Linear Data Structure

Definition

A **data structure** is said to be ***linear*** if its elements form a sequence or a **linear** list.

Examples:

- Array
- Linked List
- Stacks
- Queues

Operations on linear Data Structures

- *Traversal* : Visit every part of the **data structure**
- *Search* : Traversal through the data structure for a given element
- *Insertion* : Adding new elements to the **data structure**
- *Deletion:* Removing an element from the **data structure**.
- *Sorting* : Rearranging the elements in some type of order(e.g. Increasing or Decreasing)
- *Merging* : Combining two similar **data** structures into one

STACK – a pile of elements

Introduction:

1. Stack is basically a data object
2. The operational semantic (meaning) of stack is LIFO i.e. last in first out

   Definition: It is an ordered list of elements n, such that n>0 in which all insertions and deletions are made at one end called the top.

   Primary operations defined on a stack:
   1. PUSH : add an element at the top of the list.
   2. POP : remove at the top of the list.
   3. Also "IsEmpty()" and IsFull()" function, which tests whether a stack is empty or full respectively.

Example:

1. **Practical daily life :** a pile of heavy books kept in a vertical box, dishes kept one on top of another

2. **In computer world:** In processing of subroutine calls and returns ; there is an explicit use of stack of return addresses.
   Also in evaluation of arithmetic expressions, stack is used.

   Large number of stacks can be expressed using a single one dimensional stack only. Such an array is called a multiple stack array.

PROBLEM – 1

Tower of Hanoi

Tower of Hanoi is a historical problem, which can be easily expressed using recursion. There are N disks of decreasing size stacked on one needle, and two other empty needles. It is required to stack all the disks onto a second needle in the decreasing order of size. The third needle can be used as a temporary storage. The movement of the disks must confirm to the following rules -

1. Only one disk may be moved at a time
2. A disk can be moved from any needle to any other.
3. The larger disk should not rest upon a smaller one.

Question: write a c program to implement tower of Hanoi using stack?

Solution: /* Program of towers of Hanoi. */

```
#include <stdio.h>
#include <conio.h>

void move ( int, char, char, char ) ;

void main( )
  {
    int n = 3 ;
    clrscr( ) ;
    move ( n, 'A', 'B', 'C' ) ;
    getch( ) ;
  }

  void move ( int n, char sp, char ap, char ep )
    {
        if ( n == 1 )
        printf ("\nMove from %c to %c ", sp, ep ) ;
        else
          {
            move ( n - 1, sp, ep, ap ) ;
```

```
            move ( 1, sp, ' ', ep ) ;
            move ( n - 1, ap, sp, ep ) ;
        }
    }
```

PROBLEM – 2

Function Calls and Stack

A stack is used by programming languages for implementing function calls. Write a program to check how function calls are made using stack.

Solution: /* To show the use of stack in function calls */

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

unsigned int far *ptr ;
void ( *p )( void ) ;

void f1( ) ;
void f2( ) ;

void main( )
{
    f1( ) ;
    f2( ) ;
    printf ( "\nback to main..." ) ;
    exit ( 1 ) ;
}

void f1( )
{
    ptr = ( unsigned int far * ) MK_FP ( _SS, _SP + 2 ) ;
    printf ( "\n%d", *ptr ) ;
    p = ( void ( * )( ) ) MK_FP ( _CS, *ptr ) ;
    ( *p )( ) ;
    printf ( "\nI am f1( ) function " ) ;
}

void f2( )
```

```
{
        printf ( "\nI am f2( ) function" ) ;
}
```

PUSH & POP

Algorithms

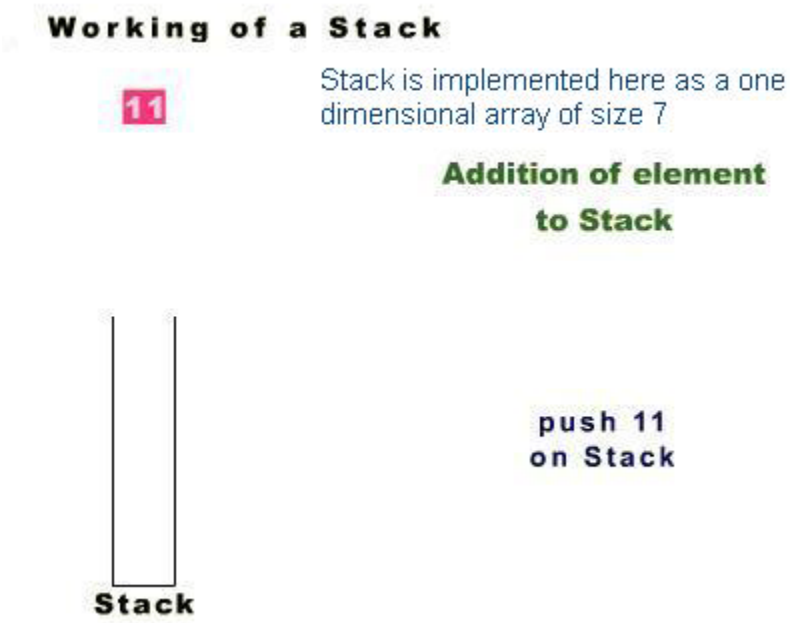
***Push* (item,array , n, top)**

