PLC READ ME

The txt fille has comment about on what is wrong with the file. Also to change file, it is in main.

1 Define the rules for recognizing all lexemes as their proper token, and clearly define integer token codes for each token required for this language • Should have Regular Grammar, Regular Expression, or Finite Automat defined

```
Identifies [A-Za-z]{6,8}
Start \^
End \$
Int_literal [0-9]* (v|r|e)
Num#
Add_op +
Sub_op -
Mult_op *
Div_op /
Left_paren \(
Right_paren \)
Mod_op %
Left_brack \{
Right_brack ]
Less_than >
Less_than_equal >=
greater_than <</pre>
Greater_than_equal <=
If_code con
Loop loop
Noequal!
EqualTo ==
Semicolon;
```

2 Define production rules for implementing the mathematical syntax of operators and operands, loops, variable declaration, selection statements • Enforce a non PEMDAS (BODMAS) order of operation, must have at least 6 levels of precedence • Keywords cannot use the words while, for, do, if, int, short, long i. Keywords should be unique, if others share your same words, you may lose more points than this problem is worth • You must clearly state the structure of your language with production rules

```
<begin> -> Start <statement> {<statement>} End
<statement>-> <ifsmt> | <loop> | <expr>
<loop> -> loop'('<boolexpr>')' '{

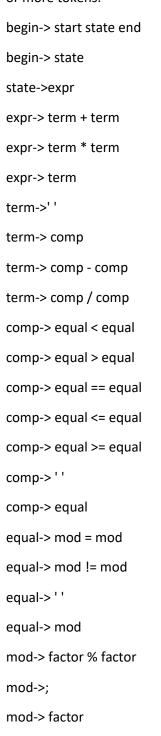
' <statement> '}' {<statement>}
<ifsmt>-> con'('<boolexpr>')''{'<statement>'}' {<statement>}
<boolexpr> -> identifer '('<|>|<=|>=|=!!')' int_literal
<expr> -> <term> {(+ | *) <term>}
<term> -> <comp> {(- | /) <comp>}
<comp>-> <eqaulity> {(<|>|<=|>=|==) <eqaulity>}
<eqaulity> -> <mod>{(=|!=)<mod>
<mod>-> <factor> {(%)<factor> |;}
<factor> -> identifes | int_literal | num ident idenfies semicolon | (<expr> ) | semicolon
```

```
3 Show whether every rule set in your language conforms to the standard of an LL Grammar
First, we would need to if it passes pairwise disjoint
<br/><begin> -> Start <statement> End | <statement>
First(begin )= Start {<ifsmt> | <loop> | <expr>}
= start { con | loop | identifes | int_literal | num ident | ( <expr> ) | semicolon
}
Passes since not have same values
<statement>-> <ifsmt> | <loop> | <expr>
First (statement) = con | loop | identifes | int | literal | num ident | ( <expr> ) | semicolon
Passes
<Loop> -> loop'('<boolexpr>')' '{'<statement>'}' <statement>
First(Loop)=loop
Passes
<ifsmt>-> con'('<boolexpr>')''{'<statement>'}' {<statement>}
First(ifsmt)=con
<boolexpr>-> identifer '('<|>|<=|>=|==|!')' int_literal
First(boolexpr)=identifier
<expr> -> <term> {(+ | *) <term>}
First(expr) -> <term>= identifes | int_literal | num ident | ( < | semicolon
<term> -> <comp> {(- | / ) <comp>}
First(term) -><comp>= identifes | int literal | num ident | (| semicolon
<comp>-> <eqaulity> {(<|>|<=|>=|==) <eqaulity>}
Firsrt(comp)=<equality>= identifes | int_literal | num ident | ( | semicolon
<eqaulity> -> <mod>{(= | !=) <mod>
First(equality)=<mod>= identifes | int_literal | num ident | ( | semicolon
<mod>-> <factor> {(%|;)<factor>}
First<mod>=<factor> -> identifes | int_literal | num ident | (| semicolon
<factor> -> identifes | int_literal | num ident idenfies semicolon | ( <expr>) | semicolon
```

First(factor) = <factor> -> identifes | int_literal | num ident | (| semicolon

Since all pass pairwise disjoint and there is no left recursive since nonterminal don't repeat the same nonterminal instead going to a different version and they all have an endpoint which would stop unnecessary recursion.

