**Starter Code:**

// Queue ADT using Array

#include<iostream>

using namespace std;

class CircularQueue {

private:

int\* queueArray;

int queueSize;

int front;

int rear;

int count;

public:

CircularQueue(int size = 10) {

queueSize = size;

queueArray = new int[queueSize];

front = 0;

rear = -1;

count = 0;

}

~CircularQueue() {

delete[] queueArray;

}

bool isEmpty() {

return (count == 0);

}

bool isFull() {

return (count == queueSize);

}

bool peek(int& element) {

if (isEmpty()) {

cout << "Queue is Empty." << endl;

return false;

}

element = queueArray[front];

return true;

}

void enqueue(int value) {

if (isFull()) {

cout << "Queue is Full." << endl;

return;

}

rear = (rear + 1) % queueSize;

queueArray[rear] = value;

count++;

cout << value << " enqueued." << endl;

}

bool dequeue(int& element) {

if (isEmpty()) {

cout << "Queue is Empty." << endl;

return false;

}

element = queueArray[front];

front = (front + 1) % queueSize;

count--;

return true;

}

void makeNull() {

front = 0;

rear = -1;

count = 0;

cout << "Now Queue is Empty." << endl;

}

int size() {

return count;

}

};

int main() {

CircularQueue q(5);

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(40);

q.enqueue(50);

int element;

q.peek(element);

cout << "Front Element: " << element << endl;

q.dequeue(element);

cout << "Dequeued Element: " << element << endl;

q.peek(element);

cout << "Front Element: " << element << endl;

q.enqueue(60);

q.peek(element);

cout << "Front Element: " << element << endl;

q.makeNull();

q.peek(element);

return 0;

}

// Queue ADT using LinkedList

#include<iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class Queue {

private:

Node\* front;

Node\* rear;

int numOfItems;

public:

Queue() {

front = nullptr;

rear = nullptr;

numOfItems = 0;

}

~Queue() {

makeNull();

}

bool isEmpty() {

return numOfItems == 0;

}

void enqueue(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (isEmpty()) {

front = newNode;

rear = newNode;

}

else {

rear->next = newNode;

rear = newNode;

}

numOfItems++;

cout << value << " enqueued." << endl;

}

bool dequeue(int& value) {

if (isEmpty()) {

cout << "Queue is Empty." << endl;

return false;

}

Node\* temp = front;

value = front->data;

front = front->next;

delete temp;

numOfItems--;

if (numOfItems == 0) {

rear = nullptr;

}

return true;

}

bool peek(int& value) {

if (isEmpty()) {

cout << "Queue is Empty. " << endl;

return false;

}

value = front->data;

return true;

}

void makeNull() {

while (isEmpty()) {

int val;

dequeue(val);

}

cout << "Queue is Now Empty" << endl;

}

};

int main() {

Queue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(40);

q.enqueue(50);

int element;

q.peek(element);

cout << "Front Element: " << element << endl;

q.dequeue(element);

cout << "Dequeued Element: " << element << endl;

q.peek(element);

cout << "Front Element: " << element << endl;

q.enqueue(60);

q.peek(element);

cout << "Front Element: " << element << endl;

q.makeNull();

q.peek(element);

return 0;

}

**TASK 1:**

#include <iostream>

#include <stack>

using namespace std;

class Queue {

private:

stack<int> stack1;

stack<int> stack2;

public:

void enqueue(int x) {

stack1.push(x);

}

int dequeue() {

if (isEmpty()) {

cout << "Queue is empty" << endl;

return -1;

}

if (stack2.empty()) {

while (!stack1.empty()) {

stack2.push(stack1.top());

stack1.pop();

}

}

int x = stack2.top();

stack2.pop();

return x;

}

int peek() {

if (isEmpty()) {

cout << "Queue is empty" << endl;

return -1;

}

if (stack2.empty()) {

while (!stack1.empty()) {

stack2.push(stack1.top());

stack1.pop();

