



CS-2001 DATA STRUCTURE

Dr. Hashim Yasin

**National University of Computer
and Emerging Sciences,
Faisalabad, Pakistan.**

EXPRESSIONS

Expressions

3

- An **algebraic expression** is a legal combination of operands and the operators.
- **Operand** is the *quantity* on which a mathematical operation is performed.
- **Operator** is a *symbol* which signifies a mathematical or logical operation.

Expressions

4

- **INFIX:** expressions in which operands surround the operator.
- **POSTFIX:** operator comes after the operands, also known as *Reverse Polish Notation (RPN)*.
- **PREFIX:** operator comes before the operands, also Known as Polish notation.

Example

- Infix: $A+B$
- Postfix: $AB+$
- Prefix: $+AB$

Examples

5

Infix

$A+B$

$(A+B) * (C + D)$

$A-B/(C*D^E)$

PostFix

$AB+$

$AB+CD+*$

$ABCDE^*/-$

Prefix

$+AB$

$*+AB+CD$

$-A/B*C^DE$

INFIX TO POSTFIX CONVERSION



Infix to Postfix ... Algorithm

7

Algorithm: Q is the given infix expression & we want P.

1. Scan Q from left to right and repeat steps 2 to 6 for each element of Q until the STACK is empty.
2. If an operand is encountered, add it to P
3. If a left parenthesis is encountered, push it onto STACK.
4. If an operator X is encountered, then:
 - a. Repeatedly pop from STACK and add to P each operator which has same or higher precedence than X
 - b. Push X on STACK

Infix to Postfix ... Algorithm

8

5. If a right parenthesis is encountered, then:
 - a. Repeatedly pop from STACK and add to P each operator until a left parenthesis is encountered
 - b. Remove the left parenthesis. [Do not add it to P]
6. Exit

Infix to Postfix ... Examples

9

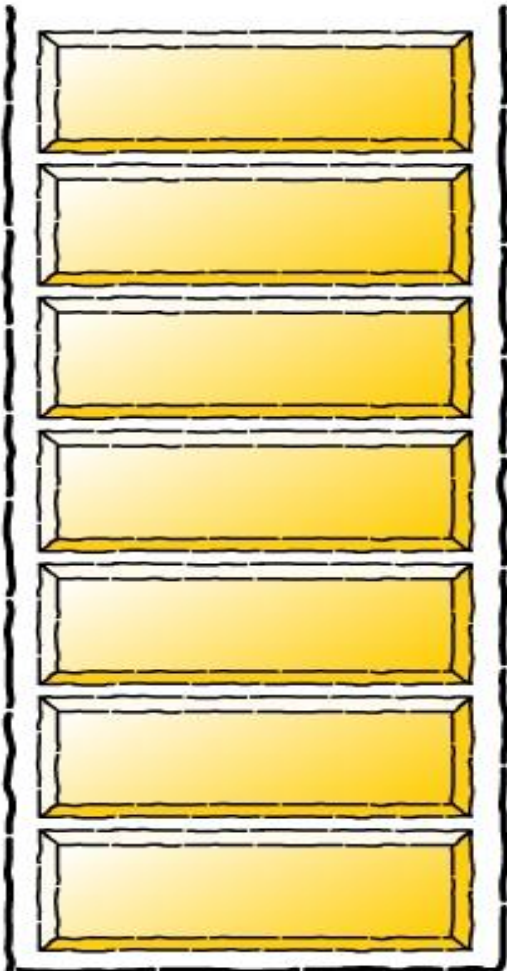
$$(((A + B) * (C - E)) / (F + G))$$

- stack: <empty>
- output: [A B + C E - * F G + /]

Infix to Postfix ... Examples

10

stackVect



infixVect

$(a + b - c) * d - (e + f)$

postfixVect

$a b + c - d * e f + -$

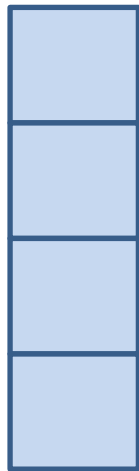
Infix to Postfix ... Examples

11

Transform Infix to Postfix

Ex: $10 + 2 * 8 - 3$

- Because the expression is ended, we pop all the operators in the stack



10 2 8 * + 3 -

EVALUATING A POSTFIX EXPRESSION



Evaluating a Postfix Expression

13

Algorithm: P is the given postfix expression.

