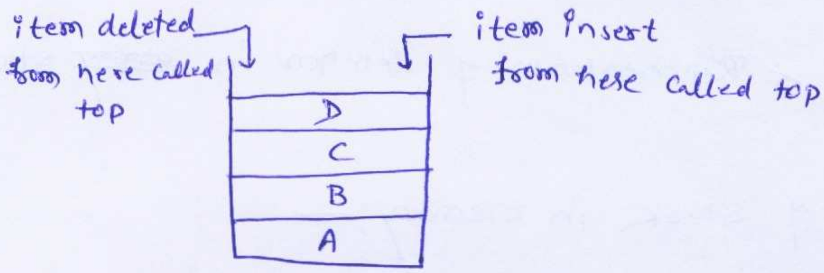
UNIT-3

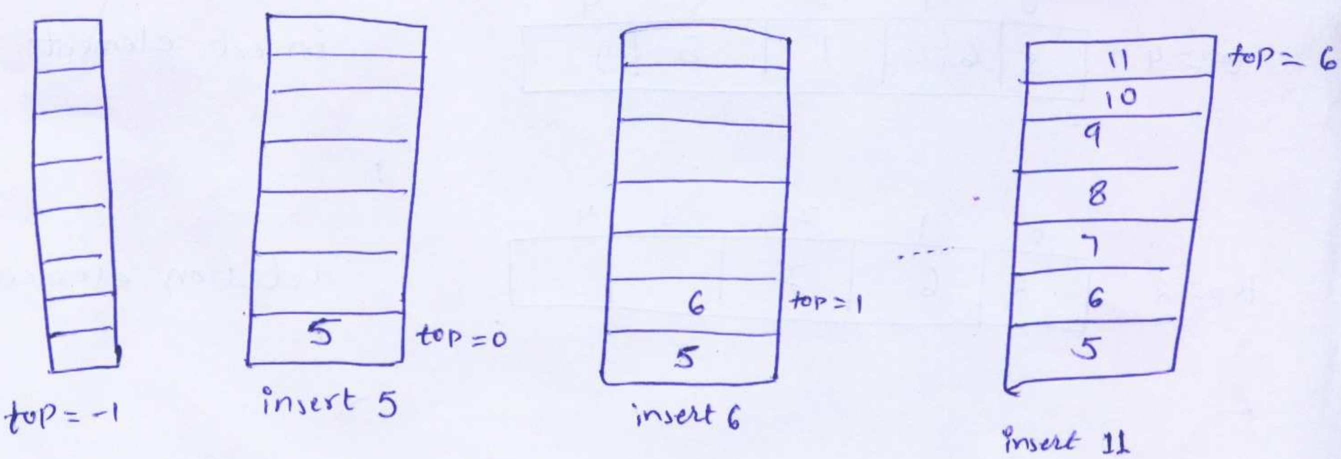
Stack & Queue

Introduction :— A Stack is a linear data structure in which insert of new element or deletion of existing element always takes place at the same end.

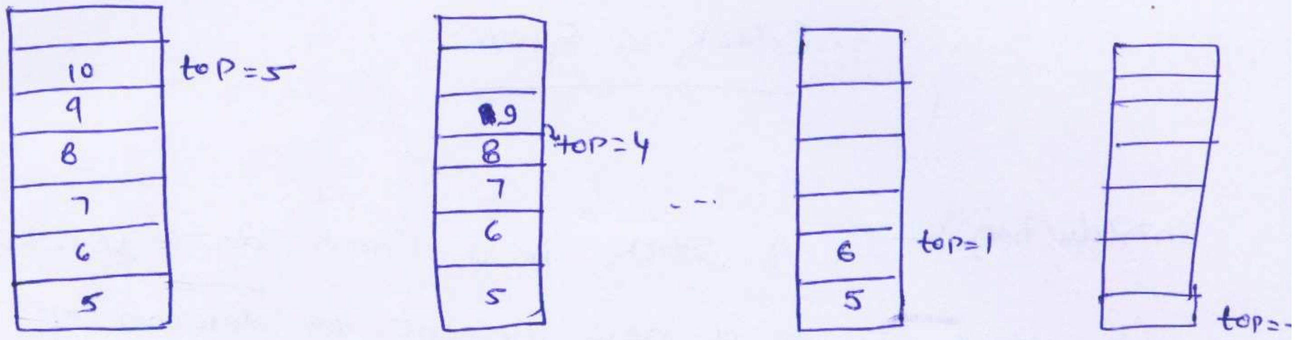


This place is called <u>Top</u> of the stack.

Stack are also called Last In First Out (LIFO) order.



—: Representation of insertion in stack :—

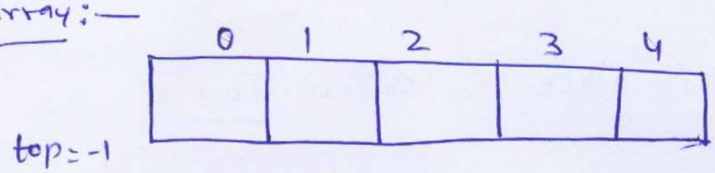


delete 11 delete 10 delete 7 Empty stack

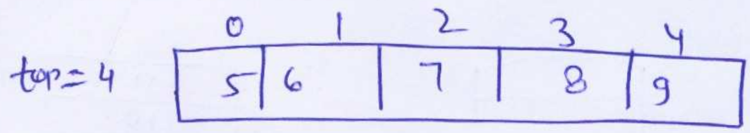
: Representation of deletion in ~~array~~ stack:—

