

MIT WORLD PEACE UNIVERSITY

Fundamental Data Structures  
Second Year B. Tech, Semester 1

---

---

JOB SCHEDULING USING CIRCULAR QUEUE

---

---

PRACTICAL REPORT  
ASSIGNMENT 9

Prepared By

Krishnaraj Thadesar  
Cyber Security and Forensics  
Batch A1, PA 20

November 28, 2022

# Contents

<b>1 Aim</b>	<b>1</b>
<b>2 Objectives</b>	<b>1</b>
<b>3 Problem Statements</b>	<b>1</b>
<b>4 Theory</b>	<b>1</b>
4.1 Queue . . . . .	1
4.2 Types of Queues . . . . .	1
4.3 Applications of Queue . . . . .	2
<b>5 Platform</b>	<b>2</b>
<b>6 Input</b>	<b>2</b>
<b>7 Output</b>	<b>2</b>
<b>8 Test Conditions</b>	<b>2</b>
<b>9 Code</b>	<b>3</b>
9.1 Pseudo Code for Add to linear Queue . . . . .	3
9.2 Pseudo Code for Delete to linear Queue . . . . .	3
9.3 Pseudo Code to check isEmpty for linear Queue . . . . .	3
9.4 Pseudo Code for check isFull for linear Queue . . . . .	4
9.5 Pseudo Code for Add to Circular Queue . . . . .	4
9.6 Pseudo Code for Delete to Circular Queue . . . . .	4
9.7 Pseudo Code to check isEmpty for Circular Queue . . . . .	5
9.8 Pseudo Code for check isFull for Circular Queue . . . . .	5
9.9 C Implementation of Problem Statement . . . . .	5
9.10 Input and Output . . . . .	7
<b>10 Time Complexity</b>	<b>11</b>
<b>11 Conclusion</b>	<b>11</b>
<b>12 FAQs</b>	<b>11</b>

### 1 Aim

Writing a C Program to simulate job scheduling using a linear queue.

### 2 Objectives

1. To study Queue and its operations
2. To study the importance of queue as a data structure in computer science

### 3 Problem Statements

*Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write a program for simulating job queue. Write functions to add job and delete job from queue.*

### 4 Theory

#### 4.1 Queue

A queue is a linear data structure that follows the FIFO (First In First Out) principle. It is a simple data structure that allows adding and removing elements in a particular order. A real-life example of a queue is a line of people at a ticket counter. The first person in the line is the first one to get the ticket and the last person in the line is the last one to get the ticket. The elements are added at the end of the queue and are removed from the front of the queue. The operations that can be performed on a queue are:

1. Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
2. Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
3. Front: Get the front item from queue.
4. Rear: Get the last item from queue.

#### 4.2 Types of Queues

1. **Linear Queue:** The elements are stored in a linear fashion. The elements are added from one end and removed from the other end. The insertion and deletion operations are performed at the two ends of the queue. The insertion operation is called enqueue and the deletion operation is called dequeue.
2. **Circular Queue:** The elements are stored in a circular fashion. The elements are added from one end and removed from the other end. The insertion and deletion operations are performed at the two ends of the queue. The insertion operation is called enqueue and the deletion operation is called dequeue.

3. **Priority Queue:** This queue is a special type of queue. Its specialty is that it arranges the elements in a queue based on some priority. The priority can be something where the element with the highest value has the priority so it creates a queue with decreasing order of values. The priority can also be such that the element with the lowest value gets the highest priority so in turn it creates a queue with increasing order of values.
4. **Deque:** Dequeue is also known as Double Ended Queue. As the name suggests double ended, it means that an element can be inserted or removed from both the ends of the queue unlike the other queues in which it can be done only from one end. Because of this property it may not obey the First In First Out property.

### 4.3 Applications of Queue

1. **CPU Scheduling:** The operating system uses a queue to keep track of the processes that are in the ready state. The process that is at the front of the queue is the one that is currently being executed by the CPU. The process that is at the rear of the queue is the one that is waiting to be executed.
2. **Disk Scheduling:** The operating system uses a queue to keep track of the disk requests that are waiting to be serviced. The request that is at the front of the queue is the one that is currently being serviced by the disk. The request that is at the rear of the queue is the one that is waiting to be serviced.
3. **Call Center Phone Systems:** The phone system uses a queue to keep track of the incoming phone calls. The call that is at the front of the queue is the one that is currently being serviced by the phone system. The call that is at the rear of the queue is the one that is waiting to be serviced.

