# INFIX TO POSTFIX CONVERTER

**Algorithm of Code**

Create an empty stack and empty result string.

1. Scan the infix expression character by character.
2. If the scanned character is an operand, add it to result string.
3. If the scanned character is an operator;

   **3.1**. If the precedence of the scanned operator is greater than the precedence of the operator in the stack or the stack is empty or the stack contains a '(' , push it.

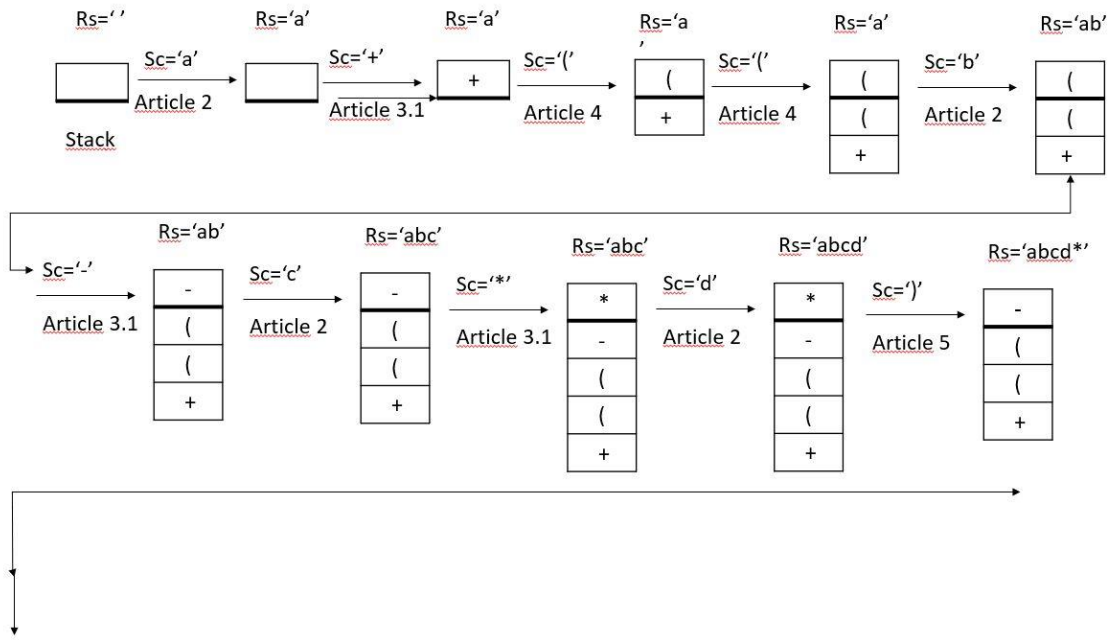
   **3.2.** Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator and add result string. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is '(', push it to the stack.
5. If the scanned character is ')', pop the stack and and pop it until a '(' is encountered, and discard both the parenthesis and add scanned characters to result string.
6. Repeat steps 2-6 until infix expression is scanned.
7. Pop from the stack and add result string until it is not empty

