CHAPTER 2: THE STACK(3 HOURS)

BY: ER. PRALHAD CHAPAGAIN

LECTURER

CONTENT

INTRODUCTION

- A stack is an ordered collection of homogenous data elements where the insertion and deletion operation occurs at only one ends called **top** of the stack.
- As all the insertion and deletion is performed from top of the stack, the last item that is inserted in a stack is the first one to be deleted.
- Therefore stack is also called as **last in first out (LIFO)** data structure.
- The insertion (addition) operation is referred to as PUSH and the deletion (remove) operation as POP.
- A stack is said to be empty or underflow, if the stack contains no elements. At this point, the **top** of the stack is present at the bottom of the stack.
- In addition, it is overflow when the stack becomes full, i.e. , no other elements can be pushed onto the stack. At this point, **top** pointer is at the highest location of the stack.
- Consider an array of size[4]with the name stack i.e. stack[4] then the insertion and deletion operation that can be performed on stack is shown below:

INTRODUCTION

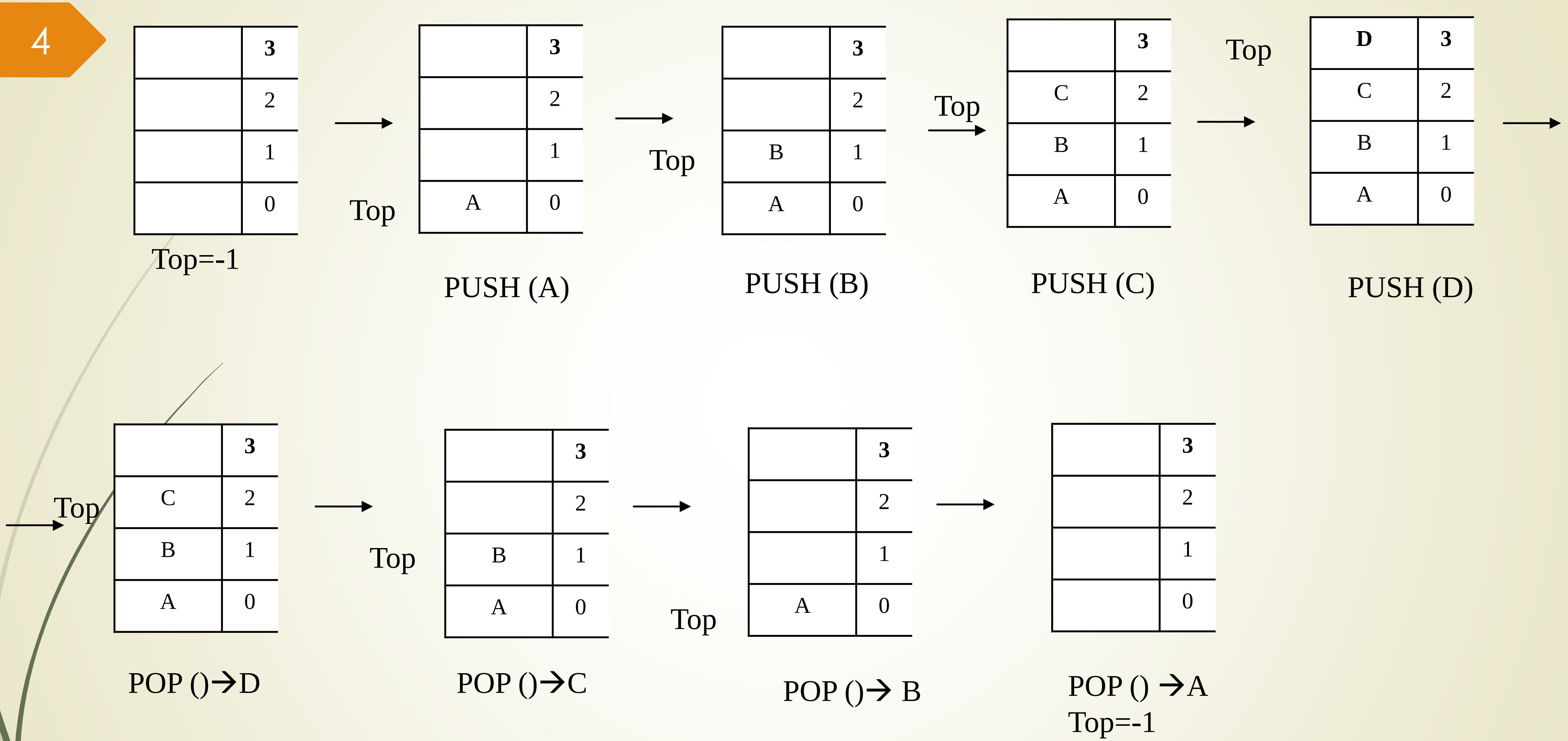


Fig: PUSH and POP operation

STACK AS AN ADT/ OPERATION ON STACK

- Stack is generally implemented with two basic operation such as PUSH and POP.
- These two operations are also called as primitive operation on stack.

