1.

#include <iostream>

#include <string>

using namespace std;

class Student {

public:

    string name;

    string rollNo;

    string dept;

    Student(string n, string r, string d)

        : name(n), rollNo(r), dept(d) {}

    void display() {

        cout << name << " | " << rollNo << " | " << dept << endl;

    }

};

class Node {

public:

    Student data;

    Node\* next;

    Node(Student s) : data(s), next(nullptr) {}

};

class StudentList {

    Node\* head;

public:

    StudentList() : head(nullptr) {}

    void add(Student s) {

        Node\* node = new Node(s);

        if (!head) {

            head = node;

            return;

        }

        Node\* temp = head;

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

        temp->next = node;

    }

    void remove(string roll) {

        if (!head) {

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

            return;

        }

        if (head->data.rollNo == roll) {

            Node\* temp = head;

            head = head->next;

            delete temp;

            cout << "Student removed" << endl;

            return;

        }

        Node\* temp = head;

        Node\* prev = nullptr;

        while (temp) {

            if (temp->data.rollNo == roll) {

                prev->next = temp->next;

                delete temp;

                cout << "Student removed" << endl;

                return;

            }

            prev = temp;

            temp = temp->next;

        }

        cout << "Student not found" << endl;

    }

    void displayAll() {

        if (!head) {

            cout << "No students in the list" << endl;

            return;

        }

        Node\* temp = head;

        while (temp) {

            temp->data.display();

            temp = temp->next;

        }

    }

    ~StudentList() {

        Node\* temp = head;

        while (temp) {

            Node\* next = temp->next;

            delete temp;

            temp = next;

        }

    }

};

int main() {

    StudentList students;

    students.add(Student("Ali Khan", "1001", "Computer Science"));

    students.add(Student("Sara Ahmed", "1002", "Mechanical Engineering"));

    students.add(Student("Omar Farooq", "1003", "Economics"));

    cout << "All students:" << endl;

    students.displayAll();

    students.add(Student("Ayesha Malik", "1004", "Law"));

    cout << "\nAfter adding one student:" << endl;

    students.displayAll();

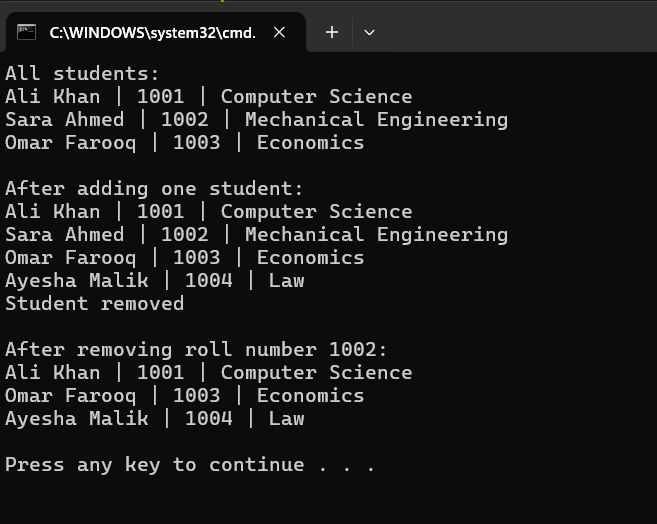
    students.remove("1002");

    cout << "\nAfter removing roll number 1002:" << endl;

    students.displayAll();

    return 0;

}



2.

#include <iostream>

#include <string>

using namespace std;

class BookNode {

public:

    string id, title, author;

    BookNode\* next;

    BookNode(string i, string t, string a) : id(i), title(t), author(a), next(nullptr) {}

    void print() {

        cout << "ID: " << id << ", Title: " << title << ", Author: " << author << endl;

    }

};

class BookList {

private:

    BookNode\* head;

    int count;

public:

    BookList() : head(nullptr), count(0) {}

    void addFirst(BookNode\* book) {

        book->next = head;

        head = book;

        count++;

        cout << "Book added at beginning" << endl;

