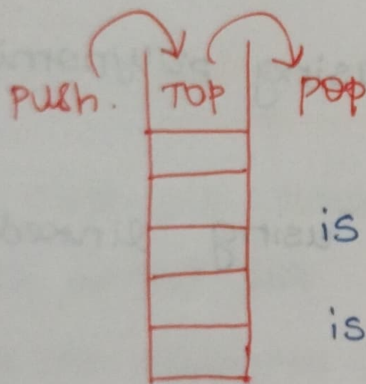
UNIT-II

What is stack?

Stack is the linear data structure in which the insertion and deletion operations are performed at only one end. Here the insertion and deletion are performed at a single position called "TOP". This means, elements are inserted at the top and removed from top only.

The operation is based on "LIFO" - LAST IN FIRST OUT.



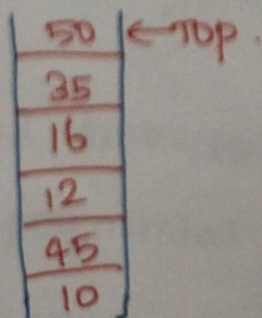
In a stack, the insertion operation is called "PUSH" and deletion operation is called "pop".

Definition:-

Stack is a linear data structure in which the operations are performed based on LIFO principle.

A collection of similar data items in which both the insertion and deletion are performed based on "LIFO".

Example:- Insert 10, 45, 12, 16, 35 and 50.



Operations on a stack:-

The following are the operations are performed on stack

1. push
2. pop
3. Display.

stack data structures can be implied in two ways.

->using arrays

->using linked lists.

when stack is implemented using arrays, stack can organize an only limited number of elements. when, stack is implemented using linked list. stack can organize using unlimited number of elements.

STACK USING ARRAYS:

stack can be implemented using 1D array. But we can store only fixed number of data values.

Just create 1D array and insert the elements using LIFO principle. using "top" variable.

First step:

->include all header files

->declare all necessary functions

-> create 1 Dimensional array int stack[SIZE]

->define TOP variable and Initialize TOP = -1.

-> Create main method and define the following operations

- Push operations
- pop operation and
- Display operation.

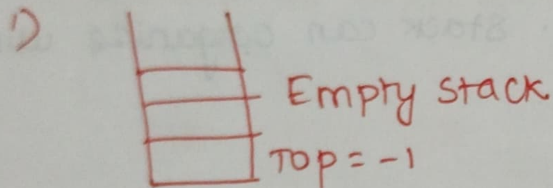
PUSH operation:-

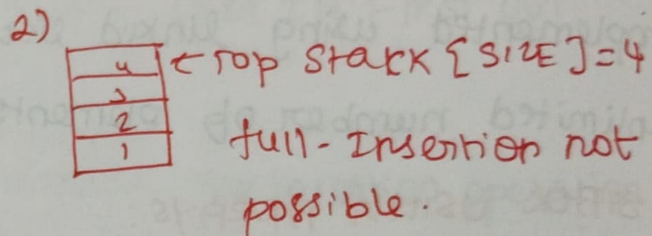
push() operation / function is used to insert an element into the stack at TOP position.

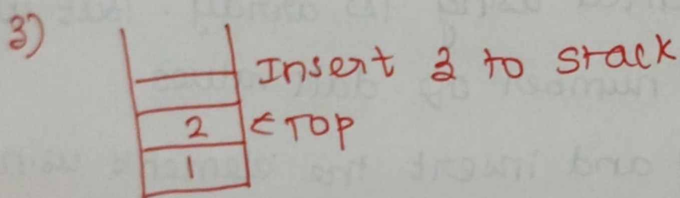
Step 1:- check whether stack is FULL. (top == SIZE-1)

Step 2: If it is full, then display "STACK IS FULL").
Insertion is not possible and function termination.

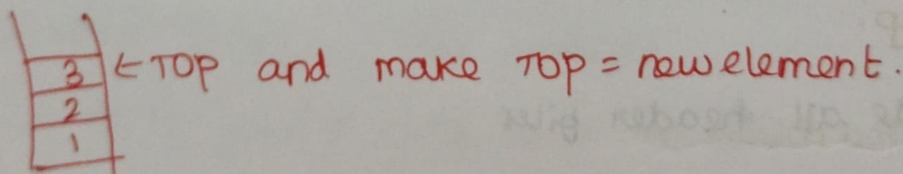
Step 3:- If stack is not full, increment the (top++) and set stack[top] to (stack[top] = value).







Now increment top by top++ by one position.



Pop operation:- Deleting a value from the stack.

pop() function is used.

Step 1:- check whether stack is empty. (top == -1).

Step 2: If it is empty, display stack is empty, hence deletion is not possible.

Step 3:- If not empty, delete stack[TOP], decrement [top--].

Display () :

Step 1 :- Check whether stack is empty (top == -1)

Step 2 :- If it is empty, then display stack is empty

Step 3 : If it is not empty, define variable 'I'. Display stack [I] and decrement I value only one by one. (I--).

Step 4 :- Repeat process, until I becomes zero ('0').

