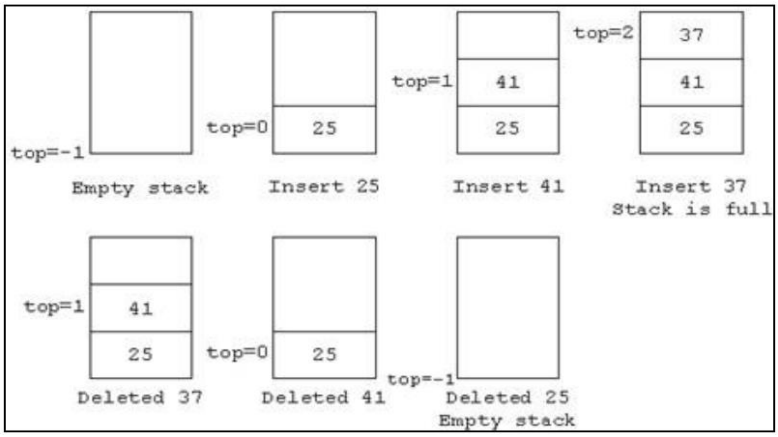
Chapter 2. Stack and Queue

Stack:

- Stack is an ordered collection of data in which insertion and deletion operation is performed at only one end called the **Top of the Stack (TOS)**.
- The element last inserted will be the first to be deleted. Hence, stack is known as Last in First Out (**LIFO**) structure.
- Stack Operations:
    - **Push:** It is used to add an item in the stack. If the stack is full, then it is said to be **overflow condition.**
    - **Pop:** It is used to remove an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an **Underflow condition.**
- Initial value of top is -1 i.e., **top = -1**
- Example: Stack Operations for the size of stack 3



Operations:

Push Operation:-

```
If (top == (max-1)]
        Printf("stack over flow");
Else
{
Top == top + 1;
Stack_arr[top] = pushed –item;
}
```

Pop Operation:-

```
If (top ==-1)
        Printf("stack underflow");
Else
{
        Printf("Poped element is %d",stack_arr[top]);

            Top=top -1;

 }
```

Algorithm for Push and Pop Operation:

Push:

1. Start
2. If(top=size-1) // Check for stack overflow
   Print" Stack is full"
3. Otherwise,
   i. Increase top by 1
   ii. Read data and store at top position
4. Stop

Pop:

1. Start
2. If(top=-1) //Check for stack Underflow

   Print" Stack is empty"

   Otherwise,

   Decrease top by 1
3. Stop

Stack Application: Evaluation of infix, postfix and prefix expressions

- An expression is defined as number of operands or data items combined with several operators.
- Three types of notations for an expression:
    - **Infix notation:**
        - An expression where operators are used in-between operands. e.g., A+B
        - It is easy for us human to read, write, and speak in infix notation but the same does not go well with computing devices.
    - **Prefix Notation:**
        - An expression where the operator is written ahead of operands. e.g., +AB
    - **Postfix Notation:**
        - An expression where the operator is written after the operands. e.g., AB+
- Operator and their precedence level:
    - Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence.

Operator	**Precedence**	**Value**
Exponentiation (**$, ^**)	Highest	3
***, /, %**	Next highest	2
+, -	Lowest	1

Note: **[{( )}]** will be evaluated first.

Algorithm to convert infix to postfix:

1. Let Infix be a string that stores infix expression and Postfix be a string that stores the postfix result.
   Scan the Infix Expression character from left to right.
   1. Start
   2. If character is operand

      Append it to postfix

   3. If character is operator
      i. If stack is empty OR stack's top is '(' OR precedence of character > precedence of stack's top

         Push character to the stack

      ii. Else

         While stack's top is operator AND precedence of stack's top >= precedence of character

            Append the operator from stack to Postfix and pop it from stack

         Push character to the stack

   4. If character is'('

      Push character to the stack.

