Queue

swatimali@somaiya.edu

Outline

- Queue – concept
- Queue ADT
- Queue Types
- Queue implementations
- Queue applications
- Summary
- Queries?

Queue

- First In First Out
- Elements are added at one end called rear and removed only from one end called front
- Gives access only to two elements- one at the front and one at the rear end

What is this good for ?

- A queue at restaurant, office, bus stand, clinic
- Maintain waiting processes in OS
- Multiplayer strict alternate move game

A Queue

- Definition:
  - An ordered collection of homogenous data items
  - Where elements are added at rear and removed from the front end
- Operations:
  - Create an empty queue
  - check if it is empty and/or full
  - Enqueue: add an element at the rear
  - Dequeue: remove the element in front
  - Destroy : remove all the elements one by one and destroy the data structure

The Queue ADT: Value definition

Abstract typedef QueueType(ElementType ele)

Condition: none

Queue ADT: Operator definition

1. Abstract QueueType CreateQueue()

Precondition: none

Postcondition: Empty Queue is created

2. Abstract QueueType Enqueue(QueueType Queue, ElementType Element)

Precondition: Queue not full or NotFull(Queue)= True

Postcondition: Queue = Queue' + Element at the rear

Or Queue = original queue with new Element at the rear

Queue ADT: Operator definition

3. Abstract ElementType dequeue(QueueType Queue)

Precondition: Queue not empty <u>or</u> NotEmpty(Queue)= True

Postcondition: Dequeue= element at the front

Queue= Queue - Element at the front

<u>Or</u> Queue = original queue with front element deleted

4. Abstract DestroyQueue(QueueType Queue)

Precondition: Queue not empty <u>or</u> NotEmpty(Queue)= True

Postcondition: Element from the Queue are removed one by one starting from front to rear.

NotEmpty(Queue)= False

Queue ADT: Operator definition

5. Abstract Boolean NotFull(QueueType Queue)

Precondition: none

Postcondition: NotFull(Queue)= true if Queue is not full

NotFull(Queue)= False if Queue is full.

6. Abstract Boolean NotEmpty(QueueType Queue)

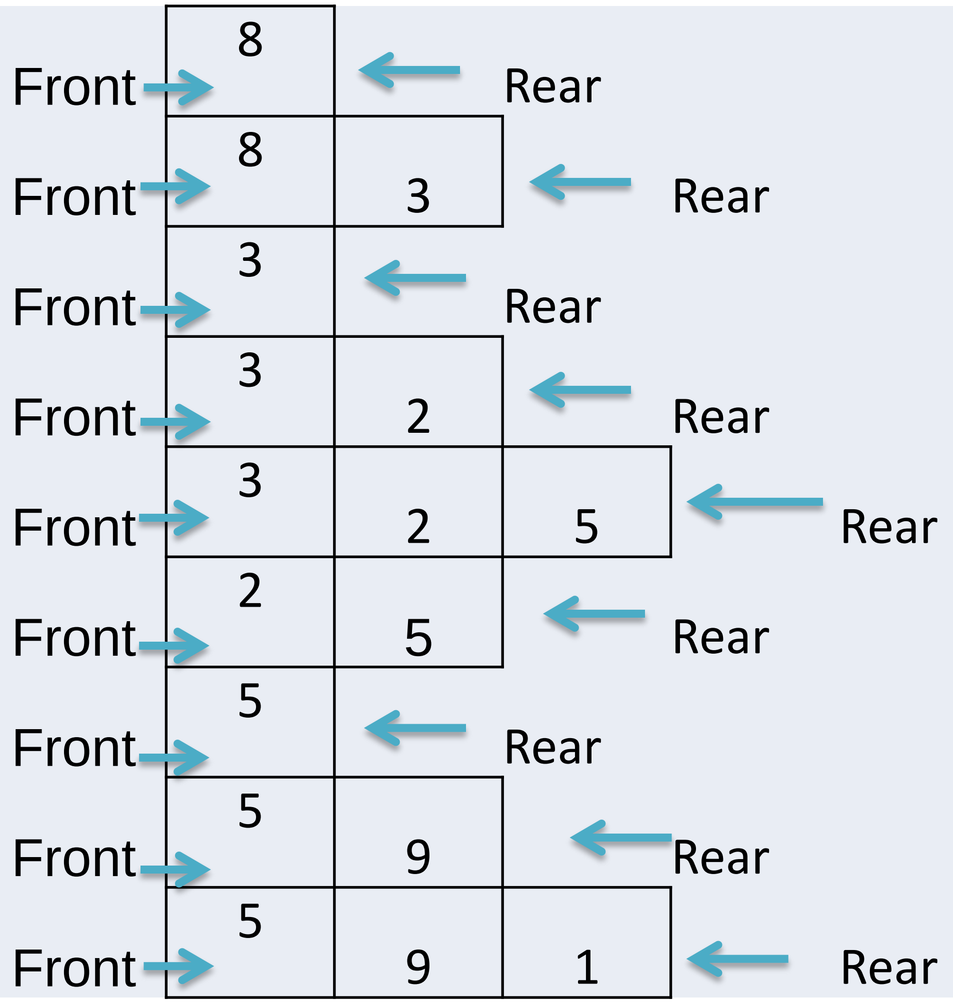
Precondition: none

Postcondition: NotEmpty(Queue)= true if queue is not empty

NotEmpty(Queue)= False if Queue is empty.

Exercise: Queue

- –Enqueue(8)
- –Enqueue(3)
- –Dequeue()
- –Enqueue (2)
- –Enqueue(5)
- –Dequeue()
- –Dequeue()
- –Enqueue(9)
- –Enqueue(1)



SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Issues?

Front, rear									
8									

Enqueue(8), Enqueue(3), Dequeue(), Enqueue (2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Somaiya
TRUST

Types of queues

- Simple queue- additions at rear and deletions from front
- Circular queue- last node is connected to first node, deletions at front end while insertions are done at rear end
- Doubly ended queue- deletions and insertions can be done at both the ends, has two pairs of fronts and rears, both
- Priority queue- every element has predefined priority
  - Max priority : element with max priority is removed first
  - min priority: element with min priority is removed first

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Simple Queue

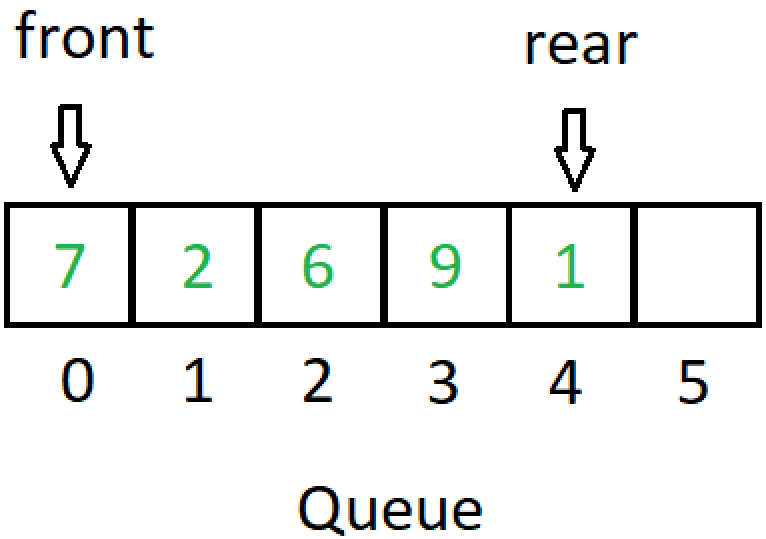


Image courtesy: GeeksforGeeks.org

Somaiya TRUST

Circular Queue

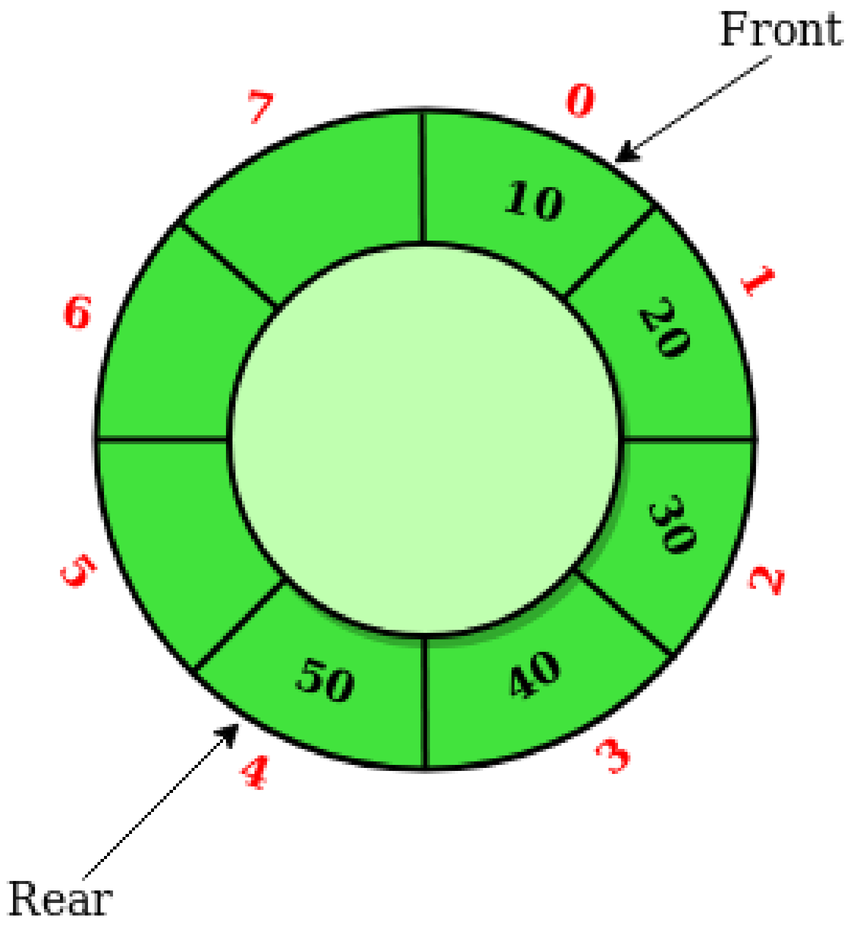


Image courtesy: GeeksforGeeks.org

Doubly ended Queue- Dequeue/deck

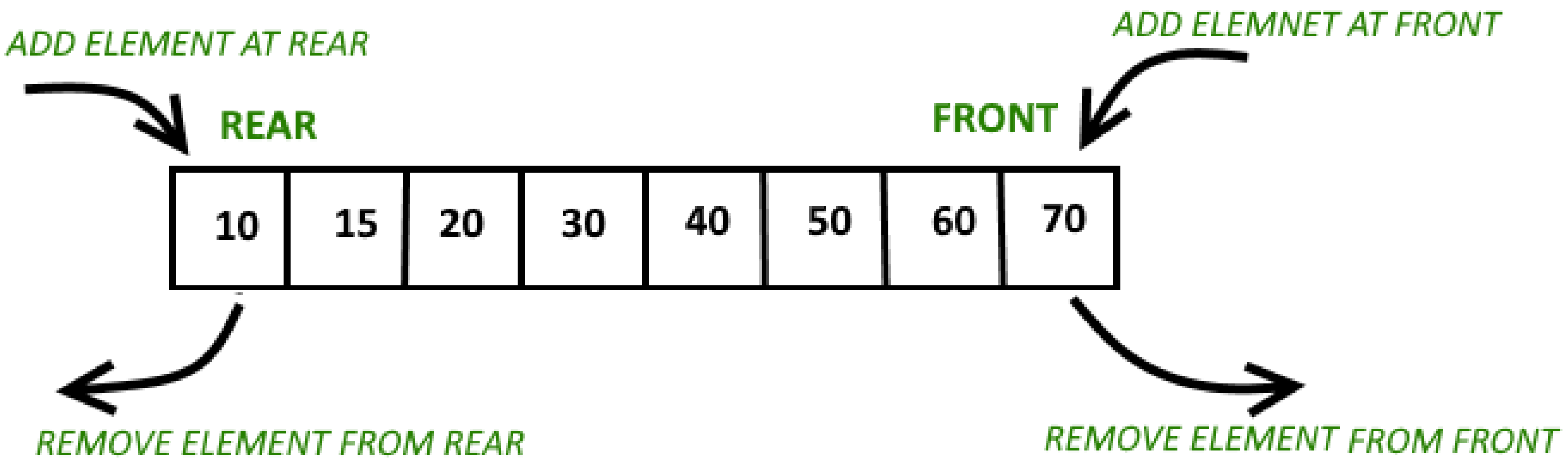


Image courtesy: GeeksforGeeks.org

Priority Queue

Index	Front					Rear		
Data	10	5	3	98	12	36		
Priority	4	4	3	2	1	1		

Max Priority queue

Implementing Queues: Simple queue with Array

Queue indices:										
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:										

- Initally, front=rear=-1 (Empty queue)
- Enqueue(8), Enqueue(3), Dequeue(), Enqueue (2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Queues: Simple queue with Array

Queue indices:	Front Rear									
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:	8									

- Initally, front=rear=-1 (Empty queue)
- Enqueue(8), Enqueue(3), Dequeue(), Enqueue (2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Somaiya TRUST

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Queues: Simple queue with Array

Queue indices:	Front	Rear							
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:	8	3								

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue (2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Somaiya TRUST

Implementing Queues: Simple queue with Array

Queue indices:		Front Rear								
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:		3								

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue (2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Implementing Queues: Simple queue with Array

Queue indices:		Front	Rear							
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:		3	2							

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Implementing Queues: Simple queue with Array

Queue indices:		Front		Rear						
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:		3	2	5						

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Implementing Queues: Simple queue with Array

Queue indices:			Front	Rear						
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:			2	5						

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Pop(), Enqueue(9), Enqueue(1)

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Queues: Simple queue with Array

Queue indices:				Front Rear						
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:				5						

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Somaiya TRUST

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Queues: Simple queue with Array

Queue indices:				Front	Rear					
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:				5	9					

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1)

Somaiya TRUST

Implementing Queues: Simple queue with Array

Queue indices:				Front		Rear				
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:				5	9	1				

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1),

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Queues: Simple queue with Array

Queue indices:						Front Rear				
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:						1				

- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1), Dequeue(), Dequeue(), Dequeue()

Somaiya
TRUST

Implementing Queues: Simple queue with Array

Queue indices:										
Array Index:	0	1	2	3	4	5	6	7	8	9
Data:										

- **Front=-1, Rear=-1**
- Enqueue(8), Enqueue(3), Dequeue(), Enqueue(2), Enqueue(5), Dequeue(), Dequeue(), Enqueue(9), Enqueue(1), Dequeue(), Dequeue(), Dequeue()

Queue ADT: Array Implementation

1. Enqueue
   - Insertion in full queue
   - Insertion in initially empty queue
   - General case
2. Dequeue

   -deletion from empty queue

   -deleting the last remained value in the queue

   - General case

Queue ADT: Array Implementation

1. Algorithm QueueType CreateQueue()

//This Algorithm returns an empty Queue

```
{ front =-1;
Rear=-1
Return queue;
}
```

Queue ADT: Array Implementation

2. Algorithm QueueType Enqueue(QueueType Queue, ElementType Element)

// This algorithm accepts a QueueType Queue and ElementType Element as input and adds ‘Element’ at the rear of ‘Queue’. Front and rear are the integer indices those point to the front and rear elements in the queue

```
{
        if NotFull(Queue)= True
        { Queue[++rear]= Element // add the element at rear
        if (front==-1) then front =0; // insertion of first element
        }
        Else “Error Message”
}
```

Queue ADT: Array Implementation

3. Algorithm ElementType Dequeue(QueueType Queue)

// This algorithm accepts a queue as input and returns 'Element' at the front of 'queue'. Temp is a temporary variable used to hold the value being deleted.

```
{  if NotEmpty(Queue)= True
     {temp= Queue[front];
           if (front==rear) then front=rear=-1; //deletion of last element
           else front++; // general case
      return(temp)
     }
Else  print "Error Message"
}
```

Queue ADT: Array Implementation

4. Abstract DestroyQueue(QueueType Queue)

//This algorithm returns all the elements from Queue in FIFO order and destroys the data structure

```
{ if NotEmpty(Queue) = true
    while(NotEmpty(Queue))
        print Dequeue(Queue)
  else print "Error Message"
}
```

Queue ADT: Array Implementation

5. Abstract Boolean NotFull(QueueType Queue)

```
// This algorithm returns true if the Queue is not full, false otherwise.
{ if  (rear != MaxSize)
        return True
  else
        return False
}
```

6. Abstract Boolean NotEmpty(QueueType Queue)

```
// This algorithm returns true if the Queue is not empty, false otherwise.
{ if (front != -1)
        return True
  else
        return False
}
```

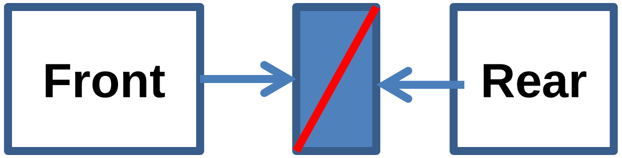
Implementing Queue: Linked List

```
Struct NodeType{
            ElementType Element;
            NodeType *Next;
     }
```

1. Algorithm QueueType CreateQueue()

//This Algorithm creates and returns an empty Queue, pointed by two pointers-front and rear

```
{ createNode(front);
createNode(rear);
Front=rear=NULL;
}
```



Implementing Queue: Linked List

2. QueueType Enqueue(QueueType Queue, NodeType NewNode)

// This Algorithm adds a NewNode at the rear of ‘queue’. rear is a pointer that points to the last node in the queue

```
{ if (front==rear==NULL) // first element in Queue
        NewNode->Next = NULL;
        front=NewNode;
        rear=NewNode;
 Else rear->Next=NewNode;// General case
        rear=NewNode;
}
```

Implementing Queue: Linked List

3. Algorithm ElementType DeQueue(QueueType Queue)

//This algorithm returns value of ElementType stored at the front of queue. Temp is a temproary node used in the dequeuer process.

```
{ if (front==rear==NULL)
          Print "Underflow"
          exit;
 Else
          {
          createNode(Temp);
          Temp=front;
          front= front->next;
          if (front=NULL)
                    rear=NULL;
          Return(temp->Data);
          }
```

Implementing Stacks: Linked List

4. Abstract DestroyQueue(QueueType Queue)

//This algorithm returns values stored in data structure and free the memory used in data structure implementation.

```
{{ if front==NULL
        Print "Underflow"
        exit;
 Else   // createNode(Temp);
        while(NotEmpty(Queue))
        {
        return(Dequeue(Queue));
        }
}
```

Implementing Queue: Linked List

6. Abstract DisplayQueue(QueueType Queue)

//This algorithm Prints all the Elements stored in stack. Temp purpose?

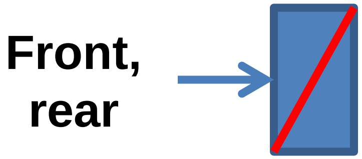
```
{ if front==NULL
        Print "Error Message"
 Else {createNode(Temp)
        Temp=front;
        While(Temp!=Null)
                Print(Temp->Data);
                Temp= Temp->next;
        }
}
```

Implementing Queue: Linked List

- Enqueue(8)
- Enqueue(3)
- Dequeue()
- Enqueue(2)
- Enqueue(5)
- Dequeue()
- Dequeue()
- Enqueue(9)
- Enqueue(1)

Implementing Queue: Linked List

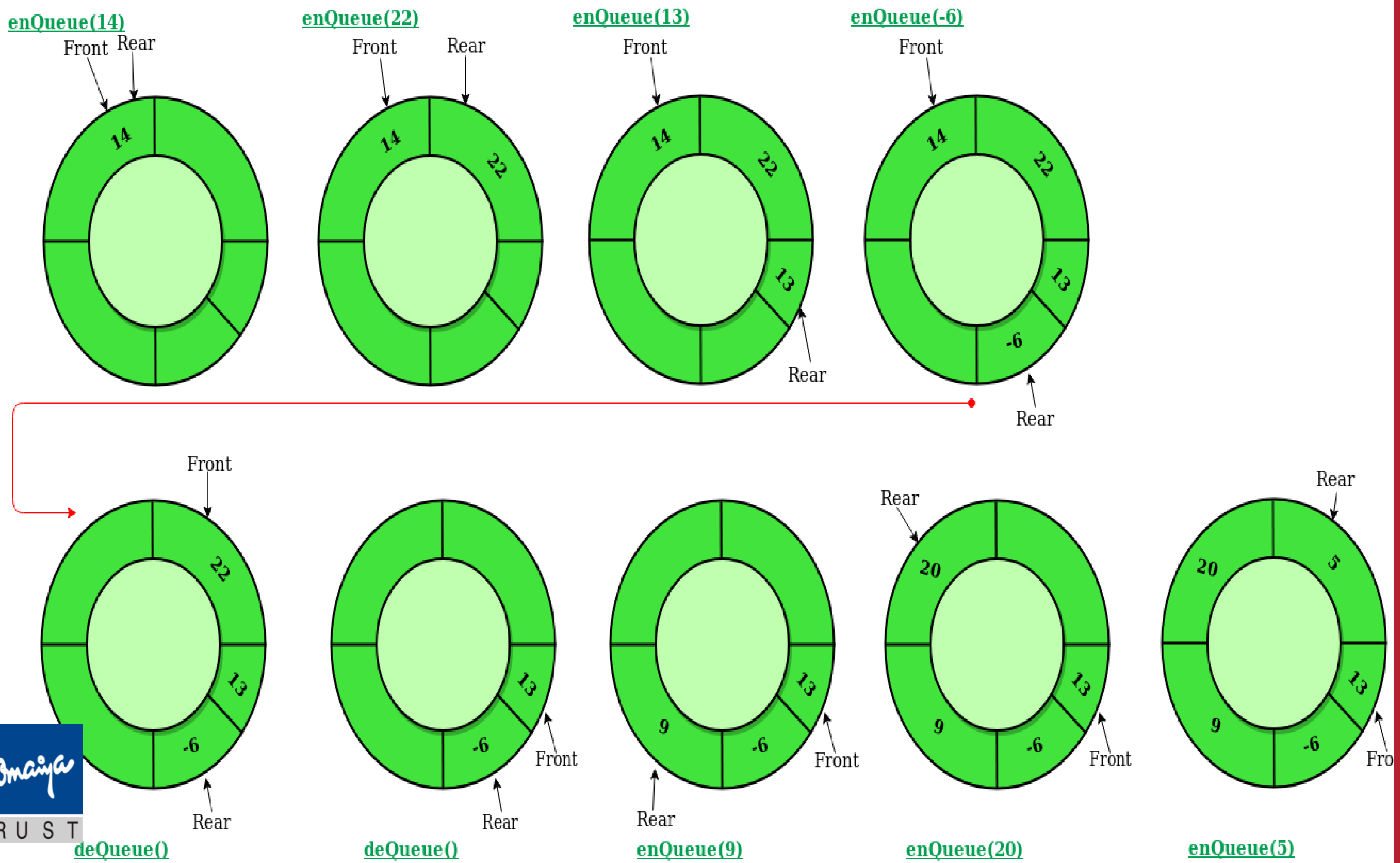
- Create empty queue



Enqueue(8)

Implementing Circular Queue

Implementing Queues: Simple queue with Array



Circular Queue: Array Implementation

1. Enqueue
   - Insertion in full queue
   - Insertion in initially empty queue
   - General case
2. Dequeue
   -deletion from empty queue
   -deleting the last remained value in the queue
   - General case

Circular Queue: Array Implementation

1. Algorithm QueueType CreateCQueue()

//This Algorithm returns an empty Queue

{ front =-1;

Rear=-1

Return queue;

}

Circular Queue: Array Implementation

2. Algorithm QueueType CEnqueue(QueueType CQueue, ElementType Element)

// This algorithm accepts a QueueType Queue and ElementType Element as input and adds ‘Element’ at the rear of ‘Queue’. Front and rear are the integer indices those point to the front and rear elements in the queue. Array CQueue[0:Size-1] is an array that stores queue elements.

```
{
        if NotFull(CQueue)= True
        {if (rear == SIZE – 1 && front != 0)
                rear=0;
        else rear= rear+1;
        CQueue[rear]= Element // add the element at rear
        if (front==-1) then front =0; // insertion of first element
        }
        Else “Error Message”
}
```

Circular Queue: Array Implementation

3. Algorithm ElementType Dequeue(QueueType CQueue)

// This algorithm accepts a queue as input and returns 'Element' at the front of 'queue'. Temp is a temporary variable used to hold the value being deleted. Array CQueue[0:Size-1] is an array that stores queue elements.

```
{  if NotEmpty(CQueue)= True
    {temp= CQueue[front];
        if (front==rear) then front=rear=-1; //deletion of last element
        else if (front==size-1) then front=0;//front was pointing last
                                                        location
        Else front++; // general case
        return(temp)
    }
Else  print "Error Message"
}
```

Circular Queue: Array Implementation

4. Abstract DestroyQueue(QueueType CQueue)

//This algorithm returns all the elements from Queue in FIFO order and destroys the data structure

```
{ if NotEmpty(CQueue) = true
     while(NotEmpty(CQueue))
          print Dequeue(CQueue)
   else print "Error Message"
}
```

Circular Queue: Array Implementation

5. Abstract Boolean NotFull(QueueType CQueue)

// This algorithm returns true if the Queue is not full, false otherwise. Array CQueue[0:Size-1] is an array that stores queue elements. Rear and front are the indices those point to first and last element in circular queue, respectively.

```
{ if  ((rear == SIZE-1 && front == 0) || (rear == front-1))
            return False
  else
            return True
}
```

6. Abstract Boolean NotEmpty(QueueType CQueue)

// This algorithm returns true if the Queue is not empty, false otherwise.

```
{ if (front != -1)
            return True
  else
            return False
```

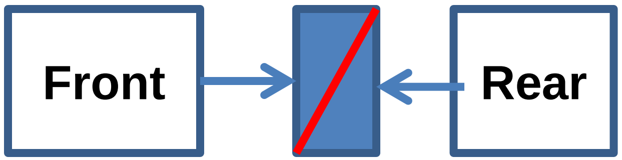
Implementing Circular Queue: Linked List

```
Struct NodeType{
        ElementType Element;
        NodeType Next;
    }
```

1. Algorithm QueueType CreateQueue()

//This Algorithm creates and returns an empty Queue, pointed by two pointers-front and rear

```
{ createNode(front);
createNode(rear);
Front=rear=NULL;
}
```



Implementing Queue: Linked List

2. QueueType Enqueue(QueueType CQueue, NodeType NewNode)

// This Algorithm adds a NewNode at the rear of ‘queue’. rear is a pointer that points to the last node in the queue

```
{
        If(front==rear==NULL)
                Front=rear=newnode // insertion of first element
                rear->next=newnode //circular queue definition
        else //general case
                temp=front;
                while(temp!=rear) {
                        temp=temp->next;
                        temp->next = newnode;
                        newnode->next = rear->next;
                        rear=newnode;
                }//while
}//enqueue
```

Enqueue another algorithm

2. QueueType Enqueue(QueueType CQueue, NodeType NewNode)

// This Algorithm adds a NewNode at the rear of ‘queue’. rear is a pointer that points to the last node in the queue

```
{
        If(front==rear==NULL)
                Front=rear=newnode // insertion of first element
                rear->next=newnode //circular queue definition
        else //general case
                rear->next= newnode;
                rear=newnode;
                newnode->next=front;

}//enqueue
```

Implementing Queue: Linked List

3. Algorithm ElementType DeQueue(QueueType CQueue)

//This algorithm returns value of ElementType stored at the front of queue. Temp is a temporary node used in the dequeuer process.

```
{ if (front==rear==NULL)
            Print "Underflow"
            exit;
 Else if (front==rear)
            { temp= front;
              front=rear=NULL;
              return(temp->data);
            }
Else {
            temp=front;
            front=front->next;
            rear->next= front;
            return(temp->data);
            }
}//Dequeue
```

Implementing Stacks: Linked List

4. Abstract DestroyQueue(QueueType CQueue)

//This algorithm returns values stored in data structure and free the memory used in data structure implementation.

```
{ if front==NULL
        Print "Underflow"
        exit;
 Else {  createNode(Temp);
        while(NotEmpty(CQueue))
        {
        return(Dequeue(CQueue));
        }
        }//else
}
```

Implementing Queue: Linked List

6. Abstract DisplayQueue(QueueType Queue)

//This algorithm Prints all the Elements stored in stack. Temp purpose?

{ if front==NULL

Print "Error Message"

Else {

Student Assignment

}

Joseph’s Problem

- There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of person = n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle.

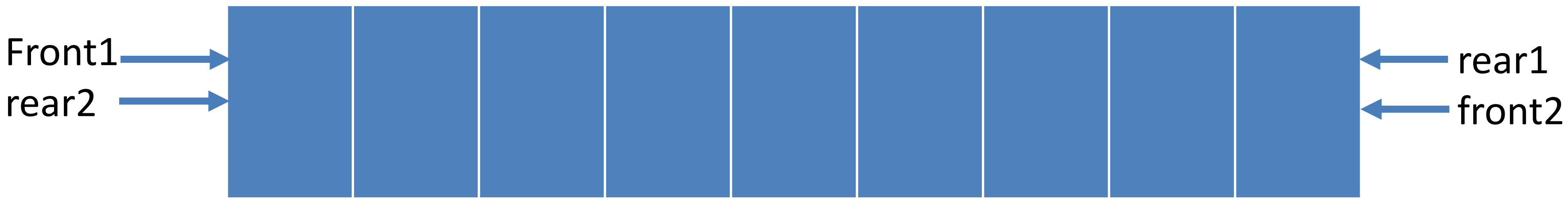
- Given the people = {Arya, Jon, Robb, Catelyn, Rose, Bran, Tyrion, Cersei, Sansa, Brienne}

k=4, Figure out name of the surviving person assuming that they are standing in the same sequence as given in the set. Show the solution step by step.

Doubly ended queue(Deque/deck)

Doubly ended queue(Deque)

Definition: queue has two pairs of fronts and rears on either end.



Doubly ended Queue: Array Implementation

1. Enqueue
    - Insertion in full queue
    - Insertion in initially empty queue
    - General case
2. Dequeue

    -deletion from empty queue

    -deleting the last remained value in the queue

    - General case

DQue: Array Implementation

1. Algorithm QueueType CreateDQueue()

```
//This Algorithm returns an empty Queue
{ front1 =-1;
Rear1=-1;
Front2=-1;
Rear2=-1;
Return dqueue;
}
```

DQue: Array Implementation

2. Algorithm QueueType DEnqueue(QueueType DQueue, ElementType Element, int end)

// This algorithm accepts a QueueType DQueue and ElementType Element as input and adds 'Element' at the rear of 'Queue'. Front and rear are the integer indices those point to the front and rear elements in the queue. Array DQueue[0:Size-1] is an array that stores queue elements. The integer variable end defines where the element is to be added; 1=right end and 2=left end.

```
{
        if(end==2 && rear2==0)  then LeftEnd=Full; exit;
        if(end==1 && rear1==maxsize-1) then RightEnd=Full; exit;
        if(rear1=-1) //insertion of first element
        { front1=front2=rear1=rear2=MaxSize/2; //set indices in such a way that queue has
scope to grow in both directions
        deque[rear1]=element;
        }
        else if(end==1) //insertion in right end using rear1, general case
                deque[++rear1]=element
                front2=rear2
        else if(end==2) ) //insertion in left end using rear2, general case
                deque[--rear2]=element;
                front1=rear2
```

Dque Queue: Array Implementation

3. Algorithm ElementType Dequeue(QueueType Dqueue, int end)

// This algorithm accepts a queue as input and returns ‘Element’ at the front of ‘queue’. Temp is a temporary variable used to hold the value being deleted. Array CQueue[0:Size] is an array that stores queue elements. The integer variable end defines from where the element is to be deleted; 1=left end and 2=right end

```
{ if (front1==-1) then underflow; exit; // deleting from empty data structure?
   if(front1==front2==rear1==rear2) { // only element in deque
              temp=Deque[front1]
              front1=front2=rear1=rear2=-1
              }//if
     else if(end==1) { // deletion in left end with front1?
              temp=Deque[front1]
              front1++; rear2++;
              }//else if
     else if(end==2) { // deletion in right end with front2?
              temp=temp=Deque[front2]
              front2--; rea1--;
              } //else if
  return(temp)
}
```

Deque Queue: Array Implementation

4. Abstract DestroyQueue(QueueType DQueue)

//This algorithm returns all the elements from Queue in FIFO order and destroys the data structure

```
{ if NotEmpty(DQueue) = true
     while(NotEmpty(DQueue))
          print Dequeue(DQueue)
   else print "Error Message"
}
```

Deque Queue: Array Implementation

5. Abstract Boolean NotFull(QueueType CQueue)

Student assignment

6. Abstract Boolean NotEmpty(QueueType CQueue)

Student assignment

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

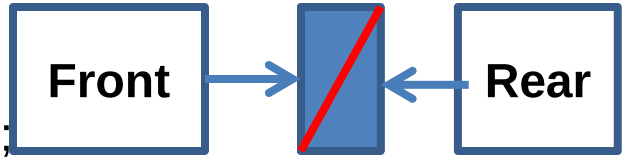
Implementing Deque: Linked List

```
Struct NodeType{
                    ElementType Element;
                    NodeType Next;
          }
```

1. Algorithm QueueType CreateQueue()

//This Algorithm creates and returns an empty Queue, pointed by two pointers-front and rear

```
{ createNode(front1);
createNode(rear1);
 createNode(front2);
createNode(rear2);
Front1=rear1=front2=rear2=NULL;
}
```



Somaiya TRUST

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing DQue: Linked List

2. QueueType Enqueue(QueueType CQueue, NodeType NewNode, int end)

// This Algorithm adds a NewNode at the rear of ‘queue’. rear is a pointer that points to the last node in the queue

```
{  if(rear1==Null) //if inserting first element?
            front1=rear1=front2=rear2=NewNode;
   else if(end==1)
           { rear1->next= NewNode;
           front2= NewNode;
           rear1= NewNode;
           }
   else if(end==2)
           { NewNode->next= rear2;
           rear2=NewNode;
           front1=NewNode;
           }
}//enqueue
```

Implementing DQue: Linked List

3. Algorithm ElementType DeQueue(QueueType Dqueue, int end)

//This algorithm returns value of ElementType stored at the front of queue. Temp is a temporary node used in the dequeuer process.

```
{ if (front1==NULL)          Print "Underflow"            exit;  //underflow
 Else if (front1==rear1) //last node in the data structure
                { temp= front1;
                  front1=rear1=front2=rear2=NULL;
                  return(temp->data);
                }
Else if (end==1)  //deleting the left end element at front1
                {
                temp=front1;
                front1=front1->next;
                rear2= front1;  or rear2= rear2->next;
                return(temp->data);
                }

Else if (end==2) //deleting the right end element at front2
                { temp=front2;
                temp2=front1;
                while(temp2->next!=front2)
                                temp2=temp2->next; //While loop
                rear1= temp2;
                front2= temp2;
                rear1->next = NULL
                return(temp->data);
                }
}//Dequeue
```

Implementing Dqueue: Linked List

4. Abstract DestroyQueue(QueueType DQueue)

//This algorithm returns values stored in data structure and free the memory used in data structure implementation.

```
{ if (front1==NULL)
        Print "Underflow"
        exit;
 Else {  createNode(Temp);
        while(NotEmpty(Dqueue))
        {
        return(Dequeue(Dqueue,1));
        }
        }//else
}
```

Implementing Queue: Linked List

6. Abstract DisplayQueue(QueueType DQueue)

//This algorithm Prints all the Elements stored in stack. Temp purpose?

```
{ if front==NULL
        Print "Error Message"
 Else {
Student Assignment


}
```

Priority queue

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Priority queue

Definition: A collection of heterogeneous elements, accessed in FIFO manner wherein each element has an additional priority associated with it.

Types-

- MinPriority Queue- smaller the number, higher the priority.
- MaxPriority Queue- Larger the number, higher the priority.

Front → | | rear ←

Index	0	1	2	3	4	5	6
Element	20	12	2	34	76	11	98
Priority	1	1	2	3	4	4	5

MinPriority Queue

Somaiya TRUST

Priority Queue: Array Implementation

1. Enqueue
   - Insertion in full queue
   - Insertion in initially empty queue
   - General case
2. Dequeue
   -deletion from empty queue
   -deleting the last remained value in the queue
   - General case

Priority Queue: Array Implementation

```
Struct PriQueue{      int data;
                      int priority
               };
Struct PriQueue PQ[MaxSize];
```

1. Algorithm QueueType CreatePQueue()

//This Algorithm returns an empty Queue

```
{ front =-1;
Rear=-1
}
```

Priority Queue: Array Implementation

2. Algorithm QueueType PEnqueue(QueueType PQueue, ElementType Element, int p)

// This algorithm accepts a QueueType Pqueue, ElementType Element and its associated priority ‘p’ as input and adds ‘Element’ at the rear of ‘Queue’. Front and rear are the integer indices those point to the front and rear elements in the queue. Array PQueue[0:MaxSize-1] is an array that stores queue elements.

```
{ Struct PriQueue key;
            if(rear==MaxSize-1) then  overflow; exit; //PQueue is full
else        if (front==rear==-1) // inserting first element
            { front=rear=0;
             PQ[rear].data= element;
            PQ[rear].priority = p;
            }
}else if { rear++// increment rear to accommodate new element
            PQ[rear].data=element;
             PQ[rear].priority=p;
      //find a proper place for new element as per its priority using insertion sort logic
           key=PQ[rear]
            j=rear-1;
            while(j>=0 && PQ[j].priority < key.priority)
            { PQ[j+1]=PQ[j];
              j--;
            }
            PQ[j+1]= key; //assign both data value and priority
}
```

Priority Queue: Array Implementation

3. Algorithm ElementType Dequeue(QueueType PQueue)

// This algorithm accepts a queue as input and returns ‘Element’ at the front of ‘queue’. Temp is a temporary variable used to hold the value being deleted. Array CQueue[0:Size] is an array that stores queue elements.

```
{  struct PriQueue temp;
 if (front=-1) then underflow; exit; // deleting from empty data structure?
   if(front==rear) { // only element in PQueue
            temp=PQ[front] ;
            front=rear=-1;
            }//if
      else  { // General case
            temp=PQque[front]
            front++;
            }//else
return(temp)
}
```

Priority Queue: Array Implementation

4. Abstract DestroyQueue(QueueType PQueue)

//This algorithm returns all the elements from Queue in FIFO order and destroys the data structure

```
{ if NotEmpty(PQueue) = true
     while(NotEmpty(PQueue))
          print Dequeue(PQueue)
   else print "Error Message"
}
```

Priority Queue: Array Implementation

5. Abstract Boolean NotFull(QueueType PQueue)

Student assignment

6. Abstract Boolean NotEmpty(QueueType PQueue)

Student assignment

Implementing Priority: Linked List

Struct NodeType{

ElementType Element;

integer priority;

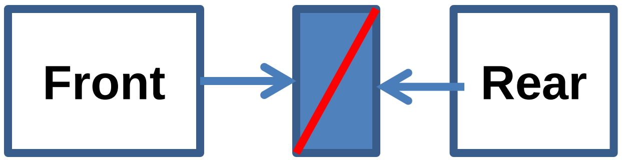
NodeType Next;

}

1. Algorithm QueueType CreateQueue()

//This Algorithm creates and returns an empty Queue, pointed by two pointers- front and rear

```
{ createNode(front);
createNode(rear);
Front=rear=NULL;
}
```



SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Priority Queue: Linked List

2. QueueType Enqueue(QueueType PQueue, NodeType NewNode, int p)

// This Algorithm adds a NewNode at the rear of ‘queue’. rear is a pointer that points to the last node in the queue

```
{  if(rear==Null) //if inserting first element?
            front=rear=NewNode;
   else if(front.priority >  NewNode->priority) //insertion before the first node
            { NewNode->next= front;
            front= NewNode;
            }
   else   { temp = front; current=NULL;
            while(temp->priority < =NewNode->priority && temp->next!=Null)
                    current=temp; temp=temp->next;
            if(temp->priority > NewNode->Priority) //insertion in between
                    Newnode->next= temp;
                    current-> next= NewNode;
            if(temp->next==NULL) // insertion after rear
                    temp->next=NewNode;
                    rear=NewNode;
            }
}//enqueue
```

Somaiya TRUST

Implementing Priority Queue: Linked List

3. Algorithm ElementType DeQueue(QueueType PQueue)

//This algorithm returns value of ElementType stored at the front of queue. Temp is a temporary node used in the dequeuer process.

```
{ if (front==NULL)
          Print "Underflow"
          exit;
 Else if (front==rear) // deleting the last remaining node in the PQueue
          { temp= front;
            front=rear=NULL;
            return(temp->data);
          }
Else                     // general case
          {
          temp=front;
          front=front->next;
          return(temp->data);
          }

}//Dequeue
```

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Implementing Dqueue: Linked List

4. Abstract DestroyQueue(QueueType PQueue)

//This algorithm returns values stored in data structure and free the memory used in data structure implementation.

```
{ if (front==NULL)
        Print "Underflow"
        exit;
 Else {  createNode(Temp);
        while(NotEmpty(PQueue))
        {
        return(Dequeue(PQueue));
        }
        }//else
}
```

Somaiya TRUST

Implementing Queue: Linked List

6. Abstract DisplayQueue(QueueType DQueue)

//This algorithm Prints all the Elements stored in stack. Temp purpose?

```
{ if front==NULL
        Print "Error Message"
 Else {
Student Assignment


}
```

Queries?

Thank you!