1. Scan P from left to right and repeat steps 3 & 4 for each element of P until the sentinel “)” is encountered.
2. If an operand is encountered, push it on STACK
3. If an operator is encountered, then:
 - a. Pop two operands from STACK: A & B
 - b. Evaluate: A operator B
 - c. Push result on STACK
4. Set value equal to the top element on STACK
5. Exit

Evaluating a Postfix Expression

14

WHILE more input items exist

 Get an item

 IF item is an operand

`stack.Push(item)`

 ELSE

`stack.Pop(operand2)`

`stack.Pop(operand1)`

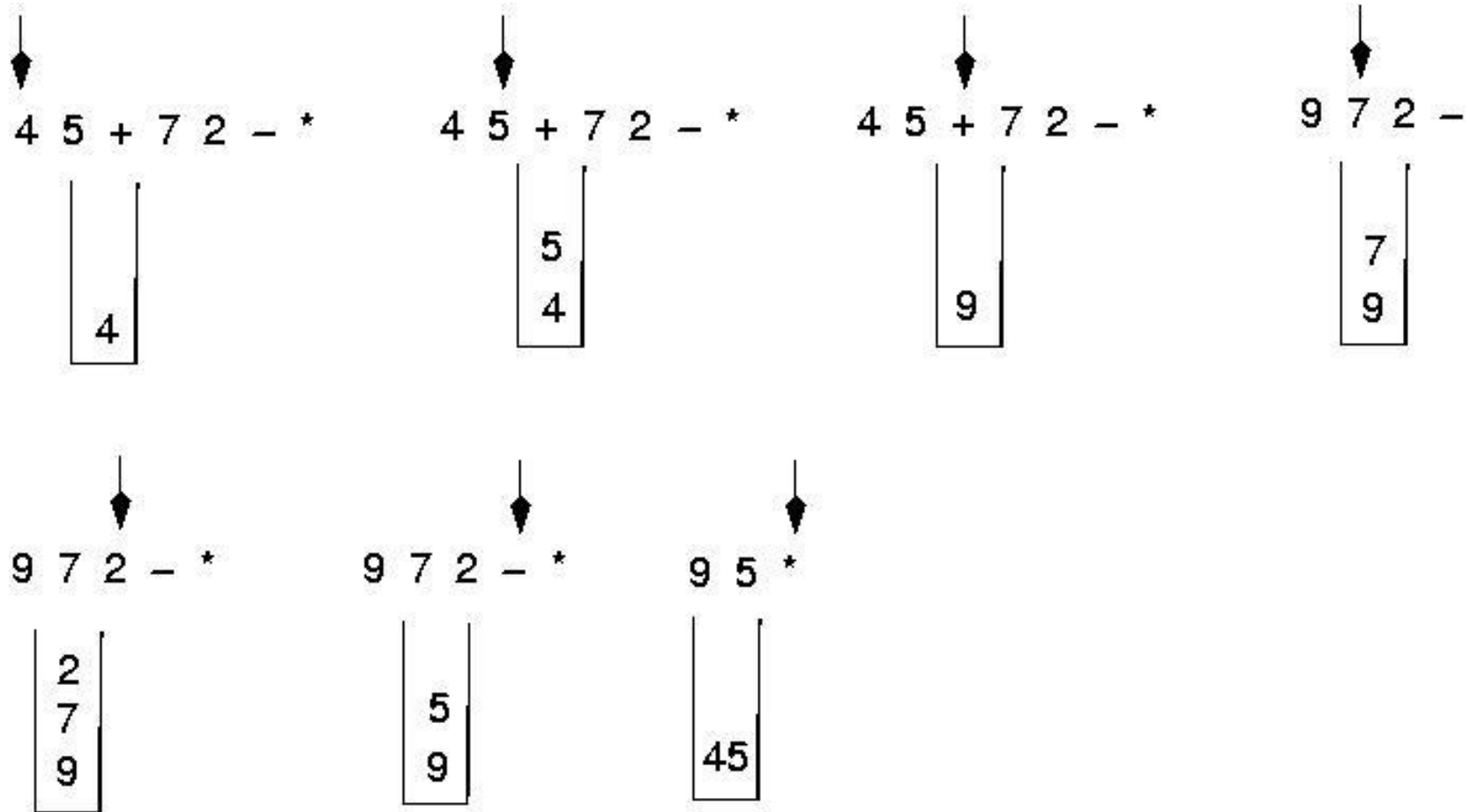
 Compute result

`stack.Push(result)`

`stack.Pop(result)`

Evaluating a Postfix Expression

15



P= 623+-382/+*2\$3+

S.N.	Symbol Scan	Operand 1	Operand 2	Value	STACK
1.	6				6
2.	2				6,2
3.	3				6,2,3
4.	+	2	3	5	6,5
5.	-	6	5	1	1



P= 623+-382/+*2\$3+

S.N.	Symbol Scan	Operand 1	Operand 2	Value	STACK
1.	6				6
2.	2				6,2
3.	3				6,2,3
4.	+	2	3	5	6,5
5.	-	6	5	1	1



Next ?

$$P = 623 + -382 / + * 2 \$ 3 +$$

S.N.	Symbol Scan	Operand 1	Operand 2	Value	STACK
1.	6				6
2.	2				6,2
