Expressions

Lecture Question

Task: Evaluate an expression tree

- In an object named datastructures.ExpressionTree write a method named evaluateTree that takes the root of an expression tree (BinaryTreeNode[String]) as a parameter and returns the evaluation of the tree as a Double
- The operators can be *, /, +, and -

* This question will be open until midnight

Infix Expressions

(12-4) - (8+9/3)

- The standard way to write an expression
- Operators placed **between** two operands
- Order of operations must be considered
- Parentheses used to override order of operations

Evaluating Infix Expressions

- PEMDAS
  - Parentheses -> Exponentiation -> Multiplication/Division -> Addition/Subtraction

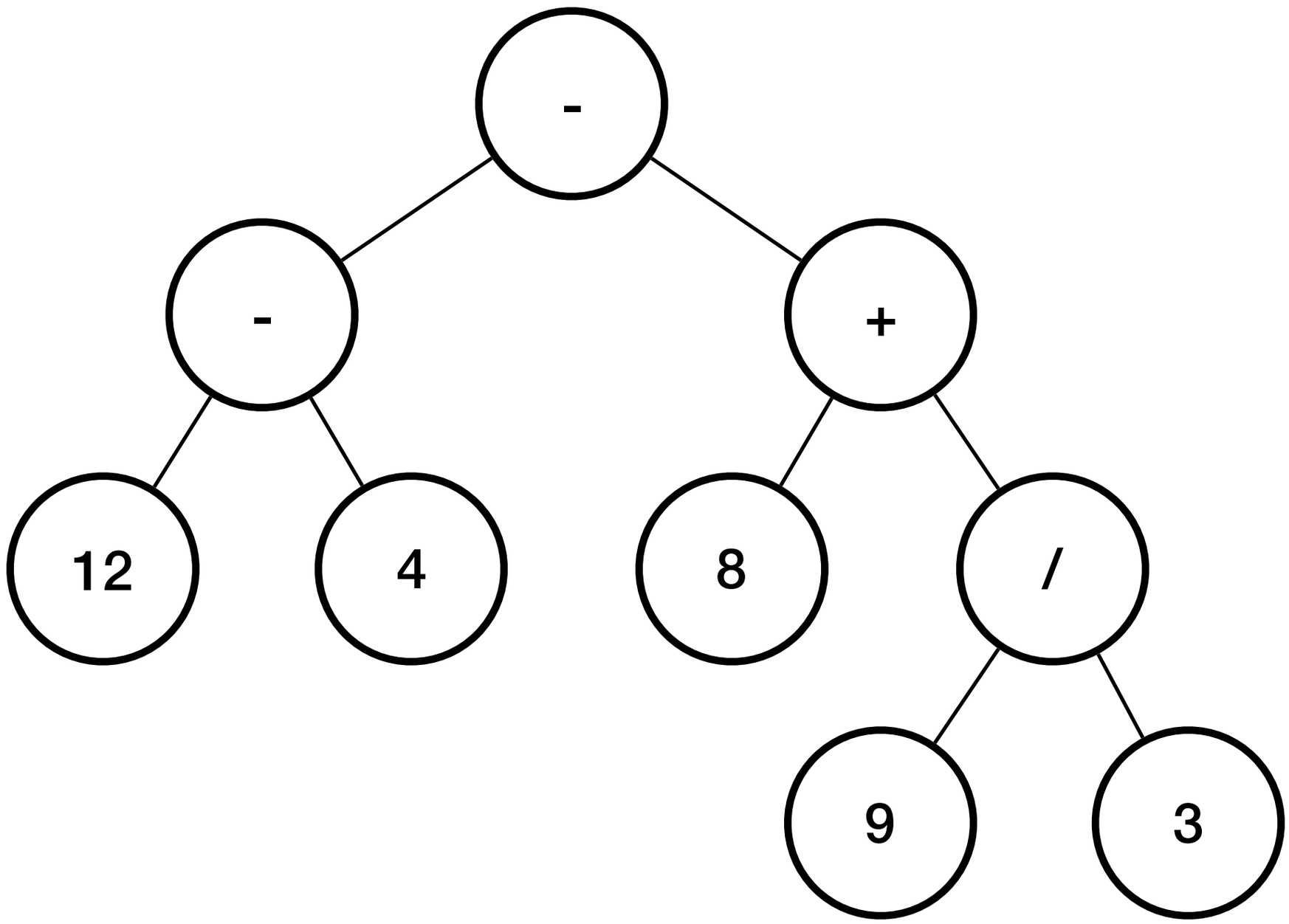
- (**12-4**) - (8+9/3)
- 8 - (8+**9/3**)
- 8 - (**8+3**)
- **8 - 11**
- **-3**

