

2022/2023

Lab 4: Queue

FAKULTI TEKNOLOGI KEJURUTERAAN KELAUTAN DAN INFORMATIK

**DATA STRUCTURE & ALGORITHM**



**VERSION 1**

STUDENT INFORMATION

**Please fill in your details:**

**Name: Arun Mugilan**

**MATRIC NUMBER:s63746**

**GROUP:k2**

**LAB:MP3**

**DATE:22/11/2022**

Table of Contents

[INSTRUCTIONS 1](#_Toc21852310)

[TASK 1: Apply And Test The Implementation Of Queue Using Array 2](#_Toc21852311)

[TASK 2: Observing The Limitation Of A Linear Queue 7](#_Toc21852312)

[TASK3: Implementation Of Circular Queue 9](#_Toc21852313)

# INSTRUCTIONS

Manual makmal ini adalah untuk kegunaan pelajar-pelajar Fakulti Teknologi Kejuruteraan Kelautan dan Informatik, Universiti Malaysia Terengganu (UMT) sahaja. Tidak dibenarkan mencetak dan mengedar manual ini tanpa kebenaran rasmi daripada penulis.

Sila ikuti langkah demi langkah sebagaimana yang dinyatakan di dalam manual.

*This laboratory manual is for use by the students of the Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu (UMT) only. It is not permissible to print and distribute this manual without the official authorisation of the author.*

*Please follow step by step, as described in the manual.*

# TASK 1: Apply and test the implementation of QUEUE USING ARRAY

## Objective

In this task, students must be able to:

* Apply the simple implementation of the queue.
* Test the implementation

## Estimated Time

[60 Minutes]

### definition of queue

Queues are an essential data structure that is found in vast amounts of software from user mode to kernel mode applications that are core to the system. Fundamentally they honour a first in first out (FIFO) strategy, that is the item first put into the queue will be the first served, the second item added to the queue will be the second to be served and so on.

We usually say that elements enter a queue at the back and are removed from the front. A metaphor for this terminology is a line of people waiting to get on an amusement park ride. People waiting for such a ride enter at the back of the line and get on the ride from the front of the line. There are many other applications of queues. Stores, theatres, reservation centres, and other similar services typically process customer requests according to the FIFO principle. A queue would, therefore, be a logical choice for a data structure to handle calls to a customer service centre or a wait-list at a restaurant. FIFO queues are also used by many computing devices, such as a networked printer, or a Web server responding to requests.

### basic operations in queue

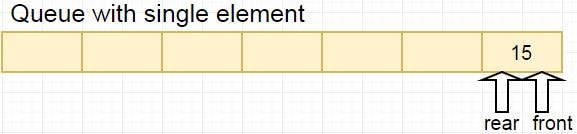
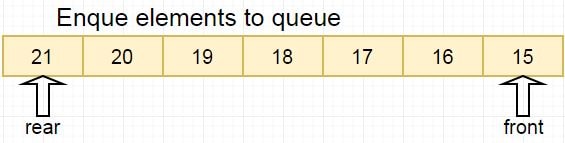
A linear queue only allows you to access the item at the front of the queue; when you add an item to the queue that item is placed at the back of the queue.

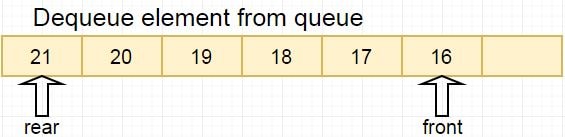
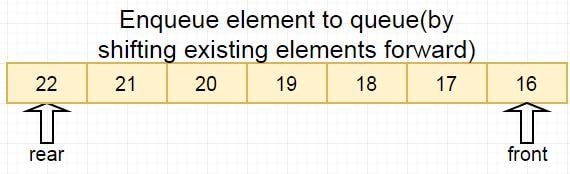
Queues usually have the following three core methods:

**Enqueue:** places an element at the back of the queue.

**Dequeue:** retrieves the element at the front of the queue and removes it from the queue.

**Peek:** retrieves the element at the front of the queue without removing it from the queue.

The Queue in the example below has a single element with value 15. So, both the front and rear point to the same single element:  
  
We now enqueue elements to make the queue full. So, the element with value 21 is our rear element while element with value 15 is our first element as follows:  


Next Dequeue an element with value 15 from the queue. So our first element will now be an element having value 16 as follows:  
  
If elements are now to be added to the queue, space must be made by shifting all the queue elements forward. So, all the elements from 16 to 21 are moved forward to create space for our new rear element 22.  


Based on the above explanation, we can see that the operations of a linear queue involve shifting of elements during the operations. This situation can be avoided by making use of a Circular Queue (to be implemented in Task 3).

### steps:

1. Open Netbeans and create new java application project.
2. Name your project as QueueExperiment and click finish.
3. Change author profiles to :
   1. Name :
   2. Program: <put your program. E.g., SMSK(SE) or SMSK with IM
   3. Course : CSF3104
   4. Lab : <enter lab number>
   5. Date : <enter lab date>
4. In the same QueueExperiment project’s package, create a new file named CharQueue.java and insert the following codes:

A picture containing screenshot

Description automatically generated

A screenshot of a cell phone

Description automatically generated

1. Next, test the class using the following codes. Put the codes in QueueExperiment.java.

A screenshot of a cell phone

Description automatically generated

1. Save, compile and run your source codes.
2. Observe the output and upload the screenshot using the control box below:

Logo, company name

Description automatically generated

1. Now, insert another element ‘k’ in your existing queue. Compile and run your codes again. Do you find any error? Record your output in the following box:

Graphical user interface, text, application, email

Description automatically generated

1. Fix the above code and explain your step.

**Answer:**

QueueExperiment myQueue = new QueueExperiment(11);

Change the 10 to 11 so that “k” can enter queue

1. Next, modify write codes in CharQueue.java to display all elements in the queue.

**Answer: [Copy and paste ALL your java codes (from step 1 until 10) from Netbeans into the following text box]**

**package queueexperiment;**

**public class QueueExperiment {**

**char q[];**

**int front, rear, size;**

**public QueueExperiment(int m){**

**size = m;**

**q = new char[size];**

**front = rear = -1;**

**}**

**void enqueue(char n) {**

**if (isFull()) {**

**System.out.println("Queue is full");**

**} else {**

**if (front == -1) {**

**front = rear = 0;**

**} else {**

**rear++;**

**}**

**q[rear] = n;**

**}**

**}**

**char dequeue() {**

**if (isEmpty()) {**

**System.out.println("Queue is empty");**

**return (char) 250;**

**} else {**

**char x = q[front];**

**front++;**

**return x;**

**}**

**}**

**char peek() {**

**return q[front];**

**}**

**boolean isEmpty() {**

**return (front == -1 || front > rear);**

**}**

**boolean isFull() {**

**return (rear == size - 1);**

**}**

**public static void main(String[] args) {**

**QueueExperiment myQueue = new QueueExperiment(11);**

**myQueue.enqueue('a');**

**myQueue.enqueue('b');**

**myQueue.enqueue('c');**

**myQueue.enqueue('d');**

**myQueue.enqueue('e');**

**myQueue.enqueue('f');**

**myQueue.enqueue('g');**

**myQueue.enqueue('h');**

**myQueue.enqueue('i');**

**myQueue.enqueue('j');**

**myQueue.enqueue('k');**

**System.out.println("\nFirst element in Queue :" + myQueue.peek());**

**}**

**}**

# TASK 2: observing the limiTAtion of a linear queue

## objective

In this task, students will test the dequeue method of the previously created queue and observe the limitation of the linear queue.