}

}

return stack2.top();

}

bool isEmpty() {

return (stack1.empty() && stack2.empty());

}

};

int main() {

Queue q;

cout << boolalpha;

cout << q.isEmpty() << endl;

q.enqueue(20);

cout << (q.peek() == 20) << endl;

q.enqueue(30);

q.enqueue(50);

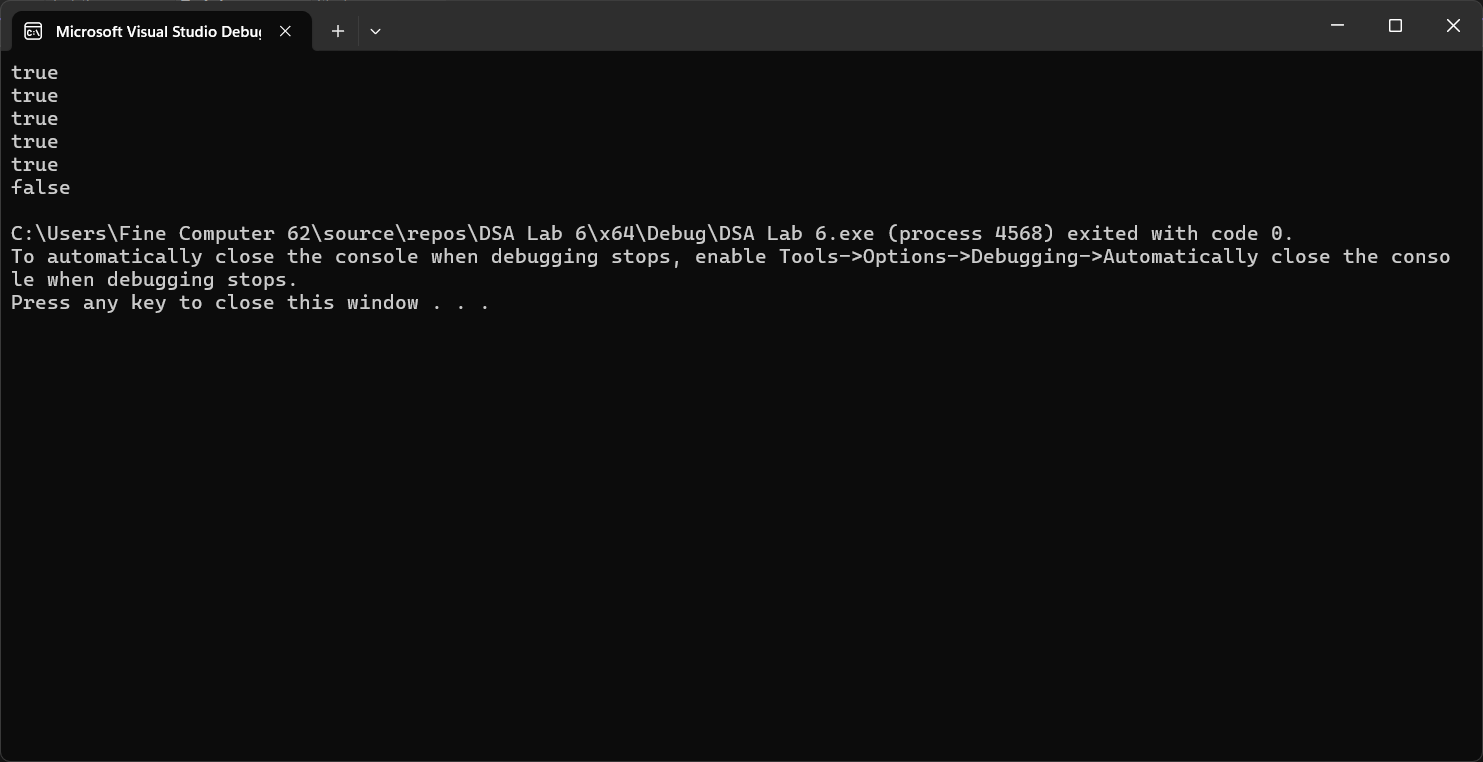
cout << (q.peek() == 20) << endl;

cout << (q.dequeue() == 20) << endl;

cout << (q.peek() == 30) << endl;

cout << q.isEmpty() << endl;

return 0;