Create a LR (1) parse table for your language. And show the trace of 4 code samples. Each must have 6 or more tokens.



mod->''

factor-> id

factor-> num

factor->(expr)

factor-> numid

factor->;

factor-> factor + state

factor->end

state-> loop

loop-> Loop(boolexp){

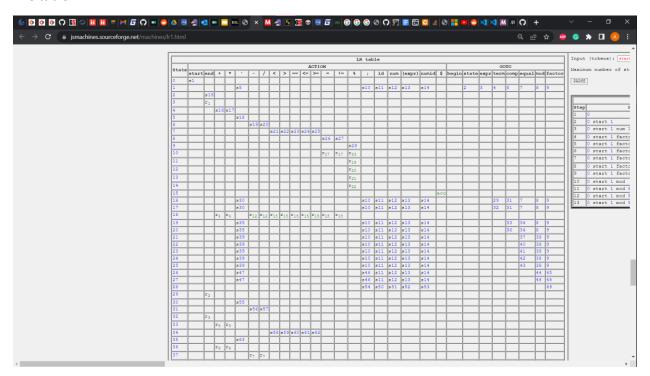
state} state

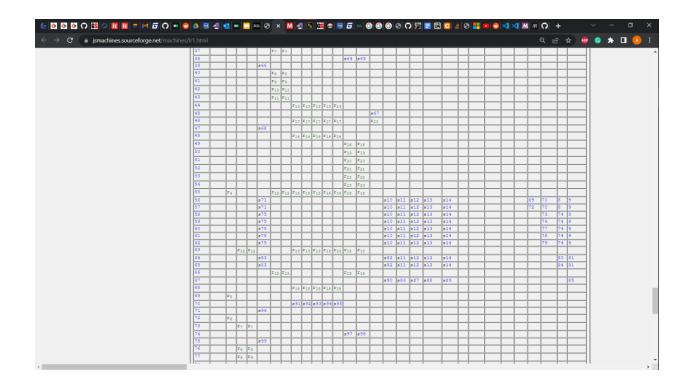
boolexp-> comp

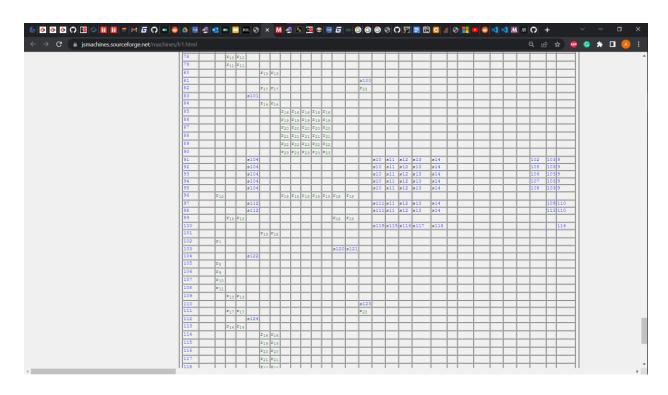
state-> ifsmt

ifsmt-> con(boolexp){ state } state

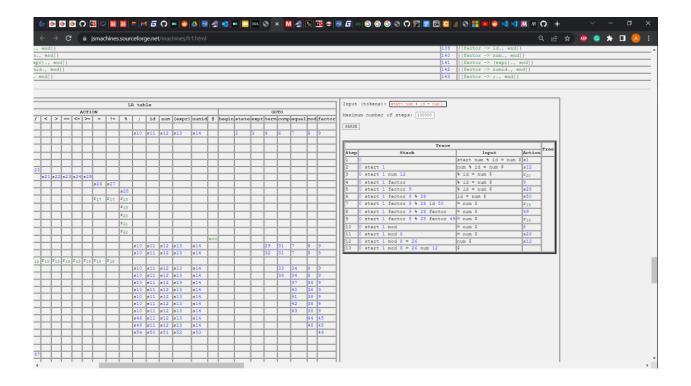
The table







et/machines/lr1.html																								Q 15	÷ 1	tr	⊕ ©	*	
105	ra														=	_				_							-11		
106	r ₉		-	+	+	H	+	₩	+	-	-	-	-				-	H	+	-	_	-	\vdash		H		41		
107	r ₁		-	+	-	-	+	\vdash	\rightarrow	-	-	-	-		-		-	₩	-	-	-	\neg			\vdash		41		
108	r ₁		-	+	-	₩	+	\rightarrow	\rightarrow	-	\rightarrow	\rightarrow	\rightarrow		-		-	₩	\rightarrow	-	-	\dashv	-		\vdash	_	41		
109	-1	r ₁₃	T	+	-	-	+	₩	+	-	-	-	-		-		-	₩	-	-	-	\dashv	-		H		41		
110	\vdash	-13	-13	+	-	-	+	\vdash	\rightarrow	-	-	123	\rightarrow		_	_	-	₩	\rightarrow	-	-	\neg			\vdash	_	41		
111		r ₁₇	r17	\pm	_	1		₩	\rightarrow	_		F23	\rightarrow	-				\vdash	\rightarrow	_	_	\neg			Ħ		1		
112		1	51	24	_	1		₩	\rightarrow	\rightarrow	_	-	\rightarrow	-i				\vdash	\rightarrow	_	_	\neg			H		1		
113		r14		\neg		\vdash	\neg	\vdash	\rightarrow	_	\neg	\neg	\rightarrow					\vdash	\rightarrow	\neg	$\overline{}$	\neg			\Box		11		
114				r ₁	6 F16	\vdash	\neg	\vdash	\rightarrow	7				-				\vdash	\rightarrow						\Box		1		
115			_		9 F19		\perp	\forall	\rightarrow	-								\vdash							\Box		1		
116					0 F20			\Box	1																		1		
