```
Token: separator  Lexeme: $$
<Rat16F> -> $$ <Opt Function Definitions>
$$ <Opt Declaration List> <Statement List> $$

Token: keyword  Lexeme: function
<Opt Function Definitions> -> <Function Definitions> | <Empty>
<Function Definitions> -> <Function> | <Function> <Function Definitions>
<Function> -> function  <Identifier> [ <Opt Parameter List> ] <Opt Declaration List> <Body>

Token: identifier Lexeme: Hello

Token: separator  Lexeme: [

Token: identifier Lexeme: me
<Opt Parameter List> ->  <Parameter List> | <Empty>
<Parameter List> -> <Parameter> | <Parameter> , <Parameter List>
<Parameter> -> <IDs> : <Qualifier>
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: :

Token: keyword  Lexeme: real
<Qualifier> -> integer | boolean | real

Token: separator  Lexeme: ,

Token: identifier Lexeme: alice
<Parameter> -> <IDs> : <Qualifier>
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: :

Token: keyword  Lexeme: integer
<Qualifier> -> integer | boolean | real

Token: separator  Lexeme: ,

Token: identifier Lexeme: bob
<Parameter> -> <IDs> : <Qualifier>
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: :

Token: keyword  Lexeme: boolean
<Qualifier> -> integer | boolean | real

Token: separator  Lexeme: ]

Token: keyword  Lexeme: boolean
<Opt Declaration List> -> <Declaration List> | <Empty>
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real

Token: identifier Lexeme: modest
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: ;

Token: keyword  Lexeme: boolean
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real

Token: identifier Lexeme: mouse
<IDs> -> <Identifier> | <Identifier>, <IDs>

```

```
Token: separator  Lexeme: ;

Token: keyword  Lexeme: real
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real

Token: identifier Lexeme: pink
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: ;

Token: keyword  Lexeme: integer
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real

Token: identifier Lexeme: floyd
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: ;

Token: separator  Lexeme: {
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real
<Body> -> { <Statement List> }

Token: identifier Lexeme: alice
<Statement List> -> <Statement> | <Statement> <Statement List>
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;

Token: operator Lexeme: :=

Token: identifier Lexeme: me
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true | false

Token: separator  Lexeme: ;
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: keyword  Lexeme: if
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif

Token: separator  Lexeme: (

Token: identifier Lexeme: pink
<Condition> -> <Expression> <Relop> <Expression>
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true | false

Token: operator Lexeme: +
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: identifier Lexeme: floyd
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: operator Lexeme: >
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
<Relop> -> = | /= | > | < | => | <=

Token: identifier Lexeme: modest
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: operator Lexeme: /
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon

Token: identifier Lexeme: mouse
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: separator  Lexeme: )
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: identifier Lexeme: me
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;

Token: operator Lexeme: :=

Token: identifier Lexeme: alice
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: separator  Lexeme: ;
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: keyword  Lexeme: else
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token: identifier Lexeme: me
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;

Token: operator Lexeme: :=

Token: identifier Lexeme: modest
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false
```

```

Token: operator Lexeme: +
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: identifier Lexeme: mouse
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: separator  Lexeme: ;
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: keyword  Lexeme: endif
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token: keyword  Lexeme: return
<Return> -> return ; | return <Expression> ;

Token: identifier Lexeme: me
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: separator  Lexeme: [

Token: identifier Lexeme: modest
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: ,

Token: identifier Lexeme: alice

Token: separator  Lexeme: ]

Token: separator  Lexeme: ;
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: separator  Lexeme: }
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token: separator  Lexeme: $$
<Function Definitions> -> <Function> | <Function> <Function Definitions>
<Function> -> function  <Identifier> [ <Opt Parameter List> ] <Opt Declaration List> <Body>

Token: keyword  Lexeme: while
```

```
<Opt Declaration List> -> <Declaration List> | <Empty>
<Declaration List> -> <Declaration> ; | <Declaration> ; <Declaration List>
<Declaration> -> <Qualifier> <IDs>
<Qualifier> -> integer | boolean | real
<Statement List> -> <Statement> | <Statement> <Statement List>
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token: separator  Lexeme: (

Token: integer  Lexeme: 1
<Condition> -> <Expression> <Relop> <Expression>
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: operator Lexeme: >
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon
<Relop> -> = | /= | > | < | => | <=

Token: integer  Lexeme: 0
<Expression> -> <Term> <Expression Prime>
<Term> -> <Factor> <Term Prime>
<Factor> -> - <Primary> | <Primary>
<Primary> -> <Identifier> | <Integer> | <Identifier> [<IDs>] | (<Expression>) | <Real> | true |
false

Token: separator  Lexeme: )
<Term Prime> -> * <Factor> <Term Prime> | / Factor <Term Prime> | epsilon
<Expression Prime> -> +<Term> <Expression Prime> | -<Term> <Expression Prime> | epsilon

Token: separator  Lexeme: {
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }

Token: keyword  Lexeme: read
<Statement List> -> <Statement> | <Statement> <Statement List>
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);

Token: separator  Lexeme: (

Token: identifier Lexeme: Hello
<IDs> -> <Identifier> | <Identifier>, <IDs>

Token: separator  Lexeme: )

Token: separator  Lexeme: ;

Token: separator  Lexeme: }
<While> -> while (<Condition>) <Statement>
```

```
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token: separator  Lexeme: $$
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>
<Statement> -> <Compound> | <Assign> | <If> |  <Return> | <Write> | <Read> | <While>
<Compound> -> { <Statement List> }
<Assign> -> <Identifier> := <Expression>;
<If> -> if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif
<Return> -> return ; | return <Expression> ;
<Write> -> print (<Expression>);
<Read> -> read (<IDs>);
<While> -> while (<Condition>) <Statement>

Token:  Lexeme:
```