PUSH operation

- The processing of adding a new element to the top of stack is called PUSH operation.
- For each PUSH operation, the value of top is increased by 1.
- If the array is full and no new element can be accommodate, then the stack overflows condition occurs.
- E.g. PUSH (A,S)→ insert element A at the top of the stack, S

POP operation:

- The process of deleting an element from the top of the stack is called POP operation.
- After every POP operation, the stack is decremented by one.
- If there is no element in the stack and the POP operation is performed then the stack underflow condition occurs.

STACK AS AN ADT/ OPERATION ON STACK

- E.g. POP(S) → deleting the element from top of the stack, S

- Also, we can define following additional operation that can be performed on stack.
- **TOP (S)** → Return the elements at the top of the stack.
- **ISEMPTY (S)** → Check the stack, S, is empty or not. If the stack is empty it returns true (1) otherwise false (0)
- **ISFULL (S)** → Check the stack, S, is full or not. If the stack is full it returns true (1) otherwise false (0)
- **MAKENULL (S)** → Make stack S is an empty stack.
- **SEARCH (X,S)** → Get the location of item X in stack S.
- **TRAVERSE (S)** → Read each elements only once from the stack S
- **MAX (S)** → Get the maximum value from the stack, S
- **MIN (S)** → Get the minimum value from the stack, S

ALGORITHM FOR PUSH AND POP OPERATION

PUSH OPERATION:

- Assuming a stack named STACK of size MAX with the TOP pointer that points to the top most element of the stack and DATA is the data item to be pushed.

```
1. If TOP=MAX-1 then
    a. Display "Stack Overflow"
    b. Exit
2. Read DATA
3. TOP=TOP+1
4. STACK[TOP]← DATA
5. Exit
```

ALGORITHM FOR PUSH AND POP OPERATION

POP OPERATION:

- Assuming a stack named STACK of size MAX with the TOP pointer that points to the top most element of the stack and DATA is the data item to be popped.

```
1. If TOP=-1 then
    a. Display "Stack underflow"
    b. Exit
2. DATA←STACK[TOP]
3. TOP=TOP-1
4. Exit
```

STACK IMPLEMENTATION

- Static Implementation (Using Arrays)
- Dynamic Implementation (Using Pointer)

Static Implementation:

- Static implementation using array is very simple technique but is not a flexible way, as the size of the stack has to be declared during the program design, because after that, the size can not be varied.
- Moreover, static implementation is not an efficient method when resource optimization is concerned.

Dynamic Implementation:

- when a stack is implemented by using link list i.e. uses pointer to implement the stack, then the stack is said to be dynamic.
- In this case memory is dynamically allocated to the stack and the memory size can grow and shring at run time.

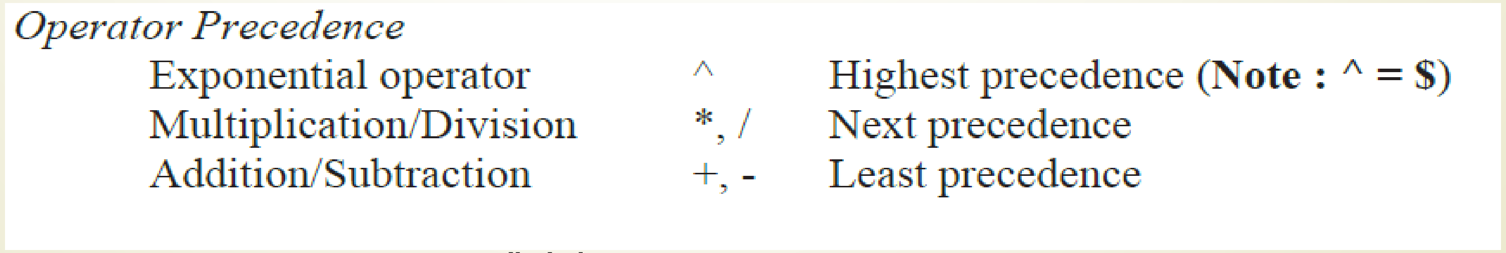
STACK APPLICATION

- It is used to reverse the string.
- It is used to implement function call.
- It is used to maintaining the undo list for an application.
- It is used for checking the validity for expression
- It is used for converting infix expression to post fix or prefix expression
- To keep page visited history in web browser.
- Sorting
- Backtracking

INFIX, PREFIX AND POSTFIX NOTATION

Infix notation: The operator symbol is placed between two operand. E.g. a+b

- **Prefix notation:** The operator symbol is placed before two operand E.g. +ab
- **Postfix Notation:** The operator is placed after two operand E.g. ab+
- Using infix notation, if the expression consists of more than one operator and brackets, the precedence rule (BODMAS) should be applied to decide which operator or which section of expression are evaluated first.
- So, the computer usually evaluates an infix expression first by converting it to postfix and evaluating the postfix expression by using stack.
- As soon as an operator appears in the postfix expression during scanning of postfix expression the topmost operands are popped off and are calculated by applying the encountered operator.