   5. If character is ')'

      Append the operator from stack to postfix and pop if from stack until stack's top is '('

      Pop '('from stack

   6. Repeat 2 to 5 until infix expression is scanned.
   7. Append the operator from stack to postfix and pop it from stack until stack is empty.
   8. stop

Example 1: Infix Expression: (A +B) *C

Scanned Character	Stack	Postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
)		AB+
*	*	AB+
C	*	AB+C
		AB+C*

Postfix: **AB+C***

Example 2: Infix Expression: $\boldsymbol{A + B - (C * D/E + F) - G * H}$ **[2073 Bhadra]**

Scanned Character	Stack	Postfix
A		**A**
+	**+**	**A**
B	**+**	**AB**
-	**-**	**AB+**
(	**-(**	**AB+**
C	**-(**	**AB+C**
*****	**-(***	**AB+C**
D	**-(***	**AB+CD**
/	**-(/**	**AB+CD***
E	**-(/**	**AB+CD*E**
+	**-(+**	**AB+CD*E/**
F	**-(+**	**AB+CD*E/F**
)	**-**	**AB+CD*E/F+**
-	**-**	**AB+CD*E/F+-**
G	**-**	**AB+CD*E/F+-G**
*****	**-***	**AB+CD*E/F+-G**
H	**-***	**AB+CD*E/F+-GH**
		AB+CD*E/F+-GH*-

Q. $A + B - C * (D - E + F/G)/H$

Q. $A * B/C - D + (E/F * G)/(K - L)$

Algorithm to convert Infix to Prefix:

1. Reverse the infix expression
2. Obtain the "nearly" postfix expression of the modified expression
3. Reverse the postfix expression

Example 1: Infix expression: (A +B) *C

2. Reversing the infix expression

   C * (B+A)

3. Converting modified expression to postfix expression

Scanned Character	Stack	Postfix
C		C
*	*	C
(	*(	C
B	*(	CB
+	*(+	CB
A	*(+	CBA
)	*	CBA+
		CBA+*

Postfix: **CBA+***

4. Reversing the postfix expression

   ***+ABC**

Algorithm to evaluate the postfix expression:

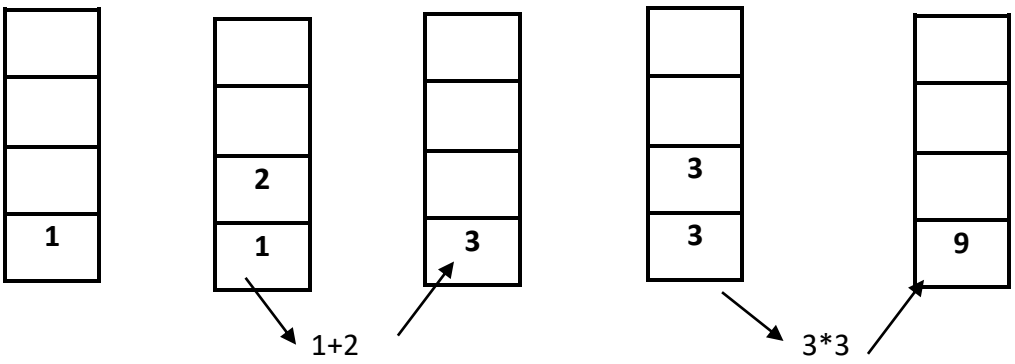
1. Start
2. If character is number

   Push on the stack

3. If character is operator(op)
   a. Val1=pop
   b. Val2=pop
   c. Perform result=val2 op val1
   d. Push the result into stack
4. Repeat 2 to 3 until postfix expression is scanned.
5. Output the result
6. Stop

