```
/**
 * ExpressionTree is a binary expression tree supporting the operators
 * +, -, *, and /.  Only 1-digit operands are allowed.
 *
 * As a student at Union College, I am part of a community that values
intellectual effort, curiosity and discovery. I understand that in
order to truly claim my educational and academic achievements, I am
obligated to act with academic integrity. Therefore, I affirm that I
will carry out my academic endeavors with full academic honesty, and I
rely on my fellow students to do the same.
 *
 * @author Blair Hagen
 * @version 5/19/16
 */
public class ExpressionTree
{
    private TreeNode root;

    /**
     * Creates an expression tree out of the given prefix string
     */
    public ExpressionTree(String expression)
    {
        root = buildSubtree(new CharacterIterator(expression));
    }

    /**
     * Takes a prefix string (wrapped in a CharacterIterator)
     * and creates an expression tree out of the next operand it
     * sees (base case) or the next operator and two subsequent
     * operands it sees (recursive case).  It returns a pointer
     * to the newly-built subtree.
     *
     * @param input object that easily gets the next char in a String
     * @return a pointer to the root of the subtree it builds.
     */
    private TreeNode buildSubtree(CharacterIterator input)
    {
      TreeNode n = new TreeNode(input.getNext());
      if (n.value == '+' || n.value == '-' || n.value == '*' || n.value
== '/') {
           n.llink = buildSubtree(input);
           n.rlink = buildSubtree(input);
      }
     return n;
    }

    /**
     * Performs an inorder traversal, creating an infix version of the
     * expression.
     *
     * @param N the root of some subtree of an expression tree
     * @return the mathematical expression starting at
     * node N in infix notation (fully parenthesized)
     */
    private String infixString(TreeNode N)
    {
     String result = "";
        if (N != null) {
           if (!N.isLeaf()) {
                result += "(";
           }

           result += infixString(N.llink);
           result += N.toString();
           result += infixString(N.rlink);

           if (!N.isLeaf()) {
                result += ")";
           }
        }
        return result;
    }

    /**
     * Performs an inorder traversal, creating an infix version of the
     * expression.
     *
     * @return a String with the mathematical expression in infix
notation
     * (fully parenthesized)
     */
    public String infixString()
    {
        return infixString(root);
    }

    /**
```

```
     * Performs a preorder traversal, creating a prefix version of the
     * expression.
     *
     * @param N the root of of some subtree of an expression tree
     * @return the mathematical expression starting at node N in
prefix notation
     */
    private String prefixString(TreeNode N)
    {
     String result = "";
        if (N != null)
        {
           result += N.toString();
           result += prefixString(N.llink);
           result += prefixString(N.rlink);
        }
        return result;
    }

    /**
     * Performs a preorder traversal, creating a prefix version of the
     * expression.
     *
     * @return a String with the mathematical expression in prefix
notation
     */
    public String prefixString()
    {
        return prefixString(root);
    }
}
```

```
/**
 * Runs tests to be sure you created the expression tree correctly.
 *
 * @author Blair Hagen and Aaron Cass and Chris Fernandes
 * @version 5/18/16
 *
 */
```

```
public class ExpressionTests
{
    public static void main(String [] args)
    {
        Testing.startTests();
        Testing.setVerbose(true);

        testNoOperator();
        testOneOperator();
        testTwoOperators();
        testThreeOperators();
        testEightOperators();
        testRightTree();
        testLeftTree();

        Testing.finishTests();
    }

    private static void printResults(String input, String expected) {
     ExpressionTree e = new ExpressionTree(input);
        Testing.assertEquals(input + " was input, prefix string should 
match",
                             input,
                             e.prefixString());
        Testing.assertEquals(input + " was input, infix string should 
be " + expected,
                             expected,
                             e.infixString());
    }

    private static void testNoOperator()
    {
     Testing.testSection("Expressions with One Operand");

     String prefixInput = "8";
     String expectedInfix = "8";
     printResults(prefixInput, expectedInfix);
    }

    private static void testOneOperator()
    {
        Testing.testSection("Expressions with one operator");

        String prefixInput = "+56";
        String expectedInfix = "(5+6)";
        printResults(prefixInput, expectedInfix);

        prefixInput = "-23";
        expectedInfix = "(2-3)";
        printResults(prefixInput, expectedInfix);
    }

    private static void testTwoOperators()
    {
        Testing.testSection("Expressions with two operators");

        String prefixInput = "/-123";
        String expectedInfix = "((1-2)/3)";
        printResults(prefixInput, expectedInfix);

        prefixInput = "/1-23";
        expectedInfix = "(1/(2-3))";
        printResults(prefixInput, expectedInfix);
    }

    private static void testThreeOperators()
    {
      Testing.testSection("Expressions with three operators");

      String prefixInput = "*/235";
      String expectedInfix = "((2/3)*5)";
      printResults(prefixInput, expectedInfix);

      prefixInput = "/*235";
      expectedInfix = "((2*3)/5)";
      printResults(prefixInput, expectedInfix);
    }

    private static void testEightOperators()
    {
      Testing.testSection("Expressions with eight operators");

      String prefixInput = "-*32+/54/+-19*876";
      String expectedInfix = "((3*2)-((5/4)+(((1-9)+(8*7))/6)))";
      printResults(prefixInput, expectedInfix);
    }

    private static void testRightTree()
    {
     Testing.testSection("Make a left heavy tree");

     String prefixInput = "*9/8+7*6-54";
     String expectedInfix = "(9*(8/(7+(6*(5-4)))))";
     printResults(prefixInput, expectedInfix);
    }

    private static void testLeftTree()
    {
     Testing.testSection("Make a right heavy tree");

     String prefixInput = "*/+*-456789";
     String expectedInfix = "(((((4-5)*6)+7)/8)*9)";
     printResults(prefixInput, expectedInfix);
    }
}
```