// Data Structure Assignment - III (C++ Solutions)

#include <iostream>

using namespace std;

// Q1: Singly Linked List

class SNode {

public:

int data;

SNode\* next;

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

};

class SinglyLinkedList {

SNode\* head;

public:

SinglyLinkedList() : head(nullptr) {}

void insert(int val) {

SNode\* newNode = new SNode(val);

if (!head) head = newNode;

else {

SNode\* temp = head;

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

temp->next = newNode;

}

}

void deleteNode(int val) {

if (!head) return;

if (head->data == val) {

SNode\* toDelete = head;

head = head->next;

delete toDelete;

return;

}

SNode\* temp = head;

while (temp->next && temp->next->data != val)

temp = temp->next;

if (temp->next) {

SNode\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

}

}

void traverse() {

SNode\* temp = head;

while (temp) {

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

temp = temp->next;

}

cout << "NULL\n";

}

};

// Q2: Doubly Linked List

class DNode {

public:

int data;

DNode\* prev;

DNode\* next;

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

};

class DoublyLinkedList {

DNode\* head;

public:

DoublyLinkedList() : head(nullptr) {}

void insert(int val) {

DNode\* newNode = new DNode(val);

if (!head) head = newNode;

else {

DNode\* temp = head;

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

temp->next = newNode;

newNode->prev = temp;

}

}

void deleteNode(int val) {

DNode\* temp = head;

while (temp && temp->data != val) temp = temp->next;

if (!temp) return;

if (temp->prev) temp->prev->next = temp->next;

else head = temp->next;

if (temp->next) temp->next->prev = temp->prev;

delete temp;

}

void display() {

DNode\* temp = head;

while (temp) {

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

temp = temp->next;

}

cout << "NULL\n";

}

};

// Q3: Circular Linked List

class CNode {

public:

int data;

CNode\* next;

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

};

class CircularLinkedList {

CNode\* head;

public:

CircularLinkedList() : head(nullptr) {}

void insert(int val) {

CNode\* newNode = new CNode(val);

if (!head) {

head = newNode;

newNode->next = head;

} else {

CNode\* temp = head;

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

temp->next = newNode;

newNode->next = head;

}

}

void display() {

if (!head) return;

CNode\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << "(back to head)\n";

}

};

// Q4: Stack using array

class Stack {

int arr[100], top;

public:

Stack() { top = -1; }

void push(int x) { if (top < 99) arr[++top] = x; }

void pop() { if (top >= 0) top--; }

int peek() { return (top >= 0) ? arr[top] : -1; }

void display() {

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

cout << endl;

}

};

// Q5: Queue using array

class Queue {

int arr[100], front, rear;

public:

Queue() { front = rear = -1; }

void enqueue(int x) {

if (rear < 99) {

if (front == -1) front = 0;

arr[++rear] = x;

}

}

void dequeue() {

if (front != -1 && front <= rear) front++;

}

void display() {

for (int i = front; i <= rear; i++) cout << arr[i] << " ";

cout << endl;

}

};

int main() {

cout << "Singly Linked List:\n";

SinglyLinkedList sll;

sll.insert(10); sll.insert(20); sll.insert(30);

sll.traverse();

sll.deleteNode(20);

sll.traverse();

cout << "\nDoubly Linked List:\n";

DoublyLinkedList dll;

dll.insert(1); dll.insert(2); dll.insert(3);

dll.display();

dll.deleteNode(2);

dll.display();

cout << "\nCircular Linked List:\n";

CircularLinkedList cll;

cll.insert(5); cll.insert(10); cll.insert(15);

cll.display();

cout << "\nStack:\n";

Stack st;

st.push(1); st.push(2); st.push(3);

st.display();

st.pop();

st.display();

cout << "\nQueue:\n";

Queue q;

q.enqueue(10); q.enqueue(20); q.enqueue(30);

q.display();

q.dequeue();

q.display();

return 0;

}