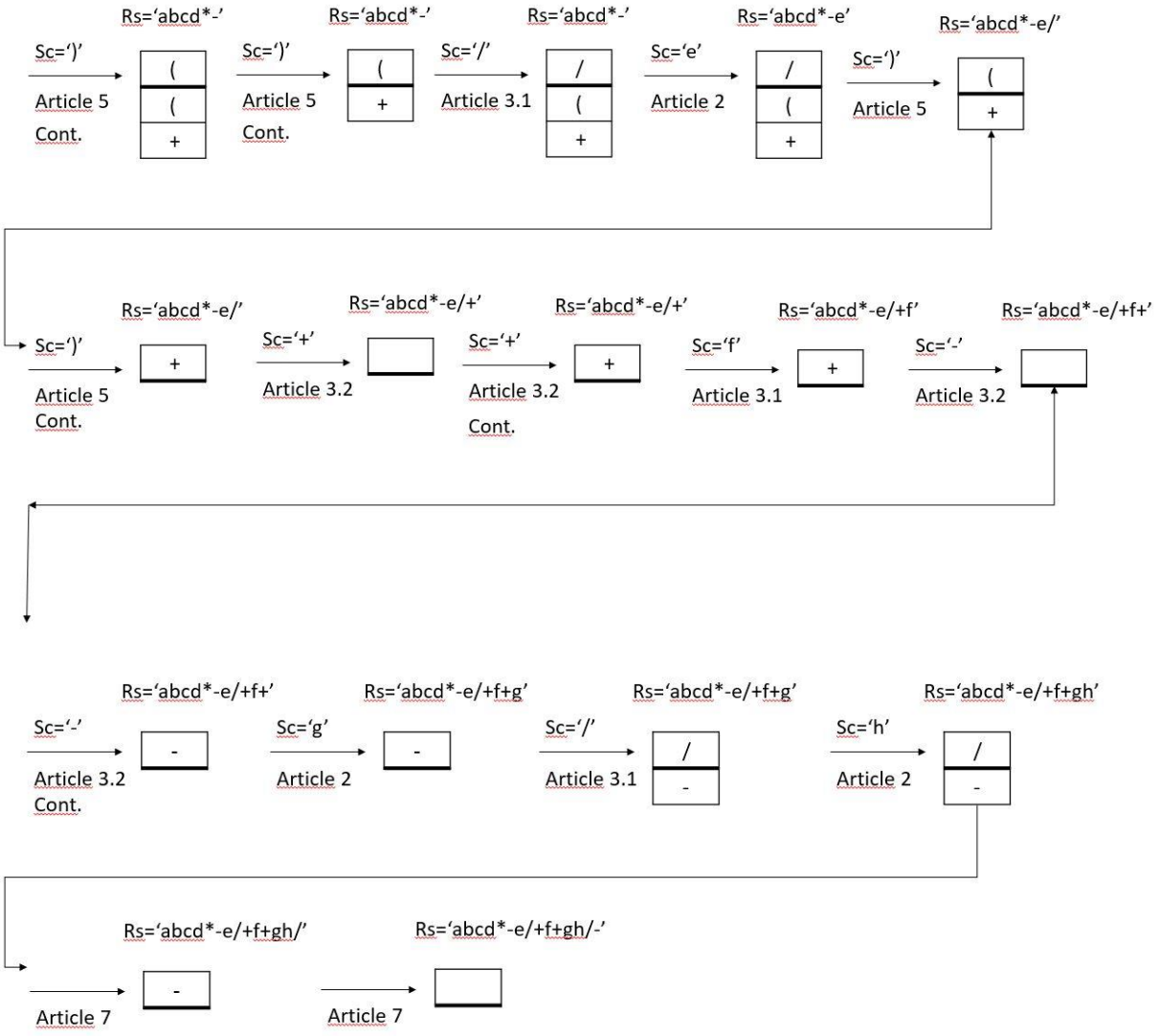
Rs=Result String

Sc = Scanned char from input

Input = " a + (( b – c * d ) / e ) + f – g / h"

