Project 2 Report

Jacob Sherrill
The University of Tulsa
CS4013 – Compiler Construction
October 2016

Introduction

For this project, I have created a syntax analyzer for a subset of the Pascal programming language. This program produces a listing file and a token file from an input pascal source file and a reserved word file. Syntax errors are detected.

Methodology

I massaged the initial Pascal grammar that was given. After that, a parse table was created. From there, the syntax analyzer was programmed. Here is the grammar after each transformation step as well as the first and follow sets and the parsing table.

Grammar with null productions removed

1.1 *program* → **program id** (*identifier_list*) ; *declarations subprogram_declarations cmpd_stmnt* .
1.2 *program* → **program id** (*identifier_list*) ; *subprogram_declarations compound_statement* .
1.3 *program* → **program id** (*identifier_list*) ; *declarations compound_statement*
1.4 *program* → **program id** (*identifier_list*) ; *compound_statement*
2.1 *identifier_list* → **id**
2.2 *identifier_list* → *identifier_list,* **id**
3.1 *declarations* → *declarations* **var id** : *type* ;
3.2 *declarations* → **var id** : *type* ;
4.1 *type* → *standard_type*
4.2 *type* → **array [num .. num] of** *standard_type*
5.1 *standard_type* → **integer**
5.2 *standard_type* → **real**
6.1 *subprogram_declarations* → *subprogram_declarations subprogram_declaration* ;
6.2 *subprogram_declarations* → *subprogram_declaration*
7.1 *subprogram_declaration* → *subprogram_head declarations compound_statement*
7.2 *subprogram_declaration* → *subprogram_head declarations subprogram_declarations cmpd_stmt*
8.1 *subprogram_head* → **procedure id** *arguments* ;
8.2 *subprogram_head* → **procedure id** ;
9.1 *arguments* → (*parameter_list*)
10.1 *parameter_list* → *id* : *type*
10.2 *parameter_list* → *parameter_list* ; *id* : *type*
11.1 *compound_statement* → **begin** *optional_statements* **end**
11.2 *compound_statement* → **begin end**
12.1 *optional_statements* → *statement_list*
13.1 *statement_list* → *statement*
13.2 *statement_list* → *statement_list* ; *statement*
14.1 *statement* → *variable* **assignop** *expression*
14.2 *statement* → *procedure_statement*
14.3 *statement* → *compound_statement*
14.4 *statement* → **if** *expression* **then** *statement* **else** *statement*
14.5 *statement* → **while** *expression* **do** *statement*
14.6 *statement* → **if** *expression* **then** *statement*
15.1 *variable* → **id**
15.2 *variable* → **id** [*expression*]

16.1 *procedure_statement* → **call id**
16.2 *procedure_statement* → **call id** (*expression_list*)
17.1 *expression_list* → *expression*
17.2 *expression_list* → *expression_list* , *expression*
18.1 *expression* → *simple_expression*
18.2 *expression* → *simple_expression* **relop** *simple_expression*
19.1 *simple_expression* → *term*
19.2 *simple_expression* → *sign term*
19.3 *simple_expression* → *simple_expression* **addop** *term*
20.1 *term* → *factor*
20.2 *term* → *term* **mulop** *factor*
21.1 *factor* → **id**
21.2 *factor* → **id** (*expression_list*)
21.3 *factor* → ***num***
21.4 *factor* → (*expression*)
21.5 *factor* → **not** *factor*
21.6 *factor* → **id** [*expression*]
22.1 *sign* → **+**
22.2 *sign* → **-**

Removal of immediate left recursion (step 2b)

Grammar after removing immediate left recursion:

1.1 *program* → **program id** (*identifier_list*) ; *declarations subprogram_declarations cmpd_stmnt* .
1.2 *program* → **program id** (*identifier_list*) ; *subprogram_declarations compound_statement* .
1.3 *program* → **program id** (*identifier_list*) ; *declarations compound_statement* .
1.4 *program* → **program id** (*identifier_list*) ; *compound_statement* .
2.1.1 *identifier_list* → **id** *identifier_list_tail*
2.1.2 *identifier_list_tail* → , **id** *identifier_list_tail*
2.1.3 *identifier_list_tail* → Ɛ
3.1.1 *declarations* → **var id** : *type* ; *declarations_tail*
3.1.2 *declarations_tail* → **var id** : *type* ; *declarations_tail*
3.1.3 *declarations_tail* → Ɛ
4.1 *type* → *standard_type*
4.2 *type* → **array** [**num** .. **num**] **of** *standard_type*
5.1 *standard_type* → **integer**
5.2 *standard_type* → **real**
6.1.1 *subprogram_declarations* → *subprogram_declaration subprogram_declarations_tail*
6.1.2 *subprogram_declarations_tail* → *subprogram_declaration* ; *subprogram_declarations_tail*
6.1.3 *subprogram_declarations_tail* → Ɛ
7.1 *subprogram_declaration* → *subprogram_head declarations compound_statement*
7.2 *subprogram_declaration* → *subprogram_head declarations subprogram_declarations cmpd_stmt*
8.1 *subprogram_head* → **procedure id** *arguments* ;
8.2 *subprogram_head* → **procedure id** ;
9.1 arguments → (*parameter_list*)
10.1.1 *parameter_list* → **id** : *type parameter_list_tail*
10.1.2 *parameter_list_tail* → ; **id** : *type parameter_list_tail*

