Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

COMP 202

Mini Project

Submitted by
Shrawak Bhattarai(11)
Ramraj Chimouriya(14)
CE 2nd Year/1st Sem

Submitted To
Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Table of Contents

1. Introduction

We were assigned to do mini projects for our course **COMP 202**. Lists of mini projects were given to us to select three. We were assigned **mini project no. 6** from the list.

In this project, we present a C++ implementation of the queue ADT using a circularly linked list. For this purpose,our approach is to use the circularly linked list class, called ***CircularLinkedList*** from C header file, called ***circularlinkedlist.h***.

In order to implement a queue operations, we created a class ***CircularLinkedQueue*** in C header file ***circularlinkedqueue.h***. These classes are further described briefly below.

1.1. Task

6. Implement a queue using a circularly linked list with the following operations:

• enqueue – Inserts an element at the end of the queue

• dequeue – Displays and deletes an element from the front of the queue

• rear – Displays the last element of the queue

• front – Displays the first element of the queue

• isEmpty – Returns true if the queue is empty and false otherwise

• isFull – Returns true if the queue is full and false otherwise

What are the time complexities of these operations in your implementation?

You are supposed to use the linked list implementation done in Lab 2.

2. circularlinkedlist.h

In this C header file, two classes are defined, namely ***Node*** and ***CircularLinkedList***.

Node class represent each nodes of circularly linked list of class ***CircularLinkedList***. Each node in a circularly linked list has a **next** pointer and an element value **info**. But, rather than having a head or tail, the nodes of a circularly linked list are linked into a cycle i.e. if we traverse the nodes of a circularly linked list from any node by following next pointers, we eventually visit all the nodes and cycle back to the node from which we started.

Even though a circularly linked list has no beginning or end, we nevertheless need some node to be marked as a special node, which we call **cursor**. In our case, ***CircularLinkedList*** has a private data pointer member **cursor** which represents tail of list and element next to **cursor** is head. Member functions of ***CircularLinkedList*** are discussed on chapter **2.2**.

2.1. Class Structure

2.1.1. Class Structure of *Node*

```
class Node      //circularly linked list node
{
   public:
      int info;  //linked list element value
     Node* next;   //next item in the list

};
```

2.1.2. Class Structure of *CircularLinkedList*

```
class CircularLinkedList        //a circularly linked list
{
   private:
      Node* cursor;              //the cursor pointing tail of list
   public:
      CircularLinkedList();    //constructor
      ~CircularLinkedList(); //destructor
      bool isEmpty();          //is list empty?
      int head();              //returns element following tail
      int tail();              //returns element at tail
```

```
    void addToTail(int ele); //add element after tail
    void removeFromHead();  //display and remove element at head
};
```

2.2. Member Functions of *CircularLinkedList*

2.2.1. CircularLinkedList()

Initialize a circular linked list with cursor set to null.

2.2.2. isEmpty()

Returns true if list is empty.

2.2.3. head()

Return the element immediately after the cursor; an error results if the list is empty.

2.2.4. tail()

Return the element referenced by the cursor; an error results if the list is empty.

2.2.5. addToTail(ele)

Insert a new node with element **ele** immediately after the cursor; if the list is empty, then this node becomes the cursor and its next pointer points to itself.

2.2.6. removeFromHead()

Remove the node immediately after the cursor (not the cursor itself, unless it is the only node); if the list becomes empty, the cursor is set to null.

3. circularlinkedqueue.h

Only a class ***CircularLinkedQueue*** is defined in this C header file.

CircularLinkedQueue has data members **C**, **n** and **max. C** is object of class ***CircularLinkedList***. ***n*** and **max are** integers representing number of nodes and maximum number of elements that queue can store respectively. Member functions of ***CircularLinkedList*** are discussed thoroughly on chapter **3.2**.

3.1. Class Structure of *CircularLinkedQueue*

```
class CircularLinkedQueue  //queue as circularly linked list
{
   private:
      CircularLinkedList C;  //circularly linked list of elements of queue
      int n;                 //number of nodes
      int max;               //maximum number of element
    public:
      CircularLinkedQueue (int max_size);  //constructor
      ~CircularLinkedQueue();              //destructor
      bool isEmpty();                      //is the queue empty?
      bool isFull();                       //is the queue full?
      int front();                         //front element of queue
      int rear();                          //rear element of queue
      void enqueue(int ele);               //add element at rear
      void dequeue();                      //display and remove element from front
};
```

3.2. Member Functions of *CircularLinkedQueue*

3.2.1. CircularLinkedQueue(max_size)

Initialize a queue using circular linked list with a null ***CircularLinkedList*** object **C**, number of nodes **n** set to 0 and maximum size of queue **max** set to **max_size**.

3.2.2. isEmpty()

Returns true if **n** is equal to 0.

3.2.3. isFull()

Returns true if **n** is equal to **max**.

3.2.4. front()

The class ***CircuralLinkedList*** has a member function **head()**, which returns the head of list. Here, **front()** function invokes **head()** function of member object **C**, which in return, returns the front element of queue.

3.2.5. rear()

The class ***CircuralLinkedList*** has a member function **tail()**, which returns the head of list. Here, **rear()** function invokes **tail()** function of member object **C**, which in return, returns the rear element of queue.

3.2.6. enqueue(ele)

enqueue(ele) first invoke the function **addToTail(ele)** of **C**,which inserts element **ele** just after the **cursor**, that is, just after the rear of the queue. After that, it increase **n** by 1, to count number of elements in queue.

