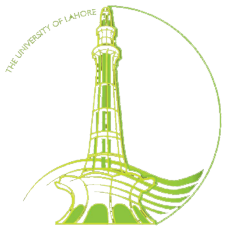

THE UNIVERSITY
OF LAHORE
ISLAMABAD
CAMPUS

Data Structures & Algorithms
(CS09203)

Lab Report

Name:	Jassim Bashir
Registration #:	SEU-F16-101
Lab Report #:	02
Dated:	16-04-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 2
Queue with Array implementation

Objective
The objective of this session is to understand the various operations on queues using array structure in C++.

Software Tool
1. Language: C++
2. Compiler: Dev C++
3. OS: Windows 10

1 Theory

This manual discusses an important data structure, called a queue. The idea of a queue in computer science is the same as the idea of the queues to which you are accustomed in everyday life. There are queues of customers in a bank or in a grocery store and queues of cars waiting to pass through a toll booth. Similarly, because a computer can send a print request faster than a printer can print, a queue of documents is often waiting to be printed at a printer. The general rule to process elements in a queue is that the customer at the front of the queue is served next and that when a new customer arrives, he or she stands at the end of the queue. That is, a queue is a First In First Out datastructure. A queue is a set of elements of the same type in which the elements are added a tone end, called the back or rear, and deleted from the other end, called the front. For example, consider a line of customers in a bank, where in the customers are waiting to withdraw/deposit money or to conduct some other business. Each new customer gets in the line at the rear. Whenever a teller is ready for a new customer, the customer at the front of the line is served. The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue. As in a stack, the middle elements of the queue are in accessible, even if the queue elements are stored in an array. Queue: A datastructure in which the elements are added at one end, called the rear, and deleted from the other end, called the front; a First-In-First-Out (FIFO) datastructure. Queues may be represented in the

computer in various ways, usually by means at one-way list or linear arrays. Unless otherwise stated or implied each of our queues will be maintained by a linear array QUEUE and two pointer variable FRONT containing the location of the front element of the queue and REAR containing the location of the rear element of the queue. The condition FRONT=NULL will indicate that the queue is empty. Whenever an element is deleted from the queue the value of FRONT is increased by one. This can be implemented by the assignment. FRONT = FRONT + 1. Similarly, when ever an element is added to the queue the value of REAR is increased by one. This can be implemented by the assignment. REAR = REAR + 1. This means that after N insertions the rear element of the queue will occupy QUEUE[N] or in otherwords eventually the queue will occupy the last part of the array. This occurs even though the queue itself may not contain many elements. Suppose we want to insert an element ITEM into a queue at the time the queue does occupy the last part of the array i.e. when REAR=N. One way is to do this simply move the entire queue to the beginning of the array changing FRONT and REAR accordingly, and the n inserting ITEM as above. This procedure may by very expensive.The procedure we adopt is to assume that the array QUEUE is circular that is that QUEUE[1] comes after QUEUE[N] in the array. With this assumption, we insert ITEM in to the queue by assigning ITEM to QUEUE[1]. Specifically, instead of increasing REAR to N+1 were set REAR=1 and then assign QUEUE[REAR] = ITEM. Similarly, if FRONT=N and an element is deleted then were set FRONT=1 instead of increasing FRONT to +1. Suppose that our queue only contains one element i.e. suppose that FRONT = REAR NULL And suppose that the element is deleted. Then we assign FRONT = NULL and REAR= NULL to indicate that the queue is empty.

2 Task

2.1 Procedure: Task 1

Write a C++ code to perform insertion and deletion in queue using arrays applying the algorithms given in the manual.

```
#include<iostream>
using namespace std;
#define size 5
```

Figure 1: Queue

```
int ar[size],front=0,rear=0;
int main()
{
int ch;
void enqueue();
void dequeue();
void display();
while(1)
{
cout<<"\n1. FOR INSERTION";
cout<<"\n2. FOR DELETION";
cout<<"\n3. FOR DISPLAY";
cout<<"\n4. TO EXIT";
cout<<"\n\nEnter your choice: ";
cin>>ch;
switch(ch)
{
case 1:
enqueue();
break;
case 2:
dequeue();
break;
case 3:
```

```
display();
break;
case 4:
exit(0);
default:
cout<<"\nInvalid Entry"<<endl;
}
}
}
void enqueue()
{
int j;
if (rear==size && front==0)
cout<<"\nQueue is full"<<endl;
else
{
cout<<"\nEnter the number:";
cin>>j;
ar[rear]=j;
}
rear++;
}
void dequeue()
{
int j,i;
if (front==rear)
cout<<"\nQueue is empty"<<endl;
else
{
j=ar[front];
front++;
cout<<endl<<j<<" is deleted."<<endl;
}
}
void display()
{
int i,temp=front;
if (front==rear)
cout<<"\nQueue is empty"<<endl;
else
```

```
{
cout<<endl<<"\nElements in the queue:";
for(i=temp;i<rear;i++)
{
cout<<ar[i]<<" ";
}
}
}
```

3 Conclusion

In this lab we learned how to create queue, its functioning and implementation. In this program we learned to add and delete an element from the queue.