10.1.3 *parameter_list_tail* → Ɛ
11.1.1 *compound_statement* → **begin** *compound_statement_rest*
11.1.2 *compound_statement_rest* → *optional_statements* **end**
11.1.3 *compound_statement_rest* → **end**
12.1 *optional_statements* → *statement_list*
13.1.1 *statement_list* → *statement statement_list_tail*
13.1.2 *statement_list_tail* → ; *statement statement_list_tail*
13.1.3 *statement_list_tail* → Ɛ
14.1 *statement* → *variable* **assignop** *expression*
14.2 *statement* → *procedure_statement*
14.3 *statement* → *compound_statement*
14.4 *statement* → **if** *expression* **then** *statement* **else** *statement*
14.5 *statement* → **while** *expression* **do** *statement*
14.6 *statement* → **if** *expression* **then** *statement*
15.1 *variable* → **id**
15.2 *variable* → **id** [*expression*]
16.1 *procedure_statement* → **call id**
16.2 *procedure_statement* → **call id** (*expression_list*)
17.1.1 *expression_list* → *expression expression_list_tail*
17.1.2 *expression_list_tail* → , *expression expression_list_tail*
17.1.3 *expression_list_tail* → Ɛ
18.1 *expression* → *simple_expression*
18.2 *expression* → *simple_expression* **relop** *simple_expression*
19.1.1 *simple_expression* → *term simple_expression_tail*
19.1.2 *simple_expression* → *sign term simple_expression_tail*
19.1.3 *simple_expression_tail* → **addop** *term simple_expression_tail*
19.1.4 *simple_expression_tail* → Ɛ
20.1.1 *term* → *factor term_tail*
20.1.2 *term_tail* → **mulop** *factor term_tail*
20.1.3 *term_tail* → Ɛ
21.1 *factor* → **id**
21.3 *factor* → ***num***
21.4 *factor* → (*expression*)
21.5 *factor* → **not** *factor*
21.6 *factor* → **id** [*expression*]
22.1 *sign* → +
22.2 *sign* → **-**

Firsts/Follows

Production	Firsts	Follows
program	program	$
program'	procedure begin var	$
program''	procedure begin	$
identifier_list	id	)
identifier_list'	, e	)
declarations	var	procedure begin
declarations'	var e	procedure begin
type	integer real array	;)
standard_type	integer real	;)
subprogram_declarations	procedure	begin
subprogram_declarations'	procedure e	begin
subprogram_declaration	procedure	;
subprogram_declaration'	begin procedure var	;
subprogram_declaration''	procedure begin	;
subprogram_head	procedure	begin procedure var
subprogram_head'	(;	begin procedure var
arguments	(	;
parameter_list	id	)
parameter_list'	; e	)
compound_statement	begin	. ; end
compound_statement'	id call begin while if end	. ; end else
optional_statements	id call begin while if	end
statement_list	id call begin while If	end
statement_list'	; e	end
statement	id call begin while if	Else ; end
statement'	else e	Else ; end
variable	id	assignop
variable'	[e	assignop
procedure_statement	call	Else ; end
procedure_statement'	(e	Else ; end
expression_list	id num (not + -	)
expression_list'	, e	)
expression_list	id num (not + -	)] do then , else ; end
expression'	relop e	)] do then , else ; end
simple_expression	id num (not + -	Relop)] do then , else ; end
simple_expression'	addop e	Relop)] do then , else ; end
term	id num (not	addop relop)] do then , else ; end
term'	mulop e	addop relop)] do then , else ; end
factor	id num (not	mulop addop relop)] do then , else ; end
factor'	[, e	mulop addop relop)] do then , else ; end
sign	+ -	id num (not

Parse Table

	program	procedure	begin	end
program	program → program id (id_list) ; program'			
program'		program' →sub_decs cmpd_stmt .	program' → cmpd_stmt .	
program''		program'' → subprog_decs cmpd_stmt .	program'' → cmpd_stmt .	
identifier_list				
identifier_list'				
declarations				
declarations'		decs' → e	decs' → e	
type				
standard_type				
subprogram_declarations		subprgm_decs → subprgm_dec ; subprgm_decs'		
subprogram_declarations'		subprgm_decs → subprgm_dec ; subprgm_dec	subprgm_decs' → e	
subprogram_declaration		subprgm_head subprg_dec'		
subprogram_declaration'		subprgm_dec' → subprgm_decs cmpd_stmt	subprgm_dec → cmpd_stmt	
subprogram_declaration''		subprgm_dec'' → subprgm_decs cmpd_stmt	subprgm_dec'' → cmpd_stmt	
subprogram_head		subprgm_head → procedure id subprgm_head'		
subprogram_head'				
arguments		Arguments → (parameter_list)		
parameter_list				
parameter_list'				
compound_statement			cmpd_stmt → begin cmpd_stmt'	
compound_statement'			cmpd_stmt' → opt_stmts end	cmpd_stmt' → end
optional_statements			opt_stmts → stmt_list	
statement_list			stmt_list → stmt stmt_list'	
statement_list'				stmt_list → e
statement			Stmt → cmpd_stmt	
statement'				Stmt' → e
variable				
variable'				
procedure_statement				
procedure_statement'				procedure_stmt' → e
expression_list				
expression_list'				
expression				
expression'				Exp' → e
simple_expression				
simple_expression'				simple_expression' → e
term				
term'				Term' → e
factor				
factor'				Factor' → e
sign				

end	var	id	integer	real	array
	program' → declarations program''				
		id_list → id id_list'			
	decs → var id : type ; decs'				
	decs' → var id : type ; decs'				
			type → stdtype	type → stdtype	type → array [num .. num] of standard_type
			stdtype → integer	stdtype → real	
	subprgm_dec' → decs subprgm_dec''				
		param_list → id : type param_list'			
cmpd_stmt' → end		cmpd_stmt' → opt_stmts end			
		opt_stmts → stmt_list			
		stmt_list → stmt stmt_list'			
stmt_list → e					
		Stmt → variable assignop expression			
Stmt' → e					
		Variable → id variable'			
procedure_stmt' → e					
		expression_list → expression expression_list			
		Expression → simple_exp exp'			
Exp' → e					
		simple_expression → term simple_expr'			
simple_expression' → e					
		Term → factor term'			
Term' → e					
		Factor → id factor'			