## 5 Platform

**Operating System:** Arch Linux x86-64

**IDEs or Text Editors Used:** Visual Studio Code

**Compilers :** gcc on linux for C

## 6 Input

The element or Job number to pass to the queue

## 7 Output

The Queue

## 8 Test Conditions

DeleteQ(), AddQ(), AddQ(),delete(),delete(),delete()

## 9 Code

### 9.1 Pseudo Code for Add to linear Queue

```
1 AddQ()
2 {
3     if (rear == MAX-1)
4     {
5         printf("Queue Overflow");
6     }
7     else
8     {
9         if (front == -1)
10        {
11            front = 0;
12        }
13        printf("Enter the value to be added in the queue : ");
14        scanf("%d", &add_item);
15        rear = rear + 1;
16        queue_array[rear] = add_item;
17    }
18 }
```

### 9.2 Pseudo Code for Delete to linear Queue

```
1
2 DeleteQ()
3 {
4     if (front == - 1 || front > rear)
5     {
6         printf("Queue Underflow \n");
7         return ;
8     }
9     else
10    {
11        printf("Element deleted from queue is : %d\n", queue_array[front]);
12        front = front + 1;
13    }
14 }
```

### 9.3 Pseudo Code to check isEmpty for linear Queue

```
1
2 isEmpty()
3 {
4     if (front == -1)
5     {
6         printf("Queue is empty");
7     }
8     else
9     {
10        printf("Queue is not empty");
11    }
12 }
```

### 9.4 Pseudo Code for check isFull for linear Queue

```
1
2 isFull()
3 {
4     if (rear == MAX-1)
5     {
6         printf("Queue is full");
7     }
8     else
9     {
10        printf("Queue is not full");
11    }
12 }
```

### 9.5 Pseudo Code for Add to Circular Queue

```
1
2 AddQ()
3 {
4     if ((rear + 1) % MAX == front)
5     {
6         printf("Queue Overflow \n");
7     }
8     else
9     {
10        if (front == -1)
11        {
12            front = 0;
13        }
14        printf("Inset the element in queue : ");
15        scanf("%d", &add_item);
16        rear = (rear + 1) % MAX;
17        queue_array[rear] = add_item;
18    }
19 }
```

### 9.6 Pseudo Code for Delete to Circular Queue

```
1
2 DeleteQ()
3 {
4     if (front == -1)
5     {
6         printf("Queue Underflow \n");
7     }
8     else
9     {
10        printf("Element deleted from queue is : %d\n", queue_array[front]);
11        if (front == rear)
12        {
13            front = -1;
14            rear=-1;
15        }
16        else
17        {
```

```
18     front = (front + 1) % MAX;
19 }
20 }
21 }
```

### 9.7 Pseudo Code to check isEmpty for Circular Queue

```
1
2 isEmpty()
3 {
4     if (front == -1)
5     {
6         printf("Queue is empty");
7     }
8     else
9     {
10        printf("Queue is not empty");
11    }
12 }
```

### 9.8 Pseudo Code for check isFull for Circular Queue

```
1
2 isFull()
3 {
4     if ((rear + 1) % MAX == front)
5     {
6         printf("Queue is full");
7     }
8     else
9     {
10        printf("Queue is not full");
11    }
12 }
```

### 9.9 C Implementation of Problem Statement

```
1 // Queue thing
2 #include <stdio.h>
3 #define MAX_SIZE 5
4 int front = -1, rear = -1;
5
6 int queue[MAX_SIZE];
7
8 int isFull(void)
9 {
10     if (rear == MAX_SIZE - 1)
11         return (1);
12     else
13         return (0);
14 }
15
16 int isEmpty(void)
17 {
18     if (rear == front)
```

```
19 {
20     return (1);
21 }
22 else
23 {
24     return (0);
25 }
26 }
27
28 int enqueue(int item)
29 {
30     if (!isFull())
31     {
32         rear++;
33         queue[rear] = item;
34     }
35     else
36     {
37         printf("\nQUEUE OVERFLOW!\n");
38     }
39 }
40
41 int dequeue(void)
42 {
43     if (isEmpty())
44     {
45         printf("Queue is Empty \n\n QUEUE UNDERFLOW!!\n\n");
46         return (-1);
47     }
48     else
49     {
50         // printf("Removed this thing %c\n", stack[top]);
51         front++;
52         return (queue[front]);
53     }
54 }
55 void display_queue(void)
56 {
57     int i;
58
59     if (isEmpty())
60     {
61         printf("\n\nQueue is empty\n\n");
62     }
63     else
64     {
65         printf("Queue is: \n");
66         for (i = front + 1; i <= rear; i++)
67         {
68             printf("%d \n", queue[i]);
69         }
70     }
71 }
72 int main(void)
73 {
74     int choice;
75     int temp;
76
77     choice = 0;
```



```
78 while (choice != 6)
79 {
80     printf("Enter what you want to do: \n\
81     1. Add Job to the Ready Queue\n\
82     2. Kill or Terminate Job from the Queue\n\
83     3. See the Queue\n\
84     4. Check if Ready Queue is Empty and Processor is free\n\
85     5. Check if Ready Queue is full\n\
86     6. Exit\n\n");
87     scanf("%d", &choice);
88     switch (choice)
89     {
90     case 1:
91         printf("Enter the Process Number of the Job you want to add. \n");
92         scanf(" %d", &temp);
93         enqueue(temp);
94         display_queue();
95         break;
96     case 2:
97         printf("Killing or Terminating Job from the Queue\n");
98         dequeue();
99         display_queue();
100        break;
101     case 3:
102         display_queue();
103         break;
104     case 4:
105         if (isEmpty())
106         {
107             printf("Yup, Queue is empty\n");
108         }
109         else
110         {
111             printf("Nope Queue isnt empty\n");
112             display_queue();
113         }
114         break;
115     case 5:
116         if (isFull())
117         {
118             printf("\nYes the Queue is full!\n");
119         }
120         else
121         {
122             printf("No Queue isnt full!\n");
123         }
124         break;
125     default:
126         printf("\nThank You\n");
127         break;
128     }
129 }
130 return 0;
131 }
```