Expression Trees

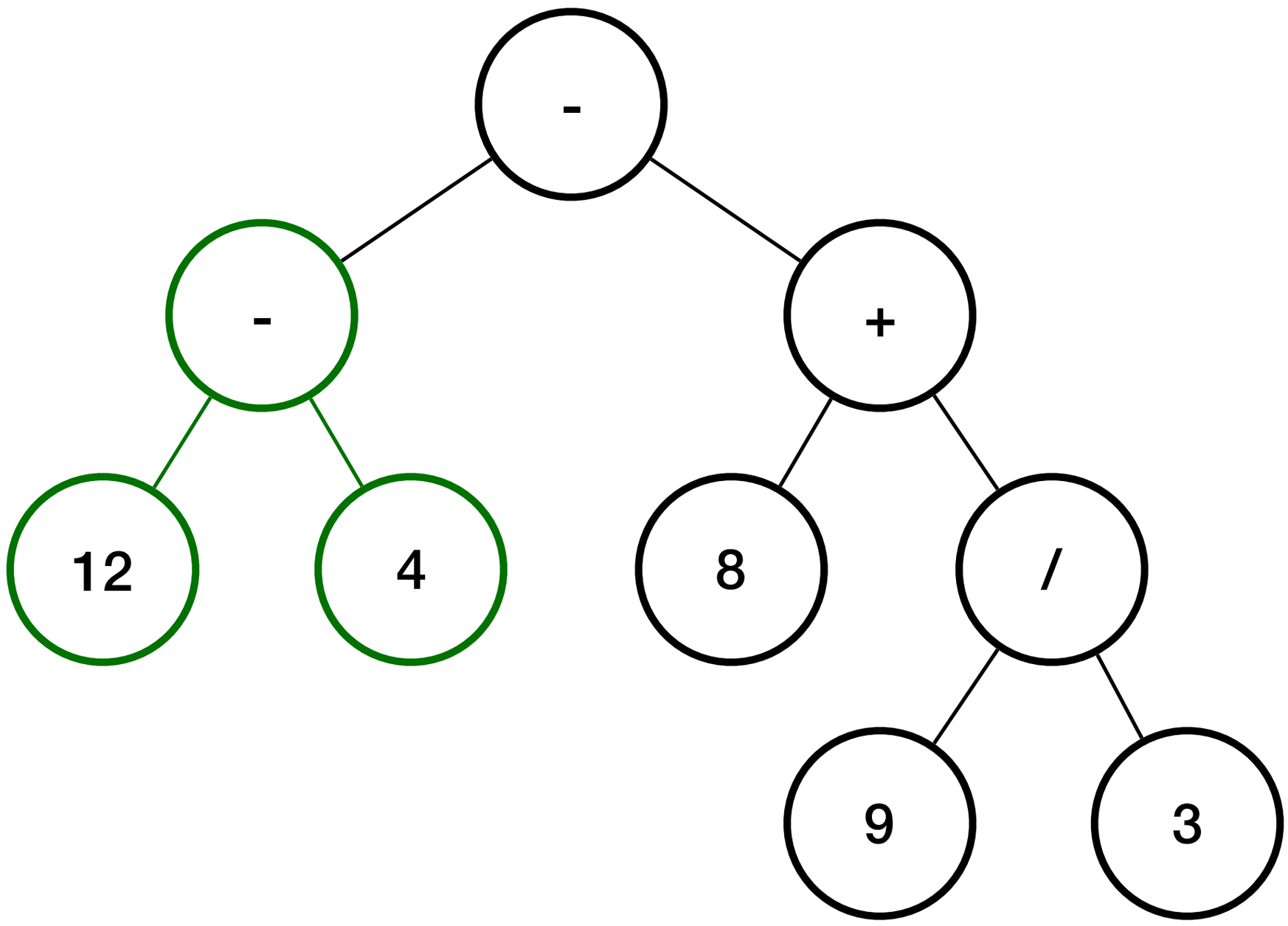
- Represent an expression as a binary tree
- Nodes can be
  - Operands
  - Operators
- An operand is a literal value
- An operator is evaluated by using its left and right children as operands
  - Operands can be operators

