Stack

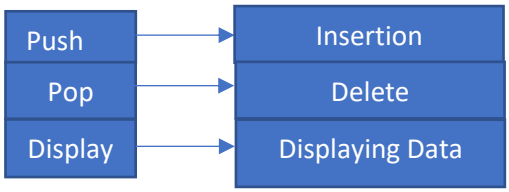
LIFO - Last In First Out

FILO - First In Last Out

0	1	2	3	4

Top = -1[shows the stack is empty]

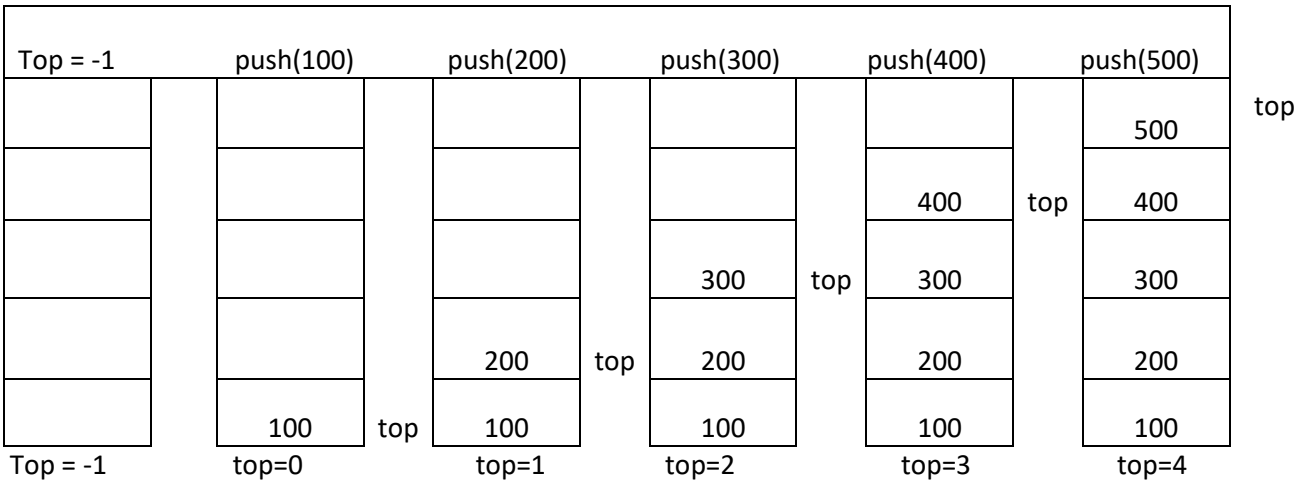
It has three operations push,pop,display (insertion,delete,displaying data)



Let's consider a stack of size 5 to see the following operations.

Push Operation:-

Algorithm	**Program of push (Function part):**
increment top st(top) = value st(top) = value	Void Push(val) { if top ==size-1 { Print stack is full/ overflow } else { top++; st[top] = val; } }