STACK USING LINKED LIST :-

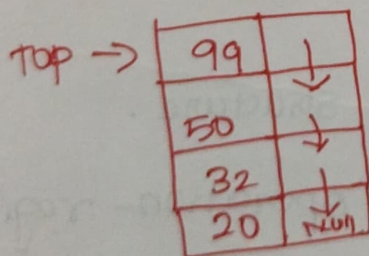
Major issue in array implementation of list is limited number of array. Here the amount of data is specified. Hence stack using array is not possible, when size does not known. So, it can be implemented using linked list.

List stack works four unlimited number of data.

-> here 1st node is in bottom

-> top denote the top element.

-> Next field of First element is "Null".



Need for stack memory:

-> stack memory is temporary data structure.

-> Essential element of stack memory operation- register which is called stack pointer.

-> This is required to evaluate expressions which has operands and operations.

-> Used for syltematic memory management.

Expressions

An expression is a collection of operators and operands that represents a specific value.

operands are the values on which operators can perform the task. operator is a symbol for performing tasks.

Types:

Infix expression

postfix expression

prefix expression.

Infix Expression:

here operator is used between two operands.

op1 op op2

Ex:- a + b
↓ ↓ ↓
op1 op op2.

postfix expression:

Here operator is used after operands. That is the operator follows the operands.

op1 op2 op ⇒ ab+

Prefix expression:

Here operator is used before the operands.

op op1 op2 Ex:- +ab.

we can able convert the one form of expression to another expression.

postfix evaluation

Algo:

1. Read all the symbols one by one from left to right in given exp
2. If reading symbol is operand, then push it on to stack
3. If reading symbol is operator (+, -, *, / etc...) then Perform two pop operations and store the two poped operands. Then again perform reading symbol operation using operand 1 and operand 2, push it on the stack.
4. Finaly perform pop operation, display popped value as final result.

Ex: Infix epression (5+3) * (8-2)

↓ postfix expression 53 +82 - *.

Infix to postfix conversion:

Algo:

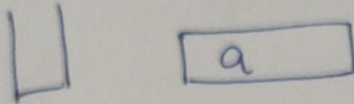
1. Read one character at a time
2. If it is operand then it can be placed onto the output array
3. If it is an operator, then push on to the stack.
4. Left paranmeses also pushed onto the stack. (.
5. If the input is close symbol, then pop the stack until the left paranthesis and write onto the output array.

6. If we reach end of the input, pop the stack until its empty and write the symbols on to the output array. High priority operator can be placed above low priority.

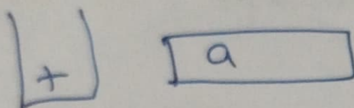
Example:- parenthesis are not written in the output.

a + b * c + (d * e + f) * g.

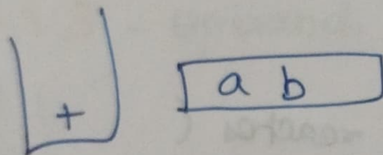
Read a

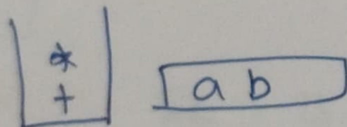


Read +

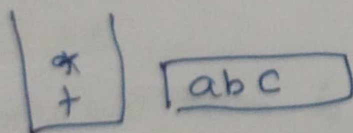


Read b

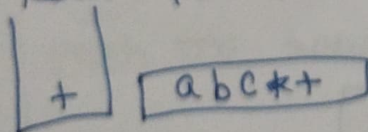




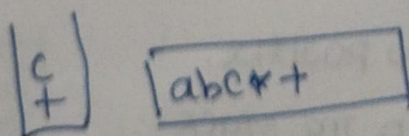
Read c



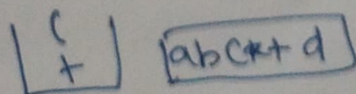
Read +, so pop *, also pop + already in array insert new +.



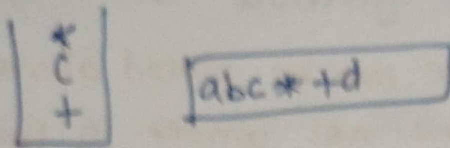
Read (



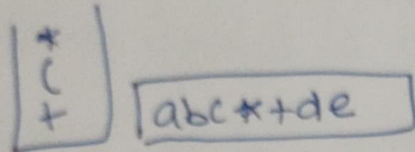
Read d



Read *

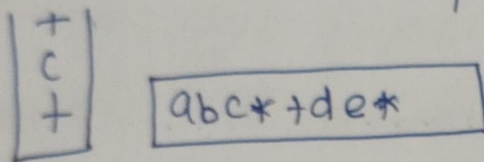


Read e

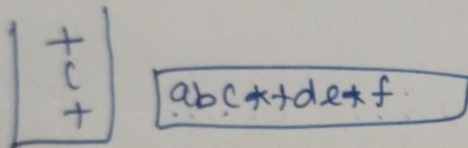


Read +

+ has low priority, pop *

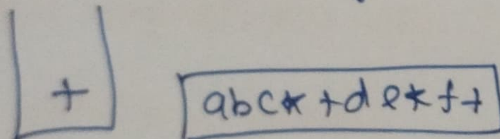


Read f

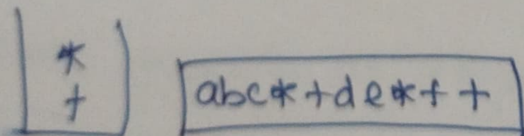


Read)

