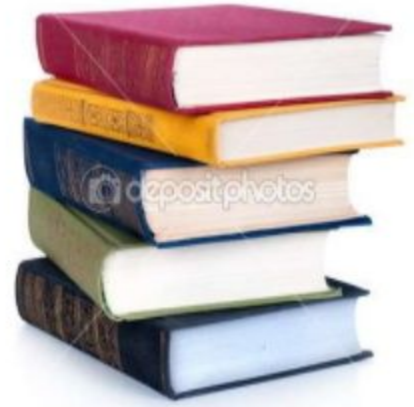
The Stack

Chapter 2

Stack

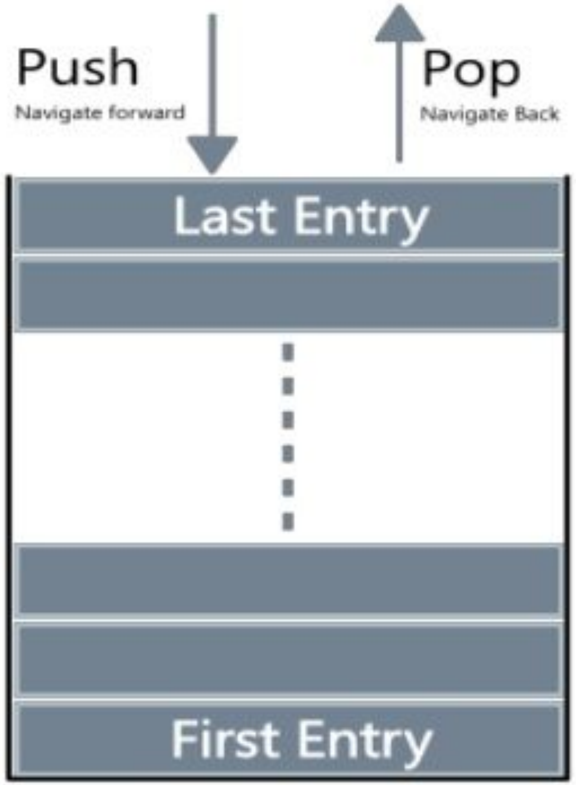
- A stack is a linear data structure that follows the Last in, First out principle (i.e. the last added elements are removed first).
- Stack is an ordered collection of items in to which new items may be inserted and from which items may be deleted at one end called top of the stack

EXAMPLES OF STACK:



Operations that can be performed on STACK:

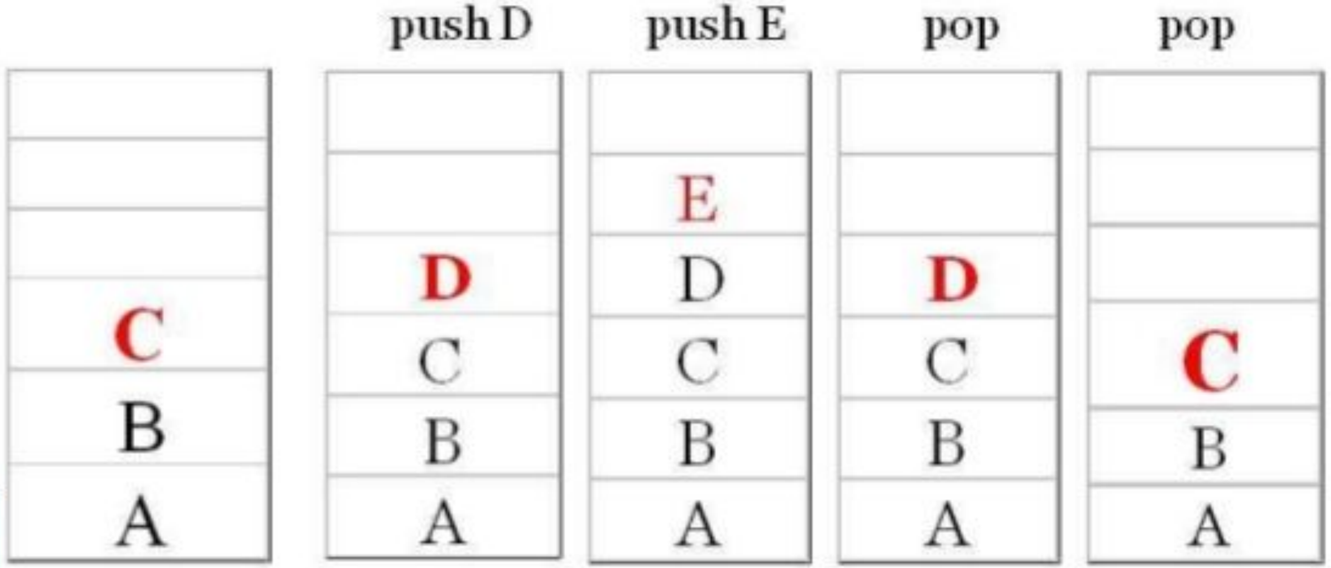
- PUSH.
- POP.



