TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. Linear Data Structure
    a. Stack (continued)
    b. Queue

Stack: Abstract Data Type

- Stack
  - Operations
    - PUSH (data insertion into stack)
    - POP (data deletion from stack)
    - Display (showing element of stack)

Primary operation

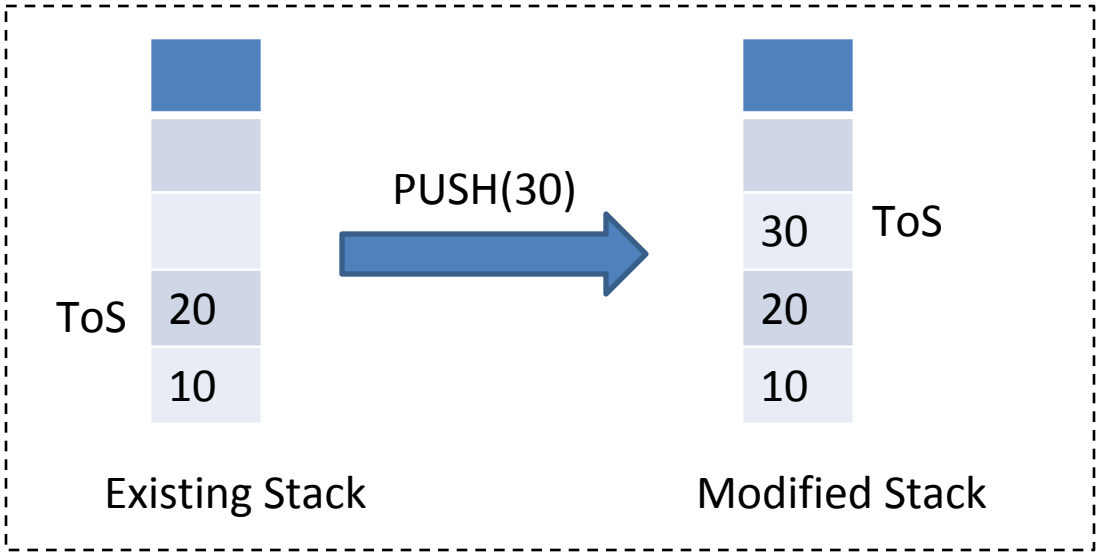
    - IsFullStack (checks if stack is overflow)
    - IsEmptyStack (checks if stack is underflow)

Auxiliary operation

Stack: Abstract Data Type

- Stack Operation
  - PUSH
    - This function inserts one element at the top most position of the stack if the stack is not full
    - The newly inserted data is pointed by ToS

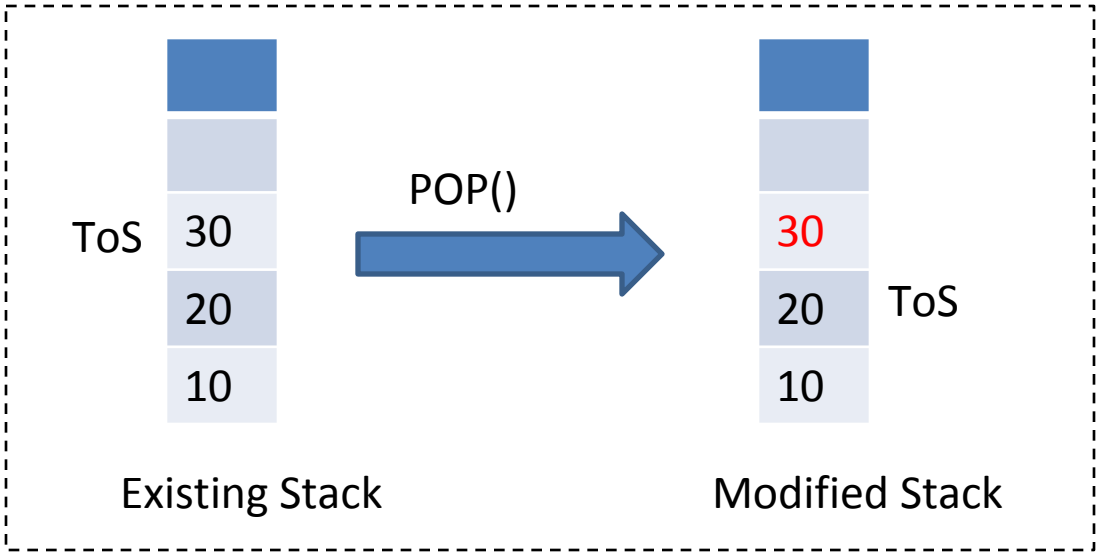
```
void PUSH(element)
{
  if  IsFullStack = TRUE
    return
  else
    tos := tos+1
    Stack(tos) := element
}
```



Stack: Abstract Data Type

- Stack Operation
  - POP
    - This operation deletes the top most element of the stack if it is not empty
    - The current top most element will be pointed by ToS

```
int POP()
{
 if  IsEmptyStack = TRUE
   return NULL
 else
   data := Stack(tos)
   tos := tos-1
   return data
}
```



Stack: Abstract Data Type

- Stack Operation
  - Display (or PEEK)
    - This function displays the top most element of the stack.
    - It does not change the ToS pointer
    - All elements can also be displayed through an auxiliary pointer without shifting ToS

```
int PEEK()
{
  if  IsEmptyStack = TRUE
    return NULL
  else
    return Stack(ToS)
}
```

```
void Display()
{
  if  IsEmptyStack = TRUE
    print "stack empty"
  else
     for i= ToS to 0
         print Stack(i)
}
```

Stack: Abstract Data Type

- Stack Operation
  - IsFullStack
    - This function checks whether the stack is full or not
    - We cannot push data into stack if it is full

```
Boolean IsFullStack()
{
  if  ToS = Max_Size
    return TRUE
  else
    return FALSE
}
```

Stack: Abstract Data Type

- Stack Operation
  - IsEmptyStack
    - This function checks whether the stack is empty or not
    - We cannot pop or display the stack if it is empty

```
Boolean IsEmptyStack()
{
  if  ToS = NULL
    return TRUE
  else
    return FALSE
}
```

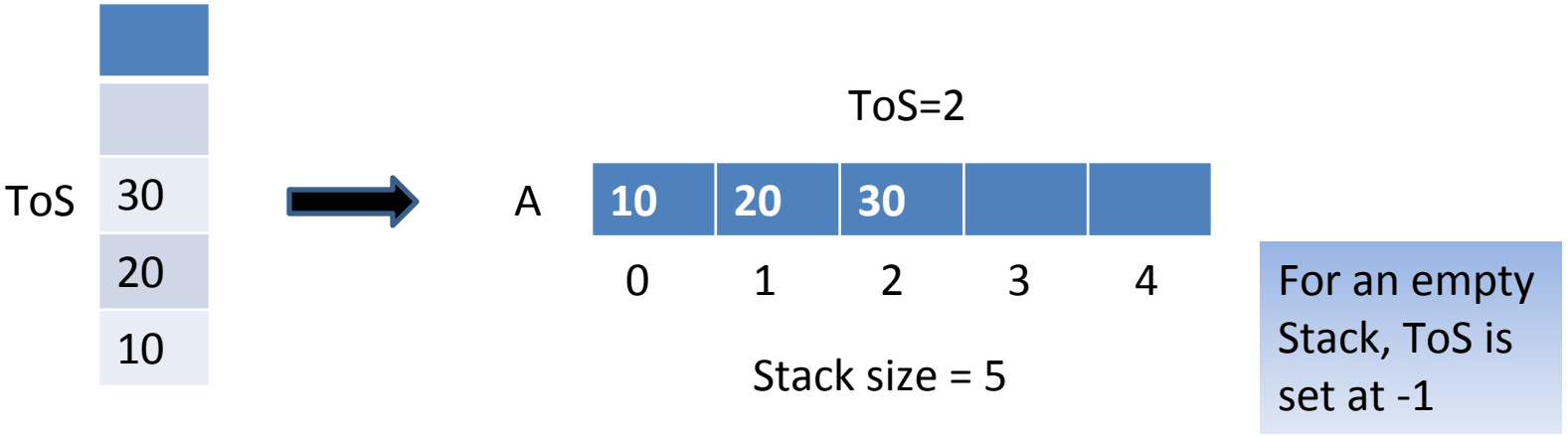
Stack: Abstract Data Type

- Stack Operation: complexity

Operation	Time Complexity
PUSH()	O(1)
POP()	O(1)
Display()	O(1)
IsFullStack()	O(1)
IsEmptyStack()	O(1)
* Display all elements	O(n)

Stack: Abstract Data Type

- Stack
  - Implementation
    - Using array
    - Using Linked-list (will be described later)
  - Array implementation



Stack: Abstract Data Type

- Stack

  - Disadvantage
    - Stack size is fixed
    - Stack size may be increased dynamically but it is very cost effective (for array implementation)
    - Data insertion, deletion and accessing is restricted (can be performed one by one and from one end of the stack)

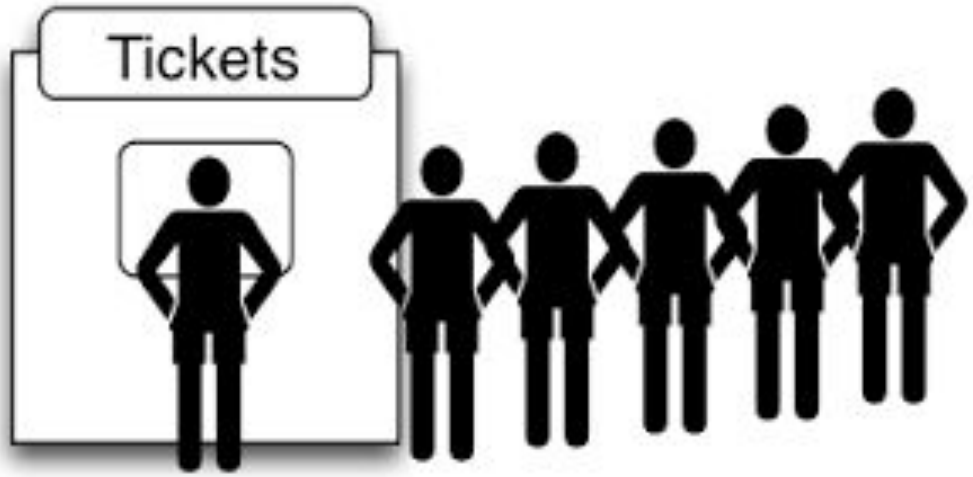
Stack: Abstract Data Type

- Stack
  - Applications
    - In-fix to post-fix expression conversion
    - Evaluation of mathematical expression (post-fix)
    - Implementing function calls including recursion
    - Page visiting history in web browser
    - Undo sequence in a text editor
    - Implementing operations of other data structures
      - Traversal in Tree data structure
      - Traversal in Graph data structure

Queue
(Abstract Data Type)

Queue

- Real Scenario to Data Structure

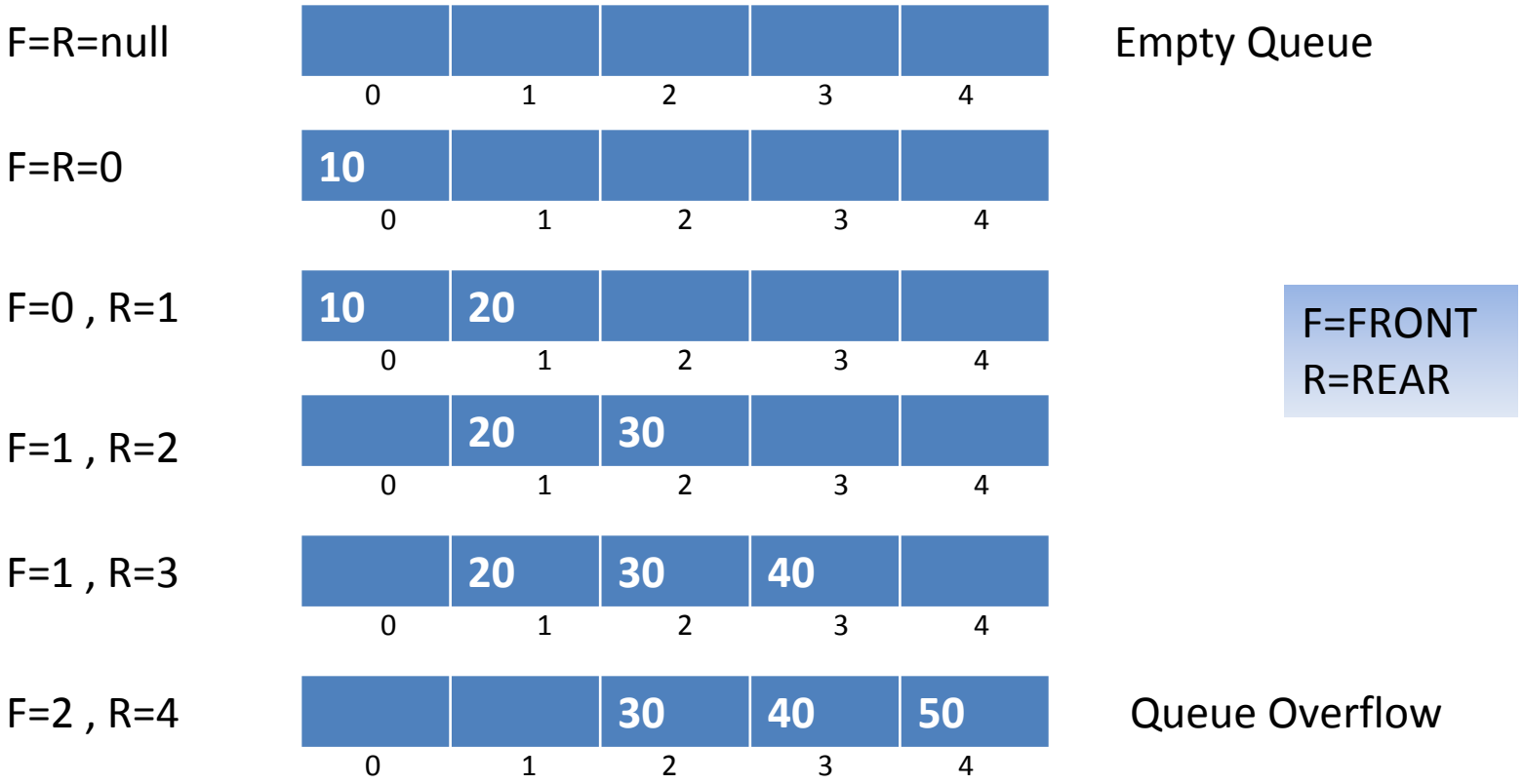


Queue: Abstract Data Type

- Queue
  - Properties
    - Queue is defined as a **First In First Out (FIFO)** data structure
      - The first data inserted into the Queue to be deleted first
    - The associated operations of a Queue are also defined with the data structure that is why it is considered as an **ADT**
    - The **first element** of the Queue is pointed by **FRONT** pointer
    - The **last element** of the Queue is pointed by **REAR** pointer
    - New element is inserted at **REAR** end
    - An element is accessed or deleted from **FRONT** end

Queue: Abstract Data Type

- Queue
  - Representation



Queue: Abstract Data Type

- Queue
  - Operations
    - ENQUEUE (data insertion into queue)
    - DEQUEUE (data deletion from queue)

Primary operation

    - Front / Display (showing element of queue)
    - QueueSize (returns the total element)
    - IsFullQueue (checks if Queue is overflow)
    - IsEmptyQueue (checks if Queue is underflow)

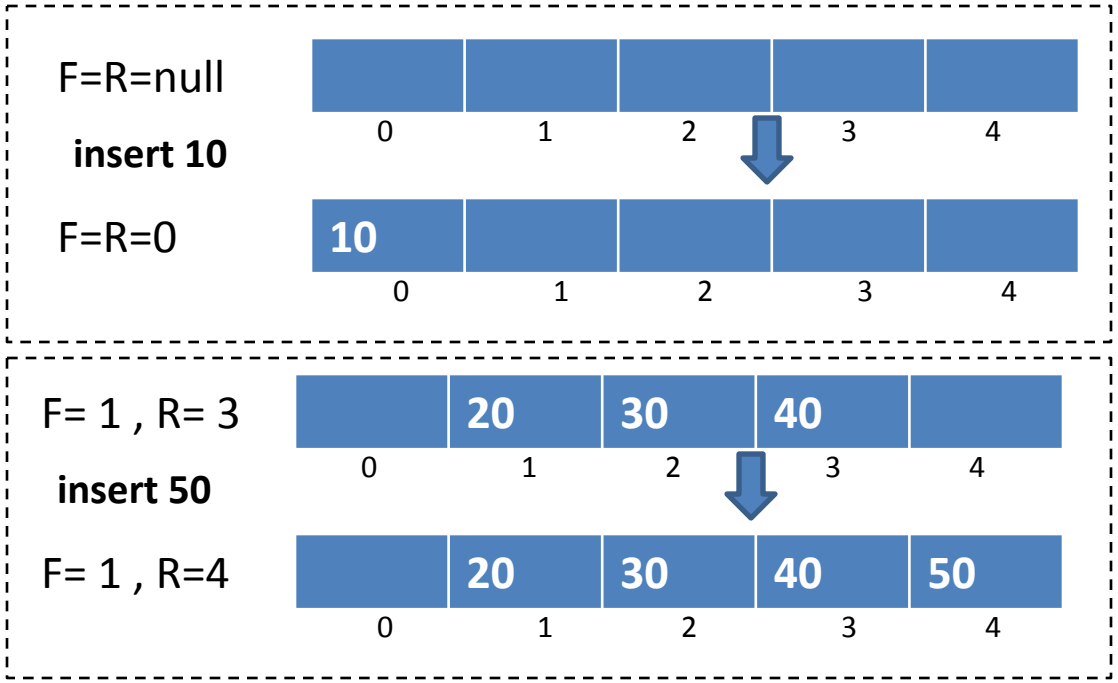
Auxiliary operation

Queue: Abstract Data Type

- Queue variations
  - Simple Queue
    - Data can be inserted at one end, deleted from the other end
    - Rear pointer will always be in right side of front pointer
  - Circular Queue
    - Data can be inserted circularly if front end is vacant
    - Rear pointer can come, circularly, before front pointer
  - Double Ended Queue (Deque)
    - Data can be inserted or removed from either end of the queue
  - Priority Queue
    - Each element is associated with a priority, based on which it is processed (accessed or deleted)

Queue: Abstract Data Type

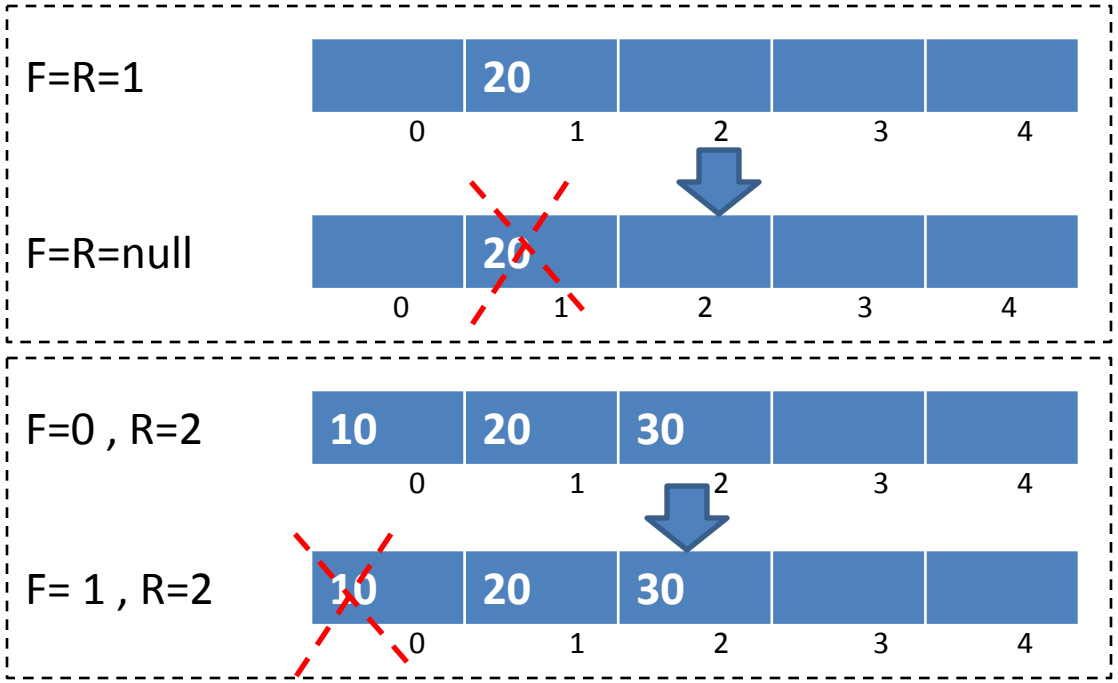
- Queue Operation: simple queue
  - ENQUEUE
    - This function inserts one element at the REAR position of the Queue if it is not full




```
void ENQUEUE(data)
{
 if  IsFullQueue = TRUE
    print  Q is full
 else
    if   IsEmptyQueue = TRUE
        F := 0 and R:= 0
    else
       R:= R+1
    Q(R) := data
}
```

Queue: Abstract Data Type

- Queue Operation: simple queue
  - DEQUEUE
    - This operation deletes the front element of the Queue if it is not empty




```
int DEQUEUE()
{
 if  IsEmptyQueue = TRUE
   return NULL
 else
    if   F = R
       data := Q(F)
       F := null and R:= null
    else
       data := Q(F)
       F:= F+1
   return data
}
```

Queue: Abstract Data Type

- Queue Operation: simple queue
  - Front / Display
    - Front function displays the front element of the Queue
    - All elements can also be displayed through an auxiliary pointer without shifting FRONT or REAR

```
int Front()
{
  if  IsEmptyQueue = TRUE
    return NULL
  else
    return Q(F)
}
```

```
void Display()
{
  if  IsEmptyQueue = TRUE
    print  Q is empty
  else
     for i= F to R
        print  Q(i)
}
```

Queue: Abstract Data Type

- Queue Operation: simple queue
  - QueueSize
    - This function returns the counting of elements present in the current queue

```
int QueueSize()
{
  if  F = null and R = NULL
     return 0
  else
      for i = F to R
         count := count +1
      return count
}
```

Queue: Abstract Data Type

- Queue Operation: simple queue
  - IsFullQueue
    - This function checks whether the Queue is full or not
    - We cannot insert data into Queue if it is full

```
Boolean IsFullQueue()
{
  if  R = Max_Size
    return TRUE
  else
    return FALSE
}
```

Queue: Abstract Data Type

- Queue Operation: simple queue
  - IsEmptyQueue
    - This function checks whether the Queue is empty or not
    - We cannot delete or display the Queue if it is empty

```
Boolean IsEmptyQueue()
{
  if  F = null and R = null
    return TRUE
  else
    return FALSE
}
```

Queue: Abstract Data Type

- Queue Operation: complexity

Operation	Time Complexity
Enqueue()	O(1)
DeQueue()	O(1)
Display()	O(n)
QueueSize()	O(n)
IsFullQueue()	O(1)
IsEmptyQueue()	O(1)

Queue: Abstract Data Type

- Queue

  - Disadvantage
    - Queue size is fixed
    - Queue size may be increased dynamically but it is very cost effective (for array implementation)
    - Data insertion, deletion and accessing is restricted

Queue: Abstract Data Type

- Queue
  - Applications
    - Job scheduling by Operating System for same priority jobs
    - Handling multiprogramming in computing devices
    - Implementing first-come-first-serve based real-life operations
    - Waiting time of a customers at call centre
    - Implementing operations of other data structures
      - Various Graph operations (shortest path finding, finding MST)
      - Traversal in Graph data structure

Queries?