# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BESE 13AB**

# 

# Lab 5: Implmentation of Queues in different problems

**Date: 20th October, 2023**

**Time: 10 am - 1 pm & 2 pm - 5 pm**

**Name:** Aimen Munawar

**CMS ID:** 415867

**Class:** BESE-13-A

# Lab Instructor: Anum Asif

# 

# Lab 5: Implmentation of Queues in different problems

**Introduction**

This lab is based on queues and its implementation statically and dynamically.

**Objectives**

Objective of this lab is to get familiar with the queues and implement it in a programming language

**Tools/Software Requirement**

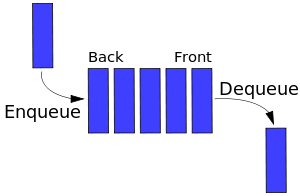
Visual Studio 2012 or gcc or g++

**Helping Material**

Lecture slides, text book

**Description**

In [computer science](http://en.wikipedia.org/wiki/Computer_science), a queue is a particular kind of [abstract data type](http://en.wikipedia.org/wiki/Abstract_data_type) or [collection](http://en.wikipedia.org/wiki/Collection_(computing)) in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a [First-In-First-Out (FIFO) data structure](http://en.wikipedia.org/wiki/FIFO_(computing)).



The following sets of operation are generally supported by queue:

1. void Enqueue(element) – add an element at the rear end of the queue

2. element Dequeue() – removes and display the element from the front end of the queue

3. bool isEmpty() – checks if the queue is empty or not

4. bool isFull() – checks if the queue is full or not

5. void Clear() – release the memory allocated by queue

6. void FirstElement() – display the contents of first element of queue at front location

**Lab Tasks**

You have to implement a waiting room management system in an emergency ward of a hospital. Your program will assign an Id number to a patient in a first come first serve basis. The lower the id, the sooner the service will be provided to the patient.

Your program will contain the following methods:

**RegisterPatient():** This method assigns an Id (which is auto-generated) to a patient and register him/her to the system.

**ServePatient():** This method calls a patient to provide hospital service to him/her.

**CancelAll():** This method cancels all appointments of the patients so that the doctor can go to lunch.

**CanDoctorGoHome():** This method returns true if no one is waiting, otherwise, returns false.

**ShowAllPatient():** This method shows all ids of the waiting patients in SORTED order. (Hint: use the sorting methods learnt in class using the appropriate data-structure for each task) [Sorted according to their names]

Solve the above problem using an *array based queue*.

