

COS 301 Fall 2009
Homework #3 Due Thurs Nov 11

For problems 1-3, remember that postfix and prefix expressions can be generated by post-order and pre-order tree traversals. You should do these manually by creating parse trees – you might have to do it again on an exam.

1. (2 pts) Translate each of the following expressions to Polish (prefix) and Reverse Polish (postfix). Assume conventional priority and associativity of operators, where \wedge is exponentiation

- a. $x + y^z - d$
- b. $(a + b) / (c + d) * e$
- c. $x - j * c^{\wedge}(d - e) - 1$
- d. $((a - c) + (d - e)) / (d - c * a) + z$

2. (2 pts) Translate the following expressions from prefix to infix notation

- a. $+ y t$
- b. $+ - z x / z c$
- c. $/ - x y + a b$
- d. $* - j i + / c b a$

3. (2 pts) Translate each of the following expressions from postfix to infix notation

- a. $a b * w -$
- b. $w x y z - + *$
- c. $x y + a b - x * /$
- d. $a b c + - x y z / * +$

4. (4 pts) The original algorithm for converting infix to postfix expressions was designed by Edsger Dijkstra and is known as Dijkstra's Shunting Yard Algorithm. The top two Google references for a search on Shunting Yard Algorithm are

http://en.wikipedia.org/wiki/Shunting-yard_algorithm

and http://en.literateprograms.org/Shunting_yard_algorithm_%28C%29

The wikipedia article has a link to the original paper where the algorithm was described.

For this assignment, locate source code that implements the Shunting Yard Algorithm in two languages (your project language and your comparison language, typically Java), and run and test the code using the expressions from problem 1 above. You may have to substitute numeric values for variables and you may have to use another operator for exponentiation.

Because the algorithm is only used in the context of an expression parser, you will likely need some other code to actually get it run and show some output. Implementations in languages are typically embedded in larger driver projects so you may not have to do any actual programming – you just need to read and understand the code.

Submit (1) source code that you found with a link to the source code; and (2) sample output

5. Extra Credit (2 pts). Describe precisely how the code would have to be modified for APL, where all operators are right-associative and of equal precedence.