```

Token:  Keyword         Lexeme: function

        <Rat18F> -> <Opt Function Definitions> $$ <Opt Declaration List> <Statement List>
        <Opt Function Definitions> ->  <Function Definitions> | <Empty>

Token:  Identifier      Lexeme: myInvalidFunction

        <Function Definitions> -> <Function> | <Function> <Function Definitions>
        <Function> ->  function <Identifier> ( <Opt Parameter List> ) <Opt Declaration List> <Body>
        <Identifier>

Token:  Separator       Lexeme: (


Token:  Identifier      Lexeme: var1x

        <Opt Parameter List> -> <Parameter List> | <Empty>
        <Parameter List> -> <Parameter> | <Parameter> , <Parameter List>
        <Parameter> -> <IDs> : <Qualifier>
        <IDs> -> <Identifier> | <Identifier>, <IDs>
        <Identifier>

Token:  Separator       Lexeme: :


Token:  Keyword         Lexeme: int

        <Qualifier> ->  int | boolean | real

Token:  Separator       Lexeme: )


Token:  Separator       Lexeme: {

        <Opt Declaration List> -> <Declaration List> | <Empty>
        <Empty> -> Îµ
        <Body> -> { <Statement List> }

Token:  Keyword         Lexeme: return

        <Statement List> -> <Statement> | <Statement> <Statement List>
        <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>

Token:  Separator       Lexeme: (

        <Return> -> return; |  return <Expression>;
        <Expression> -> <Term> <ExpressionPrime>
        <Term> -> <Factor> <TermPrime>
        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false

Token:  Identifier      Lexeme: var1x

        <Expression> -> <Term> <ExpressionPrime>
        <Term> -> <Factor> <TermPrime>
        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false
        <Identifier>

Token:  Operator        Lexeme: *

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

Token:  Identifier      Lexeme: var2x
```

```

        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false
        <Identifier>

Token:  Separator       Lexeme: )

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
        <ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>
        <Empty> -> Îµ

Token:  Operator        Lexeme: +

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
        <ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

Token:  Integer         Lexeme: 50

        <Term> -> <Factor> <TermPrime>
        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false
        <Integer>

Token:  Separator       Lexeme: ;

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
        <ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>
        <Empty> -> Îµ

Token:  Separator       Lexeme: }


Token:  Separator       Lexeme: $$


Token:  Keyword         Lexeme: int

        <Opt Declaration List> -> <Declaration List> | <Empty>
        <Declaration List> -> <Declaration>; | <Declaration>; <Declaration List>
        <Declaration> -> <Qualifier> <IDs>
        <Qualifier> ->  int | boolean | real

Token:  Identifier      Lexeme: a

        <IDs> -> <Identifier> | <Identifier>, <IDs>
        <Identifier>

Token:  Separator       Lexeme: ,


Token:  Identifier      Lexeme: b

        <IDs> -> <Identifier> | <Identifier>, <IDs>
        <Identifier>

Token:  Separator       Lexeme: ;


Token:  Identifier      Lexeme: a

        <Statement List> -> <Statement> | <Statement> <Statement List>
        <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
        <Assign> -> <Identifier> = <Expression>;
        <Identifier>
```

```

Token:  Operator        Lexeme: =


Token:  Integer         Lexeme: 0

        <Expression> -> <Term> <ExpressionPrime>
        <Term> -> <Factor> <TermPrime>
        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false
        <Integer>

Token:  Separator       Lexeme: ;

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
        <ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>
        <Empty> -> Îµ

Token:  Identifier      Lexeme: b

        <Statement List> -> <Statement> | <Statement> <Statement List>
        <Statement> -> <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
        <Assign> -> <Identifier> = <Expression>;
        <Identifier>

Token:  Operator        Lexeme: =


Token:  Integer         Lexeme: 1

        <Expression> -> <Term> <ExpressionPrime>
        <Term> -> <Factor> <TermPrime>
        <Factor> -> - <Primary> | <Primary>
        <Primary> -> <Identifier> | <Integer> | <Identifier> ( <IDs> ) | ( <Expression> ) | <Real> | true | false
        <Integer>

Token:  Separator       Lexeme: ;

        <TermPrime> -> * <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>
        <ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>
        <Empty> -> Îµ

Token:  Illegal         Lexeme: invalidVar1


ERROR: Illegal symbol ’invalidVar1’ Line: 15
```