    }

    void addLast(BookNode\* book) {

        if (!head) {

            head = book;

        } else {

            BookNode\* temp = head;

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

            temp->next = book;

        }

        count++;

        cout << "Book added at end" << endl;

    }

    void addAt(int pos, BookNode\* book) {

        if (pos < 1 || pos > count + 1) {

            cout << "Invalid position (1 to " << count + 1 << ")" << endl;

            delete book;

            return;

        }

        if (pos == 1) {

            addFirst(book);

            return;

        }

        BookNode\* temp = head;

        for (int i = 1; i < pos - 1; i++) temp = temp->next;

        book->next = temp->next;

        temp->next = book;

        count++;

        cout << "Book added at position " << pos << endl;

    }

    void removeFirst() {

        if (!head) {

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

            return;

        }

        BookNode\* temp = head;

        head = head->next;

        delete temp;

        count--;

        cout << "Book removed from beginning" << endl;

    }

    void removeLast() {

        if (!head) {

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

            return;

        }

        if (!head->next) {

            delete head;

            head = nullptr;

        } else {

            BookNode\* temp = head;

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

            delete temp->next;

            temp->next = nullptr;

        }

        count--;

        cout << "Book removed from end" << endl;

    }

    void removeAt(int pos) {

        if (!head) {

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

            return;

        }

        if (pos < 1 || pos > count) {

            cout << "Invalid position (1 to " << count << ")" << endl;

            return;

        }

        if (pos == 1) {

            removeFirst();

            return;

        }

        BookNode\* temp = head;

        BookNode\* prev = nullptr;

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

            prev = temp;

            temp = temp->next;

        }

        prev->next = temp->next;

        delete temp;

        count--;

        cout << "Book removed from position " << pos << endl;

    }

    void search(string id) {

        if (!head) {

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

            return;

        }

        BookNode\* temp = head;

        int pos = 1;

        while (temp) {

            if (temp->id == id) {

                cout << "Book found at position " << pos << ": ";

                temp->print();

                return;

            }

            temp = temp->next;

            pos++;

        }

        cout << "Book with ID " << id << " not found" << endl;

    }

    void showAll() {

        if (!head) {

            cout << "No books available" << endl;

            return;

        }

        cout << "\nBook List" << endl;

        cout << "Total: " << count << endl;

        BookNode\* temp = head;

        int pos = 1;

        while (temp) {

            cout << pos << ". ";

            temp->print();

            temp = temp->next;

            pos++;

        }

    }

    ~BookList() {

        BookNode\* current = head;

        while (current) {

            BookNode\* next = current->next;

            delete current;

            current = next;

        }

    }

};

int main() {

    BookList store;

    cout << "\n1. Adding books..." << endl;

    store.addFirst(new BookNode("101", "ABC", "Ali"));

    store.addLast(new BookNode("102", "DEF", "Ahmed"));

    store.addAt(2, new BookNode("103", "GHI", "Bilal"));

    store.showAll();

    cout << "\n2. Removing books..." << endl;

    store.removeFirst();

    store.showAll();

    store.removeLast();

    store.showAll();

    cout << "\n3. Adding more books..." << endl;

    store.addFirst(new BookNode("104", "JKL", "John"));

    store.addLast(new BookNode("105", "MNO", "Doe"));

    store.addAt(2, new BookNode("106", "PQR", "Joel"));

    store.showAll();

    cout << "\n4. Removing at position..." << endl;

    store.removeAt(2);

    store.showAll();