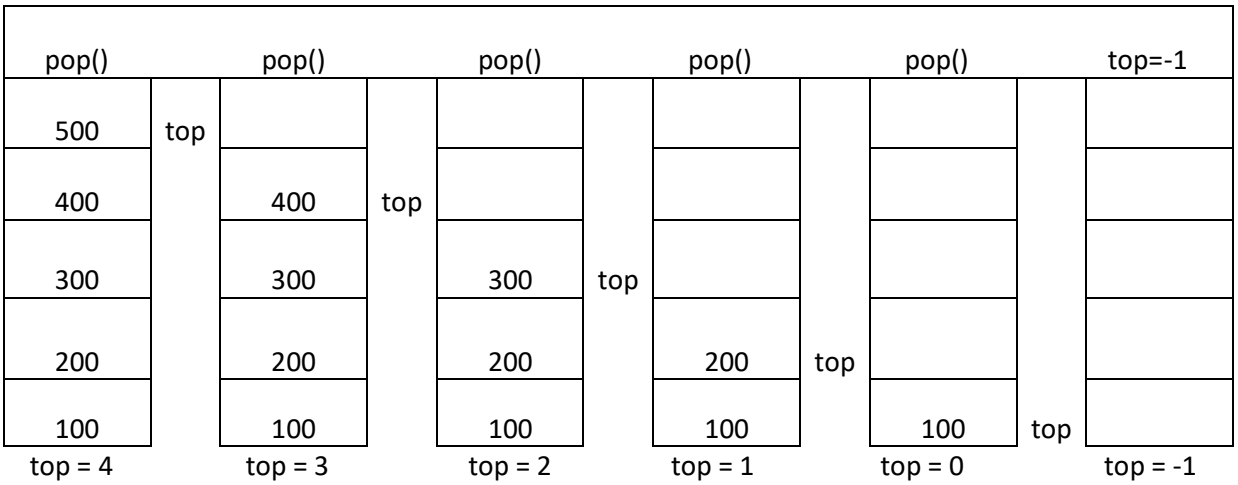


In each step, top is increased by 1 and finally it reached the (size -1)th position. Now if we try to insert another element it says stack is full.

Pop Operation:-

Algorithm	**Program for pop (Function part):-**

Decrement top
When top != -1
Then we can delete
	int pop()
	{
	if top == -1
	{ Print stack is empty/ underflow }
	else
	{ val = st[top]; st[top--] = 0; return val; }
	}

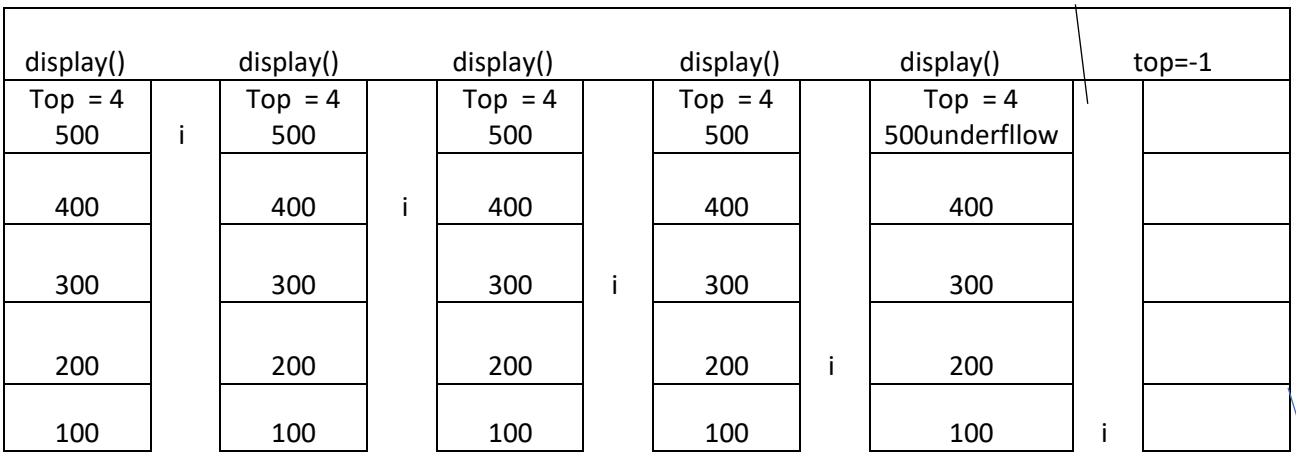


In each step, top is decreased by one and finally it has cleared all the elements in the stack and made the stack as empty. It is impossible to delete an element from the empty stack.

