Jason Tran

PLC

Test 2

- Notes on the submission file:
  - There are 7 files: 4 .txt test files, 1 code file Lex_Parse.py, 1 readme file (this one), and 1 PDF of the LR(1) parse table.
  - Lexer and Parser code are both in 1 file called Lex_Pase.py.
  - Only run Lex_Parse.py
  - Choose test file to run by changing the hard coded path in run time code:

```
#token list created
mytokens, lex_stat = mylex.tokenize("no_error_test1.txt")
```

Choose the test file by typing the file name in the tokenize function

PART A: Define Token rules (using Regex)

Token name	Regex rule	Token name	Regex rule
case_key:	case	**module:**	%
itr_key:	itr	**assign:**	=
other_key:	other	**EQ:**	==
times_key:	times	**NEQ:**	!=
int_key:	nat	**LT:**	<
lit_int8b:	0|-?[1-9])[0-9]*	**GT:**	>
lit_int8b	0|-?[1-9])[0-9]*_b8	**LTE:**	<=
lit_int4b	0|-?[1-9])[0-9]*_b4	**GTE:**	>=
lit_int2b	0|-?[1-9])[0-9]*_b2	**L_paren:**	\(
lit_int1b	0|-?[1-9])[0-9]*_b1	**R_paren:**	\)
var_name:	[a-zA-Z_]{1,7}	**L_bracket:**	\[
end_stmt:	\.	**R_bracket:**	\]
declare_var:	var	**BEGIN:**	BEGIN
add:	\+	**END:**	END
subtract:	-		
multiply:	*		
divide:	/		

Section B: define production rules

```
<Start>          ::= BEGIN <Statement_list>

<Statement_list>    ::=  <Statement> <Statement_list>
                         | END

<Statement>   ::= nat <Var_decl>
             | case <Case>
             | itr <Itr>
             | var_name <Var_assign>

<Var_decl> ::= var_name "."

<Var_assign> ::=  "=" <Math_expr>


<Case> ::=    <Boolean_expr> "[" <If_true> other "[" <If_false> "."

<Boolean_expr> ::= "(" <Number> <Rela_op> <Number> ")"
<Rela_op> ::= ">" | "<" | "==" | "!=" | "<=" | ">="

<If_true> ::=  <Statement> <If_true> | "]"
<If_false> ::=   <Statement> <If_false> | "]"


<Itr> ::= <Number> times "[" <To_repeat> "."
<To_repeat> ::=  <Statement> <To_repeat> | "]"
```

PART B: (page 2)

<Math_expr> ::= <sum>
<Sum> ::= <Mul> "+" <Sum>
| <Mul>
<Mul> ::= <Div> "*" <Mul>
| <Div>
<Div> ::= <Subtr> "/" <Div>
| <Subtr>
<Subtr> ::= <Mod> "-" <Subtr>
| <Mod>
<Mod> ::= <Factor> "%" <Mod>
| <Factor>
<Factor> ::= "(" <Math_expr> ")"
| <Number>

<Number> ::= lit_int8b
| lit_int4b
| lit_int2b
| lit_int1b
| var_name

- Notes on production rules:
  - var_name is any var_name token

- 6 levels of precedence in math expressions (Low to High):
  1. Sum
  2. Multiply
  3. Divide
  4. Subtract
  5. Module
  6. Parentheses

PART C: PROVE RULE SET CONFORM TO LL

My production rules conform to LL because it does not have left hand recursion and it passes the pairwise disjointness test.

- No left hand recursion:
    - In my grammar, there are 4 non-terminals that uses recursion (<Statement_list>, <If_true>, <If_false>, <To_repeat>). However, each of these rules have an alternatives that contains only one terminal symbol. Those terminal symbols would stop the recursion (like a base case). In all cases, there is no left hand recursion.
    - In my mathmetical_expression Non-terminal symbol <math_expr>, there is recursion being used but it does not lead to indirect left hand recursion in any case.

