

# **Lab Manual for Data Structures**

Lab 4

Queue Implementation in C++

# Table of Contents

1.	Introduction	43
1.1	Queue Operations	43
1.3	Relevant Lecture Readings	43
2.	Activity Time boxing	43
3.	Objectives of the experiment	43
4.	Concept Map	44
4.1	Queue as a static array	44
5.	Homework before Lab	45
5.1	Problem Solution Modeling	46
5.2	Practices from home	46
5.3.1	Task-2	46
6.	Procedure& Tools	46
6.1	Tools	46
6.2	Walk through Tasks [Expected time = 20mins]	46
7.	Practice Tasks	48
7.1	Practice Task 1 [Expected time = 20mins]	48
7.2	Practice Task 2 [Expected time = 20 mins]	49
7.3	Practice Task 3 [Expected time = 20 mins]	50
7.4	Out comes	50
7.5	Testing	50
8.	Evaluation Task (Unseen) [Expected time = 60mins for two tasks]	52
9.	Evaluation criteria	52
10.	Further Readings	52
10.1	Web sites related to C++ tutorials related to queues	52
10.2	Web sites containing supporting material	52

## Lab 4: Stack Applications and Queue Implementation in C++

### 1. Introduction

This lab will help students to learn about using the stacks as postfix calculator and infix to postfix convertor. Also this lab will introduce use of FIFO (First In First Out) Queue and operations which may perform on this queue. A queue is a specific type of abstract data type in which the elements are kept in order and the only operations on the collection are the addition of elements to the rear terminal position, which is called addQueue or enqueue, and removals of elements from the front terminal position, known as removeQueue or dequeue. In a FIFO data structure, the first element which is added to the queue will be the first element to be removed from queue.

#### 1.1 Queue Operations

Two key operations which are defined for queues are addQueue (enqueue) and deleteQueue (dequeue). As elements cannot be deleted from an empty queue and cannot be added to a full queue, therefore we need two more operations called IsEmptyQueue (to check whether a queue is empty) and IsFullQueue (to check whether a queue is full). Another operation which is required for queues is initializeQueue: to initialize a queue in empty state, it means it should not contain any elements in it when queue is created. To retrieve the front and last elements of queue we need to have two operations called front and last operations.

#### 1.3 Relevant Lecture Readings

- Revise Lecture No. 9 and 10 available at \\dataserver\jinnah\$\ in instructor's folder.
- From books: C++ Data Structures by Nell Dale (Page 199-210) and Data structures using C++ by D. S. Malik (Page 428-437).
- From books: C++ Data Structures by Nell Dale (Page 225-245) and Data structures using C++ by D. S. Malik (Page 455-462).

### 2. Activity Time boxing

Table 1: Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5.1	Design Evaluation	15mins	15mins
6.2	Walk through tasks	15mins	15mins
7	Practice tasks	20 mins for task 1, 30mins for task 2 and 30 mins for task 3.	80mins
9	Evaluation Task	60mins for all assigned tasks	60mins

### 3. Objectives of the experiment

- To get knowledge of implementing queues using static arrays in C++.
- To understand the operations related to queues in C++.

## 4. Concept Map

This concept map will help students to understand the main concepts of topic covered in lab.

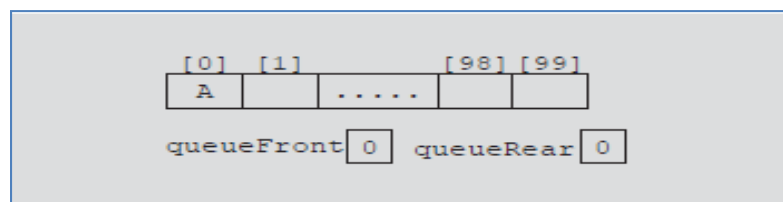
### 4.1 Queue as a static array

We will be using an array to store the queue elements, the variables `queueFront` and `queueRear` will be used to keep track of the first and last elements of the queue, and the variable `maxQueueSize` is used to specify the maximum size of the queue. Thus, we need at least four member variables in class for queue.