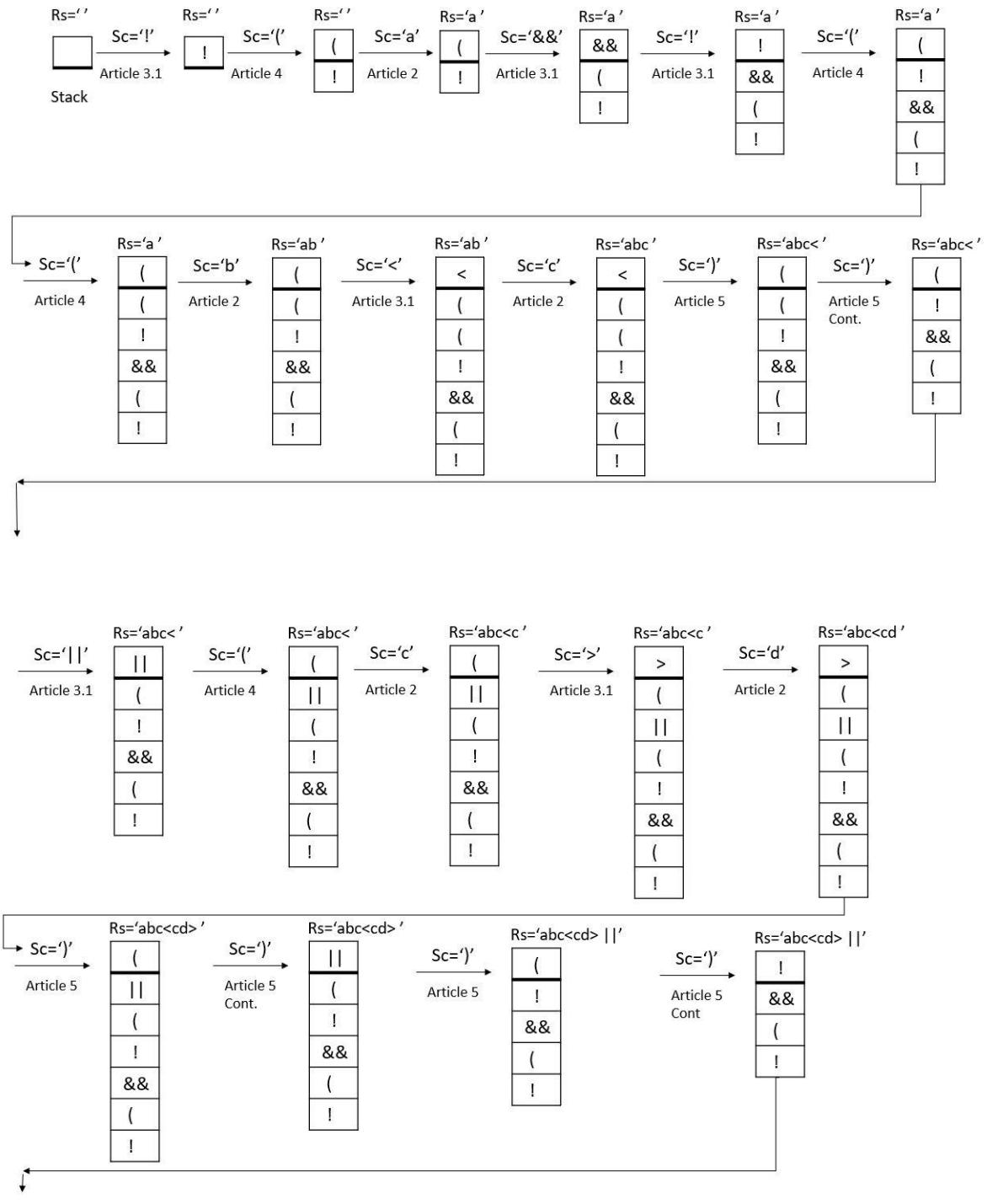
---

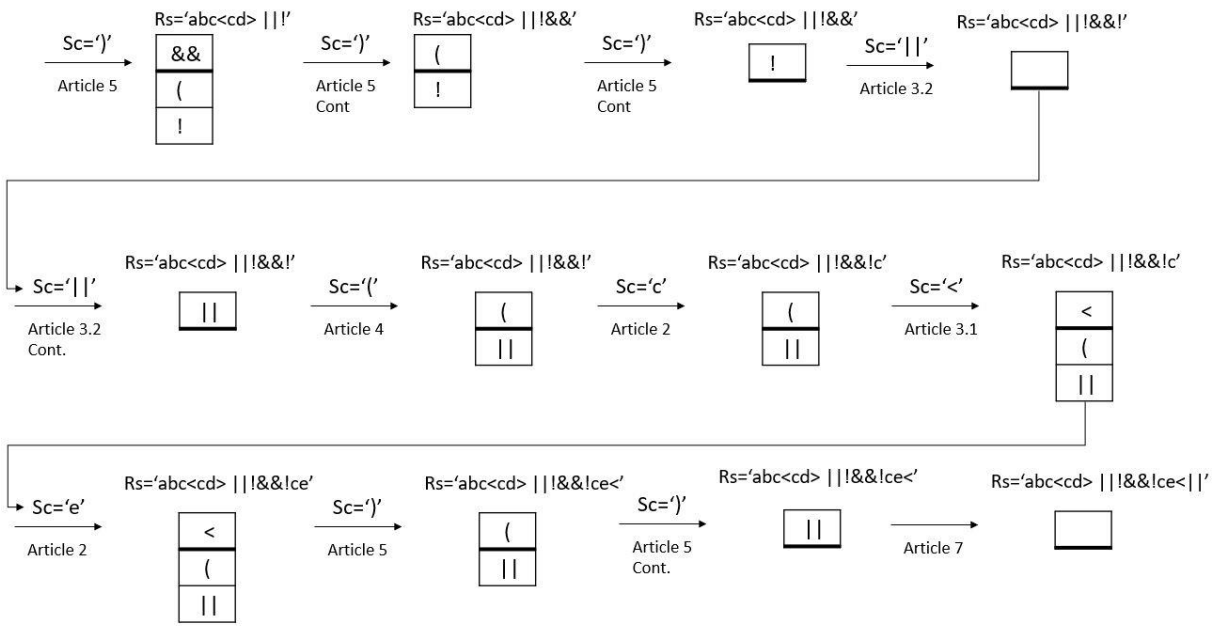





Result = " abcd*-e/+f+gh/- "

Input = “ ! ( a && ! (( b < c ) || ( c > d ))) || ( c < e ) ”






Result=" abc<cd> ||!&&!ce<|| "

# INFIX TO PREFIX CONVERTER

**Algorithm of Code**

Create an empty stack and empty result string.

1. Reverse infix expression.
2. Scan the infix expression character by character.
3. If the scanned character is an operand, add it to result string.
4. If the scanned character is an operator;

   **4.1** If the precedence of the scanned operator is greater than the precedence of the operator in the stack or the stack is empty or the stack contains a '(' , push it.

   **4.2** Else, pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator and add result string. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
5. If the scanned character is '(', push it to the stack
6. If the scanned character is ')', pop the stack and and pop it until a '(' is encountered, and discard both the parenthesis and add result string.

7. Repeat steps 2-6 until infix expression is scanned.
8. Pop from the stack and add result string until it is not empty
9. Reverse result string.

