DSA Assignment 4

1.)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

// Head pointer for list

Node\* head = NULL;

// Function to create a new node

Node\* createNode(int val) {

Node\* newNode = new Node();

newNode->data = val;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(int val) {

Node\* newNode = createNode(val);

newNode->next = head;

head = newNode;

}

void insertAtEnd(int val) {

Node\* newNode = createNode(val);

if (head == NULL) {

head = newNode;

return;

}

Node\* temp = head;

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

temp->next = newNode;

}

void insertBefore(int key, int val) {

if (head == NULL) return;

if (head->data == key) {

insertAtBeginning(val);

return;

}

Node\* temp = head;

while (temp->next != NULL && temp->next->data != key)

temp = temp->next;

if (temp->next == NULL) {

cout << "Key not found\n";

return;

}

Node\* newNode = createNode(val);

newNode->next = temp->next;

temp->next = newNode;

}

void insertAfter(int key, int val) {

Node\* temp = head;

while (temp != NULL && temp->data != key)

temp = temp->next;

if (temp == NULL) {

cout << "Key not found\n";

return;

}

Node\* newNode = createNode(val);

newNode->next = temp->next;

temp->next = newNode;

}

void deleteFromBeginning() {

if (head == NULL) return;

Node\* temp = head;

head = head->next;

delete temp;

}

void deleteFromEnd() {

if (head == NULL) return;

if (head->next == NULL) {

delete head;

head = NULL;

return;

}

Node\* temp = head;

while (temp->next->next != NULL)

temp = temp->next;

delete temp->next;

temp->next = NULL;

}

void deleteNode(int key) {

if (head == NULL) return;

if (head->data == key) {

deleteFromBeginning();

return;

}

Node\* temp = head;

while (temp->next != NULL && temp->next->data != key)

temp = temp->next;

if (temp->next == NULL) {

cout << "Node not found\n";

return;

}

Node\* del = temp->next;

temp->next = del->next;

delete del;

}

void searchNode(int key) {

Node\* temp = head;

int pos = 1;

while (temp != NULL) {

if (temp->data == key) {

cout << "Found at position: " << pos << endl;

return;

}

temp = temp->next;

pos++;

}

cout << "Not found\n";

}

void displayList() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << "NULL\n";

}

int main() {

int choice, val, key;

do {

cout << "\nMenu:\n1.Insert at Beginning\n2.Insert at End\n3.Insert Before\n4.Insert After\n"

<< "5.Delete from Beginning\n6.Delete from End\n7.Delete Node\n8.Search\n9.Display\n0.Exit\nEnter choice: ";

cin >> choice;

switch (choice) {

case 1: cout << "Enter value: "; cin >> val; insertAtBeginning(val); break;

case 2: cout << "Enter value: "; cin >> val; insertAtEnd(val); break;

case 3: cout << "Enter key and value: "; cin >> key >> val; insertBefore(key, val); break;

case 4: cout << "Enter key and value: "; cin >> key >> val; insertAfter(key, val); break;

case 5: deleteFromBeginning(); break;

case 6: deleteFromEnd(); break;

case 7: cout << "Enter key: "; cin >> key; deleteNode(key); break;

case 8: cout << "Enter key: "; cin >> key; searchNode(key); break;

case 9: displayList(); break;

case 0: cout << "Exiting...\n"; break;

default: cout << "Invalid choice\n";

}

} while (choice != 0);

return 0;

}

2.)

#include <iostream>

#define MAX 5

using namespace std;

class CircularQueue {

int arr[MAX];

int front, rear;

public:

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

bool isEmpty() { return front == -1; }

bool isFull() { return (rear + 1) % MAX == front; }

void enqueue(int val) {

if (isFull()) {

cout << "Queue is Full\n"; return;

}

if (isEmpty()) front = rear = 0;

else rear = (rear + 1) % MAX;

arr[rear] = val;

}

void dequeue() {

if (isEmpty()) {

cout << "Queue is Empty\n"; return;

}

cout << "Dequeued: " << arr[front] << endl;

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

else front = (front + 1) % MAX;

}

void display() {

if (isEmpty()) { cout << "Queue is Empty\n"; return; }

cout << "Circular Queue: ";

int i = front;

while (true) {

cout << arr[i] << " ";

if (i == rear) break;

i = (i + 1) % MAX;

}

cout << endl;

}

void peek() {

if (isEmpty()) cout << "Queue is Empty\n";

else cout << "Front Element: " << arr[front] << endl;

}

};

int main() {

CircularQueue q;

int choice, value;

do {

cout << "\n1. Enqueue\n2. Dequeue\n3. Display\n4. Peek\n5. Exit\nEnter choice: ";

cin >> choice;

switch (choice) {

case 1: cout << "Enter value: "; cin >> value; q.enqueue(value); break;

case 2: q.dequeue(); break;

case 3: q.display(); break;

case 4: q.peek(); break;

case 5: cout << "Exiting...\n"; break;

default: cout << "Invalid choice\n";

}

} while (choice != 5);

return 0;

}

3.)

#include <iostream>

#include <queue>

#include <stack>

using namespace std;

void interleave(queue<int> &q) {

if (q.size() % 2 != 0) {

cout << "Queue size must be even.\n";

return;

}

int half = q.size() / 2;

queue<int> firstHalf;

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

firstHalf.push(q.front());

q.pop();

}

while (!firstHalf.empty()) {

q.push(firstHalf.front());

firstHalf.pop();

q.push(q.front());

q.pop();

}

}

int main() {

queue<int> q;

int n, val;

cout << "Enter number of elements: ";

cin >> n;

cout << "Enter elements: ";

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

cin >> val;

q.push(val);

}

interleave(q);

cout << "Interleaved Queue: ";

while (!q.empty()) {

cout << q.front() << " ";

q.pop();

}

cout << endl;

return 0;

}

4.)

#include <iostream>

#include <queue>

#include <unordered\_map>

using namespace std;

int main() {

string s;

cout << "Enter string: ";

cin >> s;

queue<char> q;

unordered\_map<char,int> freq;

for (char c : s) {

freq[c]++;

q.push(c);

while (!q.empty() && freq[q.front()] > 1)

q.pop();

if (q.empty()) cout << -1 << " ";

else cout << q.front() << " ";

}

cout << endl;

return 0;

}

5.)a.)

#include <iostream>

#include <queue>

using namespace std;

class StackTwoQ {

queue<int> q1, q2;

public:

void push(int x) {

q2.push(x);

while (!q1.empty()) {

q2.push(q1.front());

q1.pop();

}

swap(q1, q2);

}

void pop() {

if (q1.empty()) { cout << "Stack is Empty\n"; return; }

cout << "Popped: " << q1.front() << endl;

q1.pop();

}

void top() {

if (q1.empty()) { cout << "Stack is Empty\n"; return; }

cout << "Top: " << q1.front() << endl;

}

};

int main() {

StackTwoQ s;

s.push(10);

s.push(20);

s.top();

s.pop();

s.top();

return 0;

}

5.)b.)

#include <iostream>

#include <queue>

using namespace std;

class StackOneQ {

queue<int> q;

public:

void push(int x) {

int size = q.size();

q.push(x);

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

q.push(q.front());

q.pop();

}

}

void pop() {

if (q.empty()) { cout << "Stack is Empty\n"; return; }

cout << "Popped: " << q.front() << endl;

q.pop();

}

void top() {

if (q.empty()) { cout << "Stack is Empty\n"; return; }

cout << "Top: " << q.front() << endl;

}

};

int main() {

StackOneQ s;

s.push(5);

s.push(15);

s.top();

s.pop();

s.top();

return 0;

}