

```

1  /**
2  * Evaluates an expression and returns its value.
3  *
4  * @param source
5  *       the {@code StringBuilder} that starts with an expr string
6  * @return value of the expression
7  * @updates source
8  * @requires <pre>
9  * [an expr string is a proper prefix of source, and the longest
10 * such, s, concatenated with the character following s, is not a prefix
11 * of any expr string]
12 * </pre>
13 * @ensures <pre>
14 * valueOfExpr =
15 *   [value of longest expr string at start of #source] and
16 * #source = [longest expr string at start of #source] * source
17 * </pre>
18 */
19
20 public static int valueOfExpr(StringBuilder source) {
21     int value = valueOfTerm(source);
22     while (source.charAt( ) == '+' || source.charAt( ) == '-') {
23         StringBuilder op = source.deleteCharAt( );
24         int nextTerm = valueOfTerm(source);
25         if (op.charAt( ) == '+') {
26             value = value + nextTerm;
27         } else /* "-" */ {
28             value = value - nextTerm;
29         }
30     }
31     return value;
32 }
33
34 /**
35 * Evaluates a term and returns its value.
36 *
37 * @param source
38 *       the {@code StringBuilder} that starts with a term string
39 * @return value of the term
40 * @updates source
41 * @requires <pre>
42 * [a term string is a proper prefix of source, and the longest
43 * such, s, concatenated with the character following s, is not a prefix
44 * of any term string]
45 * </pre>
46 * @ensures <pre>
47 * valueOfTerm =
48 *   [value of longest term string at start of #source] and
49 * #source = [longest term string at start of #source] * source
50 * </pre>
51 */
52 private static int valueOfTerm(StringBuilder source) {
53     int value = valueOfFactor(source);
54     while (source.charAt( ) == '/' || source.charAt( ) == '*') {
55         StringBuilder op = source.deleteCharAt( );
56         int nextTerm = valueOfFactor(source);
57         if (op.charAt( ) == '*') {
58             value = value * nextTerm;
59         } else /* "/" */ {
60             value = value / nextTerm;
61         }
62     }
63     return value;
64 }
65
66 /**
67 * Evaluates a factor and returns its value.
68 *
69 * @param source

```

```

70     * the {@code StringBuilder} that starts with a factor string
71     * @return value of the factor
72     * @updates source
73     * @requires <pre>
74     * [a factor string is a proper prefix of source, and the longest
75     * such, s, concatenated with the character following s, is not a prefix
76     * of any factor string]
77     * </pre>
78     * @ensures <pre>
79     * valueOfFactor =
80     * [value of longest factor string at start of #source] and
81     * #source = [longest factor string at start of #source] * source
82     * </pre>
83     */
84     private static int valueOfFactor(StringBuilder source) {
85         int value = valueOfDigitSeq(source);
86         while (source.charAt( ) == '(' || source.charAt( ) == ')') {
87             // removes them but idk what to do with them
88             StringBuilder op = source.deleteCharAt( );
89             int nextTerm = valueOfDigitSeq(source);
90             // if (op.charAt(0) == '*') {
91             //     value = value * nextTerm;
92             // } else /* "/" */ {
93             //     value = value / nextTerm;
94             // }
95
96         }
97         return value;
98     }
99
100     /**
101     * Evaluates a digit sequence and returns its value.
102     *
103     * @param source
104     *     the {@code StringBuilder} that starts with a digit-seq string
105     * @return value of the digit sequence
106     * @updates source
107     * @requires <pre>
108     * [a digit-seq string is a proper prefix of source, which
109     * contains a character that is not a digit]
110     * </pre>
111     * @ensures <pre>
112     * valueOfDigitSeq =
113     * [value of longest digit-seq string at start of #source] and
114     * #source = [longest digit-seq string at start of #source] * source
115     * </pre>
116     */
117     private static int valueOfDigitSeq(StringBuilder source) {
118         // String numbers = "0123456789";
119         StringBuilder seq = new StringBuilder();
120         while (source.length() >  ) {
121             //     StringBuilder op = source.deleteCharAt(0);
122             //     if (op.indexOf(numbers) >= 0) {
123             //         seq.append(op);
124             //     }
125             //     op = new StringBuilder();
126             seq.append(valueOfDigit(source));
127         }
128         //return seq.toString().Integer.parseInt();
129         return Integer.parseInt(seq.toString());
130     }
131
132     /**
133     * Evaluates a digit and returns its value.
134     *
135     * @param source
136     *     the {@code StringBuilder} that starts with a digit
137     * @return value of the digit
138     * @updates source

```

```
139 * @requires 1 < |source| and [the first character of source is a digit]
140 * @ensures <pre>
141 * valueOfDigit = [value of the digit at the start of #source] and
142 * #source = [digit string at start of #source] * source
143 * </pre>
144 */
145 private static int valueOfDigit(StringBuilder source) {
146     String numbers = "0123456789";
147     StringBuilder seq = new StringBuilder();
148     while (source.length() > 0) {
149         StringBuilder op = source.deleteCharAt(0);
150         if (op.indexOf(numbers) >= 0) {
151             seq.append(op);
152         }
153         op = new StringBuilder();
154     }
155     //return seq.toString().Integer.parseInt();
156     return Integer.parseInt(seq.toString());
157 }
```