Stack

Stack is a linear list where any element is added at the top of the list and any element is deleted (accessed) from the top of the list. So, for stack an indicator or pointer must be used to indicate or point the top element of the stack.

Add operation for a stack is called “push” operation and deletion operation is called “pop” operation. Stack is a LIFO (Last In First Out) structure. That means the element which was added last will be deleted or accessed first.

The elements of a stack are added from bottom to top, means push operation is performed from bottom to top. The elements are deleted from top to bottom, which means pop operation is performed from top to bottom.

Stack can be implemented using two ways; using array and using linked list.

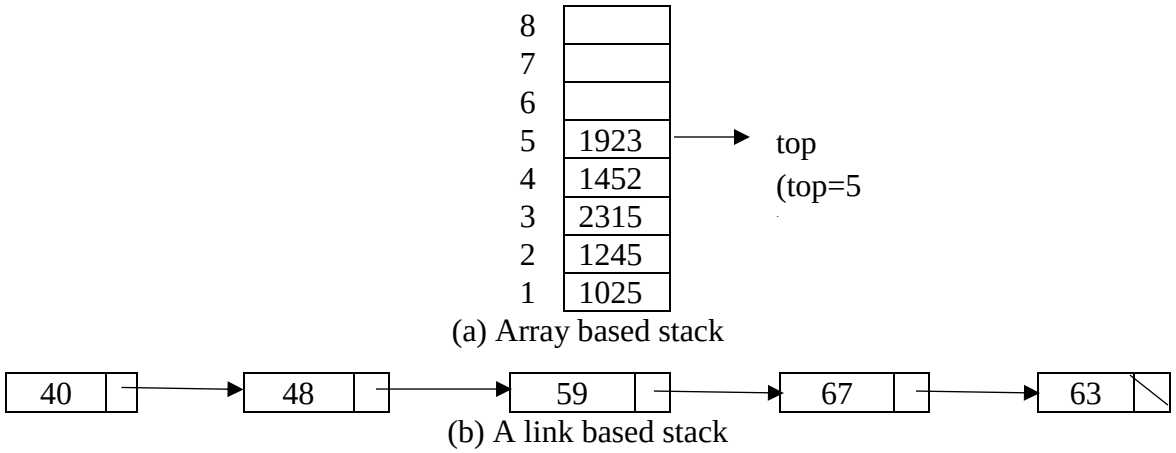


Fig. 1: Graphical representation of stack

Array Based Stack

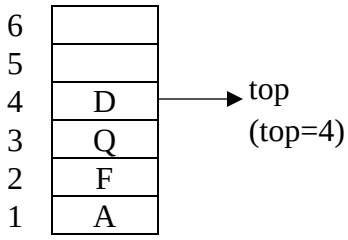
The stack, which is implemented using array is called array based stack. To create an array based stack, at first we have to declare an array with required size.

Push Operation

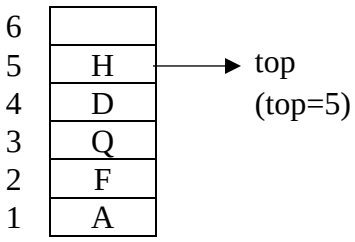
Push operation means to add an element to a stack.

Here, we shall use array based stack. So, an array will be treated as a stack. We need an indicator or index identifier to add element to the stack and this indicator will mark the top of the stack. To add an element we have to check whether the array is already full or not. If the array is already full then we cannot add any element, otherwise we can.

Here, top is an indicator indicates the top element of the stack and item is an element to be added to the stack, M is the size of the stack (array). Overflow occurs when we try to insert an element into the stack, which is already full.



(a) Before push



(b) After push

Fig. 2: Pictorial view of push operation

Algorithm to add an element to a stack

1. Declare the stack and top:

stack[1…..M], top;

2, Add an item in the stack:

if(top<M)

{

top=top+1;

stack[top]=item;

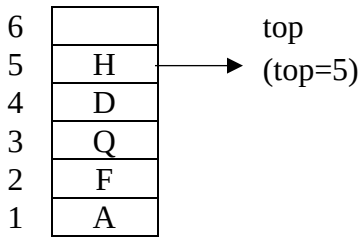
}

else print “Over Flow”;

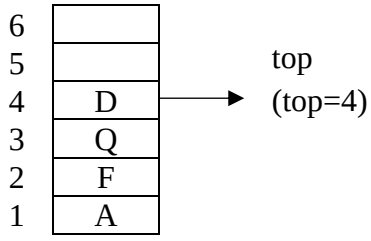
3. Output will be the updated stack.

Pop Operation

Pop operation means to delete (access) an element from a stack. Here, top is an indicator indicates the top element of the stack, M is the size of the array and x is a variable where we access top element of the stack.



(a) Before pop



(b) After pop

Fig. 3: Pictorial view of pop operation

Algorithm to delete an element from a stack

1. Declare the stack and top:

```
stack[1…..M], top;
```

2. Access the top element:

```
if (top==0) print “stack is empty”;
else {
        x=stack[top];
        top=top-1;
}
```

3. Output: updated list.

Applications of stack

Checking the validity of an arithmetic expression

Checking an expression is nothing but checking

1. Whether there are equal number of right and left parenthesis.
2. Whether right parenthesis is preceded by a matching left parenthesis.