Listing 1: Main.Cpp

### 9.10 Input and Output

## FDS Assignment 9

---

```
1 Enter what you want to do:
2     1. Add Job to the Ready Queue
3     2. Kill or Terminate Job from the Queue
4     3. See the Queue
5     4. Check if Ready Queue is Empty and Processor is free
6     5. Check if Ready Queue is full
7     6. Exit
8
9 1
10 Enter the Process Number of the Job you want to add.
11 1
12 Queue is:
13 1
14 Enter what you want to do:
15     1. Add Job to the Ready Queue
16     2. Kill or Terminate Job from the Queue
17     3. See the Queue
18     4. Check if Ready Queue is Empty and Processor is free
19     5. Check if Ready Queue is full
20     6. Exit
21
22 1
23 Enter the Process Number of the Job you want to add.
24 2
25 Queue is:
26 1
27 2
28 Enter what you want to do:
29     1. Add Job to the Ready Queue
30     2. Kill or Terminate Job from the Queue
31     3. See the Queue
32     4. Check if Ready Queue is Empty and Processor is free
33     5. Check if Ready Queue is full
34     6. Exit
35
36 1
37 Enter the Process Number of the Job you want to add.
38 3
39 Queue is:
40 1
41 2
42 3
43 Enter what you want to do:
44     1. Add Job to the Ready Queue
45     2. Kill or Terminate Job from the Queue
46     3. See the Queue
47     4. Check if Ready Queue is Empty and Processor is free
48     5. Check if Ready Queue is full
49     6. Exit
50
51 1
52 Enter the Process Number of the Job you want to add.
53 5
54 Queue is:
55 1
56 2
57 3
58 5
59 Enter what you want to do:
```

## *FDS Assignment 9*

---

```
60     1. Add Job to the Ready Queue
61     2. Kill or Terminate Job from the Queue
62     3. See the Queue
63     4. Check if Ready Queue is Empty and Processor is free
64     5. Check if Ready Queue is full
65     6. Exit
66
67 2
68 Killing or Terminating Job from the Queue
69 Queue is:
70 2
71 3
72 5
73 Enter what you want to do:
74     1. Add Job to the Ready Queue
75     2. Kill or Terminate Job from the Queue
76     3. See the Queue
77     4. Check if Ready Queue is Empty and Processor is free
78     5. Check if Ready Queue is full
79     6. Exit
80
81 2
82 Killing or Terminating Job from the Queue
83 Queue is:
84 3
85 5
86 Enter what you want to do:
87     1. Add Job to the Ready Queue
88     2. Kill or Terminate Job from the Queue
89     3. See the Queue
90     4. Check if Ready Queue is Empty and Processor is free
91     5. Check if Ready Queue is full
92     6. Exit
93
94 2
95 Killing or Terminating Job from the Queue
96 Queue is:
97 5
98 Enter what you want to do:
99     1. Add Job to the Ready Queue
100    2. Kill or Terminate Job from the Queue
101    3. See the Queue
102    4. Check if Ready Queue is Empty and Processor is free
103    5. Check if Ready Queue is full
104    6. Exit
105
106 1
107 Enter the Process Number of the Job you want to add.
108 5
109 Queue is:
110 5
111 5
112 Enter what you want to do:
113     1. Add Job to the Ready Queue
114     2. Kill or Terminate Job from the Queue
115     3. See the Queue
116     4. Check if Ready Queue is Empty and Processor is free
117     5. Check if Ready Queue is full
118     6. Exit
```