```
        {
        If ( n> = top)
                Then print "Stack is full" ;
        Else
                {
                        top = top + 1;
                        array[top] = item ;
                }
        }
```

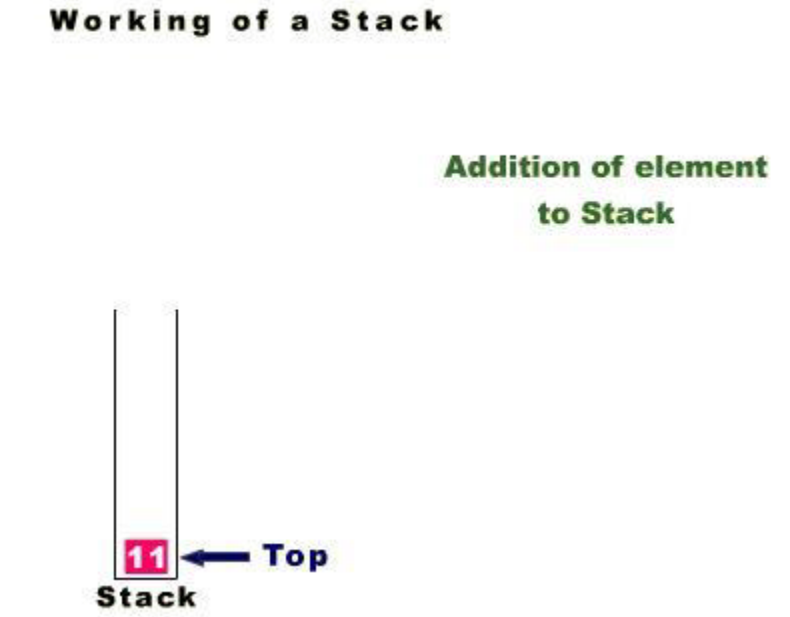
Pop (item,array,top)

