PLC Test 2

Name: Ethan Tang

a.)

Tokens	Regular Expression
INT_LIT	(/d+)
VAR_IDENT	([A-Za-z_]{6, 8})
INT_TYPE_BYTE	(#b)
INT_TYPE_WORD	(#w)
INT_TYPE_DWORD	(#d)
INT_TYPE_QWORD	(#q)
IF_STMT	(\$i)
FOR_LOOP	(\$f)
WHILE_LOOP	(\$w)
BEGIN	(!b)
END	(!e)

b.)

Order of Operation: BDSMAO

Brackets first

Division and Subtraction (left to right)

Multiplication and Addition (left to right)

Order(Exponents)

<program> → !b <stmtlist> !e	Not Left Recursive
<stmtlist> → <stmt> | <stmt>; <stmtlist>	Not Left Recursive
<stmt> → <if_stmt> | <while> | <declr> | <for>	Not Left Recursive
<if_stmt> → `$i``(`<bool_expr>`)`<stmt>	Not Left Recursive
<while> → `$w``(`<bool_expr>`)`<stmt>	Not Left Recursive
<for> → `$f` `(` INT_TYPE `>>` INT_LIT `)``:`<stmt>	Not Left Recursive
<declr> → INT_TYPE VAR_IDEN `=` <expr>	Not Left Recursive
<expr> → <term> { (`*`|`+`) <term>}	Not Left Recursive
<term> → <factor> { (`/`|`-`|`%`) <factor>}	Not Left Recursive
<factor> → VAR_IDEN | INT_LIT | `(` <expr> `)`	Not Left Recursive
<bool_expr> → <brel> { (`!=`|`==`) <brel>}	Not Left Recursive
<brel> → <bexpr> { (`>=`|`<=`|`<`|`>`) <bexpr>}	Not Left Recursive
<bexpr> → <bterm> { (`*`|`+`) <bterm>}	Not Left Recursive
<bterm> → <bfactor> { (`/`|`-`|`%`) <bfactor>}	Not Left Recursive
<bfactor> → VAR_IDEN | INT_LIT | `(` <expr> `)`	Not Left Recursive

c.)

None of the grammar rules' LHS is appearing at the beginning of the RHS so this language's rule sets are not left recursive.

This grammar rule set is also not ambiguous because the LHS of the rules does not appear in its RHS hence there is only one derivation tree.

This language conforms to the LL grammar because it is not left recursive and unambiguous.

d.)

This language is unambiguous as explained in part c.

e.)

Code written in Lexeme.java

f.)

Code written in Syntax.java

g.)

First file test1.in has 5 lexical errors. They are represented by error code 444.

Error 1: #, invalid because # had to be followed by b, w, d, or q to represent a valid number type.

Error 2: q, invalid identifier. Identifier has to consist of 6 to 8 letters or underscores

Error 3: $d, invalid keyword. $d lexeme does not exist

Error 4: notvr, invalid identifier. Identifier has to consist of 6 to 8 letters or underscores

Error 5: ><, invalid relational operator. >< does not exist as a lexeme.

Second file test2.in has 5 syntax errors. They are scanned by Syntax.java. Any errors detected will force the program to end early.

Error 1: Line 2, missing integer type on variable assignment.

Error 2: Line 3, missing semicolon at the end of the statement.

Error 3: Line 4, missing UNTIL token after the first identifier inside the for loop condition.

Error 4: Line 5, unidentified relational operator >> inside the while loop condition.

Error 5: line 6, missing identifier on variable declaration.

Third file test3.in and fourth file test4.in have no error.

h.)

Parse Table for the <expr> rule of this language:

LR table

State	ACTION										GOTO		
	+	*	-	/	%	(	)	id	int	$	E	T	F
0						s4		s5	s6		1	2	3
1	s7	s8											
2	r_2	r_2	s9	s10	s11								
3	r_6	r_6	r_6	r_6	r_6								
4						s15		s16	s17		12	13	14
5	r_8	r_8	r_8	r_8	r_8								
6	r_9	r_9	r_9	r_9	r_9								
7						s20		s21	s22			18	19
8						s4		s5	s6			23	3
9						s4		s5	s6				24
10						s4		s5	s6				25
11						s4		s5	s6				26
12	s28	s29					s27						
13	r_2	r_2	s30	s31	s32		r_2						
14	r_6	r_6	r_6	r_6	r_6		r_6						
15						s15		s16	s17		33	13	14
16	r_8	r_8	r_8	r_8	r_8		r_8						
17	r_9	r_9	r_9	r_9	r_9		r_9						
18	acc	acc	s34	s35	s36					acc			
19	r_6	r_6	r_6	r_6	r_6					r_6			
20						s15		s16	s17		37	13	14
21	r_8	r_8	r_8	r_8	r_8					r_8			
22	r_9	r_9	r_9	r_9	r_9					r_9			
23	r_1	r_1	s9	s10	s11								
24	r_3	r_3	r_3	r_3	r_3								
25	r_4	r_4	r_4	r_4	r_4								
26	r_5	r_5	r_5	r_5	r_5								
27	r_7	r_7	r_7	r_7	r_7								
28						s15		s16	s17			38	14
29						s15		s16	s17			39	14
30						s15		s16	s17				40
31						s15		s16	s17				41
32						s15		s16	s17				42
33	s28	s29					s43						
34						s20		s21	s22				44
35						s20		s21	s22				45
36						s20		s21	s22				46
37	s28	s29					s47						
38	acc	acc	s30	s31	s32		acc						
39	r_1	r_1	s30	s31	s32		r_1						
40	r_3	r_3	r_3	r_3	r_3		r_3						
41	r_4	r_4	r_4	r_4	r_4		r_4						
42	r_5	r_5	r_5	r_5	r_5		r_5						
43	r_7	r_7	r_7	r_7	r_7		r_7						
44	r_3	r_3	r_3	r_3	r_3					r_3			
45	r_4	r_4	r_4	r_4	r_4					r_4			
46	r_5	r_5	r_5	r_5	r_5					r_5			
47	r_7	r_7	r_7	r_7	r_7					r_7			

