Lexical Rules

Each token should be separated by a space. It has identifiers that starts with a letter and followed by 0-z number of letters. The code will start and end with start and end.

* Declare: enter the variables and words needed and end with a ;.
* Assign: starts with a variable that’s in the stack. It will return an error if not. It will end with a ;.
* Conditions: Starts with cond with () and anything in there that will explain the procedure. Then it will have {} with any statements needed.
* Loops: start with a word from the list, then it will have () and {} just like the conditions types. It will try to run if it’s false.

Math: it will only be valid if it passes the regex. The numbers and tokens have to be separated by a space to be considered legible by both computer and person.

* 10 + 4
* 3 + numOne

Tokens:

* Math Operators
  + ADD: +
  + SUB: -
  + MUL: \*
  + DIV: /
  + MOD: %
  + OPENPAREN: ()
  + CLOSEPAREN: ()
* Mathematical Equators
  + LST: <
  + GRT: >
  + LSTE: <=
  + GRTE: >=
  + EQ: ==
  + NEQU: !=
* Integer
  + tiny: -128-127
  + small: -32768-32767
  + medium: -2147483648-2147483647
  + large: -9223372036854775808-9223372036854775808
* Keywords
  + var: [a-zA-z]{6,8}
  + cond: cond
  + redo: repeat
  + begin: begin
  + end: end
* Extras
  + assign: =
  + blockOpen: {
  + blockClose: }
  + parenOpen: (
  + parenClose: )
* Order of Non-Pemdas
  + (), %, +, -, \*, / (left to right)
* Production Rules
  + <file> 🡪 begin <statement\_block> end
  + <statement\_block> 🡪 {<file> ‘;’}
  + <while\_statement> 🡪 repeat ‘(‘ <bool> ‘)’ ‘{‘ <statement\_block> ‘}’
  + <as\_s> 🡪 <var> = <expression> ‘;’
  + <if\_statement> 🡪 or ‘(‘ <bool> ‘)’ ‘{‘ <statement\_block> ‘}’
  + <declare> 🡪 <type> <var> ‘;’
  + <type> 🡪 (tiny|small|medium|large)
  + <var> 🡪 [a-zA-Z] {6,8}
  + <math> 🡪 <op> { (‘\*’|’\’|’%’) <op>}
  + <op> 🡪 <op> {(‘+’|’-‘) <op>}
  + <factor> 🡪 [0-9] + | <var> | ‘(‘ <math> ‘)’
  + <bool> 🡪 <math> (‘<=’ | ‘>=’ | ‘<’ | ‘>’) <math>
* Grammar Rules
  + E -> E + T Expression + Term
  + E -> E - T Expression - Term
  + E -> T Term
  + T -> T \* F Term \* Expression
  + T -> T / F Term / Expression
  + T -> F Factor
  + F -> -F Negative Factor
  + F -> +F Positive Factor
  + F ->( E ) Expression inside parentheses
  + F -> a Constant

C

The code is LL grammar approved because \_\_\_\_\_\_\_\_

D

The code does not have ambiguous grammar because \_\_\_\_\_\_

G

1. 5 lexical errors

smart varOne;

varOne = 4 ADD OPENPAREN 1 ADD 4 CLOSEPAREN;

conds OPENPAREN varOne GTE 10 CLOSEPAREN {

varOne EQU 333;

}

Lagos varTwo;

varToo EQU varOne ADD 12;

* it’s small not smart
* conds is cond
* GTE is supposed to be GRTE
* Lagos is supposed to be large
* varToo is supposed to be varTwo

1. 5 syntax errors

small varOne;

varOne EQU 1SUB 2 ADD 3;

cond varOne LS TE 20) {

varOne EQU 12;

cond (varOne EQU 20) {

varOne = varOne ADD 2;

large varTwo;

varTwo = 1 SUB 2 MUL OPENPAREN 3 SUB 2 ;

}

* There needs to be a space between 1 and SUB
* LS and TE needs to have no space
* Needs a } after the second cond
* Needs a CLOSEPAREN at the end of the last line
* Needs an OPENPAREN after the first OPENPAREN
* EQU needs to be = at varOne = 12

1. No errors

tiny varOne;

varOne = 4;

large varTwo;

varTwo = 10 SUB 3 ADD OPENPAREN 3 SUB OPENPAREN 2 MUL 6 CLOSEPAREN CLOSEPAREN MUL varOne;

cond OPENPAREN varOne NEQU varTwo CLOSEPAREN {

cond OPENPAREN varOne LSTE varTwo CLOSEPAREN {

varOne = varTwo MUL 4;

}

}

small varThree;

varThree = 0;

REPEAT OPENPAREN varThree GRTE 0 CLOSEPAREN {

varThree = varThree ADD 1;

cond OPENPAREN varThree EQU 3 CLOSEPAREN {

varThree = varThree MUL 2;

}

}

medium varOne;

varOne = 4 SUB OPENPAREN -2 MUL 3 CLOSEPAREN MUL 4;

medium varTwo;

varTwo = 2 MUL varOne;

small iter;

iter = 3;

repeat OPENPAREN iter GRTE 3 CLOSEPAREN {

iter = iter ADD 1;

}

E -> E + T

E -> E - T

E -> T

T -> T \* F

T -> T / F

T -> F

F -> -F

F -> +F

F ->( E )

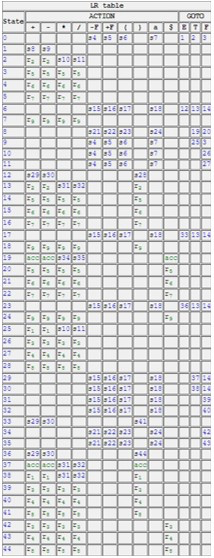
F -> a

H (Fails)

Graphical user interface, application

Description automatically generated with medium confidence Table

Description automatically generated

 Graphical user interface

Description automatically generated

Pass

Table

Description automatically generated 