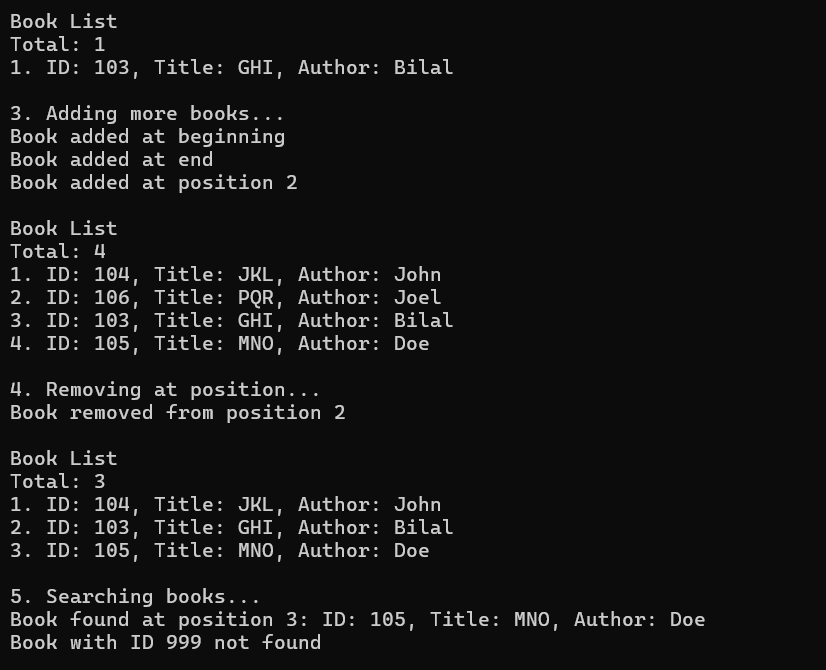
    cout << "\n5. Searching books..." << endl;

    store.search("105");

    store.search("999");

    return 0;

}



3.

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int d) : data(d), next(nullptr) {}

};

class LinkedList {

public:

    Node\* head;

    LinkedList() : head(nullptr) {}

    void pushBack(int d) {

        Node\* n = new Node(d);

        if (!head) {

            head = n;

            return;

        }

        Node\* temp = head;

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

        temp->next = n;

    }

    void display() {

        Node\* temp = head;

        while (temp) {

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

            temp = temp->next;

        }

        cout << "null" << endl;

    }

};

LinkedList\* updateList(LinkedList\* l) {

    LinkedList\* even = new LinkedList();

    LinkedList\* odd = new LinkedList();

    Node\* temp = l->head;

    while (temp) {

        if (temp->data % 2 == 0)

            even->pushBack(temp->data);

        else

            odd->pushBack(temp->data);

        temp = temp->next;

    }

    if (even->head && !odd->head) return even;

    if (odd->head && !even->head) return odd;

    Node\* tail = even->head;

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

    tail->next = odd->head;

    return even;

}

int main() {

    LinkedList\* l = new LinkedList();

    cout << "Enter 6 numbers: ";

    for (int i = 0; i < 6; i++) {

        int n;

        cin >> n;

        l->pushBack(n);

    }

    cout << "\nOriginal list:" << endl;

    l->display();

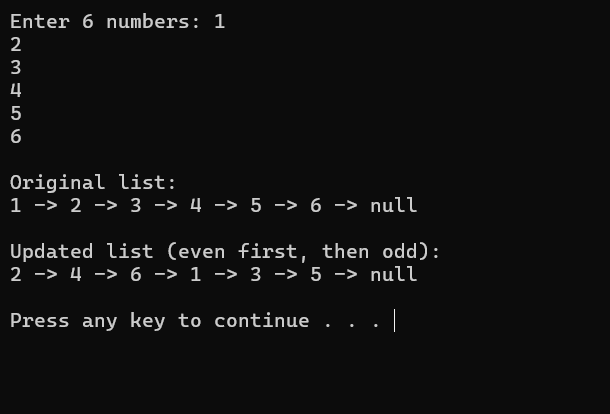
    l = updateList(l);

    cout << "\nUpdated list (even first, then odd):" << endl;

    l->display();

    return 0;

}



4.