PUSH : It is used to insert items into the stack.

POP: It is used to delete items from stack.

TOP: It represents the current location of data in stack.

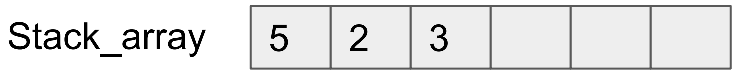


Implementation of Stack

- Array Implementation
- Linked List

Array Implementation

- In array we can push elements one by one from 0th position 1th position ……… (n-1)th position. Any element can be added or deleted at any place.
- We can push or pop elements from the top of the stack only.
    - When there is no place for adding element in the array, then this is called stack overflow. So first we check the value of top with the size of array.
    - When there is no element in the stack, then value of top will be -1. So we check the value of top before deleting the element from the stack.



Here stack is implemented using array and top of the stack value is 2

PUSH()

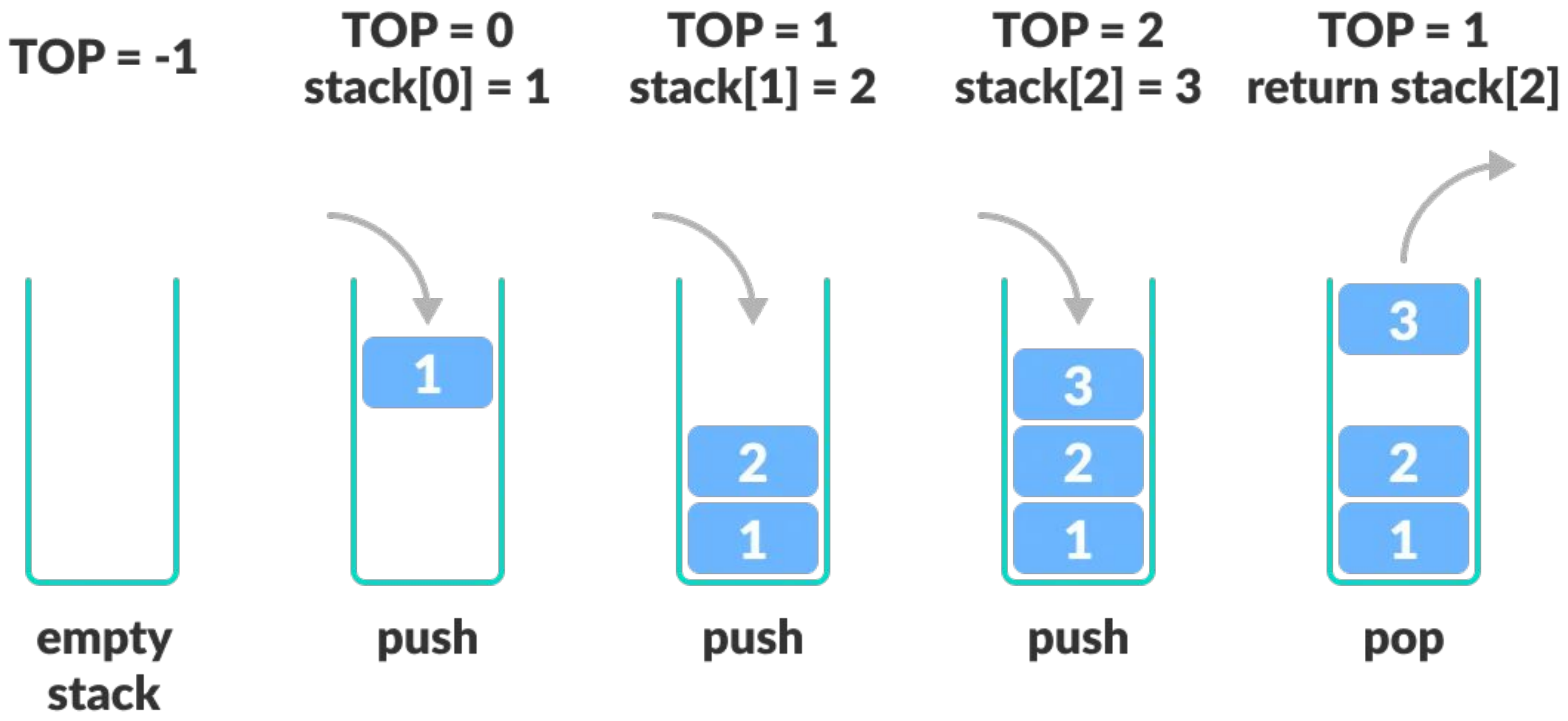
- If
    - (top=Max_size-1) then print Stack is Full
- Else
    - Enter a number
    - Store it in array
    - Top = top+1

