**CS323 Assignment 2 Documentation**

1. **Problem Statement**

* Write a syntax analyzer using Top down parser
* Rewrite the grammar RAT17F to remove any left recursion and backtracking.
* Use the lexer(file) generated in the assignment 1 to get the tokens.
* Able to print out both tokens and lexemes and the production rules.
* Able to read a file.txt containing the source code of Rat17F to generate tokens and write out the results.
* Able to have error handling:
  + If a syntax error occurs, the parser print out error line number, token, lexeme, error type.
  + Exit program.
* Able to parse the entire program if it is syntactically correct.

1. **How to use your program**

Locate “Executable files” folder in the submission. We provide version for Windows OS. Follow below instruction to run the program

WINDOWS OS

* Put all file.txt files that you want to test into same folder with the executable file (i.e. Project2.exe in Windows folder)
* Run the program by double-click on “Project2.exe”
* Type txt file name and see the result on “output.txt” or on DOS/WINDOWS screen.

\*Important: Make sure you put the file.txt and Project2.exe in the same place to run the program successfully.

1. **Design of your program**

* Build a recursive descent parser (RDP) based on syntax rules and lexer.
* Remove 2 left recursion syntax rules and 8 backtracking syntax rules.
* For each syntax rule function, do the same procedure:
  + print out the production rule.
  + Following the production rule
    - If it starts with a non terminal, call non-terminal Function
    - If it starts with a terminal character, check if token matches. If it matches, call lexer to get the next token and check it if is a terminal or a non terminal.
    - If it misses the requirements, error handling occurs.
* For removing left recursion syntax rules functions,

**<Expression> ::= <Term> <Expression Prime>**

**<Expression Prime> ::= + <Term> <Expression Prime> | - <Term> <Expression Prime> | <Empty>**

**<Term> ::= <Factor> <Term Prime>**

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

* + Rewrite grammar to remove left recursion.
  + print out production rules
  + Following the production rules.
* For backtracking syntax rules,

**<Function Definitions> ::= <Function> < Function Definitions Prime>**

**<Function Definitions Prime> ::= <Function Definitions> | <Empty>**

**<Parameter List> ::= <Parameter> <Parameter List Prime>**

**<Parameter List Prime> ::= , <Parameter List> | <Empty>**

**<Declaration List> := <Declaration> ; <Declaration List Prime>**

**<Declaration List Prime> := <Declaration List> | <Empty>**

**<IDs> ::= <Identifier> <IDs Prime>**

**<IDs Prime> ::= , <IDs> | <Empty>**

**<Statement List> ::= <Statement> <Statement List Prime>**

**<Statement List Prime> ::= <Statement List> | <Empty>**

**<If> ::= if ( <Condition> ) <Statement> <If Prime>**

**<If Prime> ::= fi | else <Statement> fi**

**<Return> ::= return <Return Prime>**

**<Return Prime> ::= ; | <Expression> ;**

**<Primary> ::= <Identifier> <Primary Prime> | <Integer> | ( <Expression> ) | <Real> | true | false**

**<Primary Prime> ::= [<IDs>] | <Empty>**

* + Rewrite grammar to remove backtracking.
  + print out production rules
  + Following the production rules.
* program is written in C++11
* 5 main components of our programs: Par.cpp, Main.cpp, Par.h, Lex.h, Lex.cpp
* Build the class Par inherits all methods and member variables from Lex and call first function Rat17F(), then it calls other functions from top to the bottom recursively
* Base class: Lex
* Derived class: Par
* Par.h keeps the declarations of the following
  + private variables:
    - 1 bool: used for switch on or off
  + public methods (members):
    - 28 functions for syntax rules.
    - 8 functions to remove backtracking
    - 2 functions to remove left recursion
    - 1 function set switch to choose option
    - 1 function print errors to file
    - 1 function print to write out to the output file
    - 1 mutators which is setSwitch() to turn on/off syntax rules
    - 1 constructor
    - 1 destructor
* Par.cpp provides the definition of each functions above
* Main.cpp is used for running the program and read/write file.txt

1. **Any Limitation**

*None*

1. **Any shortcomings**

*None*