Tanzeela Asghar

2021-BSE-032

BSE-3A

SOFTWARE ENGGENERING

DATA STRUCTURES & ALGORITHMS

LAB # 5

(QUEUES)

SUBMITTED TO: ***SIR REHAN AHMED SIDDIQUE***

DATED: ***oct 30,2022***

Task 1:

Give answers to the following.

1. **Show the contents of a (linear) queue and position of front and rear markers (at each step) once the following sequence of statements is executed**

Queue Q;

1. Q.enqueue(10);	front **10** rear
2. Q.enqueue(20);	**10** front | **20** rear
3. Q.enqueue(30);	**10** front | **20** rear | **30** rear
4. Q.dequeue();	| **20** front | **30** rear
5. Q.dequeue();	| | front **30** rear
6. Q.enqueue(40);	| | **30** front | **40** rear
7. Q.dequeue()	| | | **40** rear
8. Q.dequeue()	| | | |

1. **Consider a circular QUEUE with N=8 memory cells. Find the number of elements in QUEUE for the following positions of front and rear.**

front = 0 ; rear = 4 ;	**4**
front = 2 ; rear = 0 ;	**7**
front = 4 ; rear = 6 ; And two elements are dequeued.	**1**

2. **Suppose q is an instance of a circular queue and the queue size is 4. Show the contents of the queue and positions of the front and rear markers once the following sequence of statements is executed. The initial contents of the queue are listed in the following.**

front →

rear →

q.dequeue();		**front 60**	**Rear 80**			
q.dequeue();			**front 80 rear**			
q.enqueue(15);			**front 80**	**rear 15**		
q.enqueue(25);			**front 80**	**15**	**rear 25**	
q.enqueue(105);			**Front 80**	**15**	**25**	**rear 105**

Code Task # 01

Create a class Queue that implements the functionality of a queue providing all the required operations (Enqueue(), Dequeue(), is_Empty(), is_Full(),display(), getFront(), getRare()).

```
// ds lab 5.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include<iostream>
using namespace std;
int const q_size = 10;
typedef int q_element;
class queue
{
private:
        q_element arr[q_size];
  q_element front, rear;
public:                                                queue() //CONSTRUCTOR
                                                       {
                front = rear = -1;
        }

        bool empty() //EMPTY FUNCTION
        {
                if(front == -1 && rear == -1)
                        return true;
                else
                return false;
        }

        bool full()    //FULL FUNCTION
        {
                if(front==rear)
                return true;
                else
                        return false;
        }

        void frontVal() //RETRIEVE FRONT VALUE
  {
        cout<<"\nThe value of front is: "<<arr[front];
        }
```

```
void rearVal() //RETRIEVE REAR VALUE
{
cout<<"\nThe value of rear is: "<<arr[rear];
}

void addQueue(q_element n) //ADD VALUE
{

if(rear == q_size - 1)
{
cout<<"Overflow Condition!\n";
}
else if(empty())
{

front++;
rear++;
arr[rear]=n;
}

else
{

}

r
```

