# KASHAN BAIG

# CT-24178

# SEC - D

**Q1.**

#include <iostream>

using namespace std;

class node{

    public:

    int data;

    node\* next;

    node(int *val*){

        data = *val*;

        next = nullptr;

    }

};

class circularqueue{

    public:

    node\* head;

    node\* tail;

    circularqueue(){

        head = tail =nullptr;

    }

     void enqueue(int *val*) {

        node\* newnode = new node(*val*);

        if (head == nullptr) {

            head = tail = newnode;

            tail->next = head;

        } else {

            tail->next = newnode;

            tail = newnode;

            tail->next = head;

        }

        cout << *val* << " enqueued\n";

    }

    void dequeue() {

        if (head == nullptr) {

            cout << "Queue is empty\n";

            return;

        }

        if (head == tail) {

            cout << head->data << " dequeued\n";

            delete head;

            head = tail = nullptr;

        } else {

            node\* temp = head;

            cout << head->data << " dequeued\n";

            head = head->next;

            tail->next = head;

            delete temp;

        }

    }

    int peek() {

        if (head == nullptr) {

            cout << "Queue is empty\n";

            return -1;

        }

        return head->data;

    }

    void display(){

        node\* temp = head;

        cout << "QUEUE\n";

        do {

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

            temp = temp->next;

        } while (temp != head);

        cout <<endl;

    }

};

    int main(){

        circularqueue q;

        q.enqueue(10);

        q.enqueue(20);

        q.display();

        q.dequeue();

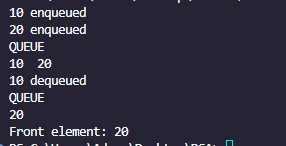
        q.display();

        cout << "Front element: " << q.peek() << endl;

        return 0;

    }

**OUTPUT:**

****

**Q2.**

class node{

    public:

    int data;

    node\* next;

    node\* prev;

    node(int *val*){

        data = *val*;

        next = prev =  nullptr;

    }

};

class double\_ended\_queue{

    public:

    node\* head;

    node\* tail;

    double\_ended\_queue(){

        head = tail = nullptr;

    }

    void enque\_front(int *val*){

        node\* newnode = new node(*val*);

        if(head==nullptr){

            head = tail = newnode;

        }

        else{

            newnode->next = head;

            head->prev = newnode;

            head = newnode;

        }

    }

        void enque\_back(int *val*){

        node\* newnode = new node(*val*);

        if(head==nullptr){

            head = tail = newnode;

        }

        else{

            tail->next = newnode;

            newnode->prev = tail;

            tail = newnode;

        }

    }

    void dequeue\_front(){

        if(head == nullptr){

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

        return;

        }

        node\* temp = head;

        head = head->next;

        if(head != nullptr){

        head->prev = nullptr;

        }

        else{

        tail = nullptr;

        }

        delete temp;

    }

    void dequeue\_back(){

        if(head == nullptr){

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

        return;

        }

        if(head == tail){

        delete head;

        head = tail = nullptr;

        return;

        }

        node\* temp = tail;

        tail = tail->prev;

        tail->next = nullptr;

        delete temp;

    }

    int peek() {

        if (head == nullptr) {

            cout << "Queue is empty\n";

            return -1;

        }

        return head->data;

    }

    void display(){

        node\* temp = head;

        cout<<"NULL<->";

        while(temp!=nullptr){

            cout<<temp->data<<"<->";

            temp = temp->next;

        }

        cout<<"NULL"<<endl;

    }

};

    int main(){

*// Q2*

        double\_ended\_queue dq;

        dq.enque\_front(10);

        dq.enque\_back(20);

        dq.enque\_front(5);

        dq.enque\_back(30);

        dq.display();

        cout << "Front element: " << dq.peek() << endl;

        dq.dequeue\_front();

        dq.display();

        dq.dequeue\_back();

        dq.display();

**OUTPUT:**

****

**Q3.**

class node{

    public:

    int data;

    node\* next;

    node\* prev;

    node(int *val*){

        data = *val*;

        next = prev =  nullptr;

    }

};

class doubly\_list{

    public:

    node\* head;

    node\* tail;

    doubly\_list(){

        head = tail = nullptr;