3.	3				6,2,3
4.	+	2	3	5	6,5
5.	-	6	5	1	1
6.	3				1,3
7.	8				1,3,8
8.	2				1,3,8,2
9.	/	8	2	4	1,3,4
10.	+	3	4	7	1,7
11.	*	1	7	7	7
12.	2				7,2
13.	\$	7	2	49	49
14.	3				49,3
15.	+	49	3	52	52

INFIX TO PREFIX CONVERSION

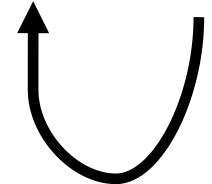


Infix to Prefix Conversion

20

Move each operator to the left of its operands & remove the parentheses:

$((A + B) * (C + D))$

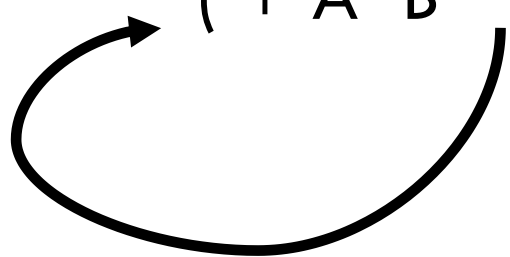


Infix to Prefix Conversion

21

Move each operator to the left of its operands & remove the parentheses:

(+ A B * (C + D))

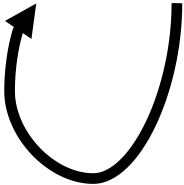


Infix to Prefix Conversion

22

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)



Infix to Prefix Conversion

23

Move each operator to the left of its operands & remove the parentheses:

$* + A \ B \ + C \ D$

Order of operands does not change!

Example;

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



S.N.	Scan symbol	Prefix stack	Opstack
1.))
2.	F	F)
3.	-	F)-
4.)	F)-)
5.	E	FE)-)
6.	*	FE)-)*
7.	D	FED)-)*
8.	(FED*)-

Next ?

