Bahria University,

Karachi Campus



COURSE: CSC-221 DATA STRUCTURES AND ALGORITHM

TERM: FALL 2020, CLASS: BSE- 3 (A)

Submitted By:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ADIL WAHEED) (65190)

Submitted To:

Engr. Dr. Farah/ Engr. Ramshaa

Signed Remarks: Score:

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | DATE | LAB NO | LAB OBJECTIVE | SIGN |
| 01 | 1-10-2020 | 01 | ONE AND TWO DIMENSIONAL ARRAY |  |
| 02 | 09-10-20 | 02 | Linear Search & Sorting Algorithms |  |
| 03 | 13-10-20 | 03 | Recusrion |  |
| 04 | 30/10/2020 | 04 | Binary Search Algorithm |  |
| 05 | 30/10/2020 | 05 | Merge Sort |  |
| 06 | 30/10/2020 | 06 | Quick Sort |  |
| 07 | 4/11/2020 | 07 | Stack |  |
| 08 | 12/11/12 | 08 | Queue |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| SNO | DATE | LAB NO | LAB OBJECTIVE | SIGN |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_08\_\_\_\_

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.**   * + 1. **Method 1: Ordering in/ after Enqueue method**     2. **Method 2: Separate queues for different priorities.** |
|  |  |
|  |  |

Submitted On:

\_\_\_\_\_\_\_\_\_\_\_\_

(Date: 12/11/20)

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

**Solution:**

internal class queue

{

int[] a = new int[20];

int front = 0;

int rear = -1;

internal bool isempty()

{

if (front == -1 && rear == -1)

{

Console.WriteLine("Queue Is Empty");

return true;

}

else

{

Console.WriteLine("Queueu is not empty");

return false;

}

}

internal bool isfull()

{

if (rear == a.Length - 1)

{

Console.WriteLine("Queue Is full");

return true;

}

else

{

Console.WriteLine("Queue Is Not full");

return false;

}

}

internal int Enqueue(int data)

{

if (isfull())

return 0;

rear = rear + 1;

a[rear] = data;

Console.WriteLine("{0} Inserted ", data);

return 1;

}

internal int dequeue()

{

if (isempty())

return 0;

int data = a[front];

front = front + 1;

Console.WriteLine("Dequeue deleted:{0}", data);

return 1;

}

}

static void Main(string[] args)

{

//Design & implement all methods of Simple Queue.

queue q = new queue();

q.Enqueue(2);

q.Enqueue(9);

q.Enqueue(12);

q.Enqueue(22);

q.dequeue();

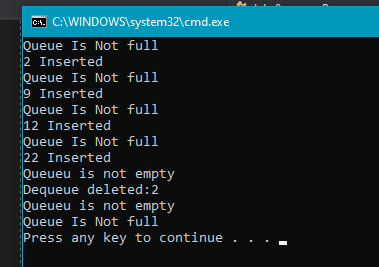
q.isempty();

q.isfull();

}

}

**OUTPUT**:



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

**Solution:**

internal class circularqueue

{

int[] a = new int[20];

int front = 0;

int rear = 0;

internal bool isempty()

{

if (front == -1 && rear == -1)

{

Console.WriteLine("Circular Queue Is Empty");

return true;

}

else

{

Console.WriteLine("Circular Queueu is not empty");

return false;

}

}

internal bool isfull()

{

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

{

Console.WriteLine("Circular Queue Is full");

return true;

}

else

{

Console.WriteLine("Circular Queue Is Not full");

return false;

}

}

internal int Enqueue(int data)

{

if (isfull())

return 0;

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

a[rear] = data;

Console.WriteLine("{0} Inserted ", data);

return 1;

}

internal int dequeue()

{

if (isempty())

return 0;

front = rear = 1;

int data = a[front];

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

Console.WriteLine("Circular Dequeue deleted:{0}", data);

return 1;

}

}

static void Main(string[] args)

{

circularqueue c = new circularqueue();

c.Enqueue(2);

c.Enqueue(9);

c.Enqueue(12);

c.Enqueue(22);

c.dequeue();

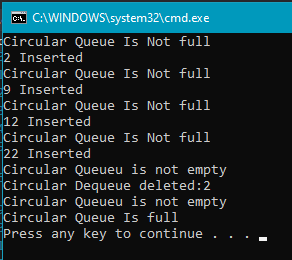
c.isempty();

c.isfull();

}

}

**OUTPUT**:



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

1. **Method 1: Ordering in/ after Enqueue method**
2. **Method 2: Separate queues for different priorities.**

**Solution:**

class Node

{

public int priority;

public int info;

public Node link;

public Node(int i, int pr)

{

info = i;

priority = pr;

link = null;

}

}

class PriorityQueueL

{

private Node front;

public PriorityQueueL()

{

front = null;

}

public void Insert(int element, int elementPriority)

{

Node temp, p;

temp = new Node(element, elementPriority);

/\* Queue is empty or element to be added has priority more than first element \*/

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

{

temp.link = front;

front = temp;

}

else

{

p = front;

while (p.link != null && p.link.priority <= elementPriority)

p = p.link;

temp.link = p.link;

p.link = temp;

}

}

public int Delete()

{

int element;

if (IsEmpty())

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

else

{

element = front.info;

front = front.link;

}

return element;

}

public bool IsEmpty()

{

return (front == null);

}

public void Display()

{

Node p = front;

if (IsEmpty())

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

else

{

Console.WriteLine("Queue is :");

Console.WriteLine("Element Priority");

while (p != null)

{

Console.WriteLine(p.info + " " + p.priority);

p = p.link;

}

}

Console.WriteLine("");

}

}

static void Main(string[] args)

{

int choice, element, elementPriority;

PriorityQueueL pq = new PriorityQueueL();

while (true)

{

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

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

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

Console.WriteLine("4.Quit");

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

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

if (choice == 4)

break;

switch (choice)

{

case 1:

Console.WriteLine("Enter the element to be inserted: ");

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

Console.WriteLine("Enter its priority : ");

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

pq.Insert(element, elementPriority);

break;

case 2:

Console.WriteLine("Deleted element is: " + pq.Delete());

break;

case 3:

pq.Display();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

}

**OUTPUT**:

