Algorithm 1 in-fix to post-fix conversion

```
1: function shuntingYard(tokenList[])
2:     while tokenList.isEmpty() = FALSE do
3:         token ← tokenList.pop();
4:         if token.isNum() = TRUE then
5:             numStack.push(token.getVal())
6:         else if token.isVar() = TRUE then
7:             value ← getVar(token)
8:             numStack.push(value)
9:         else if token.isLeftParen() = TRUE then
10:            opStack.push(token.getVal())
11:        else if token.isRightParen() = TRUE then
12:            while opStack.peek() ≠ ' ( ' do
13:                op ← opStack.pop()
14:                num1 ← numStack.pop()
15:                num2 ← numStack.pop()
16:                result ← eval(op, num1, num2)
17:                numStack.push(result)
18:            end while
19:            opStack.pop()
20:        else if token.isOperator() = TRUE then
21:            while opStack.isEmpty() = FALSE and opPrec(opstack.peek(), token) = TRUE do
22:                op ← opStack.pop()
23:                num1 ← numStack.pop()
24:                num2 ← numStack.pop()
25:                result ← eval(op, num1, num2)
26:                numStack.push(result)
27:            end while
28:            opStack.push(token)
29:        end if
30:    end while
31:    while opStack.isEmpty() = FALSE do
32:        op ← opStack.pop()
33:        num1 ← numStack.pop()
34:        num2 ← numStack.pop()
35:        result ← eval(op, num1, num2)
36:        numStack.push(result)
37:    end while
38:    return numStack.pop()
39: end function
```

Algorithm 2 determine operator precedence

```
1: function opPrec(op1, op2)
2:     if op2 = ' ( ' or op2 = ' ) ' then
3:         return false
4:     else if (op1 = ' / ' or op1 = ' * '  then
5:     end if
6: end function
```

Algorithm 3 evaluate math

```
1: function eval(op, num1, num2)
2:     if  then
3:     else if  then
4:     end if
5: end function
```