Operator Precedence

Exponential operator	^	Highest precedence (**Note :** ^ = $)
Multiplication/Division	*, /	Next precedence
Addition/Subtraction	+, -	Least precedence

CONVERSION OF INFIX TO POSTFIX EXPRESSION

ALGORITHM FOR CONVERTING INFIX EXPRESSION TO POSTFIX EXPRESSION USING STACK

- Suppose Q is an arithmetic expression written in infix notation, and P is the expression written in postfix notation, then algorithm that finds the equivalent postfix expression is given below:

1. Scan Q from left to right for each element of Q until end of infix expression is encountered
2. If scanned element is
   a. **Operand:**
      I. Add the operand to the postfix expression, P
   b. **Operator:**
      I. If the precedence of the operator on the top of the stack is greater or equal to precedence of scanned operator, pop the operator from the stack and append it to the postfix expression, P. Repeat this step until the operator at the top of the stack has precedence less than the scanned operator or stack become empty.

CONVERSION OF INFIX TO POSTFIX EXPRESSION

  II. Push the scanned operator on the top of the stack.

c. Left parenthesis:

  I. Push the left parenthesis on to the top of stack.

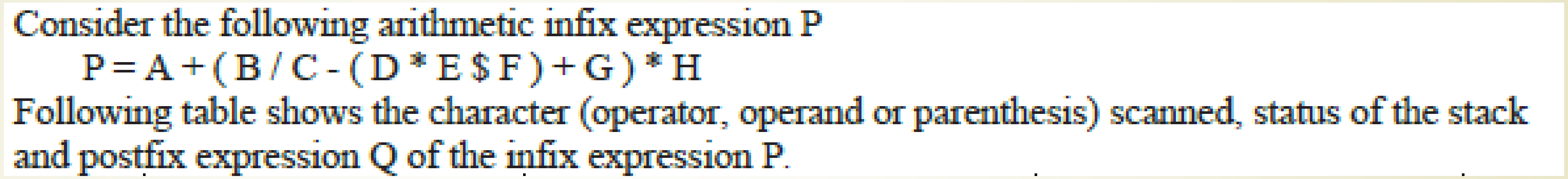
d. Right parenthesis

  I. Pop stack element one by one and append to the postfix expression, until a last parenthesis is popped.

3. POP all the entries from the stack and append them to postfix expression

CONVERSION OF INFIX TO POSTFIX EXPRESSION

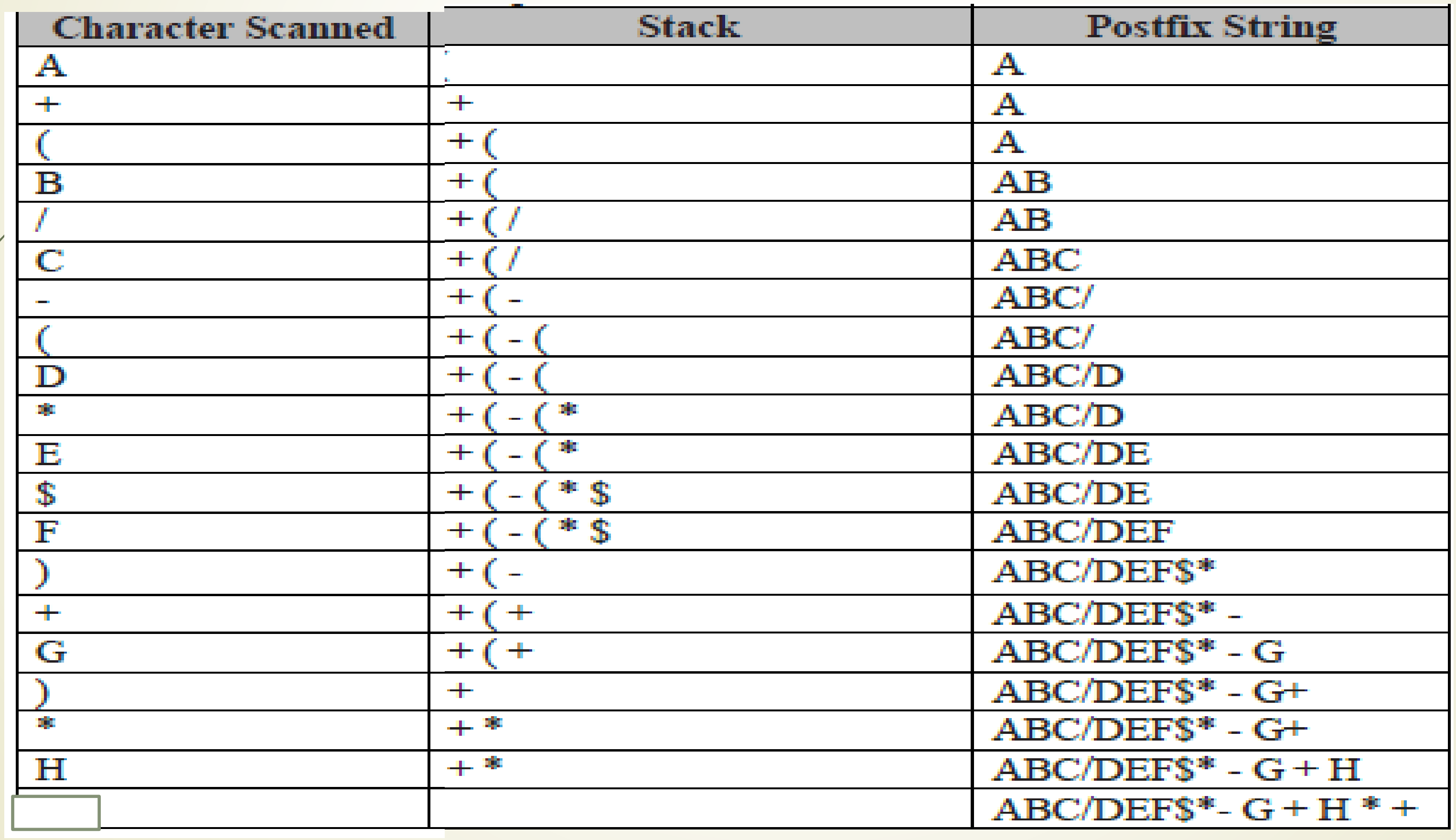
Example:



