**CPSC 323 Documentation**

**Problem Statement**

The problem statement for this assignment is to build upon the previous assignment, which was where we wrote a lexical analyzer (lexer) that can run "Rat21SU", a conventional and easy-to-understand programming language. We printed the token and lexemes last time, but the goal this time is to print out the rules of Rat21SU to show the productions that occurred.

**How to Use the Program**

The project is written in C++. Below are the instructions on how to execute the program.

1. Download the compressed folder on your computer to the location of your choice.
2. Locate the downloaded file and extract the compressed folder.
3. Open the Linux Terminal.
4. Navigate to the directory where the files are located.
5. Navigate to the inside of the “.exe” directory.
6. Make sure that the three test case files are in the same directory.
7. Run the .exe file from the terminal.

To know if the program ran successfully, the results in the output files would show the correct tokens, lexemes, and production rules.

**Design of Your Program**

In this assignment, we updated the given syntax rules from assignment 1. In rules 16 and 16.5, 17 and 17.5, we removed the left-recursions. We have to remove the left-recursions because the parser does not know when to stop expanding. In rules 3 and 3.5, 6 and 6.5, and 10 and 10.5, we performed back-tracing. Back-tracing is the rephrasing of the same or previous tokens.

**Updated Syntax Rules**

**Left-recursion:** In rules 16 and 16.5, 17 and 17.5, we removed the left-recursion. We remove the left-recursion first by listing out the nonterminals in the sequence that they occur. For each nonterminal, if the right-hand side begins with a nonterminal, then substitute the A and remove any direct recursion. After, replace A and remove direct left-recursion.

For example, in rule 16 (original), the syntax rule was “<Expression> ::= <Expression> + <Term> | <Expression> - <Term> | <Term>”. The nonterminals were listed in the sequence it is expected to occur. We then removed any direct recursion, which brought us to the next step to revising the rule, “<Expression> ::= <Term><Expression>’”. Lastly, we remove the direct left-recursion and replace the “A”. The revised rule is “ <Expression>’ ::= + <Term><Expression>’| - <Term><Expression>’”.

**Back-tracing:** In rules 3.5, 6.5, and 10.5, we performed back-tracing on the preceding rules. We used left-factorization in which we factor out the same symbols of the right-hand side of the production for the same non-terminal.

For example, in rule 3 (original), the syntax rule was “<Declaration List> ::= <Declaration> ; | <Declaration> ; <Declaration List>”. We then rephrased the existing/previous tokens. The revised rule we get is “<BackTraceDeclarationList> ::= <declaration List> | <Empty>”.

**Syntax Rules**

R1. <Rat21SU> ::= %% <Opt Declaration List> <Statement List> %%

R2. <Opt Declaration List> ::= <Declaration List> | <Empty>

**R3. <Declaration List> ::= <Declaration> ; <BackTraceDeclarationlist>**

**R3.5 <BackTraceDeclarationList> ::= <Declaration List> | <Empty>**

R4. <Declaration> ::= <Qualifier > <identifier>

R5: <Qualifier> ::= integer | boolean

**R6. <Statement List> ::= <Statement> <BackTraceStatementList>**

**R6.5 <BackTraceStatementList> ::= <Statement List> | <Empty>**

R7. <Statement> ::= <Compound> | <Assign> | <If> | <Get> | <Put> | <While>

R8. <Compound> ::= begin <Statement List> end

R9. <Assign> ::= <Identifier> = <Expression> ;

**R10. <If> ::= if ( <Condition> ) <Statement >[elseState] endif**

**R10.5. [elseState] ::= else <Statement> | epsilon**

R11. <Put> ::= put ( <identifier> );

R12. <Get> ::= get ( <Identifier> );

R13. <While> ::= while ( <Condition> ) <Statement>

R14. <Condition> ::= <Expression> <Relop> <Expression>

R15. <Relop> ::= == | > | < | /=

**R16. <Expression> ::= <Term><ExpressionPrime>**

**R16.5 <ExpressionPrime> ::= + <Term><ExpressionPrime>| - <Term><ExpressionPrime> | <Empty>**

**R17. <Term> ::= <Factor><TermPrime>**

**R17.5. <TermPrime> ::= \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>**

R18. <Factor> ::= - <Primary> | <Primary>

R19. <Primary> ::= <Identifier> | <Integer> | ( <Expression> ) | true | false

R20. <Empty> ::= ε

**Any Limitation**

None

**Any Shortcomings**

None