call	do	while	if	then	else	num
cmpd_stmt' → opt_stmts end		cmpd_stmt' → opt_stmts end	cmpd_stmt' → opt_stmts end			
opt_stmts → stmt_list		opt_stmts → stmt_list	opt_stmts → stmt_list			
stmt_list → stmt stmt_list'		stmt_list → stmt stmt_list'	stmt_list → stmt stmt_list'			
Stmt → procedure_statement		Stmt → while exp do stmt	Stmt → if expression then stmt stmt'			
					Stmt' → else stmt OR stmt' → e	
procedure_stmt → call id procedure_stmt'						
					procedure_stmt' → e	
						expression_list → expression expression_li
						Expression → simple_exp exp'
	Exp' → e			Exp'	Exp' → e	
						simple_expression → term simple_expr'
	simple_expression' → e			simp	simple_expression' → e	
						Term → factor term'
	Term' → e			Term	Term' → e	
						Factor → num
	Factor' → e			Fact	Factor' → e	

not	relop	addop	mulop	assignop	[	]
				Variable' → e	Variable' → [expression]	
expression_list → expression expression_list						
Expression → simple_exp exp'						
	Expression' → relop simple_exp					E
simple_expression → term simple_expr'						
	simple_expression' → e	simple_expression' → addop term simple_expr'				s
Term → factor term'						
	Term' → e	Term' → e	Term' → mulop factor term'			T
Factor → not factor					Factor' → [exp]	
	Factor' → e	Factor' → e	Factor' → e			f

]	+	-	(	)	,	;
				id_list' → e	id_list' → , id id_list'	
			subprgm_head' → args ;			subprgm_head' → ;
				param_list' → e		param_list' → ; id : type param_list'
						stmt_list' → ; stmt_list'
						Stmt' → e
]						
			procedure_stmt' → (expr_list)			procedure_stmt' → e
	expression_list → expression expression_lis	expression_list → expression expression_lis	expression_list → expression expression_list			
				exp_list' → e	exp_list' - , exp exp_list'	
	Expression → simple_exp exp'	Expression → simple_exp exp'	Expression → simple_exp exp'			
	Exp' → e			Exp' → e	Exp' → e	Exp' → e
	simple_expression → sign term simple_exp	simple_expression → sign term simple_exp	simple_expression → term simple_expr'			
	simple_expression' → e			simple_express	simple_expression'	simple_expression' → e
			Term → factor term'			
	Term' → e			Term' → e	Term' → e	Term' → e
			Factor → (exp)			
	factor' → e			Factor' → e	Factor' → e	Factor' → e
	Sign → +	Sign → -				

Left factoring of grammar (step 3) – 10/27/2016

1.1 *program* → **program id** (*identifier_list*) ; *program_rest*

1.2.3 *program_rest* → *subprogram_declarations compound_statement* .
1.2.4 *program_rest* → *compound_statement* .
1.2.1 *program_rest* → *declarations program_rest'*.

1.3.1.1 *program_rest'* → *subprogram_declarations compound_statement* .
1.3.1.2 *program_rest'* → *compound_statement* .

2.1.1 *identifier_list* → **id** *identifier_list_tail*
2.1.2 *identifier_list_tail* → , **id** *identifier_list_tail*
2.1.3 *identifier_list_tail* → Ɛ

3.1.1 *declarations* → **var id** : *type* ; *declarations_tail*
3.1.2 *declarations_tail* → **var id** : *type* ; *declarations_tail*
3.1.3 *declarations_tail* → Ɛ

4.1 *type* → *standard_type*
4.2 *type* → **array** [**num** .. **num**] **of** *standard_type*

5.1 *standard_type* → **integer**
5.2 *standard_type* → **real**

6.1.1 *subprogram_declarations* → *subprogram_declaration ; subprogram_declarations_tail*
6.1.2 *subprogram_declarations_tail* → *subprogram_declaration* ; *subprogram_declarations_tail*
6.1.3 *subprogram_declarations_tail* → Ɛ

7.1 *subprogram_declaration* → *subprogram_head declarations subprogram_declaration_part*
7.2.1 *subprogram_declaration_part* → *compound_statement*
7.2.2 *subprogram_declaration_part* → *subprogram_declarations compound_statement*
7.2.3 *subprogram_declaration_part* → *declarations subprogram_declarations_tail_tail*
7.3.1 *subprogram_declarations_tail_tail* → *subprogram_declarations compound_statement*
7.3.2 *subprogram_declarations_tail_tail* → *compound_statement*

8.1 *subprogram_head* → **procedure id** *subprogram_head_part*
8.2.1 *subprogram_head_part* → *arguments ;*
8.2.2 *subprogram_head_part* → *;*

9.1 arguments → (*parameter_list*)

10.1.1 *parameter_list* → **id** : *type parameter_list_tail*
10.1.2 *parameter_list_tail* → ; **id** : *type parameter_list_tail*
10.1.3 *parameter_list_tail* → Ɛ

11.1.1 *compound_statement* → **begin** *compound_statement_rest*
11.1.2 *compound_statement_rest* → *optional_statements* **end**

11.1.3 *compound_statement_rest* → **end**

12.1 *optional_statements* → *statement_list*

13.1.1 *statement_list* → *statement statement_list_tail*
13.1.2 *statement_list_tail* → ; *statement statement_list_tail*
13.1.3 *statement_list_tail* → Ɛ

14.1 *statement* → *variable* **assignop** *expression*
14.2 *statement* → *procedure_statement*
14.3 *statement* → *compound_statement*
14.4 *statement* → **while** *expression* **do** *statement*
14.5 statement → **if** *expression* **then** *statement statement_part*
14.6.1 statement_part → **else** *statement*
14.6.2 statement_part → Ɛ

15.1 *variable* → **id** *variable_part*
15.2.1 *variable_part* → [*expression*]
15.2.2 *variable_part* → Ɛ

16.1 *procedure_statement* → **call id** *procedure_statement_rest*
16.2 *procedure_statement_rest* → (*expression_list*)
16.3 *procedure_statement_rest* → Ɛ