Before writing the algorithms for queue operations, we need to decide that how `queueFront` and `queueRear` will be used to access the queue elements. How `queueFront` and `queueRear` indicate that status of queue that the queue is empty or full? If `queueFront` gives the index of the first element of the queue and `queueRear` gives the index of the last element of the queue. To add an element to the queue, first we advance `queueRear` to the next array position and then add the element to the position that `queueRear` is pointing to. To delete an element from the queue, first we retrieve the element that `queueFront` is pointing to and then advance `queueFront` to the next element of the queue. It means `queueFront` will be changing after each `deleteQueue` operation and `queueRear` will be changing after each `addQueue` operation.

Let us see what will happen when `queueFront` changes after a `deleteQueue` operation and `queueRear` changes after an `addQueue` operation. Assume that size of array to holding queue elements is 100.

Initially, the queue is empty. After the operation: `addQueue (Queue, 'A') ;`  
Figure 1 shows the effect of this operation on array.



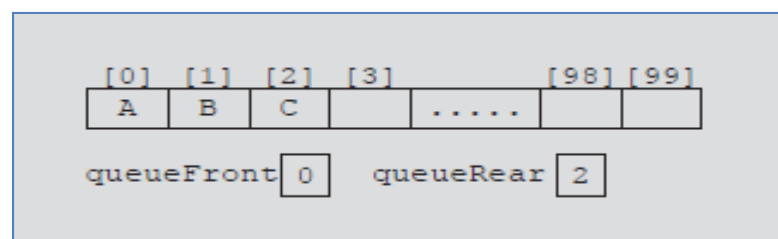
**Figure 1: Insertion of an element in queue at rear.**

After two more `addQueue` operations:

`addQueue (Queue, 'B') ;`

`addQueue (Queue, 'C') ;`

The array is as shown in Figure 2 after the effect of these two operations.

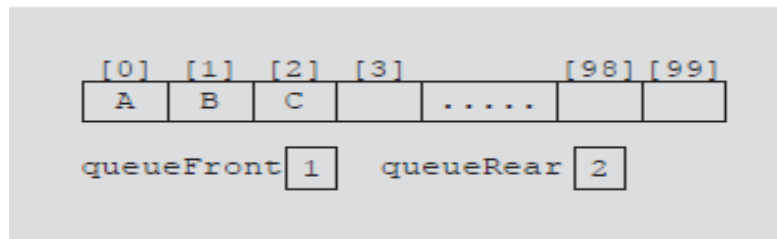


**Figure 2: Insertion of two more elements in queue at rare.**

Now consider the deleteQueue operation:

```
deleteQueue();
```

After this operation, the array containing the queue is as shown in Figure 3.

**Figure 3: Removal of element from queue at front.**

An array for a queue is normally initialized by setting queueFront and queueRear to -1.

```
queueRear=-1; queueFront=-1;
```

When an element is required to be added in queue, we need to check that whether queue overflow condition occurs or not, it means whether queue will exceed its maximum size on addition of an element to queue or not? If x is an element to be inserted in a queue based on array “Queue” with a maximum size MAX\_QUEUE, then following code segment represents implementation of addQueue or enqueue operation.

```
if(queueRear> MAX_QUEUE-1)
{ cout<<"queue over flow";
queueFront=queueRear=-1;
return; }
Queue[++queueRear]=x;
cout<<"inserted" <<x;
```

When an element is required to be remove from a queue, we need to check that whether queue is empty or not, because we cannot remove an element from an empty queue. Following code segment represents this operation.

```
if(queueFront==queueRear)
{ cout<<"queue under flow"; return; }
cout<<"deleted" <<Queue[++queueFront];
```

## 5. Homework before Lab

This homework will help students to study the concepts of topic before start of lab.

### 5.1 Problem Solution Modeling

After studying the introduction and concept map sections you should be ready to provide the solution of following problems. Design the solutions of the problems in C++ and bring your code to lab so that lab instructor should assess and grade it.

### 5.2 Practices from home

#### 5.3.1 Task-2

Compare implementations of stack with a queue using static array and provide at least three differences between these two implementations.

## 6. Procedure & Tools

This section provides information about tools and programming procedures used for the lab.

### 6.1 Tools

Microsoft Visual Studio 2017 with Visual C++ compiler configured.

