COVER PAGE

CS323 Programming Assignments

Fill out all entries 1 - 7. If not, there will be deductions!

1. Names [1. Andy Do Nguyen] Section [2]

   [2. Citlali Cortes Garcia] Section [2]

   [3. Andres Ancira] Section [2]

2. Assignment Number [2]

3. Due Date [11/9]

4. Submission Date [11/8]

5. Executable File name [assignment2_main.exe]
(A file that can be executed without compilation by the instructor, such as .exe, .jar, etc - NOT a source file such as .cpp)

6. Names of the testcase files - input test file output test file

test 1. [	test1.rat25] [	test1.out]
test 2. [	test2.rat25] [	test2.out]
test 3. [	test3.rat25] [	test3.out]

7. Operating System [Window11]
(Window – preferred)

To be filled out by the Instructor:

Comments and Grade:

CS323 Documentation

1. Problem Statement

Rewrite the given rules for Rat25 to remove any left recursion and backtracking. Using the previously written lexer function in assignment1, get tokens so that the parser can use it to create a parse tree using RDP. Also create error handling to show syntax errors with a meaningful error message including token, lexeme, line number, and error type.

2. How to use your program

Steps on how to run our program:
note: **if you want to use your own test case make sure to have it be in the same directory as the .exe**
1) double click the assignment2_main.exe
2) a terminal will popup then you can type in a file name with a .rat25 file extension
3) a confirmation message will popup telling you if it passed (no error in code) or failed (code has error in it)
4) exit the terminal by typing in exit or ctrl+d
5) open output (.out) files to check outputs

example output from test1.rat25:

If Failed:
Parser error at line 3: Unexpected token 'x' of type IDENTIFIER. Expected ';' after declaration

If Passed:
<Rat25F> ::= <Opt Function Definitions> # <Opt Declaration List> <Statement List> #
<Opt Function Definitions> ::= <Empty>
Token: Separator, Lexeme: #
<Opt Declaration List> ::= <Empty>
<Statement List> ::= <Statement> <Statement List'>
<Statement> ::= <Assign>
<Assign> ::= <Identifier> = <Expression> ;
Token: Identifier, Lexeme: a
Token: Operator, Lexeme: =
<Expression> ::= <Term> <Expression'>
<Term> ::= <Factor> <Term'>
<Factor> ::= <Primary>
<Primary> ::= <Identifier> <Primary_Tail>
Token: Identifier, Lexeme: b
<Primary_Tail> ::= <Empty>
<Term'> ::= <Empty>
Token: Operator, Lexeme: +
<Expression'> ::= + <Term> <Expression'>

<Term> ::= <Factor> <Term'>
<Factor> ::= <Primary>
<Primary> ::= <Identifier> <Primary_Tail>
Token: Identifier, Lexeme: c
<Primary_Tail> ::= <Empty>
<Term'> ::= <Empty>
<Expression'> ::= <Empty>
Token: Separator, Lexeme: ;
<Statement List'> ::= <Empty>
Token: Separator, Lexeme: #

3. Design of your program

We want to use RDP to create our parser, to do that we had to remove all cases of left recursion and backtracking. An example of eliminating left recursion was on R25 and R26 adding in their prime counterparts. We also used left factoring to remove backtracking like on R18 and R19 by adding in their tail counterparts. Using these rules we were able to create a function for each rule.

R1. <Rat25F> ::= <Opt Function Definitions> # <Opt Declaration List> <Statement List> #
R2. <Opt Function Definitions> ::= <Function Definitions> | <Empty>
R3. <Function Definitions> ::= <Function> <Function Definitions'>
R3'. <Function Definitions'> ::= <Function> <Function Definitions'> | <Empty>
R4. <Function> ::= function <Identifier> (<Opt Parameter List>) <Opt Declaration List> <Body>
R5. <Opt Parameter List> ::= <Parameter List> | <Empty>
R6. <Parameter List> ::= <Parameter> <Parameter List'>
R6'. <Parameter List'> ::= , <Parameter> <Parameter List'> | <Empty>
R7. <Parameter> ::= <IDs> <Qualifier>
R8. <Qualifier> ::= integer | boolean | real
R9. <Body> ::= { <Statement List> }
R10. <Opt Declaration List> ::= <Declaration List> | <Empty>
R11. <Declaration List> ::= <Declaration> ; <Declaration List'>
R11'. <Declaration List'> ::= <Declaration> ; <Declaration List'> | <Empty>
R12. <Declaration> ::= <Qualifier> <IDs>
R13. <IDs> ::= <Identifier> <IDs'>
R13'. <IDs'> ::= , <Identifier> <IDs'> | <Empty>
R14. <Statement List> ::= <Statement> <Statement List'>
R14'. <Statement List'> ::= <Statement> <Statement List'> | <Empty>
R15. <Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
R16. <Compound> ::= { <Statement List> }
R17. <Assign> ::= <Identifier> = <Expression> ;
R18. <If> ::= if (<Condition>) <Statement> <If_Tail>
R18_Tail. <If_Tail> ::= fi | else <Statement> fi

R19. <Return> ::= return <Return_Tail>
R19_Tail. <Return_Tail> ::= ; | <Expression> ;
R20. <Print> ::= put (<Expression>);
R21. <Scan> ::= get (<IDs>);
R22. <While> ::= while (<Condition>) <Statement>
R23. <Condition> ::= <Expression> <Relop> <Expression>
R24. <Relop> ::= == | != | > | < | <= | =>
R25. <Expression> ::= <Term> <Expression'>
R25'. <Expression'> ::= + <Term> <Expression'> | - <Term> <Expression'> | <Empty>
R26. <Term> ::= <Factor> <Term'>
R26'. <Term'> ::= * <Factor> <Term'> | / <Factor> <Term'> | <Empty>
R27. <Factor> ::= - <Primary> | <Primary>
R28. <Primary> ::= <Identifier> <Primary_Tail> | <Integer> | (<Expression>) | <Real> | true | false
R28_Tail. <Primary_Tail> ::= (<IDs>) | <Empty>
R29. <Empty> ::= E

4. **Any limitation**

None

5. **Any shortcomings**

None