```
        {
                if ( top<= 0)
                        Then print " stack is empty".
                Else
                        {
                                item = array[top];
                                top = top - 1;
                        }
        }
```

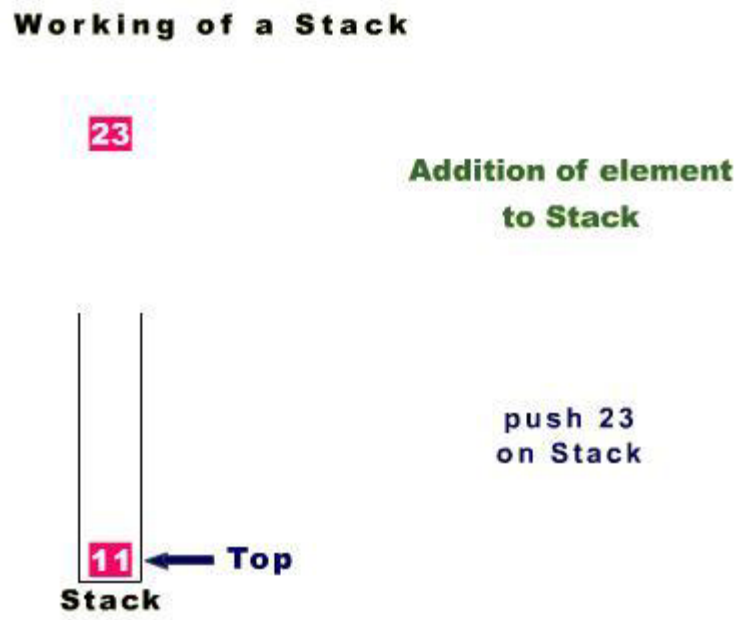
PUSH OPERATION



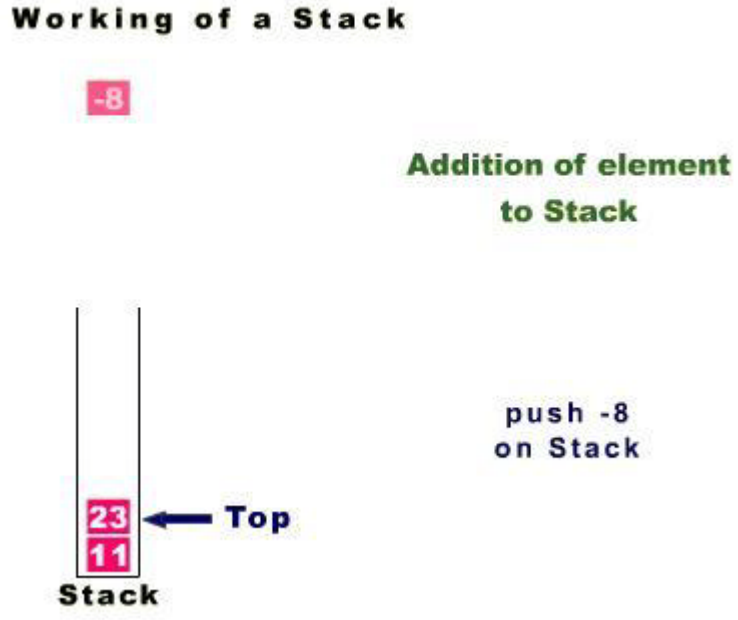
Step - 1



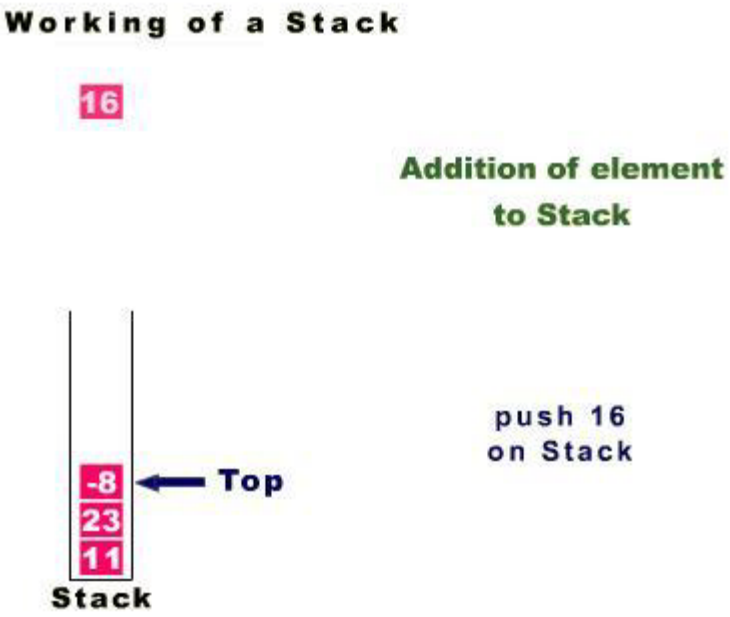
Step – 2


Working of a Stack
23
Addition of element
to Stack
push 23
on Stack
11
Top
Stack

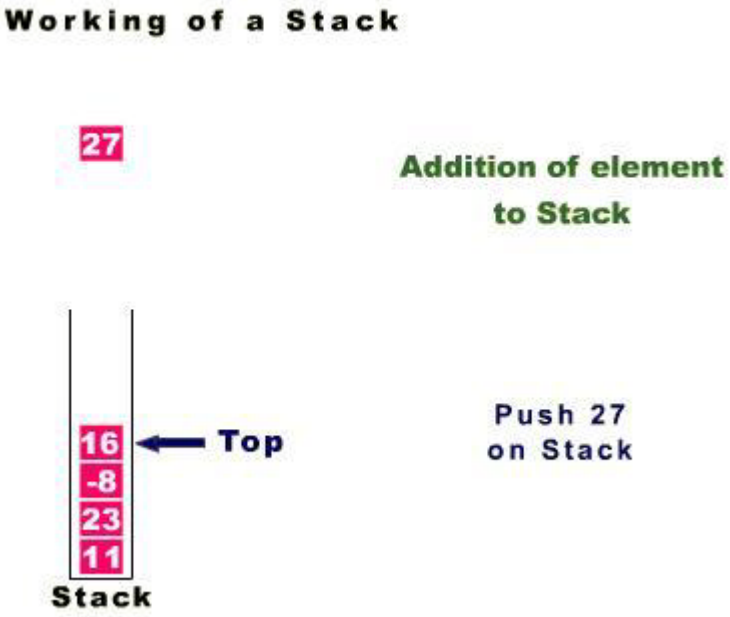
Step – 3


Working of a Stack
-8
Addition of element
to Stack
push -8
on Stack
23
11
Top
Stack

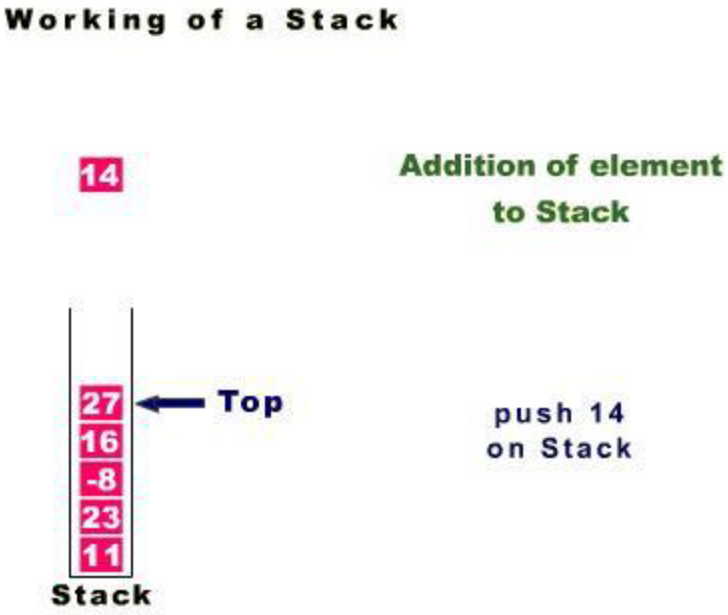
Step – 4


Working of a Stack
16
Addition of element
to Stack
-8
23
11
Top
Stack
push 16
on Stack

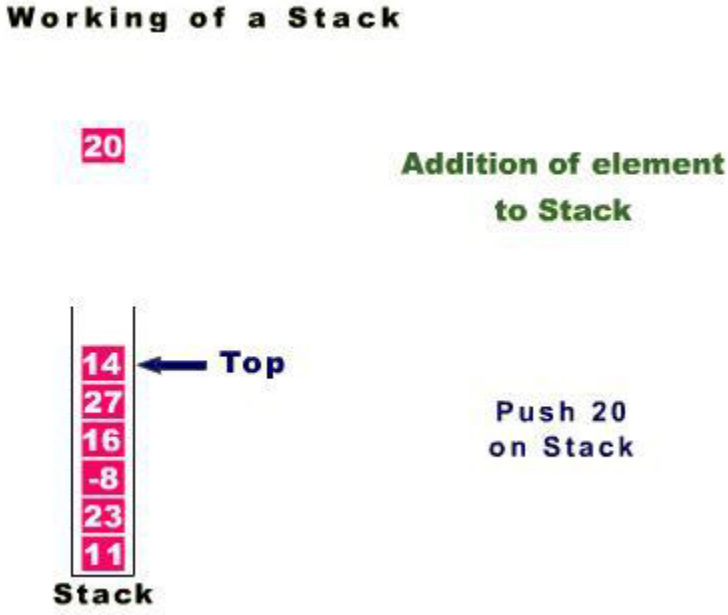
Step – 5


Working of a Stack
27
Addition of element
to Stack
16
-8
23
11
Top
Stack
Push 27
on Stack

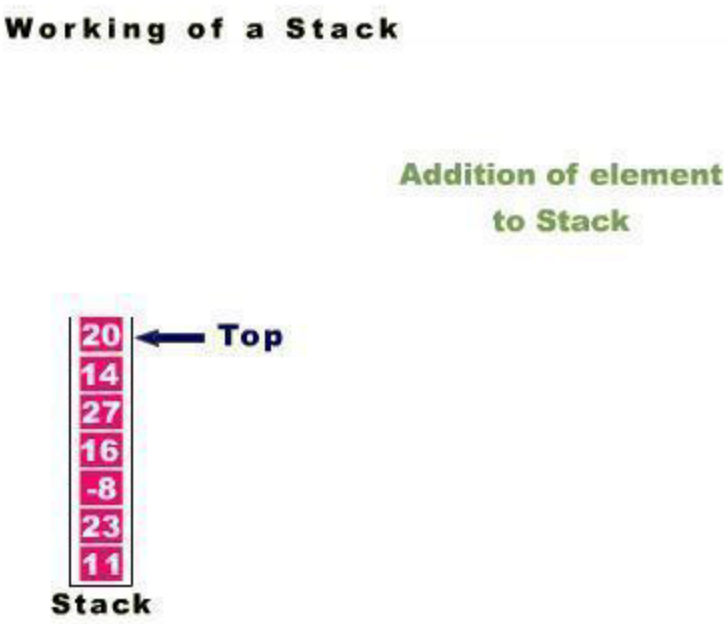
Step – 6


Working of a Stack
14
Addition of element
to Stack
27
16
-8
23
11
Top
Stack
push 14
on Stack

Step – 7

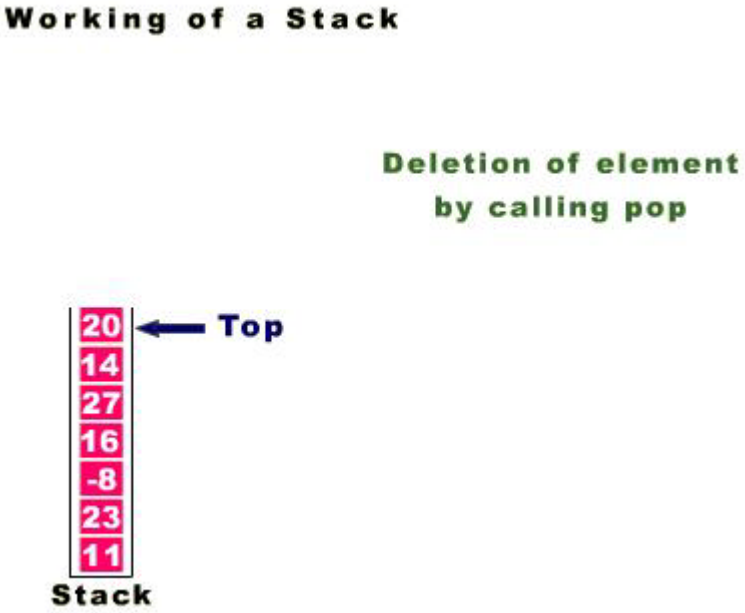

Working of a Stack
20
Addition of element
to Stack
14
27
16
-8
23
11
Top
Stack