117		\Box	7		1 r21		\rightarrow	\vdash	7																\Box		1		
118	\vdash	\vdash	\neg		2 F22		\rightarrow	⇈	\rightarrow	→	_	_	\rightarrow					\vdash	\rightarrow	_	_	\neg			\vdash		11		
119	\vdash	\vdash	\neg		3 F23		\neg	\vdash	\rightarrow	-	_	\rightarrow	\rightarrow					\vdash	\rightarrow	\neg	_	\neg			\Box		11		
120	\vdash	\vdash	s1			1	+	\vdash	\rightarrow	-	-		127	-11	s12	s13	814	\vdash	-	-	_	-			125	126	11		
121		\vdash	s1		\neg	\vdash	\neg	\vdash	十	-	\neg		127				s14	\vdash	\rightarrow	$\overline{}$	_	\neg			129		11		
122	rı	5		Ť			\neg	П	r	18 2	18	T	T	T)					T								11		
123				Ť					T			1	135	131	a132	e133	s134									130			
124		r18	r ₁₈	Т			\top	П	T	\neg	\neg	\neg	T												\Box		1		
125	r1	3		T			\top	П	T	T		T	T						T						\Box		1		
126												136																	
127	r ₁	7				Ш	\perp	Ш			2	23						Ш									Ш		
128			a1	.37										[
129	r ₁							ш										Ш					\square		\sqcup				
130		r ₁₆				ш		Ш										Ш					\square		\perp				
131		r ₁₉						Ш																					
132		r ₂₀						Ш																					
133		r ₂₁																											
134		r ₂₂						Ш				_																	
135		r23	r ₂₃			ш		ш				_						Ш					\square		Ш				
136		\perp	_	4		\Box	_	\sqcup	_	_	_	- 1	143	139	5140	5141	5142	\Box	_	_	_		\Box		Ш	138	-11		
137	r1		_	4		\perp	4	\sqcup	_	_	_	_	_	!			_	\perp	_	_	_	_	\square		Ш		-11		
138	rı		_	4		ш	_	ш	_	_	_	_	_	!				\perp	_		_	_	\square		Ш		-11		
139	r1		_	4		Щ	_	ш	_	_	_	_	_					ш	_			_	\Box		Ш		-11		
140	r ₂							Ш																					
141	r ₂							\square																					
142	r ₂							Ш																			41		
143	r ₂	3						ш																			Ш		