Expression Tree

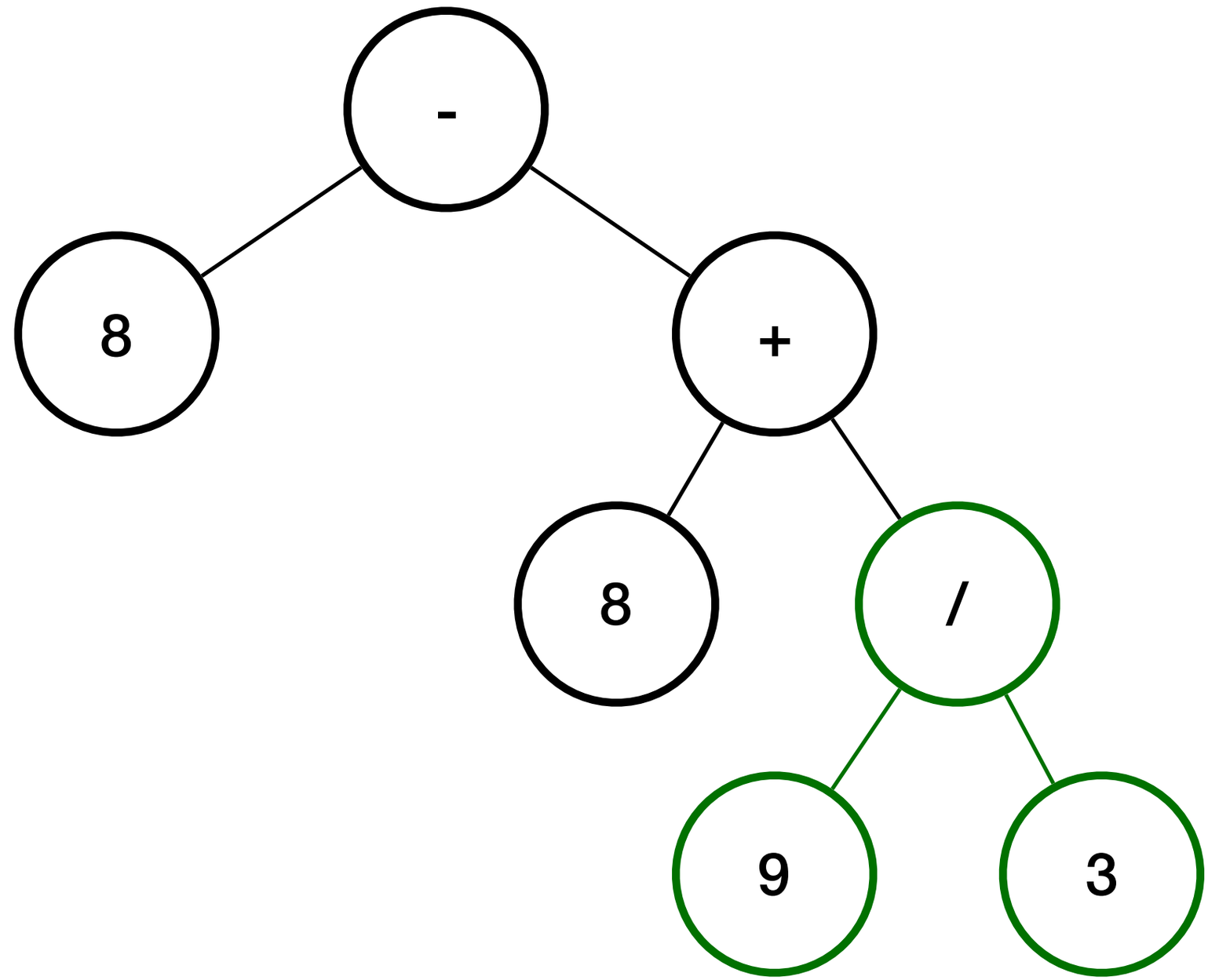
- (12-4) - (8+9/3) as an expression tree



