

```

1  Token: separator Lexeme: $$
2  <Rat16F> -> $$ <Opt Function Definitions>
3  $$ <Opt Declaration List> <Statement List> $$
4
5  Token: keyword Lexeme: function
6  <Opt Function Definitions> -> <Function Definitions> | <Empty>
7  <Function Definitions> -> <Function> | <Function> <Function Definitions>
8  <Function> -> function <Identifier> [ <Opt Parameter List> ] <Opt Declaration List> <Body>
9
10 Token: identifier Lexeme: Subtract
11
12 Token: separator Lexeme: [
13
14 Token: identifier Lexeme: imVal
15 <Opt Parameter List> -> <Parameter List> | <Empty>
16 <Parameter List> -> <Parameter> | <Parameter> , <Parameter List>
17 <Parameter> -> <IDs> : <Qualifier>
18 <IDs> -> <Identifier> | <Identifier>, <IDs>
19
20 Token: separator Lexeme: :
21
22 Token: keyword Lexeme: integer
23 <Qualifier> -> integer | boolean | real
24
25 Token: separator Lexeme: ]
26
27 Token: keyword Lexeme: real
28 <Opt Declaration List> -> <Declaration List> | <Empty>
29 <Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
30 <Declaration> -> <Qualifier> <IDs>
31 <Qualifier> -> integer | boolean | real
32
33 Token: identifier Lexeme: retVal92
34 <IDs> -> <Identifier> | <Identifier>, <IDs>
35
36 Token: separator Lexeme: ;
37
38 Token: separator Lexeme: {
39 <Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
40 <Declaration> -> <Qualifier> <IDs>
41 <Qualifier> -> integer | boolean | real
42 <Body> -> { <Statement List> }
43
44 Token: identifier Lexeme: imVal
45 <Statement List> -> <Statement> | <Statement> <Statement List>
46 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
47 <Compound> -> { <Statement List> }
48 <Assign> -> <Identifier> := <Expression>;
49
50 Token: operator Lexeme: :=
51
52 Token: identifier Lexeme: imVal
53 <Expression> -> <Term> <Expression Prime>
54 <Term> -> <Factor> <Term Prime>
55 <Factor> -> - <Primary> | <Primary>
56 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
57
58 Token: operator Lexeme: -
59 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
60 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
61
62 Token: separator Lexeme: (
63 <Term> -> <Factor> <Term Prime>
64 <Factor> -> - <Primary> | <Primary>
65 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

```

```

66
67 Token: integer Lexeme: 2
68 <Expression> -> <Term> <Expression Prime>
69 <Term> -> <Factor> <Term Prime>
70 <Factor> -> - <Primary> | <Primary>
71 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
72
73 Token: operator Lexeme: *
74 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
75
76 Token: identifier Lexeme: imVal
77 <Factor> -> - <Primary> | <Primary>
78 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
79
80 Token: separator Lexeme: )
81 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
82 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
83
84 Token: separator Lexeme: ;
85 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
86 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
87
88 Token: identifier Lexeme: retVal92
89 <If> -> if (<Condition>) <Statement> endif |
90 if (<Condition>) <Statement> else <Statement> endif
91 <Return> -> return ; | return <Expression> ;
92 <Write> -> print (<Expression>);
93 <Read> -> read (<IDs>);
94 <While> -> while (<Condition>) <Statement>
95 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
96 <Compound> -> { <Statement List> }
97 <Assign> -> <Identifier> := <Expression>;
98
99 Token: operator Lexeme: :=
100
101 Token: identifier Lexeme: imVal
102 <Expression> -> <Term> <Expression Prime>
103 <Term> -> <Factor> <Term Prime>
104 <Factor> -> - <Primary> | <Primary>
105 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
106
107 Token: separator Lexeme: ;
108 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
109 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
110
111 Token: keyword Lexeme: return
112 <If> -> if (<Condition>) <Statement> endif |
113 if (<Condition>) <Statement> else <Statement> endif
114 <Return> -> return ; | return <Expression> ;
115
116 Token: identifier Lexeme: retVal92
117 <Expression> -> <Term> <Expression Prime>
118 <Term> -> <Factor> <Term Prime>
119 <Factor> -> - <Primary> | <Primary>
120 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
121
122 Token: separator Lexeme: ;
123 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
124 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
125
126 Token: separator Lexeme: }
127 <Write> -> print (<Expression>);
128 <Read> -> read (<IDs>);