Example;

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



S.N.	Scan symbol	Prefix stack	Opstack
1.))
2.	F	F)
3.	-	F)-
4.)	F)-)
5.	E	FE)-)
6.	*	FE)-)*
7.	D	FED)-)*
8.	(FED*)-
9.	(FED*-	
10.	*	FED*-	*
11.)	FED*-	*)
12.)	FED*-	*))
13.	C	FED*-C	*))
14.	/	FED*-C	*)) /
15.	B	FED*-CB	*)) /
16.	(FED*-CB /	*)
17.	-	FED*-CB /	*) -
18.	A	FED*-CB / A	*) -
19.	(FED*-CB / A -	*
		FED*-CB / A -*	

Hence, the required prefix expression is $*-A/BC-*DEF$

EVALUATING A PREFIX EXPRESSION



Evaluation of Prefix Expression

27

1) Read prefix string from right to left until there is a data.

2) Repeat;

If char is operand add to prestack

If char is operator

- operand 1 = pop prestack.
- operand 2 = pop prestack.
- result = value after applying operator between operand 1 and operand 2.
- push the result into prestack.

3) pop prestack get required value.

Tracing

+-*+12/421\$42



S.N.	Scan Symbol	Operand 1	Operand 2	Value	Prestack
1.	2				2
2.	4				2,4
3.	\$	4	2	16	16
4.	1				16,1
5.	2				16,1,2

Next ?

Tracing

+-*+12/421\$42



S.N.	Scan Symbol	Operand 1	Operand 2	Value	Prestack
1.	2				2
2.	4				2,4
3.	\$	4	2	16	16
4.	1				16,1
5.	2				16,1,2
6.	4				16,1,2,4
7.	/	4	2	2	16,1,2
8.	2				16,1,2,2
9.	1				16,1,2,2,1
10.	+	1	2	3	16,1,2,3
11.	*	3	2	6	16,1,6
12.	-	6	1	5	16,5
13.	+	5	16	21	21

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

Reverse Infix expression: $)E-D^C(*B+A$

Reverse brackets: $(E-D^C)*B+A$

Token	Action	Result	Stack	Notes
(Push (to stack		(
E	Add E to the result	E	(
-	Push - to stack	E	(-	
D	Add D to the result	ED	(-	
^	Push ^ to stack	ED	(- ^	
C	Add C to the result	EDC	(- ^	
)	Pop ^ from stack and add to result	EDC^	(-	Do process until (is popped from stack
	Pop - from stack and add to result	EDC^-	(
	Pop (from stack	EDC^-		
*	Push * to stack	EDC^-	*	
B	Add B to the result	EDC^-B	*	
+	Pop * from stack and add to result	EDC^-B		- has lower precedence than ^
	Push + to stack	EDC^-B*	+	
A	Add A to the result	EDC^-B*A	+	
	Pop + from stack and add to result	EDC^-B*A+		Given expression is iterated, do Process till stack is not Empty, It will give the final result

Prefix Expression (Reverse Result): $+A*B-^CDE$

POSTFIX TO INFIX CONVERSION



Postfix to Infix ... Algorithm

32

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

1. If a **character is operand**, push it to stack.
2. If a **character is an operator**,
 1. pop operand from the stack, say it's **s1**.
 2. pop operand from the stack, say it's **s2**.
 3. perform **(s2 operator s1)** and push it to stack.
3. Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.
4. Return the result.