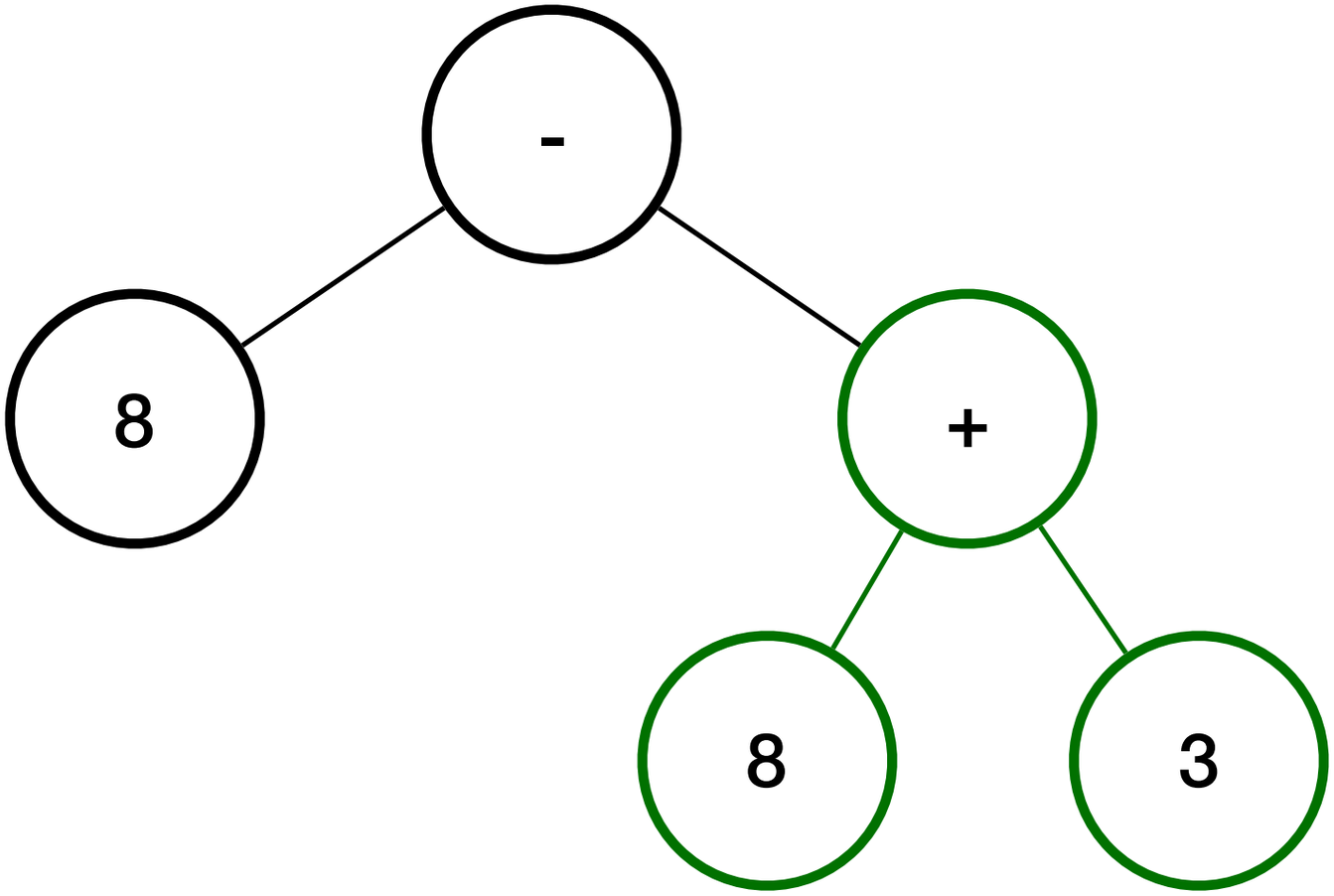
Evaluating Expression Tree



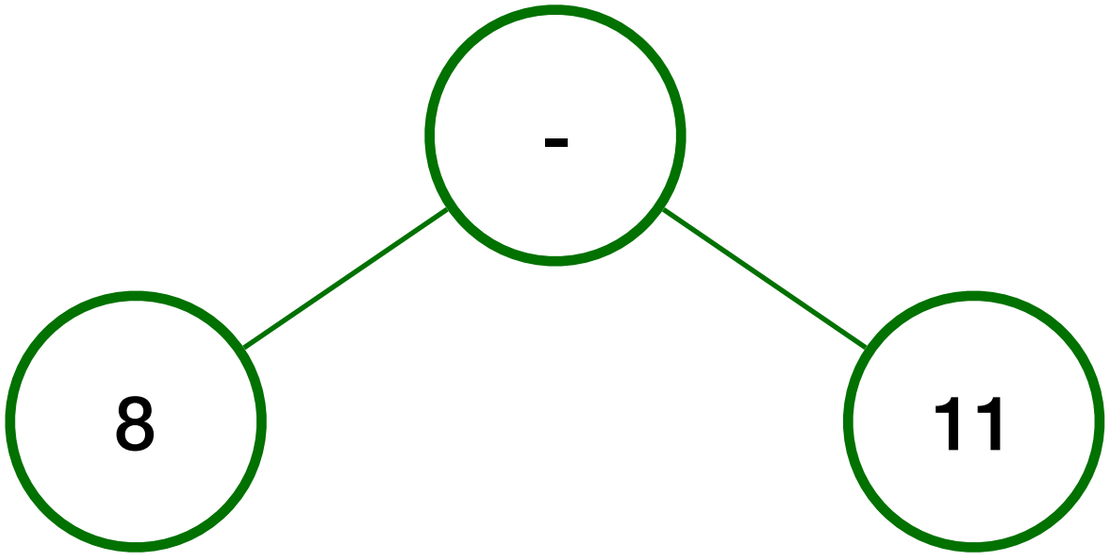
Evaluating Expression Tree



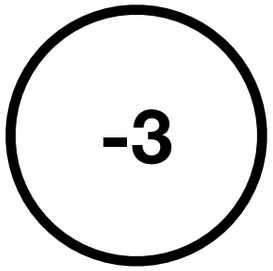
Evaluating Expression Tree



Evaluating Expression Tree



Evaluating Expression Tree



Expression Tree Traversals

- Modified in-order traversal that adds parentheses around each operator
- Generates a fully parenthesized infix expression
- ((12-4)-(8+(9/3)))

```
def fullyParenthesizedInOrderTraversal[A](node: BinaryTreeNode[A], f: A => Unit): Unit = {
  if (node != null) {
    val operator = List("^", "*", "/", "+", "-").contains(node.value)
    if (operator) {
      print("(")
    }
    fullyParenthesizedInOrderTraversal(node.left, f)
    f(node.value)
    fullyParenthesizedInOrderTraversal(node.right, f)
    if (operator) {
      print(")")
    }
  }
}
```