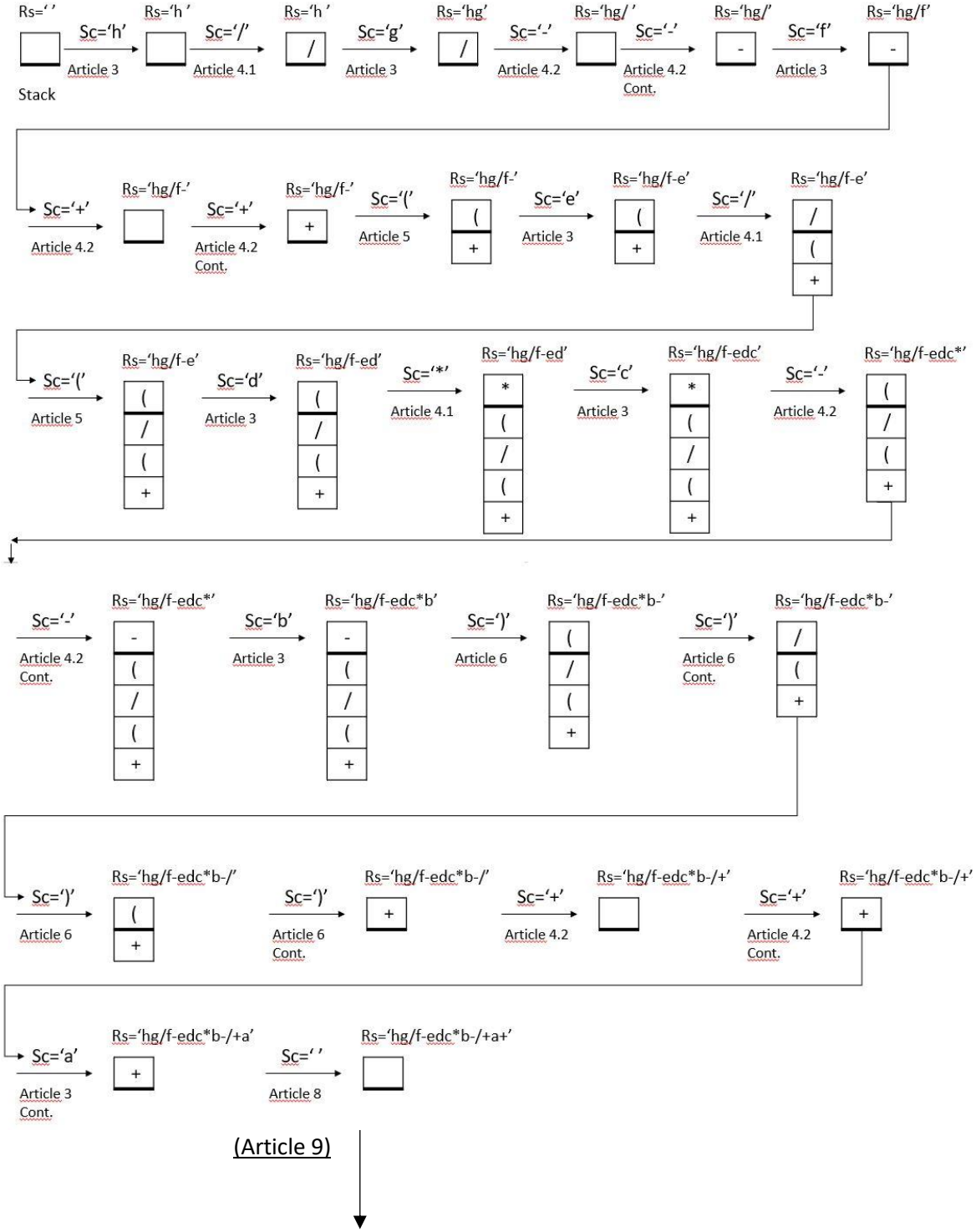
Rs=Result String

Sc = Scanned char from input

Input= " a + (( b - c * d ) / e ) + f - g / h"

(Article 1)