Representation of stack in memory:—

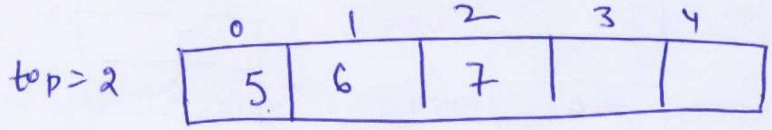
By using array:—



Empty stack

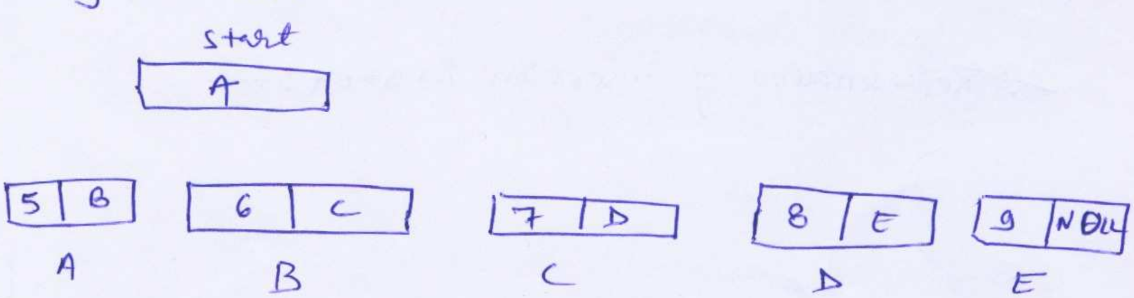


insert element



deletion elements.

By using link list:—



Operations on stack;—

Two operations can be performed on stack

i) PUSH :— If we want to insert an new element in the stack then this condition is called PUSH.

ii) POP — If we want to delete an ~~⊕~~ existing element stack then condition is called POP.

overflow :— If we want to insert an element and stack is full then this condition is called overflow.

underflow — If we want to delete an element and stack is empty then this condition is called underflow.

Algorithms for Insertion or PUSH :—

PUSH(STACK, TOP, MAXSIZE, VALVE)

step 1 :- If top = MAXSIZE-1 then
Write : overflow and Return.
{End of if statement}

~~step2 Set Top~~

step-2 set Top = Top + 1

step-3 set STACK[TOP] = VALUE;

step-4 Exit.

Algorithm for delete or pop :—

pop(TOP, STACK, VALUE)

Step-1 → If Top = -1 then
write "Underflow" and return
{End of If statement}

Step-2 Set VALUE = STACK[TOP] {Assign Top element to value}

Step-3 Set Top = Top -1

Step 4 Return.

Some other operations on stack
i) Peep operation
ii) Update operation

i) Peep operation :— If we want to extract the information stored at some location in a stack then peep operation is required. In this operation, we can move the pointer to the ~~desired~~ desired location and then information is extracted associated at the location.

Algo for peep operation —

step1 - If (~~stack~~ Top - I + 1 < 0 then { check stack empty or not }

write " underflow" and Return

{End of If }

step-2 set Value = stack[Top - I + 1]
{ Give element at I^{th} location from Top of the stack }

3 write : Value

4 End

ii) Update operation: —

When the information at any location in a stack is to be changed. If we want to update ~~to~~ information at the i^{th} location in the stack, we have to move the top pointer to the i^{th} location from the top of the stack and then change the value at that location.

Algorithms : —

step 1 If Top - I + 1 < 0 then
write overflow and exit
{End of if}

step-2 Set stack [TOP - I +1] = Value

step-3 End.

Example of stacks :—

i) plates in cafe, where every plates added at the top of stack. similarly every new plates takenoff the stack is also from the top of stack. This means that last plate added to a stack is the first plate to be removed.

2) Container of books.

3) Computer stock

4) stack of pannies (Coins)

5) stack of folded towels.

Stack Application :—

i) Conversion of expression
ii) Evalution of expression
iii) Recursion

① Conversion of expression;—

Expression
- → Infix expression (A + B)
- → Post fix ~~not~~ expression (A B +)
  (Reverse polish Notation)
- → Pre fix expression (+ AB)
  (Polish Notation)

Priority of operator. :—

Priority	operator
1	[ ], { }, < >, ( ) (Parentheses)
2	exponentiation ^
3	*, /, %
4	+, −
5	<, >, <=, >=
6	==, != (Comparision operators
7	&& (logical ~~OR~~ AND)
8	|| (logical OR)

Example :- Convert infix to postfix of following expression

① → (A+B) * (C-D) + E/F ②

step-1 (A+B) * (C - D) +(E/F)

step-2 = (AB+) * (CD-) + (EF\)

step3 = AB+ CD - * + EF\

4 = AB + CD - * EF\ +

2) (A+B) * C/D + e^f/g

= (AB+) * C/D + e^f/g

= (AB+) * C/D + ef^/g

= AB+ * C/D + ef^g/

= AB+ ~~CD/*~~ + ef^g/
C*/D

= ~~AB+ CD/ * ef^g/+~~

= AB+ C *D\ + ef^g/

= AB+ C *D/ ef^g/ +

3 $A+(B*C)-((D/E \wedge F)*G)/H$

$= A+(BC*)-((D/EF\wedge)*G)/H$

$= A+(BC*)-((DEF\wedge/)*G)/H$

$= A+BC* - DEF\wedge/G*/H$

$= A+BC* - DEF\wedge/G*H/$

$= ABC*+ - DEF\wedge/G*H/$

$= ABC*+DEF\wedge/G*H/-$

Evaluation of postfix:—

For evaluation, the postfix expression is scanned from left to right. When an operand is found it is pushed onto the stack and when an operator is found the last two operands are popped from the stack. Then required operation is applied to them and their result is pushed onto the stack. Here we have no need of information of operator precedence.

Ex: Convert the following expression in postfix notation-

$$5*(6+2)-12/4$$

Sol: 5 6 2 + * 12 4 / −