

```
1 • MOVE ( )
2 • TURNLEFT ( )
3 • TURNRIGHT ( )
4 • INFECT ( )
5 • SKIP ( )
6 • HALT ( )
7 • JUMP ( )
8 • JUMP_IF_NOT_NEXT_IS_EMPTY ( )
9 • JUMP_IF_NOT_NEXT_IS_NOT_EMPTY ( )
10 • JUMP_IF_NOT_NEXT_IS_WALL ( )
11 • JUMP_IF_NOT_NEXT_IS_NOT_WALL ( )
12 • JUMP_IF_NOT_NEXT_IS_FRIEND ( )
13 • JUMP_IF_NOT_NEXT_IS_NOT_FRIEND ( )
14 • JUMP_IF_NOT_NEXT_IS_ENEMY ( )
15 • JUMP_IF_NOT_NEXT_IS_NOT_ENEMY ( )
16 • JUMP_IF_NOT_RANDOM ( )
17 • JUMP_IF_NOT_TRUE ( )
```

```
18
19 PROGRAM Example1 IS
20 BEGIN
21     IF next-is-wall THEN
22         turnright
23         turnright
24         infect
25     END IF
26 END Example1
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
41 PROGRAM Example2 IS
42 BEGIN
43     IF next-is-wall THEN
44         turnright
45         turnright
46         infect
47     ELSE
48         infect
49         move
50     END IF
51 END Example2
52
53
54
55
56
57
58
59
60
61
62
63
64
65
```

```
66 PROGRAM Example3 IS
67 BEGIN
68     WHILE next-is-not-empty DO
69         IF next-is-wall THEN
```

```

70         turnright
71         turnright
72         infect
73     ELSE
74         infect
75         move
76     END IF
77 END WHILE
78 END Example3
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98 PROGRAM Example4 IS
99
100     INSTRUCTION TurnBackAndInfect IS
101         turnright
102         turnright
103         IF next-is-enemy THEN
104             infect
105         END IF
106     END TurnBackAndInfect
107
108 BEGIN
109     WHILE true DO
110         TurnBackAndInfect
111     END WHILE
112 END Example4
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132 /**
133  * Returns the location of the next primitive instruction to execute in
134  * compiled program {@code cp} given what the bug sees {@code wbs} and
135  * starting from location {@code pc}.
136  *
137  * @param cp
138  *         the compiled program

```

```

139 * @param wbs
140 *         the {@code CellState} indicating what the bug sees
141 * @param pc
142 *         the program counter
143 * @return the location of the next primitive instruction to execute
144 * @requires <pre>
145 * [cp is a valid compiled BL program] and
146 * 0 <= pc < cp.length and
147 * [pc is the location of an instruction byte code in cp, that is, pc
148 *  cannot be the location of an address]
149 * </pre>
150 * @ensures <pre>
151 * [return the address of the next primitive instruction that
152 *  should be executed in program cp given what the bug sees wbs and
153 *  starting execution at address pc in program cp]
154 * </pre>
155 */
156 public static int nextPrimitiveInstructionAddress(int[] cp, CellState wbs,
157     int pc) {
158     // don't know what is CellState
159 }

```