Postfix to Infix ... Examples

33

Input: Postfix expression: $A B +$

Output: Infix expression- $(A + B)$

Input: Postfix expression: $ABC/-AK/L-*$

Output: Infix expression: $((A-(B/C))*((A/K)-L))$

Postfix Expression : ABC/-AK/L-*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform B/C and push (B/C) to stack
	Pop B from stack	[A]	
	Push (B/C) to stack	[A, (B/C)]	

Postfix Expression : ABC/-AK/L-*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform B/C and push (B/C) to stack
	Pop B from stack	[A]	
	Push (B/C) to stack	[A, (B/C)]	
-	Pop (B/C) from stack	[A]	Pop two operands from stack, (B/C) and A. Perform A-(B/C) and push (A-(B/C)) to stack
	Pop A from stack	[]	
	Push (A-(B/C)) to stack	[(A-(B/C))]	
A	Push A to stack	[(A-(B/C)), A]	
K	Push K to stack	[(A-(B/C)), A, K]	
/	Pop K from stack	[(A-(B/C)), A]	Pop two operands from stack, K and A. Perform A/K and push (A/K) to stack
	Pop A from stack	[(A-(B/C))]	
	Push (A/K) to stack	[(A-(B/C)), (A/K)]	

Postfix Expression : ABC/-AK/L-*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform B/C and push (B/C) to stack
	Pop B from stack	[A]	
	Push (B/C) to stack	[A, (B/C)]	
-	Pop (B/C) from stack	[A]	Pop two operands from stack, (B/C) and A. Perform A-(B/C) and push (A-(B/C)) to stack
	Pop A from stack	[]	
	Push (A-(B/C)) to stack	[(A-(B/C))]	
A	Push A to stack	[(A-(B/C)), A]	
K	Push K to stack	[(A-(B/C)), A, K]	
/	Pop K from stack	[(A-(B/C)), A]	Pop two operands from stack, K and A. Perform A/K and push (A/K) to stack
	Pop A from stack	[(A-(B/C))]	
	Push (A/K) to stack	[(A-(B/C)), (A/K)]	
L	Push L to stack	[(A-(B/C)), (A/K), L]	
-	Pop L from stack	[(A-(B/C)), (A/K)]	Pop two operands from stack, L and (A/K). Perform (A/K)-L and push ((A/K)-L) to stack
	Pop (A/K) from stack	[(A-(B/C))]	
	Push ((A/K)-L) to stack	[(A-(B/C)), ((A/K)-L)]	
*	Pop ((A/K)-L) from stack	[(A-(B/C))]	Pop two operands from stack, (A/K)-L and A-(B/C). Perform (A-(B/C))*((A/K)-L) and push ((A-(B/C))*((A/K)-L)) to stack
	Pop (A-(B/C)) from stack	[]	
	Push ((A-(B/C))*((A/K)-L)) to stack	[(A-(B/C))*((A/K)-L)]	

Infix Expression: ((A-(B/C))*((A/K)-L))

POSTFIX TO PREFIX CONVERSION



Postfix to Prefix ... Algorithm

38

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

1. If the **character is operand**, push it to stack.
2. If the **character is operator**,
 1. Pop operand from the stack, say it's **s1**.
 2. Pop operand from the stack, say it's **s2**.
 3. perform **(operator s2 s1)** and push it to stack.
3. Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.
4. Return the result.

Postfix Expression : ABC/-AK/L-*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform /BC and push /BC to stack
	Pop B from stack	[A]	
	Push /BC to stack	[A, /BC]	

Postfix Expression : ABC/-AK/L-*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform /BC and push /BC to stack
	Pop B from stack	[A]	
	Push /BC to stack	[A, /BC]	
-	Pop /BC from stack	[A]	Pop two operands from stack, /BC and A. Perform -A/BC and push -A/BC to stack
	Pop A from stack	[]	
	Push -A/BC to stack	[-A/BC]	
A	Push A to stack	[-A/BC, A]	
K	Push K to stack	[-A/BC, A, K]	
/	Pop K from stack	[-A/BC, A]	Pop two operands from stack, K and A. Perform /AK and push /AK to stack
	Pop A from stack	[-A/BC]	
	Push /AK to stack	[-A/BC, /AK]	

