**Problem Statement**

Write a syntax analyzer that parses an entire program if it is syntactically correct. If a syntax error occurs, the parser should generate a meaningful error message such as token, lexeme, line number, and error type etc. The parser should print to an output file the tokens, lexemes, and the production rules used. The RAT20F grammar must be rewritten to perform any left-factorizations and remove any left-recursions.

**Usage**

1. Open Command Prompt (Windows) with a working directory containing the app and the three test files
2. Run the program with the first argument as the input file and the second argument as the output file

**Program Design**

Each production in the RAT20F grammar is implemented as their own function which are called by a previous production’s function. For example, production R2. <Opt Function Definitions> is called in the function for production R1. <Rat20F>. Some functions accept records in order to verify them either within themselves or within a child function. Some functions return records in order for the program to verify them in other production functions. Some check for lexemes within themselves and to streamline the code, we made a Lexeme\_Check function that calls the lexer and checks for the expected lexeme. If the program fails to find the expected lexeme, it calls the Syntax\_Error function. This function will print an error message stating that the program failed to find an expected lexeme and lexeme it received instead. It will then exit the program. To produce the desired output file, the lexer function was changed to print the token and the lexeme of every completed record it found.

**Limitations**

Unfortunately, the output file will be cramped and will occasionally not have any production rules under certain records as the production rules are printed at the beginning of each production function and the program often calls the lexer during the middle of functions.

**Shortcomings**

None

**Rewritten RAT20F Grammar**

R1. <Rat20F> ::= <Opt Function Definitions> $$ <OptDeclaration List> <Statement List> $$

R2. <Opt Function Definitions> ::= <Function Definitions> | <Empty>

**R3. <Function Definitions> ::= <Function> <Function Definitions>'**

**R4. <Function Definitions>' ::= <Function Definitions> | <Empty>**

R5. <Function> ::= function <Identifier> ( <Opt Parameter List> ) <Opt Declaration List> <Body>

R6. <Opt Parameter List> ::= <Parameter List> | <Empty>

**R7. <Parameter List> ::= <Parameter> <Parameter List>'**

**R8. <Parameter List>' ::= , <Parameter List> | <Empty>**

R9. <Parameter> ::= < IDs > <Qualifier>

R10. <Qualifier> ::= Int | boolean | real

R11. <Body> ::= { <Statement List> }

R12. <Opt Declaration List> ::= <Declaration List> | <Empty>

**R13. <Declaration List> ::= <Declaration> ; <Declaration List>'**

**R14. <Declaration List>' ::= <Declaration List> | <Empty>**

R15. <Declaration> ::= <Qualifier> <IDs>

**R16. <IDs> ::= <Identifier> <IDs>'**

**R17. <IDs>' ::= , <IDs> | <Empty>**

**R18. <Statement List> ::= <Statement> <Statement List>'**

**R19. <Statement List>' ::= <Statement> | <Empty>**

R20. <Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>

R21. <Compound> ::= { <Statement List> }

R22. <Assign> ::= <Identifier> = <Expression> ;

**R23. <If> ::= if ( <Condition> ) <Statement> <If>'**

**R24. <If>' ::= fi | else <Statement> fi**

**R25. <Return> ::= return <Return>'**

**R26. <Return>' ::= ; | <Expression> ;**

R27. <Print> ::= put ( <Expression> );

R28. <Scan> ::= get ( <IDs> );

R29. <While> ::= while ( <Condition> ) <Statement>

R30. <Condition> ::= <Expression> <Relop> <Expression>

R31. <Relop> ::= == | != | > | < | <= | =>

**R32. <Expression> ::= <Term> <Expression>'**

**R33. <Expression>' ::= + <Term> <Expression>' | - <Term> <Expression>' | <Empty>**

**R34. <Term> ::= <Factor> <Term>'**

**R35. <Term>' ::= \* <Factor> <Term>' | / <Factor> <Term>' | <Empty>**

R36. <Factor> ::= - <Primary> | <Primary>

**R37. <Primary> ::= <Identifier> <Primary>' | <Integer> | ( <Expression> ) | <Real> | true |**

**false**

**R38. <Primary>' ::= ( <IDs> ) | <Empty>**

R39. <Empty> ::= ε

Note: <Identifier>, <Integer>, <Real> are token types

**Removal of Left-Recursions**

Original: <Expression> ::= <Expression> + <Term> | <Expression> - <Term> | <Term>

α1 = +<Term>, α2 = -<Term>, δ = <Term>

<Expression> ::= <Term> <Expression>'

<Expression>' ::= + <Term> <Expression>' | - <Term> <Expression>' | ε

<Term> ::= <Term> \* <Factor> | <Term> / <Factor> | <Factor>

α1 = \*<Factor>, α2 = /<Factor>, δ = <Factor>

<Term> ::= <Factor> <Term>'

<Term>' ::= \* <Factor> <Term>' | / <Factor> <Term>' | ε

**Left-Factorizations**

Original: <Function Definitions> ::= <Function> | <Function> <Function Definitions>

<Function Definitions> ::= <Function> <Function Definitions>'

<Function Definitions>' ::= <Function Definitions> | ε

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

<Parameter List> ::= <Parameter> <Parameter List>'

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

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

<Declaration List> ::= <Declaration> ; <Declaration List>'

<Declaration List>' ::= <Declaration List> |

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

<IDs> ::= <Identifier> <IDs>'

<IDs>' ::= , <IDs> | <Empty>

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

<Statement List> ::= <Statement> <Statement List>'

<Statement List>' ::= <Statement> | <Empty>

Original: <If> ::= if ( <Condition> ) <Statement> fi | if ( <Condition> ) <Statement> else

<Statement> fi

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

<If>' ::= fi | else <Statement> fi

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

<Return> ::= return <Return>'

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

Original: <Primary> ::= <Identifier> | <Identifier> ( <IDs> ) | <Integer> | ( <Expression> ) | <Real>

| true | false

<Primary> ::= <Identifier> <Primary>' | <Integer> | ( <Expression> ) | <Real> | true |

false

<Primary>' ::= ( <IDs> ) | <Empty>