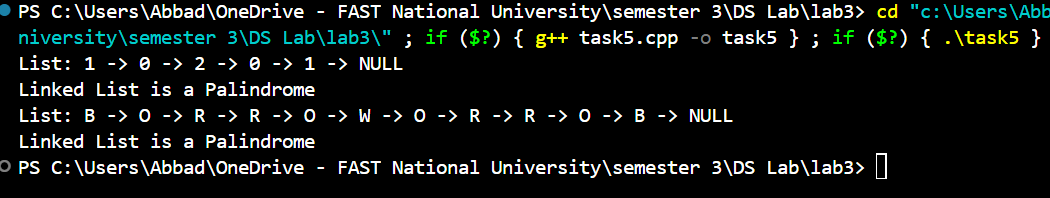
    } else {

        cout << "Linked List is NOT a Palindrome\n";

    }

    return 0;

}



Task6:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int val) {

        data = val;

        next = NULL;

    }

};

class SLinkList {

public:

    Node\* head;

    Node\* tail;

    SLinkList() {

        head = NULL;

        tail = NULL;

    }

    void insertAtTail(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            return;

        }

        tail->next = n;

        tail = n;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void deleteValue(int val) {

        if (head == NULL) {

            cout << "List is empty.\n";

            return;

        }

        Node\* curr = head;

        Node\* prev = NULL;

        while (curr != NULL && curr->data != val) {

            prev = curr;

            curr = curr->next;

        }

        if (curr == NULL) {

            cout << "Value not present in the list.\n";

            return;

        }

        if (curr == head) {

            head = head->next;

            if (curr == tail) tail = NULL;

            delete curr;

            return;

        }

        prev->next = curr->next;

        if (curr == tail) tail = prev;

        delete curr;

    }

};

int main() {

    int arr[7] = {5, 3, 1, 8, 6, 4, 2};

    cout << "Array: ";

    for (int i = 0; i < 7; i++) cout << arr[i] << " ";

    cout << endl;

    SLinkList list;

    for (int i = 0; i < 7; i++) list.insertAtTail(arr[i]);

    cout << "Initial Linked List: ";

    list.display();

    int val;

    cout << "Enter a value to delete: ";

    cin >> val;

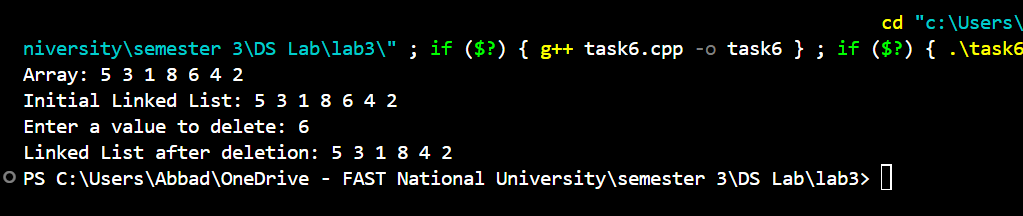
    list.deleteValue(val);

    cout << "Linked List after deletion: ";

    list.display();

    return 0;

}



Task 7:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node() {

        data = 0;

        next = NULL;

    }

    Node(int val) {

        data = val;

        next = NULL;

    }

};

class Circular {

public:

    Node\* head;

    Node\* tail;

    Circular() {

        head = NULL;

        tail = NULL;

    }

    void insertAtEnd(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            tail->next = head;

        } else {

            tail->next = n;

            tail = n;

            tail->next = head;

        }

    }

    void insertAtFront(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            tail->next = head;

        } else {

            n->next = head;

            head = n;

            tail->next = head;

        }

    }

    void insertAtAnyPosition(int pos, int val) {

        if (pos == 0) {

            insertAtFront(val);

            return;

        }

        Node\* n = new Node(val);

        Node\* curr = head;

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

            if (curr->next == head) {

                cout << "Position out of bounds" << endl;

                delete n;

                return;

            }

            curr = curr->next;

        }

        n->next = curr->next;

        curr->next = n;

        if (curr == tail) tail = n;

    }

    void display() {

        if (head == NULL) {

            cout << "List is empty." << endl;

            return;

        }

        Node\* temp = head;

        do {

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

            temp = temp->next;

        } while (temp != head);

        cout << endl;

    }

    void deleteByValue(int val) {

        if (head == NULL) return;

        if (head == tail && head->data == val) {

            delete head;

            head = tail = NULL;

            return;

        }

        if (head->data == val) {

            Node\* temp = head;

            head = head->next;

            tail->next = head;

            delete temp;

            return;

        }

        Node\* curr = head;

        Node\* prev = NULL;

        do {

            prev = curr;

            curr = curr->next;

            if (curr->data == val) {

                prev->next = curr->next;

                if (curr == tail) tail = prev;

                delete curr;

                return;

            }

        } while (curr != head);

        cout << "Value " << val << " not found in the list." << endl;