POP()

- If
    - Top = -1 , then print stack is empty
- Else
    - Select the top element from the array and print it
    - Top =top -1

display()

- If
    - top=-1, then print Stack is Empty
- Else
    - Display all the elements in the array


TOP = -1
TOP = 0
stack[0] = 1
TOP = 1
stack[1] = 2
TOP = 2
stack[2] = 3
TOP = 1
return stack[2]
1
2
1
3
2
1
3
2
1
empty
stack
push
push
push
pop

Stack as an ADT

```
class stack{
    int arr[SIZE];
    int top;
    public:
    stack(){
        top=-1;
    }
    void push (int data);
    void pop();
    void display();
    };
```

Algorithm for inserting elements into stack

Push()

```
1.  If top=size-1
        Then write “Stack is full”
        Else
1.  Read item or data
2.  top= top+1
3.  stack[top]=item
4.  Stop
```

Algorithm for deleting element from stack

```
Pop()
1.  If top=-1
        Then write Stack is empty
        Else
1.  Item = stack[top]
2.  Top =top - 1
```

Application of Stack

Reversing Strings

Conversion of an expression from infix to postfix.

Evaluation of arithmetic expression from postfix expression

Precedence and Associativity

(,)

^

/, *

+, -

Algorithm for conversion of infix to postfix expression

- **Step 1** : Scan the Infix Expression from left to right.
- **Step 2** : If the scanned character is an operand, append it with final Infix to Postfix string.
- **Step 3** : Else,
    - **Step 3.1** : If the precedence order of the scanned(incoming) operator is greater than the precedence order of the operator in the stack (or the stack is empty or the stack contains a ‘(‘ or ‘[‘ or ‘{‘), push it on stack.
- **Step 3.2** : Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
- **Step 4** : If the scanned character is an ‘(‘ or ‘[‘ or ‘{‘, push it to the stack.
- **Step 5** : If the scanned character is an ‘)’or ‘]’ or ‘}’, pop the stack and and output it until a ‘(‘ or ‘[‘ or ‘{‘ respectively is encountered, and discard both the parenthesis.
- **Step 6** : Repeat steps 2-6 until infix expression is scanned.
- **Step 7** : Print the output
- **Step 8** : Pop and output from the stack until it is not empty.

Example: A+B-C+D

Input Expression	Stack	Postfix expression
A		A
+	+	A
B	+	AB
-	-	AB+
C	-	AB+C
+	+	AB+C-
D	+	AB+C-D+

K+L-M*N+(O^P)*w/u/v*T+Q

Input	Stack	Output
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	*-	KL+M
N	*-	KL+MN
+	+	KL+MN*-
(	(+	KL+MN*-
O	(+	KL+MN*-O

Input	Stack	Output
^	^(+	KL+MN*-O
P	^(+	KL+MN*-OP
)	+	KL+MN*-OP^
*	*+	KL+MN*-OP^
w	*+	KL+MN*-OP^W*
/	/+	KL+MN*-OP^W*
u	/+	KL+MN*-OP^W*U
/	/+	KL+MN*-OP^W*U/
v	/+	KL+MN*-OP^W*U/V
*	*+	KL+MN*-OP^W*U/V/
T	*+	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+

CONVERSION OF INFIX INTO POSTFIX

2+(4-1)*3 into 241-3*+

CURRENT SYMBOL	ACTION PERFORMED	STACK STATUS	POSTFIX EXPRESSION
(	PUSH C	C	2
2			2
+	PUSH +	(+	2
(	PUSH (	(+(	24
4			24
-	PUSH -	(+(-	241
1	POP		241-
)		(+	241-
*	PUSH *	(+*	241-
3			241-3
	POP *		241-3*
	POP +		241-3*+
)			

Exercises

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

Evaluation of postfix expression

In this case the stack contains the operands instead of operator

Whenever any operator occurs on scanning we evaluate with last two element of the stack.