”h / g – f + ( e / ( d * c - b )) + a “

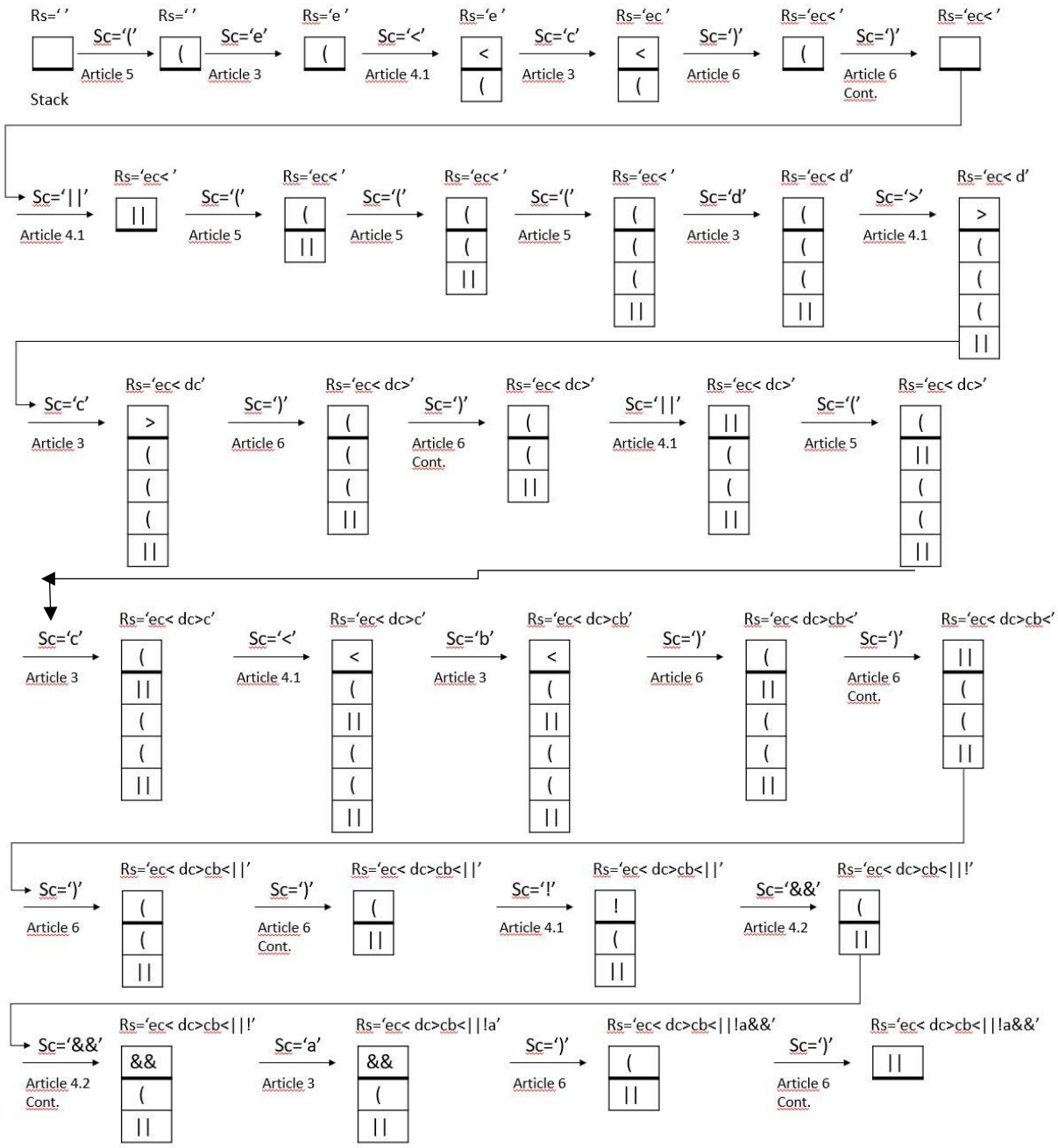


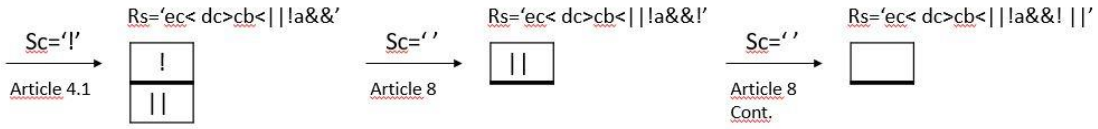

Result = “ +a+/-b*cde-f/gh ”

Input= “ ! ( a && ! (( b < c ) || ( c > d ))) || ( c < e )”

(Article 1)

”(e < c ) || ((( d > c ) || ( c < b )) ! && a ) !“






(Article 9)

Result =" |||!&&a!||<bc>cd<ce "

# POSTFIX EVALUATOR

**Algorithm of Code**

Create an empty stack

1. Scan the postfix expression left to right ,character by character.
2. If the scanned character is an operand, push it to the stack.
3. If the scanned character is an operator;

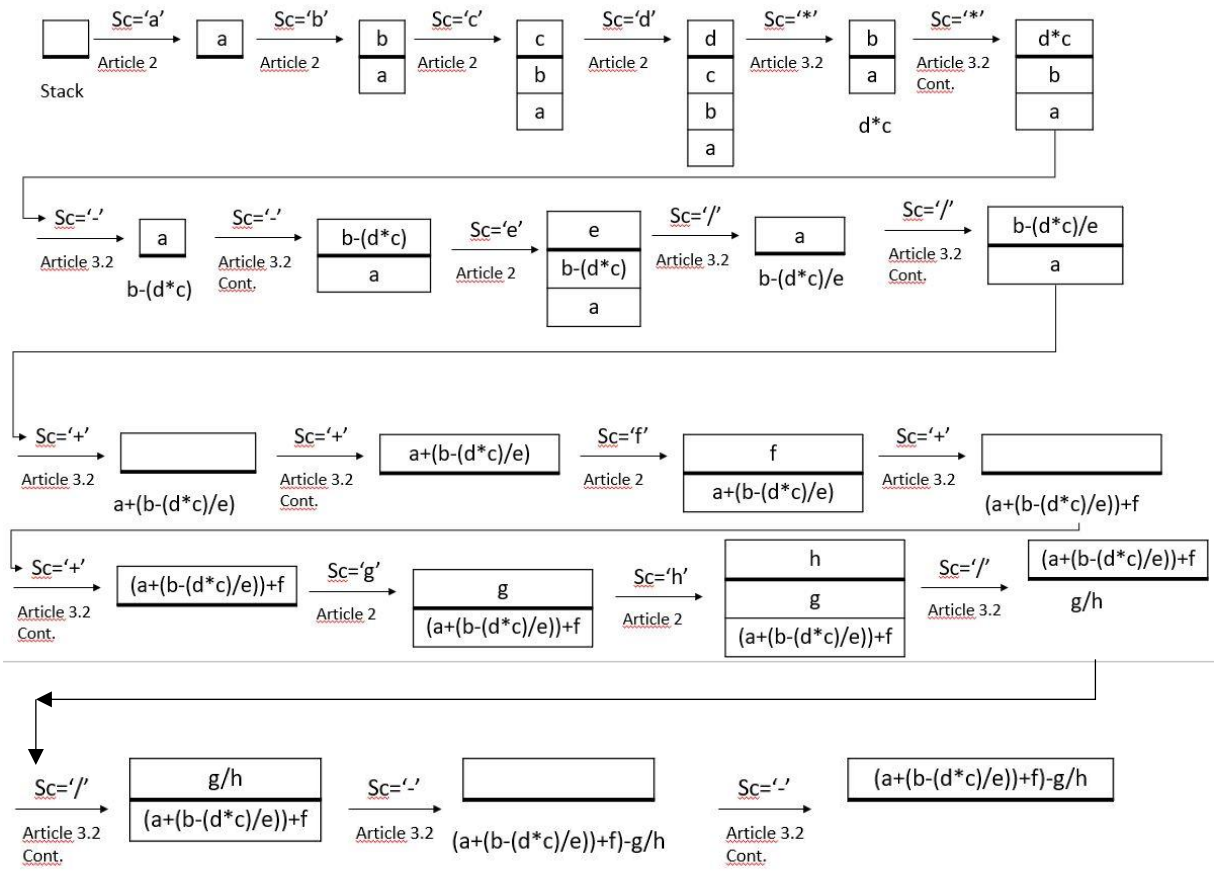
   **3.1** If operator is '!', pop one element from the stack and evaluate it according to the '!' operator and push the result to stack.

