



## **Data Structures and Algorithms ( CS09203 )**

### **Lab Report**

Name: Farhan Naseer  
Registration #: SEU-F16-125  
Lab Report #: 02  
Dated: 16-04-2018  
Submitted To: Sir. Usman Ahmed

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

## Experiment # 02

### 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. I use Code Blocks with GCC compiler.

## 1 Theory

Queue using Array: -

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 tollbooth. 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 data structure.

A queue is a set of elements of the same type in which the elements are added at one 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, wherein 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 inaccessible, even if the queue elements are stored in an array.

Queue: A data structure 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) data structure.

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.

$$\text{FRONT} = \text{FRONT} + 1$$

Similarly, whenever an element is added to the queue the value of `REAR` is increased by one. This can be implemented by the assignment.

$$\text{REAR} = \text{REAR} + 1$$

This means that after `N` insertions the rear element of the queue will occupy `QUEUE[N]` or in other words 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 then inserting `ITEM` as above. This procedure may be 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` into the queue by assigning `ITEM` to `QUEUE[1]`. Specifically, instead of increasing `REAR` to `N+1` we reset `REAR=1` and then assign

$$\text{QUEUE}[\text{REAR}] = \text{ITEM}$$

Similarly, if `FRONT=N` and an element is deleted then we reset `FRONT=1` instead of increasing `FRONT` to `N+1`.

Suppose that our queue only contains one element i.e. suppose that



Figure 1: output

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 5

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

### 2.2

```
#include<iostream>
#include<conio.h>
#define SIZE 5
using namespace std;
int q[SIZE], front=0, rear=0;
void enqueue()
{
```

```

int no;
if (rear==SIZE && front==0)
cout<<"queue is full";
else
{
cout<<"enter the num:";
cin>>no;
q[rear]=no;
}
rear++;
}
void dequeue()
{
int no,i;
if (front==rear)
cout<<"queue is empty";
else
{
no=q[front];
front++;
cout<<"\n"<<no<<" _removed from the queue\n";
}}
void display()
{
int i,temp=front;
if (front==rear)
cout<<"the queue is empty";
else
{
cout<<"\n element in the queue:";
for (i=temp;i<rear;i++)
{
cout<<q[i]<<" _";
}}
}
int main()
{
int ch;
while(1)
{
cout<<"\n 1. _add element";

```

```

cout<<"\n_2._remove_element";
cout<<"\n_3.display";
cout<<"\n_4.exit";
cout<<"\n_enter_your_choice:";
cin>>ch;
switch(ch)
{
case 1:
enqueue();
break;
case 2:
dequeue();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
cout<<"\n_invalid_choice";
}}}
```

### 3 Conclusion

In today lab we have learned about the Queue which is first in first out (FIFO) In which we have done four function first insertion then deletion and displaying the queue and last to exit the program.