Example 1: AB+C*

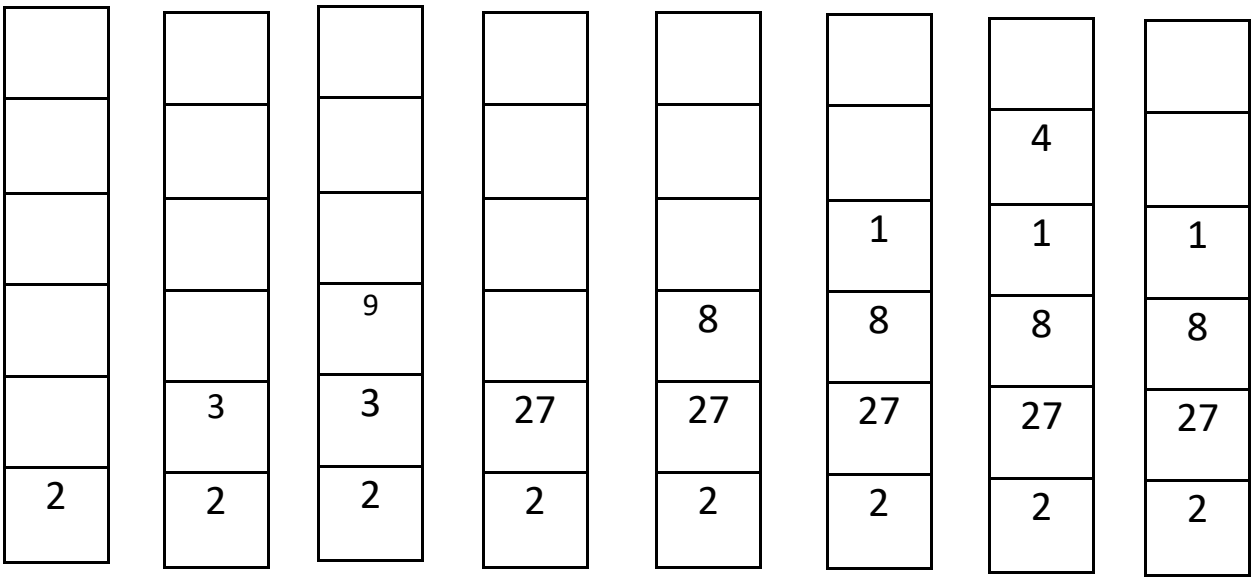
Let A=1, B=2 and C=3

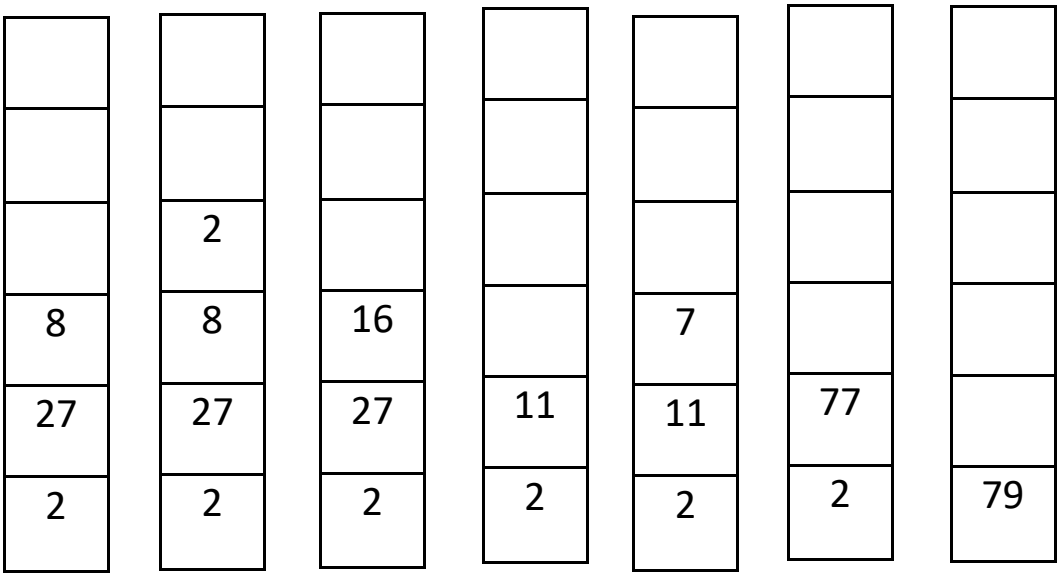


Ans: 9

Example 2: ABC*DEF^/G*-H*+

Let A= 2, B= 3, C=9, D=8, E=1, F=4, G= 2, H=7

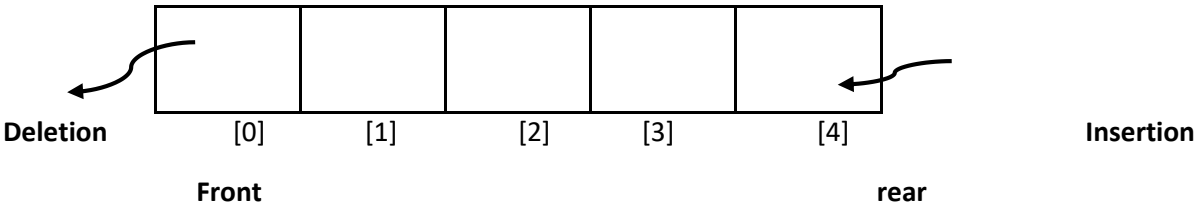




Ans: 79

Queue:

- A queue is an ordered collections of items from which items may be deleted at one end called the front of the queue and in to which items may be inserted at the other end called rear of the queue.
- It follows FIFO policy.
- **Queue operations:**
  - **Enqueue:** It adds an item to the queue. If the queue is full, then it is said to be an overflow condition.
  - **Dequeue:** It removes an item from the queue. The items are removed in the same order in which they are added. If the queue is empty, then it is said to be an Underflow condition.



Linear Queue:

- A linear queue is a linear data structure that serves the request first, which has been arrived first. It consists of data elements which are connected in a linear fashion.
- Initial condition:

  Front=0 and Rear=-1
- **Algorithm for Insertion:**
  1. Start
  2. If (rear = = MaxSize -1)

     Printf("queue overflow");
  3. Else

     If (front = = -1)

     Front = 0;

     Rear = rear + 1;

     Queue [rear] =item;
  4. Stop

- **Algorithm for Deletion:**
  1. Start
  2. If (front = = -1)|| (front >rear)

     Printf ("Queue under flow");

     Return;

  3. Else

     Printf ("Element detected from queue", queue [front]);

     front = front +1;

  4. Stop

Example:

REAR =-1 and FRONT=0

After 1 insertion REAR=0 and insert item 10 into queue.

REAR =0 and FRONT=0

10				

Now insert 20 into queue

REAR =1 and FRONT=0

10	*20*			

Suppose now we delete one item from queue, as we know that deletion can be done from FRONT end in queue.
Now, REAR =1 and FRONT=1

	20			

Now we insert 30, 40 and 50 into queue respectively.
REAR =2 and FRONT=1

	20	*30*		

REAR =3 and FRONT=1

	20	*30*	*40*	

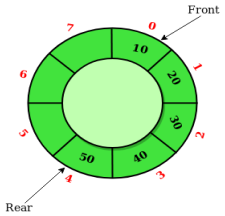
REAR =4 and FRONT=1

	20	*30*	*40*	*50*

Circular Queue:

- A circular queue is one in which the insertion of new element is done at the very first location of the queue if the last location of the queue is full.
- It connects the last position of the queue to the first position of the queue.
- Initial Condition:

  Front=-1 and rear =-1



- **Algorithm for Insertion:**

1. Start
2. if((front == 0 && rear == MAX-1) || (front == rear+1))

   Print "Queue Overflow" and return

3. otherwise
   i. if(front == -1)

      Set front = 0 and rear = 0

   ii. otherwise

      Increase rear as, rear = (rear + 1) % MAX
4. Insert the item rear position cqueue[rear] = item
5. Stop
6.

- **Algorithm for Deletion:**

1. Start
2. if(front == -1)

   Print "Queue Underflow" and return

3. otherwise
4. Element deleted from queue is cqueue[front]
   i. if(front == rear)