   **3.2.** Else pop two element from the stack and evaluate them according to the operator and push the result to stack.

4. Repeat steps 2 and 3 until postfix expression is scanned.
5. At the end of scan, pop result from the stack.

Sc = Scanned char from input

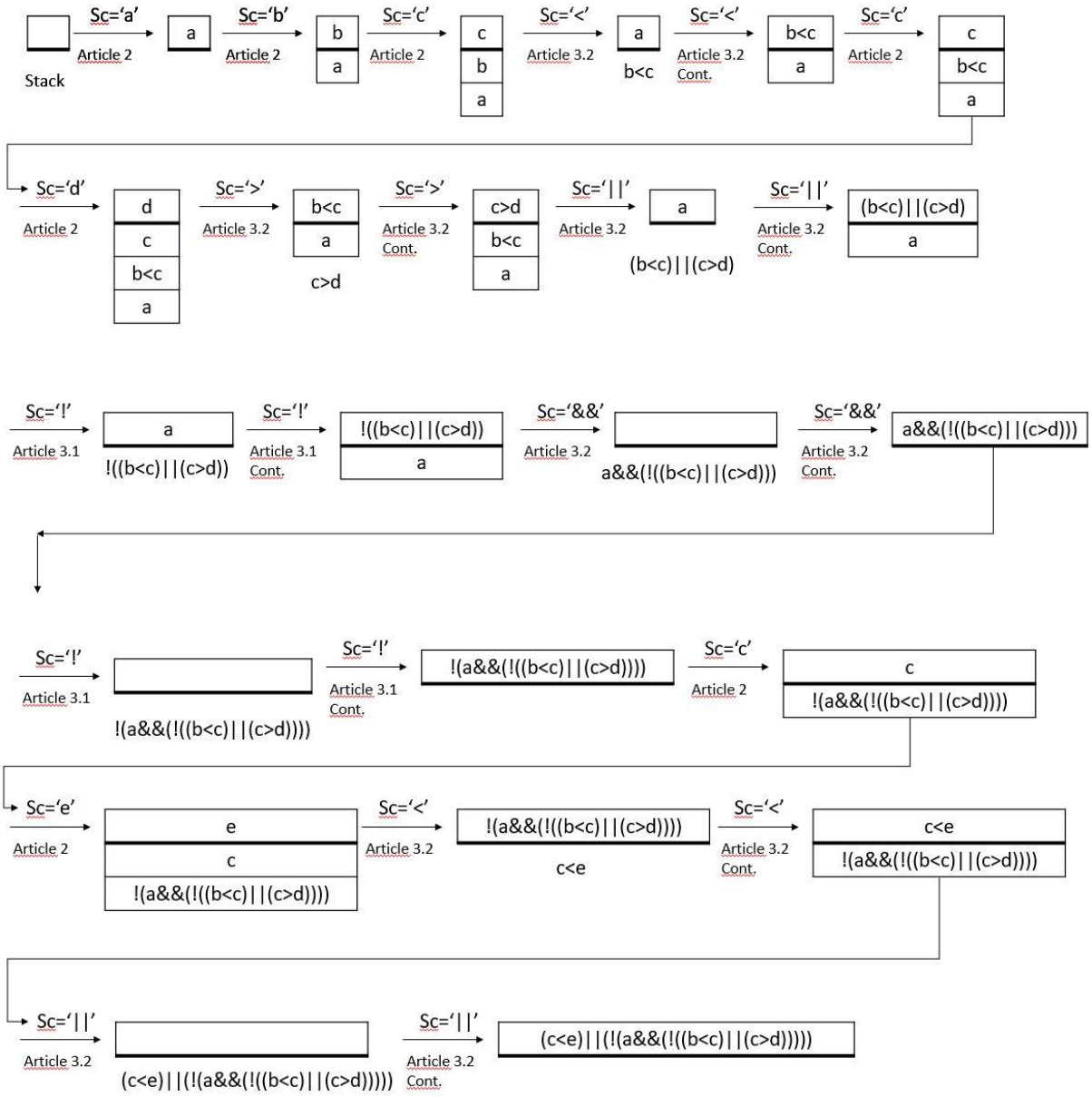
Input = “ abcd*-e/+f+gh/- ”




