Psuedo Code

```
public class PostfixAndInfix {
    Double A = 5.0, B = -1.0, C = 7.0, D = -4.0, E = 11.0, F = 9.0, G = 3.2, H = 6.325,
           I = 8.0, J = -3.333896;
    String infixExpression;
    String postfixExpression;
    Double postfixValue;

    /**
     * First constructor.
     */
    public PostfixAndInfix() { }

    /**
     * Second constructor.
     *
     * @param infixExpression
     */
    public PostfixAndInfix(String infixExpression) {
        this.infixExpression = infixExpression;
    }

    /**
     * Convert the infix expression into a postfix expression.
     *
     * @return True if the infix expression could be converted into a postfix
     *         expression, else false.
     */
    public Boolean infixConversion() {
        // Create edge cases for identifying:
        //      1) Parentheses not matching
        //          return false;
        //      2) Empty infix expressions
        //          return false;
        //      3) Invalid infix syntax
        //          return false;
        //      4) Skipping over blank spaces
        //          return false;


        /*
        while there are items in infix expression {
            if current item is an operand {
                write it to postfix expression
            } else if current item is a '(' {
                push it onto the stack
            } else if our current item is ')' {
                while the item on top of the stack is not a '(' {
                    pop stack and write the operator to the end of the postfix expression
                }
                pop the '(' and throw it away
            } else {  // handle arithmetic operator, but here we only care about the binary operators
                while stack is not empty AND the precedence of item on top of stack is > the current
item {
                    pop stack and write that operator to the end of postfix expression
                }
                push current item onto stack
            }
            advance to next item in infix expression
        }

        while stack is not empty {
            pop and write the operator to postfix expression
        }
        */

        return true;
    }

    /** Evaluate the postfix expressions. */
    public void postfixEvaluation() {
        // Check if the postfix expression has been created yet {
        //      throw new Exception
        // }

        /*
        while there are items in the postfix expression {
            if the current item is an operand {
                push item onto stack
            } else {
                pop stack and attach operand to right of the operator
                pop stack again and attach operand to left of the operator
                evaluate and push the results
            }
            advance to the next item in the postfix expression
        }

        if only one number left on stack {
            pop and this is the final result
        } else {
            postfix has error in it (probably the original infix has syntax error)
        }
        */
    }

    /** Main method. */
    public static void main(String[] args) {
        String[] infixArray = {"(((A + B) - (C - D)) / (E - F))",
                               "(((D)))",
                               "((A))",
                               "((A",
                               "(B",
                               "D))",
                               "))",
                               "(( D",
                               "((A + B))",
                               "((A * B))",
                               "(A / B)",
                               "A * B",
                               "A / B + C",
                               "A * (B - C) ",
                               "(((E * F)))",
                               "D",
                               "( A - B * C)",
                               "A- B / C",
                               "( A / B * C)",
                               "( A - C * C)",
                               "( A * C * C) ",
                               "( D / C * C)",
                               "A - C * C ",
                               "A - B * C +D / E",
                               "A*G - C * C * D",
                               "A B - C + C * D",
                               "(( A - B * C) * D * E) + ( F / G * H + I )",
                               "(A - B) * (( C * D ) + E)",
                               "(((  )C(  ) )C(((  ))))",
                               "(((  )C(  )(((  ))))",
                               "A * C E / C) + (C A - B)",
                               "A * ( B / C) + D * ( A - B)",
                               "A * ( B / C) + D * A - B"};


        // Iterate through the above array, testing and converting all infix
        // expressions.
        for (String s : infixArray) {
            PostfixAndInfix expression = new PostfixAndInfix(s);

            // If it's not possible to convert the infix expression to a postfix
            // expression.
            if (! expression.infixConversion()) {
                continue;
            }

            expression.postfixEvaluation();
            System.out.println(expression.infixExpression + " --> " +
                               expression.postfixExpression + " --> " +
                               expression.postfixValue);

        }
    }
}
```

Question 1

By tracing the discussed algorithm regarding how to convert an infix expression into a postfix expression, please explain in a step-by-step manner how to convert the example infix expression **((1+(5-2)-4)/(6-5))** to the postfix expression **1 2 + 3 4 - - 6 5 - /**.

