# CS 340 Fall 2014

Assignment 2

Due Friday October 3

## Expression Tree

- Implement and test the ExpressionTree class shown on the following slides.
- The constructor and the methods buildPostfix and buildInfix can assume the expression passed as a parameter is syntactically correct.
- Tokens (operands and operators) are separated by blanks
- evaluate returns the integer value of the expression tree
- All calculations will be done with integer arithmetic
- Use the Java library class Stack in the implementation of buildPostfix and buildInfix

## Expression operators

- The operators in precedence order are
  - ! (unary minus)
  - ^ (exponentiation)
  - *, /
  - +.-
- Exponentiation and unary minus are right associative
- All other operators are left associative
- Precedence and associativity only matter for infix expressions
- Infix expressions can use parentheses to change the order of expression evaluation

# ExpressionTree.java

```
public class ExpressionTree {

    private class Node {
        Node left;
        String data;
        Node right;

        Node(Node l, String d, Node r) {
            left = l;
            data = d;
            right = r;
        }
    }

    public static final int INFIX = 1;
    public static final int POSTFIX = 2;

    Node root;
```

## ExpressionTree.java

```
public ExpressionTree(String exp, int format) {
//PRE: exp is a legal postfix expression && (format == INFIX || format == POSTFIX)
//Build an expression tree from the expression exp
        if (format == INFIX)
                buildInfix(exp);
        else
                buildPostfix(exp);
}

private void buildPostfix(String exp) {
//PRE: exp is a legal postfix expression
//Build an expressions tree from the postfix expression exp
}

private void buildInfix(String exp) {
//PRE: exp is a legal infix expression
//Build an expressions tree from the infix expression exp

}
```

# ExpressionTree.java

```
public int evaluate() {
//return the int value of the expression tree
        return evaluate(root);
}

private int evaluate(Node r) {
//return the int value of the expression tree with root r

}
```

# ExpressionTree.java

```
public String toPostfix() {
//return the postfix representation of the expression tree
        return toPostfix(root);
}

private String toPostfix(Node r) {
//return the postfix representation of the tree with root r

}

public String toInfix() {
//return the fully parenthesized infix representation of the expression tree

        return toInfix(root);
}

private String toInfix(Node r) {
//return the fully parenthesized infix representation of the tree with root r
}
```

# ExpressionTree.java

```
public static void main(String args[]) throws IOException {
        BufferedReader b1 = new BufferedReader(new FileReader(args[0]));
        ExpressionTree e;
        String exp = b1.readLine();
        while (!exp.equals("")) {
                e = new ExpressionTree(exp,ExpressionTree.POSTFIX);
                System.out.println("Infix format: " + e.toInfix());
                System.out.println("Postfix format: " + e.toPostfix());

                System.out.println("Expression value: "+e.evaluate());
                System.out.println();
                exp = b1.readLine();
        }
```

# ExpressionTree.java

```
        exp = b1.readLine();
        while (exp != null) {
                e = new ExpressionTree(exp,ExpressionTree.INFIX);
                System.out.println("Infix format: " + e.toInfix());
                System.out.println("Postfix format: " + e.toPostfix());

                System.out.println("Expression value: "+e.evaluate());
                System.out.println();
                exp = b1.readLine();
        }


    }
}
```

## Input File Format

- One or more lines where each line contains a syntactically correct postfix expression
- One blank line
- One or more lines where each line contains a syntactically correct infix expression
- Remember in expressions all tokens are separated by blanks

# Example Input File

```
2 3 +
2 3 + 5 3 - *
2 3 ^ 8 90 10 / * +

2 + 3
( 2 + 3 ) * ( 5 - 3 )
2 ^ 3 + 8 * ( 90 / 10 )
( ( 2 ^ 3 ) + ( 8 * ( 90 / 10 ) ) )
```

# Example Output

```
Infix format: ( 2 + 3 )
Postfix format: 2 3 +
Expression value: 5

Infix format: ( ( 2 + 3 ) * ( 5 - 3 ) )
Postfix format: 2 3 + 5 3 - *
Expression value: 10

Infix format: ( ( 2 ^ 3 ) + ( 8 * ( 90 / 10 ) ) )
Postfix format: 2 3 ^ 8 90 10 / * +
Expression value: 80

Infix format: ( 2 + 3 )
Postfix format: 2 3 +
Expression value: 5

Infix format: ( ( 2 + 3 ) * ( 5 - 3 ) )
Postfix format: 2 3 + 5 3 - *
Expression value: 10

Infix format: ( ( 2 ^ 3 ) + ( 8 * ( 90 / 10 ) ) )
Postfix format: 2 3 ^ 8 90 10 / * +
Expression value: 80

Infix format: ( ( 2 ^ 3 ) + ( 8 * ( 90 / 10 ) ) )
Postfix format: 2 3 ^ 8 90 10 / * +
Expression value: 80
```

## Assignment Submission

- You must email me the following files
  - ExpressionTree.java
    - Make sure the file includes a comment with you name at the top of the file
  - One test file (your program must work correctly with this test file)