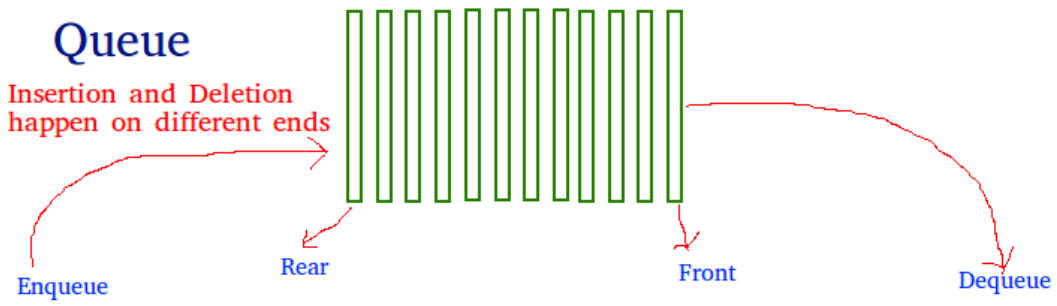
Data Structure

Section #4 "Queues"

Queues

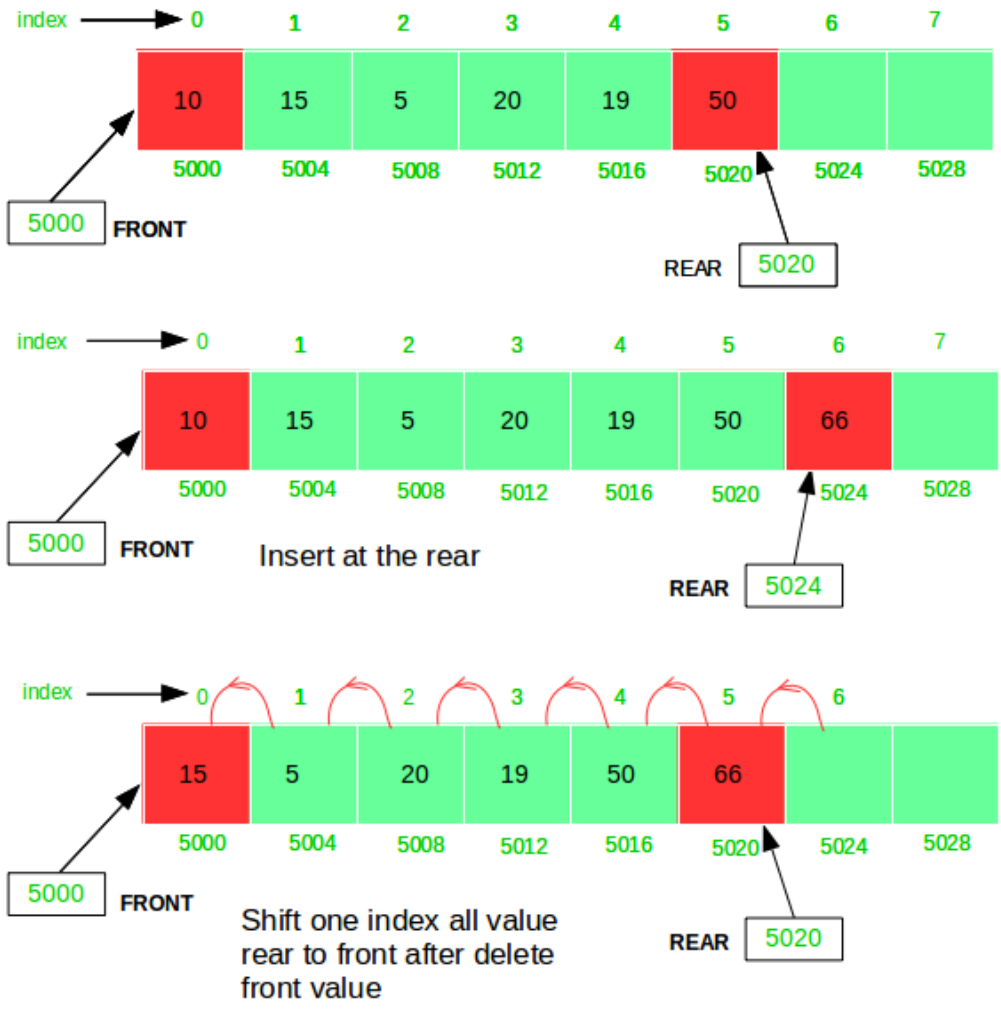
A queue is an abstract data structure that contains a collection of elements. Queue implements the FIFO mechanism i.e. the element that is inserted first is also deleted first. In other words, the least recently added element is removed first in a queue.

