****

**LAB #10**

**NAME:** MANAHIL WASEEM

**SAP ID:** 54035

**DEPARTMENT:**  BS SE-3

**SUBMITTED TO.**  MAAM REHANA

1. **Create a program that simulates a circular queue for processing customer service requests. Each request has a request ID. Write functions to add requests to the queue, process the oldest request, and display the current queue of requests with array.**

**SOLUTION:**

#include <iostream>

#define MAX 5

using namespace std;

class Queue {

private:

int queue[MAX];

int front, rear;

public:

// Constructor to initialize front and rear

Queue() {

front = rear = -1;

}

// Check if the queue is empty

bool isEmpty() {

return (front == -1);

}

// Check if the queue is full

bool isFull() {

return ((rear + 1) % MAX == front);

}

// Add request to the queue

void enqueue(int requestID) {

if (isFull()) {

cout << "Queue is full. Cannot add more requests." << endl;

return;

}

if (isEmpty()) {

front = 0;

}

rear = (rear + 1) % MAX;

queue[rear] = requestID;

cout << "Request " << requestID << " added to the queue." << endl;

}

// Process the oldest request in the queue

void dequeue() {

if (isEmpty()) {

cout << "Queue is empty. No requests to process." << endl;

return;

}

cout << "Processing request " << queue[front] << endl;

if (front == rear) {

// Queue has only one element

front = rear = -1;

} else {

front = (front + 1) % MAX;

}

}

// Display the current queue of requests

void display() {

if (isEmpty()) {

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

return;

}

cout << "Current requests in the queue: ";

int i = front;

while (true) {

cout << queue[i] << " ";

if (i == rear) break;

i = (i + 1) % MAX;

}

cout << endl;

}

};

int main() {

Queue queue;

int choice, requestID;

do {

cout << "\n1. Add request\n2. Process request\n3. Display queue\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter request ID: ";

cin >> requestID;

queue.enqueue(requestID);

break;

case 2:

queue.dequeue();

break;

case 3:

queue.display();

break;

case 4:

cout << "Exiting..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

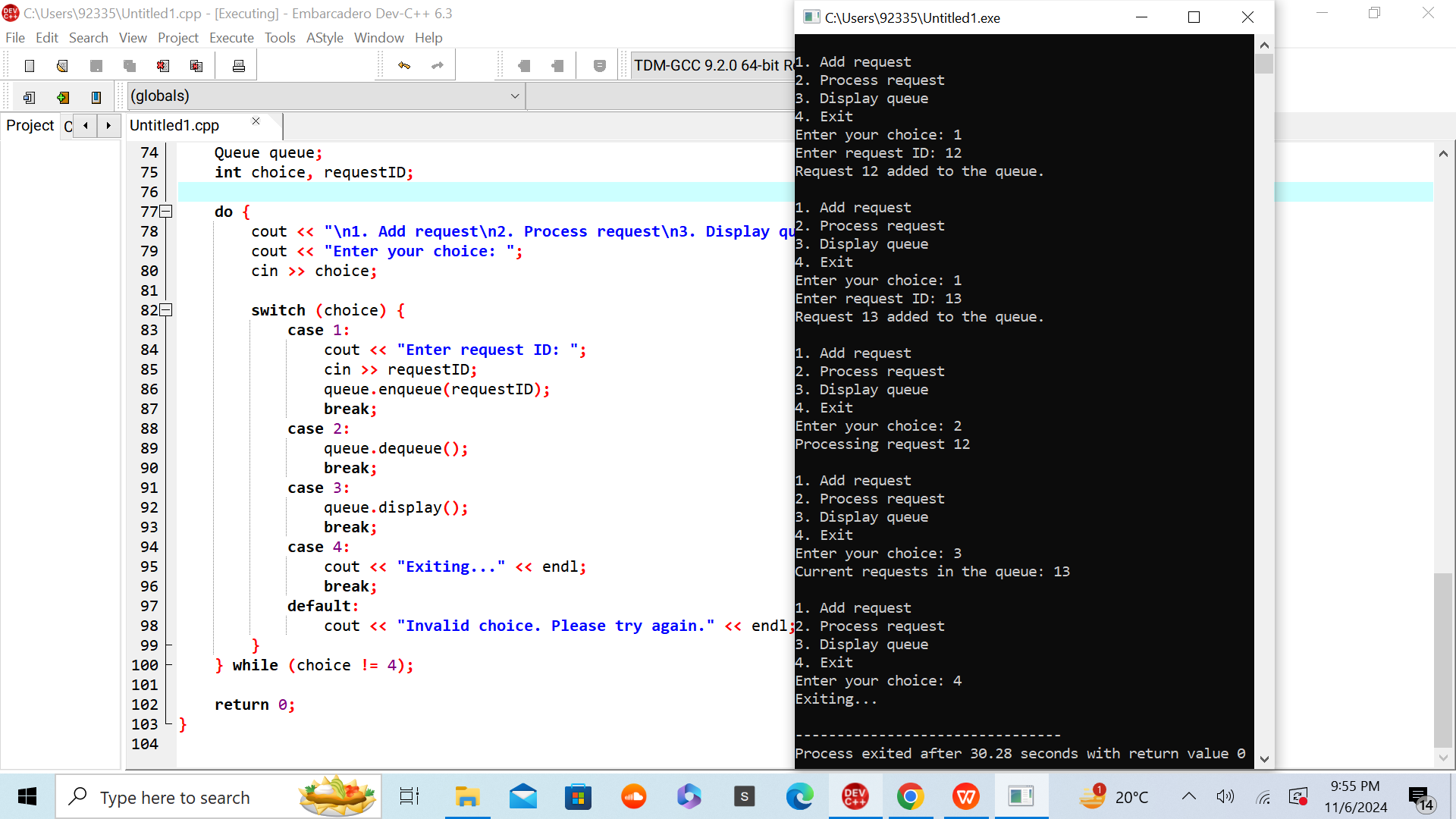
}

} while (choice != 4);

return 0;