---

Result = “ (a+(b-(d*c)/e))+f)-g/h ”

Input=” abc<cd> |||&&!ce<|| ”




Result = “ (c<e)||(!(a&&(!((b<c)||(c>d))))) “

# PREFIX EVALUATOR

**Algorithm of Code**

Create an empty stack

1. Scan the postfix expression right to left ,character by character.
2. If the scanned character is an operand, push it to the stack.
3. If the scanned character is an operator;

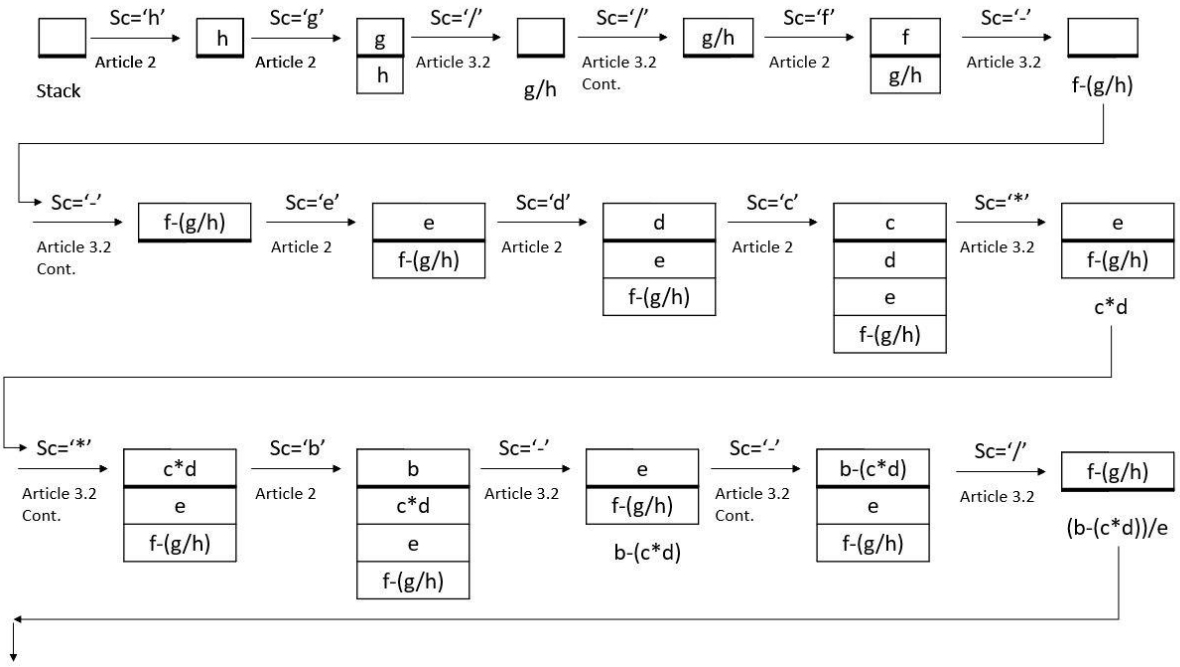
   **3.1** If operator is '!', pop one element from the stack and evaluate it according to the '!' operator and push the result to stack.

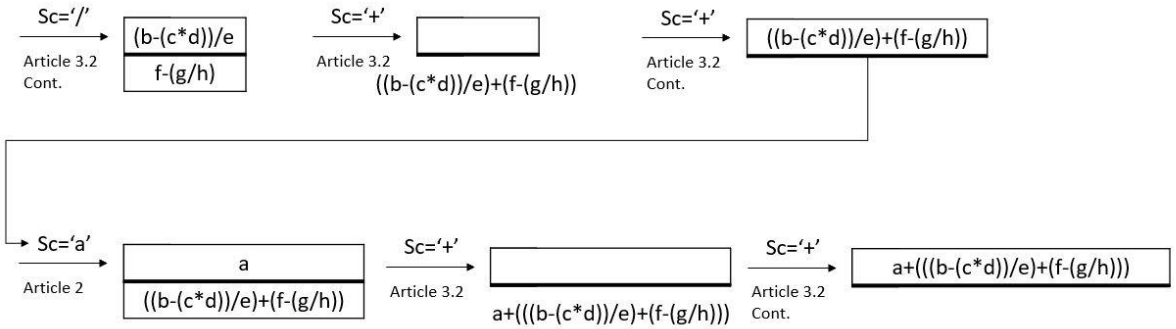
   **3.2.** Else pop two element from the stack and evaluate them according to the operator and push the result to stack.