A program that implements the queue using an array is given as follows −



Basic Operations

- **Front:** Get the front item from queue.
- **Rear:** Get the last item from queue.
- **enQueue(value)**
- **deQueue()**
- **peek()** − Gets the element at the front of the queue without removing it.
- **isfull()** − Checks if the queue is full.
- **isempty()** − Checks if the queue is empty.



Applications of Queue Data Structure

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.

```
package javaapplication18;
class Queue {
    private static int front, rear, capacity;
    private static int queue[];

    Queue(int c)
    {
        front = rear = 0;
        capacity = c;
        queue = new int[capacity];
    }

    void queueEnqueue(int data)
```

```
    {
        // check queue is full or not
        if (capacity == rear) {
            System.out.printf("\nQueue is full\n");
            return;
        }

        // insert element at the rear
        else {
            queue[rear] = data;
            rear++;
        }
        return;
    }

    void queueDequeue()
    {
        if (front == rear) {
            System.out.printf("\nQueue is
empty\n");
            return;
        }

        else {
            for (int i = 0; i < rear - 1; i++) {
                queue[i] = queue[i + 1];
            }

            // store 0 at rear indicating there's
no element
            if (rear < capacity)
                queue[rear] = 0;

            // decrement rear
            rear--;
        }
        return;
    }

    void queueDisplay()
    {
        int i;
```

```
          if (front == rear) {
               System.out.printf("\nQueue is
Empty\n");
               return;
          }

          // traverse front to rear and print elements
          for (i = front; i < rear; i++) {
               System.out.printf(" %d <-- ",
queue[i]);
          }
          return;
     }

     void queueFront()
     {
          if (front == rear) {
               System.out.printf("\nQueue is
Empty\n");
               return;
          }
          System.out.printf("\nFront Element is: %d",
queue[front]);
          return;
     }
     void search(int searchItem)
     {
          int counter = 0;
          for (int i = front; i < rear; i++)
          {
               if (queue[i] == searchItem)
               {
                    counter++;
                    break;
               }
          }
          if (counter>0)
               System.out.println("Item exists");
          else

               System.out.println("Item not exists");
     }
```

```
}

public class JavaApplication18 {
     public static void main(String[] args) {
          // Create a queue of capacity 4
          Queue q = new Queue(4);

          // print Queue elements
          q.queueDisplay();

          // inserting elements in the queue
          q.queueEnqueue(20);
          q.queueEnqueue(30);
          q.queueEnqueue(40);
          q.queueEnqueue(50);
          q.search(20);
          // print Queue elements
          q.queueDisplay();

          // insert element in the queue
          q.queueEnqueue(60);

          // print Queue elements
          q.queueDisplay();

          q.queueDequeue();
          q.queueDequeue();
          System.out.printf("\n\nafter two node
deletion\n\n");

          // print Queue elements
          q.queueDisplay();

          // print front of the queue
          q.queueFront();
     }

}
```

Task #4

Ex 1: Write a program to print the size of the queue, the front element of the queue, and the back element of the queue.

Ex 2: Create two queues add elements to both the queues and swap the queues and print the swapped queues.