- Pass the pairwise disjointness test:
    - FIRST <Start> ::= {BEGIN}
    - FIRST <Statement_list> ::= {nat, case, itr, var_name, END}
    - FIRST <Statement> ::= {nat}, {case}, {itr}, {var_name}
    - FIRST <Var_decl> ::= {var_name}
    - FIRST <Var_assign> ::= {"="}
    - FIRST <Case > ::= {"("}
    - FIRST <Boolean_expr> ::= {"("}
    - FIRST <Rela_op> ::= {">"}, {"<"}, {"=="}, {"!="}, {"<="}, {">="}
    - FIRST <If_true> ::= {nat, case, itr, var_name}, {"]"}
    - FIRST <If_false> ::= {nat, case, itr, var_name}, {"]"}
    - FIRST <Itr> ::= {lit_int8b}, { lit_int4b }, {lit_int2b }, { lit_int1b }, {var_name}
    - FIRST <To_repeat> ::= {nat, case, itr, var_name}, {"["}
    - FIRST <Math_expr> ::= {"(", {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
    - FIRST <Sum> ::= {"(", {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
    - FIRST <Mul> ::= {"(", {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
    - FIRST <Div> ::= {"(", {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}

PART C (PAGE 2)

- FIRST <Subtr> ::= {“(“, {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
- FIRST <Mod> ::= {“(“, {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
- FIRST <Factor> ::= {“(“, {lit_int8b, lit_int4b, lit_int2b , lit_int1b, var_name}}
- FIRST <Number> ::= {lit_int8b}, { lit_int4b }, {lit_int2b }, { lit_int1b }, {var_name}

➔ Because there are no matching terminal symbols in each of the non terminals’ FIRST sets (the alternatives of each non-terminal is disjoint), these grammar rules passes the pairwise disjointness test

PART D: PROVE GRAMMAR IS UNAMBIGUOUS

- My grammar is not ambiguous because It passes the pairwise disjointness test and because It is not possible to generate a string that has more than one parse tree in this grammar.

PART E: LEXICAL ANALYZER

The lexical analyzer should:

- Recognize all tokens
- Produce a list of those tokens
- Print error message for lexical error
- Be object-oriented
- Have comments

Example 1: test file with no errors

no_error_test.txt

```
BEGIN

nat a
a = 1 + ( 2 % 3 ) .

END

```

Output:

```
Lexical error:  No
Syntax error:  Yes

token list:

[BEGIN:BEGIN,
 nat:int_key,
 a:var_name,
 a:var_name,
 =:assign,
 1:lit_int8b,
 +:add,
 (:L_paren,
 2:lit_int8b,
 %:module,
 3:lit_int8b,
 ):R_paren,
 .:end_stmt,
 END:END]
```

The lexical analyzer recognized all tokens in the test file, produced a complete list of those tokens, confirmed that there were no lexical errors

PART E: (Page 2)

Code snippet of lexical analyzer:

```
#---------------------------------------------------------------
#                            LEXER
#---------------------------------------------------------------
class Lexer:
    #read program (text file) and create a list of tokens
    def tokenize(self, file):···
```

This tells us that the lexical analyzer is object-oriented and it has comments for its function

Test file with lexical error:

```
BEGIN

nat &&}

END

```

Its output:

```
[Running] python -u "d:\Coding\Python files\PLC\Exam
BEGIN

nat &&}

END


Lexical error:  &&}  at index:  2  (invalid lexeme)
```

This shows that the Lexer will print an error message when there is a lexical error.

PART F: SYNTACTICAL ANALYZER

My program only tells the user if there is a syntax error, it does not ouput a specific error message for most errorss

Examples:

- Test file with correct syntax:

```
BEGIN

nat a .
a = 13 - ( 3 % ( 8 / 2 ) - 50 ) .
nat b .
b = 0 .
itr 5 times [ b = b + 1 . ] .

END
```

- Its output:

```
BEGIN

nat a .
a = 13 - ( 3 % ( 8 / 2 ) - 50 ) .
nat b .
b = 0 .
itr 5 times [ b = b + 1 . ] .

END


Lexical error:  No
Syntax error:  No
```

This means that the variable declaration, the mathematical expression, the variable initialization, and the itr loop are all in the correct grammar.

