**CPSC 323**

**ASSIGNMENT 2 SYNTAX ANALYSIS**

1. **Problem Statement**

* Write a program call syntax analyzer to generate grammar of RAT17F language by using top down parser method.
* Using the lexical program from assignment one to recognize the tokens and parse those tokens into a right grammar.
* Remove left recursion and backtracking for every grammar rule.
* Requirement to print out token, token types and the grammar rule for each one of them.
* Most important part is the error handling that can use to check the grammar is correct or not.
* Error handling will show if the grammar correct then the program will keep checks the next tokens grammar otherwise it will be get terminate and exit.
* Error handling also give the line number of the syntax error and the correction for that.

1. **How to use the program**

Five Steps:

1. Locate the folder name exe from the submission zip file. And copy paste the test case into that folder.
2. Then click the executable file names RAT17F.
3. Type exactly the name of the test case txt (be more specific have to type all include .txt)
4. The result will only show on the terminal (no output.txt for this program)

*Note : If terminal has limited line show ( go ahead right click on the terminal par then -> change the height to 500 or 900)*

1. Using Windows OS for this program.

*Note: My code has too many lines so I design not do the ofstream for syntax analysis but I will provide one solution if user run my code on visual studio( on solution explorer -> right click on project name -> properties -> debugging -> command argument -> type “ dir > output.txt ” ) the output file will be show up on project files.*

1. **Designing for this program**

* This program is using top down parsing recursive descent.
* This parse will take all the tokens and look for prefix of rules when it find correct prefix then go to the next token to match the whole grammar rule.
* This program contains three files: phrase.h, prase.h, and main.cpp.
* Functions:

two vectors for all tokens and all errors

23 void functions for any grammar rule.

**left recursive:**

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

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

E-> TE’

E’ -> + TE’| -TE’| ε

**<Term> ::=  <Factor> <Term Prime>**

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

T-> FT’

T’ -> /FT’|\*FT’| **ε**

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

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

FD -> F FD’

FD’ -> F FD’ | **ε**

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

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

PL -> P PL’

PL’ -> , P PL’ | **ε**

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

DL->D ; DL’

DL’->D;DL’| ε

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

ID->i ID’

ID’->,i  ID’ | ε

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

SL->S SL’

SL’->S SL’| ε

**4. Limitation**

My code runs on x64 visual studio (any version)**.**( i think x86 has some problem with my code because it took a lot of memory to solve problem).

**5.Any shortcomings**

This code doesn’t have ofstream output.txt.