## estimated time

[10 Minutes]

### steps:

1. Open queue project that you have created in Task 1.
2. Without re-sizing the size of the array for the queue put the following codes in the main method:

A screenshot of a cell phone

Description automatically generated

1. Put codes to display the elements in the queue as well. Did you see any error? Put your output in the following control box:

Graphical user interface, text, application, email

Description automatically generated

1. Next, replace the code in Step 2 with the following:

A screenshot of a cell phone

Description automatically generated

1. Compile and run the codes. Observe the result. Upload the result using the control box below:

Graphical user interface, text

Description automatically generated

1. Why do you think the situation happens? Give your opinion by investigating the codes in CharQueue.java.

*Hint: Read the notes regarding the Queue*.

**Answer:**

**This uses the concept of FIFO. First element to enter the queue is the first one to get dequeue.**

**[Copy and paste ALL your java codes (from step 2 until 4) from Netbeans into the following text box]**

package queueexperiment;

public class QueueExperiment {

char q[];

int front, rear, size;

public QueueExperiment(int m){

size = m;

q = new char[size];

front = rear = -1;

}

void enqueue(char n) {

if (isFull()) {

System.out.println("Queue is full");

} else {

if (front == -1) {

front = rear = 0;

} else {

rear++;

}

q[rear] = n;

}

}

char dequeue() {

if (isEmpty()) {

System.out.println("Queue is empty");

return (char) 250;

} else {

char x = q[front];

front++;

return x;

}

}

char peek() {

return q[front];

}

boolean isEmpty() {

return (front == -1 || front > rear);

}

boolean isFull() {

return (rear == size - 1);

}

public static void main(String[] args){

QueueExperiment myQueue = new QueueExperiment(10);

myQueue.enqueue('a');

myQueue.enqueue('b');

myQueue.enqueue('c');

myQueue.enqueue('d');

myQueue.dequeue();

myQueue.dequeue();

myQueue.enqueue('e');

myQueue.enqueue('f');

myQueue.enqueue('g');

myQueue.enqueue('h');

myQueue.enqueue('i');

myQueue.enqueue('j');

myQueue.enqueue('k');

myQueue.enqueue('l');

System.out.println("\nElements from Queue :");

System.out.println(myQueue.dequeue());

while (!myQueue.isEmpty()) {

System.out.println(myQueue.dequeue());

}

}

}

# task 3: implementation of circular queue

## objective

In this task, students will implement the circular queue to overcome the limitation of the linear queue that has been coded previously.

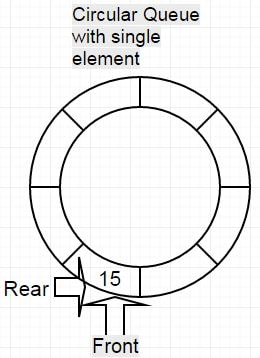
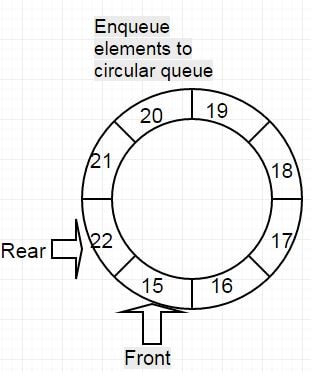
## estimated time

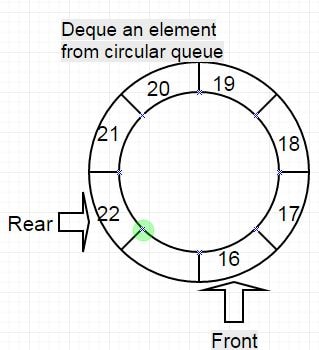
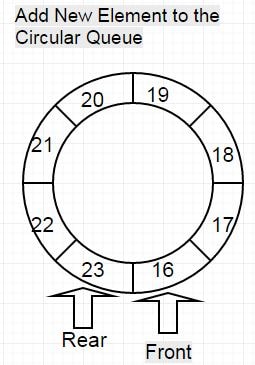
[45 minutes]

### definition of circular queue

A circular queue is an excellent abstraction for applications in which elements are cyclically arranged, such as for multiplayer, turn-based games, or round-robin scheduling of computing processes. In this task, we will implement a demonstration of the use of a circular queue.