Expression Tree Traversals

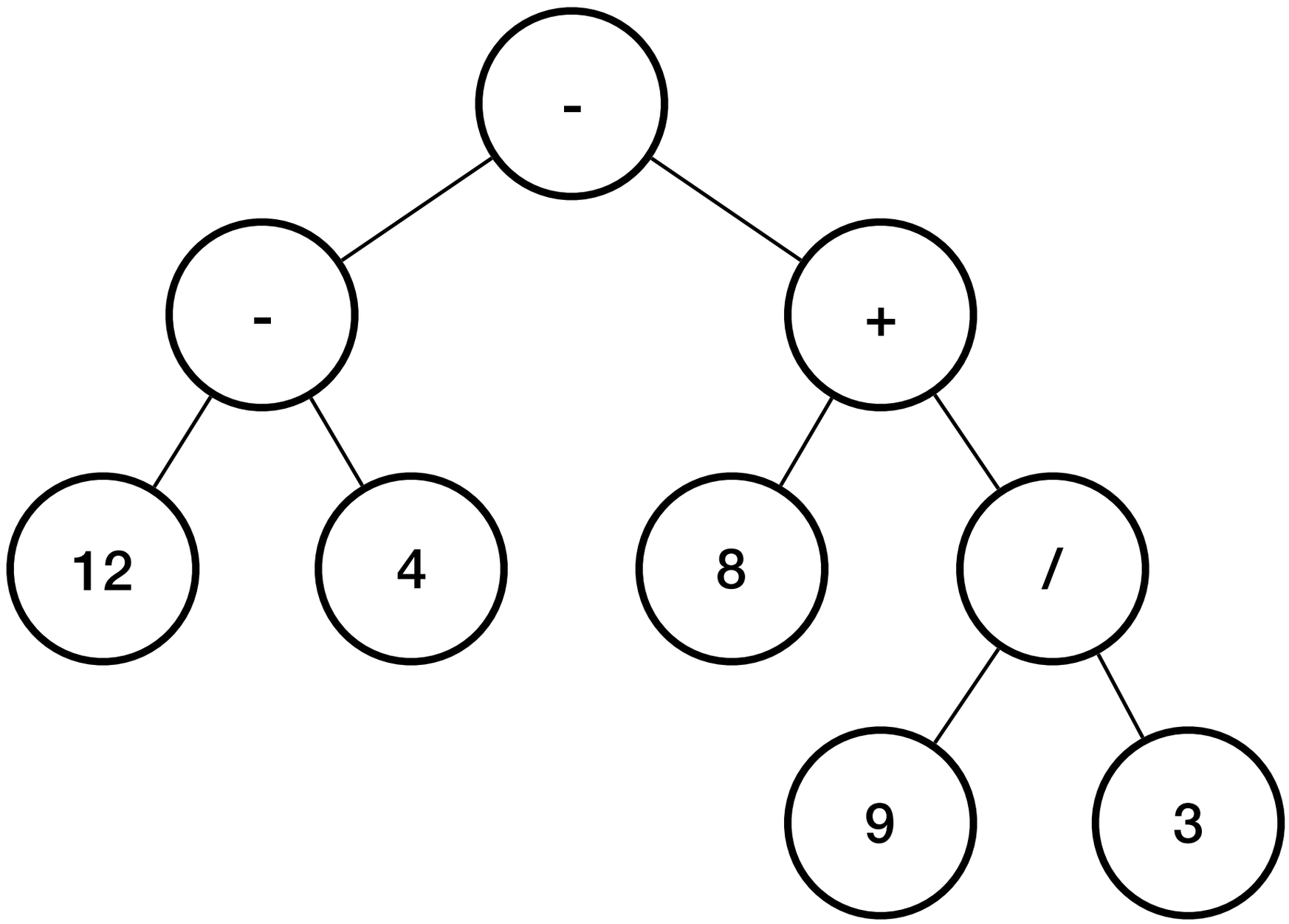
- Unmodified post-order traversal generates a postfix express (Reverse Polish Notation)

- 12 4 - 8 9 3 / + -

```
postOrderTraversal(root, (token: String) => print(token + " "))
```

Expression Tree Traversals

- 12 4 - 8 9 3 / + -



Postfix Expressions

- 12 4 - 8 9 3 / + -

- Advantages:
  - No parentheses needed
  - No order of operations to consider
  - Easy for computers to read
- Disadvantages
  - Hard for humans to read