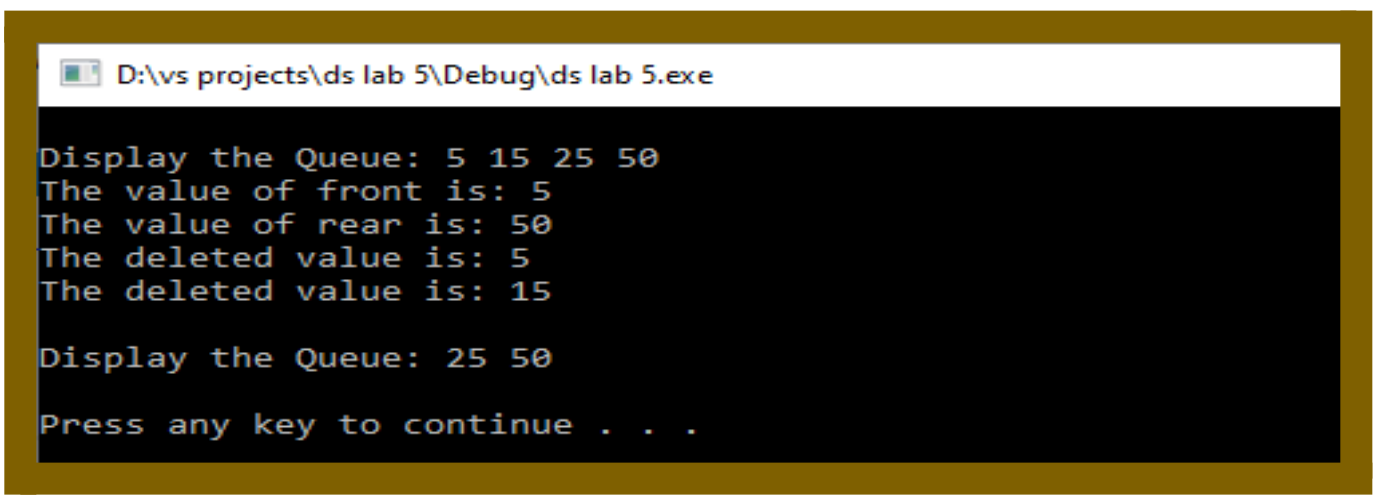
```
e
a
r
+
+
;
a
r
r
[
r
```

```
e
a
r
]
=
n
;

}
```

```
void deQueue()     //DELETE VALUE
        {       if(empty())
        {
                cout<<"Underflow Condition!\n";
        }

        else if(front == rear)
        {
                                                front = rear =-1;
        }

        else
        {
                                                cout<<"\
                                                nThe
                                                deleted
                                                value is:
        }                                       "<<arr[fr
        }                                       ont];
                                                front++;
void print()     //PRINT VALUE
{
```

```
        cout<<"\nDisplay the Queue: ";
                for(int i=front; i<=rear; i++)
                        cout<<arr[i]<<" ";
}

};

int _tmain(int argc, _TCHAR* argv[])
{
        queue q1;
        q1.addQueue(5);
        q1.addQueue(15);
        q1.addQueue(25);
        q1.addQueue(50);
        q1.print();
        q1.frontVal();
        q1.rearVal();
        q1.deQueue();
        q1.deQueue();cout<<endl;
        q1.print();
        cout<<endl;
        cout<<endl;
        system("pause");
        return 0;
}
```

OUTPUT:



40
60
80

Code Task # 02

Create a class Circular Queue that implements the functionality of a queue providing all the required operations (Enqueue(), Dequeue(), Empty(), Full() and getFront()).

```
// ds lab 5.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include<iostream>
using namespace std;
int const q_size = 4;
typedef int q_element;
class queue
{
private:
        q_element arr[q_size];
  q_element front, rear;
public:                                                queue() //CONSTRUCTOR
                                                       {
                front = rear = -1;
        }

        bool empty() //EMPTY FUNCTION
        {
                if(front == -1 && rear == -1)
                        return true;
                else
                return false;
        }

        bool full()    //FULL FUNCTION
        {
                if((rear+1)%q_size== front)
                return true;
                else
                        return false;
        }

        void frontVal() //RETRIEVE FRONT VALUE
  {
        cout<<"\nThe value of front is: "<<arr[front];
        }
```

```
        void rearVal() //RETRIEVE REAR VALUE
  {
        cout<<"\nThe value of rear is: "<<arr[rear];
        }

        void addQueue(q_element n) //ADD VALUE
        {

if(full())
                                                        {
                                                                cout<<"Overflow Condition!\n";
                                                        }
else
{

        rear = (rear + 1)% q_size;
        arr[rear]= n;
}
        }

void deQueue()    //DELETE VALUE
{

        front = (front + 1)% q_size;
        rear--;
}
void print()    //PRINT VALUE
{
        cout<<"\n    Queue: ";
                for(int i=0; i<=rear; i++)
                        cout<<arr[i]<<" ";
}

};

int _tmain(int argc, _TCHAR* argv[])
{
        queue q;

        q.addQueue(10);
        q.addQueue(20);
        q.addQueue(30);
        q.addQueue(40);
        q.print();
        q.deQueu
```

```
q.addQueue(50); q.print();
cout<<endl; cout<<endl;

        system("pause");
        return 0;
    }
```

OUTPUT: