Unit-2
Arrays, Stack and Queue

Khushi Lad

Recalling C++

- An array is a collection of variables of the same type that are referenced by a common name.
- The individual elements of an array are referred by their index or subscript value.
- Arrays are a way to group a number of items into a larger unit.
- Arrays can have items of simple types like int or float or even of user-defined types like structures and objects

Array

- An array is a finite, ordered and collection of homogeneous data elements.
- Collection of homogeneous data elements termed -- means what?
- It means --- all the elements of an array are of the same datatype only.
- Array is finite, because it contains only a limited number of elements; and ordered, as all the elements are stored one by one in contiguous locations of computer memory in a linear ordered fashion.

- An array is known as linear data structure, because all elements of the array are stored in a linear order.

One Dimensional array

- The array is given a name and its elements are referred to by their subscript or indices.
- C++ array‟s index numbering starts with 0.
- The general form of an array declaration is… type array-name [size];
- For example:
- The declaration of an array marks of base type int which can holds 10 elements is…

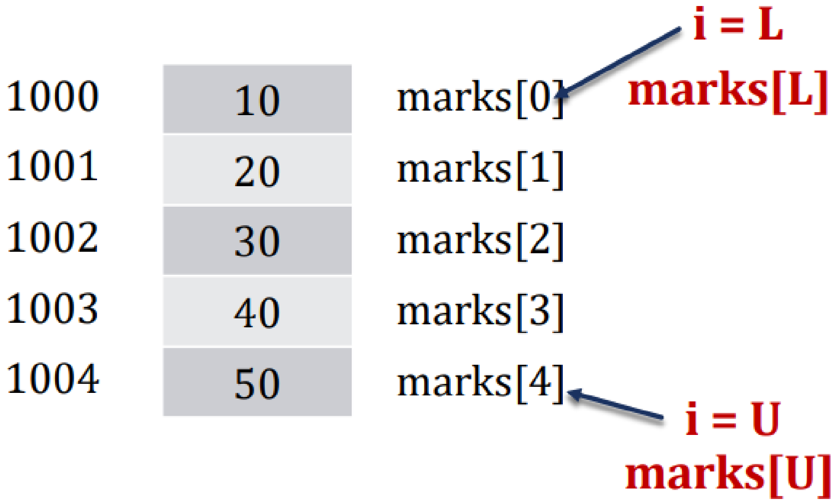
int marks[10];

Operation in array

- There are various operations that can be performed on an array. They are:
- 1. Traversing
- 2. Sorting
- 3. Searching
- 4. Insertion
- 5. Deletion
- 6. Merging

Traversing

- This operation is used to visit all the elements in an array.
- Algorithm:
- Steps:
- 1. i = L
- 2. While i ≤ U do
- 3. process (a[i])
- 4. i = i + 1
- 5. EndWhile
- 6. Stop



1000	10	marks[0]
1001	20	marks[1]
1002	30	marks[2]
1003	40	marks[3]
1004	50	marks[4]

i = L
marks[L]

i = U
marks[U]

Sorting

- This operation is used to sort all the elements in an array in a specified order (ascending/descending)
- Algorithm:

```
i = U
While i ≥ L do
    j = L
    While j < i do
        If Order(a[j], a[j+1]) = FALSE
            Swap(a[j], a[j+1])
        EndIf
        j = j + 1
    EndWhile
    i = i – 1
EndWhile
Stop
```

Searching

- This operation is used to search an index or the location of an element in an array.

```
1. i = L, found = 0, location = 0
2. While (i ≤ U) and (found=0) do
3.     If Compare(a[i], KEY) = TRUE then
4.         found = 1
5.         location = I
6.     Else
7.         i = i +1
8.     EndIF
9. EndWhile
10. If found = 0 then
11.     Print “Search is unsuccessfully: KEY is not in the array”
```

- 12. Else Print “Search is successfully: KEY is in the array at location”, location 1
- 3. EndIf
- 14. Return (Location)
- 15. Stop

Insertion

- This operation is used to insert a new value at a particular location in an array.
- This operation is also used to insert an element into an array, providing that that the array is not full.

Algorithm

```
1. If A[U] ≠ Null then
2.     Print "Array is full! No insertion is possible!"
3.     Exit
4. Else
5.     i = U
6.     While i > Location do
7.         A[i] = A[i-1]
8.         i = i – 1
9.     EndWhile
10.    A[Location] = KEY
11. EndIF
12. Stop
```

Deletion

- This operation is used to delete a particular element from an array.
- If an element is deleted from L location to U-1 location, then push up each element, after the victim element, by one position.
- Therefore, firstly if an element is deleted at the tail of an array, then no push up of any element is required.
- So, here no intermediate location will be made empty, that is, an array should be packed; and empty locations are at the tail of an array.

Algorithm

```
1. i = SearchArray(A, KEY)
2. If (i = 0) then
3.         Print "KEY is not found! No deletion can take place!"
4.         Exit
5.   Else
6.       While i < U do
7.           A[i] = A[i + 1]
8.           i = i + 1
9.       EndWhile
10. EndIf
11.     A[U] = NULL
12.     U = U – 1
13. Stop
```

Application of Array

- Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables.
- Many databases, small and large, consist of one-dimensional arrays whose elements are records.
- Arrays are used to implement other data structures, such as Stacks, Queues, Deques, Linked Lists, Heaps, Hash Tables, etc.
- Arrays can be used to determine partial or complete control flow in programs, as a compact alternative to the multiple “if” statements.
- There are abundant applications of arrays in the computation.
- That is why almost every programming language includes this datatype as a built-in datatype.

Introduction to stack

- A stack is linear data structure.
- It is very much useful in various applications of computer science.
- The implementation of the majority of systems programs is simplified using this data structure.
- stack is something which follows the **last-in first-out** strategy.

Definition

- A stack is an ordered collection of homogeneous data elements where the insertion and deletion operations take place at one end only.
- In the case of a stack…
- The **insertion operation** are specially termed as **PUSH**,
- The **deletion operation** are specially termed as **POP**
- The **position of the stack**, where these operations are performed is known as the **TOP** of the stack.

Operation on stack

- The basic operations required to manipulate a stack are:
- 1. PUSH
- 2. POP
- 3. STATUS (PEEP) - To know the present state of a stack. i.e. the top data value of the stack.

Algorithms for PUSH operation:

- Steps:

```
1. If TOP ≥ SIZE then
2.         Print “Stack is Full!”
3.   Else
4.          TOP = TOP + 1
5.          A[TOP] = ITEM
6. EndIF
7. Exit
```

- Here, we have assumed that the array index varies from 1 to SIZE.
- A[TOP] is array A with TOP as the pointer.

Algorithms for POP operation:

- Steps:

```
1. If TOP = -1 then
2.         Print “Stack is Empty!”
3.         Exit
4.     Else
5.           ITEM = A[TOP]
6.           TOP = TOP – 1
7. EndIF
8. Exit
```

- Here, we have assumed that the array index varies from 1 to SIZE.
- A[TOP] is array A with TOP as the pointer.

Algorithms for STATUS operation:

- Steps:

```
1. If TOP =-1 then
2.         Print “Stack is Empty!”
3.     Else
4.         If (TOP =SIZE-1) then
5.             Print “Stack is Full!”
6.         Else
7.             Print “The element at TOP is”, A[TOP]
8.             free = SIZE-TOP_1
9.             Print “Percentage of free stack is”, free
10.        EndIF
11. EndIF
12. Exit
```

Application of stack

- 1. Evaluation of Arithmetic Expressions
- 2. Implementation of Recursion
    - a. Factorial Calculation
    - b. Quick Sort
    - c. Tower of Hanoi Problem

Evaluation of Arithmetic Expression

- An arithmetic expression consists of operands and operators.
- Operands are variables or constants and
- Operators are of various types such as…
- 1. Arithmetic Unary and Binary Operators
- 2. Relational Operators
- 3. Boolean Operators
- 4. In addition to these, parentheses such as “ (” and “) ” are also used.

- Operators are of various types such as…
- 1. Arithmetic Unary and Binary Operators

  e.g. - (Unary), +, -, *, /, ^, %, etc.
- 2. Relational Operators

  e.g. , <=, < >, >=, etc.
- 3. Boolean Operators

  e.g. AND, OR, NOT, XOR, etc.
- 4. In addition to these, parentheses such as “ (” and “) ” are also used.

- For example,
- A + B * C / D – E ^ F * G

Operators	Priority	Associativity
()	1	--
^	2	Right to left
*, /	3	Left to right
+,-	4	Left to right

- There are three notations to represent an arithmetic expression:
- 1. Infix
- 2. Prefix
- 3. Postfix
- The conventional way of writing an expression is called **infix.**
- For example: A + B, C – D, E * F, G/H, etc.
- This is called infix, because the operator comes in between the operands.

- 2. In **prefix**, the operator come before the operands.
- For example: +AB, –CD, *EF, /GH, etc.
- It was introduced by **Polish mathematician**, so it is also termed as **Polish Notation.**
- 3. In **postfix or suffix**, the operator is suffixed by operands.
- For example: AB+, CD-, EF, G/H, etc.
- This notation is just reverse of the Position notation. So, it is termed as **reversed Polish notation**

The following points may be observed from the three notations:

- 1. In both prefix and postfix equivalents of an infix expression, the variables are in the same relative positions.
- 2. The expressions in prefix or postfix form are completely parenthesis free.
- 3. The operators are rearranged according to the rules of precedence of operators.

Conversion of an Infix expression to Postfix expression:

- For example:
- 1. A + B * C
- 2. (A + B) * C
- 3. A + (B * C)
- 4. (A + B) / (C - D)
- 5. (A + (B * C)) / (D – (E * F))
- 6. (A + B) * C / D
- 7. (A + B) * (C - D) / E
- 8. A + B * C – D / E * F
- 9. (A + B * C - D) / (E * F)
- 10. (A + B) * C – D / (E * F)

Algorithm for Infix To Postfix conversion:

```
1. TOP = 0, PUSH( „(‟)
2. While (TOP > 0) do
3.         item = E.ReadSymbol()
4.         x = POP()
5.         Case: item = operand
6.             PUSH(x)
7.             Output(item)
8. Case: item = „)‟
9.             While x ≠ „(‟ do
10.                Output(x)
11.                x = POP
12. EndWhile
```

```
13. Case: ISP(x) ≥ ICP(item)
14.         While(ISP(x) ≥ ICP(item)) do
15.             Output(x)
16.             x = POP()
17.         EndWhile
18.         PUSH(x)
19.         PUSH(item)
20.     Case: ISP(x) < ICP(item)
21.         PUSH(x)
22.         PUSH(item)
23.     Otherwise: Print "Invalid Expression!"
24. EndWhile
25. Stop
```

Evaluation of a Postfix expression

- For example:
- 1. ABC*D/+ [A=2, B=3, C=4, D=6]
- 2. 5 6 2 + * 12 4 / -
- 3. 7 5 2 + * 4 1 5 - / -
- 4. 30, 5, 2, ^, 12, 6, /, +, -, 3
- 5. 4, 10, 5, +, *, 15, 3, 1, -

Algorithm for Evaluating the Postfix expression:

```
1. Append a special delimiter „#‟ at the end of the expression.
2. item = E.ReadSymbol() // Read the first symbol from E
3. While (item ≠ „#‟) do
4.     If (item = operand) then
5.         PUSH(item) // Operand is the first push into the stack
6.     Else
7.         op = item // The item is an operator
8.         y = POP() // The right-most operand of the current operator
9.         x = POP() // The left-most operand of the current operator
10.        t = x op y // Perform the operation with operator „op‟ and operands x, y
11.        PUSH(t) // Push the result into stack
12. EndIf
```

```
13.         item = E.ReadSymbol() // Read the next item from E
14.     EndWhile
15.         value = POP() // Get the value of the expression
16. Return(value)
17. Stop
```

Implementation of Recursion

- Recursion is an important tool to describe a procedure having several repetitions of the same.
- A procedure is termed recursion, if the procedure is defined by itself.
- For example:
- The implementation of three popular recursive computations are:
- 1. Calculation of factorial value
- 2. Quick sort
- 3. Tower of Hanoi problem

Factorial Calculation

- Algorithm for calculating the factorial for an integer N: Steps:

```
1. If (N = 0) then
2.         fact = 1 // Termination condition of repetition
3.     Else
4.         fact = N * Factorial(N - 1)
5. EndIf
6.        Return(fact) // Return the result
7. Stop
```

- Now, to implement this, we require two stacks: -
- One for storing the parameter N and
- - Another to hold the return address.
- No stack is necessary to store local variables, as the procedure does not possess any local variable.

Algorithm for calculating the factorial for an integer N with Stack

```
1. val = N, top = 0, addr = Step15 // N means N!
2. PUSH(val, addr) // Initialize the stack
3. val = val – 1, addr = Step 11 // Next value and return address
4. If (val = 0) then
5.         fact = 1 // Termination condition of repetition
6.         Go to Step 12
7.   Else
8.         PUSH(val, addr) // val pushed into PARAM and addr pushed into ADDR
9.          Go to Step 3
10. EndIf
11.          fact = val * fact
12.          val = POP_PARAM(), addr = POP_ADDR()
13.          Go to addr
14.          Return (fact)
15. Stop
```

Computation of a factorial (recursively) using a stack.

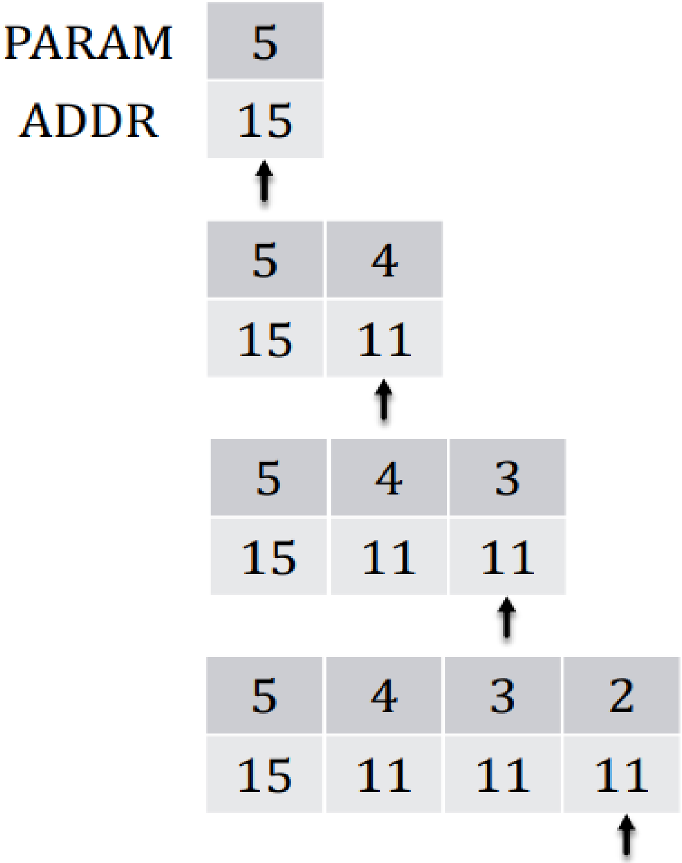
Val=5, top=0, addr=step15

PUSH(5, Step15)
Val=4, addr=Step11

Val ≠ 0
PUSH(4,step 11)

Val=3, addr=Step11
Val ≠ 0
PUSH(3, Step11)

Val=2, addr=Step11
Val ≠ 0
PUSH(2, Step11)



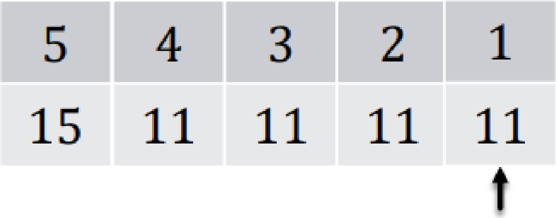
PARAM	5
ADDR	15

5	4
15	11

5	4	3
15	11	11

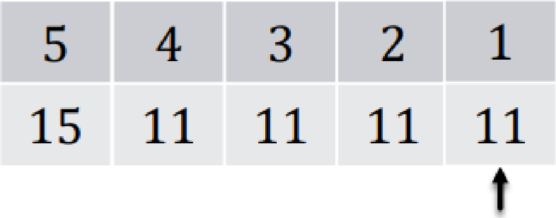
5	4	3	2
15	11	11	11

Val=1, addr=Step11
Val ≠ 0
PUSH(2, Step11)



5	4	3	2	1
15	11	11	11	11

Val=0, addr=Step11
Val = 0
Fact = 1

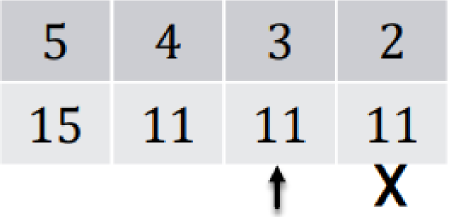


5	4	3	2	1
15	11	11	11	11

Val=1, addr=Step11
Fact = 1*1 (=1)

5	4	3	2	1
15	11	11	11	11

Val=2, addr=Step11
Fact = 2*1 (=2)

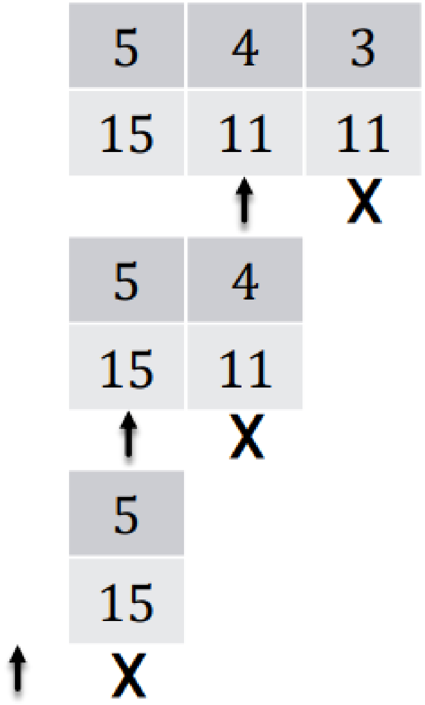


5	4	3	2
15	11	11	11

Val=3, addr=Step11
Fact = 3*2 (=6)

Val=4, addr=Step11
Fact = 4*6 (=24)

Val=5, addr=Step15
addr=Step11
Fact = 5*24 (=120)



Tower of Hanoi problem:

- Tower of Hanoi shows how recursion may be used as a tool in developing an algorithm to solve particular problem.
- This problem has a historical root in the ritual of an ancient tower of Brahma.
- The problem can be described as follows:
- There are three peg A, B and C. And suppose on peg A, there are placed „n‟ of disks with decreasing size. The aim of the game is to move the discs from peg A to peg C using peg B as an auxiliary.

The rules of the game are:

- 1. Only one disc may be moved at a time.
- 2. Disc may be moved from any peg to any other peg.
- 3. At no time can a larger disc be placed on a smaller disc.

Therefore, here we have 3 discs placed in peg A.

Then, the solution to the Tower of Hanoi problem for n=3…

<u>Step 01</u>: Move top disc from peg A to peg C

Step 02: Move top disc from peg A to peg B

Step 03: Move top disc from peg C to peg B

Step 04: Move top disc from peg A to peg C

Step 05: Move top disc from peg B to peg A

Step 06: Move top disc from peg B to peg C

Step 07: Move top disc from peg A to peg C

- The solution of this problem can be stated recursively as follows:
- Move N discs from peg A to C via the peg B means....
- \- Moving the first (N - 1) discs from peg A to B.
- \- Moving the disc from peg A to C.
- \- Moving all (N - 1) discs from peg B to C.

Algorithm for Moving the discs

- Steps:
- 1. If N > 0 then // If N = 0, then it will terminate
- 2. Move(N – 1, ORG, DES, INT)
- 3. ORG - > DES (Move from ORG to DES)
- 4. Move(N – 1, INT, ORG, DES)
- 5. EndIf
- 6. Stop

- Let us implement this recursion using stacks.
- For this purpose, we have to assume the following stacks:
- STN – is to store the number of discs.
- STA – is to store the peg of origin.
- STB – is to store the intermediate of peg.
- STC – is to store the destination of discs.
- STADD – for the return address.
- PUSH(N, A, B, C, R) and POP(N, A, B, C, R) are the two stack operations over these stacks and they are expressed as...

- PUSH(N, A,B,C, R)
- Top = Top + 1
- STN[Top] = N
- STA[Top] = A
- STB[Top] = B
- STC[Top] = C
- STADD[Top] = R

POP(N, A,B,C, R)
N = STN[Top]
A = STA[Top]
B = STB[Top]
C = STC[Top]
R = STADD[Top]
Top= Top – 1

Algorithm of Tower of Hanoi:

```
• Steps:
• 1. top = NULL // Initially all the stacks are empty
• 2. org = „A‟, int = „B‟, des = „C‟, n = N // Initialization of the
  parameters
• 3. Addr = Step 26 // Return to the end step
• 4. PUSH (n, org, int, des, add) // Push the initial value to the stacks
• 5. If (STN[top] = 0) then // Terminal condition reached
• 6.         Go to STADD[top]
• 7. Else  // Translation of Move(N – 1, A, C, B)
• 8.          n = STN[top] – 1
• 9.          org = STA [top]
• 10.        int = STC[top]
• 11.        des = STB[top]
• 12.        addr = Step 15 // After completing these moves return to
  Step 6
• 13.        Go to step 4
```

```
14. EndIf
15.     POP(n, org, ini, des, r)
16.     Print "Move disc from:" org -> des     // Move the nth disc from A to C
17.     Do the following:
18.     n = STN[top] – 1 // Translation of Move(N – 1, B, A, C)
19.     org = STB [top]
20.     int = STA[top]
21.     des = STC[top]
22.     addr = Step 24
23.     Go to step 4
24.     POP(n, org, ini, des, r)
25.     Go to r // Return address
26. Stop
```

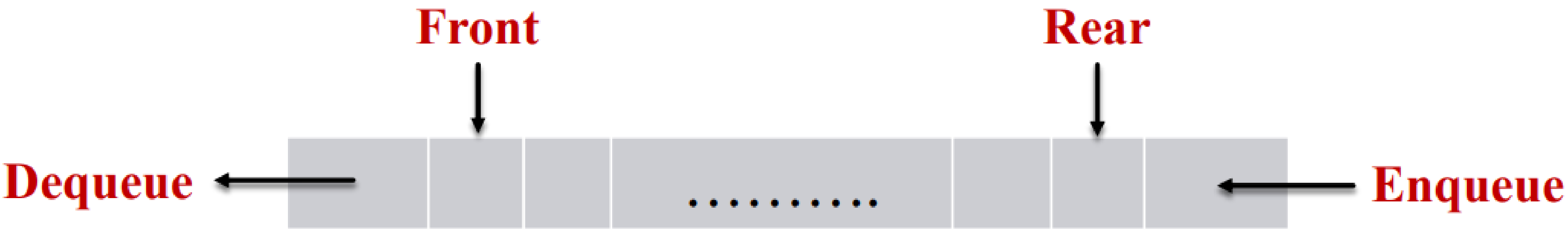
Queue & its operation

- **Introduction.**
- A queue is a linear data structure.
- It is a simple, but very powerful data structure to solve numerous computer applications.
- Like stack, queue is area also useful to solve various system programs.

- it is evident that a data in a queue is processed in the same order as it had entered, that is, on a first-in first-out basis.
- This is why a queue is also termed as…
- FIFO (First-In-First-Out).

What is Queue

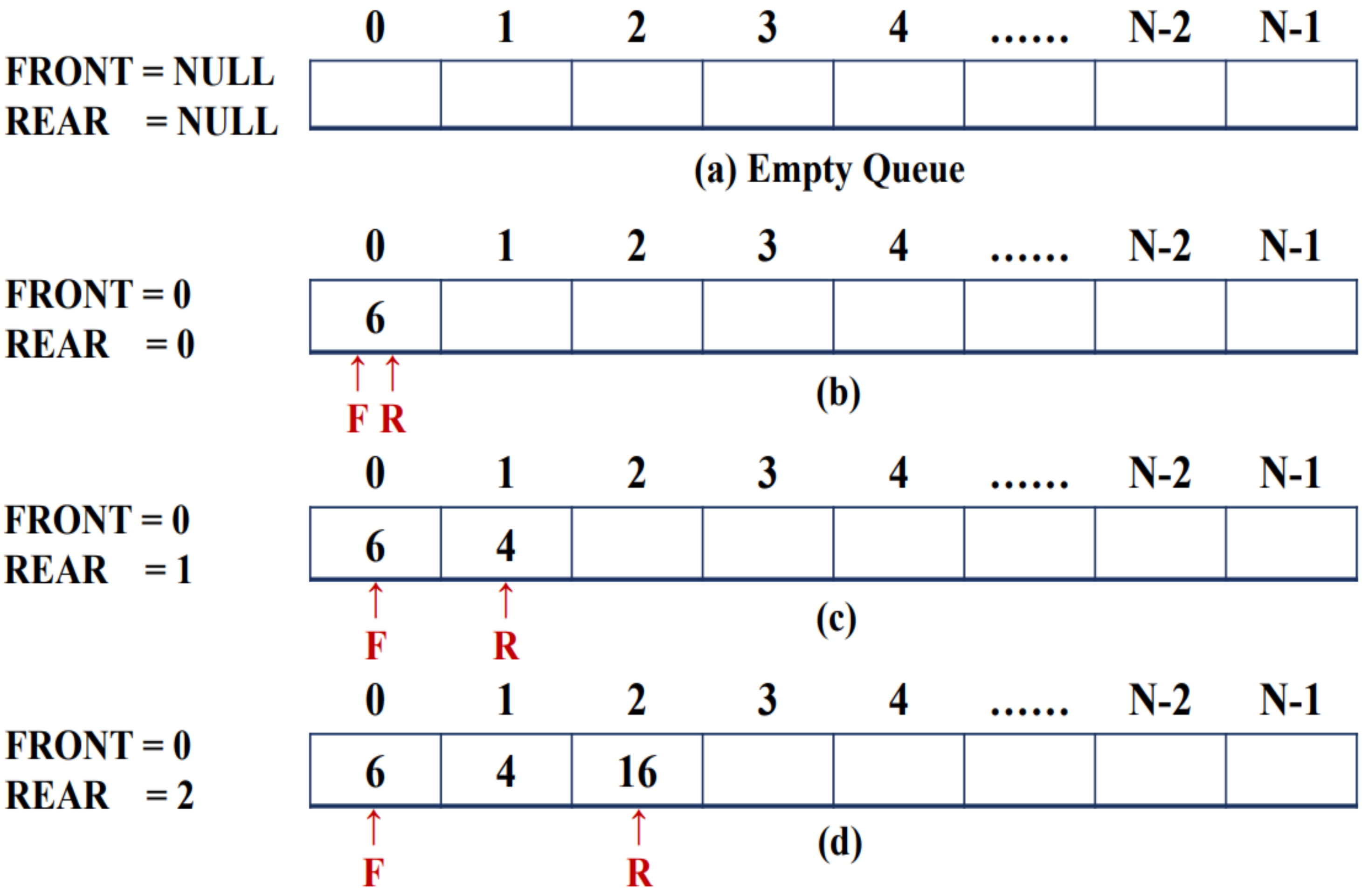
- A queue is an ordered collection of homogeneous data elements; in contrast with the stack, here the insertion and deletion operations take place at two extreme ends.
- In the case of a queue…
- The insertion operation are specially termed as ENQUEUE,
- The deletion operation are specially termed as DEQUEUE and
- The positions of the queue, where these operations are performed is known as the FRONT and REAR of the queue.



Model of a Queue

- In a queue, all insertion and deletion can take place at one end, which is know as FRONT pointer of the queue.
- And only insertion can take place at other end, which is know as REAR pointer of the queue.

- For example: Insertion in an Array Queue. (Index value from 0).



FRONT = NULL
REAR = NULL

0	1	2	3	4	……	N-2	N-1

(a) Empty Queue

FRONT = 0
REAR = 0

0	1	2	3	4	……	N-2	N-1
6							

(b)

FRONT = 0
REAR = 1

0	1	2	3	4	……	N-2	N-1
6	4						

(c)

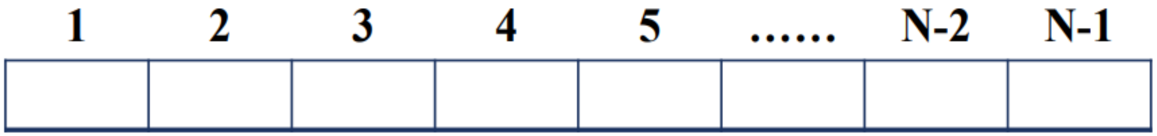
FRONT = 0
REAR = 2

0	1	2	3	4	……	N-2	N-1
6	4	16					

(d)

For example: Insertion in an Array Queue. (Index value from 1)

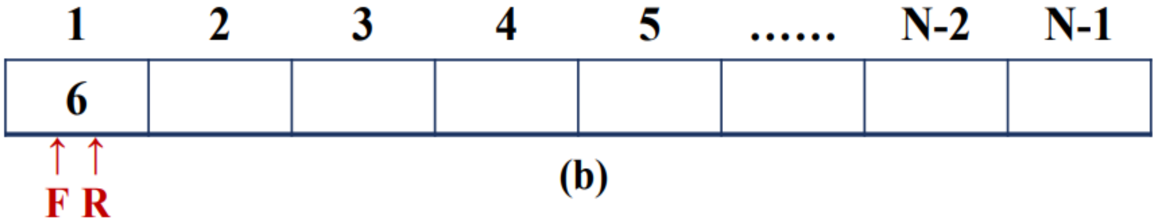
FRONT = 0
REAR = 0



1	2	3	4	5	……	N-2	N-1

(a) Empty Queue

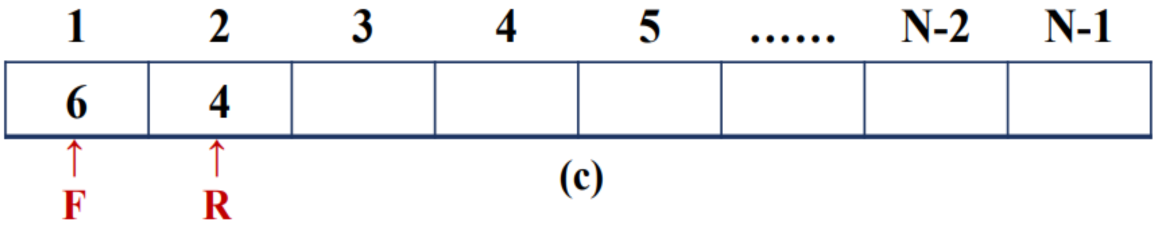
FRONT = 1
REAR = 1



1	2	3	4	5	……	N-2	N-1
6							

(b)

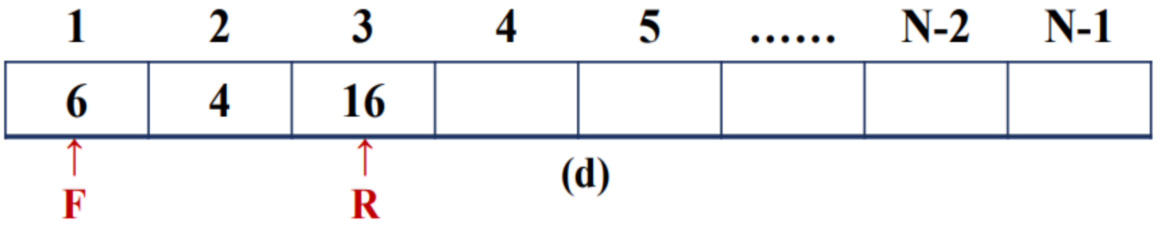
FRONT = 1
REAR = 2



1	2	3	4	5	……	N-2	N-1
6	4						

(c)

FRONT = 1
REAR = 3

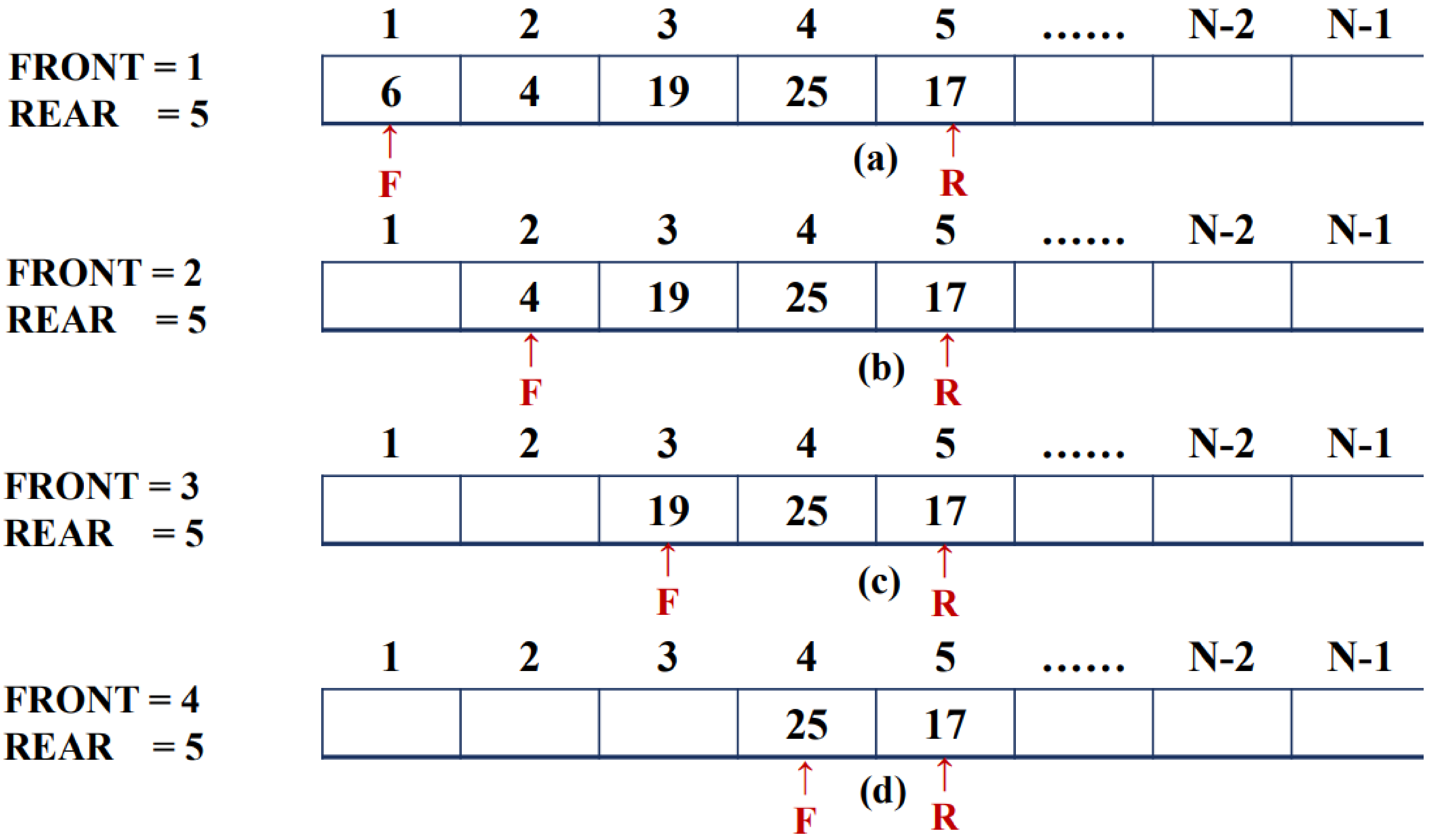


1	2	3	4	5	……	N-2	N-1
6	4	16					

(d)

- So, when Queue is full, then…
- FRONT = 1
- REAR = N

For example: Deletion in an Array Queue. (Index value from 1)



FRONT = 1
REAR = 5

1	2	3	4	5	……	N-2	N-1
6	4	19	25	17			
F				R			

(a)

FRONT = 2
REAR = 5

1	2	3	4	5	……	N-2	N-1
	4	19	25	17			
	F			R			

(b)

FRONT = 3
REAR = 5

1	2	3	4	5	……	N-2	N-1
		19	25	17			
		F		R			

(c)

FRONT = 4
REAR = 5

1	2	3	4	5	……	N-2	N-1
			25	17			
			F	R			

(d)

- So, when Queue is empty, then…
- FRONT = 0
- REAR = 0

- Therefore, a queue is a time sharing computer system, where…
- insertion is only possible at REAR pointer and
- deletion is only possible at FRONT pointer.
- After some deletion operation, when FRONT pointer and REAR pointer become same, then the last information is removed and no further deletion operation is possible.
- In the same way, when REAR pointer comes at last data of the queue (which means that no more insertion is possible.)

- Therefore, when we remove the information, FRONT pointer is increased by 1.
- And when at last FRONT and REAR pointer becomes same, then both FRONT and REAR pointer are assigned to 0 (which indicates that now again means queue is empty).
- In the same way, when we are inserting information, REAR is increased by 1 and check if REAR >= LENGTH or not.
- So, if REAR >= LENGTH, then no further insertion is possible

Operation in Queue

- The basic operations required to manipulate a Queue are:
- 1. Enqueue - To insert an item into a queue
- 2. Dequeue - To remove an item from a queue.

Algorithms for ENQUEUE operation:

- Steps:

```
1. If (REAR = N) then
2.         Print "Queue is full"
3.         Exit
4. Else
5.      If (REAR = 0) and (FRONT = 0) then
6.            FRONT = 1
7.            REAR = 1
7.      EndIf
8.            REAR = REAR + 1
9.            Q[REAR] = ITEM
10. EndIf
11. Exit
```

Algorithms for DEQUEUE operation:

- Steps:

```
1. If (FRONT = 0) then
2.        Print “Queue is empty”
3.        Exit
4. Else
5.        ITEM = Q[FRONT]
6.        If (FRONT = REAR)
7.              REAR = 0
8.              FRONT = 0
9.        Else
10.             FRONT = FRONT + 1
11.       EndIf
12. EndIf
13. Exit
```

Type of Queue

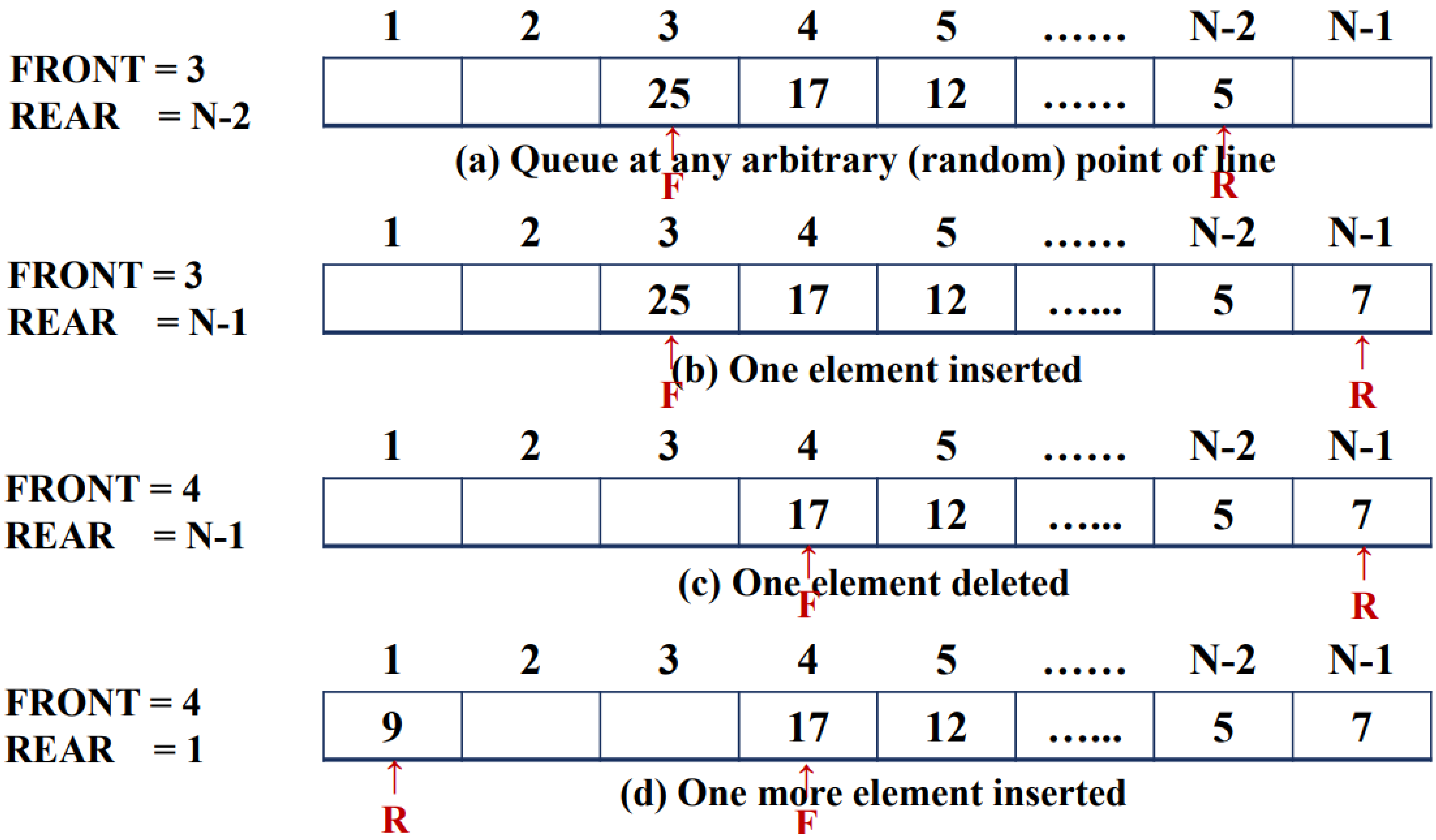
- There are various types of queues.
- They are:
- 1. Simple
- 2. Circular
- 3. Deque (Double-ended)
- 4. Priority.

Circular Queue

- Circular queue are the queue implemented in circular form rather than a straight line.
- Now, as we know that there was a major drawback for using simple queue, because...
- all the insertion is done at REAR pointer and
- all the deletion is only possible on other end which is at FRONT pointer.
- Therefore, after removing the items by FRONT pointer, these item‟s space becomes blank and always be blank only, because all insertions are taking place at REAR pointer.

- So, when the REAR pointer reaches the end, insertion will be denied even if room is available at the front.
- Therefore, one way to avoid this is to use a… circular array.
- Circular queue has overcome the problem of this unutilized space in linear queue implemented as arrays.
- In circular queue, if REAR = LENGTH of array and if insertion is done, then it is possible to insert more information as foremost items if any deletion take place

For example: Insertion and Deletion in a Circular Queue.



FRONT = 3
REAR = N-2

1	2	3	4	5	……	N-2	N-1
		25	17	12	……	5	
		↑ F				↑ R	

(a) Queue at any arbitrary (random) point of line

FRONT = 3
REAR = N-1

1	2	3	4	5	……	N-2	N-1
		25	17	12	……	5	7
		↑ F					↑ R

(b) One element inserted

FRONT = 4
REAR = N-1

1	2	3	4	5	……	N-2	N-1
			17	12	……	5	7
			↑ F				↑ R

(c) One element deleted

FRONT = 4
REAR = 1

1	2	3	4	5	……	N-2	N-1
9			17	12	……	5	7
↑ R			↑ F				

(d) One more element inserted

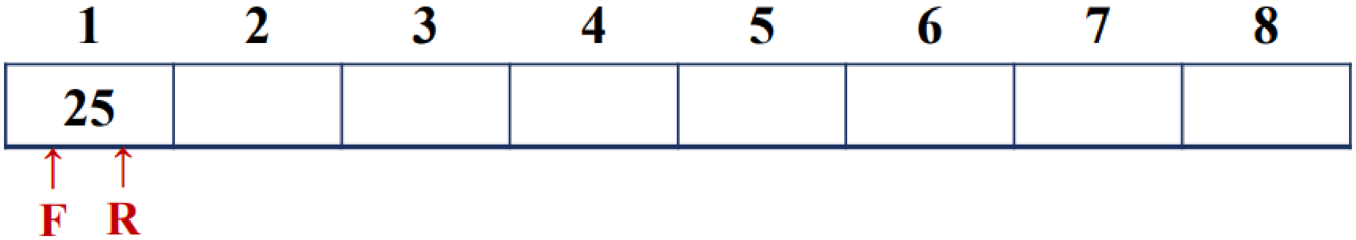
- So, when Circular Queue is empty, then… FRONT = 0 REAR = 0
- And, when Circular Queue is full, then… FRONT = (REAR MOD LENGTH) + 1

Algorithms for ENQUEUE operation in a Circular Queue:

```
1. If (FRONT = 0) then
2.       FRONT = 1
3.       REAR = 1
4.       CQ[REAR] = ITEM
5. Else
6.       next = (REAR MOD LENGTH) + 1
7.       If (next ≠ FRONT) then
8.             REAR = next
9.             CQ[REAR] = ITEM
10.      Else
11.            Print "Queue is full"
12.      EndIf
13. EndIf
14. Exit
```

For Example: ENQUEUE operation in a Circular Queue

FRONT = 1
REAR = 1



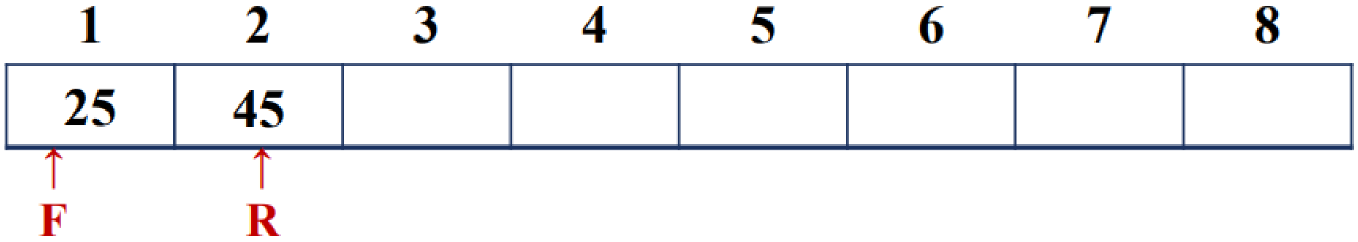
1	2	3	4	5	6	7	8
25							

```
next = (REAR MOD LENGTH) + 1
     = (1 % 8) + 1 = 2
```

```
If (next ≠ FRONT) then
        REAR = next
        CQ[REAR] = ITEM

If (2 ≠ 1) then
        REAR = 2
        CQ[REAR] = ITEM
```

FRONT = 1
REAR = 2



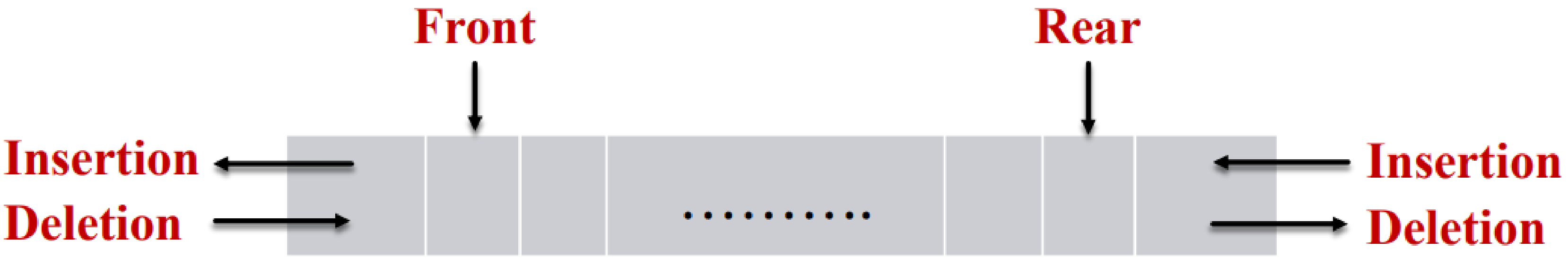
1	2	3	4	5	6	7	8
25	45						

Algorithms for DEQUEUE operation in a Circular Queue:

```
1. If (FRONT = 0) then
2.     Print "Queue is empty"
3.     Exit
4. Else
5.     ITEM = CQ[FRONT]
6.     If (FRONT = REAR)
7.         FRONT = 0
8.         REAR = 0
9.     Else
10.        FRONT = (FRONT MOD LENGTH) + 1
11.    EndIf
12. EndIf
13. Exit
```

3. Deque:

- The term deque has originated from Double Ended Queue.
- Unlike a queue, in deque, both insertion and deletion operations can be performed at both ends of the structure.



A Deque Structure

- Therefore, we can say that, a deque can be used as a Stack as well as Queue.
- There are two known variations of deque.
- They are: 1. Input-restricted deque
- 2. Output-restricted deque
- These two types of variations are intermediate between a queue and a deque.

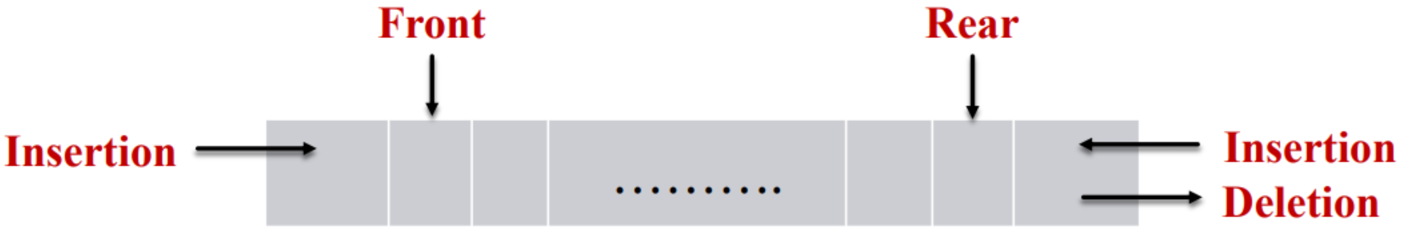
1. Input-restricted deque:

- An input-restricted deque is a deque which allows insertions at one end only (i.e. at REAR), but allows deletion at both the ends.

Input-restricted deque

Output-restricted deque:

- An output-restricted deque is a deque where deletions take place at one end only (i.e. at FRONT), but allows insertions at both ends.

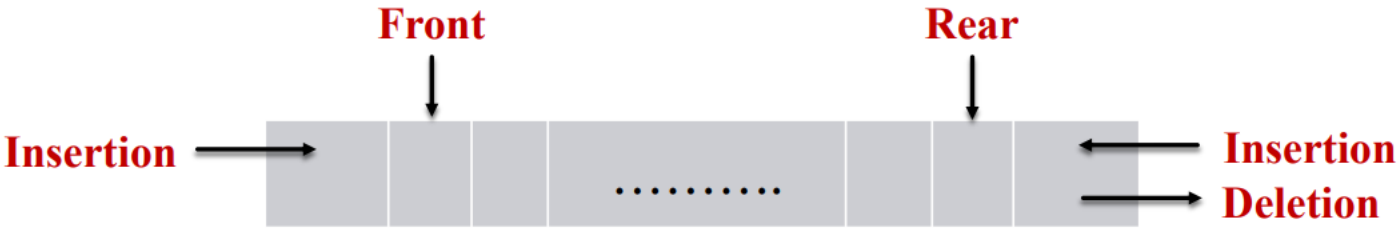


Output-restricted deque

- There are four operations possible on a deque. They are:
- 1. Push - To insert ITEM at the FRONT end of a deque.
- 2. POP - To remove the FRONT item from a deque.
- 3. Inject - To insert the ITEM at the REAR end of a deque.
- 4. Eject. - To remove the REAR ITEM from a deque.

Algorithms for PUSH operation in a Deque:

```
If (FRONT = 1 and REAR = Length or FRONT = REAR + 1) then
    Print “Deque is full”
    Exit
Else
    If (FRONT = 0) then
        FRONT = 1
        REAR = 1
        DQ[REAR] = ITEM
    Else
        If (FRONT = 1) then
            FRONT = Length
        Else
            FRONT = FRONT – 1
        EndIf
            DQ[FRONT] = ITEM
    EndIf
EndIf
STOP
```



Output-restricted deque

- There are four operations possible on a deque. They are:
- 1. Push - To insert ITEM at the FRONT end of a deque.
- 2. POP - To remove the FRONT item from a deque.
- 3. Inject - To insert the ITEM at the REAR end of a deque.
- 4. Eject - To remove the REAR ITEM from a deque.

Algorithms for PUSH operation in a Deque:

- Steps:

```
1. If (FRONT = 1 and REAR = Length or FRONT = REAR + 1) then
2.       Print "Deque is full"
3.       Exit
4. Else
5.       If (FRONT = 0) then
6.            FRONT = 1
7.            REAR = 1
8.            DQ[REAR] = ITEM
9.       Else
10.           If (FRONT = 1) then
11.                FRONT = Length
12.           Else
13.                FRONT = FRONT – 1
14.           EndIf
15.           DQ[FRONT] = ITEM
16.      EndIf
17. EndIf
18. STOP
```

- **2. Algorithms for POP operation in a Deque:**
- This algorithm is same as the algorithm of DEQUEUE operation in a Circular Queue.
- **3. Algorithms for Inject operation in a Deque:**
- This algorithm is same as the algorithm of ENQUEUE operation in a Circular Queue.

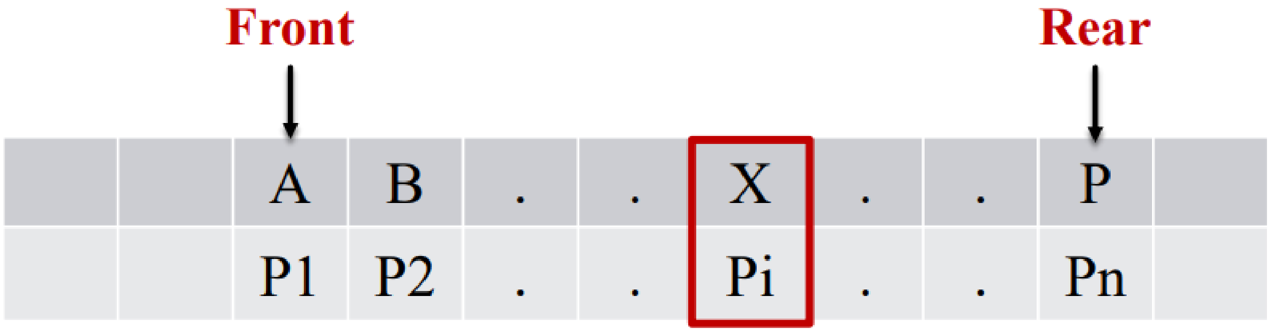
Algorithms for Eject operation in a Deque:

- Steps:

```
1. If (FRONT = 0) then
2.        Print “Deque is empty”
3.        End
4. Else
5.        If (FRONT = REAR) then
6.             ITEM = DQ[REAR]
7.             FRONT = REAR = 0
8.        Else
9.             ITEM = DQ[REAR]
10.            If (REAR = 1) then
11.                 REAR = LENGTH
12.            Else
13.                 REAR = REAR – 1
14.            EndIf
15.       EndIf
16. EndIf
17. STOP
```

4. Priority Queue:

- In a priority queue, each element has been assigned a value, called the priority of the element; and an element can be inserted or deleted not only at the ends, but at any position on the queue.



		A	B	.	.	X	.	.	P	
		P1	P2	.	.	Pi	.	.	Pn	

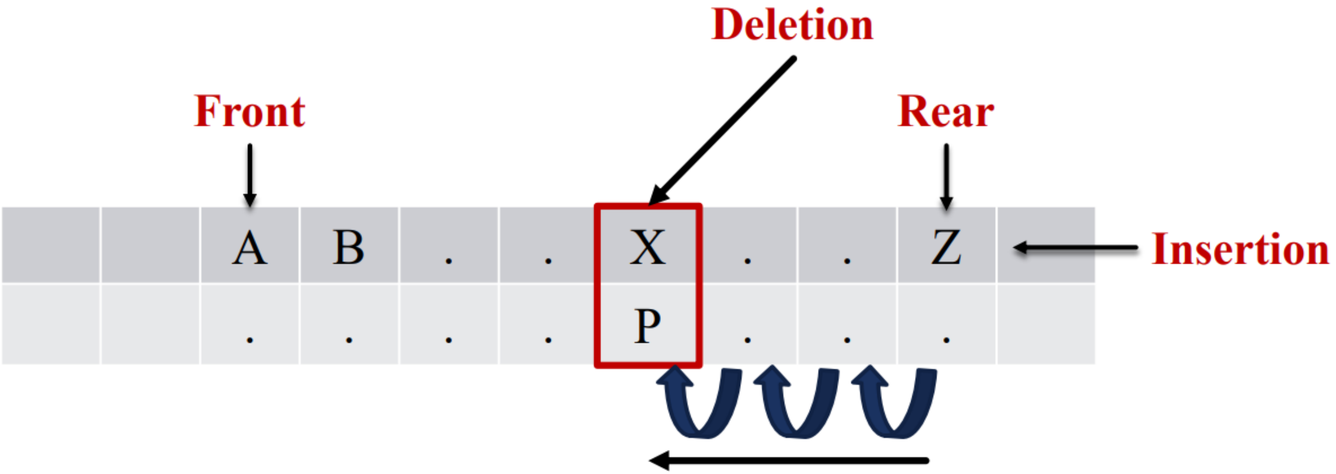
View of a priority queue

- Now, the name priority queue is a misnomer (contradiction) in the sense that the data structure is not a queue as per the definition.
- Because a priority queue does not strictly follow first-in first-out (FIFO) principle which is the basic principle of a queue.
- There are various models of priority queue known in different applications.

- For example:
- In a particular model of priority queue,
- 1. An element of higher priority is processed before any element of lower priority.
- 2. Two elements with the same priority are processed according to the order in which they were added to the queue.
- For example:
- A time sharing system - In a time sharing system, the programs of the higher priority are process first.

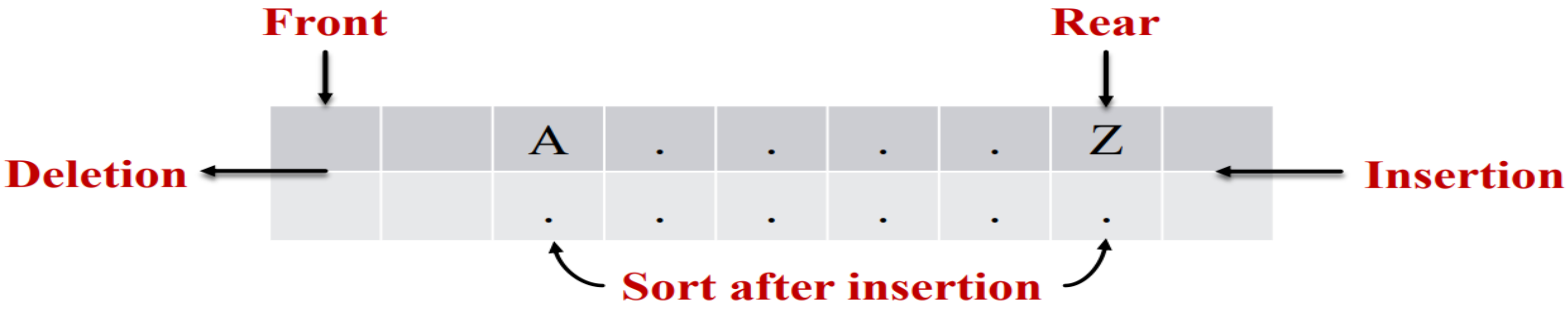
- Here, process means two basic operations that is insertion or deletion.
- There are various ways of implementing the structure of a priority queue.
- They are:
- 1.Using a simple/circular array
- 2.Mutli-queue implementation
- 3.Using a double linked list
- 4.Using heap tree.

- Priority Queue: (Conti…)
- But how it works???
- The element will be inserted at the REAR end as usual.
- But the deletion operation will be performed in either of the following ways:
- 1. Starting from the FRONT pointer, traverse the array for an element of the highest priority. Delete that element from the queue.
- If that element is not the front-most element, then shift all the elements after the deleted element, once stroke each to fill up the vacant position.



- **Deletion operation in an array representation of a priority queue**
- However, this implementation is very inefficient as it involves searching the queue for the highest priority element and shifting all the elements after the deletion.

- Therefore, the other better implementation is…
- 2. Add the elements at the REAR end as earlier. Then by using a stable sorting algorithm, sort the elements of the queue, so that the highest priority element is at the FRONT end.



Another array implementation of a priority queue

Application of queue

- There are numerous applications of queue structures known in computer system.
- 1. One major application of queues is in simulation. (e.g. Simulation of a Traffic Control System)
- 2. Another important application of queues is observed in the implementation of various aspects of an operating system.
- 3. A multiprogramming environment uses several queues to control various programs. (e.g. CPU scheduling)
- 4. Queues are very much useful to implement various algorithms like scheduling algorithms. (e.g. Round Robin algorithm)

- 5. To implement printer spooler, so that jobs can be printed in the order of their arrival.
- 6. All types of customer service centers are designed using the concept of queues. (e.g. Railway Reservation, Airway Reservation)