# Department of Computing

**Name: Mahum Samar**

**CMS ID : 290647**

**CS250: Data Structure and Algorithms**

**Class: BSCS 9B**

# lab 06: queue

# Task 01

## Code:

#include <iostream>

using namespace std;

# define MAX 100

class LinearQueue

{

public:

int queue[MAX];

int front;

int rear;

LinearQueue()

{

//constructor

front = 0;

rear = -1;

}

bool isEmpty()

{

//method to check if the list is empty.

if (rear < front)

{

return true;

}

else

{

return false;

}

}

bool isFull()

{

//method to check if the list is full

if (rear == MAX - 1)

{

return true;

}

else

{

return false;

}

}

void EnQueue(int x)

{

//method to input the element at the end of the queue.

if (isFull())

{

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

}

else

{

rear++;

queue[rear] = x;

}

}

int DeQueue()

{

//method to remove the first element of the queue

if (isEmpty())

{

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

}

else

{

//if elements in the Queue

int x = queue[front];

front++;

return x;

}

}

void FirstElement()

{

//method to print the first element of the queue

if (isEmpty())

{

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

}

else

{

cout << queue[front] << endl;

}

}

void Clear()

{

//method to empty the queue

if (isEmpty())

{

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

}

else

{

front = 0;

rear = -1;

}

}

void ClearAndDisplay()

{

//method to empty the queue as well as display its elements

if (isEmpty())

{

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

}

else

{

while (!isEmpty())

{

cout << DeQueue() << " ";

}

cout << endl;

}

}

void Display()

{

//method to display the queue but not empty it.

LinearQueue\* lq = new LinearQueue();

if (isEmpty())

{

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

}

else

{

while (!isEmpty())

{

//printing the elemnts of the queue as well as storing it in the temporary queue

int x = DeQueue();

cout << x << " ";

lq->EnQueue(x);

}

while (!(lq->isEmpty()))

{

//removing the elemnts from the temporary queue and moving it back to the original queue

int x = lq->DeQueue();

EnQueue(x);

}

}

cout << endl;

}

};

int main()

{

//creating the linear queue

LinearQueue\* linearQueue = new LinearQueue();

cout << "Enqueue 5, 4, 3, 2 in the queue." << endl;

cout << endl;

linearQueue->EnQueue(5);

linearQueue->EnQueue(4);

linearQueue->EnQueue(3);

linearQueue->EnQueue(2);

cout << "Displaying the whole queue." << endl;

linearQueue->Display();

cout << endl;

cout << "Printing the first element of the queue." << endl;

linearQueue->FirstElement();

cout << endl;

cout << "Displaying the whole queue." << endl;

linearQueue->Display();

cout << endl;

cout << "DeQueue the first element of the queue." << endl;

linearQueue->DeQueue();

cout << endl;

cout << "Now again displaying the first element of the queue." << endl;

linearQueue->FirstElement();

cout << endl;

cout << "Displaying the whole queue." << endl;

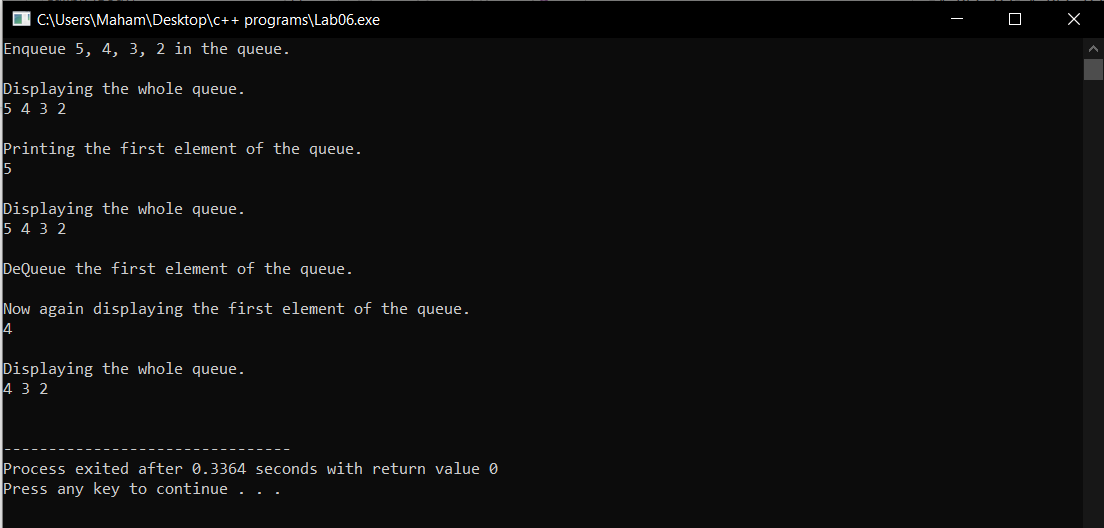
linearQueue->Display();

cout << endl;

return 0;

}

## output:



# Task 02

## Code:

#include <iostream>

using namespace std;

# define MAX 100

class CircularQueue

{

//creating a generic class of circular queue

public:

int queue[MAX]; //creating generic queue

int front;

int rear;

int lengthOfQueue;