## FDS Assignment 9

---

```
119
120 1
121 Enter the Process Number of the Job you want to add.
122 65
123
124 QUEUE OVERFLOW!
125 Queue is:
126 5
127 5
128 Enter what you want to do:
129     1. Add Job to the Ready Queue
130     2. Kill or Terminate Job from the Queue
131     3. See the Queue
132     4. Check if Ready Queue is Empty and Processor is free
133     5. Check if Ready Queue is full
134     6. Exit
135
136 5
137
138 Yes the Queue is full!
139 Enter what you want to do:
140     1. Add Job to the Ready Queue
141     2. Kill or Terminate Job from the Queue
142     3. See the Queue
143     4. Check if Ready Queue is Empty and Processor is free
144     5. Check if Ready Queue is full
145     6. Exit
146
147 4
148 Nope Queue isnt empty
149 Queue is:
150 5
151 5
152 Enter what you want to do:
153     1. Add Job to the Ready Queue
154     2. Kill or Terminate Job from the Queue
155     3. See the Queue
156     4. Check if Ready Queue is Empty and Processor is free
157     5. Check if Ready Queue is full
158     6. Exit
159
160 2
161 Killing or Terminating Job from the Queue
162 Queue is:
163 5
164 Enter what you want to do:
165     1. Add Job to the Ready Queue
166     2. Kill or Terminate Job from the Queue
167     3. See the Queue
168     4. Check if Ready Queue is Empty and Processor is free
169     5. Check if Ready Queue is full
170     6. Exit
171
172 2
173 Killing or Terminating Job from the Queue
174
175
176 Queue is empty
177
```

```
178 Enter what you want to do:
179     1. Add Job to the Ready Queue
180     2. Kill or Terminate Job from the Queue
181     3. See the Queue
182     4. Check if Ready Queue is Empty and Processor is free
183     5. Check if Ready Queue is full
184     6. Exit
185
186 1
187 Enter the Process Number of the Job you want to add.
188 1
189
190 QUEUE OVERFLOW!
191
192
193 Queue is empty
194
195 Enter what you want to do:
196     1. Add Job to the Ready Queue
197     2. Kill or Terminate Job from the Queue
198     3. See the Queue
199     4. Check if Ready Queue is Empty and Processor is free
200     5. Check if Ready Queue is full
201     6. Exit
202
203 6
204
205 Thank You
```

Listing 2: Output

## 10 Time Complexity

- AddQ() :  $O(1)$
- DeleteQ() :  $O(1)$
- isEmpty() :  $O(1)$
- isFull() :  $O(1)$

## 11 Conclusion

Thus, implemented Job Scheduling using Linear Queue in C.

## 12 FAQs

### 1. What are the advantages and disadvantages of a linear queue ?

#### Advantages

- (a) A large amount of data can be managed efficiently with ease.
- (b) Operations such as insertion and deletion can be performed with ease as it follows the first in first out rule.
- (c) Queues are useful when a particular service is used by multiple consumers.

- (d) Queues are fast in speed for data inter-process communication.
- (e) Queues can be used in the implementation of other data structures.

**Disadvantages**

- (a) As the insertion in the queue is from the rear end and in the case of Linear Queue of fixed size insertion is not possible when rear reaches the end of the queue.
- (b) But in the case of Circular Queue, the rear end moves from the last position to the front position circularly.

**2. What are the advantages and disadvantages of a circular queue ?**

**Advantages**

- (a) Easier for insertion-deletion: In the circular queue, elements can be inserted easily if there are vacant locations until it is not fully occupied, whereas in the case of a linear queue insertion is not possible once the rear reaches the last index even if there are empty locations present in the queue.
- (b) Efficient utilization of memory: In the circular queue, there is no wastage of memory as it uses the unoccupied space, and memory is used properly in a valuable and effective manner as compared to a linear queue.
- (c) Ease of performing operations: In the linear queue, FIFO is followed, so the element inserted first is the element to be deleted first. This is not the scenario in the case of the circular queue as the rear and front are not fixed so the order of insertion-deletion can be changed, which is very useful.

**Disadvantages**

- (a) Circular queue is not suitable for implementing the BFS algorithm as it is not FIFO.
- (b) Circular queue is not suitable for implementing the DFS algorithm as it is not LIFO.
- (c) Searching an element takes  $O(N)$  time.
- (d) Maximum size of a queue must be defined prior.

**3. Give various applications to the queue .**

- (a) Queue used to model real-world situations such as people waiting in line at a bank, airplanes waiting to take off, or data packets waiting to be transmitted over the Internet.
- (b) There are various queues quietly doing their job in your computer's (or the network's) operating system.
- (c) There's a printer queue where print jobs wait for the printer to be available.
- (d) Stores, reservation centers, 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 transaction processing for such applications. For example, it would be a natural choice for handling calls to the reservation center of an airline.
- (e) CPU Scheduling
- (f) Disk Scheduling
- (g) Handling of interrupts in real-time systems

- (h) ATM Booth Line
- (i) ticket Counter Line
- (j) Key press sequence on the keyboard
- (k) CPU task scheduling
- (l) Waiting time of each customer at call centers.