**THE INSTITUTE OF FINANCE MANAGEMENT**

**(IFM)**



**FACULTY OF COMPUTING, INFORMATION SYSTEMS AND MATHEMATICS**

**BACHELOR’S IN COMPUTER SCIENCE**

**GROUP ASSIGNMENT**

**DATA STRUCTURE AND ALGORITHM ASSIGNMENT**

**BCS II**

|  |  |  |
| --- | --- | --- |
| **NAME:** | **REG NO:** | **MARKS** |
| **KEVIN J. MSEMAKWELI** | **IMC/BCS/2010929** |  |
| **DAUD, DAVID C.** | **IMC/BCS/2011171** |  |
| **FREDRICK J. KILABUKO** | **IMC/BCS/2012326** |  |
| **MAHADY Y. ALLY** | **IMC/BCS/2010946** |  |
| **CHRISTOPHER LISSO** | **IMC/BCS/2013141** |  |

**THE SOLUTION OF THE ASSIGNMENT QUESTION 1**

1. **CPP CODE**

#include <iostream>

using namespace std;

class Patient

{

public:

int serialNo;

int age;

Patient \*next;

// initial stage of the que before a patient

Patient ()

{

serialNo = 999;

age = 0;

next = NULL;

}

// from user inputs

Patient (int t, int a)

{

serialNo = t;

age = a;

next = NULL;

}

};

class Tickets

{

public:

Patient \*front;

Patient \*rear;

int serialNo;

Tickets ()

{

front = NULL;

rear = NULL;

serialNo = 999;

}

bool isEmpty ()

{

if (front == NULL && rear == NULL)

return true;

else

return false;

}

int count ()

{

int count = 0;

Patient \*temp = front;

while (temp != NULL)

{

count++;

temp = temp->next;

}

return count;

}

void enqueue(int input)

{

Patient \*p = new Patient; /\*we use new because we want the node

to persist in the memory, even after

the function call is over.

(Dynamic memory allocation)\*/

p->age = input;

if (isEmpty())

{

front = rear = p;

}

// enqueue from front when priority is met

else if(input <= 05 or input >= 60)

{

p->next = front;

front = p;

cout << front->age;

}

// enqueue from rear for age with no priority

else

{

rear->next = p;

rear = p;

}

serialNo++;

p->serialNo = serialNo;

}

// function for removing patients from queue.

void dequeue()

{

Patient \*temp = front;

if (isEmpty())

{

cout << endl

<< "There no Patients to Attend."

<< endl;

}

else if (front == rear)

{

front = NULL;

rear = NULL;

cout << endl

<< "Attending Patient."

<< endl

<< "Ticket No:” << temp->serialNo

<< endl

<< "Age : " << temp->age;

delete temp;

}

else

{

front = front->next;

cout << endl

<< "Attending Patient."

<< endl

<< "Ticket No : " << temp->serialNo

<< endl

<< "Age : " << temp->age;

delete temp;

}

}

// Function for displaying the number of patients

void display()

{

Patient \*temp = front;

if (isEmpty())

cout << endl

<< "There no Waiting Patients.";

else

{

cout << endl

<< "The next Patient." << endl

<< "Ticket No : " << temp->serialNo

<< endl

<< "Age : " << temp->age;

}

}

// function for listing the Tickets of patients

void list()

{

Patient \* temp = front;

if (isEmpty())

{

cout << endl

<< "There no Patients.";

}

else

{

cout<<endl<<"Ticket No.\tAge"<<endl;

while ( temp != NULL)

{

cout<<temp->serialNo<<"\t\t"<<temp->age

<<endl;

temp = temp -> next;

}

}

}

};

int main()

{

Tickets t;

int age, option;

Patient \*temp = new Patient;

do

{

cout << endl;

cout << "1. Add new Patient" << endl;

cout << "2. Attend Patient" << endl;

cout << "3. Unattended Patients" << endl;

cout << "4. Display Next Patient" << endl;

cout << "5. List all Patients" << endl

<< endl;

cout << "Enter your option number : ";

cin >> option;

switch (option)

{

case 1:

cout << endl

<< "Enter Patients Age : ";

cin >> age;

t.enqueue(age);

cout << endl;

break;

case 2:

t.dequeue();

cout << endl;

break;

case 3:

cout << endl

<< "No. of Unattended Patients : " << t.count()

<< endl;

break;

case 4:

t.display();

cout << endl;

break;

case 5:

t.list();

cout<<endl;

break;

default:

break;

}

} while (option != -1);

}

1. **EXPLAIN THE DATA STRUCTURE YOU CHOOSE AND GIVE THE REASON**

According to the question said, we were regarded to choose LINKED LIST as the type of the Data structure. Linked List is the most important data structure. A Linked list is made up of nodes. We call every flower on this particular garland to be a node. And each of the node points to the next node in this list as well as it has data (here it is type of flower). Furthermore, a linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next.

This is because it offers to solve cases based on the following reasons;

1. **Dynamic in nature**.

A linked list is a dynamic arrangement so it can grow and shrink at run time by allocating and de-allocating memory. So, there is no need to give the initial size of the linked list

1. **Unlimited number of data.**

In this reason, Linked List based on number of data that does not be predicted or number of data keeps changing during execution of the program

1. **This queue is automatic intelligent.**

Due to If two elements in a priority queue have the same priority value, they’ll be arranged using the FIFO principle this will help us in our project let’s assume that two patients come at the same such as elder and child, this may be easier to solve under priority queue due to it will select the first come be serviced.

1. **Insertion and Deletion Operations**

Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.

For that reasons, Linked List is preferred to be chosen as the requirements given.

1. **CHALLENGE FACED DURING THE DEVELOPMENT**

As we choose Linked List Data Structure to handle the queue and prioritize the elderly and children, we face some challenge during implementation of the priority.

**IMPLEMENTING THE PRIORITY.**

Implementation of the priority was quite challenging, since we wanted to reduce the time complexity to the O (1) or O(nlogn). We analyzed three functions of two algorithm for unsorted and sorted linked list and come with the following results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Linked List** | **Insertion Of Age** | **Getting the Min Age / Max Age** | **Dequeue Max Age / Min Age** |
| Unsorted | **O(1)** | **O(N)** | **O(N) + O(1) = O(N)** |
| Sorted | **O(N) + O(1) = O(N)** | **O(1)** | **O(1)** |

Unsorted linked List :

* In inserting new element, you just insert from front or from rear, so the time complexity is order of 1 O(1)
* Getting the children or elder Age, you have to traverse through the linked list up to the number of elements in the linked List. So it is O(N).
* For dequeuing due to priority, you have to traverse through all elements based on the condition = O(N). Then deletion of the element is O(1).

O(N) + O(1) = O(N)

Sorted Linked List :

* Inserting a new element due to priority, you have to traverse through all elements to find the best position of the element = O(N). Then insertion of the element is constant time O(1).

O(N) + O(1) = O(N)

* Since the list is sorted, finding and getting the element is at constant time O(N).
* Also for dequeuing the location of the element will be known either at the head or at the tail. So it will be a constant time for removing the element O(1)

We analyzed the results and concluded that there both expensive one at enqueuing of a new patient and the other at dequeuing a patient.

So we choose to stick with the unsorted list, but implementing the priority only on the process of insertion.