Input (tokens): `id * ( id + id )`

Maximum number of steps: `100`

PARSE

Trace

Step	Stack	Input	Action	Tree
1	0	id * (id + id) $	s5	E
2	0 id 5	* (id + id) $	r_8	T
3	0 F	* (id + id) $	3	F
4	0 F 3	* (id + id) $	r_6	id
5	0 T	* (id + id) $	2	
6	0 T 2	* (id + id) $	r_2	
7	0 E	* (id + id) $	1	
8	0 E 1	* (id + id) $	s8	
9	0 E 1 * 8	(id + id) $	s4	
10	0 E 1 * 8 (4	id + id) $	s16	
11	0 E 1 * 8 (4 id 16	+ id) $	r_8	
12	0 E 1 * 8 (4 F	+ id) $	14	
13	0 E 1 * 8 (4 F 14	+ id) $	r_6	
14	0 E 1 * 8 (4 T	+ id) $	13	
15	0 E 1 * 8 (4 T 13	+ id) $	r_2	
16	0 E 1 * 8 (4 E	+ id) $	12	
17	0 E 1 * 8 (4 E 12	+ id) $	s28	
18	0 E 1 * 8 (4 E 12 + 28	id) $	s16	
19	0 E 1 * 8 (4 E 12 + 28 id 16	) $	r_8	
20	0 E 1 * 8 (4 E 12 + 28 F	) $	14	
21	0 E 1 * 8 (4 E 12 + 28 F 14	) $	r_6	
22	0 E 1 * 8 (4 E 12 + 28 T	) $	38	
23	0 E 1 * 8 (4 E 12 + 28 T 38	) $	acc	

Code Sample 1: id * (id + id) PASS

Input (tokens): `id * ( id + id )`

Maximum number of steps: `100`

PARSE

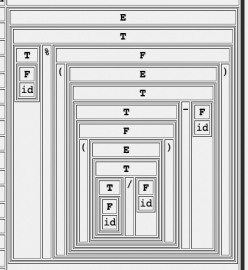
Trace				Tree
Step	Stack	Input	Action	
1	0	id * (id + id) $	s5	E
2	0 id 5	* (id + id) $	r_8	T
3	0 F	* (id + id) $	3	F
4	0 F 3	* (id + id) $	r_6	id
5	0 T	* (id + id) $	2	
6	0 T 2	* (id + id) $	r_2	
7	0 E	* (id + id) $	1	
8	0 E 1	* (id + id) $	s8	
9	0 E 1 * 8	(id + id) $	s4	
10	0 E 1 * 8 (4	id + id) $	s16	
11	0 E 1 * 8 (4 id 16	+ id) $	r_8	
12	0 E 1 * 8 (4 F	+ id) $	14	
13	0 E 1 * 8 (4 F 14	+ id) $	r_6	
14	0 E 1 * 8 (4 T	+ id) $	13	
15	0 E 1 * 8 (4 T 13	+ id) $	r_2	
16	0 E 1 * 8 (4 E	+ id) $	12	
17	0 E 1 * 8 (4 E 12	+ id) $	s28	
18	0 E 1 * 8 (4 E 12 + 28	id) $	s16	
19	0 E 1 * 8 (4 E 12 + 28 id 16	) $	r_8	
20	0 E 1 * 8 (4 E 12 + 28 F	) $	14	
21	0 E 1 * 8 (4 E 12 + 28 F 14	) $	r_6	
22	0 E 1 * 8 (4 E 12 + 28 T	) $	38	
23	0 E 1 * 8 (4 E 12 + 28 T 38	) $	acc	

