



THE UNIVERSITY
OF LAHORE
**ISLAMABAD
CAMPUS**

**Data Structures and Algorithms
(CS09203)**

Lab Report

Name: Rehman ullah baig

Reg No: CSU-F16-118

Lab Report: 2

Submitted To: Sir, Usman Ahmed

Lab: 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 Tools: -

To achieve the goals of objectives, I use Dev C++.

Theory: -

Queue using Array: -

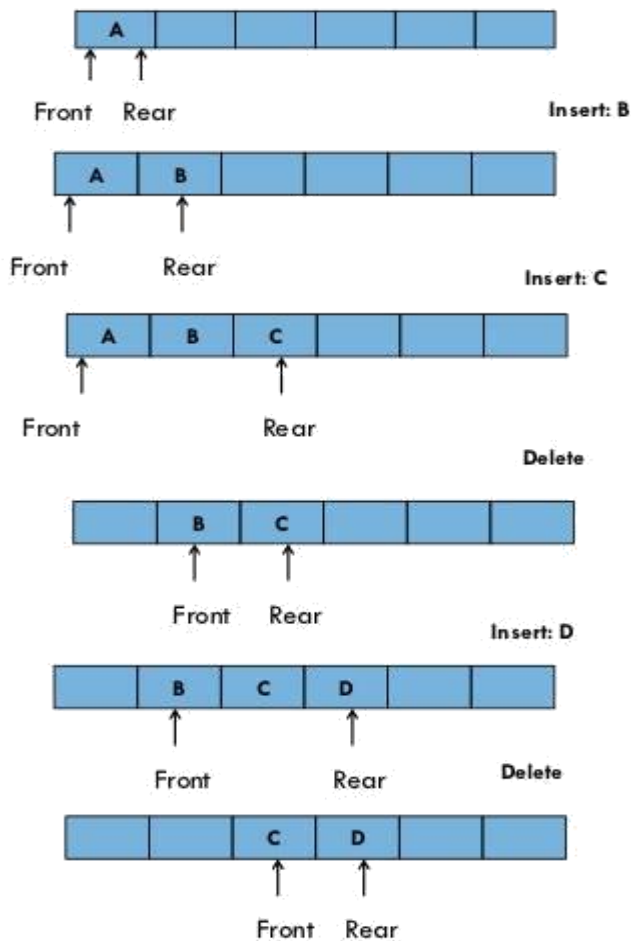
This manual discusses an important data structure, called a queue. The idea of a queue in computer science is the same as the idea of the queues to which you are accustomed in everyday life. There are queues of customers in a bank or in a grocery store and queues of cars waiting to pass through a tollbooth. 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.

$$FRONT = FRONT + 1$$

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

$$REAR = REAR + 1$$

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

$$QUEUE[REAR] = ITEM$$

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

$$FRONT \text{ to } N+1.$$

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

$$FRONT = REAR \neq NULL$$

And suppose that the element is deleted. Then we assign $FRONT = NULL$ and $REAR = NULL$ to indicate that the queue is empty.

Algorithm for insertion into the Queue: -
 QINSERT (QUEUE, N, FRONT, REAR, ITEM)
 This procedure inserts an element ITEM into a queue.

```

1.[Queue already filled?]
If FRONT = 1 and REAR = N or if FRONT = REAR+1 then
Write OVERFLOW and Return.
2.[Find new value of REAR]
If FRONT = NULL then [Queue initially empty] Set FRONT =1
and REAR = 1.
    else if REAR = N then
Set REAR = 1.

else
Set REAR = REAR + 1. [End of if
Structure]
3.Set QUEUE[REAR] = ITEM [This insert new element]
4.Return.
```

Algorithm for Deletion from Queues: -

QDELETE (QUEUE, N, FRONT, REAR, ITEM)

This procedure deletes an element from a queue and assigns it to the variable ITEM.

```

1.[Queue already empty?]
If FRONT = NULL then Write Underflow and Return.
2.Set ITEM = QUEUE[FRONT]
```

3.[Find new value of FRONT]

If FRONT = REAR then [Queue has only one element to start]

Set FRONT = NULL and REAR = NULL

else if FRONT = N then

Set FRONT = 1

else

Set FRONT = FRONT + 1 [End of

If Structure]

4.Return.

Lab Task: -

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



```
E:\UOL\Data structure\DSA Lab\lab 2.exe
Press 1 to Enter Data
Press 2 to Display Data
Press 3 to Remove Data
Press 4 to Exit
Enter any number to select:
```

Conclusion

(Concerned Teacher/Lab Engineer)