17.1.1 *expression_list* → *expression expression_list_tail*
17.1.2 *expression_list_tail* → , *expression expression_list_tail*
17.1.3 *expression_list_tail* → Ɛ

18.1.1 *expression* → *simple_expression expression_part*
18.2.1 *expression_part* → **relop** *simple_expression*
18.2.2 *expression_part* → Ɛ

19.1.1 *simple_expression* → *term simple_expression_tail*
19.1.2 *simple_expression* → *sign term simple_expression_tail*
19.1.3 *simple_expression_tail* → **addop** *term simple_expression_tail*
19.1.4 *simple_expression_tail* → Ɛ

20.1.1 *term* → *factor term_tail*
20.1.2 *term_tail* → **mulop** *factor term_tail*
20.1.3 *term_tail* → Ɛ

21.1 *factor* → **id** *factor_part*
21.2 *factor* → ***num***
21.3 *factor* → (*expression*)
21.4 *factor* → **not** *factor*
21.5.1 *factor_part* → [*expression*]
21.5.3 *factor_part* → Ɛ

22.1 *sign* → +

22.2 *sign* → **-**

Implementation

I used the Java programming language to create the syntaxanalyzer. I am using Git for version control of my code.

Discussion and Conclusions

Massaging the grammar was a lengthy process. After creating the parse table, programming the syntax analyzer was trivial.

References

Aho, Alfred et al. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, 1986. p. 746.

Appendix I: Sample Inputs and Outputs

Input: “Source.txt” (Aho, 746.)

```
program example(input, output);
var x: integer;
var y: integer;
var c: real;
var d: real;
procedure gcd(a: integer; b: integer);
begin
	if b = 0 then gcd := a
	else gcd := b
end;

begin
	if x = 5 then y := 5
end.
```

Output: “Source.txt”: Token file

```
Line No.	Lexeme		TOKEN-TYPE	ATTRIBUTE
2		program		7 (RES)		0
2		example		25 (ID)		0 (ptr to sym tab)
2		(		4 (CATCHALL)	3 (LEFTPAREN)
2		input		25 (ID)		1 (ptr to sym tab)
2		,		4 (CATCHALL)	7 (COMMA)
2		output		25 (ID)		2 (ptr to sym tab)
2		)		4 (CATCHALL)	4 (RIGHTPAREN)
2		;		4 (CATCHALL)	5 (SEMICOLON)
3		var		8 (RES)		0
3		x		25 (ID)		3 (ptr to sym tab)
3		:		4 (CATCHALL)	6 (COLON)
3		integer		13 (RES)	0
3		;		4 (CATCHALL)	5 (SEMICOLON)
4		var		8 (RES)		0
4		y		25 (ID)		4 (ptr to sym tab)
4		:		4 (CATCHALL)	6 (COLON)
4		integer		13 (RES)	0
4		;		4 (CATCHALL)	5 (SEMICOLON)
5		var		8 (RES)		0
5		c		25 (ID)		5 (ptr to sym tab)
5		:		4 (CATCHALL)	6 (COLON)
5		real		16 (RES)	0
5		;		4 (CATCHALL)	5 (SEMICOLON)
6		var		8 (RES)		0
6		d		25 (ID)		6 (ptr to sym tab)
6		:		4 (CATCHALL)	6 (COLON)
6		real		16 (RES)	0
6		;		4 (CATCHALL)	5 (SEMICOLON)
7		procedure	17 (RES)		0
7		gcd		25 (ID)		7 (ptr to sym tab)
7		(		4 (CATCHALL)	3 (LEFTPAREN)
7		a		25 (ID)		8 (ptr to sym tab)
7		:		4 (CATCHALL)	6 (COLON)
7		integer		13 (RES)	0
7		;		4 (CATCHALL)	5 (SEMICOLON)
7		b		25 (ID)		9 (ptr to sym tab)
7		:		4 (CATCHALL)	6 (COLON)
7		integer		13 (RES)	0
7		)		4 (CATCHALL)	4 (RIGHTPAREN)
```

```
7                   ;                   4 (CATCHALL)  5 (SEMICOLON)
8                   begin         5 (RES)               0
9                   if            10 (RES)                    0
9                   b                   25 (ID)       loc9 (ptr to sym tab)
9                   =                   1 (RELOP)     1 (EQ)
9                   0                   26 (INT)      0 (NULL)
9                   then          11 (RES)                    0
9                   gcd                 25 (ID)       loc7 (ptr to sym tab)
9                   :=                  21 (ASSIGNOP) 1 (ASSIGN)
9                   a                   25 (ID)       loc8 (ptr to sym tab)
10                  else          12 (RES)                    0
10                  gcd                 25 (ID)       loc7 (ptr to sym tab)
10                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
10                  b                   25 (ID)       loc9 (ptr to sym tab)
11                  end           6 (RES)               0
11                  ;                   4 (CATCHALL)  5 (SEMICOLON)
13                  begin         5 (RES)               0
14                  if            10 (RES)                    0
14                  x                   25 (ID)       loc3 (ptr to sym tab)
14                  =                   1 (RELOP)     1 (EQ)
14                  5                   26 (INT)      0 (NULL)
14                  then          11 (RES)                    0
14                  y                   25 (ID)       loc4 (ptr to sym tab)
14                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
14                  5                   26 (INT)      0 (NULL)
15                  end           6 (RES)               0
15                  .                   4 (CATCHALL)  8 (DOT)
                    98 (EOF)                  0 (NULL)
```

Output: “Source.txt”: Listing File

```
1
2           program example(input, output);
3           var x: integer;
4           var y: integer;
5           var c: real;
6           var d: real;
7           procedure gcd(a: integer; b: integer);
8           begin
9                   if b = 0 then gcd := a
10                  else gcd := b
11          end;
12
13          begin
14                  if x = 5 then y := 5
15          end.
```