PART F: (PAGE 2)

- Test file with syntax error in itr loop (missing brackets):

```
BEGIN

nat a .
a = 13 - ( 3 % ( 8 / 2 ) - 50 ) .
nat b .
b = 0 .
itr 5 times  b = b + 1 .

END

```

- Its output:

```
BEGIN

nat a .
a = 13 - ( 3 % ( 8 / 2 ) - 50 ) .
nat b .
b = 0 .
itr 5 times  b = b + 1 .

END


Syntax error: "[" expected
Lexical error:  No
Syntax error:  Yes
```

This shows that the syntax analyzer can find the correct syntax error in the text file

PART G: TEST FILES

- No_error_test1.txt:

```
no_error_test1.txt
BEGIN

nat a .
nat b .

a = 13_b1 - ( 3 % ( 8 / 2 ) - 50 ) .
b = a % 7 - 2 * 3 + ( a - 10 / ( a * 2 ) ) .

END
```

- Its output:

```
Lexical error:  No
Syntax error:  No

token list:

[BEGIN:BEGIN,
 nat:int_key,
 a:var_name,
 .:end_stmt,
 nat:int_key,
 b:var_name,
 .:end_stmt,
 a:var_name,
 =:assign,
 13_b1:lit_int1b,
 .:end_stmt,
 b:var_name,
 =:assign,
 a:var_name,
 +:add,
 7:lit_int8b,
 -:subtract,
 2:lit_int8b,
 *:multiply,
 3:lit_int8b,
 +:add,
 (:L_paren,
 a:var_name,
 -:subtract,
 10:lit_int8b,
 *:multiply,
 (:L_paren,
 a:var_name,
 *:multiply,
 2:lit_int8b,
 ):R_paren,
 ):R_paren,
 .:end_stmt,
 END:END]
```

PART G: TEST FILES

- No_error_test2.txt:

```
no_error_test2.txt
BEGIN

nat a .
nat b .
nat count .
a = 0 .

case ( a == 2 ) [ itr 6 times [ a = a + 1 . ] .
                  b = 0 .
                  b = b + a .
                  count = count + 1 .
]
other [ b = -99 . ] .

END
```

- Its output:

```
Lexical error:  No
Syntax error:  No

token list:

[BEGIN:BEGIN,
 nat:int_key,
 a:var_name,
 .:end_stmt,
 nat:int_key,
 b:var_name,
 .:end_stmt,
 nat:int_key,
 count:var_name,
 .:end_stmt,
 a:var_name,
 =:assign,
 0:lit_int8b,
 .:end_stmt,
 case:case_key,
 (:L_paren,
 a:var_name,
 ==:EQ,
 2:lit_int8b,
 ):R_paren,
 [:L_bracket,
 itr:itr_key,
 6:lit_int8b,
 times:times_key,
 [:L_bracket,
 a:var_name,
 =:assign,
 a:var_name,
 +:add,
 1:lit_int8b,
 .:end_stmt,
 ]:R_bracket,
 .:end_stmt,
 b:var_name,
 =:assign,
 0:lit_int8b,
 .:end_stmt,
 b:var_name,
 =:assign,
 b:var_name,
 +:add,
 a:var_name,
 .:end_stmt,
 count:var_name,
 =:assign,
 count:var_name,
 +:add,
 1:lit_int8b,
 .:end_stmt,
 ]:R_bracket,
 other:other_key,
 [:L_bracket,
 b:var_name,
 =:assign,
 -99:lit_int8b,
 .:end_stmt,
 ]:R_bracket,
 .:end_stmt,
 END:END]
```

PART G: TEST FILES

- lex_test.txt: has 5 lexical errors

```
lex_test.txt
1   BEGIN
2
3   nat a.
4   nat $ .
5   nat count .
6   :) === 0 .
7
8   case ( @ == 2 ) [ itr 6 times [ a = a + 1 . ] .
9                     b = 0 .
10                    b = b + a .
11                    count = count + 1 .
12  ]
13  other [ b = -99 . ] .
14
15  END
```