Code:

```
#include<iostream>
#include<stdio.h>
#include <unistd.h>
#include<cstdlib>
#define SIZE 5
using namespace std;
int Data[SIZE];
int front=-1;
int rear=-1;
void Enter(int m) {
    if(rear>4) {
        cerr<<"Queue is full..!!\n";
        front=rear=-1;
    }else {
        Data[++rear]=m;
        cout<<"\nData is sucessfully Entered..!!\n";
    }
}

void Delete() {
    if(front==rear) {
        cerr<<"Queue is Empty..!!\n";
    }else {
        cout<<"Deleted "<<Data[++front]<<endl;
    }
}

void display()
{
    if(rear==front)
    {
        cout <<" Queue empty\n";
        return;
    }
    for(int i=front+1;i<=rear;i++)
        cout <<Data[i]<<"\n";
}

void list() {
    cout<<"Press 1 to Enter Data"<<endl;
    cout<<"Press 2 to Display Data"<<endl;
    cout<<"Press 3 to Remove Data"<<endl;
    cout<<"Press 4 to Exit"<<endl;
}
```

```

int choice;
string Ask="y";
int input;
int main()
{
do {
system("cls");
list();
cout<<"Enter any number to select:";
cin>>choice;
switch (choice){
case 1:
do {
system("cls");
cout<<"\t\tQueue Data Entry\n\n";
cout<<"Enter a number:"<<endl;
cin>>input;
Enter(input);
cout<<"Want to continue?"<<endl;
cout<<" y/n";
cin>>Ask;
}while(Ask!="n");
break;

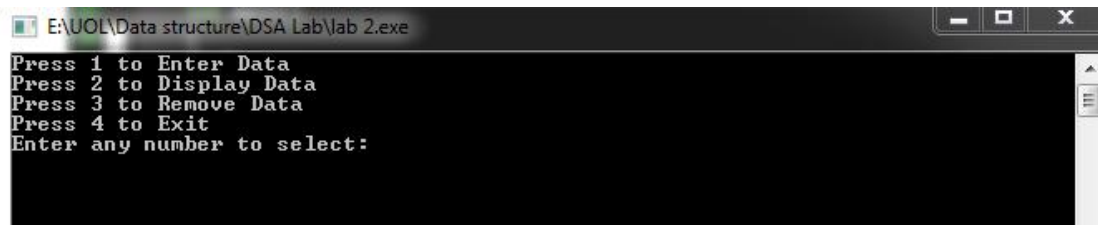
case 2:
do {
system("cls");
cout<<"\t\tQueue Current Data\n\n";
display();
cout<<"Want to continue? "<<endl;
cout<<"y/n";
cin>>Ask;
}while(Ask!="n");
break;

case 3:
do {
system("cls");
cout<<"\t\tData Deleted from Queue\n\n";
Delete();
cout<<"Want to continue?"<<endl;
cout<<"y/n";
cin>>Ask;
}while(Ask!="n");
break;
}
}while(choice!=4);

```

```
return 0;  
}
```

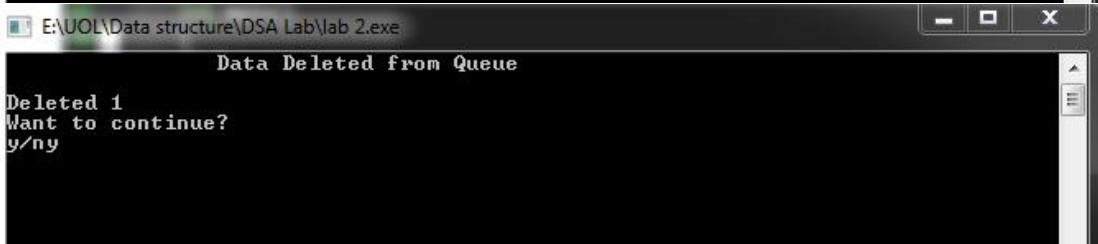
Output:



```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Press 1 to Enter Data  
Press 2 to Display Data  
Press 3 to Remove Data  
Press 4 to Exit  
Enter any number to select:
```



```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Queue Data Entry  
Enter a number:  
3  
Data is sucessfully Entered...!!  
Want to continue?  
y/n
```



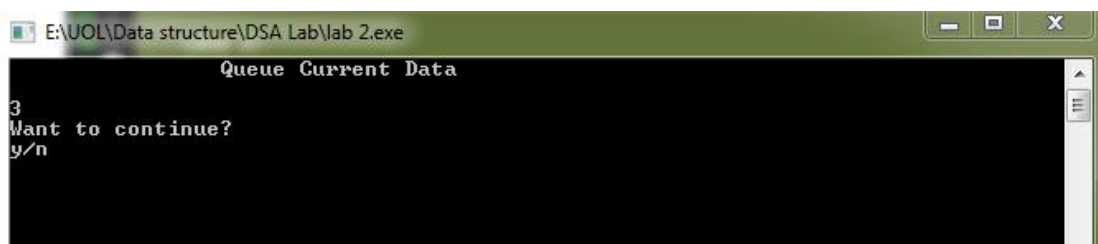
```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Data Deleted from Queue  
Deleted 1  
Want to continue?  
y/ny
```



```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Data Deleted from Queue  
Deleted 2  
Want to continue?  
y/ny
```



```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Queue Current Data  
1  
2  
3  
Want to continue?  
y/n
```



```
E:\UOL\Data structure\DSA Lab\lab 2.exe  
Queue Current Data  
3  
Want to continue?  
y/n
```