1. (1+(2-(3-4))/(6-5))
    1. stack = {
    2. postfix =
2. (1+(2-(3-4))/(6-5))
    1. stack = { --> (
    2. postfix =
3. 1+(2-(3-4))/(6-5))
    1. stack = { --> (, --> (
    2. postfix = '
4. +(2-(3-4))/(6-5))
    1. stack = { --> (, --> (
    2. postfix = 1
5. 2-(3-4))/(6-5))
    1. stack = + --> { --> (, --> (
    2. postfix = 1
6. -(3-4))/(6-5))
    1. stack = + --> { --> (, --> (
    2. postfix = 1 2
7. (3-4))/(6-5))
    1. stack = { --> (, --> (
    2. postfix = 1 2 +
8. (3-4))/(6-5))
    1. stack = - --> { --> (, --> (
    2. postfix = 1 2 +
9. 3-4))/(6-5))
    1. stack = { --> - --> { --> (, --> (
    2. postfix = 1 2 +
10. -4))/(6-5))
    1. stack = { --> - --> { --> (, --> (
    2. postfix = 1 2 + 3
11. 4))/(6-5))
    1. stack = - --> { --> - --> { --> (, --> (
    2. postfix = 1 2 + 3
12.))/(6-5))
    1. stack = - --> { --> - --> { --> (, --> (
    2. postfix = 1 2 + 3 4
13.)/(6-5))
    1. stack = - --> { --> - --> { --> (, --> (
    2. postfix = 1 2 + 3 4 -
14. /(6-5))
    1. stack = { --> (, --> (
    2. postfix = 1 2 + 3 4 - -
15. (6-5))
    1. stack = / --> { --> (, --> (
    2. postfix = 1 2 + 3 4 - -
16. 6-5))
    1. stack = { --> (, --> (, --> (
    2. postfix = 1 2 + 3 4 - -
17. -5))
    1. stack = { --> (, --> (, --> (
    2. postfix = 1 2 + 3 4 - - 6
18. 5))
    1. stack = - --> { --> (, --> (, --> (
    2. postfix = 1 2 + 3 4 - - 6
19.))
    1. stack = - --> { --> (, --> (, --> (
    2. postfix = 1 2 + 3 4 - - 6 5
20.)
    1. stack = / --> (
    2. postfix = 1 2 + 3 4 - - 6 5 -
21.
    1. stack = "
    2. postfix = 1 2 + 3 4 - - 6 5 - /

Question 2

Please evaluate the following postfix expression and write your answer. Show the state of the evaluation stack just before the first arithmetic operation is applied and then just after each additional operation is performed. All operands listed below are single-digit integers. All arithmetic is integer arithmetic.

Postfix expression: 1 2 + 3 4 - - 6 5 - /

1. 1 2 + 3 4 - - 6 5 - /
    1. stack = "
2. 2 + 3 4 - - 6 5 - /
    1. stack = 1
3. + 3 4 - - 6 5 - /
    1. stack = 2 --> 1
4. 3 4 - - 6 5 - /
    1. int right = pop() --> 2
    2. operand --> +
    3. int left = pop() --> 1
    4. stack = 3
5. 4 - - 6 5 - /
    1. stack = 3 --> 3
6. - - 6 5 - /
    1. stack = 4 --> 3 --> 3
7. - 6 5 - /
    1. int right = pop() --> 4
    2. operand --> -
    3. int left = pop() --> 3
    4. stack = 3 --> 3
8. 6 5 - /
    1. int right = pop() --> -1
    2. operand --> -
    3. int left = pop() --> 3
    4. stack = 4
9. 5 - /
    1. stack = 6 --> 4
10. - /
    1. stack = 5 --> 6 --> 4
11. /
    1. int right = pop() --> 5
    2. operand --> -
    3. int left = pop() --> 6
    4. stack = 1 --> 4
12. "
    1. int right = pop() --> 1
    2. operand --> /
    3. int left = pop() --> 4
    4. stack = 4