```java

    /**
     * Evaluates an expression and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with an expr string
     * @return value of the expression
     * @updates source
     * @requires <pre>
     * [an expr string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any expr string]
     * </pre>
     * @ensures <pre>
     * valueOfExpr =
     *   [value of longest expr string at start of #source]  and
     * #source = [longest expr string at start of #source] * source
     * </pre>
     */
    public static int valueOfExpr(StringBuilder source) {
        int value = valueOfTerm(source);
        while (source.charAt( ) == '+' || source.charAt( ) == '-') {
            StringBuilder op = source.deleteCharAt( );
            int nextTerm = valueOfTerm(source);
            if (op.charAt( ) == '+') {
                value = value + nextTerm;
            } else /* "-" */ {
                value = value - nextTerm;
            }
        }
        return value;
    }

    /**
     * Evaluates a term and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a term string
     * @return value of the term
     * @updates source
     * @requires <pre>
     * [a term string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any term string]
     * </pre>
     * @ensures <pre>
     * valueOfTerm =
     *   [value of longest term string at start of #source]  and
     * #source = [longest term string at start of #source] * source
     * </pre>
     */
    private static int valueOfTerm(StringBuilder source) {
        int value = valueOfFactor(source);
        while (source.charAt( ) == '/' || source.charAt( ) == '*') {
            StringBuilder op = source.deleteCharAt( );
            int nextTerm = valueOfFactor(source);
            if (op.charAt( ) == '*') {
                value = value * nextTerm;
            } else /* "/" */ {
                value = value / nextTerm;
            }
        }
        return value;
    }

    /**
     * Evaluates a factor and returns its value.
     *
     * @param source
```

```
     *            the {@code StringBuilder} that starts with a factor string
     * @return value of the factor
     * @updates source
     * @requires <pre>
     * [a factor string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any factor string]
     * </pre>
     * @ensures <pre>
     * valueOfFactor =
     *   [value of longest factor string at start of #source]  and
     * #source = [longest factor string at start of #source] * source
     * </pre>
     */
    private static int valueOfFactor(StringBuilder source) {
        int value = valueOfDigitSeq(source);
        while (source.charAt( ) == '(' || source.charAt( ) == ')') {
            // removes them but idk what to do with them
            StringBuilder op = source.deleteCharAt( );
            int nextTerm = valueOfDigitSeq(source);
//            if (op.charAt(0) == '*') {
//                value = value * nextTerm;
//            } else /* "/" */ {
//                value = value / nextTerm;
//            }

        }
        return value;
    }

    /**
     * Evaluates a digit sequence and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a digit-seq string
     * @return value of the digit sequence
     * @updates source
     * @requires <pre>
     * [a digit-seq string is a proper prefix of source, which
     * contains a character that is not a digit]
     * </pre>
     * @ensures <pre>
     * valueOfDigitSeq =
     *   [value of longest digit-seq string at start of #source]  and
     * #source = [longest digit-seq string at start of #source] * source
     * </pre>
     */
    private static int valueOfDigitSeq(StringBuilder source) {
        // String numbers = "0123456789";
        StringBuilder seq = new StringBuilder();
        while (source.length() >  ) {
//            StringBuilder op = source.deleteCharAt(0);
//            if (op.indexOf(numbers) >= 0) {
//                seq.append(op);
//            }
//            op = new StringBuilder();
            seq.append(valueOfDigit(source));
        }
        //return seq.toString().Integer.parseInt();
        return Integer.parseInt(seq.toString());
    }

    /**
     * Evaluates a digit and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a digit
     * @return value of the digit
     * @updates source
```

```
     * @requires 1 < |source| and [the first character of source is a digit]
     * @ensures <pre>
     * valueOfDigit = [value of the digit at the start of #source]  and
     * #source = [digit string at start of #source] * source
     * </pre>
     */
    private static int valueOfDigit(StringBuilder source) {
        String numbers = "0123456789";
        StringBuilder seq = new StringBuilder();
        while (source.length() >  ) {
            StringBuilder op = source.deleteCharAt( );
            if (op.indexOf(numbers) >=  ) {
                seq.append(op);
            }
            op = new StringBuilder();
        }
        //return seq.toString().Integer.parseInt();
        return Integer.parseInt(seq.toString());
    }
```