    }

};

int main() {

    Circular cll;

    cout << "Inserting at end (10, 20):" << endl;

    cll.insertAtEnd(10);

    cll.insertAtEnd(20);

    cll.display();

    cout << "Inserting at front (5):" << endl;

    cll.insertAtFront(5);

    cll.display();

    cout << "Inserting at position 1 (15):" << endl;

    cll.insertAtAnyPosition(1, 15);

    cll.display();

    cout << "Deleting node with value 20:" << endl;

    cll.deleteByValue(20);

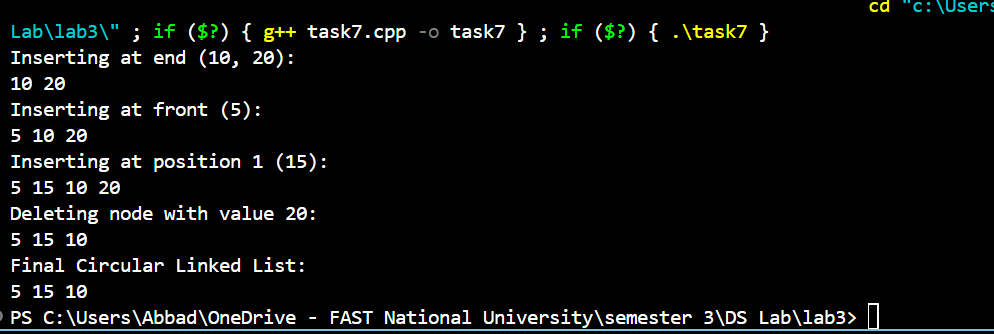
    cll.display();

    cout << "Final Circular Linked List:" << endl;

    cll.display();

    return 0;

}



Task 8:

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node(int val) {

        data = val;

        prev = next = NULL;

    }

};

class DList {

public:

    Node\* head;

    Node\* tail;

    DList() {

        head = tail = NULL;

    }

    void insertAtTail(int val) {

        Node\* n = new Node(val);

        if (head == NULL) {

            head = n;

            tail = n;

            return;

        }

        tail->next = n;

        tail = n;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    static DList concatenate(DList& L, DList& M) {

        if (!L.head) return M;

        if (!M.head) return L;

        L.tail->next = M.head;

        M.head->prev = L.tail;

        L.tail = M.tail;

        return L;

    }

};

int main() {

    DList L, M;

    L.insertAtTail(1);

    L.insertAtTail(2);

    L.insertAtTail(3);

    M.insertAtTail(4);

    M.insertAtTail(5);

    M.insertAtTail(6);

    cout << "List L: "; L.display();

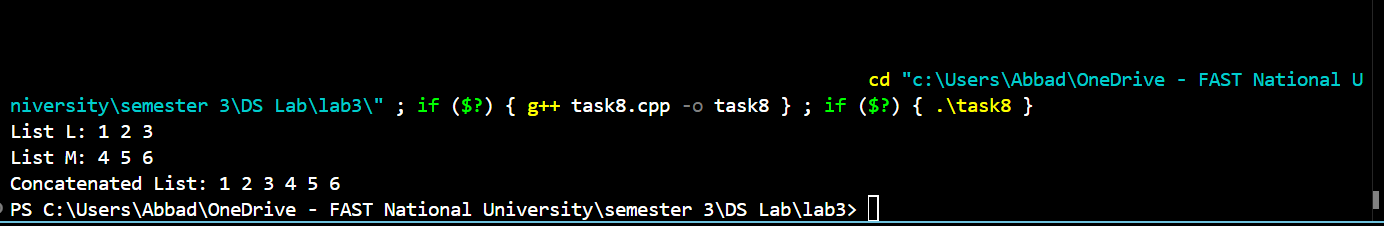
    cout << "List M: "; M.display();

    DList result = DList::concatenate(L, M);

    cout << "Concatenated List: "; result.display();

    return 0;

}



Task9:

#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; }

    void insertAtTail(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 display() {

        Node\* temp = head;

        while (temp != NULL) {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void rearrange() {

        if (!head || !head->next) return;

        Node\* odd = head;

        Node\* even = head->next;

        Node\* evenHead = even;

        while (odd->next && even->next) {

            odd->next = even->next;

            odd = odd->next;

            even->next = odd->next;

            even = even->next;

        }

        odd->next = NULL;

        Node\* prev = NULL;

        Node\* curr = evenHead;

        Node\* next = NULL;

        while (curr) {

            next = curr->next;

            curr->next = prev;

            prev = curr;

            curr = next;

        }

        Node\* reversedEvenHead = prev;

        Node\* temp = head;

        while (temp->next) temp = temp->next;

        temp->next = reversedEvenHead;

    }

};

int main() {

    LinkedList list;

    int arr[] = {10,4,9,1,3,5,9,4};

    for (int x : arr) list.insertAtTail(x);

    cout << "Original List: ";

    list.display();

    list.rearrange();

    cout << "Modified List: ";

    list.display();

    return 0;

}

