infix to postfix conversion

Infix ⇒ A + B

operand, operator, operand

A
A + B
(A+B) + (C+D)

+ (A + B) → infix

(A+B) + (C+D) → infix
AB+ + CD+
AB+CD++ → postfix

Prefix ⇒ infix A + B → +AB (prefix)

operand, operator, operand

A
– AB → A-B
++AB-CD
(A+B)+(C-D)

A+B ⇒ +AB → Prefix
AB+ → postfix

(A+B) + (C-D) → infix
+AB + -CD
++AB-CD → prefix

Postfix ⇒ infix A + B → AB+ → postfix exp

operand, operator, operand

A
AB+
AB+CD-+

postfix expression

A → infix
A+B ⇒ AB+ → postfix
(A+B) + (C-D) → infixexp
AB+ + CD-
match
AB+CD-+

infix to prefix and postfix exp

1. (A+B) * C-D

Prefix ⇒ exp → (A+B) * C-D
→ (L→R)

step→1 → +AB * C–D
oper, op, oper, oper

step→2 ⇒ *+ABC – D

step→3 ⇒ –*+ABCD → prefix

* –

A+B
+AB

(A+B) - (C*D)
+AB – *CD
→ +AB *CD

L, op, R
A+B ⇒ AB+
input, output

Postfix ⇒ exp → infix (A+B) * C-D

exp → (A+B) * C –D
Step1 → AB+ * C – D
Step2 → AB+C* –D
Steps → AB+C*D–

(A+B) * (C-D) → infix
⇒ step1 ⇒ AB+ * CD–
Step2 ⇒ AB+CD-* → postfix

* /
L → R

(A+B) * C – D
+AB * C – D
*+ABC – D
–*+ABCD

A * B + C / D
*AB + C/D
*AB + /CD
+*AB/CD
a + b +ab
result → +*AB/CD

Input: A * B + C / D
infix exp

A * B + C/D
*AB + /CD
+*AB/CD
prefix

A*B+C/D → postfix
AB* + CD/
AB*CD/+ → postfix

Input: (A – B/C) * (A/K-L) → prefix and postfix

Prefix
exp → (A – B/C) * (A/K –L)
Step1 ⇒ (A – /BC) * (/AK –L)
Step2 ⇒ –A/BC * –/AKL
Step3 ⇒ *–A/BC–/AKL → prefix exp

Postfix ⇒
exp → (A – B/C) * (A/K –L)
Step1 ⇒ (A – BC/) * (AK/ –L)
Step2 → ABC/– * AK/L–
→ infix to postfix exp
Step3 → ABC/– AK/L–* → postfix exp

Rules for the conversion from infix to postfix expression

1. Print the operand as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator on to the stack.
3. If the incoming symbol is '(', push it on to the stack.
4. If the incoming symbol is ')', pop the stack and print the operators until the left parenthesis is found.
5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6. If the incoming symbol has lower precedence than the top of the stack, pop and print the top of the stack. Then test the incoming operator against the new top of the stack.
7. If the incoming operator has the same precedence with the top of the stack then use the associativity rules. If the associativity is from left to right then pop and print the top of the stack then push the incoming operator. If the associativity is from right to left then push the incoming operator.
8. At the end of the expression, pop and print all the operators of the stack.