If both the above conditions are satisfied, expression is valid.

Ex:

- ((A+B) or A+B(are not valid expressions because they violate condition 1.
-)a+b(-c violate condition 2.
- (a+b)) violate both the conditions.
- (a+b) * (c+d) is a valid expression.

Stacks can be used to check this.

Exp: [(A + B) - { C + D }] - [F + G]

Symbol Scanned	STACK
[	[
(	[, (
A	[, (
+	[, (
B	[, (
)	[
-	[
{	[, {
C	[, {
+	[, {
D	[, {
}	[
]	
-	
[	[
F	[
+	[
G	[
]	

In the above example we see the stack is empty at the end, so the expression is valid.

Converting an infix arithmetic expression to its postfix form

Expressions are usually represented in what is known as *Infix notation*, in which each operator is written between two operands (i.e., A + B).

Polish notation (prefix notation) - It refers to the notation in which the operator is placed before its two operands. Here no parentheses are required, i.e., +AB.

Reverse Polish notation (postfix notation) - It refers to the analogous notation in which the operator is placed after its two operands. Again, no parentheses is required in Reverse Polish notation, i.e., AB+.

There are 3 levels of precedence for 5 binary operators as given below:

Highest: Exponentiation (^ or ↑)

Next highest: Multiplication (* or x) and division (/ or ÷)

Lowest: Addition (+) and Subtraction (-)

Algorithm for Infix to Postfix

Step 1: Consider the next element in the input.

Step 2: If it is operand, display it.

Step 3: If it is opening parenthesis, insert it on stack.

Step 4: If it is an operator, then

- If stack is empty, insert operator on stack.
- If the top of stack is opening parenthesis, insert the operator on stack
- If it has higher priority than the top of stack, insert the operator on stack.
- Else, delete the operator from the stack and display it, repeat Step 4.

Step 5: If it is a closing parenthesis, delete the operator from stack and display them until an opening parenthesis is encountered. Delete and discard the opening parenthesis.

Step 6: If there is more input, go to Step 1.

Step 7: If there is no more input, delete the remaining operators to output.

Exp: 3 * 3 / (4 – 1) + 6 * 2

Expression	**STACK**	**Postfix expression**
3		3
*	*	3
3	*	3, 3
/	/	3, 3, *
(	/ (	3, 3, *
4	/ (	3, 3, *, 4
-	/ (-	3, 3, *, 4
1	/ (-	3, 3, *, 4, 1
)	/	3, 3, *, 4, 1, -
+	+	3, 3, *, 4, 1, -, /
6	+	3, 3, *, 4, 1, -, /, 6
*	+ *	3, 3, *, 4, 1, -, /, 6, 2
2	+ *	3, 3, *, 4, 1, -, /, 6, 2
		3, 3, *, 4, 1, -, /, 6, 2, *, +

Exp: (A * B - (C - D)) / (E + F)

Input symbol	**STACK**	**Postfix Expression**
(	(	
A	(	A
*	(*	A
B	(*	A B
-	(-	A B *
(	(- (	A B *
C	(- (	A B * C
-	(- (-	A B * C
D	(- (-	A B * C D
)	(-	A B * C D -
)		A B * C D - -
/	/	A B * C D - -
(	/ (	A B * C D - -
E	/ (	A B * C D - - E
+	/ (+	A B * C D - - E
F	/ (+	A B * C D - - E F
)	/	A B * C D - - E F +
		A B * C D - - E F + /

Evaluating a postfix expression

To evaluate the postfix expression we scan the expression from left to right. The steps involved in evaluating a postfix expression are:

Step 1: If an operand is encountered, push it on STACK.
Step 2: If an operator "*op*" is encountered
i. Pop two elements of STACK, where A is the top element and B is the next top element.
ii. Evaluate B *op* A.
iii. Push the result on STACK.
Step3: The evaluated value is equal to the value at the top of STACK.

Exp: 5 6 2 + * 12 4 / -

Input symbol	**STACK**	**Operation (B *op* A)**
5	5	
6	5, 6	
2	5, 6, 2	
+	5, 8	[6+2] (A=2, B=6)
*	40	[5*8] (A=8, B=5)
12	40, 12	
4	40, 12, 4	
/	40, 3	[12/4] (A=3, B=12)
-	37	[40-3] (A=3, B=40)

Exp: 4 3 2 5 * - +

Input symbol	**STACK**	**Operation (B *op* A)**
4	4	
3	4, 3	
2	4, 3, 2	
5	4, 3, 2, 5	
*	4, 3, 10	[2*5] (A=5, B=2)
-	4, -7	[3-10] (A=10, B=3)
+	-3	[4+(-7)] (A=-7, B=4)

What will be the correct output from the following sequence of operations?

```
push(5)
push(8)
pop
push(2)
push(5)
pop
pop
pop
push(1)
pop
```

Answer: 8 5 2 5 1

Explanation

Input	**STACK**	**Output**
push (5)	5	
push (8)	5, 8	
pop	5	8
push (2)	5, 2	
push (5)	5, 2, 5	
pop	5, 2	8, 5
pop	5	8, 5, 2
pop		8, 5, 2, 5
push (1)	1	
pop		8, 5, 2, 5, 1