Consider the following arithmetic infix expression P

P = A + (B / C - (D * E $ F) + G) * H

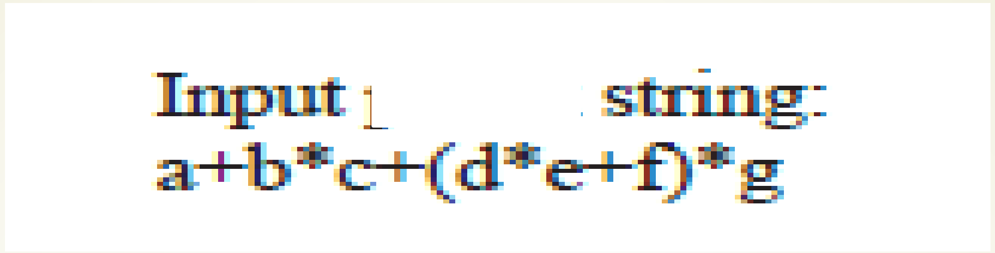
Following table shows the character (operator, operand or parenthesis) scanned, status of the stack and postfix expression Q of the infix expression P.

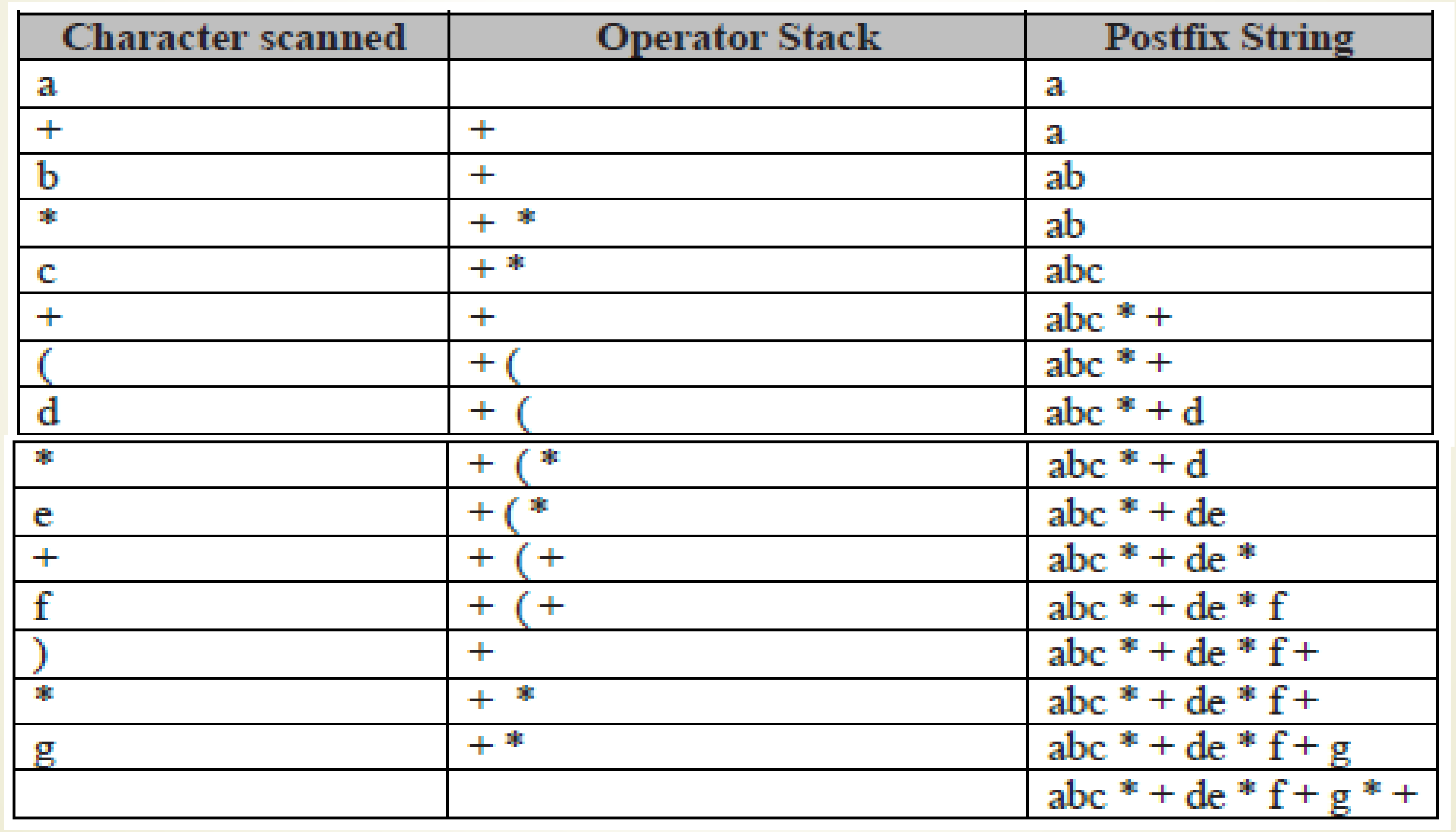


Character Scanned	Stack	Postfix String
A		A
+	+	A
(	+ (	A
B	+ (	AB
/	+ (/	AB
C	+ (/	ABC
-	+ (-	ABC/
(	+ (- (	ABC/
D	+ (- (	ABC/D
*	+ (- (*	ABC/D
E	+ (- (*	ABC/DE
$	+ (- (* $	ABC/DE
F	+ (- (* $	ABC/DEF
)	+ (-	ABC/DEF$*
