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:

Date: 30/12/2021

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

**Solution:**

**Queue Class**

class Queue

{

int front = -1;

int rear = -1;

int[] arr;

public Queue(int size)

{

arr = new int[size];

}

public void Enqueue(int element)

{

if (!isFull())

{

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

{

front++;

}

arr[++rear] = element;

}

}

public bool isEmpty()

{

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

{

Console.WriteLine("Queue is Empty");

return true;

}

return false;

}

public void Dequeue()

{

if (isEmpty())

{

Console.Write(" that's why Dequeue not possible");

return;

}

front++;

}

public int peek()

{

if (!isEmpty())

{

return arr[front];

}

return -1;

}

public bool isFull()

{

if (rear >= arr.Length - 1)

{

Console.WriteLine("Queue is full");

return true;

}

return false;

}

public void Display()

{

if (!isEmpty())

{

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

{

Console.Write(arr[i] + " ");

}

Console.WriteLine();

}

}

}

**Main Method**

static void Main(string[] args)

{

Queue myQueue = new Queue(5);

myQueue.isEmpty();

myQueue.Enqueue(4);

myQueue.Enqueue(14);

myQueue.Enqueue(24);

myQueue.Enqueue(34);

myQueue.Enqueue(44);

myQueue.Display();

myQueue.isFull();

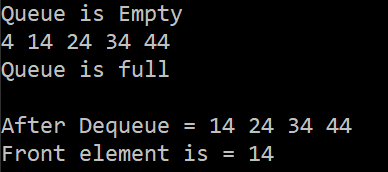
Console.Write("\nAfter Dequeue = ");

myQueue.Dequeue();

myQueue.Display();

Console.WriteLine("Front element is = "+myQueue.peek());

}

**Output:**

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

**Solution:**

**Circular Queue**

internal class Circular\_Queue

{

private int rear, front, length;

int[] a;

public Circular\_Queue(int l)

{

front = -1;

rear = -1;

length = l;

a = new int[length];

}

public bool IsFull()

{

if ((front == 0 && rear == length - 1) || (rear == (front - 1) % (length - 1)))

{

return true;

}

else

{

return false;

}

}

public bool IsEmpty()

{

if (front == -1)

{

return true;

}

else

{

return false;

}

}

public void Enqueue(int x)

{

if (IsFull())

{

Console.WriteLine("Queue is full");

Console.ReadKey();

return;

}

else if (front == -1)

{

front = rear = 0;

a[rear] = x;

}

else if (rear == length - 1 && front != 0)

{

rear = 0;

a[rear] = x;

}

else

{

rear++;

a[rear] = x;

}

}

public void Dequeue()

{

if (IsEmpty())

{

Console.WriteLine("Queue is Empty");

Console.ReadKey();

return;

}

else if (rear == front)

{

front = -1;

rear = -1;

}

else if (front == length - 1)

{

front = 0;

}

else

{

front++;

}

}

public void Display()

{

if (IsEmpty())

{

Console.WriteLine("Queue is Empty");

Console.ReadKey();

return;

}

if (rear >= front)

{

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

{

Console.Write(a[i] + " ");

}

}

else if (rear < front)

{

for (int i = front; i <= length - 1; i++)

{

Console.Write(a[i] + " ");

}

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

{

Console.Write(a[i] + " ");

}

}

else

{

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

{

Console.Write(a[i] + " ");

}

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

{

Console.Write(a[i] + " ");

}

}

Console.ReadKey();

}

public bool Menu()

{

Console.Write("1)Enqueue\t2)Dequeue\t3)Display Queue \t 4)Exit\nChoose:");

switch (Console.ReadLine())

{

case "1":

Console.Write("Enter value to Enqueue : ");

int n = int.Parse(Console.ReadLine());

Enqueue(n);

return true;

case "2":

Dequeue();

return true;

case "3":

Display();

return true;

case "4":

return false;

default:

return true;

}

}

}

**Main method**

static void Main(string[] args)

{

Console.Write("Enter Queue length : ");

int n = int.Parse(Console.ReadLine());

Circular\_Queue cq = new Circular\_Queue(n);

bool m = true;

while (m)

{

m = cq.Menu();

}

}

**Output:**

Graphical user interface, text

Description automatically generated

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

Method 1: Ordering in/ after Enqueue method.

