

```

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

```

```
68 <Write> -> print (<Expression>);
69 <Read> -> read (<IDs>);
70 <While> -> while (<Condition>) <Statement>
71 <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>
72 <Compound> -> { <Statement List> }
73 <Assign> -> <Identifier> := <Expression>;
74 <If> -> if (<Condition>) <Statement> endif |
75 if (<Condition>) <Statement> else <Statement> endif
76 <Return> -> return ; | return <Expression> ;
77 <Write> -> print (<Expression>);
78 <Read> -> read (<IDs>);
79 <While> -> while (<Condition>) <Statement>
80
81 Token: Lexeme:
```