

PLC READ ME

The txt file 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 <

Greater_than_equal <=

If_code con

Loop loop

Noequal !

EqualTo ==

Semicolon ;

v is 1 byte num is 2 byte r is 4 byte e is 8 byte

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>-> <ifstmt> | <loop> | <expr>

<loop> -> loop'(<boolexpr>)' '{

' <statement> '}' {<statement>}

<ifstmt>-> con'(<boolexpr>)' '{<statement>}' {<statement>}

<boolexpr> -> identifier '(<|>|<=|>=|==|!>)' int_literal

<expr> -> <term> {(+ | *) <term>}

<term> -> <comp> {(- | /) <comp>}

<comp>-> <equality> {(<|>|<=|>=|==) <equality>}

<equality> -> <mod>{(=|!=)<mod>

<mod>-> <factor> {(%)<factor> |;}

<factor> -> identifies | int_literal | num ident identifies 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 see if it passes pairwise disjoint

$\langle \text{begin} \rangle \rightarrow \text{Start } \langle \text{statement} \rangle \text{ End} \mid \langle \text{statement} \rangle$

$\text{First}(\text{begin}) = \text{Start } \{ \langle \text{ifstmt} \rangle \mid \langle \text{loop} \rangle \mid \langle \text{expr} \rangle \}$

$= \text{start } \{ \text{con} \mid \text{loop} \mid \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{expr} \rangle) \mid \text{semicolon} \}$

Passes since not have same values

$\langle \text{statement} \rangle \rightarrow \langle \text{ifstmt} \rangle \mid \langle \text{loop} \rangle \mid \langle \text{expr} \rangle$

$\text{First}(\text{statement}) = \text{con} \mid \text{loop} \mid \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{expr} \rangle) \mid \text{semicolon}$

Passes

$\langle \text{Loop} \rangle \rightarrow \text{loop}'(\langle \text{boolexpr} \rangle)' \{ \langle \text{statement} \rangle \}' \langle \text{statement} \rangle$

$\text{First}(\text{Loop}) = \text{loop}$

Passes

$\langle \text{ifstmt} \rangle \rightarrow \text{con}'(\langle \text{boolexpr} \rangle)' \{ \langle \text{statement} \rangle \}' \{ \langle \text{statement} \rangle \}$

$\text{First}(\text{ifstmt}) = \text{con}$

$\langle \text{boolexpr} \rangle \rightarrow \text{identifer } (\langle \text{<|>|<=>|=|!} \rangle) \text{ int_literal}$

$\text{First}(\text{boolexpr}) = \text{identifier}$

$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \{ (+ \mid *) \langle \text{term} \rangle \}$

$\text{First}(\text{expr}) \rightarrow \langle \text{term} \rangle = \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{<|>|<=>|=|!} \rangle \mid \text{semicolon}$

$\langle \text{term} \rangle \rightarrow \langle \text{comp} \rangle \{ (- \mid /) \langle \text{comp} \rangle \}$

$\text{First}(\text{term}) \rightarrow \langle \text{comp} \rangle = \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{<|>|<=>|=|!} \rangle \mid \text{semicolon}$

$\langle \text{comp} \rangle \rightarrow \langle \text{equality} \rangle \{ (\langle \text{<|>|<=>|=|!} \rangle) \langle \text{equality} \rangle \}$

$\text{First}(\text{comp}) = \langle \text{equality} \rangle = \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{<|>|<=>|=|!} \rangle \mid \text{semicolon}$

$\langle \text{equality} \rangle \rightarrow \langle \text{mod} \rangle \{ (= \mid !=) \langle \text{mod} \rangle \}$

$\text{First}(\text{equality}) = \langle \text{mod} \rangle = \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{<|>|<=>|=|!} \rangle \mid \text{semicolon}$

$\langle \text{mod} \rangle \rightarrow \langle \text{factor} \rangle \{ (\% \mid ;) \langle \text{factor} \rangle \}$

$\text{First}(\text{mod}) = \langle \text{factor} \rangle \rightarrow \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid (\langle \text{<|>|<=>|=|!} \rangle \mid \text{semicolon}$

$\langle \text{factor} \rangle \rightarrow \text{identifes} \mid \text{int_literal} \mid \text{num ident} \mid \text{identifies semicolon} \mid (\langle \text{expr} \rangle) \mid \text{semicolon}$

First(factor)= <factor> -> identifies | 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.

begin-> start state end

begin-> state

state->expr

expr-> term + term

expr-> term * term

expr-> term

term->' '

term-> comp

term-> comp - comp

term-> comp / comp

comp-> equal < equal

comp-> equal > equal

comp-> equal == equal

comp-> equal <= equal

comp-> equal >= equal

comp-> ' '

comp-> equal

equal-> mod = mod

equal-> mod != mod

equal-> ' '

equal-> mod

mod-> factor % factor

mod->;

mod-> factor

The screenshot shows a web browser window with the address bar displaying 'jsmachines.sourceforge.net/machines/r1.html'. The main content area is a large grid, approximately 20 columns wide and 30 rows high. The grid contains various numbers and symbols, including 'F', 'a', and '123'. The numbers are arranged in a pattern that suggests a sequence or a data set. The browser's address bar and the grid's content are the primary focus of the image.

The one that pass

[illegible]

It pass the test at step 13

Input (tokens):

Maximum number of steps:

Trace				Tree
Step	Stack	Input	Action	
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 \$	r20	
4	0 start 1 factor	% id = num % num \$	9	
5	0 start 1 factor 9	% id = num % num \$	s28	
6	0 start 1 factor 9 % 28	id = num % num \$	s50	
7	0 start 1 factor 9 % 28 id 50	= num % num \$	r19	
8	0 start 1 factor 9 % 28 factor	= num % num \$	49	
9	0 start 1 factor 9 % 28 factor 49	= num % num \$	r16	
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 \$	r20	
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 \$	s87	
17	0 start 1 mod 8 = 26 factor 45 % 67 num 87	\$		

It pass the test at step 17

The one that fail

Input (tokens):

Maximum number of steps:

Trace				Tree
Step	Stack	Input	Action	
1	0	start num % id = num; \$	s1	
2	0 start 1	num % id = num; \$	s12	
3	0 start 1 num 12	% id = num; \$	r20	
4	0 start 1 factor	% id = num; \$	9	
5	0 start 1 factor 9	% id = num; \$	s28	
6	0 start 1 factor 9 % 28	id = num; \$	s50	
7	0 start 1 factor 9 % 28 id 50	= num; \$	r19	
8	0 start 1 factor 9 % 28 factor	= num; \$	49	
9	0 start 1 factor 9 % 28 factor 49	= num; \$	r16	
10	0 start 1 mod	= num; \$	8	
11	0 start 1 mod 8	= num; \$	s26	
12	0 start 1 mod 8 = 26	num; \$		

It fail at step 12

Input (tokens):

Maximum number of steps:

Trace				Tree
Step	Stack	Input	Action	
1	0	start numid id = num; end \$	s1	
2	0 start 1	numid id = num; end \$	s14	
3	0 start 1 numid 14	id = num; end \$		

It fails at step 3