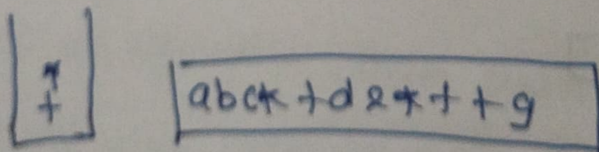
take all symbols until it reaches (



Read *



Read g



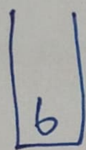
abc*+de*f+g*+ → postfix.

pop all remaining symbols in the stack.

POSTFIX EVALUATION

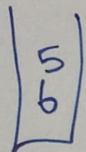
The given input is 6523 + 8 * + 3 + *

Read 6

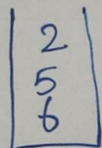


If it is operand, push it into the stack

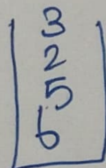
Read 5 - operand. so push into stack



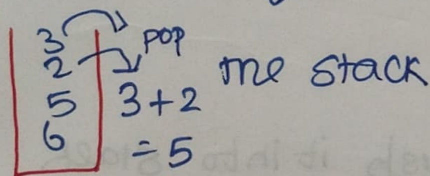
Read 2 , operand. so push into stack



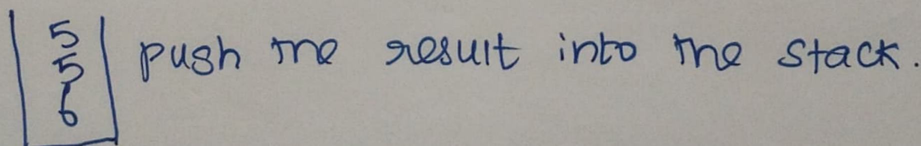
Read 3 - operand. so push into stack



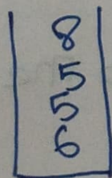
Read + If it is operator, pop two operand from the stack



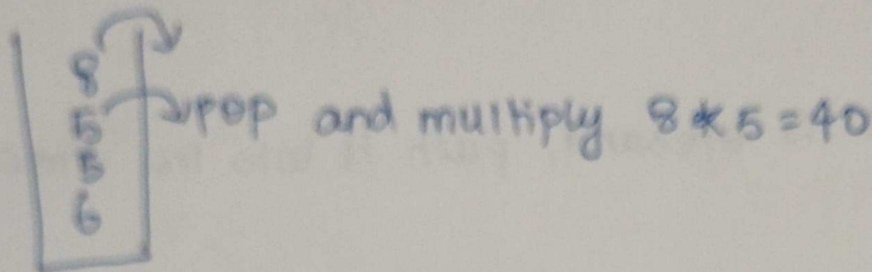
already we have 6 and 5 in the



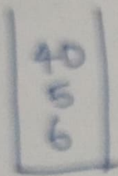
Read 8 - If it is operand so push 8 into the stack.



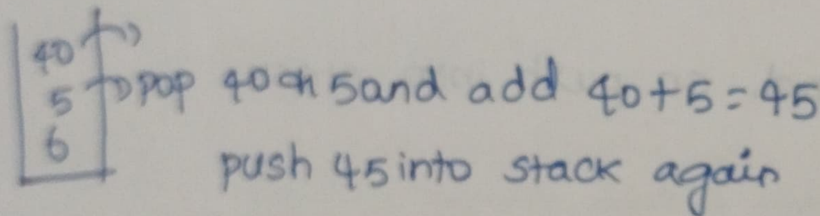
Read * - It is an operator, so pop out two elements from stack.



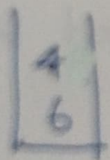
Hence stack becomes



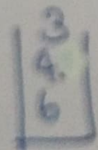
Read + . It is an operator, so pop out two element from stack.



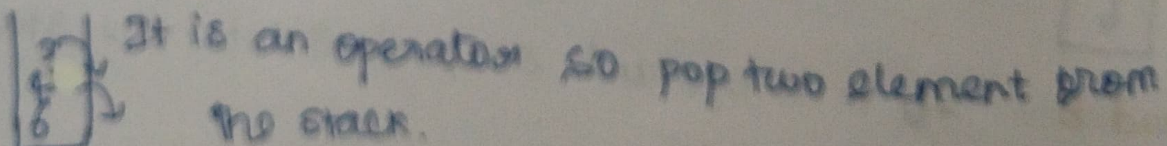
Hence stack becomes



Read 3 - It is an operand, so push it into stack

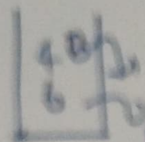


Read +



'+' so add 3+43 = 46 and push it into the stack.

Read * -



It is an operator, so pop two elements from stack.

pop '*'. multiply 48 * 6 = 288

hence:- 6523+8*+3+* = 288