QUEUE DATA STRUCTURE (LECTURE 2)

Queue Data Structure

- **Queue is a linear data structure which follows FIFO (First in First out) principle in inserting and removing elements.**
- The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- In a queue all insertions are made at the **Rear end** and deletions are made at the **Front end**.
- Ex: a queue in food store

Queue Operations/ Functions

- **insert()** - insert a new element into the queue
- **remove()** - remove and return front element from the queue
- **peekfront()** – return the front element without removing
- **isFull()** - check if queue is full
- **isEmpty()** - check if queue is empty

- Queue is said to be in **Overflow state** when it is completely full and is said to be in **Underflow state** if it is completely empty

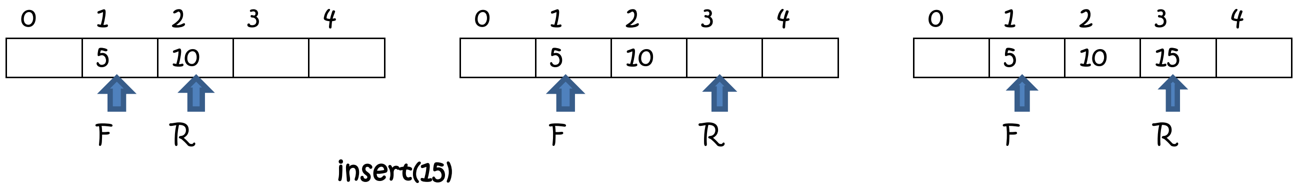
Queue Usage in a Computer System

- **Printer**
- **Word processing:** stores keystroke data as you type at the keyboard
- **Pipeline**

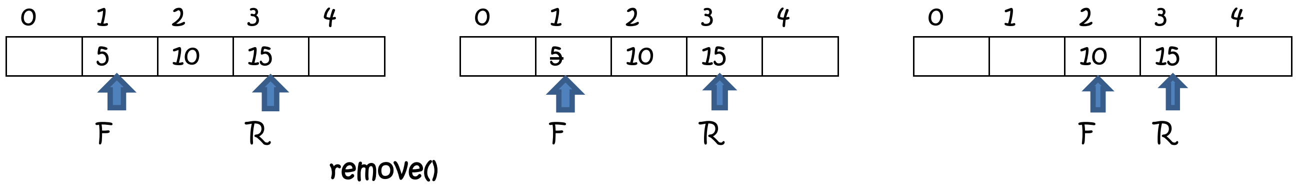
Queue applications

- Queues are more useful in OS:
  - Processor management creates ready queue of the process by the CPU Scheduler Algorithm
  - Batch Processing
  - File Manager
  - Job scheduling & Device scheduling
  - Every IO devices has their own queue to collect requests from different applications or processing
- Queues are used to traversing all nodes of the graph
- Queues are used to traversing all nodes of the tree
- It is also used in different Artificial Intelligent programs

Queue - insert



Queue - remove



Queue implementation

```
class QueueX {
        private int maxSize;                                 //max number of locations
        private int[] queueArray;                            //array definition
        private int front, rear, noOfItems;                  //index definitions

        public QueueX(int s) {                               //constructor
          maxSize = s;
          queueArray = new int[maxSize];                     //array implementation
          front = noOfItems = 0;
          rear = -1;
        }

        public void insert(int j) {                          //insert method
          if(rear == maxSize-1) {
                    System.out.println("Queue overflow");
          }
          else {
                    queueArray[++rear] = j;
                    noOfItems++;
          }
        }

        public int remove() {                                //remove method
          if(noOfItems == 0) {
                    System.out.println("Queue underflow");
                    return 0;
          }
          else {
                    noOfItems--;
                    return queueArray[front++];
          }
        }

        public int peekfront() {                             //peekfront method
          if(noOfItems == 0) {
                    System.out.println("Queue is empty");
                    return 0;
          }
          else
                    return queueArray[front];
        }

        public boolean isEmpty() {
          return (noOfItems == 0);
        }
        public boolean isFull() {
          return (rear == maxSize-1);
        }
}//end of class
class QueueMain{                                                       //main class
        public static void main(String[] args) {
          QueueX q = new QueueX(5);                                    //instantiation and constructor calling
          q.insert(5);
          q.insert(10);
          q.insert(15);
          System.out.println(q.remove());
          System.out.println(q.peekfront());
        }
}
```

Circular Queue Data Structure

- These are also called ring buffers
- The problem in using the linear queue can be overcome by using circular queue
- When we want to insert a new element we can insert it at the beginning of the queue, if the queue is not full we can make the rear start from the beginning by wrapping around

Circular Queue implementation

```
class QueueX {
        private int maxSize;                    //max number of locations
        private int[] queueArray;               //array definition
        private int front, rear, noOfItems;     //index definitions

        public QueueX(int s) {                  //constructor
          maxSize = s;
          queueArray = new int[maxSize];        //array implementation
          front = noOfItems = 0;
          rear = -1;
        }

        public void insert(int j) {             //insert method
          if(noOfItems == maxSize) {
                System.out.println("Queue overflow");
          }
          else {
                if(rear == maxSize-1)
                        rear =  -1;             //if rear is the last element assign -1 to rear
                queueArray[++rear] = j;         //when rear(-1) incremented it will be 0 [-1+1=0]
                noOfItems++;
          }
        }
        public int remove() {                   //remove method
          if(noOfItems == 0) {
                System.out.println("Queue underflow");
                return 0;
          }
          else {
                noOfItems--;
                int temp = queueArray[front++];
                if(front == maxSize)
                        front = 0;
                return temp;
          }
        }
        public int peekfront() {                //peekfront method //same as queue
          if(noOfItems == 0) {
                System.out.println("Queue is empty");
                return 0;
          }
          else
                return queueArray[front];
        }
        public boolean isEmpty() {
          return (noOfItems == 0);
        }
        public boolean isFull() {
          return (noOfItems == maxSize);
        }
}//end of class
```