CPSC 323 Project #2

Sameera Yayavaram (sameera.yayavaram@csu.fullerton.edu)
Joseph Chau (flyballer@csu.fullerton.edu)
Xiaome Ji (xiaomeiji@csu.fullerton.edu)

Problem Statement

We created a syntax analyser using a top down parser, RDP. The procedure returns the tokens and lexemes and the production rules used in the entire program. If the text file has a syntactical error, the analyser throws an exception.

How to use your program

The procedure prompts the user for input and output file with the Rat18S source code. If the compiler does not find the file, then use full file path.

Examples:
Input path:
/Users/Stark/Documents/GitHub/lexer/LA/test2.txt

Output path:
/Users/Stark/Documents/GitHub/lexer/LA/test2_output.txt

The procedure, then, identifies each lexeme and its token from the input file and production rule stores the results in the output file.

Design of the program

Each production rule after eliminating left recursion and rewriting the grammar has its own function. The parser calls the first rule Rat18S to start the program off and then the functions call themselves. The lexer is used to get all the tokens and the lexemes and the parser is used to see if the program is syntactically correct.
The main data structure we use is a vector to store all the tokens from the program so that we are able to traverse through the entire text file. The vector allows us to add tokens without defining the size of the array at the beginning.

Any limitations

None

Any shortcomings

The tokens are printed twice when a prime function is encountered. The program also cannot make it past the Function production rule as the Body function always throws an error.

CPSC 323 Project 2 Rules

1. <Rat18S> → <Opt Function Definitions> %% <Opt Declaration List> <Statement List>
2. <Opt Function Definitions> → <Function Definitions> | <Empty>
3. <Function Definitions> → <Function> <Function Definitions'>
4. <Function Definitions'> → <Function Definitions> | ε
5. <Function> → function <Identifier> [<Opt Parameter List>] <Opt Declaration List> <Body>
6. <Opt Parameter List> → <Parameter List> | <Empty>
7. <Parameter List> → <Parameter> <Parameter List'>
8. <Parameter List'> → , <Parameter List> | ε
9. <Parameter> → <IDs> : <Qualifier>
10. <Qualifier> → int | boolean | real
11. <Body> → { <Statement List> }
12. <Opt Declaration List> → <Declaration List> | <Empty>
13. <Declaration List> → <Declaration> ; <Declaration List'>
14. <Declaration List'> → <Declaration List> | ε
15. <Declaration> → <Qualifier> <IDs>
16. <IDs> → <Identifier> <IDs'>
17. <IDs'> → , <IDs> | ε
18. <Statement List> → <Statement> <Statement List'>
19. <Statement List'> → <Statement List> | ε
20. <Statement> → <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
21. <Compound> → { <Statement List> }
22. <Assign> → <Identifier> = <Expressions> ;
23. <If> → if (<Condition>) <Statement> <If'> endif
24. <If'> → else <Statement> | ε
25. <Return> → return <Return'> ;
26. <Return'> → <Expression> | ε
27. <Print> → put (<Expression>);
28. <Scan> → get(<IDs>);
29. <While> → while (<Condition>) <Statement>
30. <Condition> → <Expression> <Relop> <Expression>
31. <Relop> → == | ^= | > | < | => | =<
32. <Expression> → <Term> <Expression'>
33. <Expression'> → + <Term> <Expression'> | - <Term> <Expression'> | ε
34. <Term> → <Factor> <Term'>
35. <Term'> → * <Factor> <Term'> | / <Factor> <Term'> | ε
36. <Factor> → - <Primary> | <Primary>
37. <Primary> → <Identifier> | <Integer> | <Identifier> (<IDs>) | (<Expression>) | <Real> | true | false
38. <Empty> → ε