1. line 3: a. is not a valid variable name (a and "." Need to be separate)
2. line 4: $ is not a token in this language
3. line 6: :) is not a token in this language
4. line 6: === is not a token in this language
5. line 8: @ is not a token in this language

- Its output:

```
Lexical error:  a.  at index:  2  (invalid lexeme)
```

PART G: TEST FILES

- syn_test.txt: has 5 syntax errors

```
syn_test.txt
1   BEGIN
2
3   a a .
4   nat b = 10 .
5   nat 30 .
6   a = 0 .
7
8   case  a == 2  [ itr 6 times [ a = a + 1 . ] .
9                 b = 0
10                b = b + a .
11                count = count + 1 .
12  ]
13  other [ b = -99 . ] .
14
15  END
```

1. line 3: var_name can only be followed by “=”, a a . is not correct for var_assign
2. line 4: nat is first token -> this Is a var_declaration statement. Var_declaration does not have “=” or integer literals -> “=” and 10 should not be there
3. line 5: nat is first token -> this Is a var_declaration statement. Token nat can only be followed by a var_name, not 30.
4. line 8: “ a == 2 “ needs to have parentheses, correct syntax is “(a == 2)”
5. line 9: This is a var_assign statement, it needs “.” To end the statement.

- Its output:

```
Error: expecting "="
Lexical error:  No
Syntax error:  Yes
```

PART H: LR(1) PARSE TABLE

- The grammar input into the parser generator:

```
LR(1) grammar ('' is ε):
S' -> S
S -> BEGIN SL
SL -> ST SL
SL -> END

ST -> nat VD
ST -> case C
ST -> itr I
ST -> var_name VA

VD -> var_name .

VA -> = ME .

C -> BE [ T other [
F .
BE -> ( N O N )
O -> >
O -> <
O -> ==
O -> !=
O -> <=
O -> >=
T -> ST T
T -> ]
```

```
LR(1) grammar ('' is ε):
T -> ST T
T -> ]
F -> ST F
F -> ]

I -> N times [ RP ]
.
RP -> ST RP
RP -> ]

ME -> SUM
SUM -> MUL + SUM
SUM -> MUL
MUL -> DIV * MUL
MUL -> DIV
DIV -> SUB / DIV
DIV -> SUB
SUB -> MOD - SUB
SUB -> MOD
MOD -> FAC % MOD
MOD -> FAC
FAC -> ( ME )
FAC -> N
N -> int
N -> var_name
```

S = <Start> , SL = <Statement_list>, ST = <Statement>,

VD = <Var_decl>, VA = <Var_assign>, C = <Case>,

BE = <Boolean_expr>, O = <rela_op>, T = <If_true>,

F = <If_false>, I = <Itr>, RP = <To_repeat>,

ME = <Math_expr>, SUM = <Sum>, MUL = <Mul>,

DIV = <Div>, SUB = <Subtr>, MOD = <Mod>,

FAC = <Factor>, N = <Number>, int = {lit_int1b, lit_int2b, lit_4b, lit_8b}

PART H: (PAGE 2)

- FIRST table generated:

FIRST table	
Nonterminal	**FIRST**
S'	{BEGIN}
S	{BEGIN}
SL	{END,nat,case,itr,var_name}
ST	{nat,case,itr,var_name}
VD	{var_name}
VA	{=}
C	{(}
BE	{(}
O	{>,<,==,!=,<=,>=}
T	{nat,case,itr,var_name,]}
F	{nat,case,itr,var_name,]}
I	{int,var_name}
RP	{nat,case,itr,var_name,]}
ME	{(,int,var_name}
SUM	{(,int,var_name}
MUL	{(,int,var_name}
DIV	{(,int,var_name}
SUB	{(,int,var_name}
MOD	{(,int,var_name}
FAC	{(,int,var_name}
N	{int,var_name}

- This table matches exactly with the pairwise disjoint test performed in part C

PART H: (PAGE 3)

- **LR(1) parse table generated**:

LR table

State	ACTION																											GOTO																				
	BEGIN	END	nat	case	itr	var_name	.	=	[	other	(	)	>	<	==	!=	<=	>=	]	times	+	*	/	-	%	int	$	S'	S	SL	ST	VD	VA	C	BE	O	T	F	I	RP	ME	SUM	MUL	DIV	SUB	MOD	FAC	N
0	s2																												1																			
1																											acc																					
2		s5	s6	s7	s8	s9																								3	4																	
3																											r_{1}																					
4		s5	s6	s7	s8	s9																								10	4																	
5																											r_{3}																					
6						s12																										11																
7											s15																							13	14													
8						s19																				s18													16										17
9								s21																									20															
10																											r_{2}																					
11		r_{4}	r_{4}	r_{4}	r_{4}	r_{4}																																										
12							s22																																									
13		r_{5}	r_{5}	r_{5}	r_{5}	r_{5}																																										
14									s23																																							
15						s26																				s25																							24
16		r_{6}	r_{6}	r_{6}	r_{6}	r_{6}																																										
17																				s27																												
18																				r_{38}																												
19																				r_{39}																												
20		r_{7}	r_{7}	r_{7}	r_{7}	r_{7}																																										
21						s38					s35															s37																28	29	30	31	32	33	34	36
22		r_{8}	r_{8}	r_{8}	r_{8}	r_{8}																																										
23			s42	s43	s44	s45													s41												40						39											
24													s47	s48	s49	s50	s51	s52																		46												
25													r_{38}	r_{38}	r_{38}	r_{38}	r_{38}	r_{38}																														
26													r_{39}	r_{39}	r_{39}	r_{39}	r_{39}	r_{39}																														
27									s53																																							
28							s54																																									
29							r_{25}																																									
30							r_{27}														s55																											
31							r_{29}														r_{29}	s56																										
32							r_{31}														r_{31}	r_{31}	s57																									
33							r_{33}														r_{33}	r_{33}	r_{33}	s58																								
34							r_{35}														r_{35}	r_{35}	r_{35}	r_{35}	s59																							
35						s70					s67															s69																60	61	62	63	64	65	66	68
36							r_{37}														r_{37}	r_{37}	r_{37}	r_{37}	r_{37}																							
37							r_{38}														r_{38}	r_{38}	r_{38}	r_{38}	r_{38}																							
38							r_{39}														r_{39}	r_{39}	r_{39}	r_{39}	r_{39}																							
39										s71																																						
40			s42	s43	s44	s45													s41												40						72											
41										r_{19}																																						
42						s74																										73																
43											s15																							75	76													
44						s19																				s18													77										78
45								s80																									79															
46						s83																				s82																							81
47						r_{12}																				r_{12}																							
48						r_{13}																				r_{13}																							
49						r_{14}																				r_{14}																							
50						r_{15}																				r_{15}																							
51						r_{16}																				r_{16}																							
52						r_{17}																				r_{17}																							
53			s42	s43	s44	s45													s86												85									84								
54		r_{9}	r_{9}	r_{9}	r_{9}	r_{9}																																										

PART H: (PAGE 4)

