Test Case 1: (< 10 lines)

Code (8 lines):

@@

boolean valid;

@@

if(5+5 > 2)

valid := False;

else

return 6;

endif

Terminal Run:

Arts-MacBook-Pro:Syntax Analyzer Arty$ python3.4 Syntax\_Analyzer.py

Enter file you would like to open (type "quit" to exit): testcase1.txt

Lexer working...

File open!

...Lexer complete!

Your Tokens and Lexemes have been saved as testcase1.RAT in the working directory.

Syntax Analyser running...

...Syntax Analyser finished!

There were no errors!

Your syntactic analysis of testcase1 has been saved as testcase1.SA in the working directory.

Would you like to process another file? (yes/no): no

Goodbye!

Contents of testcase2.SA:

Token: operator Lexeme: @@ Line: 1

<Rat15S> ::= <Opt Function Definitions> @@ <Opt Declaration List> @@ <Statement List>

<Opt Function Definitions> ::= <Function Definitions> | <Empty>

<Empty> ::= epsilon

Token: keyword Lexeme: boolean Line: 2

<Opt Declaration List> ::= <Declaration List> | <Empty>

<Declaration List> := <Declaration> ; | <Declaration> ; <Declaration List>

<Declaration> ::= <Qualifier> <IDs>

<Qualifier> ::= int | boolean | real

Token: identifier Lexeme: valid Line: 2

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: ; Line: 2

Token: operator Lexeme: @@ Line: 3

Token: keyword Lexeme: if Line: 4

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<If> ::= if ( <Condition> ) <Statement > <ifPrime>

Token: separator Lexeme: ( Line: 4

Token: integer Lexeme: 5 Line: 4

<Condition> ::= <Expression> <Relop> <Expression>

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: + Line: 4

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

Token: integer Lexeme: 5 Line: 4

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: > Line: 4

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

<Relop> ::= = | != | > | < | => | <=

Token: integer Lexeme: 2 Line: 4

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 4

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: identifier Lexeme: valid Line: 5

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Assign> ::= <Identifier> := <Expression> ;

Token: operator Lexeme: := Line: 5

Token: identifier Lexeme: False Line: 5

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ; Line: 5

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: keyword Lexeme: else Line: 6

<ifPrime> ::= endif | else <Statement> endif

Token: keyword Lexeme: return Line: 7

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Return> ::= return ; | return <Expression> ;

Token: integer Lexeme: 6 Line: 7

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ; Line: 7

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: keyword Lexeme: endif Line: 8

Test Case 2: (< 20 lines)

Code (16 lines):

function conver[fahr:int]

{

return 5\*(fahr-32)/9;

}

@@

int low, high, step;

@@

read(low, high, step);

while(low<high)

{

write(low);

write(convert[low]);

low := low + step;

}

Terminal Run:

Arts-MacBook-Pro:Syntax Analyzer Arty$ python3.4 Syntax\_Analyzer.py

Enter file you would like to open (type "quit" to exit): testcase2.txt

Lexer working...

File open!

...Lexer complete!

Your Tokens and Lexemes have been saved as testcase2.RAT in the working directory.

Syntax Analyser running...

...Syntax Analyser finished!

There were no errors!

Your syntactic analysis of testcase2 has been saved as testcase2.SA in the working directory.

Would you like to process another file? (yes/no): no

Goodbye!

Contents of testcase2.SA:

Token: keyword Lexeme: function Line: 1

<Rat15S> ::= <Opt Function Definitions> @@ <Opt Declaration List> @@ <Statement List>

<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: conver Line: 1

Token: separator Lexeme: [ Line: 1

Token: identifier Lexeme: fahr Line: 1

<Opt Parameter List> ::= <Parameter List> | <Empty>

<Parameter List> ::= <Parameter> | <Parameter> , <Parameter List>

<Parameter> ::= < IDs > : <Qualifier>

Token: separator Lexeme: : Line: 1

Token: keyword Lexeme: int Line: 1

<Qualifier> ::= int | boolean | real

Token: separator Lexeme: ] Line: 1

Token: separator Lexeme: { Line: 2

<Opt Declaration List> ::= <Declaration List> | <Empty>

<Empty> ::= epsilon

<Body> ::= { < Statement List> }

Token: keyword Lexeme: return Line: 3

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Return> ::= return ; | return <Expression> ;

Token: integer Lexeme: 5 Line: 3

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: \* Line: 3

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

Token: separator Lexeme: ( Line: 3

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: identifier Lexeme: fahr Line: 3

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: - Line: 3

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

Token: integer Lexeme: 32 Line: 3

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 3

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: operator Lexeme: / Line: 3

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

Token: integer Lexeme: 9 Line: 3

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ; Line: 3

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: } Line: 4

Token: operator Lexeme: @@ Line: 6

Token: keyword Lexeme: int Line: 7

<Opt Declaration List> ::= <Declaration List> | <Empty>

<Declaration List> := <Declaration> ; | <Declaration> ; <Declaration List>

<Declaration> ::= <Qualifier> <IDs>

<Qualifier> ::= int | boolean | real

Token: identifier Lexeme: low Line: 7

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: , Line: 7

Token: identifier Lexeme: high Line: 7

Token: separator Lexeme: , Line: 7

Token: identifier Lexeme: step Line: 7

Token: separator Lexeme: ; Line: 7

Token: operator Lexeme: @@ Line: 9

Token: keyword Lexeme: read Line: 10

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Read> ::= read ( <IDs> );

Token: separator Lexeme: ( Line: 10

Token: identifier Lexeme: low Line: 10

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: , Line: 10

Token: identifier Lexeme: high Line: 10

Token: separator Lexeme: , Line: 10

Token: identifier Lexeme: step Line: 10

Token: separator Lexeme: ) Line: 10

Token: separator Lexeme: ; Line: 10

Token: keyword Lexeme: while Line: 11

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<While> ::= while ( <Condition> ) <Statement>

Token: separator Lexeme: ( Line: 11

Token: identifier Lexeme: low Line: 11

<Condition> ::= <Expression> <Relop> <Expression>

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: < Line: 11

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

<Relop> ::= = | != | > | < | => | <=

Token: identifier Lexeme: high Line: 11

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 11

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: { Line: 12

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Compound> ::= { <Statement List> }

Token: keyword Lexeme: write Line: 13

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Write> ::= write ( <Expression>);

Token: separator Lexeme: ( Line: 13

Token: identifier Lexeme: low Line: 13

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 13

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: ; Line: 13

Token: keyword Lexeme: write Line: 14

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Write> ::= write ( <Expression>);

Token: separator Lexeme: ( Line: 14

Token: identifier Lexeme: convert Line: 14

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: [ Line: 14

Token: identifier Lexeme: low Line: 14

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: ] Line: 14

Token: separator Lexeme: ) Line: 14

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: ; Line: 14

Token: identifier Lexeme: low Line: 15

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Assign> ::= <Identifier> := <Expression> ;

Token: operator Lexeme: := Line: 15

Token: identifier Lexeme: low Line: 15

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: + Line: 15

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

Token: identifier Lexeme: step Line: 15

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ; Line: 15

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: } Line: 16

Test Case 3: (> 20 lines)

Code (23 lines):

function helloWorld[]

{

read(fileNamedSue);

write(fileNamedSally);

}

@@

real who, what, where;

@@

while(who > where)

{

write(2+2);

if (what < who)

{

write(who\_what\_where\_why);

}

endif

}

Terminal Run:

Arts-MacBook-Pro:Syntax Analyzer Arty$ python3.4 Syntax\_Analyzer.py

Enter file you would like to open (type "quit" to exit): testcase3.txt

Lexer working...

File open!

...Lexer complete!

Your Tokens and Lexemes have been saved as testcase3.RAT in the working directory.

Syntax Analyser running...

...Syntax Analyser finished!

There were no errors!

Your syntactic analysis of testcase3 has been saved as testcase3.SA in the working directory.

Would you like to process another file? (yes/no): no

Goodbye!

Contents of testcase3.SA:

Token: keyword Lexeme: function Line: 1

<Rat15S> ::= <Opt Function Definitions> @@ <Opt Declaration List> @@ <Statement List>

<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: helloWorld Line: 1

Token: separator Lexeme: [ Line: 1

Token: separator Lexeme: ] Line: 1

<Opt Parameter List> ::= <Parameter List> | <Empty>

<Empty> ::= epsilon

Token: separator Lexeme: { Line: 2

<Opt Declaration List> ::= <Declaration List> | <Empty>

<Empty> ::= epsilon

<Body> ::= { < Statement List> }

Token: keyword Lexeme: read Line: 3

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Read> ::= read ( <IDs> );

Token: separator Lexeme: ( Line: 3

Token: identifier Lexeme: fileNamedSue Line: 3

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: ) Line: 3

Token: separator Lexeme: ; Line: 3

Token: keyword Lexeme: write Line: 5

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Write> ::= write ( <Expression>);

Token: separator Lexeme: ( Line: 5

Token: identifier Lexeme: fileNamedSally Line: 5

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 5

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: ; Line: 5

Token: separator Lexeme: } Line: 6

Token: operator Lexeme: @@ Line: 8

Token: keyword Lexeme: real Line: 10

<Opt Declaration List> ::= <Declaration List> | <Empty>

<Declaration List> := <Declaration> ; | <Declaration> ; <Declaration List>

<Declaration> ::= <Qualifier> <IDs>

<Qualifier> ::= int | boolean | real

Token: identifier Lexeme: who Line: 10

<IDs> ::= <Identifier> | <Identifier>, <IDs>

Token: separator Lexeme: , Line: 10

Token: identifier Lexeme: what Line: 10

Token: separator Lexeme: , Line: 10

Token: identifier Lexeme: where Line: 10

Token: separator Lexeme: ; Line: 10

Token: operator Lexeme: @@ Line: 12

Token: keyword Lexeme: while Line: 14

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<While> ::= while ( <Condition> ) <Statement>

Token: separator Lexeme: ( Line: 14

Token: identifier Lexeme: who Line: 14

<Condition> ::= <Expression> <Relop> <Expression>

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: > Line: 14

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

<Relop> ::= = | != | > | < | => | <=

Token: identifier Lexeme: where Line: 14

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 14

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: { Line: 15

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Compound> ::= { <Statement List> }

Token: keyword Lexeme: write Line: 17

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Write> ::= write ( <Expression>);

Token: separator Lexeme: ( Line: 17

Token: integer Lexeme: 2 Line: 17

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: + Line: 17

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

Token: integer Lexeme: 2 Line: 17

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 17

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: ; Line: 17

Token: keyword Lexeme: if Line: 18

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<If> ::= if ( <Condition> ) <Statement > <ifPrime>

Token: separator Lexeme: ( Line: 18

Token: identifier Lexeme: what Line: 18

<Condition> ::= <Expression> <Relop> <Expression>

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: operator Lexeme: < Line: 18

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

<Relop> ::= = | != | > | < | => | <=

Token: identifier Lexeme: who Line: 18

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 18

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: { Line: 19

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Compound> ::= { <Statement List> }

Token: keyword Lexeme: write Line: 20

<Statement List> ::= <Statement> | <Statement> <Statement List>

<Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Write> | <Read> | <While>

<Write> ::= write ( <Expression>);

Token: separator Lexeme: ( Line: 20

Token: identifier Lexeme: who\_what\_where\_why Line: 20

<Expression> ::= <Term> <ExpressionPrime>

<Term> ::= <Factor> <TermPrime>

<Factor> ::= - <Primary> | <Primary>

<Primary> ::= <Identifier> | <Integer> | <Identifier> [<IDs>] | ( <Expression> ) | <Real> | true | false

Token: separator Lexeme: ) Line: 20

<TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <empty>

<Empty> ::= epsilon

<ExpressionPrime> ::= + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <empty>

<Empty> ::= epsilon

Token: separator Lexeme: ; Line: 20

Token: separator Lexeme: } Line: 21

Token: keyword Lexeme: endif Line: 22

<ifPrime> ::= endif | else <Statement> endif

Token: separator Lexeme: } Line: 23