#include <iostream>

#include <string>

using namespace std;

class Node {

public:

    char data;

    Node\* next;

    Node(char c) : data(c), next(nullptr) {}

};

class LinkedList {

public:

    Node\* head;

    LinkedList() : head(nullptr) {}

    void pushfront(char c) {

        Node\* n = new Node(c);

        n->next = head;

        head = n;

    }

    void display() {

        Node\* temp = head;

        while (temp) {

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

            temp = temp->next;

        }

        cout << "null" << endl;

    }

};

bool ispal(Node\* head) {

    Node\* slow = head;

    Node\* fast = head;

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

        slow = slow->next;

        fast = fast->next->next;

    }

    LinkedList bottom;

    Node\* temp = slow->next;

    while (temp) {

        bottom.pushfront(temp->data);

        temp = temp->next;

    }

    temp = bottom.head;

    Node\* temp2 = head;

    while (temp && temp2) {

        if (temp2->data != temp->data) return false;

        temp = temp->next;

        temp2 = temp2->next;

    }

    return true;

}

int main() {

    LinkedList test;

    test.pushfront('r');

    test.pushfront('a');

    test.pushfront('c');

    test.pushfront('e');

    test.pushfront('c');

    test.pushfront('a');

    test.pushfront('r');

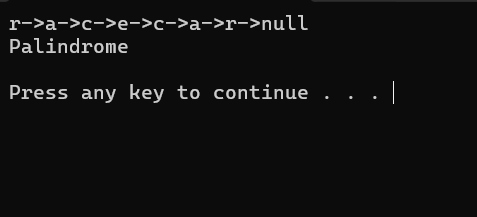
    test.display();

    if (ispal(test.head)) cout << "Palindrome" << endl;

    else cout << "Not Palindrome" << endl;

    return 0;

}



5.

#include <iostream>

#include <string>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node\* prev;

    Node() : data(0), next(nullptr), prev(nullptr) {}

    Node(int d) : data(d), next(nullptr), prev(nullptr) {}

    Node(int d, Node\* n, Node\* p) : data(d), next(n), prev(p) {}

};

class LinkedList {

public:

    Node\* head;

    Node\* tail;

    LinkedList() : head(nullptr), tail(nullptr) {}

    void insertFront(int x) {

        if (!head) {

            head = tail = new Node(x);

        } else {

            Node\* n = new Node(x, head, nullptr);

            head->prev = n;

            head = n;

        }

    }

    void insertRear(int x) {

        if (!head) {

            head = tail = new Node(x);

        } else {

            Node\* n = new Node(x, nullptr, tail);

            tail->next = n;

            tail = n;

        }

    }

    void display() {

        Node\* temp = head;

        while (temp) {

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

            temp = temp->next;

        }

        cout << "null" << endl;

    }

};

int main() {

    LinkedList queue;

    queue.insertFront(10);

    queue.insertFront(20);

    queue.insertRear(30);

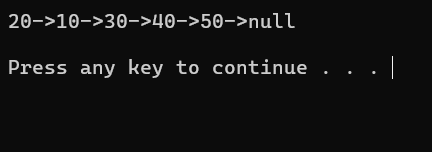
    queue.insertRear(40);

    queue.insertRear(50);

    queue.display();

    return 0;

}



6.

#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node\* next;

    Node(int d) : data(d), next(nullptr) {}

};

class LinkedList {

public:

    Node\* head;

    LinkedList() : head(nullptr) {}

    void pushback(int d) {

        Node\* n = new Node(d);

        if (!head) {

            head = n;

            return;

        }

        Node\* temp = head;

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

        temp->next = n;

    }

    void display() {

        for (Node\* temp = head; temp; temp = temp->next)

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

        cout << "null" << endl;

    }

};

Node\* mergeSorted(Node\* h1, Node\* h2) {

    if (!h1) return h2;

    if (!h2) return h1;

    Node\* head = nullptr;

    Node\* tail = nullptr;

    if (h1->data <= h2->data) {

        head = tail = h1;

        h1 = h1->next;

    } else {

        head = tail = h2;

        h2 = h2->next;

    }

    while (h1 && h2) {

        if (h1->data <= h2->data) {

            tail->next = h1;

            tail = h1;

            h1 = h1->next;

        } else {

            tail->next = h2;

            tail = h2;

            h2 = h2->next;

