

# Expression Tree Code Demo (cont.)

Cpt S 321

Washington State University

# Let's see some solutions

What we did last time:

1. Extract classes into separate files
2. Isolated the hardcoded operators to one method only (TBC!)

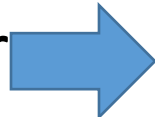
# Solutions we identified

- Throw more descriptive exceptions
- ~~• 2. Isolated the hardcoded operators~~
- Allow support for new operators without needing to change the logic in every method
- ~~• 1. Extract classes into separate files~~
- Consider operator precedence/associativity
- Parse the expression string and build the expression tree more elegantly
- Get rid of the redundant code

Where next?

# Pointers for solutions ExpressionTreeCodeDemo

- **Parse the expression string and build the expression tree more elegantly**

1. Transform the expression into a postfix order  [Dijkstra's Shunting Yard Algorithm](#)
2. [Construct the tree using the postfix order expression and a stack](#)

# Dijkstra's Shunting Yard Algorithm

1. If the incoming symbols is an operand, output it..
2. If the incoming symbol is a left parenthesis, push it on the stack.
3. If the incoming symbol is a right parenthesis: discard the right parenthesis, pop and print the stack symbols until you see a left parenthesis. Pop the left parenthesis and discard it.
4. If the incoming symbol is an operator and the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

# Dijkstra's Shunting Yard Algorithm

5. If the incoming symbol is an operator and has either **higher precedence** than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is **right associative** -- push it on the stack.
6. If the incoming symbol is an operator and has either **lower precedence** than the operator on the top of the stack, or has the same precedence as the operator on the top of the stack and is **left associative** -- continue to pop the stack until this is not true. Then, push the incoming operator.
7. At the end of the expression, pop and print all operators on the stack.  
(No parentheses should remain.)

# Construct the Expression Tree

- Read the postfix expression one symbol at the time
- If the symbol is an operand then create a tree with it and push it to the stack
- If the symbol is an operator then pop the last two trees from the stack and create a new tree with the operator as the root, last element of the stack as right subtree, and the one before last element of the stack as left subtree. Push this newly created tree to the stack.
- Stop when no more symbols are left. The result is the only element of the stack.