### 6.2 Walk through Tasks

[Expected time = 20mins]

In this task we will implement and test the evaluation of post fix expression and a program for a queue using static array using C++. Figure 4 and 5 represents the source code view of queue program in visual studio. You are required to type this code and further debug and execute the program.

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class queue {
    int queue1[5];
    int rear,front;
public: queue()
    { rear=-1;
      front=-1; }
    void insert(int x)
    { if(rear > 4)
      { cout <<"queue over flow";
        front=rear=-1;
        return;
      }
      queue1[++rear]=x;
      cout <<"inserted" <<x;
    }
    void delet()
    { if(front==rear)
      { cout <<"queue under flow";
        return; }
      cout <<"deleted" <<queue1[++front];
    }
}
```

Figure 4: Implementation of a queue using static array.

Figure 5 also represents the remaining part of this code.

```

    }
    void display()
    { if(rear==front)
      { cout <<" queue empty";
        return; }
      for(int i=front+1;i<=rear;i++)
        cout <<queue1[i]<<" ";
    }
};

void main() { int ch;
             queue qu;
             while(1)
             { cout <<"\n1.insert 2.delet 3.display 4.exit\nEnter ur choice";
               cin >> ch;
               switch(ch)
               { case 1: cout <<"enter the element"; cin >> ch; qu.insert(ch); break;
                 case 2: qu.delet(); break;
                 case 3: qu.display();break;
                 case 4: exit(0);
               }
             }
             return (0);
}

```

Figure 5: Implementation of a queue using static array

Following is output window of this program in figure 4 and figure 5, when executed.

```

1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element3
inserted3
1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element5
inserted5
1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element7
inserted7
1.insert 2.delet 3.display 4.exit
Enter ur choice3
3 5 7
1.insert 2.delet 3.display 4.exit
Enter ur choice2
deleted3
1.insert 2.delet 3.display 4.exit
Enter ur choice3
5 7
1.insert 2.delet 3.display 4.exit
Enter ur choice2
deleted5
1.insert 2.delet 3.display 4.exit
Enter ur choice3
7
1.insert 2.delet 3.display 4.exit
Enter ur choice2
deleted7
1.insert 2.delet 3.display 4.exit
Enter ur choice3
queue empty
1.insert 2.delet 3.display 4.exit
Enter ur choice_

```

Figure 6: Output of program for queue using static array.

## 7. Practice Tasks

This section will provide information about the practice tasks which are required to be performed in lab session. Design solutions of problems discussed in each task and place solution code in a folder specified by your lab instructor.

**Lab instructors are guided to help students learn how ACM problems work and provide students with certain practice/home tasks on the pattern of ACM Problems.**

### 7.1 Practice Task 1

[Expected time = 20mins]

Partial solution of a Linear Queue Class is given below. Complete the missing code in the functions to run the program properly.

```

Class Queue
{
int *elements;
int size;
int rear, front;
////////// Constructor
Queue (int maxsize)
{
    size=maxsize;
    elements=new int[size];
    rear = 0, front = -1;
}
////////// Destructor
~ Queue()
{
    delete []elements;
}
////////// Output the Queue structure
void showStructure ()
{
    if(!isEmpty())
    {
        for(int i=0;i<rear;i++)
        {
            cout<<"Element:"<<elements[i]<<endl;
        }
    }
    else
    {
        cout<<"Display: No items to be displayed. Queue is empty\n";
    }
}
////////// Queue manipulation operations
// Insert in queue
void Enqueue (int newDataItem)
{
    if(!isFull())

```

```

    {
        if(front==-1)
            {front=0;}
        elements[rear]=newDataItem;
        rear++;
    }
    else
    {
        cout<<"Insert: Cannot insert more items. List is full\n";
    }
}
// Remove data item
void Dequeue ()
{
    if(!isEmpty())
    {
        //implement your logic here
        // delete a data item from front end and shift all the remaining items to fill the gap at
        the front.
    }
    else
    {
        cout<<"Remove: Cannot remove the item. List is empty\n";
    }
}
// Clear queue
void clear ()
{
    //implement your logic here
    // Removes all the data items in a queue but the queue itself
}
//////////////////// Queue status operations