    }

    void insert\_front(int *val*){

        node\* newnode = new node(*val*);

        if(head==nullptr){

            head = tail = newnode;

        }

        else{

            newnode->next = head;

            head->prev = newnode;

            head = newnode;

        }

    }

    void insert\_back(int *val*){

        node\* newnode = new node(*val*);

        if(head==nullptr){

            head = tail = newnode;

        }

        else{

            tail->next = newnode;

            newnode->prev = tail;

            tail = newnode;

        }

    }

    void pop\_front(){

        if(head == nullptr){

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

        return;

        }

        node\* temp = head;

        head = head->next;

        if(head != nullptr){

        head->prev = nullptr;

        }

        else{

        tail = nullptr;

        }

        delete temp;

    }

    void pop\_back(){

        if(head == nullptr){

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

        return;

        }

        if(head == tail){

        delete head;

        head = tail = nullptr;

        return;

        }

        node\* temp = tail;

        tail = tail->prev;

        tail->next = nullptr;

        delete temp;

    }

    void display(){

        node\* temp = head;

        cout<<"NULL<->";

        while(temp!=nullptr){

            cout<<temp->data<<"<->";

            temp = temp->next;

        }

        cout<<"NULL"<<endl;

    }

    void concatenate(doubly\_list& *other*) {

        if (*other*.head == nullptr) return;

        if (head == nullptr) {

            head = *other*.head;

            tail = *other*.tail;

        } else {

            tail->next = *other*.head;

*other*.head->prev = tail;

            tail = *other*.tail;

        }

    }

};

    int main(){

*// Q3*

         doubly\_list L, M, N;

         for (int i = 2; i <= 10; i += 2) {

        L.insert\_back(i);

    }

    for (int i = 1; i <= 9; i += 2) {

        M.insert\_back(i);

    }

    cout << "List L (Even): ";

    L.display();

    cout << "List M (Odd): ";

    M.display();

    N = L;

    N.concatenate(M);

    cout << "List N (Concatenation of L and M): ";

    N.display();

**OUTPUT:**

****

**Q4.**

 void sortDescending() {

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

        bool swapped;

        Node\* curr;

        Node\* lastSorted = nullptr;

        do {

            swapped = false;

            curr = head;

            while (curr->next != lastSorted) {

                if (curr->data < curr->next->data) { *// swap if smaller*

                    int temp = curr->data;

                    curr->data = curr->next->data;

                    curr->next->data = temp;

                    swapped = true;

                }

                curr = curr->next;

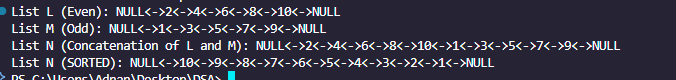
            }

            lastSorted = curr;

        } while (swapped);

    }

**OUTPUT:**

****

**Q5.**

class BrowserHistory {

public:

    vector<string> history;

    int curr;

    int last;

    BrowserHistory(string *homepage*) {

        history.push\_back(*homepage*);

        curr = 0;

        last = 0;

    }

    void visit(string *url*) {

        if (curr + 1 < history.size()) {

            history[curr + 1] = *url*;

        } else {

            history.push\_back(*url*);

        }

        curr++;

        last = curr;

    }

    string back(int *steps*) {

        curr = max(0, curr - *steps*);

        return history[curr];

    }

    string forward(int *steps*) {

        curr = min(last, curr + *steps*);

        return history[curr];

    }

};

    int main(){

*// Q4*

    BrowserHistory browserHistory("leetcode.com");

    browserHistory.visit("google.com");

    browserHistory.visit("facebook.com");

    browserHistory.visit("youtube.com");

    cout << browserHistory.back(1) << endl;

    cout << browserHistory.back(1) << endl;

    cout << browserHistory.forward(1) << endl;

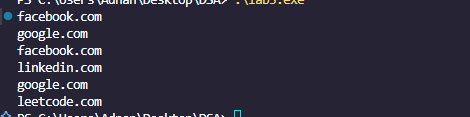
    browserHistory.visit("linkedin.com");

    cout << browserHistory.forward(2) << endl;

    cout << browserHistory.back(2) << endl;

    cout << browserHistory.back(7) << endl;

**OUTPUT:**

****