Infix to postfix expression conversions

BY

DR. RASHMI NAVEEN RAJ, I&CT DEPT.,

rashmi.naveen@manipal.edu

LECTURE- 7, OCT 22 (1.30PM TO 3.30PM), 2021

Evaluation of Arithmetic expressions

- The Representation and evaluation of expression is of great interest to computer scientists.
- An expression will have operators with different precedence and different associativity

20

- Ex: 8-3*4 = ? -4
- Expression needs to be evaluated based on the precedence of operators
- 3 types of expressions:
  1. Infix Expression : A+B
  2. Postfix Expression : AB+
  3. Prefix expression : +AB

Infix to Postfix expression conversion:examples

- *, /, % -> same precedence, left to right associativity---level1
- +, - -> same precedence , , left to right associativity ---level 2

Postfix (operand1 operand2 opr) Prefix

① 2+3-4 ⇒ 23+4-

23+ -4

② 2+3*4 ⇒ 234*+

2 + 34*

operand1 operand2

③ 2*3+4 ✓ 23*4+

23* + 4

Infix to Postfix expression conversion: examples

Ex4: a*b+c/d

a*b+c/d

ab* + c/d

opnd1 operand2

ab* + cd/

= ab*cd/+

Infix to Postfix expression conversion: examples

Ex5: a*(b+c)/d

$a * bc+ / d$

$= abc+*d/$

Ex6: (a*b)+(c/d)

$ab* + cd/$

$= ab*cd/+$

Infix to Postfix expression conversion: examples

Ex7: (a+b)*c/d-e

ab+ *c/d-e

ab+ c*/d-e

= ab+ c*d/e-

Ex8: (a+b)*(c-d)/((e-f)*(g+h))

= ab+ * cd- /(ef- * gh+)

ab+cd-* / ef-gh+*

= ab+cd- *ef-gh+*/

Infix to Postfix conversion: examples

a+b*c-d/e*f

Postfix=? abc*+de/f*-

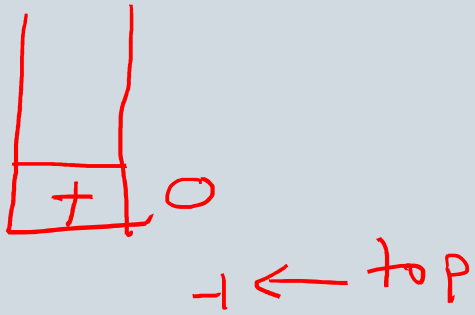
a/b%c+d*e-f*g = ab/c%+de*-fg*

Postfix=?

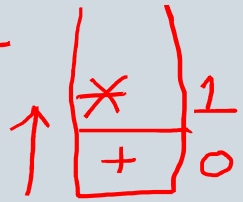
ab/c%de*+fg*-

$\boxed{a + b * c} \Rightarrow abc*+$

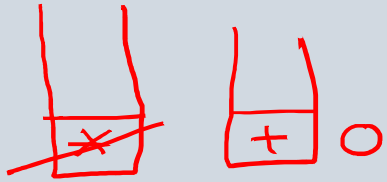
$a + b \Rightarrow$ postfix[] = ab+



$a + b * c \Rightarrow$ postfix[] = abc*+



$a * b + c \Rightarrow$ postfix[] = ab*c+



Infix to Postfix conversion using stack:

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	(a+b)*(c-d)/((e-f)*(g+h))

	Token	Stack content [0] [1]	Top	Postfix
1	a	Empty	-1	a
2	-	-	0	a
3	b	-	0	ab
4	+	+	0	ab- pop '-' push '+'
5	c	+	0	ab-c
6	*	+ *	1	ab-c
7	d	+ *	1	ab-cd
8	'\0'			ab-cd*+

a-b+c*d

~~a+b*c~~

Infix to Postfix conversion using stack

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	(a+b)*(c-d)/((e-f)*(g+h))