Input: “SourceSynErrors.txt”: File With Syntax Errors

```
program example(input, output);
var x; integer;
var y; integer;
var c;;
var d; real;
procedure gcd(a: integer; b: integer):
begin
	if b = 0 then gcd := a
	else gcd := b
end.

begin
	if x = 5 then y := 5
end.
```

Output: “SourceSynErrors.txt”: Token File

```
Line No.	Lexeme		TOKEN-TYPE	ATTRIBUTE
3		program		7 (RES)			0
3		example			25 (ID)			0 (ptr to sym tab)
3		(			4 (CATCHALL)	3 (LEFTPAREN)
3		input			25 (ID)			1 (ptr to sym tab)
3		,			4 (CATCHALL)		7 (COMMA)
3		output			25 (ID)			2 (ptr to sym tab)
3		)			4 (CATCHALL)	4 (RIGHTPAREN)
3		;			4 (CATCHALL)	5 (SEMICOLON)
4		var			8 (RES)			0
4		x			25 (ID)			3 (ptr to sym tab)
4		;			4 (CATCHALL)	5 (SEMICOLON)
4		integer		13 (RES)			0
4		;			4 (CATCHALL)	5 (SEMICOLON)
5		var			8 (RES)			0
5		y			25 (ID)			4 (ptr to sym tab)
5		;			4 (CATCHALL)	5 (SEMICOLON)
5		integer		13 (RES)			0
5		;			4 (CATCHALL)	5 (SEMICOLON)
6		var			8 (RES)			0
6		c			25 (ID)			5 (ptr to sym tab)
6		;			4 (CATCHALL)	5 (SEMICOLON)
6		;			4 (CATCHALL)	5 (SEMICOLON)
7		var			8 (RES)			0
7		d			25 (ID)			6 (ptr to sym tab)
7		;			4 (CATCHALL)	5 (SEMICOLON)
7		real			16 (RES)			0
7		;			4 (CATCHALL)	5 (SEMICOLON)
8		procedure		17 (RES)				0
8		gcd			25 (ID)			7 (ptr to sym tab)
8		(			4 (CATCHALL)	3 (LEFTPAREN)
8		a			25 (ID)			8 (ptr to sym tab)
8		:			4 (CATCHALL)		6 (COLON)
8		integer		13 (RES)			0
8		;			4 (CATCHALL)	5 (SEMICOLON)
8		b			25 (ID)			9 (ptr to sym tab)
8		:			4 (CATCHALL)		6 (COLON)
8		integer		13 (RES)			0
8		)			4 (CATCHALL)	4 (RIGHTPAREN)
8		:			4 (CATCHALL)		6 (COLON)
9		begin		5 (RES)			0
10		if			10 (RES)			0
```

```
10                  b                   25 (ID)       loc9 (ptr to sym tab)
10                  =                   1 (RELOP)     1 (EQ)
10                  0                   26 (INT)      0 (NULL)
10                  then          11 (RES)                  0
10                  gcd                 25 (ID)       loc7 (ptr to sym tab)
10                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
10                  a                   25 (ID)       loc8 (ptr to sym tab)
11                  else          12 (RES)                  0
11                  gcd                 25 (ID)       loc7 (ptr to sym tab)
11                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
11                  b                   25 (ID)       loc9 (ptr to sym tab)
12                  end           6 (RES)               0
12                  .                   4 (CATCHALL)  8 (DOT)
14                  begin         5 (RES)               0
15                  if            10 (RES)                  0
15                  x                   25 (ID)       loc3 (ptr to sym tab)
15                  =                   1 (RELOP)     1 (EQ)
15                  5                   26 (INT)      0 (NULL)
15                  then          11 (RES)                  0
15                  y                   25 (ID)       loc4 (ptr to sym tab)
15                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
15                  5                   26 (INT)      0 (NULL)
16                  end           6 (RES)               0
16                  .                   4 (CATCHALL)  8 (DOT)
                    98 (EOF)                  0 (NULL)
```

Output: “SourceLexErrors.txt”: Listing File

```
1
2
3               program example(input, output);
4               var x; integer;
SYNTAX ERROR: Expected :, received ;
5               var y; integer;
SYNTAX ERROR: Expected :, received ;
6               var c;;
SYNTAX ERROR: Expected :, received ;
SYNTAX ERROR: Expecting one of 'integer', 'real', 'array',  given: ;
7               var d; real;
SYNTAX ERROR: Expected :, received ;
8               procedure gcd(a: integer; b: integer):
SYNTAX ERROR: Expected ;, received :
9               begin
10                      if b = 0 then gcd := a
11                      else gcd := b
12              end.
SYNTAX ERROR: Expected ;, received .
13
14              begin
15                      if x = 5 then y := 5
16              end.
```

Input: “SourceLexSynErrors.txt”: File With Lexical and Syntax Errors

```
program example(input, output);
var x; @;
var y:= #;
var c;
var d; longreal;
procedure gcd(a: integer; b: integer):
begin
	if b = 01 then gcd := a
	else gcd := b
end.

begin
	if x = 5 then y := 5
end.
```