// check if queue is empty
bool isEmpty ()
{
    // implement your logic here
}

// check if queue is full
bool isFull ()
{
    // implement your logic here
}
};

```

## 7.2 Practice Task 2

**[Expected time = 20 mins]**

Write a program to implement Parking area reservation. The parking area has only one way out and one-way inn; so scheduling has to be done for the arrival and departure of vehicles in the parking area. At one time, only one vehicle can arrive or leave the area. Scheduling has to be done on the basis of first come, first serve. Whichever vehicle comes first at the parking area will be given the space in the parking area. Similarly, whichever vehicle requests for departure first, will

be given the space to the exit.

[Hint: Arrival of vehicle=enqueue, Departure of vehicle=dequeue]

### 7.3 Practice Task 3

[Expected time = 20 mins]

A queue can be used to simulate the flow of customers through a check-out line in a departmental store. Your program must store Name, Customer\_id and Bill of each customer. You can model the flow of customers using a queue in which each data item corresponds to a customer in the line. Customers come one by one and leave the queue after check-out. Each customer takes approx. 5 minutes to check out. Implement the given system and provide the following functionality to user:

- 1) Add a customer to queue
- 2) Delete a customer from queue after check out
- 3) Calculate the total number of customers served
- 4) Calculate the time required to serve all the remaining customers

### 7.4 Out comes

After completing this lab, student will be able to understand and develop programs related to exception handling and use of stacks for evaluation of postfix arithmetic expressions, validating parenthesis in infix arithmetic expressions in C++ using Microsoft Visual Studio 2017 environment.

### 7.5Testing

#### Test Cases for Practice Task-1

Sample Inputs-1	Sample Outputs-1
Enter values to be inserted for queue	
2	
3	
4	
7	
8	
	Dequeue: 2
	Dequeue: 3
	Dequeue: 4
	Values in queue: 7 8

#### Test Cases for Practice Task-2

Sample Inputs-1	Sample Outputs-1
Insert elements in queue:	
Car ID: SH435	
Car ID: PI567	
Car ID: AB984	
	Element removed from rear of queue is
	Car ID: SH435

### Test Cases for Practice Task-3

#### Sample input output:

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 1  
Enter customer name: Ali  
Press Y/y to continue: y

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 1  
Enter customer name: Ahmed  
Press Y/y to continue: y

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 1  
Enter customer name: Aliya  
Press Y/y to continue: y

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 2  
Ali is removed from the queue  
Press Y/y to continue: y

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 3  
Number of customers served:1  
Press Y/y to continue: y

Press 1 to add new customer to queue  
Press 2 to remove a customer from queue  
Press 3 to Calculate the total number of customers served  
Press 4 to Calculate the time required to serve all the remaining customers  
Enter choice: 4  
Time required to serve all the remaining customers:04 mins  
Press Y/y to continue: n

**8.Evaluation Task (Unseen)****[Expected time = 60mins for two tasks]**

The lab instructor will give you unseen task depending upon the progress of the class.

**9. Evaluation criteria**

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 2: Evaluation of the Lab

Sr. No.	Task No	Description	Marks
1	4	Problem Modeling	20
2	6	Procedures and Tools	10
3	7,8	Practice tasks and Testing	35
4	8.1	Evaluation Tasks (Unseen)	20
5		Comments	5
6		Good Programming Practices	10

**10. Further Readings****10.1 Web sites related to C++ tutorials related to queues**

1. <http://www.cprogramming.com/tutorial/computersciencetheory/queue.html>
3. <https://www.tutorialspoint.com/cplusplus-program-to-implement-queue-using-array>

**10.2 Web sites containing supporting material**

1. [http://thatchna.weebly.com/uploads/4/1/9/3/4193382/std\\_c\\_notes\\_03.pdf](http://thatchna.weebly.com/uploads/4/1/9/3/4193382/std_c_notes_03.pdf)
3. [http://uet.vnu.edu.vn/~chauttm/dsa2012w/slides/Lec04\\_StacksQueues.pdf](http://uet.vnu.edu.vn/~chauttm/dsa2012w/slides/Lec04_StacksQueues.pdf)