Code Sample 2: id % ((id / id) - id) + id PASS

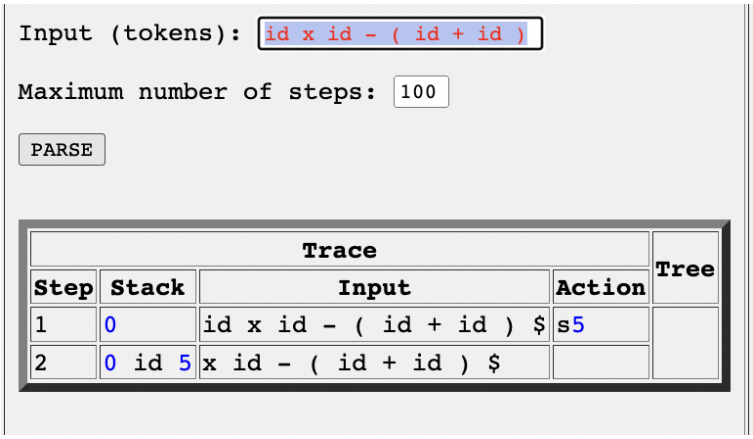
Input (tokens): `id % ( ( id / id ) - id ) + id`

Maximum number of steps: `100`

PARSE

Trace				Tree
Step	Stack	Input	Action	
1	0	id % ((id / id) - id) + id $	s5	
2	0 id 5	% ((id / id) - id) + id $	r_8	
3	0 F	% ((id / id) - id) + id $	3	
4	0 F 3	% ((id / id) - id) + id $	r_6	
5	0 T	% ((id / id) - id) + id $	2	
6	0 T 2	% ((id / id) - id) + id $	s11	
7	0 T 2 % 11	((id / id) - id) + id $	s4	
8	0 T 2 % 11 (4	(id / id) - id) + id $	s15	
9	0 T 2 % 11 (4 (15	id / id) - id) + id $	s16	
10	0 T 2 % 11 (4 (15 id 16	/ id) - id) + id $	r_8	
11	0 T 2 % 11 (4 (15 F	/ id) - id) + id $	14	
12	0 T 2 % 11 (4 (15 F 14	/ id) - id) + id $	r_6	
13	0 T 2 % 11 (4 (15 T	/ id) - id) + id $	13	
14	0 T 2 % 11 (4 (15 T 13	/ id) - id) + id $	s31	
15	0 T 2 % 11 (4 (15 T 13 / 31	id) - id) + id $	s16	
16	0 T 2 % 11 (4 (15 T 13 / 31 id 16	) - id) + id $	r_8	
17	0 T 2 % 11 (4 (15 T 13 / 31 F	) - id) + id $	41	
18	0 T 2 % 11 (4 (15 T 13 / 31 F 41	) - id) + id $	r_4	
19	0 T 2 % 11 (4 (15 T	) - id) + id $	13	
20	0 T 2 % 11 (4 (15 T 13	) - id) + id $	r_2	
21	0 T 2 % 11 (4 (15 E	) - id) + id $	33	
22	0 T 2 % 11 (4 (15 E 33	) - id) + id $	s43	
23	0 T 2 % 11 (4 (15 E 33) 43	- id) + id $	r_7	
24	0 T 2 % 11 (4 F	- id) + id $	14	
25	0 T 2 % 11 (4 F 14	- id) + id $	r_6	
26	0 T 2 % 11 (4 T	- id) + id $	13	
27	0 T 2 % 11 (4 T 13	- id) + id $	s30	
28	0 T 2 % 11 (4 T 13 - 30	id) + id $	s16	
29	0 T 2 % 11 (4 T 13 - 30 id 16	) + id $	r_8	
30	0 T 2 % 11 (4 T 13 - 30 F	) + id $	40	
31	0 T 2 % 11 (4 T 13 - 30 F 40	) + id $	r_3	
32	0 T 2 % 11 (4 T	) + id $	13	
33	0 T 2 % 11 (4 T 13	) + id $	r_2	
34	0 T 2 % 11 (4 E	) + id $	12	
35	0 T 2 % 11 (4 E 12	) + id $	s27	
36	0 T 2 % 11 (4 E 12) 27	+ id $	r_7	
37	0 T 2 % 11 F	+ id $	26	
38	0 T 2 % 11 F 26	+ id $	r_5	
39	0 T	+ id $	2	
40	0 T 2	+ id $	r_2	
41	0 E	+ id $	1	
42	0 E 1	+ id $	s7	
43	0 E 1 + 7	id $	s21	
44	0 E 1 + 7 id 21	$	r_8	
45	0 E 1 + 7 F	$	19	
46	0 E 1 + 7 F 19	$	r_6	
47	0 E 1 + 7 T	$	18	
48	0 E 1 + 7 T 18	$	acc	

Code Sample 3: id x id - (id + id) FAIL



Code Sample 4: id + id (id)) id FAIL

Input (tokens): id + id (id)) id

Maximum number of steps: 100

PARSE

Trace				Tree
Step	**Stack**	**Input**	**Action**	
1	0	id + id (id)) id $	s5	
2	0 id 5	+ id (id)) id $	r_8	
3	0 F	+ id (id)) id $	3	
4	0 F 3	+ id (id)) id $	r_6	
5	0 T	+ id (id)) id $	2	
6	0 T 2	+ id (id)) id $	r_2	
7	0 E	+ id (id)) id $	1	
8	0 E 1	+ id (id)) id $	s7	
9	0 E 1 + 7	id (id)) id $	s21	
10	0 E 1 + 7 id 21	(id)) id $		