Output: “SourceLexSynErrors.txt”: Token File

```
Line No.	Lexeme		TOKEN-TYPE	ATTRIBUTE
3		program		7 (RES)			0
3		example			25 (ID)			0 (ptr to sym tab)
3		(			4 (CATCHALL)	3 (LEFTPAREN)
3		input			25 (ID)			1 (ptr to sym tab)
3		,			4 (CATCHALL)		7 (COMMA)
3		output			25 (ID)			2 (ptr to sym tab)
3		)			4 (CATCHALL)	4 (RIGHTPAREN)
3		;			4 (CATCHALL)	5 (SEMICOLON)
4		var		8 (RES)			0
4		x			25 (ID)			3 (ptr to sym tab)
4		;			4 (CATCHALL)	5 (SEMICOLON)
4		@			99 (LEXERR)	1 (UNRECOGSYM)
4		;			4 (CATCHALL)	5 (SEMICOLON)
5		var		8 (RES)			0
5		y			25 (ID)			4 (ptr to sym tab)
5		:=			21 (ASSIGNOP)	1 (ASSIGN)
5		#			99 (LEXERR)	1 (UNRECOGSYM)
5		;			4 (CATCHALL)	5 (SEMICOLON)
6		var		8 (RES)			0
6		c			25 (ID)			5 (ptr to sym tab)
6		;			4 (CATCHALL)	5 (SEMICOLON)
7		var		8 (RES)			0
7		d			25 (ID)			6 (ptr to sym tab)
7		;			4 (CATCHALL)	5 (SEMICOLON)
7		longreal			25 (ID)			7 (ptr to sym tab)
7		;			4 (CATCHALL)	5 (SEMICOLON)
8		procedure		17 (RES)			0
8		gcd			25 (ID)			8 (ptr to sym tab)
8		(			4 (CATCHALL)	3 (LEFTPAREN)
8		a			25 (ID)			9 (ptr to sym tab)
8		:			4 (CATCHALL)		6 (COLON)
8		integer		13 (RES)			0
8		;			4 (CATCHALL)	5 (SEMICOLON)
8		b			25 (ID)			10 (ptr to sym tab)
8		:			4 (CATCHALL)		6 (COLON)
8		integer		13 (RES)			0
8		)			4 (CATCHALL)	4 (RIGHTPAREN)
8		:			4 (CATCHALL)		6 (COLON)
9		begin		5 (RES)			0
10		if		10 (RES)			0
10		b			25 (ID)		loc10 (ptr to sym tab)
```

```
10                  =                   1 (RELOP)     1 (EQ)
10                  01            99 (LEXERR)   5 (LEADZERO)
10                  then          11 (RES)                  0
10                  gcd                 25 (ID)       loc8 (ptr to sym tab)
10                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
10                  a                   25 (ID)       loc9 (ptr to sym tab)
11                  else          12 (RES)                  0
11                  gcd                 25 (ID)       loc8 (ptr to sym tab)
11                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
11                  b                   25 (ID)       loc10 (ptr to sym tab)
12                  end           6 (RES)               0
12                  .                   4 (CATCHALL)  8 (DOT)
14                  begin         5 (RES)               0
15                  if            10 (RES)                  0
15                  x                   25 (ID)       loc3 (ptr to sym tab)
15                  =                   1 (RELOP)     1 (EQ)
15                  5                   26 (INT)      0 (NULL)
15                  then          11 (RES)                  0
15                  y                   25 (ID)       loc4 (ptr to sym tab)
15                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
15                  5                   26 (INT)      0 (NULL)
16                  end           6 (RES)               0
16                  .                   4 (CATCHALL)  8 (DOT)
                    98 (EOF)                  0 (NULL)
```

Output: “SourceLexSynErrors.txt”: Listing File

```
1
2
3               program example(input, output);
4               var x; @;
LEXERR:Unrecognized Symbol:  @
SYNTAX ERROR: Expected :, received ;
SYNTAX ERROR: Expecting one of 'integer', 'real', 'array',  given: @
5               var y:= #;
LEXERR:Unrecognized Symbol:  #
SYNTAX ERROR: Expected :, received :=
SYNTAX ERROR: Expecting one of 'integer', 'real', 'array',  given: #
6               var c;
SYNTAX ERROR: Expected :, received ;
7               var d; longreal;
SYNTAX ERROR: Expecting one of 'integer', 'real', 'array',  given: var
SYNTAX ERROR: Expecting one of 'var''procedure', 'begin' given longreal
8               procedure gcd(a: integer; b: integer):
SYNTAX ERROR: Expected ;, received :
9               begin
10                      if b = 01 then gcd := a
LEXERR:Leading zero:  01
SYNTAX ERROR: Expecting one of 'id', 'num''not', '+', '-', '(' given 01
11                      else gcd := b
12              end.
SYNTAX ERROR: Expected ;, received .
13
14              begin
15                      if x = 5 then y := 5
16              end.
```

Appendix II: Program Listings

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.Scanner;
/**
 * @author Jacob Sherrill
 */



public class LexicalAnalyzer {
	static Token token = new Token(0, 0, 0);
	static char[] buffer = new char[72];
	static int lineCounter = 0;
	static int f = 0;	// Forward pointer for buffer
	static int b = 0;	// Back pointer for buffer
	static int c = 0;	// Digit counter for int, real, longreal machines
	static String idRes = "";
	static String numString = "";
	static LinkedList reservedWordTable = new LinkedList();
	static LinkedList symbolTable = new LinkedList();
	public static LinkedList<Token> tokenList = new LinkedList<Token>();

	public static LinkedList<String> listingList = new LinkedList<String>();

	static File listingFile = new File("Listing.txt");
	static File tokenFile = new File("Token");
	static File reservedFile = new File("Reserved.txt");
	static File sourceFile = new File("SourceLexSynErrors.txt");

	static PrintWriter listingWriter;
	static PrintWriter tokenWriter;
	static Scanner reservedScanner;
	static Scanner sourceScanner;

	public static void main(String [] args) throws FileNotFoundException {
		init();//Load source(r), reserved(r), listing(w), token(w)
		clearBuffer();
		getNextToken();

		// While not at the end of the source file
		while(sourceScanner.hasNextLine()) {
			// Put the source file line into the buffer
			// TODO I flipped these
			lineCounter++;
			getNextLine();

			// Write the buffer line to the listing file
			// LEXERRs will be written below each line
//			writeBufferToListing();

			// Get all tokens on the line of the source file
			getNextToken();
			if(!(sourceScanner.hasNextLine())) {
				// Last token is (EOF, NULL)
				tokenWriter.printf("\t\t\t98 (EOF)"
							+ "\t\t\t0 (NULL)\n");
				tokenList.add(new Token(98, 0, lineCounter));
			}
		}
```