```

```

129 <While> -> while (<Condition>) <Statement>
130 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
131 <Compound> -> { <Statement List> }
132 <Assign> -> <Identifier> := <Expression>;
133 <If> -> if (<Condition>) <Statement> endif |
134 if (<Condition>) <Statement> else <Statement> endif
135 <Return> -> return ; | return <Expression> ;
136 <Write> -> print (<Expression>);
137 <Read> -> read (<IDs>);
138 <While> -> while (<Condition>) <Statement>
139
140 Token: separator Lexeme: $$
141 <Function Definitions> -> <Function> | <Function> <Function Definitions>
142 <Function> -> function <Identifier> [ <Opt Parameter List> ] <Opt Declaration List> <Body>
143
144 Token: keyword Lexeme: integer
145 <Opt Declaration List> -> <Declaration List> | <Empty>
146 <Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
147 <Declaration> -> <Qualifier> <IDs>
148 <Qualifier> -> integer | boolean | real
149
150 Token: identifier Lexeme: low_av
151 <IDs> -> <Identifier> | <Identifier>, <IDs>
152
153 Token: separator Lexeme: ,
154
155 Token: identifier Lexeme: high_av
156
157 Token: separator Lexeme: ;
158
159 Token: keyword Lexeme: read
160 <Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
161 <Declaration> -> <Qualifier> <IDs>
162 <Qualifier> -> integer | boolean | real
163 <Statement List> -> <Statement> | <Statement> <Statement List>
164 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
165 <Compound> -> { <Statement List> }
166 <Assign> -> <Identifier> := <Expression>;
167 <If> -> if (<Condition>) <Statement> endif |
168 if (<Condition>) <Statement> else <Statement> endif
169 <Return> -> return ; | return <Expression> ;
170 <Write> -> print (<Expression>);
171 <Read> -> read (<IDs>);
172
173 Token: separator Lexeme: (
174
175 Token: identifier Lexeme: low_av
176 <IDs> -> <Identifier> | <Identifier>, <IDs>
177
178 Token: separator Lexeme: ,
179
180 Token: identifier Lexeme: high_av
181
182 Token: separator Lexeme: )
183
184 Token: separator Lexeme: ;
185
186 Token: keyword Lexeme: while
187 <While> -> while (<Condition>) <Statement>
188
189 Token: separator Lexeme: (
190
191 Token: identifier Lexeme: low_av
192 <Condition> -> <Expression> <Relop> <Expression>
193 <Expression> -> <Term> <Expression Prime>
194 <Term> -> <Factor> <Term Prime>
195 <Factor> -> - <Primary> | <Primary>

```

```

196 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
    false
197
198 Token: operator Lexeme: =
199 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
200 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
201 <Relop> -> = | /= | > | < | => | <=
202
203 Token: operator Lexeme: <
204
205 Token: identifier Lexeme: high_av
206 <Expression> -> <Term> <Expression Prime>
207 <Term> -> <Factor> <Term Prime>
208 <Factor> -> - <Primary> | <Primary>
209 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
    false
210
211 Token: separator Lexeme: )
212 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
213 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
214
215 Token: separator Lexeme: {
216 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
217 <Compound> -> { <Statement List> }
218
219 Token: keyword Lexeme: print
220 <Statement List> -> <Statement> | <Statement> <Statement List>
221 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
222 <Compound> -> { <Statement List> }
223 <Assign> -> <Identifier> := <Expression>;
224 <If> -> if (<Condition>) <Statement> endif |
225 if (<Condition>) <Statement> else <Statement> endif
226 <Return> -> return ; | return <Expression> ;
227 <Write> -> print (<Expression>);
228
229 Token: separator Lexeme: (
230
231 Token: identifier Lexeme: low_av
232 <Expression> -> <Term> <Expression Prime>
233 <Term> -> <Factor> <Term Prime>
234 <Factor> -> - <Primary> | <Primary>
235 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
    false
236
237 Token: separator Lexeme: )
238 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
239 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
240
241 Token: separator Lexeme: ;
242
243 Token: keyword Lexeme: print
244 <Read> -> read (<IDs>);
245 <While> -> while (<Condition>) <Statement>
246 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
247 <Compound> -> { <Statement List> }
248 <Assign> -> <Identifier> := <Expression>;
249 <If> -> if (<Condition>) <Statement> endif |
250 if (<Condition>) <Statement> else <Statement> endif
251 <Return> -> return ; | return <Expression> ;
252 <Write> -> print (<Expression>);
253
254 Token: separator Lexeme: (
255
256 Token: identifier Lexeme: Subtract
257 <Expression> -> <Term> <Expression Prime>
258 <Term> -> <Factor> <Term Prime>
259 <Factor> -> - <Primary> | <Primary>

```

```

260 <Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
    false
261
262 Token: separator Lexeme: [
263
264 Token: identifier Lexeme: low_av
265 <IDs> -> <Identifier> | <Identifier>, <IDs>
266
267 Token: separator Lexeme: ]
268
269 Token: separator Lexeme: )
270 <Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
271 <Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
272
273 Token: separator Lexeme: ;
274
275 Token: separator Lexeme: }
276 <Read> -> read (<IDs>);
277 <While> -> while (<Condition>) <Statement>
278 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
279 <Compound> -> { <Statement List> }
280 <Assign> -> <Identifier> := <Expression>;
281 <If> -> if (<Condition>) <Statement> endif |
282 if (<Condition>) <Statement> else <Statement> endif
283 <Return> -> return ; | return <Expression> ;
284 <Write> -> print (<Expression>);
285 <Read> -> read (<IDs>);
286 <While> -> while (<Condition>) <Statement>
287
288 Token: separator Lexeme: $$
289 <Assign> -> <Identifier> := <Expression>;
290 <If> -> if (<Condition>) <Statement> endif |
291 if (<Condition>) <Statement> else <Statement> endif
292 <Return> -> return ; | return <Expression> ;
293 <Write> -> print (<Expression>);
294 <Read> -> read (<IDs>);
295 <While> -> while (<Condition>) <Statement>
296 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
297 <Compound> -> { <Statement List> }
298 <Assign> -> <Identifier> := <Expression>;
299 <If> -> if (<Condition>) <Statement> endif |
300 if (<Condition>) <Statement> else <Statement> endif
301 <Return> -> return ; | return <Expression> ;
302 <Write> -> print (<Expression>);
303 <Read> -> read (<IDs>);
304 <While> -> while (<Condition>) <Statement>
305
306 Token: Lexeme:

```