+	+ (+	ABC/DEF$* -
G	+ (+	ABC/DEF$* - G
)	+	ABC/DEF$* - G+
*	+ *	ABC/DEF$* - G+
H	+ *	ABC/DEF$* - G + H
		ABC/DEF$*- G + H * +

CONVERSION OF INFIX TO POSTFIX EXPRESSION

Example:


Input string:
a+b*c+(d*e+f)*g

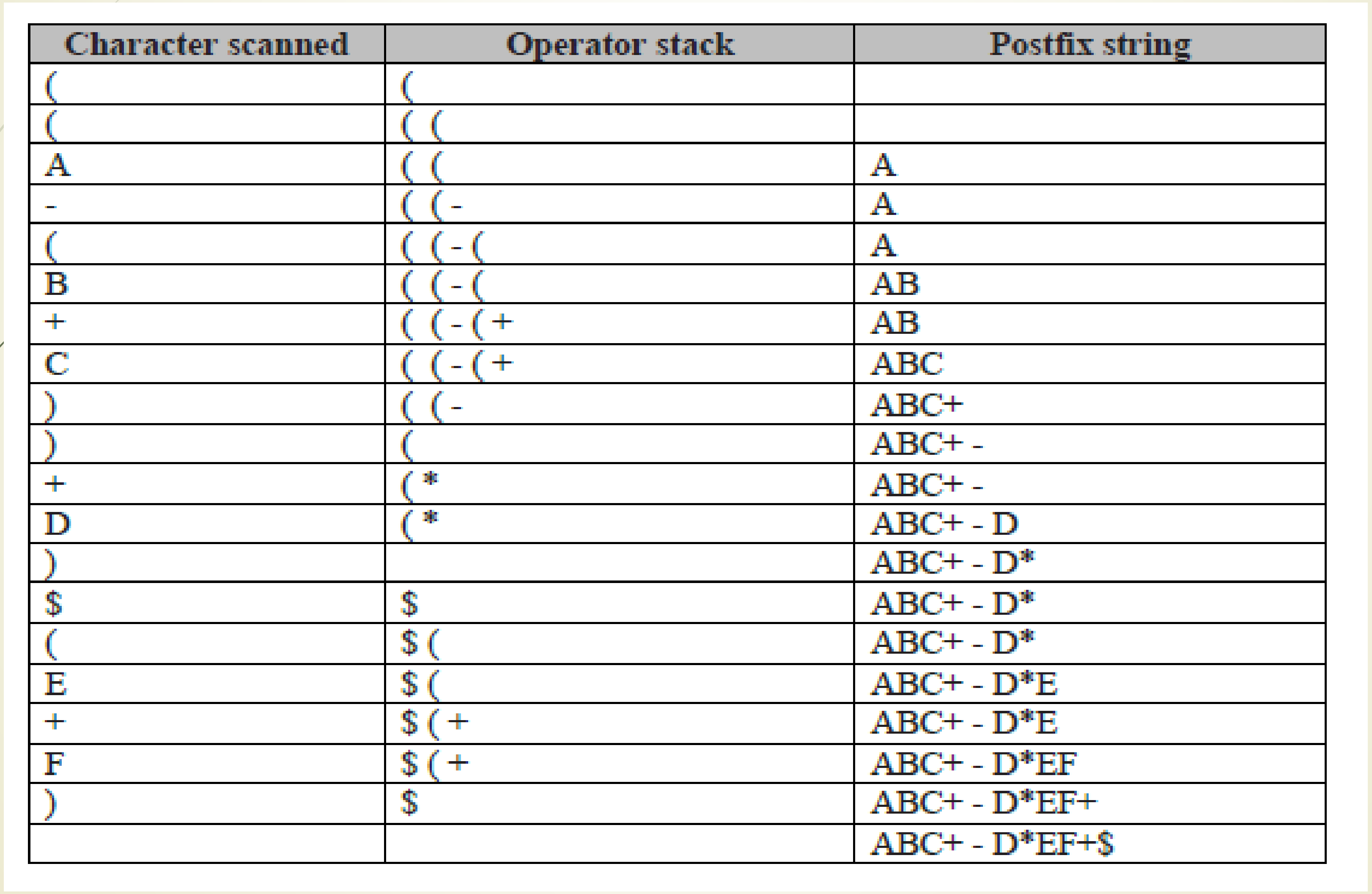


Character scanned	Operator Stack	Postfix String
a		a
+	+	a
b	+	ab
*	+ *	ab
c	+ *	abc
+	+	abc * +
(	+ (	abc * +
d	+ (	abc * + d
*	+ (*	abc * + d
e	+ (*	abc * + de
+	+ (+	abc * + de *
f	+ (+	abc * + de * f
)	+	abc * + de * f +
*	+ *	abc * + de * f +
g	+ *	abc * + de * f + g
		abc * + de * f + g * +

CONVERSION OF INFIX TO POSTFIX EXPRESSION

Example:

Input string ((A - (B+C)) * D) $ (E + F)



Character scanned	Operator stack	Postfix string
(	(	
(	((	
A	((	A
-	((-	A
(	((- (	A
B	((- (	AB
+	((- (+	AB
C	((- (+	ABC
)	((-	ABC+
)	(	ABC+ -
+	(*	ABC+ -
D	(*	ABC+ - D
)		ABC+ - D*
$	$	ABC+ - D*
(	$ (	ABC+ - D*
E	$ (	ABC+ - D*E
+	$ (+	ABC+ - D*E
F	$ (+	ABC+ - D*EF
)	$	ABC+ - D*EF+
		ABC+ - D*EF+$

For practice:

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

ANS: AB-C-DE*+F+

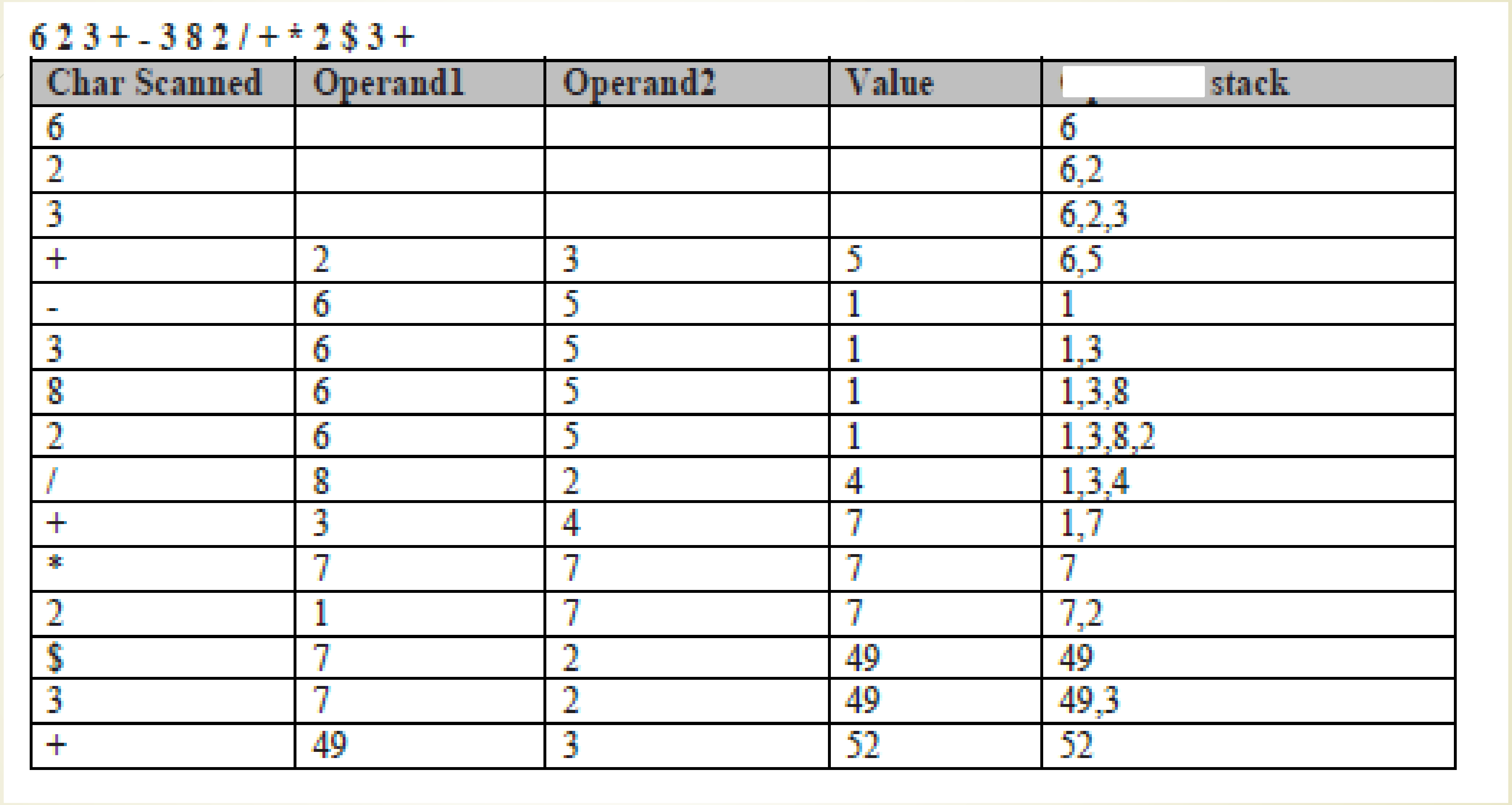
EVALUATION OF POSTFIX EXPRESSION

Once the infix expression is converted into postfix expression, we can write the **algorithm** for evaluating postfix expression as below:

1. Scan the elements in postfix expression from left to right until end of the postfix expression is encountered.
2. If the scanned element is:
   a. Operand : PUSH it into stack
   b. Operator :
      I. POP the top two values
      II. Operate the two values by using the operator (just below the top of the stack operand to top of the stack operand)
      III. PUSH the result back to the stack.
3. Get the result from top of the stack.

EVALUATION OF POSTFIX EXPRESSION

Example:



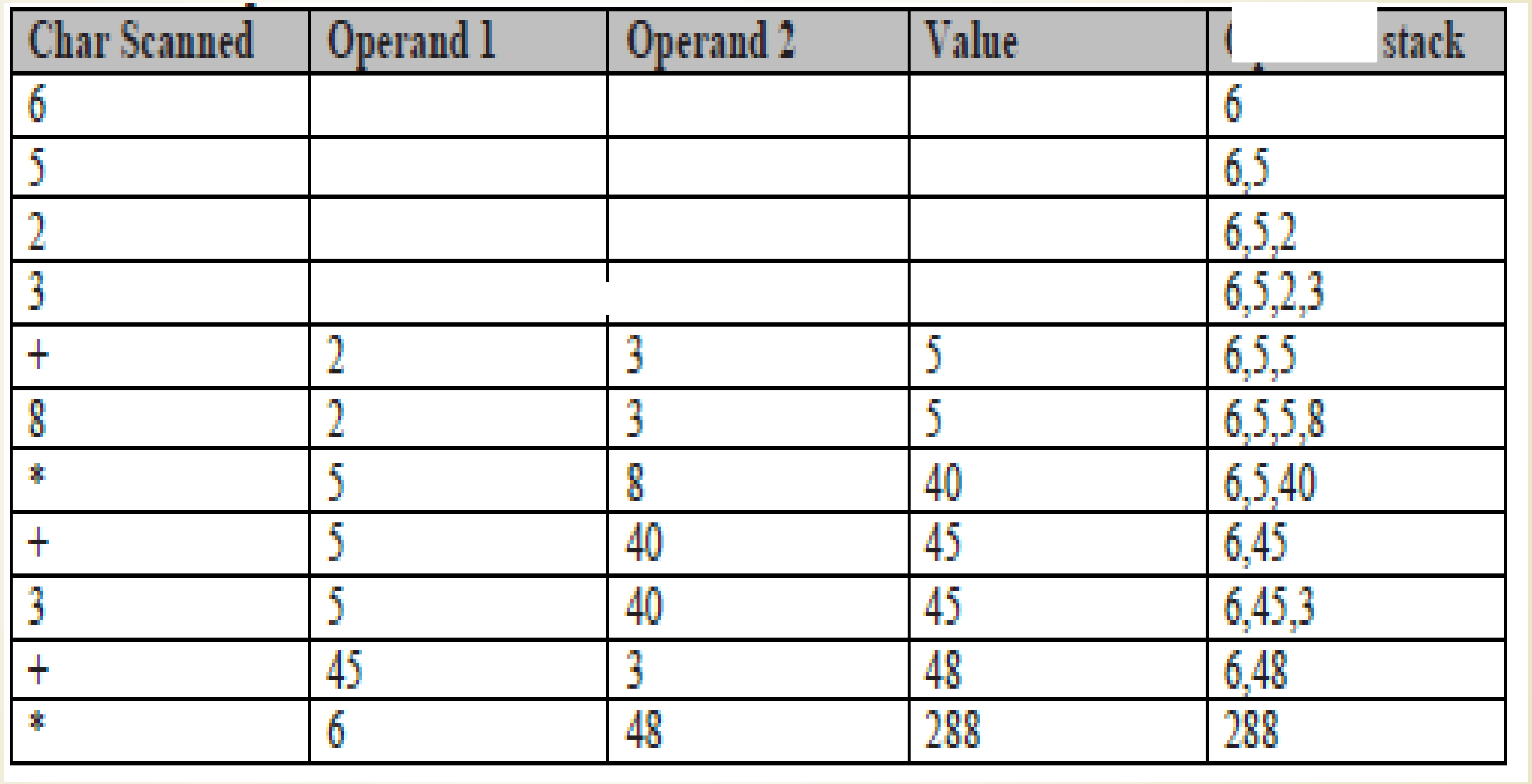
6 2 3 + - 3 8 2 / + * 2 $ 3 +

Char Scanned	Operand1	Operand2	Value	stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	7	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

EVALUATION OF POSTFIX EXPRESSION

Example:

6 5 2 3 + 8 * + 3 + *



Char Scanned	Operand 1	Operand 2	Value	O stack
6				6
5				6,5
2				6,5,2
3				6,5,2,3
+	2	3	5	6,5,5
8	2	3	5	6,5,5,8
*	5	8	40	6,5,40
+	5	40	45	6,45
3	5	40	45	6,45,3
+	45	3	48	6,48
*	6	48	288	288

EVALUATION OF POSTFIX EXPRESSION

Example:

- ABC+*DE/-   where A=5, B=6, C=2, D=12, E=4

Solution:

Given expression becomes: 5 6 2 +* 12 4 / -

Char Scanned	Operand 1	Operand 2	Value	Stack
5				5
6				5, 6
2				5,6,2
+	6	2	8	5,8
*	5	8	40	40
12	5	8	40	40, 12
4	5	8	40	40, 12, 4
/	12	4	3	40, 3
-	40	3	37	37

INFIX TO PREFIX CONVERSION

- Suppose Q is an arithmetic expression written in infix notation then the algorithm that finds the equivalent prefix expression is given below:

1. Reverse the element of arithmetic expression, Q
2. Replace → (by), { by } , [by] ,) by (, } by { and] by [
3. Convert the reversed arithmetic expression Q into postfix expression.
4. Scan Q from left to right for each element of Q until end of infix expression is encountered
5. If scanned element is
   a. Operand:
      i. Add the operand to the postfix expression, P
   b. Operator:
      i. If the precedence of the operator on the top of the stack is greater or equal to precedence of scanned operator, pop the operator from the stack and append it to the postfix expression, P. Repeat this step until the operator at the top of the stack has precedence less than the scanned operator or stack become empty.

INFIX TO PREFIX CONVERSION

   ii. Push the scanned operator on the top of the stack.

c) Left parenthesis:

   i. Push the left parenthesis on to the top of stack.

d) Right parenthesis

   i. Pop stack element one by one and append the to the postfix expression, until a last parenthesis is popped.

6. POP all the entries from the stack and append them to postfix expression
7. Reverse the obtained postfix expression element to obtain required prefix expression.

INFIX TO PREFIX CONVERSION

Example:

Convert into prefix expression : (A + B * C)

Solution: Reverse the above infix expression:

) C * B + A (

Replacing (by) and) by (, we get

(C * B + A)

Scanned Character	Stack	Postfix Expression
(	(	
C	(	C
*	(*	C
B	(*	C B
+	(+	C B *
A	(+	C B * A
)		C B * A +

We know, Prefix is equal to reverse of postfix expression. So, the required prefix expression is:

+ A * B C

INFIX TO PREFIX CONVERSION

Example:

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

Solution: Reverse the above infix expression:

)D-C(*)B+A(

Replacing (by) and) by (, we get

(D-C)*(B+A)

Scanned Character	Stack	Postfix Expression
(	(	
D	(	D
-	(-	D
C	(-	DC
)		DC-
*	*	DC-
(	*(	DC-
B	*(	DC-B
+	*(+	DC-B
A	*(+	DC-BA
)	*	DC-BA+
		DC-BA+*

We know, Prefix is equal to reverse of postfix expression. So, the required prefix expression is:

*+AB-CD

EVALUATION OF PREFIX EXPRESSION

1. Accept Prefix expression
2. Reverse the input prefix expression or scan the element in prefix expression from right to left until the end of the prefix expression is encountered.
3. If the scanned character is operand, PUSH it onto the stack.
4. If it is operator, then POP two elements from stack and perform the operation performed by operator (from top of the stack operand to just below the top of stack operand) and PUSH the result back to stack.
5. Get result from top of the stack

EVALUATION OF PREFIX EXPRESSION

Example:

Evaluate the following prefix expression: a) /+5,3-4,2   b) +5*3,2

Char Scanned	Operand 1	Operand 2	Value	Stack
2				2
4				2,4
-	4	2	2	2
3	4	2	2	2,3
5	4	2	2	2,3,5
+	5	3	8	2,8
/	8	2	4	4

Char Scanned	Operand 1	Operand 2	Value	Stack
2				2
3				2,3
*	3	2	6	6
5	3	2	6	6,5
+	5	6	11	11

POSTFIX TO INFIX CONVERSION

1. Scanned the POSTFIX expression from left to right
2. If scanned character is
   a. Operand
      i. PUSH it into the stack
   b. Operator
      i. Pop the top two values from the stack
      ii. Put the operator, with the values as arguments and form a string
      iii. Encapsulate the resulted string with parenthesis
      iv. PUSH the resulted string back to stack
3. The value in the stack is the desired infix string.

POSTFIX TO INFIX CONVERSION

Example:

ABC-+DE-FG-H+/*

Expression	**Stack**
ABC-+DE-FG-H+/*	Null
BC-+DE-FG-H+/*	A
C-+DE-FG-H+/*	A,B
-+DE-FG-H+/*	A,B,C
+DE-FG-H+/*	A,(B-C)
DE-FG-H+/*	(A+(B-C))
E-FG-H+/*	(A+(B-C)),D
-FG-H+/*	(A+(B-C)),D,E
FG-H+/*	(A+(B-C)),(D-E)
G-H+/*	(A+(B-C)),(D-E),F
-H+/*	(A+(B-C)),(D-E),F,G
H+/*	(A+(B-C)),(D-E),(F-G)
+/*	(A+(B-C)),(D-E),(F-G),H
/*	(A+(B-C)),(D-E),((F-G)+H)
*	(A+(B-C)),((D-E)/((F-G)+H))
	(A+(B-C))*((D-E)/((F-G)+H))

Your Queries Please!!!

THANK
YOU