} 

**TASK 2:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class Queue {

private:

Node\* front;

Node\* rear;

int numOfItems;

public:

Queue() : front(nullptr), rear(nullptr), numOfItems(0) {}

~Queue() {

while (!isEmpty()) {

int temp;

dequeue(temp);

}

}

bool isEmpty() {

return numOfItems == 0;

}

void enqueue(int value) {

Node\* newNode = new Node{ value, nullptr };

if (isEmpty()) {

front = rear = newNode;

}

else {

rear->next = newNode;

rear = newNode;

}

numOfItems++;

}

bool dequeue(int& value) {

if (isEmpty()) {

return false;

}

Node\* temp = front;

value = front->data;

front = front->next;

delete temp;

numOfItems--;

if (isEmpty()) {

rear = nullptr;

}

return true;

}

bool peek(int& value) {

if (isEmpty()) {

return false;

}

value = front->data;

return true;

}

};

Queue maxSumSlidingWindow(Queue& nums, int k) {

Queue result;

Queue window;

int windowSum = 0;

int maxSum = 0;

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

int val;

if (nums.dequeue(val)) {

window.enqueue(val);

windowSum += val;

}

else {

while (!window.isEmpty()) {

int temp;

window.dequeue(temp);

nums.enqueue(temp);

}

return result;

}

}

maxSum = windowSum;

result.enqueue(maxSum);

while (!nums.isEmpty()) {

int out, in;

window.dequeue(out);

nums.dequeue(in);

windowSum = windowSum - out + in;

maxSum = max(maxSum, windowSum);

window.enqueue(in);

result.enqueue(maxSum);

}

while (!window.isEmpty()) {

int val;

window.dequeue(val);

nums.enqueue(val);

}

return result;

}

int main() {

Queue nums;

int input[] = { 1, 3, -1, -3, 5, 3, 6, 7 };

int k = 3;

for (int i = 0; i < 8; i++) {

nums.enqueue(input[i]);

}

Queue result = maxSumSlidingWindow(nums, k);

cout << "Max sum of sliding windows: ";

while (!result.isEmpty()) {

int sum;