Algorithm

1. Scan the symbol of array post fix one by one from left to right
2. If symbol is operand, two push into stack
3. If symbol is operator then pop last two element of the stack and evaluate as [top-1] operator [top] and push it to stack
4. Do the same process while scanning from left to right
5. Pop the element of the stack which will be value of evaluation of postfix arithmetic expression

Post fix expression ABCD ^+*EF^GH/*-

Evaluate postfix expression where A=4, B=5, C=4 D=2, E=2, F=2, G=9, H=3

Step	Symbol	Operand in stack
1	4	4
2	5	4,5
3	4	4,5,4
4	2	4,5,4,2
5	^	4,5,16
6	+	4,21
7	*	84

Step	Symbol	Operand in stack
8	2	84,2
9	2	84,2,2
10	^	84,4
11	9	84,4,9
12	3	84,4,9,3
13	/	84,4,3
14	*	84,12
15	-	72

Conversion of Infix to Prefix

Iterate the given expression from left to right, one character at a time

Step 1: First reverse the given expression

Step 2: If the scanned character is an operand, put it into prefix expression.

Step 3: If the scanned character is an operator and operator's stack is empty, push operator into operators' stack.

Step 4: If the operator's stack is not empty, there may be following possibilities.

If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator 's stack.

If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack untill we find a low precedence operator than the scanned character.

If the precedence of scanned operator is equal then check the associativity of the operator. If associativity left to right then simply put into stack. If associativity right to left then pop the operators from stack until we find a low precedence operator.

If the scanned character is closing round bracket (')'), push it into operator's stack.

If the scanned character is opening round bracket ('('), pop out operators from operator's stack until we find an closing bracket (')').

Repeat Step 2,3 and 4 till expression has character

Step 5: Now pop out all the remaining operators from the operator's stack and push into postfix expression.

Example: A+B*C+D into prefix

First of all reverse the infix expression: D+C*B+A

Input	Stack	Expression	Action
D		D	Add D into expression string
+	+	D	Push + into stack
C	+	DC	Add C into expression string
*	+*	DC	Precedence of * is higher so push * into stack
B	+*	DCB	Add B into expression string
+	++	DCB*	Precedence of + is lower. So pop * from stack
A	++	DCB*A	Add A into expression string
		DCB*A++	Pop all operators one by one as the end of expression is reached

Reverse the expression to get prefix expression: ++A*BCD

Example: Infix to prefix

Infix Expression: (A+B)*C

Reverse the expression: C*)B+A(

Input	Stack	Expression	Action
C		C	Add C into expression string
*	*	C	Push * into stack
)	*)	C	Push) into stack
B	*)	CB	Add B into expression string
+	*)+	CB	Push + into stack

Example: Infix to prefix

Input	Stack	Expression	Action
A	*)+	CBA	Add A into expression string
(	*	CBA+	(pair matched so pop + from stack
		CBA+*	Pop all operators one by one as we have reached end of the expression

Reverse the expression to get prefix expression *+ABC

Infix to Prefix conversion example

Infix Expression: (A+B)+C-(D-E)^F

First reverse the given infix expression: **After Reversing:** F^)E-D(-C+)B+A(

Input	Stack	Expression	Action
F		F	Add F into expression string
^	^	F	Push ^ into stack
)	^)	F	Push) into stack
E	^)	FE	Add E into expression string
-	^)-	FE	Push - into stack

Input	Stack	Expression	Action
D	^)-	FED	Add D into expression string
(	^)	FED-	‘(‘ Pair matched, so pop operator ‘-‘
-	-	FED-^	- Has less precedence than ^. So pop from stack
C	-	FED-^C	Add C into expression string
+	+	FED-^C-	Same precedence but associativity from right to left so pop from stack
)	+)	FED-^C-	Push) into stack
B	+)	FED-^C-B	Add B into expression string

Input	Stack	Expression	Action
+	+)+	FED-^C-B	Push + into stack
A	+)+	FED-^C-BA	Add A into expression string
(	+	FED-^C-BA+	‘(‘ Pair matched, so pop operator ‘+’
		FED-^C-BA++	Pop all operators one by one as we have reached end of the expression

Now **reverse** the expression to get prefix expression ++AB-C^-DEF

Convert (A+B)*(C+D) into prefix

Reverse infix expression:)D+C(*)B+A(

Input	Stack	Expression	Action
)	)		Push) into stack
D	)	D	Add D into expression string
+	)+	D	Push + into stack
C	)+	DC	Add C into expression string
(		DC+	(Pair matched so pop + from stack
*	*	DC+	Push * into stack
)	*)	DC+	Push) into stack
B	*)	DC+B	Add B into expression string
+	*)+	DC+B	Push + into stack
A	*)+	DC+BA	Add A into expression string
(	*	DC+BA+	(Pair matched so pop + from stack
		DC+BA+*	Pop elements one by one from stack

Reverse the expression to get prefix expression: *+AB+CD