Display Operation :-

Algorithm	**Program of display (Function part):**

If top == -1
Print " Stack is empty/underflow"
Otherwise run a loop print the elements in the stack from the positon of top to 0
	void display()
	{ int i;
	if(top == -1)
	{ printf(" Stack is empty/underflow"\n) ; }
	else
	{ for(i = top; i>=0 ; i--)
	{ printf("%d",st[i]); }
	printf("\n"); }
	}



Output of display is

500 400 300 200 100

In this case,

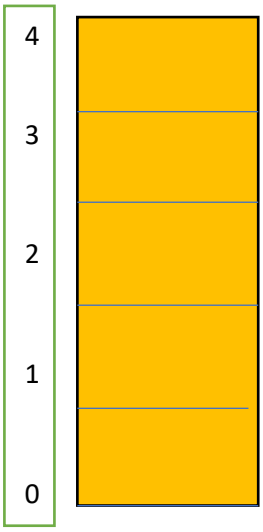
It prints stack is empty

Different Types of Notations:-

Infix a+b

Postfix ab+

Prefix +ab



^ → 3

*** , / → 2**

+,- → 1

INFIX to POSTFIX ALGORITHM:-

Infix = a+b*c-d/e+f

Postfix =

i++ loop

if i is operand:

append it to postfix

if i is operator:

if stack is empty:

push(i)

else if precedence(i) > precedence(st[top]):

push(i)

else

op = pop()

append it to postfix

continue loop until>

Example:-

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
	i										

Pre(a) = false j=a j++

4		
3		
2		
1		
0		

Top = -1

Postfix	0	1	2	3	4	5	6	7	8	9	10
	j										

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
		i									

Pre(*) =2 top++ top=*

4		
3		
2		
1		
0	*	top

Postfix	a	1	2	3	4	5	6	7	8	9	10
		j									

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
			i								

Pre(b) = false j=b j++

4		
3		
2		
1		
0	*	top

Postfix	a	b	2	3	4	5	6	7	8	9	10
			j								

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
				i							

Pre(-) = 1 1<2 top-- - and j = * ,j++

4		
3		
2		
1		
0	-	top

Postfix	a	b	*	3	4	5	6	7	8	9	10
				j							

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
					i						

Pre(c) = false j = c j++

4		
3		
2		
1		
0	-	top

Postfix	a	b	*	c	4	5	6	7	8	9	10
					j						

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
						i					

Pre(/) = 2,true 2>1 ++top top = /

4		
3		
2		
1	/	top
0	-	

Postfix	a	b	*	c	4	5	6	7	8	9	10
					j						

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
							i				

Pre(d) = false j=d j++

4		
3		
2		
1	/	top
0	-	

Postfix	a	b	*	c	d	5	6	7	8	9	10
						j					

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
								i			

Pre(^) = 3 3>2 ++top top = ^

4		
3		
2	^	top
1	/	
0	-	

Postfix	a	b	*	c	d	5	6	7	8	9	10
						j					

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
									i		

Pre(e) = false j = e j++

4		
3		
2	^	top
1	/	
0	-	

Postfix	a	b	*	c	d	e	6	7	8	9	10
							j				

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
										i	

Pre(+) = 1 1<3 & top != -1 → replace ^ by + & write ^ in postfix

4		
3		
2	+	top
1	/	
0	-	

Postfix	a	b	*	c	d	e	^	7	8	9	10
								j			

Pre(+) = 1 1<2 & top != -1 & top-- → replace / by + & write / in postfix

4		
3		
2		
1	+	top
0	-	

Postfix	a	b	*	c	d	e	^	/	8	9	10
									j		

Pre(+) = 1 1=1 & top != -1 & top - - → replace - by + & write - in postfix

4		
3		
2		
1		
0	+	top

Postfix	a	b	*	c	d	e	^	/	-	9	10
										j	

Infix	a	*	b	-	c	/	d	^	e	+	f
	0	1	2	3	4	5	6	7	8	9	10
											i

Pre(f) = false j=f j++

4		
3		
2		
1		
0	+	top

Postfix	a	b	*	c	d	e	^	/	-	f	10
											j

Top = + top != -1 → j=+

4		
3		
2		
1		
0	+	top

Postfix	a	b	*	c	d	e	^	/	-	f	+
											j

This is the required postfix expression converted from infix.