```
        // TODO token printer
        //            FOR(INT I = 0; I < TOKENLIST.SIZE(); I++) {
        //                  SYSTEM.OUT.PRINTLN(TOKENLIST.GET(I).TOKENTYPE + " " +
        //                                    TOKENLIST.GET(I).ATTRIBUTE);
        //            }
        lexicalAnalyzer();
    }
    private static String getTokenReference(int type, int attribute) {
     if(type == 1) {
            if(attribute == 1) {
                  return "=";
            }
            if(attribute == 2) {
                  return "<>";
            }
            if(attribute == 3) {
                  return "<";
            }
            if(attribute == 4) {
                  return "<=";
            }
            if(attribute == 5) {
                  return ">=";
            }
            if(attribute == 6) {
                  return ">";
            }
     }
     if(type == 2) {
            if(attribute == 1) {
                  return "+";
            }
            if(attribute == 2) {
                  return "-";
            }
            if(attribute == 3) {
                  return "or";
            }
     }
     if(type == 3) {
            if(attribute == 1) {
                  return "*";
            }
            if(attribute == 2) {
                  return "/";
            }
            if(attribute == 3) {
                  return "div";
            }
            if(attribute == 4) {
                  return "mod";
            }
            if(attribute == 5) {
                  return "and";
            }
     }
     if(type == 4) {
            if(attribute == 1) {
                  return "[";
            }
            if(attribute == 2) {
                  return "]";
            }
            if(attribute == 3) {
```

```
		return "(";
	}
	if(attribute == 4) {
		return ")";
	}
	if(attribute == 5) {
		return ";";
	}
	if(attribute == 6) {
		return ":";
	}
	if(attribute == 7) {
		return ",";
	}
	if(attribute == 8) {
		return ".";
	}
	if(attribute == 9) {
		return "..";
	}

}
if(type == 5) {
	return "begin";
}
if(type == 6) {
	return "end";
}
if(type == 7) {
	return "program";
}
if(type == 8) {
	return "var";
}
if(type == 9) {
	return "function";
}
if(type == 10) {
	return "if";
}
if(type == 11) {
	return "then";
}
if(type == 12) {
	return "else";
}
if(type == 13) {
	return "integer";
}
if(type == 14) {
	return "array";
}
if(type == 15) {
	return "of";
}
if(type == 16) {
	return "real";
}
if(type == 17) {
	return "procedure";
}
if(type == 18) {
	return "while";
}
```

```
 if(type == 19) {
       return "do";
 }
 if(type == 20) {
       return "not";
 }
 if(type == 21) {
       return ":=";
 }
 if(type == 22) {
       return "id";   // Placeholder id
 }
 if(type == 23) {
       return "longreal";
 }
 if(type == 24) {
       return "call";
 }
 if(type == 99) {
       return "LEXERR";
 }

       return null;
}



/**
 * Opens source(r), reserved(r), listing(w), token(w) files
 * @throws FileNotFoundException
 */
private static void init() throws FileNotFoundException {
       listingWriter = new PrintWriter("Listing.txt");
       tokenWriter = new PrintWriter(tokenFile);
       reservedScanner = new Scanner(reservedFile);
       sourceScanner = new Scanner(sourceFile);

       // Writes header on token file
       tokenWriter.printf("Line No.\tLexeme\t\tTOKEN-TYPE\tATTRIBUTE\n");

       // "Reserved word file: This file is read in during the
       // initialization process and its information is stored in the
       // reserved word table"
       // "Keywords are reserved and appear in boldface in the grammar"(p. 749)
       while(reservedScanner.hasNext()) {
              reservedWordTable.add(reservedScanner.next());
       }
       System.out.println("Files loaded for reading/writing");
}

/**
 * This method goes through machines and returns tokens
 * @return
 * @throws FileNotFoundException
 */
private static Object getNextToken() throws FileNotFoundException {
       // Now go through machines
       for(int i = 0; i < buffer.length && buffer[f] != '\u0000'; i++) {

              // Begin WHITESPACE machine code
              if(Character.isWhitespace(buffer[f]) || buffer[f] == '\n' ||
                            buffer[f] == '\t' || buffer[f] == '\r' ||
                            Character.isSpaceChar(buffer[f])) {
                     f++;
```

```
                        b = f;
                        while(Character.isWhitespace(buffer[f]) || buffer[f] == '\n' ||
                                        buffer[f] == '\t' || buffer[f] == '\r' ||
                                        Character.isSpaceChar(buffer[f])) {
                                f++;
                                b = f;
                                getNextToken();
                        }
                }
                // End WHITESPACE machine code

                // Begin ID/RES machine
                if(Character.isLetter(buffer[f])) {
                        idRes += buffer[f];
                        f++;
                        while(Character.isLetter(buffer[f])
                                        || Character.isDigit(buffer[f])) {
                                idRes += buffer[f];
                                f++;
                        }
                        if(idRes.length() <= 10) {
                                reservedScanner = new Scanner(reservedFile);
                                // If it's a reserved word,
                                if(reservedWordTable.contains(idRes)) {

                                        while(reservedScanner.hasNext()) {
                                                String next = reservedScanner.next();
                                                if(idRes.equals(next)) {
                                                        int tokenType = reservedScanner.nextInt();
                                                        int attr = reservedScanner.nextInt();
                                                        tokenWriter.printf(lineCounter + "\t\t\t"
                                                                        + idRes+ "\t\t" + tokenType
                                                                        + " (RES)\t\t\t"
                                                                        + attr + "\n");
                                                        tokenList.add(new Token(tokenType, attr,
                                                                        idRes, lineCounter));
                                                }
                                        }
                                }
                                //                                      Else it's not a reserved
word, and instead is an ID!
                                else {
                                        // If the symbol table contains the ID,
                                        if(symbolTable.contains(idRes)) {
                                                tokenWriter.printf(lineCounter + "\t\t\t" + idRes
                                                                + "\t\t\t25 (ID)\t\t"
                                                                + "loc" + symbolTable.indexOf(idRes)
                                                                + " (ptr to sym tab)"
                                                                + "\n");
                                                tokenList.add(new Token(25,
                                                                symbolTable.indexOf(idRes)
                                                                , idRes, lineCounter));
                                        }
                                        // Else the ID is not in the symbol table
                                        else {
                                                symbolTable.add(idRes);
                                                tokenWriter.printf(lineCounter + "\t\t\t" + idRes
                                                                + "\t\t\t25 (ID)\t\t\t"
                                                                + symbolTable.indexOf(idRes)
                                                                + " (ptr to sym tab)"
                                                                + "\n");
                                                tokenList.add(new Token(25, 0, idRes,
lineCounter));
                                        }
```