**Important Note:** Practice your knowledge of OOP with C++ when creating a solution.

**Solution:**

|  |
| --- |
| Solution |
| Task 1 Code:  #include <iostream>  #include <string>  using namespace std;  class PatientQueue {  public:  int front, rear, capacity; // Pointers and capacity  int\* patientIDs; // Array to store patient IDs  string\* patientNames; // Array to store patient names  int currentID; // Track current ID  int appointmentCount; // Track appointments  // Constructor  PatientQueue() {  front = rear = 0; // Initialize front and rear pointers  capacity = 0; // Initialize capacity  patientIDs = nullptr; // Initialize patient IDs array  patientNames = nullptr; // Initialize patient names array  currentID = 0; // Initialize current ID  appointmentCount = 0; // Initialize appointment count  }  // Set maximum capacity of the queue  void SetCapacity(int maxCapacity) {  capacity = maxCapacity; // Set capacity  patientIDs = new int[capacity]; // Allocate memory for patient IDs  patientNames = new string[capacity]; // Allocate memory for patient names  }  // Add a patient to the queue  void AddPatient(int id, string name) {  if (rear == capacity) { // Check if queue is full  cout << "Queue is full." << endl;  return;  }  else {  patientIDs[rear] = id; // Add patient ID to the queue  patientNames[rear] = name; // Add patient name to the queue  rear++; // Increment rear pointer  appointmentCount++; // Increment appointment count  }  }  // Serve the next patient in the queue  int ServeNextPatient() {  if (front == rear) { // Check if queue is empty  cout << "Queue is empty." << endl;  return 0;  }  else {  cout << "Serving patient ID: " << patientIDs[front] << ", Name: " << patientNames[front] << endl;  // Shift elements to remove served patient  for (int i = 0; i < rear - 1; i++) {  patientIDs[i] = patientIDs[i + 1];  patientNames[i] = patientNames[i + 1];  }  rear--; // Decrement rear pointer  appointmentCount--; // Decrement appointment count  }  return 0;  }  // Check if the queue is empty  bool IsEmpty() {  return front == rear;  }  // Check if the queue is full  bool IsFull() {  return rear == capacity;  }  // Clear the queue and reset pointers  void ClearQueue() {  currentID = 0; // Reset current ID  delete[] patientIDs; // Deallocate memory for patient IDs  delete[] patientNames; // Deallocate memory for patient names  patientIDs = new int[capacity]; // Allocate new memory for patient IDs  patientNames = new string[capacity]; // Allocate new memory for patient names  front = rear = 0; // Reset front and rear pointers  appointmentCount = 0; // Reset appointment count  }  // Display information of the first patient  void DisplayFirstPatient() {  if (!IsEmpty()) {  cout << "Patient ID to be served: " << patientIDs[front] << ", Name: " << patientNames[front] << "." << endl;  }  else {  cout << "Queue is empty." << endl;  }  }  // Register a new patient  void RegisterNewPatient() {  string name;  currentID++; // Increment current ID to generate a unique ID for the patient  int id = currentID;  cout << "Enter patient's name: ";  cin >> name;  AddPatient(id, name); // Add the patient to the queue  }  // Cancel all appointments  void CancelAllAppointments() {  ClearQueue(); // Clear the queue and reset pointers  front = rear = 0; // Reset front and rear pointers  cout << "All appointments canceled. Doctor may now go to lunch." << endl;  }  // Check if doctor can go home  bool CanDoctorGoHome() {  return IsEmpty(); // Doctor can go home if the queue is empty  }  // Show all patients sorted by name  void ShowAllPatients() {  if (IsEmpty()) { // Check if the queue is empty  cout << "No patients on waiting list." << endl;  return;  }  // Sort patients by name using bubble sort  for (int i = 0; i < rear; i++) {  for (int j = 0; j < rear - i - 1; j++) {  if (patientNames[j] > patientNames[j + 1]) {  string tempName = patientNames[j];  patientNames[j] = patientNames[j + 1];  patientNames[j + 1] = tempName;  int tempID = patientIDs[j];  patientIDs[j] = patientIDs[j + 1];  patientIDs[j + 1] = tempID;  }  }  }  // Display sorted list  cout << "Sorted list by patient name:" << endl;  for (int i = 0; i < rear; i++) {  cout << "Patient Name: " << patientNames[i] << ", ID: " << patientIDs[i] << endl;  }  }  };  int main() {  PatientQueue patientQueue;  int maxCapacity;  cout << "Enter the maximum capacity of the waiting room: ";  cin >> maxCapacity;  patientQueue.SetCapacity(maxCapacity);  while (1) {  char choice;  cout << "Select an option:\n"  << "A. Register a patient\n"  << "B. Serve a patient\n"  << "C. Cancel all appointments\n"  << "D. Check if Doctor can go home\n"  << "E. Show all patients, sorted by names" << endl;  cin >> choice;  switch (choice) {  case 'A':  patientQueue.RegisterNewPatient();  break;  case 'B':  patientQueue.ServeNextPatient();  break;  case 'C':  patientQueue.CancelAllAppointments();  break;  case 'D':  if (patientQueue.CanDoctorGoHome()) {  cout << "Doctor can go home." << endl;  }  else {  cout << "Doctor cannot go home yet." << endl;  }  break;  case 'E':  patientQueue.ShowAllPatients();  break;  default:  cout << "Invalid choice. Please try again." << endl;  }  }  return 0;  }  Task 1 Output Screenshot:(The screenshots are Continuous) | |

### Deliverables

Compile a single word document by filling in the solution part and submit this Word file on LMS. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.