

```

1  import components.map.Map;
12
13 /**
14  * Layered implementation of secondary method {@code parse} for {@code Program}.
15  *
16  * @author Qinuo Shi & Yiming Cheng
17  *
18  */
19 public final class Program1Parse1 extends Program1 {
20
21     /*
22     * Private members -----
23     */
24
25     /**
26     * Parses a single BL instruction from {@code tokens} returning the
27     * instruction name as the value of the function and the body of the
28     * instruction in {@code body}.
29     *
30     * @param tokens
31     *         the input tokens
32     * @param body
33     *         the instruction body
34     * @return the instruction name
35     * @replaces body
36     * @updates tokens
37     * @requires <pre>
38     *   [<"INSTRUCTION"> is a prefix of tokens] and
39     *   [<Tokenizer.END_OF_INPUT> is a suffix of tokens]
40     * </pre>
41     * @ensures <pre>
42     *   if [an instruction string is a proper prefix of #tokens] and
43     *     [the beginning name of this instruction equals its ending name] and
44     *     [the name of this instruction does not equal the name of a primitive
45     *      instruction in the BL language] then
46     *     parseInstruction = [name of instruction at start of #tokens] and
47     *     body = [Statement corresponding to the block string that is the body of
48     *              the instruction string at start of #tokens] and
49     *     #tokens = [instruction string at start of #tokens] * tokens
50     * else
51     *   [report an appropriate error message to the console and terminate client]
52     * </pre>
53     */
54     private static String parseInstruction(Queue<String> tokens,
55         Statement body) {
56         assert tokens != null : "Violation of: tokens is not null";
57         assert body != null : "Violation of: body is not null";
58         assert tokens.length() > 0 && tokens.front().equals("INSTRUCTION") : ""
59             + "Violation of: <\\"INSTRUCTION\\"> is proper prefix of tokens";
60
61         // TODO - fill in body
62         tokens.dequeue();
63
64         /*
65         * Parse INSTRUCTION
66         */
67

```

```

68      * Check for errors according to the order of BL languages.
69      */
70      /*
71      * If the BL format is not found, report an error here
72      */
73      String instrName = tokens.dequeue();
74      Reporter.assertElseFatalError(Tokenizer.isIdentifier(instrName),
75          "Name is identifier");
76
77      Reporter.assertElseFatalError(
78          !instrName.equals("move") && !instrName.equals("turnleft")
79          && !instrName.equals("turnright")
80          && !instrName.equals("infect")
81          && !instrName.equals("skip"),
82          "Name not primitive instruction");
83
84      Reporter.assertElseFatalError(tokens.dequeue().equals("IS"),
85          "Cannot find IS");
86
87      body.parseBlock(tokens);
88      Reporter.assertElseFatalError(tokens.dequeue().equals("END"),
89          "Cannot find END");
90
91      String check = tokens.dequeue();
92      Reporter.assertElseFatalError(instrName.equals(check),
93          "Beginning of instruction equals its ending name");
94
95      return instrName;
96  }
97
98  /*
99  * Constructors -----
100  */
101
102  /**
103   * No-argument constructor.
104   */
105  public Program1Parse1() {
106      super();
107  }
108
109  /*
110  * Public methods -----
111  */
112
113  @Override
114  public void parse(SimpleReader in) {
115      assert in != null : "Violation of: in is not null";
116      assert in.isOpen() : "Violation of: in.is_open";
117      Queue<String> tokens = Tokenizer.tokens(in);
118      this.parse(tokens);
119  }
120
121  @Override
122  public void parse(Queue<String> tokens) {
123      assert tokens != null : "Violation of: tokens is not null";
124      assert tokens.length() > 0 : ""

```

```
125         + "Violation of: Tokenizer.END_OF_INPUT is a suffix of tokens";
126
127     // TODO - fill in body
128     /*
129     * If the BL format is not found, report an error here
130     */
131     Map<String, Statement> map = this.newContext();
132     Statement toolBody = this.newBody();
133     Reporter.assertElseFatalError(tokens.dequeue().equals("PROGRAM"),
134         "Cannot find PROGRAM at the beginning.");
135
136     /*
137     * Extract the name
138     */
139     String name = tokens.dequeue();
140     Reporter.assertElseFatalError(Tokenizer.isIdentifier(name),
141         "Name is identifier");
142
143     /*
144     * If the BL format is not found, report an error here
145     */
146     this.setName(name);
147     Reporter.assertElseFatalError(tokens.dequeue().equals("IS"),
148         "Cannot find IS");
149
150     /*
151     * Parse the INSTRUCTION
152     */
153     while (tokens.front().equals("INSTRUCTION")) {
154         Statement toolContext = this.newBody();
155         String instrName = parseInstruction(tokens, toolContext);
156         Reporter.assertElseFatalError(!map.containsKey(instrName),
157             "It is already existed.");
158         map.add(instrName, toolContext);
159     }
160
161     /*
162     * If the BL format is not found, report an error here
163     */
164     this.swapContext(map);
165     Reporter.assertElseFatalError(tokens.dequeue().equals("BEGIN"),
166         "Cannot find BEGIN");
167
168     /*
169     * If the BL format is not found, report an error here
170     */
171     toolBody.parseBlock(tokens);
172     this.swapBody(toolBody);
173     Reporter.assertElseFatalError(tokens.dequeue().equals("END"),
174         "Cannot find END");
175
176     String nameInQueue = tokens.dequeue();
177     Reporter.assertElseFatalError(name.equals(nameInQueue),
178         "Beginning identifier equals ending identifier");
179
180     Reporter.assertElseFatalError(
181         tokens.front().equals(Tokenizer.END_OF_INPUT),
```

```
182         "Tokenizer.END_OF_INPUT is a suffix of tokens");
183
184     }
185
186     /*
187     * Main test method -----
188     */
189
190     /**
191     * Main method.
192     *
193     * @param args
194     *         the command line arguments
195     */
196     public static void main(String[] args) {
197         SimpleReader in = new SimpleReader1L();
198         SimpleWriter out = new SimpleWriter1L();
199         /*
200         * Get input file name
201         */
202         out.print("Enter valid BL program file name: ");
203         String fileName = in.nextLine();
204         /*
205         * Parse input file
206         */
207         out.println("*** Parsing input file ***");
208         Program p = new Program1Parse1();
209         SimpleReader file = new SimpleReader1L(fileName);
210         Queue<String> tokens = Tokenizer.tokens(file);
211         file.close();
212         p.parse(tokens);
213         /*
214         * Pretty print the program
215         */
216         out.println("*** Pretty print of parsed program ***");
217         p.prettyPrint(out);
218
219         in.close();
220         out.close();
221     }
222
223 }
224
```