Postfix Expression : ABC/-AK/L*

Token	Action	Stack	Notes
A	Push A to stack	[A]	
B	Push B to stack	[A, B]	
C	Push C to stack	[A, B, C]	
/	Pop C from stack	[A, B]	Pop two operands from stack, C and B. Perform /BC and push /BC to stack
	Pop B from stack	[A]	
	Push /BC to stack	[A, /BC]	
-	Pop /BC from stack	[A]	Pop two operands from stack, /BC and A. Perform -A/BC and push -A/BC to stack
	Pop A from stack	[]	
	Push -A/BC to stack	[-A/BC]	
A	Push A to stack	[-A/BC, A]	
K	Push K to stack	[-A/BC, A, K]	
/	Pop K from stack	[-A/BC, A]	Pop two operands from stack, K and A. Perform /AK and push /AK to stack
	Pop A from stack	[-A/BC]	
	Push /AK to stack	[-A/BC, /AK]	
L	Push L to stack	[-A/BC, /AK, L]	
-	Pop L from stack	[-A/BC, /AK]	Pop two operands from stack, L and /AK. Perform /AKL and push -/AKL to stack
	Pop /AK from stack	[-A/BC]	
	Push -/AKL to stack	[-A/BC, -/AKL]	
*	Pop -/AKL from stack	[-A/BC]	Pop two operands from stack, -/AKL and -A/BC. Perform *-A/BC-/AKL and push *-A/BC-/AKL to stack
	Pop -A/BC from stack	[]	
	Push *-A/BC-/AKL to stack	[-A/BC-/AKL]	
Prefix Expression: *-A/BC-/AKL			

PREFIX TO INFIX CONVERSION



Prefix to Infix ... Algorithm

43

Iterate the given expression from **right to left** (in reverse order), one character at a time

1. If **character is operand**, push it to stack.
2. If **character is operator**,
 1. pop operand from stack, say it's **s1**.
 2. pop operand from stack, say it's **s2**.
 3. perform **(s1 operator s2)** and push it to stack.
3. Once the expression iteration is completed, initialize result string and pop out from stack and add it to result.
4. Return the result.

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform A/K and push (A/K) to stack
	Pop K from stack	[L]	
	Push (A/K) to stack	[L, (A/K)]	

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform A/K and push (A/K) to stack
	Pop K from stack	[L]	
	Push (A/K) to stack	[L, (A/K)]	
-	Pop (A/K) from stack	[L]	Pop two operands from stack, (A/K) and L. Perform (A/K)-L and push ((A/K)-L) to stack
	Pop L from stack	[]	
	Push ((A/K)-L) to stack	[((A/K)-L)]	
C	Push C to stack	[((A/K)-L), C]	
B	Push B to stack	[((A/K)-L), C, B]	
/	Pop B from stack	[((A/K)-L), C]	Pop two operands from stack, B and C. Perform B/C and push (B/C) to stack
	Pop C from stack	[((A/K)-L)]	
	Push (B/C) to stack	[((A/K)-L), (B/C)]	

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform A/K and push (A/K) to stack
	Pop K from stack	[L]	
	Push (A/K) to stack	[L, (A/K)]	
-	Pop (A/K) from stack	[L]	Pop two operands from stack, (A/K) and L. Perform (A/K)-L and push ((A/K)-L) to stack
	Pop L from stack	[]	
	Push ((A/K)-L) to stack	(((A/K)-L)]	
C	Push C to stack	(((A/K)-L), C]	
B	Push B to stack	(((A/K)-L), C, B]	
/	Pop B from stack	(((A/K)-L), C]	Pop two operands from stack, B and C. Perform B/C and push (B/C) to stack
	Pop C from stack	(((A/K)-L)]	
	Push (B/C) to stack	(((A/K)-L), (B/C)]	
A	Push A to stack	(((A/K)-L), (B/C), A]	
-	Pop A from stack	(((A/K)-L), (B/C)]	Pop two operands from stack, A and (B/C). Perform A-(B/C) and push (A-(B/C))to stack
	Pop (B/C) from stack	(((A/K)-L)]	
	Push (A-(B/C)) to stack	(((A/K)-L), (A-(B/C))]	
*	Pop (A-(B/C)) from stack	(((A/K)-L)]	Pop two operands from stack, (A-(B/C) and ((A/K)-L). Perform (A-(B/C))*((A/K)-L) and push ((A-(B/C))*((A/K)-L)) to stack
	Pop ((A/K)-L) from stack	[]	
	Push ((A-(B/C))*((A/K)-L)) to stack	(((A-(B/C))*((A/K)-L))]	
Infix Expression: ((A-(B/C))*((A/K)-L))			