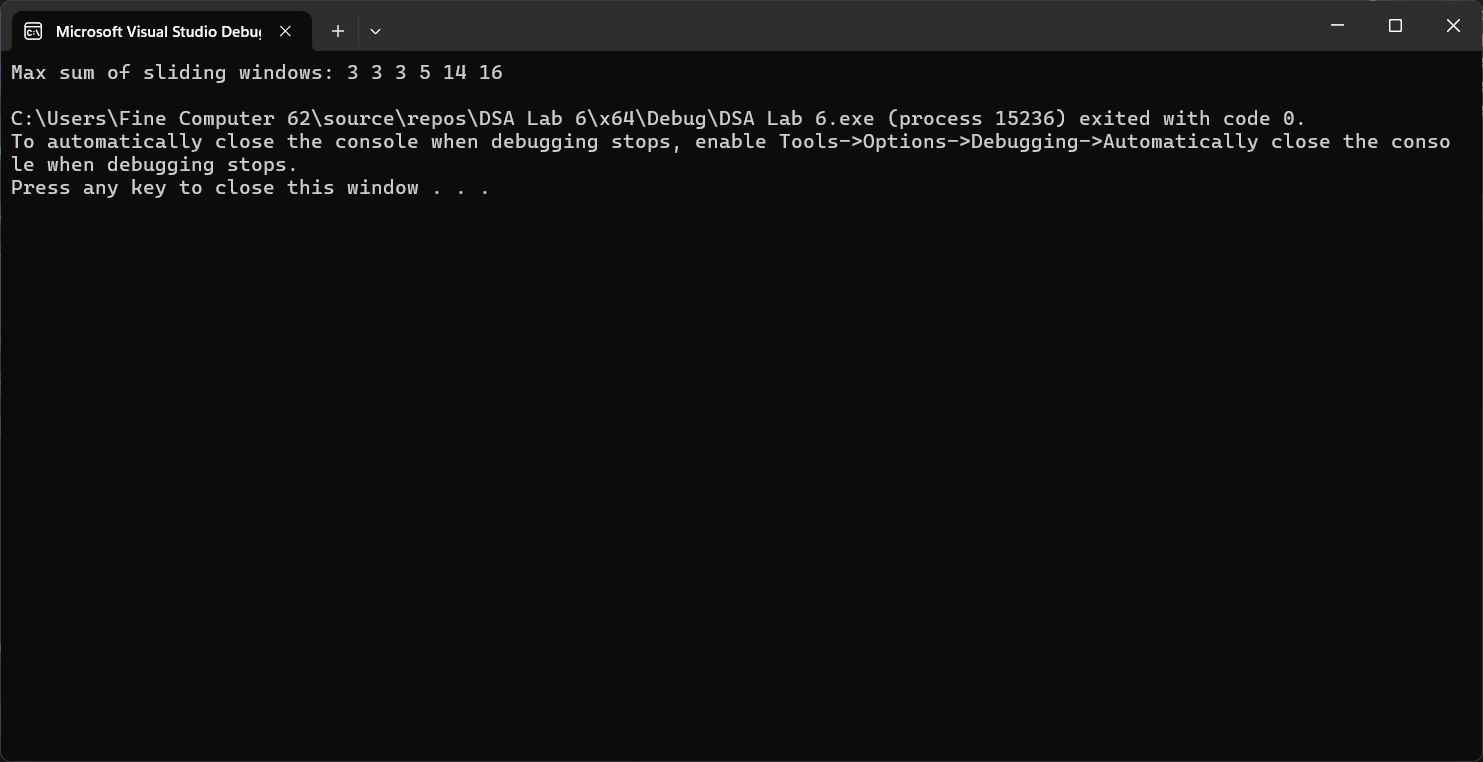
result.dequeue(sum);

cout << sum << " ";

}

cout << endl;

return 0;

} 

**TASK 3:**

#include <iostream>

using namespace std;

struct Node {

string ID;

int time;

};

class CircularQueue {

private:

Node\* arr;

int front, rear, size, capacity;

public:

CircularQueue(int cap) : front(-1), rear(-1), size(0), capacity(cap) {

arr = new Node[capacity];

}

~CircularQueue() {

delete[] arr;

}

void Enqueue(string id, int t) {

if (IsFull()) {

cout << "Queue is full\n";

return;

}

if (IsEmpty()) front = 0;

rear = (rear + 1) % capacity;

arr[rear] = { id, t };

size++;

}

void Dequeue() {

if (IsEmpty()) {

cout << "Queue is empty\n";

return;

}

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

else front = (front + 1) % capacity;

size--;

}

bool IsEmpty() {

return size == 0;

}

bool IsFull() {

return size == capacity;

}

void RoundRobin(int quantum) {

while (!IsEmpty()) {

Node current = arr[front];

cout << "Executing task " << current.ID << " for " << min(quantum, current.time) << " units\n";

current.time -= quantum;

Dequeue();

if (current.time > 0) {

Enqueue(current.ID, current.time);

}

}

}

};

int main() {

CircularQueue rr(5);

rr.Enqueue("T1", 10);

rr.Enqueue("T2", 5);

rr.Enqueue("T3", 8);

cout << "Starting Round-Robin scheduling with time quantum 2:\n";

rr.RoundRobin(2);

return 0;

} 