CircularQueue()

{

front = 0;

rear = -1;

lengthOfQueue = 0;

}

bool isEmpty()

{

//method to check if the queue is empty

if (lengthOfQueue == 0)

{

return true;

}

else

{

return false;

}

}

bool isFull()

{

//method to check if the queue is full

if (lengthOfQueue == MAX)

{

return true;

}

else

{

return false;

}

}

void EnQueue(int x)

{

//method to input the elements in the circular queue.

if (isFull())

{

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

}

else

{

rear++;

if (rear == MAX)

{

rear = 0;

}

lengthOfQueue++;

queue[rear] = x;

}

}

int DeQueue()

{

//method to remove the first element of the circular queue where the front is pointing.

if (isEmpty())

{

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

}

else

{

//if elements in the Queue

int x = queue[front];

front++;

if (front == MAX)

{

front = 0;

}

lengthOfQueue--;

return x;

}

}

void FirstElement()

{

//method to print the first element of the queue

if (isEmpty())

{

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

}

else

{

cout << queue[front] << endl;

}

}

void Clear()

{

//method to clear the whole circular queue

if (isEmpty())

{

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

}

else

{

lengthOfQueue = 0;

front = 0;

rear = -1;

}

}

void Display()

{

//method to display the whole circular queue without removing its elements

if (isEmpty())

{

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

}

else

{

int y = lengthOfQueue;

while (y != 0)

{

int x = DeQueue();

cout << x << " ";

EnQueue(x);

y--;

}

}

cout << endl;

}

};

int main()

{

//creating the circular queue

CircularQueue\* circularQueue = new CircularQueue();

cout << "Enqueue 5, 4, 3, 2 in the queue." << endl;

cout << endl;

circularQueue->EnQueue(5);

circularQueue->EnQueue(4);

circularQueue->EnQueue(3);

circularQueue->EnQueue(2);

cout << "Displaying the whole queue." << endl;

circularQueue->Display();

cout << endl;

cout << "Printing the first element of the queue." << endl;

circularQueue->FirstElement();

cout << endl;

cout << "Displaying the whole queue." << endl;

circularQueue->Display();

cout << endl;

cout << "DeQueue the first element of the queue." << endl;

circularQueue->DeQueue();

cout << endl;

cout << "Now again displaying the first element of the queue." << endl;

circularQueue->FirstElement();

cout << endl;

cout << "Displaying the whole queue." << endl;

circularQueue->Display();

cout << endl;

cout << "Clearing the whole queue." << endl;

circularQueue->Clear();

cout << endl;

cout << "Displaying the whole queue after clearing it but nothing printed." << endl;

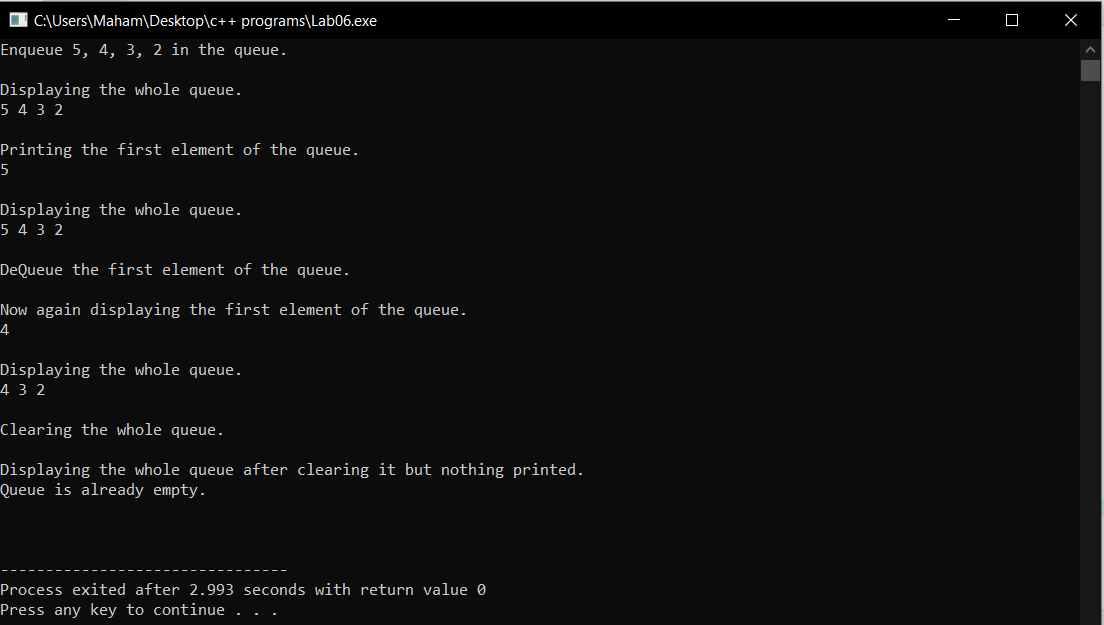
circularQueue->Display();

cout << endl;

return 0;