}

**OUTPUT**:



1. **Create a program that uses a circular queue to manage player turns in a game. Each player has a player ID. Write functions to add players to the queue, move to the next player's turn, and display the current queue of players with linked list.**

**SOLUTION:**

#include <iostream>

using namespace std;

class Node {

public:

int playerID;

Node\* next;

Node(int id){

playerID=id;

next=nullptr;

}

};

Node \*head=nullptr;

class Queue {

public:

Node\* head;

void enqueue(int playerID) {

Node\* newNode = new Node(playerID);

if (head==nullptr) {

head = newNode;

head->next = head;

} else {

Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

}

void nextTurn() {

if (head!=nullptr) {

head = head->next;

}

}

void display() {

if (head==nullptr) {

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

return;

}

Node\* temp = head;

do {

cout << "Player ID: " << temp->playerID << " -> ";

temp = temp->next;

} while (temp != head);

cout << "Back to start" << endl;

}

};

int main() {

Queue queue;

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

cout << "Initial Queue:" << endl;

queue.display();

cout << "\nNext turn:" << endl;

queue.nextTurn();

queue.display();

cout << "\nNext turn:" << endl;

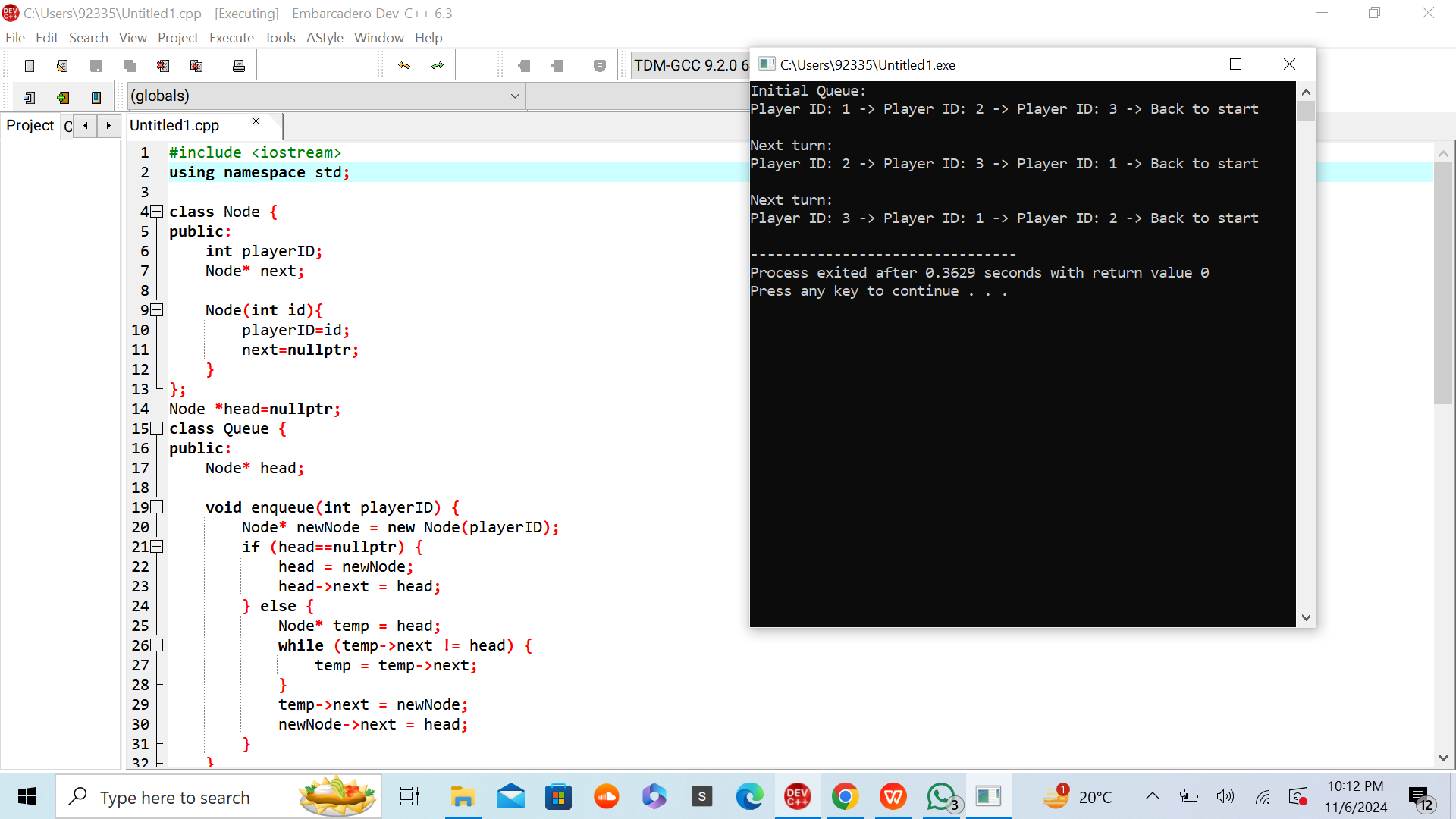
queue.nextTurn();

queue.display();

return 0;

}

**OUTPUT:**

****