LR table

State	ACTION																											GOTO																				
	BEGIN	END	nat	case	itr	var_name	.	=	[	other	(	)	>	<	==	!=	<=	>=	]	times	+	*	/	-	%	int	$	S'	S	SL	ST	VD	VA	C	BE	O	T	F	I	RP	ME	SUM	MUL	DIV	SUB	MOD	FAC	N
54		r_{9}	r_{9}	r_{9}	r_{9}	r_{9}																																										
55						s38					s35															s37																	87	30	31	32	33	34	36
56						s38					s35															s37																		88	31	32	33	34	36
57						s38					s35															s37																			89	32	33	34	36
58						s38					s35															s37																				90	33	34	36
59						s38					s35															s37																					91	34	36
60												s92																																				
61												r_{25}																																				
62												r_{27}									s93																											
63												r_{29}									r_{29}	s94																										
64												r_{31}									r_{31}	r_{31}	s95																									
65												r_{33}									r_{33}	r_{33}	r_{33}	s96																								
66												r_{35}									r_{35}	r_{35}	r_{35}	r_{35}	s97																							
67						s70					s67															s69																98	61	62	63	64	65	66	68
68												r_{37}									r_{37}	r_{37}	r_{37}	r_{37}	r_{37}																							
69												r_{38}									r_{38}	r_{38}	r_{38}	r_{38}	r_{38}																							
70												r_{39}									r_{39}	r_{39}	r_{39}	r_{39}	r_{39}																							
71									s99																																							
72										r_{18}																																						
73			r_{4}	r_{4}	r_{4}	r_{4}													r_{4}																													
74							s100																																									
75			r_{5}	r_{5}	r_{5}	r_{5}													r_{5}																													
76									s101																																							
77			r_{6}	r_{6}	r_{6}	r_{6}													r_{6}																													
78																				s102																												
79			r_{7}	r_{7}	r_{7}	r_{7}													r_{7}																													
80						s38					s35															s37																103	29	30	31	32	33	34	36
81												s104																																				
82												r_{38}																																				
83												r_{39}																																				
84																			s105																													
85			s42	s43	s44	s45													s86												85									106								
86																			r_{24}																													
87							r_{26}																																									
88							r_{28}														r_{28}																											
89							r_{30}														r_{30}	r_{30}																										
90							r_{32}														r_{32}	r_{32}	r_{32}																									
91							r_{34}														r_{34}	r_{34}	r_{34}	r_{34}																								
92							r_{36}														r_{36}	r_{36}	r_{36}	r_{36}	r_{36}																							
93						s70					s67															s69																	107	62	63	64	65	66	68
94						s70					s67															s69																		108	63	64	65	66	68
95						s70					s67															s69																			109	64	65	66	68
96						s70					s67															s69																				110	65	66	68
97						s70					s67															s69																					111	66	68
98												s112																																				
99			s42	s43	s44	s45													s115												114							113										
100			r_{8}	r_{8}	r_{8}	r_{8}													r_{8}																													
101			s42	s43	s44	s45													s41												40						116											
102									s117																																							
103							s118																																									
104									r_{11}																																							
105							s119																																									
106																			r_{23}																													
107												r_{26}																																				
108												r_{28}									r_{28}																											
109												r_{30}									r_{30}	r_{30}																										
110												r_{32}									r_{32}	r_{32}	r_{32}																									
111												r_{34}									r_{34}	r_{34}	r_{34}	r_{34}																								
112												r_{36}									r_{36}	r_{36}	r_{36}	r_{36}	r_{36}																							
113							s120																																									
114			s42	s43	s44	s45													s115												114							121										
115							r_{21}																																									
116										s122																																						
117			s42	s43	s44	s45													s86												85									123								
118			r_{9}	r_{9}	r_{9}	r_{9}													r_{9}																													
119		r_{22}	r_{22}	r_{22}	r_{22}	r_{22}																																										
120		r_{10}	r_{10}	r_{10}	r_{10}	r_{10}																																										
121							r_{20}																																									
122									s124																																							
123																			s125																													
124			s42	s43	s44	s45													s115												114							126										
125							s127																																									
126							s128																																									
127			r_{22}	r_{22}	r_{22}	r_{22}													r_{22}																													
128			r_{10}	r_{10}	r_{10}	r_{10}													r_{10}																													

Link to PDF of Parse table:[LR(1) parse table]

PART H: (PAGE 5)

- **Code sample 1**: "BEGIN var_name = int . END" (Pass)
- Input (tokens): BEGIN var_name = int . END
- This is a Var_assign statement: it follows the grammar correctly and ends the statement with "." Like it should.
- Its trace:

Trace			
Step	**Stack**	**Input**	**Action**
1	0	BEGIN var_name = int . END $	s2
2	0 BEGIN 2	var_name = int . END $	s9
3	0 BEGIN 2 var_name 9	= int . END $	s21
4	0 BEGIN 2 var_name 9 = 21	int . END $	s37
5	0 BEGIN 2 var_name 9 = 21 int 37	. END $	r_{38}
6	0 BEGIN 2 var_name 9 = 21 N	. END $	36
7	0 BEGIN 2 var_name 9 = 21 N 36	. END $	r_{37}
8	0 BEGIN 2 var_name 9 = 21 FAC	. END $	34
9	0 BEGIN 2 var_name 9 = 21 FAC 34	. END $	r_{35}
10	0 BEGIN 2 var_name 9 = 21 MOD	. END $	33
11	0 BEGIN 2 var_name 9 = 21 MOD 33	. END $	r_{33}
12	0 BEGIN 2 var_name 9 = 21 SUB	. END $	32
13	0 BEGIN 2 var_name 9 = 21 SUB 32	. END $	r_{31}
14	0 BEGIN 2 var_name 9 = 21 DIV	. END $	31
15	0 BEGIN 2 var_name 9 = 21 DIV 31	. END $	r_{29}
16	0 BEGIN 2 var_name 9 = 21 MUL	. END $	30
17	0 BEGIN 2 var_name 9 = 21 MUL 30	. END $	r_{27}
18	0 BEGIN 2 var_name 9 = 21 SUM	. END $	29
19	0 BEGIN 2 var_name 9 = 21 SUM 29	. END $	r_{25}
20	0 BEGIN 2 var_name 9 = 21 ME	. END $	28
21	0 BEGIN 2 var_name 9 = 21 ME 28	. END $	s54
22	0 BEGIN 2 var_name 9 = 21 ME 28 . 54	END $	r_{9}
23	0 BEGIN 2 var_name 9 VA	END $	20
24	0 BEGIN 2 var_name 9 VA 20	END $	r_{7}
25	0 BEGIN 2 ST	END $	4
26	0 BEGIN 2 ST 4	END $	s5
27	0 BEGIN 2 ST 4 END 5	$	r_{3}
28	0 BEGIN 2 ST 4 SL	$	10
29	0 BEGIN 2 ST 4 SL 10	$	r_{2}
30	0 BEGIN 2 SL	$	3
31	0 BEGIN 2 SL 3	$	r_{1}
32	0 S	$	1
33	0 S 1	$	acc

-

PART H: (PAGE 6)

- **Code sample 2**: "BEGIN itr int times [nat var_name .] . END" (Pass)
- Input (tokens): BEGIN itr int times [nat var_name .] . END
- This is an itr (for loop) statement: the grammar: "itr (int or var_name) times [statement] ." is properly followed
- Its trace:

Trace			
Step	Stack	Input	Action
1	0	BEGIN itr int times [nat var_name .] . END $	s2
2	0 BEGIN 2	itr int times [nat var_name .] . END $	s8
3	0 BEGIN 2 itr 8	int times [nat var_name .] . END $	s18
4	0 BEGIN 2 itr 8 int 18	times [nat var_name .] . END $	r_{38}
5	0 BEGIN 2 itr 8 N	times [nat var_name .] . END $	17
6	0 BEGIN 2 itr 8 N 17	times [nat var_name .] . END $	s27
7	0 BEGIN 2 itr 8 N 17 times 27	[nat var_name .] . END $	s53
8	0 BEGIN 2 itr 8 N 17 times 27 [53	nat var_name .] . END $	s42
9	0 BEGIN 2 itr 8 N 17 times 27 [53 nat 42	var_name .] . END $	s74
10	0 BEGIN 2 itr 8 N 17 times 27 [53 nat 42 var_name 74	.] . END $	s100
11	0 BEGIN 2 itr 8 N 17 times 27 [53 nat 42 var_name 74 . 100	] . END $	r_8
12	0 BEGIN 2 itr 8 N 17 times 27 [53 nat 42 VD	] . END $	73
13	0 BEGIN 2 itr 8 N 17 times 27 [53 nat 42 VD 73	] . END $	r_4
14	0 BEGIN 2 itr 8 N 17 times 27 [53 ST	] . END $	85
15	0 BEGIN 2 itr 8 N 17 times 27 [53 ST 85	] . END $	s86
16	0 BEGIN 2 itr 8 N 17 times 27 [53 ST 85] 86	. END $	r_{24}
17	0 BEGIN 2 itr 8 N 17 times 27 [53 ST 85 RP	. END $	106
18	0 BEGIN 2 itr 8 N 17 times 27 [53 ST 85 RP 106	. END $	r_{23}
19	0 BEGIN 2 itr 8 N 17 times 27 [53 RP	. END $	84
20	0 BEGIN 2 itr 8 N 17 times 27 [53 RP 84	. END $	s105
21	0 BEGIN 2 itr 8 N 17 times 27 [53 RP 84 . 105	END $	r_{22}
22	0 BEGIN 2 itr 8 I	END $	16
23	0 BEGIN 2 itr 8 I 16	END $	r_6
24	0 BEGIN 2 ST	END $	4
25	0 BEGIN 2 ST 4	END $	s5
26	0 BEGIN 2 ST 4 END 5	$	r_3
27	0 BEGIN 2 ST 4 SL	$	10
28	0 BEGIN 2 ST 4 SL 10	$	r_2
29	0 BEGIN 2 SL	$	3
30	0 BEGIN 2 SL 3	$	r_1
31	0 S	$	1
32	0 S 1	$	acc