PREFIX TO POSTFIX CONVERSION



Prefix to Postfix ... Algorithm

48

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

1. If the **character is operand**, push it to the stack.
2. If the **character is operator**,
 1. Pop an operand from the stack, say it's **s1**.
 2. Pop an operand from the stack, say it's **s2**.
 3. perform **(s1 s2 operator)** and push it to stack.
3. Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.
4. Return the result.

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform AK/ and push AK/ to stack
	Pop K from stack	[L]	
	Push AK/ to stack	[L, AK/]	

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform AK/ and push AK/ to stack
	Pop K from stack	[L]	
	Push AK/ to stack	[L, AK/]	
-	Pop AK/ from stack	[L]	Pop two operands from stack, AK/ and L. Perform AK/L- and push AK/L- to stack
	Pop L from stack	[]	
	Push AK/L- to stack	[AK/L-]	
C	Push C to stack	[AK/L-, C]	
B	Push B to stack	[AK/L-, C, B]	
/	Pop B from stack	[AK/L-, C]	Pop two operands from stack, B and C. Perform BC/ and push BC/ to stack
	Pop C from stack	[AK/L-]	
	Push BC/ to stack	[AK/L-, BC/]	

Prefix Expression : *-A/BC-/AKL

Iterate right to left

Token	Action	Stack	Notes
L	Push L to stack	[L]	
K	Push K to stack	[L, K]	
A	Push A to stack	[L, K, A]	
/	Pop A from stack	[L, K]	Pop two operands from stack, A and K. Perform AK/ and push AK/ to stack
	Pop K from stack	[L]	
	Push AK/ to stack	[L, AK/]	
-	Pop AK/ from stack	[L]	Pop two operands from stack, AK/ and L. Perform AK/L- and push AK/L- to stack
	Pop L from stack	[]	
	Push AK/L- to stack	[AK/L-]	
C	Push C to stack	[AK/L-, C]	
B	Push B to stack	[AK/L-, C, B]	
/	Pop B from stack	[AK/L-, C]	Pop two operands from stack, B and C. Perform BC/ and push BC/ to stack
	Pop C from stack	[AK/L-]	
	Push BC/ to stack	[AK/L-, BC/]	
A	Push A to stack	[AK/L-, BC/, A]	
-	Pop A from stack	[AK/L-, BC/]	Pop two operands from stack, A and BC/. Perform ABC/- and push ABC/- to stack
	Pop BC/ from stack	[AK/L-]	
	Push ABC/- to stack	[AK/L-, ABC/-]	
*	Pop ABC/- from stack	[AK/L-]	Pop two operands from stack, ABC/- and AK/L-. Perform ABC/-AK/L-* and push ABC/-AK/L-* to stack
	Pop AK/L- from stack	[]	
	Push ABC/-AK/L-* to stack	[ABC/-AK/L-*]	

Postfix Expression: ABC/-AK/L-*

Reading Materials

52

- Nell Dale – Chapter#4
- Schaum's Outlines – Chapter#6
- D. S. Malik – Chapter#7
- <http://www.cs.man.ac.uk/~pjj/cs2121/fix.html>
- <https://algorithms.tutorialhorizon.com>