Push 20
on Stack

Step – 8

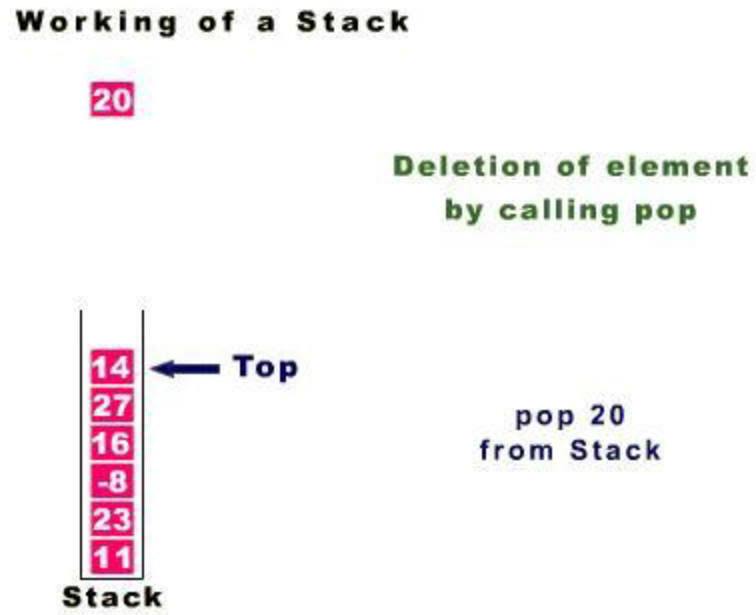


Step – 9

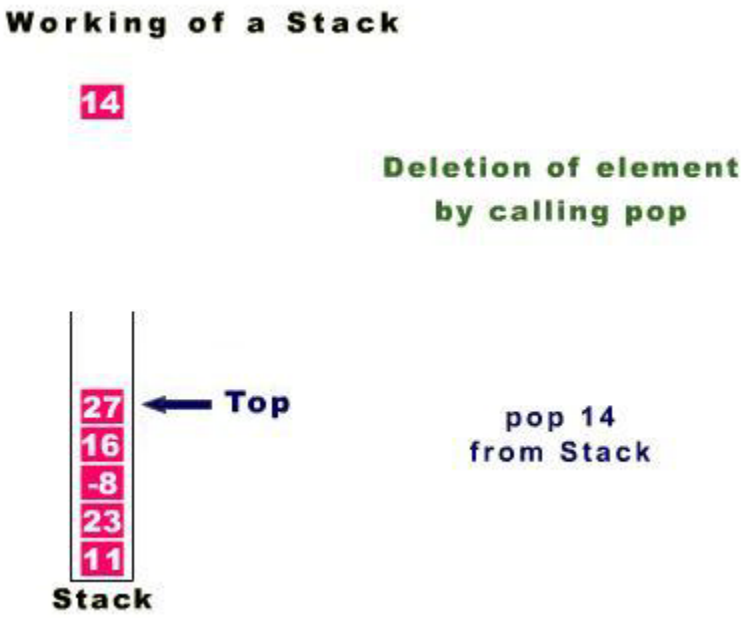
POP OPERATION



Step – 10



Step – 11



Step – 12

Arithmetic Expressions

Arithmetic expressions are expressed as combinations of:

1. Operands
2. Operators (arithmetic, Boolean, relational operators)

Various rules have been formulated to specify the order of evaluation of combination of operators in any expression.

The arithmetic expressions are expressed in 3 different notations:

1. *Infix:*

- In this if the operator is binary; the operator is between the 2 operands.
- And if the operator is unary, it precedes the operand.

2. *Prefix:*

- In this notation for the case of binary operators, the operator precedes both the operands.
- Simple algorithm using stack can be used to evaluate the final answer.

3. *Postfix:*

- In this notation for the case of binary operators, the operator is after both the corresponding operands.
- Simple algorithm using stack can be used to evaluate the final answer.

Always remember that the order of appearance of operands does not change in any Notation. What changes is the position of operators working on those operands.

RULES FOR EVALUATION OF ANY EXPRESSION:

An expression can be interpreted in many different ways if parentheses are not mentioned in the expression.

- For example the below given expression can be interpreted in many different ways:
- Hence we specify some basic rules for evaluation of any expression :

A priority table is specified for the various type of operators being used:

PRIORITY LEVEL	OPERATORS
6	** ; unary - ; unary +
5	* ; /
4	+ ; -
3	< ; > ; <= ; >= ; !> ; !< ; !=
2	Logical and operation
1	Logical or operation

Algorithm for evaluation of an expression E which is in prefix notation:

- We assume that the given prefix notation starts with IsEmpty() .
- If number of symbols = n in any infix expression then number of operations performed = some constant times n.
- Here *next token* function gives us the next occurring element in the expression in a left to right scan.
- The *PUSH* function adds element x to stack Q which is of maximum length n

```
Evaluate (E)
    {
        Top = 0;
        While (1)
        {
            x= next token (E)
            If (x = = infinity)
            {
                Print value of stack [top]
                as the output of the expression
            }
        }

    }
```

```
Else
{
    If (x = = operand)
        PUSH (Q, top, n, x);
    If (x = = operator)
    {
        Pop correct number
        of operands according to the
        the operator (unary/binary)
        and then perform the opera-
        tion and store result onto
        the stack
    }
}
```

Algorithm for evaluation of an expression E, which is in postfix notation:

- We assume that the postfix notation specifies end of expression by appending NULL at the end of expression.
- here *next token* function gives us the next occurring element in the expression in a left to right scan.

- The *PUSH* function adds element x to stack Q which is of maximum length n

Evaluate (E)

```
{
        Top = 0;
        While (1)
        {
                x= next token (E)
                If (x = = infinity)
                {
                        Return (stack [top]);
                }
        }
}
```

```
Else
{
        If (x = = operand)
                PUSH (Q, top, n, x);
        If (x = = operator)
        {
                Pop correct number of
        operands according to the ope-
        rands according to the operat-
        or (unary/binary) and then
        perform the operation and
        store result onto the stack
        }
```