Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_8\_\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Design & implement all methods of Simple Queue |
| 2 | Design & implement all methods of Circular Queue |
| 3 | Design and implement for Priority Queue.  Method 1: Ordering in/ after Enqueue method  Method 2: Separate queues for different priorities |
|  |  |

Submitted On:

30 DEC 2021

(Date: DD/MM/YY

LAB TASK

**Task No. 1 Design & implement all methods of Simple Queue**

**Solution:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace queuepractice

{

class queue

{

int[] arr;

int front;

int rear;

public queue()

{

arr = new int[5];

front = -1;

rear = -1;

}

public bool isempty()

{

if(front<0 || front>rear)

{

Console.WriteLine("Queue is Empty");

return true;

}

else

{

return false;

}

}

public bool isfull()

{

if (rear == arr.Length - 1) {

Console.WriteLine("Queue is full");

return true;

}

else

{

return false;

}

}

public void enqueue(int x)

{

if (rear == arr.Length - 1)

{

Console.WriteLine("Queue is full");

}

else if (front == -1 || rear == -1)

{

front = 0;

rear = 0;

arr[rear] = x;

}

else

{

rear = rear + 1;

arr[rear] = x;

}

}

public void dequeue()

{

if(front<0 || front>rear)

{

Console.WriteLine("Queue is Empty");

}

else if(front==rear)

{

front = rear = -1;

}

else

{

front = front + 1;

}

}

public void count()

{

if (front == -1)

{

Console.WriteLine("Queue is empty");

}

else

{

Console.WriteLine("Front value is " +arr[front]);

}

}

public void print()

{

if (front ==-1)

{

Console.WriteLine("Queue is Empty");

}

else

{

for(int i = front; i <= rear; i++)

{

Console.WriteLine(arr[i]);

}

}

}

}

}

namespace queuepractice

{

class Program

{

static void Main(string[] args)

{

queue ob = new queue();

ob.isempty();

ob.enqueue(9);

ob.enqueue(6);

ob.enqueue(8);

ob.enqueue(1);

ob.enqueue(3);

ob.print();

ob.count();

ob.isfull();

ob.dequeue();

ob.dequeue();

ob.dequeue();

ob.dequeue();

ob.dequeue();

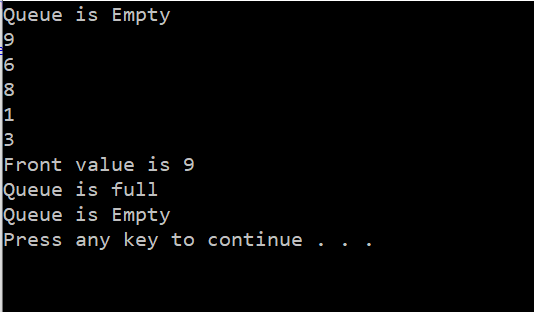
ob.print();

}

}

}

**Output:**



**Task No. 2:** Design & implement all methods of Circular Queue.

**Solution:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace queuepractice

{

class queue

{

int[] arr;

int front;

int rear;

public queue()

{

arr = new int[5];

front = -1;

rear = -1;

}

public bool isempty()

{

if(front<0 || front>rear)

{

Console.WriteLine("Queue is Empty");

return true;

}

else

{

return false;

}

}

public bool isfull()

{

if ((rear+1)%arr.Length==front) {

Console.WriteLine("Queue is full");

return true;

}

else

{

return false;

}

}

public void enqueue(int x)

{

if ((rear + 1) % arr.Length == front)

{

Console.WriteLine("Queue is full");

}

else if (front == -1 || rear == -1)

{

front = 0;

rear = 0;

arr[rear] = x;

}

else

{

rear = (rear + 1)%arr.Length;

arr[rear] = x;

}

}

public void dequeue()

{

if(front<0 || front>rear)

{

Console.WriteLine("Queue is Empty");

}

else if(front==rear)

{

front = rear = -1;

}

else

{

front = (front + 1)%arr.Length;

}

}

public void count()

{

if (front == -1)

{

Console.WriteLine("Queue is empty");

}

else

{

Console.WriteLine("Front value is " +arr[front]);

}

}

public void print()

{

if (front ==-1)

{

Console.WriteLine("Queue is Empty");

}

else if (rear >= front)

{

for (int i = front; i <= rear; i++)

Console.WriteLine( arr[i]);

}

else

{

for (int i = front; i < arr.Length; i++)

{ Console.WriteLine(arr[i]); }

for (int i = 0; i <= rear; i++)

{ Console.WriteLine(arr[i]); }

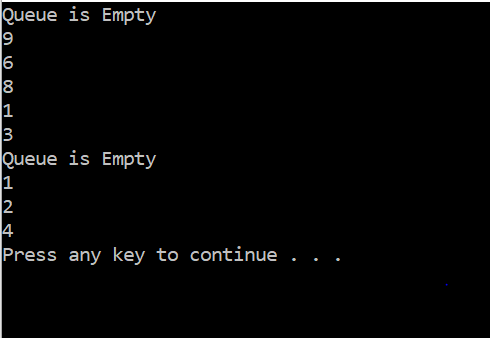
}

}

}

}

**Output:**



**Task No. 3:** Design and implement for Priority Queue.

Method 1: Ordering in/ after Enqueue method

Method 2: Separate queues for different priorities

**Solution:**

class Node

{

public int priority;

public int information;

public Node link;

public Node(int info, int prior)

{

information = info;

priority = prior;

link = null;

}

}

// PriorityQueue Class

class PriorityQueue

{

private Node front;

public PriorityQueue()

{

front = null;

}

//Insert Method

public void Insert(int item, int itemPriority)

{

Node temp, prio;

temp = new Node(item, itemPriority);

if (IsEmpty() || itemPriority < front.priority)

{

temp.link = front;

front = temp;

}

else

{

prio = front;

while (prio.link != null && prio.link.priority <= itemPriority)

prio = prio.link;

temp.link = prio.link;

prio.link = temp;

}

}

// Deletion of element Method

public int Delete()

{

int item;

if ( IsEmpty() )

throw new System.InvalidOperationException("Queue Underflow");

else

{

item = front.information;

front = front.link;

}

return item;

}

// Empty Method

public bool IsEmpty()

{

return (front==null);

}

// Display Method

public void Display()

{

Node prio=front;

if ( IsEmpty())

Console.WriteLine("Queue is empty\n");

else

{

Console.WriteLine("Queue is :");

Console.WriteLine("item Priority");

while (prio!=null)

{

Console.WriteLine(prio.information + " " + prio.priority);

prio=prio.link;

}

}

Console.WriteLine("");

}

}

class Program

{

// Main Method

static void Main(string[] args)

{

int choice, item, itemPriority;

PriorityQueue p = new PriorityQueue();

while (true)

{

Console.WriteLine("1.Insert a new item");

Console.WriteLine("2.Delete an item");

Console.WriteLine("3.Print the queue");

Console.WriteLine("4.Quit");

Console.Write("Enter your choice : ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice ==4)

break;

// Switch Case

switch (choice)

{

case 1:

Console.Write("Please enter any item to be inserted : ");

item = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter it's priority : ");

itemPriority = Convert.ToInt32(Console.ReadLine());

p.Insert(item,itemPriority);

break;

case 2:

Console.WriteLine("Deleted item is: " + p.Delete());

break;

case 3:

p.Display();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

}

}

**Output:**