Token	Stack content [0] [1]	Top	Postfix
a	Empty	-1	a
+	+	0	a
b	+	0	ab
*	+ *	1	ab
c	+ *	1	abc
—	—	0	abc*+
d	—	0	abc*+d
'\0'	empty	-1	abc*+d-

a+b*c-d

abc*+d-

Infix to Postfix conversion using stack

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	(a+b)*(c-d)/((e-f)*(g+h))

Token	Stack content [0] [1] [2]	Top	
a	Empty	-1	a
*	*	0	a
(	* (	1	a
b	* (	1	ab
+	* (+	2	ab
c	* (+	2	abc
)	*	0	abc+
/	/	0	abc+*
d	/	0	abc+*d
	Empty	-1	abc+*d/

icp ← '\0'

isp ←

Infix to Postfix conversion using stack

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	(a+b)*(c-d)/((e-f)*(g+h))

Token	Stack content [ ]	Top	
(	(	0	
a	(	0	a
*	(*	1	a
(	(* (	2	a
b	(* (	2	ab
+	(* (+	3	ab
c	(* $\boxed{(+}$	3	abc
)	(*	1	abc+
)	Empty	-1	abc+*
/	/	0	abc+*
d	/	0	abc+*d
'\0'	Empty	-1	abc+*d/

Infix to Postfix conversion using stack

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	**(a+b)*(c-d)/((e-f)*(g+h))**

icp

isp

Token	Stack content [ ]	Top	Postfix
(	(	0	
a	(	0	a
+	(+	1	a
b	(+	1	ab
)	Empty	-1	ab+
*	*	0	ab+
(	* (	1	ab+
c	* (	1	ab+c
-	* (-	2	ab+c
d	* (-	2	ab+cd
)	*	0	ab+cd-
/	/	0	ab+cd-*
(	/ (	1	ab+cd-*
e	/ ((	2	ab+cd-*
e	,,	,,	ab+cd-*e

Infix to Postfix conversion using stack

Sl. No.	Infix
1	a+b*c-d
2	a*b+c/d
3	a*(b+c)/d
4	(a*(b+c))/d
5	a/b^c+d*e-f*g
6	(a+b)*c/d-e
7	(a+b)*(c-d)/(e+f)
8	**(a+b)*(c-d)/((e-f)*(g+h))**

Token	Stack content [ ]	Top	Postfix
e	/ ((	2	ab+cd-*e
-	/ ((-	3	"
f	"	3	ab+cd-*ef
)	/ (	1	ab+cd-*ef-
*	/ (*	2	"
(	/ (* (	3	"
g	"	3	ab+cd-*ef-g
+	/ (* (+	4	"
h	"	4	ab+cd-*ef-gh
)	/ (*	2	ab+cd-*ef-gh+
)	/	0	ab+cd-*ef-gh+*
'\0'	Empty	-1	ab+cd-*ef-gh+*/

Algorithm to convert infix expression to postfix

1. Scan the infix string from left to right. Assume the operand is single digit .

2. If the scanned character is an operand, then copy to postfix string

else

// If the scanned character is an operator, then

if it is a right parenthesis then

- POP till you find first left parenthesis
- //…

- else if the precedence of the scanned operator is > then the precedence of the operator on top of the stack, then PUSH the scanned operator
- Else
- POP until you get an operator with precedence less than the scanned operator
- precedence

- Repeat the steps 1 and 2 till you get a NULL character in the infix string
- POP all the operators to the postfix string

Infix to postfix expression conversion program

```
                0       1       2     3      4      5
enum precedence{lparen, rparen, plus, minus, times, divide,
                mod, eos, operand};
```

icp[] = {20, 19, 12, 12, 13, 13, 13, 0};

isp[] = {0, 19, 12, 12, 13, 13, 13, 0}

// icp[plus] = icp[2] = 12

precedence of '+'

```
precedence get_token(char c)
{
    switch(c)
    {
      case '(' : return lparen;
      case ')' : return rparen;
      case '+' : return plus;
```

icp: incoming precedence

isp: in stack precedence

Infix to postfix expression conversion program

```
  case '-' : return minus;
  case '*' :      "      times;
  case '/' :      "      divide;
  case '%' :      "      mod;
  case '#' : return cos;
  default  : return operand;
  }

}
```