PART H: (PAGE 7)

- **Code sample 3**: "BEGIN var_name = int \ itr . END" (Fail: itr keyword should not be in Var_assign statement)
- Input (tokens): BEGIN var_name = int \ itr . END
- This is a Var_assign statement:
- Its trace:

Trace				Tree
Step	Stack	Input	Action	
1	0	BEGIN var_name = int \ itr . END $	s2	
2	0 BEGIN 2	var_name = int \ itr . END $	s9	
3	0 BEGIN 2 var_name 9	= int \ itr . END $	s21	
4	0 BEGIN 2 var_name 9 = 21	int \ itr . END $	s37	
5	0 BEGIN 2 var_name 9 = 21 int 37	\ itr . END $		

PART H: (PAGE 8)

- **Code sample 4**: “BEGIN var_name = var_name + int \ var_name END” (Fail: Var_assign statement needs an end_stmt token “.”)
- Input (tokens): BEGIN var_name = var_name + int \ var_name END
- This is a Var_assign statement:
- Its trace:

Trace				Tree
Step	Stack	Input	Action	
1	0	BEGIN var_name = var_name + int \ var_name END $	s2	
2	0 BEGIN 2	var_name = var_name + int \ var_name END $	s9	
3	0 BEGIN 2 var_name 9	= var_name + int \ var_name END $	s21	
4	0 BEGIN 2 var_name 9 = 21	var_name + int \ var_name END $	s38	
5	0 BEGIN 2 var_name 9 = 21 var_name 38	+ int \ var_name END $	r_{39}	
6	0 BEGIN 2 var_name 9 = 21 N	+ int \ var_name END $	36	
7	0 BEGIN 2 var_name 9 = 21 N 36	+ int \ var_name END $	r_{37}	
8	0 BEGIN 2 var_name 9 = 21 FAC	+ int \ var_name END $	34	
9	0 BEGIN 2 var_name 9 = 21 FAC 34	+ int \ var_name END $	r_{35}	
10	0 BEGIN 2 var_name 9 = 21 MOD	+ int \ var_name END $	33	
11	0 BEGIN 2 var_name 9 = 21 MOD 33	+ int \ var_name END $	r_{33}	
12	0 BEGIN 2 var_name 9 = 21 SUB	+ int \ var_name END $	32	
13	0 BEGIN 2 var_name 9 = 21 SUB 32	+ int \ var_name END $	r_{31}	
14	0 BEGIN 2 var_name 9 = 21 DIV	+ int \ var_name END $	31	
15	0 BEGIN 2 var_name 9 = 21 DIV 31	+ int \ var_name END $	r_{29}	
16	0 BEGIN 2 var_name 9 = 21 MUL	+ int \ var_name END $	30	
17	0 BEGIN 2 var_name 9 = 21 MUL 30	+ int \ var_name END $	s55	
18	0 BEGIN 2 var_name 9 = 21 MUL 30 + 55	int \ var_name END $	s37	
19	0 BEGIN 2 var_name 9 = 21 MUL 30 + 55 int 37	\ var_name END $		