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

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

Token: separator  Lexeme: {
<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> }

Token: identifier Lexeme: a
<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: b
<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: c
<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: }
<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: 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:
```