Method 2: Separate queues for different priorities.

**Solution:**

**Priority Queue Class**

internal class Priorty\_Queue

{

int[,] array;

int Length, front, rear;

public Priorty\_Queue(int length)

{

Length = length;

array = new int[length, 2];

front = 0;

rear = 0;

}

public bool SortHigh(int value, int x)

{

int temp = rear - 1;

if (!UnderFlow())

{

for (int i = rear - 1; i >= 0; i--)

{

if (array[i, 0] == 2 || array[i, 0] == 3)

{

array[i + 1, 0] = array[i, 0];

array[i + 1, 1] = array[i, 1];

}

else

{

temp = i;

break;

}

}

if (Enqueue(value, x, temp))

{

return true;

}

else return false;

}

else

return Enqueue(value, x);

}

public int Dequeue()

{

if (!UnderFlow())

{

int val = array[front, 1];

for (int i = front; i < rear - 1; i++)

{

array[i, 0] = array[i + 1, 0];

array[i, 1] = array[i + 1, 1];

}

rear--;

return val;

}

return -1;

}

public int Front()

{

if (!UnderFlow())

{

return array[front, 1];

}

return -1;

}

public bool Enqueue(int val, int j)

{

if (!Overflow())

{

array[rear, 1] = val;

array[rear, 0] = j;

rear++;

return true;

}

return false;

}

public bool Enqueue(int val, int j, int indx)

{

if (!Overflow())

{

if (indx >= 0)

{

array[indx, 1] = val;

array[indx, 0] = j;

rear++;

return true;

}

else

{

array[0, 1] = val;

array[0, 0] = j;

rear++;

return true;

}

}

return false;

}

public bool Overflow()

{

if (rear < Length)

return false;

return true;

}

public bool SortMedium(int value, int x)

{

int temp = rear;

if (!UnderFlow())

{

while (array[temp, 0] == 3 && temp >= 0)

{

if (temp < 0)

{

break;

}

else

{

if (temp < Length - 1)

{

array[temp + 1, 0] = array[temp, 0];

array[temp + 1, 1] = array[temp, 1];

temp--;

}

else

{

break;

}

}

}

if (temp >= 0 && temp < Length)

{

if (Enqueue(value, x, temp))

return true;

else

return false;

}

}

else

return Enqueue(value, x);

return false;

}

public bool UnderFlow()

{

if (rear == front)

return true;

return false;

}

public void Display()

{

Console.WriteLine("---- Here, is Your array -----");

Console.WriteLine("Value\tPriorty");

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

{

Console.WriteLine(array[i, 1] + "\t" + array[i, 0]);

}

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

**Main method**

static void Main(string[] args)

{

Console.Write("Enter length of Queue : ");

int length = Convert.ToInt32(Console.ReadLine());

Priorty\_Queue obj = new Priorty\_Queue(length);

while (true)

{

Console.WriteLine("\nYou want to perform any function on Queue? Press 0 to exit");

Console.Write("1- Enqueue 2- Dequeue 3- Front ? ");

int answer = Convert.ToInt32(Console.ReadLine());

if (answer == 1)

{

int priorty, tempvalue;

Console.Write("\nEnter priorty : ");

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

Console.Write("Enter value : ");

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

if (priorty == 1)

{

if (obj.SortHigh(tempvalue, priorty))

{

Console.WriteLine("Value Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else if (priorty == 2)

{

if (obj.SortMedium(tempvalue, priorty))

{

Console.WriteLine("Value Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else if (priorty == 3)

{

if (obj.Enqueue(tempvalue, priorty))

{

Console.WriteLine("\nValue Enqueued");

obj.Display();

}

else

{

Console.WriteLine("Queue Overflow");

}

}

else

{

Console.WriteLine("Invalid priorty");

}

}

else if (answer == 2)

{

int val = obj.Dequeue();

if (val >= 0)

{

Console.WriteLine("Value successfully dequeued");

Console.WriteLine("Dequeued value = " + val);

}

else

{

Console.WriteLine("Stack Underflow");

}

}

else if (answer == 3)

{

int val = obj.Front();

if (val >= 0)

{

Console.WriteLine("Front value = " + val);

}

else

{

Console.WriteLine("Stack empty");

}

}

else

{

break;

}

}

}

**Output:**