If we have open braces in our infix then we have to apply the following changes in our program:-

Precedence of '(' is -1

If i=='(' :

Pop all operator upto open bracket and append it to postfix

If i==')'

op = st[top--]

while(op!='(')

append post op

op = st[top --]

Example :- **a+(b*c+d)-e**

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
	i										

Pre(a) = false j=a j++

4		
3		
2		
1		
0		

Top = -1

Postfix	0	1	2	3	4	5	6	7	8	9	10
	j										

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
		i									

Pre(+) = 1 j=a j++

4		
3		
2		
1		
0	+	top

Postfix	a	1	2	3	4	5	6	7	8	9	10
		j									

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
			i								

Pre('(') = -1 infix[i] = '(' ++ top top = (

4		
3		
2		
1	(	top
0	+	

Postfix	a	1	2	3	4	5	6	7	8	9	10
		j									

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
				i							

Pre(b) = false j = b j++

4		
3		
2		
1	(	top
0	+	

Postfix	a	b	2	3	4	5	6	7	8	9	10
			j								

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
					i						

Pre(*) = 2 2 > -1 ++ top top = *

4		
3		
2	*	top
1	(	
0	+	

Postfix	a	b	2	3	4	5	6	7	8	9	10
			j								

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
						i					

Pre(c) = false j = c j++

4		
3		
2	*	top
1	(	
0	+	

Postfix	a	b	c	3	4	5	6	7	8	9	10
				j							

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
							i				

Pre(+) = 1 1<2 & top != -1 & top - - → replace * by + & write * in postfix then j++

4		
3		
2	+	top
1	(	
0	+	

Postfix	a	b	c	*	4	5	6	7	8	9	10
					j						

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
								i			

Pre(d) = false j = d j++

4		
3		
2	+	top
1	(	
0	+	

Postfix	a	b	c	*	d	5	6	7	8	9	10
						j					

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
									i		

Pre(‘)‘) = -1 infix[i] = ‘)‘ j = + j++ upto st[top] == ‘(‘ remove ‘(‘ from stack decrese top by 1

4		
3		
2		
1		
0	+	top

Postfix	a	b	c	*	d	+	6	7	8	9	10
							j				

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
										i	

Pre(-) = 1 1=1 replace + by – j = + j++

4		
3		
2		
1		
0	-	top

Postfix	a	b	c	*	d	+	+	7	8	9	10
								j			

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
											i

Pre(e) = false j = e j++

4		
3		
2		
1		
0	-	top

Postfix	a	b	c	*	d	+	+	e	8	9	10
									j		

Infix	a	+	(	b	*	c	+	d	)	-	e
	0	1	2	3	4	5	6	7	8	9	10
											i

Pre(e) = false j = e j++ and I reached it's end

4		
3		
2		
1		
0	-	top

Postfix	a	b	c	*	d	+	+	e	8	9	10
									j		

Top = - top != -1 → j = - and I reached it's end

4		
3		
2		
1		
0	-	top

Postfix	a	b	c	*	d	+	+	e	-	9	10
									j		

Postfix	a	b	c	*	d	+	+	e	-
									j

This is the required postfix equation of given infix.

Infix to Prefix notation:-

1. Follow the same rules of infix to postfix notation
2. Apply the algorithm of infix to postfix
3. Get the reverse of the string of the infix to postfix which will be the required infix to prefix notation

Postfix Evalution:-

If i is operand :

Push into stack

If i is operator:

Pop two values from stack

Op1

Op2

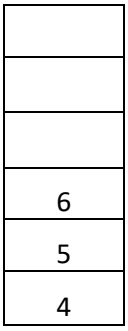
Then apply the operator as op2 op(+,-,*,/,%) op1 and store in res

Then push(res) into stack

Finally stack[0] is the answer

4	5	2	3	*	-	*	2	3	*	+

2*3 = 6

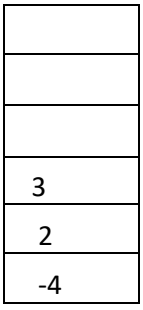


5-6=-1

(-1)*4 = (-4)

4	5	2	3	*	-	*	2	3	*	+

Completed part



3*2 = 6

6+(-4) = 2