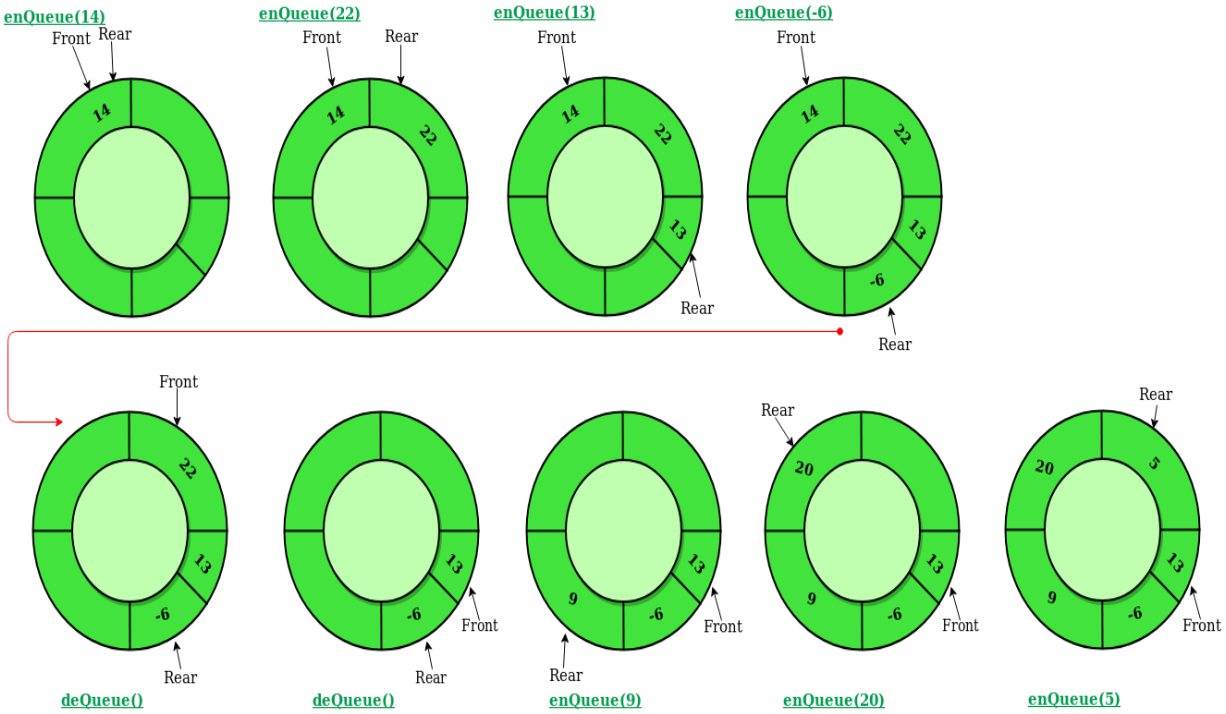
      Set front = -1 and rear = -1

   ii. otherwise

      Increase front as, front = (front + 1) % MAX

5. Stop

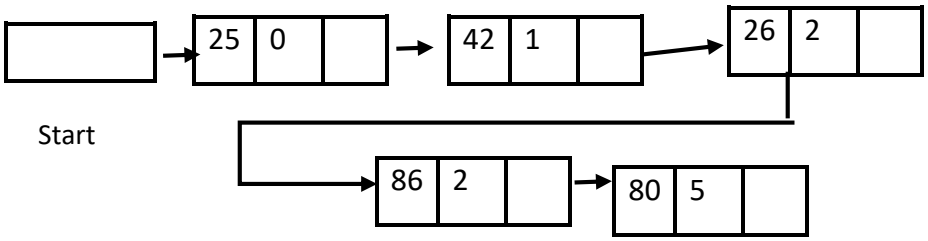
Example:



Priority Queue:

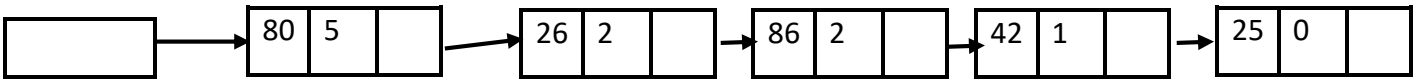
- Priority queue is an extension of queue with following properties:
  - Every item has a priority associated with it.
  - An element with high priority is dequeued before an element with low priority.
  - Two elements have the same priority, they are served according to their order in the queue.
- **Ascending Priority queue:**
  - An ascending priority queue is a collection of items into which items can be inserted arbitrarily and from which only the smallest items can be removed.
  - Lower priority number has higher priority.
  - For example: A queue may be viewed as ascending priority queue whose elements are ordered by the time of insertion. 0 has high priority

Item	Priority
25	0
80	5
26	2
42	1
86	2



- **Descending Priority Queue:**
  - A descending priority queue is similar but allows deletion of only the largest items.
  - Higher priority number to high priority
  - For example: A stack may be viewed as **descending priority** queue whose elements are ordered by the time of insertion. The element that was inserted last has the greatest insertion –time value and is the only that can be retrieved.
  -

Item	Priority
25	0
80	5
26	2
42	1
86	2



Start

Application of Priority Queue:

- It is used in data compression techniques like Huffman code.
- Priority queues are used to select the next process to run.
- It is used in bandwidth management to prioritize the important data packet.
- Used in algorithms like Dijkstra's shortest path algorithm, heap sort algorithm, etc.