### basic operations in circular queue

The circular queue has a single element with value 15. So, both the front and rear point to the same single element:  
  
We now enqueue elements to make the circular queue full. So, the element with value 22 is our rear element while element with value 15 is our first element as follows:  


Next Dequeue an element with value 15 from the circular queue. So our first element will now be an element having value 16 as follows:  
  
If a new element is to be added to the circular queue there is no need to shift the existing elements only the rear pointer will need to be moved to the appropriate location as follows:  


### steps

1. Create a new file in QueueExperiment project. Name the file as CircularQueue.java.
2. We will use a generic class for the implementation of our CircularQueue. Re-type the following codes in your Netbeans editor:

A picture containing screenshot

Description automatically generated

A screenshot of a cell phone

Description automatically generated

1. Next, within the same file (CircularQueue.java.), create the exceptions classes to handle any exceptions that may occur in our Circular Queue. Copy the following codes:

A screenshot of a social media post

Description automatically generated

1. Finally, create a test class to test our Circular Queue implementation.

Note: You may use the main method previously created for Task 1 and Task2 by simply comment on the existing codes.

A screenshot of a social media post

Description automatically generated

1. Compile and run your code. Print screen and upload the output using the following control box:

Text, letter

Description automatically generated

1. Modify the main method to accept String instead of Integer. Use these strings as the input: “Iltizam”, “tekad”, “rajin”, “usaha”, “berjaya”, “konsisten”, “yakin”, “tabah”. After dequeue operation, insert a String “kuat”. Copy your answer into the provided text box:

Text, letter

Description automatically generated

1. **[Copy and paste ALL your java codes (from step 1 until 4) from Netbeans into the following text box]**

package queueexperiment;

import java.util.Arrays;

public class CircularQueue<E> {

private int currentSize;

private E[] circularQueueElements;

private int maxSize;

private int rear;

private int front;

public CircularQueue(int maxSize) {

this.maxSize= maxSize;

circularQueueElements= (E[]) new Object[this.maxSize];

currentSize= 0;

front= -1;

rear= -1;

}

public void enqueue(E item) throws QueueFullException {

if (isFull()) {

throw new QueueFullException("Circlar Queue is full, Element cannot be added");

} else {

rear = (rear + 1) % circularQueueElements.length;

circularQueueElements[rear] = item;

currentSize++;

if (front == -1) {

front = rear;

}

}

}

public E dequeue() throws QueueEmptyException {

E deQueuedElement;

if (isEmpty()) {

throw new QueueEmptyException("Circular Queue is empty. Element cannot be retrieved");

} else {

deQueuedElement = circularQueueElements[front];

circularQueueElements[front] = null;

front = (front + 1) % circularQueueElements.length;

currentSize--;

}

return deQueuedElement;

}

public boolean isFull() {

return (currentSize == circularQueueElements.length);

}

public boolean isEmpty() {

return (currentSize == 0);

}

@Override

public String toString() {

return "Circular Queue (" + Arrays.toString(circularQueueElements) + ")";

}

}

class QueueFullException extends RuntimeException {

private static final long serialVersionUID = 1L;

public QueueFullException() {

super();

}

public QueueFullException(String message) {

super(message);

}

}

class QueueEmptyException extends RuntimeException {

private static final long serialVersionUID = 1L;

public QueueEmptyException() {

super();

}

public QueueEmptyException(String message) {

super(message);

}

}

1. Read the instruction regarding submission carefully. Submit your answer using the link provided at Oceania UMT. Please ensure your codes are submitted to the correct group.