It pass the test at step 13

```
Input (tokens): start num % id = num % num

Maximum number of steps: 100000

PARSE
```

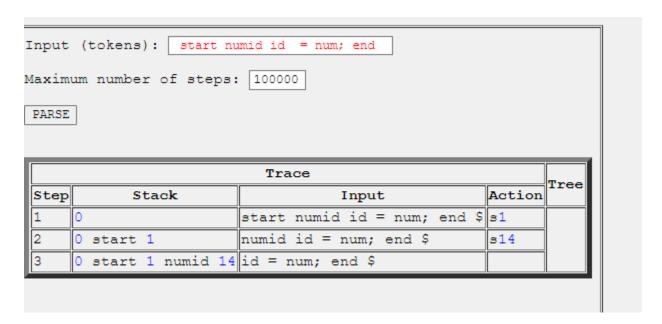
Trace													
Step	Stack	Input	Action	Tree									
1	0	start num % id = num % num \$	s1										
2	0 start 1	num % id = num % num \$	s12										
3	0 start 1 num 12	% id = num % num \$	r ₂₀										
4	0 start 1 factor	% id = num % num \$	9										
5	0 start 1 factor 9	% id = num % num \$	s2 8										
6	0 start 1 factor 9 % 28	id = num % num \$	s 50										
7	0 start 1 factor 9 % 28 id 50	= num % num \$	r ₁₉										
8	0 start 1 factor 9 % 28 factor	= num % num \$	49										
9	0 start 1 factor 9 % 28 factor 49	= num % num \$	r ₁₆										
10	0 start 1 mod	= num % num \$	8										
11	0 start 1 mod 8	= num % num \$	s26										
12	0 start 1 mod 8 = 26	num % num \$	s12										
13	0 start 1 mod 8 = 26 num 12	% num \$	r ₂₀										
14	0 start 1 mod 8 = 26 factor	% num \$	45										
15	0 start 1 mod 8 = 26 factor 45	% num \$	s67										
16	0 start 1 mod 8 = 26 factor 45 % 67	num \$	s 87										
17	0 start 1 mod 8 = 26 factor 45 % 67 num 87	Ş											

It pass the test at step 17

The one that fail

```
Input (tokens): start num % id = num;
Maximum number of steps: 100000
PARSE
                                   Trace
 Step
                     Stack
                                                    Input
                                                                     Action
                                           start num % id = num; $ s1
                                           num % id = num; $
      0 start 1
      0 start 1 num 12
                                           % id = num; $
                                                                     r<sub>20</sub>
                                           % id = num; $
      0 start 1 factor
                                                                     9
                                           % id = num; $
      0 start 1 factor 9
                                                                     s28
                                           id = num; $
                                                                     s50
      0 start 1 factor 9 % 28
      0 start 1 factor 9 % 28 id 50
                                           = num; $
                                                                     r<sub>19</sub>
      0 start 1 factor 9 % 28 factor
                                           = num; $
                                                                     49
      0 start 1 factor 9 % 28 factor 49 = num; $
                                                                     r<sub>16</sub>
      0 start 1 mod
                                           = num; $
 11
      0 start 1 mod 8
                                           = num; $
                                                                     s26
      0 start 1 mod 8 = 26
                                           num; $
```

It fail at step 12



It fails at step 3