}

## output:



# Task 03

## Code:

#include <iostream>

using namespace std;

# define MAX 100

template <typename t>

class CircularQueue

{

//creating a generic class of circular queue

public:

t queue[MAX]; //creating generic queue

int front;

int rear;

int lengthOfQueue;

CircularQueue()

{

front = 0;

rear = -1;

lengthOfQueue = 0;

}

bool isEmpty()

{

//method to check if the queue is empty

if (lengthOfQueue == 0)

{

return true;

}

else

{

return false;

}

}

bool isFull()

{

//method to check if the queue is full

if (lengthOfQueue == MAX)

{

return true;

}

else

{

return false;

}

}

void EnQueue(t x)

{

//method to input the elements in the circular queue.

if (isFull())

{

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

}

else

{

rear++;

if (rear == MAX)

{

rear = 0;

}

lengthOfQueue++;

queue[rear] = x;

}

}

t DeQueue()

{

//method to remove the first element of the circular queue where the front is pointing.

if (isEmpty())

{

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

}

else

{

//if elements in the Queue

t x = queue[front];

front++;

if (front == MAX)

{

front = 0;

}

lengthOfQueue--;

return x;

}

}

void FirstElement()

{

//method to print the first element of the queue

if (isEmpty())

{

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

}

else

{

cout << queue[front] << endl;

}

}

bool Clear()

{

//method to clear the whole circular queue

if (isEmpty())

{

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

}

else

{

lengthOfQueue = 0;

front = 0;

rear = -1;

}

return true;

}

void Display()

{

//method to display the whole circular queue without removing its elements

if (isEmpty())

{

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

}

else

{

int y = lengthOfQueue;

while (y != 0)

{

t x = DeQueue();

cout << x.name << " " << x.id << endl;

EnQueue(x);

y--;

}

}

cout << endl;

}

};

#include <iostream>

#include <string>

//importing the Task02.cpp for using its methods

#include "Task02.cpp"

using namespace std;

# define MAX 100

int noOfPatient = 0;

class Patient

{

//patient class to input the patient information.

public:

//public variables

string name;

int id;

Patient(string name)

{

//constructor to initialize the patient attributes.

noOfPatient++;

this->name = name;

this->id = noOfPatient;

}

Patient()

{

//default constructor.

}

};

class waitingRoom

{

//class for implementing the methods for the hospital management system

public:

//creating the CircularQueue object to access its methods.

CircularQueue<Patient>\* circularQueue = new CircularQueue<Patient>();

void RegisterPatient()

{

//method for registering the patients in the system

string name; //string to input the name of the patient

cout << "Enter the name of the patient : " << endl;

std::getline(std::cin >> std::ws, name); //gets the input from the user.

//creating patient object to store its name and id

Patient\* patient = new Patient(name);

//input the patient info in the queue

circularQueue->EnQueue(\*patient);

}

void ServePatient()

{

//method to call the first patient in the queue.

Patient patient = circularQueue->DeQueue();

//prints the patient name and its id.

cout << patient.name << " " << patient.id << endl;

}

void CancelAll()

{

//method to clear all the waiting queue.

circularQueue->Clear();

}

bool CanDoctorGoHome()

{

//method to check if the waiting queue is empty

if (circularQueue->isEmpty())

{

return true;

}

else

{

return false;

}

}

void ShowAllPatient()

{

//method to display all the patients in the waiting queue.

circularQueue->Display();

}

};

int main()

{

char choice;

waitingRoom\* CircularQueue = new waitingRoom();

//do-while loop so that it executes atleast once.

do

{

int option;

cout << "What do you want to do : " << endl;

cout << "1. Register a Patient. " << endl;

cout << "2. Serve a Patient " << endl;

cout << "3. Cancel all the appointments. " << endl;

cout << "4. Check if the doctor can leave. " << endl;

cout << "5. Show the names of patients in the waiting list." << endl;

cout << "6. Exit the program." << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Choose one of the option below : ";

cin >> option;

if (option >= 1 && option <= 5)

{

switch (option)

{

case 1:

{

//for registering the patient

CircularQueue->RegisterPatient();

break;

}

case 2:

{

//info of the first patient on the waiting queue.

CircularQueue->ServePatient();

break;

}

case 3:

{

//for clearing all the waiting queue.

CircularQueue->CancelAll();

cout << "All the appointments have been cancelled." << endl;

break;

}

case 4:

{

//to check if the doctor can go home or not.

if (CircularQueue->CanDoctorGoHome())

{

cout << "The doctor can go home." << endl;

}

else

{

cout << "The doctor cannot go home." << endl;

}

break;

}

case 5:

{

//for diaplaying all the patients in the waiting queue

CircularQueue->ShowAllPatient();

break;

}

case 6:

{

//for exiting the program

return 0;

}

}

}

else

{

//if user enters the wrong choice.

cout << "You enter the wrong choice. Please choose between 1 and 5." << endl;

continue;

}

cout << "Do you want to exit the program ( Y / N ) : ";

cin >> choice;

cout << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

} while (choice == 'n' || choice == 'N');

return 0;

}

## output:

