**COVER PAGE**

CPSC 323, Project 2

1. Reza Nikoopour

Eric Roe

1. Assignment Number - 2
2. Due Date – 4/7/2015
3. Turn-in Date – 4/7/2015
4. Executable File Name – recursive\_decent\_parser.py
5. Lab Room – 202 #2d
6. OS – Mac OS X \* any computer running Mac OS X should work

GRADE:

COMMENTS:

**Problem Statement**

The goal of this project is to create a program serving as a syntax analyzer for the Rat15S language. It can read a file containing tokens from the language and generate a parse tree. The format of the recursive decent parser comes from the Rat15S language’s grammar, which has been modified to remove left hand recursion. The program must parse through the tokens generated by the lexical analyzer and print the token and lexeme along with their associated production rules. Should any syntax errors be found while parsing, the program generates an error message to the user. It parses the entire Rat15S source code file and displays the parse tree to the user. The results are also written to a new file. The rest of the project involves having proper documentation, like this paper.

**Using the Program**

1. Open terminal
2. Navigate to Lib folder
3. Execute: python recursive\_decent\_parser.py <Input\_File> <Output\_File>

**Program Design**

The program was written in Python 3.

It utilizes recursive calls to go through the entire grammar. By checking the incoming tokens in order, they can be checked and filtered recursively through the production rule functions until completion. Should the next incoming token not fit the next function call, the program will produce a meaningful error message with another function call.

The functions are mainly in reverse order of their appearance in the grammar. That is, the last production rules in the grammar are the first functions in the program. This is due to the nature of python, which does not allow functions to call other functions if the other functions are not defined first.

Number of Production Rule Functions: 30

Names: Empty, Qualifier, IDs, Declaration, DeclarationList, Read, Expression, ExpressionPrime, Term, TermPrime, Factor, Write, Compound, Relop, Condition, If, Return, While, Statement, Assignment, Body, Function, FunctionDefinitions, Rat15S, Parameter, ParameterList, OptParameterList

General Function Format:  
Check the value of current token index in the TOKENS array, print the production rule using the PrintParseInfo function, and print the information needed for the current token using the PrintCurrentTokenInfo function. The right hand side of the production rule is handled by function calls for each of those production rules. The index of in TOKENS is incremented as needed.

All printings of productions rules are done through a PrintParseInfo function, which takes two strings as arguments and prints them using correct formatting. This allows for the print function to be toggled on or off based on what is needed.

Errors are handled by the SayErrorAndDie function, which prints a meaningful error with a description, line number, token type, and the lexeme. The program prints the error message, notifies the user that it failed to parse the entire file due to an error, and exits.

The “main” function, called Parser, checks the input file, generates tokens with the Lexer function, and puts the token objects into the TOKENS array. It calls the first function, Rat15S, to start the RDP.

Recap:

Token objects are stored in the TOKENS array. The program calls the first function, which recursively calls the rest of the production rules as it encounters the tokens. Any token which doesn’t fit the grammar indicates a syntax error and produces an error message based on the current position. If errorless, the program finishes with everything printed and saved to a file.

**Limitations**

It depends on the Token object format from the lexicalanalyzer.py file. But we assume everyone has the same limitation, so None.

**Shortcomings**

None

**Less Than 10 Example Input:**

|  |
| --- |
| @@  @@ |
| a := MyFavoriteNumber[4]; |
| if (2 > 3) |
| { |
| a := 3; |
| } |
| endif |
| a := test[ha]; |

**Less Than 10 Example Output:**

|  |
| --- |
| Token: Separator Lexeme: @@  <Opt Function Definitions> -> <Empty> |
| <Empty> -> epsilon |
| Token: Separator Lexeme: @@ |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| Token: Identifier Lexeme: a |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: MyFavoriteNumber |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Integer Lexeme: 4 |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Keyword Lexeme: if |
| <Statement List> -> <Statement> |
| <Statement> -> <If> |
| <If> -> if ( <Condition> ) <Statement> endif |
| Token: Separator Lexeme: ( |
| <Condition> -> <Expression> <Relop> <Expression> |
| Token: Integer Lexeme: 2 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Relop Lexeme: > |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| <Relop> -> > |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: { |
| <Statement> -> <Compound> |
| <Compound> -> { <Statement List> } |
| Token: Identifier Lexeme: a |
| <Statement List> -> <Statement> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Integer Lexeme: 3 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Separator Lexeme: } |
| Token: Keyword Lexeme: endif |
| <Statement List> -> <Statement> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: test |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: ha |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |

**Less Than 20 Example Input:**

|  |
| --- |
| /\*  This is a multi line comment |
| Its kind of cute =) |
| \*/ |
|  |
| function MyFavoriteNumber[] |
| { |
| write(4); |
| } |
| function YourFavoriteNumber[num:int] |
| { |
| write(num); |
| } |
| @@ |
| int your\_fav, where\_to\_start, when\_to\_end; /\* Declarations \*/ |
| boolean test; |
| @@ |
| your\_fav := MyFavoriteNumber[123]; |
| where\_to\_start := YourFavoriteNumber[your\_fav]; |

**Less Than 20 Example Output:**

Token: Keyword Lexeme: function

|  |
| --- |
| <Opt Function Definitions> -> <Function Definitions> |
| <Function Definitions> -> <Function> <Function Definitions> |
| <Function> -> function <Identifier> [ <Opt Parameter List ] <Opt Declaration List> <Body> |
| Token: Identifier Lexeme: MyFavoriteNumber |
| Token: Separator Lexeme: [ |
| Token: Separator Lexeme: ] |
| <Opt Parameter List> -> <Empty> |
| <Empty> -> epsilon |
| Token: Separator Lexeme: { |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| <Body> -> { <Statement List> } |
| Token: Keyword Lexeme: write |
| <Statement List> -> <Statement> |
| <Statement> -> <Write> |
| <Write> -> write ( <Expression> ); |
| Token: Separator Lexeme: ( |
| Token: Integer Lexeme: 4 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Keyword Lexeme: function |
| <Function Definitions> -> <Function> |
| <Function> -> function <Identifier> [ <Opt Parameter List ] <Opt Declaration List> <Body> |
| Token: Identifier Lexeme: YourFavoriteNumber |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: num |
| <Opt Parameter List> -> <Parameter List> |
| <Parameter List> -> <Parameter> |
| <Parameter> -> <IDs> : <Qualifier> |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: : |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: { |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| <Body> -> { <Statement List> } |
| Token: Keyword Lexeme: write |
| <Statement List> -> <Statement> |
| <Statement> -> <Write> |
| <Write> -> write ( <Expression> ); |
| Token: Separator Lexeme: ( |
| Token: Identifier Lexeme: num |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Separator Lexeme: @@ |
| Token: Keyword Lexeme: int |
| <Opt Declaration List> -> <Declaration List> |
| <Declaration List> -> <Declaration>; <Decalartion List> |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Identifier Lexeme: your\_fav |
| <IDs> -> <Identifier>, <IDs> |
| Token: Separator Lexeme: , |
| Token: Identifier Lexeme: where\_to\_start |
| <IDs> -> <Identifier>, <IDs> |
| Token: Separator Lexeme: , |
| Token: Identifier Lexeme: when\_to\_end |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ; |
| <Declaration List> -> <Declaration>; |
| Token: Keyword Lexeme: boolean |
| <Qualifier> -> boolean |
| Token: Identifier Lexeme: test |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ; |
| Token: Separator Lexeme: @@ |
| Token: Identifier Lexeme: your\_fav |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: MyFavoriteNumber |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Integer Lexeme: 123 |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: where\_to\_start |
| <Statement List> -> <Statement> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: YourFavoriteNumber |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: your\_fav |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |

**Greater Than 20 Example Input:**

|  |
| --- |
| /\* |
| This is a multi line comment |
| Its kind of cute =) |
| \*/ |
|  |
| function MyFavoriteNumber[] |
| { |
| write(4); |
| } |
| function YourFavoriteNumber[num:int] |
| { |
| write(num); |
| } |
| function ThisWillMakeThingsOverTwentyLines[start:int, end:int] |
| { |
| while(start < end) |
| { |
| write(start); |
| start := start + 1; |
| } |
| return start; |
| } |
| @@ |
| int your\_fav, where\_to\_start, when\_to\_end; /\* Declarations \*/ |
| @@ |
| your\_fav := MyFavoriteNumber[hahaha]; |
| read(your\_fav); |
| when\_to\_start := YourFavoriteNumber[your\_fav]; |
| where\_to\_start := 0; |
| when\_to\_end := 5; |
| when\_to\_end := ThisWillMakeThingsOverTwentyLines[where\_to\_start, where\_to\_end]; |

**Greater Than 20 Example Output:**

Token: Keyword Lexeme: function

|  |
| --- |
| <Opt Function Definitions> -> <Function Definitions> |
| <Function Definitions> -> <Function> <Function Definitions> |
| <Function> -> function <Identifier> [ <Opt Parameter List ] <Opt Declaration List> <Body> |
| Token: Identifier Lexeme: MyFavoriteNumber |
| Token: Separator Lexeme: [ |
| Token: Separator Lexeme: ] |
| <Opt Parameter List> -> <Empty> |
| <Empty> -> epsilon |
| Token: Separator Lexeme: { |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| <Body> -> { <Statement List> } |
| Token: Keyword Lexeme: write |
| <Statement List> -> <Statement> |
| <Statement> -> <Write> |
| <Write> -> write ( <Expression> ); |
| Token: Separator Lexeme: ( |
| Token: Integer Lexeme: 4 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Keyword Lexeme: function |
| <Function Definitions> -> <Function> <Function Definitions> |
| <Function> -> function <Identifier> [ <Opt Parameter List ] <Opt Declaration List> <Body> |
| Token: Identifier Lexeme: YourFavoriteNumber |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: num |
| <Opt Parameter List> -> <Parameter List> |
| <Parameter List> -> <Parameter> |
| <Parameter> -> <IDs> : <Qualifier> |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: : |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: { |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| <Body> -> { <Statement List> } |
| Token: Keyword Lexeme: write |
| <Statement List> -> <Statement> |
| <Statement> -> <Write> |
| <Write> -> write ( <Expression> ); |
| Token: Separator Lexeme: ( |
| Token: Identifier Lexeme: num |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Keyword Lexeme: function |
| <Function Definitions> -> <Function> |
| <Function> -> function <Identifier> [ <Opt Parameter List ] <Opt Declaration List> <Body> |
| Token: Identifier Lexeme: ThisWillMakeThingsOverTwentyLines |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: start |
| <Opt Parameter List> -> <Parameter List> |
| <Parameter List> -> <Parameter>, <Parameter List> |
| <Parameter> -> <IDs> : <Qualifier> |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: : |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Separator Lexeme: , |
| <Parameter List> -> <Parameter> |
| <Parameter> -> <IDs> : <Qualifier> |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: : |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: { |
| <Opt Declaration List> -> <Empty> |
| <Empty> -> epsilon |
| <Body> -> { <Statement List> } |
| Token: Keyword Lexeme: while |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <While> |
| Token: Keyword Lexeme: while |
| Token: Separator Lexeme: ( |
| Token: Identifier Lexeme: start |
| <Condition> -> <Expression> <Relop> <Expression> |
| Token: Identifier Lexeme: start |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Relop Lexeme: < |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| <Relop> -> < |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: { |
| <Statement> -> <Compound> |
| <Compound> -> { <Statement List> } |
| Token: Keyword Lexeme: write |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Write> |
| <Write> -> write ( <Expression> ); |
| Token: Separator Lexeme: ( |
| Token: Identifier Lexeme: start |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Separator Lexeme: ) |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: start |
| <Statement List> -> <Statement> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: start |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Operator Lexeme: + |
| <Term Prime> -> epsilon |
| <Expression Prime> -> + <Term> <Expression Prime> |
| Token: Integer Lexeme: 1 |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Separator Lexeme: } |
| Token: Keyword Lexeme: return |
| Token: Keyword Lexeme: return |
| <Statement List> -> <Statement> |
| <Statement> -> <Return> |
| Token: Keyword Lexeme: return |
| <Return> -> return <Expression>; |
| Token: Identifier Lexeme: start |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: @@ |
| Token: Keyword Lexeme: int |
| <Opt Declaration List> -> <Declaration List> |
| <Declaration List> -> <Declaration>; |
| Token: Keyword Lexeme: int |
| <Qualifier> -> int |
| Token: Identifier Lexeme: your\_fav |
| <IDs> -> <Identifier>, <IDs> |
| Token: Separator Lexeme: , |
| Token: Identifier Lexeme: where\_to\_start |
| <IDs> -> <Identifier>, <IDs> |
| Token: Separator Lexeme: , |
| Token: Identifier Lexeme: when\_to\_end |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ; |
| Token: Separator Lexeme: @@ |
| Token: Identifier Lexeme: your\_fav |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: MyFavoriteNumber |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: hahaha |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Keyword Lexeme: read |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Read> |
| <Read> -> read ( <IDs> ); |
| Token: Separator Lexeme: ( |
| Token: Identifier Lexeme: your\_fav |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ) |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: when\_to\_start |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: YourFavoriteNumber |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: your\_fav |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: where\_to\_start |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Integer Lexeme: 0 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: when\_to\_end |
| <Statement List> -> <Statement> <Statement List> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Integer Lexeme: 5 |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Integer> |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |
| Token: Identifier Lexeme: when\_to\_end |
| <Statement List> -> <Statement> |
| <Statement> -> <Assignment> |
| <Assignment> -> <Identifier> := <Expression>; |
| Token: Operator Lexeme: := |
| Token: Identifier Lexeme: ThisWillMakeThingsOverTwentyLines |
| <Expression> -> <Term> <Expression Prime> |
| <Term> -> <Factor> <Term Prime> |
| <Factor> -> <Identifier> [ <IDs> ] |
| Token: Separator Lexeme: [ |
| Token: Identifier Lexeme: where\_to\_start |
| <IDs> -> <Identifier>, <IDs> |
| Token: Separator Lexeme: , |
| Token: Identifier Lexeme: where\_to\_end |
| <IDs> -> <Identifier> |
| Token: Separator Lexeme: ] |
| Token: Separator Lexeme: ; |
| <Term Prime> -> epsilon |
| <Expression Prime> -> epsilon |
| Token: Separator Lexeme: ; |