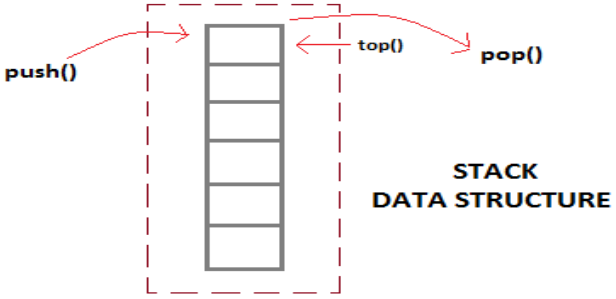
Data Structures *For* BCA 2nd Semester Lecture 3	Compiled *By* Sakhi Bandyopadhyay Dept. of Computer Science & BCA, Kharagpur College, Kharagpur 721305

Stack

Stack is an ordered list in which, insertion and deletion can be performed only at one end that is called **top**.
Stack is a recursive data structure having pointer to its top element.
Stacks are sometimes called as Last-In-First-Out (LIFO) lists i.e. the element which is inserted first in the stack, will be deleted last from the stack.



A stack can be implemented by means of Array, Structure, Pointer, and Linked List.

Basic Operations:

- **push()** – Pushing (storing) an element on the stack.
- **pop()** – Removing (accessing) an element from the stack.
- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

push operation:
Algorithm:

```
begin
        if top = n then stack full
        top = top + 1
        stack (top) : = item;
end
```

pop operation:

Algorithm:

```
begin
        if top = 0 then stack empty;
        item := stack(top);
        top = top - 1;
end
```

peek operation:
Algorithm:

```
begin
        if top = -1 then stack empty
```

```
        item = stack[top]
    return item
end
```

EXPRESSIONS:

The way to write arithmetic expression is known as a **notation**. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are –

- Infix Notation (Operand1 Operator Operand2)
- Prefix (Polish) Notation (Operand1 Operand2 Operator)
- Postfix (Reverse-Polish) Notation (Operator Operand1 Operand2)

These notations are named as how they use operator in expression.

Infix Expression: operators are used **in**-between operands. Example: a+b

Postfix Expression: operator is written after the operands. Example: ab+

Prefix Expression: operator is written ahead of operands. Example: +ab

Algorithm to convert Infix To Postfix:

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1) Push "(" onto Stack, and add ")" to the end of X.
2) Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3) If an operand is encountered, add it to Y.
4) If a left parenthesis is encountered, push it onto Stack.
5) If an operator is encountered ,then:
   a) Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
   b) Add operator to Stack.
   [End of If]
6) If a right parenthesis is encountered ,then:
   a) Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   b) Remove the left Parenthesis.
   [End of If]
   [End of If]
7) END.

For example, Infix Expression: **A+ (B*C-(D/E^F)*G)*H**, where **^** is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack, that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack, that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Algorithm to Convert Infix To Prefix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent prefix expression Y.

1. Reverse the infix expression.
2. Make Every " (" as ") " and every ") " as " ("
3. Push " (" onto Stack, and add ") " to the end of X.
4. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
5. If an operand is encountered, add it to Y.
6. If a left parenthesis is encountered, push it onto Stack.
7. If an operator is encountered ,then :
   - Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
   - Add operator to Stack.
   [End of If]
8. If a right parenthesis is encountered ,then :
   - Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   - Remove the left Parenthesis.
   [End of If]
   [End of If]
9. Reverse the postfix expression.
10. END.

Example
Infix Expression : A + (B * C - (D / E ^ F) * G) * H
Reverse the infix expression :
H *) G *) F ^ E / D (- C * B (+ A
Make Every " (" as ") " and every ") " as " ("
H * (G * (F ^ E / D) - C * B) + A

Convert expression to postfix form:

Scanned	Stack	Postfix Expression	Description
	(		Start
H	(	H	
*	(*	H	
(	(*(	H	
G	(*(	HG	
*	(*(*	HG	
(	(*(*(	HG	
F	(*(*(	HGF	
^	(*(*(^	HGF	
E	(*(*(^	HGFE	
/	(*(*(/	HGFE^	' ^ ' is at highest precedence then ' / '
D	(*(*(/	HGFE^D	
)	(*(*	HGFE^D/	
-	(*(-	HGFE^D/*	
C	(*(-	HGFE^D/*C	
*	(*(-*	HGFE^D/*C	
B	(*(-*	HGFE^D/*CB	
)	(*	HGFE^D/*CB*-	POP from top on Stack, that's why ' * ' come first
+	(+	HGFE^D/*CB*-*	' * ' is at highest precedence then ' + '
A	(+	HGFE^D/*CB*-*A	
)	Empty	HGFE^D/*CB*-*A+	END

Prefix expression are +A*-*BC*/D^EFGH

Evaluation rule of a Postfix Expression states:

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it.
3. Push back the result of the evaluation. Repeat it till the end of the expression.

Expression: 456*+

Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)