3.2.7. dequeue()

First, this function invokes **C.removeFromHead()**, thus removing the node just after the **cursor**, that is, the front of the queue. It then, decreases **n** by 1, decreasing count of elements in queue.

4. Queue using *CircularLinkedList*

A program ***main.cpp*** is written in **C++** to illustrate the use of class ***CircularLinkedQueue***.

4.1. main.cpp

```cpp
#include "circularlinkedqueue.cpp"

int main()
{
    CircularLinkedQueue Cq(5);  //Inatializing a queue Cq with max size 5

    if(Cq.isEmpty()) cout<<"Queue is empty."<<endl;  //Checking if queue is empty?

    Cq.enqueue(5);    //Enqueue
    Cq.enqueue(12);
    Cq.enqueue(3);
    Cq.enqueue(1);
    Cq.enqueue(50);

    cout<<"front ele="<<Cq.front()<<endl;  //Front element of queue
    cout<<"rear ele="<<Cq.rear()<<endl;    //Rear element of queue

    if(Cq.isFull())                  //Checking if queue is full?
    {
        Cq.enqueue(90);         //Adding element to full queue
    }

    Cq.dequeue();     //Dequeue
    Cq.dequeue();
    Cq.dequeue();

    cout<<"front ele="<<Cq.front()<<endl;   //Front element of queue
    cout<<"rear ele="<<Cq.rear()<<endl;     //Rear element of queue
}

```

Figure 1. main.cpp file screenshot

4.2. Compiling and Running main.cpp

To compile above program, ***g++***, ***clang***, ***minGw***, or ***any other C++ compiler*** is required.

Moreover, following files are mandotary:

circularlinkedlist.h

circularlinkedlist.cpp

circularlinkedqueue.h

circularlinkedqueue.cpp

These files could be downloaded from http://github.com/RamrajCh/CE2018_MP_11_14 .

These files should be in same directory as main.cpp.

Compilation and running in Linux

1. Open Terminal (Ctrl+Alt+T)
2. Change directory to your working directory.
3. Compile: **g++ main.cpp**

4. Run: **./a.out**

4.3. Result

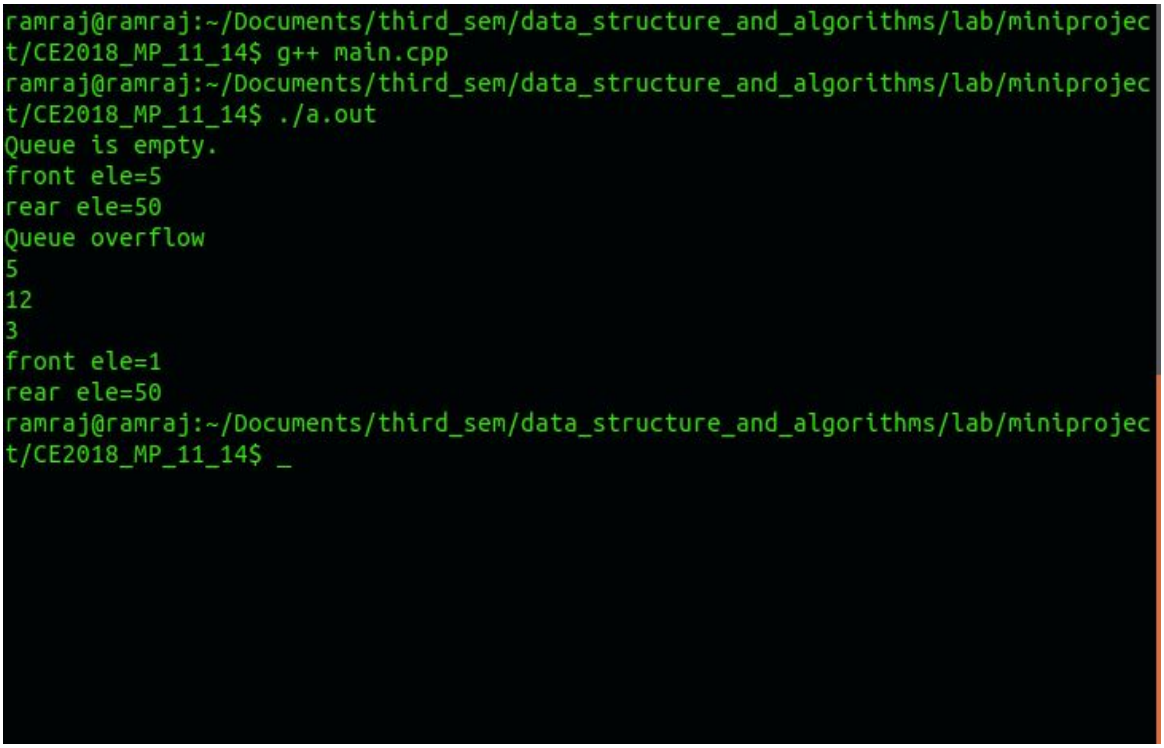

Figure 2. Compiling and Running main.cpp

4.4. Time Complexity

Implementation	Best Case	Worst Case	Time Complexity
CircularLinkedQueue()	2	2	O(1)
isEmpty()	1	1	O(1)
isFull()	1	1	O(1)
enqueue(ele)	3	8	O(1)
dequeue()	3	9	O(1)
front()	3	3	O(1)
rear()	3	3	O(1)

So, the time complexity is O(1).