        }

    }

    if (h1) tail->next = h1;

    if (h2) tail->next = h2;

    return head;

}

int main() {

    LinkedList l1;

    l1.pushback(1);

    l1.pushback(3);

    l1.pushback(5);

    LinkedList l2;

    l2.pushback(2);

    l2.pushback(4);

    l2.pushback(6);

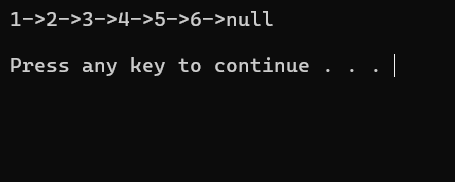
    LinkedList merged;

    merged.head = mergeSorted(l1.head, l2.head);

    merged.display();

    return 0;

}



7.

#include <iostream>

using namespace std;

class SinglyLinkedListNode {

public:

    int data;

    SinglyLinkedListNode\* next;

    SinglyLinkedListNode(int val) : data(val), next(nullptr) {}

};

class DoublyLinkedListNode {

public:

    int data;

    DoublyLinkedListNode\* next;

    DoublyLinkedListNode\* prev;

    DoublyLinkedListNode(int val) : data(val), next(nullptr), prev(nullptr) {}

};

void printSinglyLinkedList(SinglyLinkedListNode\* head) {

    for (SinglyLinkedListNode\* cur = head; cur; cur = cur->next)

        cout << cur->data << " -> ";

    cout << "null" << endl;

}

void printDoublyLinkedList(DoublyLinkedListNode\* head) {

    for (DoublyLinkedListNode\* cur = head; cur; cur = cur->next) {

        int prevData = cur->prev ? cur->prev->data : -1;

        int nextData = cur->next ? cur->next->data : -1;

        cout << "Prev: " << prevData << " <- " << cur->data << " -> Next: " << nextData << endl;

    }

}

void printCircularLinkedList(SinglyLinkedListNode\* head) {

    if (!head) { cout << "List is empty." << endl; return; }

    SinglyLinkedListNode\* cur = head;

    cout << cur->data;

    cur = cur->next;

    while (cur != head) {

        cout << " -> " << cur->data;

        cur = cur->next;

    }

    cout << " -> (back to Head: " << head->data << ")" << endl;

}

DoublyLinkedListNode\* convertSinglyToDoubly(SinglyLinkedListNode\* head) {

    if (!head) return nullptr;

    DoublyLinkedListNode\* dHead = new DoublyLinkedListNode(head->data);

    DoublyLinkedListNode\* dCur = dHead;

    for (SinglyLinkedListNode\* sCur = head->next; sCur; sCur = sCur->next) {

        DoublyLinkedListNode\* newNode = new DoublyLinkedListNode(sCur->data);

        dCur->next = newNode;

        newNode->prev = dCur;

        dCur = newNode;

    }

    return dHead;

}

SinglyLinkedListNode\* convertSinglyToCircular(SinglyLinkedListNode\* head) {

    if (!head) return nullptr;

    SinglyLinkedListNode\* cur = head;

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

    cur->next = head;

    return head;

}

int main() {

    SinglyLinkedListNode\* singlyHead = new SinglyLinkedListNode(10);

    singlyHead->next = new SinglyLinkedListNode(20);

    singlyHead->next->next = new SinglyLinkedListNode(30);

    singlyHead->next->next->next = new SinglyLinkedListNode(40);

    cout << "Original Singly Linked List:" << endl;

    printSinglyLinkedList(singlyHead);

    DoublyLinkedListNode\* doublyHead = convertSinglyToDoubly(singlyHead);

    cout << "\nConverted to Doubly Linked List:" << endl;

    printDoublyLinkedList(doublyHead);

    SinglyLinkedListNode\* circularHead = convertSinglyToCircular(singlyHead);

    cout << "\nConverted to Circular Linked List:" << endl;

    printCircularLinkedList(circularHead);

    return 0;

}