Evaluating Postfix Expressions

- Find the first operator and evaluate it using the previous 2 operands
  - Repeat until there are no operators

- **12 4 -** 8 9 3 / + -
- 8 8 **9 3 /** + -
- 8 **8 3 +** -
- **8 11 -**
- **-3**

Infix -> Postfix

- Shunting Yard
  - Convert infix to postfix
- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty
- After reading the entire input, copy the rest of the stack to the output

(12-4) - (8+9/3) → 12 4 - 8 9 3 / + -

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

Input

(12-4) - (8+9/3)

Stack

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

Input

12-4) - (8+9/3)

Stack

(

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12

Input

-4) - (8+9/3)

Stack

(

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

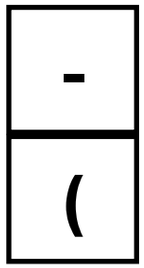
Output

12

Input

4) - (8+9/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

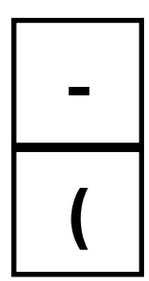
Output

12 4

Input

) - (8+9/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12 4 -

Input

) - (8+9/3)

Stack

(

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12 4 -

Input

- (8+9/3)

Stack

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12 4 -

Input

(8+9/3)

Stack

-

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

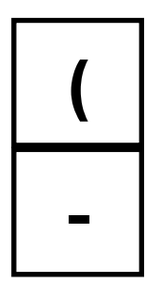
Output

12 4 -

Input

8+9/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

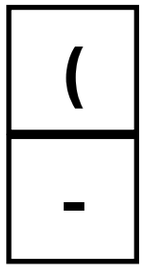
Output

12 4 - 8

Input

+9/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

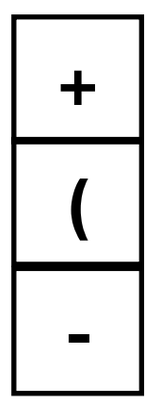
Output

12 4 - 8

Input

9/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

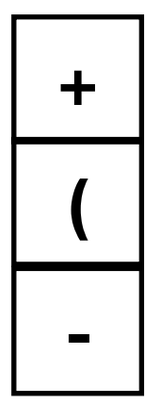
Output

12 4 - 8 9

Input

/3)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12 4 - 8 9

Input

3)

Stack

/
+
(
-

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

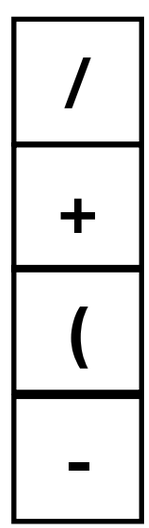
Output

12 4 - 8 9 3

Input

)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

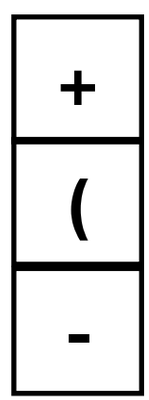
Output

12 4 - 8 9 3 /

Input

)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

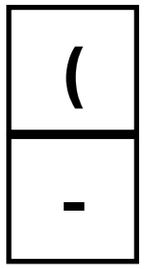
Output

12 4 - 8 9 3 / +

Input

)

Stack



Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output **Input**

12 4 - 8 9 3 / +

Stack

-

Infix -> Postfix

- Read expression left to right
- Copy operands to the output
- Push operators and parentheses on a stack
  - If reading), move top of stack to output until (is popped
  - If reading an operator, first move top of stack to output until a lower precedent operator is on top or the stack is empty

Output

12 4 - 8 9 3 / + -

Input

Stack

Lecture Question

Task: Evaluate an expression tree

- In an object named datastructures.ExpressionTree write a method named evaluateTree that takes the root of an expression tree (BinaryTreeNode[String]) as a parameter and returns the evaluation of the tree as a Double
- The operators can be *, /, +, and -

* This question will be open until midnight