4. Repeat steps 2 and 3 until postfix expression is scanned.
5. At the end of scan, pop result from the stack.

Sc = Scanned char from input

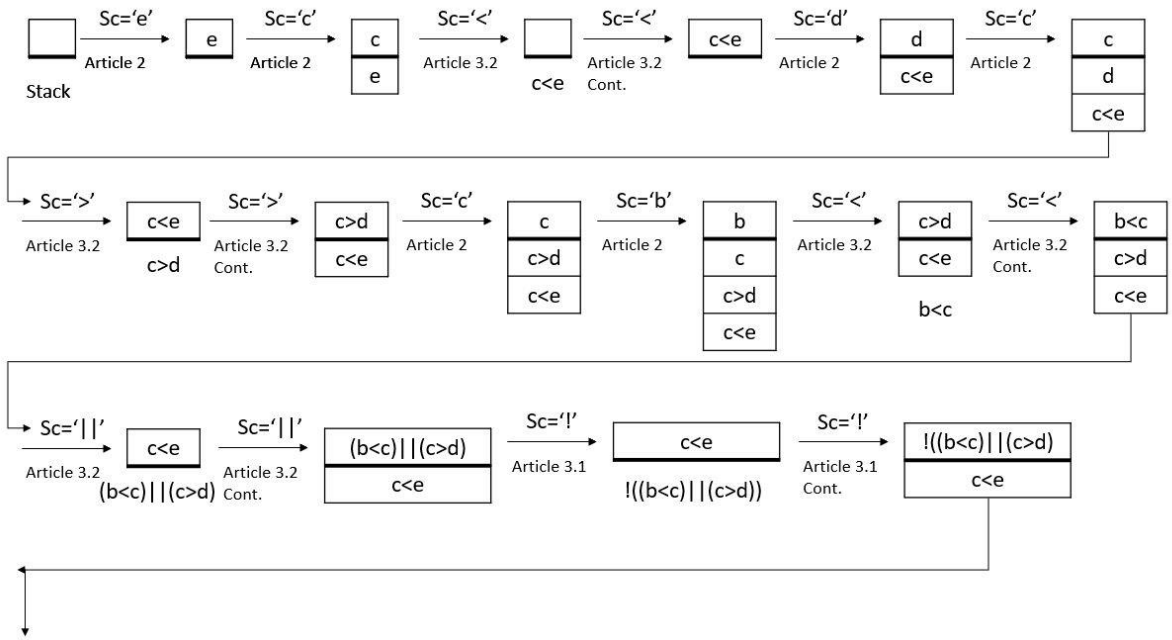
Input = "+a+/-b*cde-f/gh "

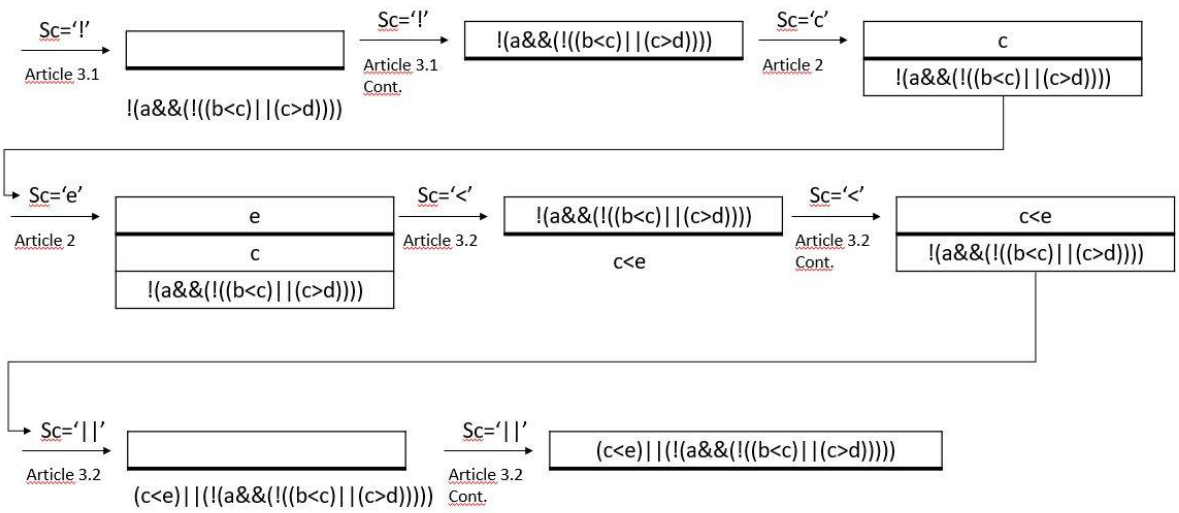





Result = " a+(((b-(c*d))/e)+(f-(g/h))) "

Input = " ||!&&a!||<bc>cd<ce "






Result = " (!(a&&(!((b<c)||(c>d))))||c<e "