```
                        }
                        b = f;
                        idRes = "";
                        getNextToken();
                    }
                    else if(idRes.length() > 10) {
                        listingList.set(lineCounter-1, listingList.get(lineCounter-1)
                                + "LEXERR:\tWord too long:\t"
                                + idRes +"\n");
//                      listingWriter.printf("LEXERR:\tWord too long:\t"
//                              + idRes +"\n");
                        tokenWriter.printf(lineCounter + "\t\t\t" + idRes
                                +  "\t99 (LEXERR)\t\t"
                                + "6 (LONGWORD)\t\n");
                        tokenList.add(new Token(99, 6, lineCounter));
                        f++;
                        idRes = "";
                    }

                }
                // End ID/RES machine

                // Begin ASSIGNOP machine code
                if(buffer[f] == ':') {
                    f++;
                    if(buffer[f] == '=') {
                        tokenWriter.printf(lineCounter + "\t\t\t:="
                                + "\t\t\t21 (ASSIGNOP)\t1 (ASSIGN)\n");
                        tokenList.add(new Token(21, 0, lineCounter));
                        f++;
                        b = f;
                        getNextToken();
                    }
                    else if(buffer[f] != '=') {
                        f = b;
                    }
                }
                // Begin RELOP machine: <>, <=, <, =, >=, >
                if(buffer[f] == '<') {
                    f++;
                    // NE
                    if(buffer[f] == '>') {
                        tokenWriter.printf(lineCounter + "\t\t\t<>"
                                + "\t\t\t1 (RELOP)\t2 (NE)\n");
                        tokenList.add(new Token(1, 2, lineCounter));
                        f++;
                        b = f;
                        getNextToken();
                    }
                    // LE
                    else if(buffer[f] == '=') {
                        f++;
                        tokenWriter.printf(lineCounter + "\t\t\t<=\t\t\t1 (RELOP)"
                                + "\t\t4 (LE)\n");
                        tokenList.add(new Token(1, 4, lineCounter));
                        b = f;
                        getNextToken();
                    }
                    // LT/other
                    else {
                        tokenWriter.printf(lineCounter + "\t\t\t<\t\t\t1 (RELOP)"
                                + "\t3 (LT)\n");
                        tokenList.add(new Token(1, 3, lineCounter));
                        b = f;
```

```
                        getNextToken();
                }
        }
        if(buffer[f] == '=') {
                f++;
                tokenWriter.printf(lineCounter + "\t\t\t=\t\t\t1 (RELOP)"
                                + "\t1 (EQ)\n");
                tokenList.add(new Token(1, 1, lineCounter));
                b = f;
                getNextToken();
        }
        if(buffer[f] == '>') {
                f++;
                if(buffer[f] == '=') {
                        f++;
                        tokenWriter.printf(lineCounter + "\t\t\t>=\t\t\t1 (RELOP)"
                                        + "\t5 (GE)\n");
                        tokenList.add(new Token(1, 5, lineCounter));
                        b = f;
                        getNextToken();
                }
                else {
                        tokenWriter.printf(lineCounter + "\t\t\t>\t\t\t1 (RELOP)\t"
                                        + "6 (GT)\n");
                        tokenList.add(new Token(1, 6, lineCounter));
                        b = f;
                        getNextToken();
                }
        }
        // End RELOP machine

        // LONGREAL machine only
        if(Character.isDigit(buffer[f])) {
                numString += buffer[f];
                // TODO changing
//              if(buffer[f] == '0' && Character.isDigit(buffer[f+1])) {
//                      // LEXERR: leading zero
//                      tokenWriter.printf(lineCounter + "\t\t\t" + numString
//                                      + "\t\t99 (LEXERR)\t"
//                                      + "5 (LEADZERO)\t" + "\n");
//                      tokenList.add(new Token(99, 5));
//                      listingWriter.printf("LEXERR:\tLeading zero"
//                                      + ":\t"+ numString +"\n");
//              }
                f++;
                c++;
                while(Character.isDigit(buffer[f]) && c <= 5) {
                        numString += buffer[f];
                        f++;
                        c++;
                }
                if(buffer[f] == '.') {
                        numString += buffer[f];
                        f++;
                        c = 0;
                        while(Character.isDigit(buffer[f]) && c <= 5) {
                                numString += buffer[f];
                                f++;
                                c++;
                        }

                        if(buffer[f] == 'E') {
                                if(c > 5) {
                                        // LEXERR: Digits after decimal point too long
```

```
                                        listingList.set(lineCounter-1,
listingList.get(lineCounter-1)
                                                + "LEXERR:\tReal second part too
long"
                                                + ":\t"+ numString +"\n");
                                        tokenWriter.printf(lineCounter + "\t\t\t" +
numString
                                                + "\t\t99 (LEXERR)\t"
                                                + "3 (RLLONGSCND)\t" + "\n");
                                        tokenList.add(new Token(99, 3, lineCounter));
//
too long"
                                        listingWriter.printf("LEXERR:\tReal second part
//
                                                + ":\t"+ numString +"\n");
                                }

                                numString += buffer[f];
                                f++;
                                c = 0;
                                // Sign logic
                                if(buffer[f] == '+' || buffer[f] == '-') {
                                        numString += buffer[f];
                                        f++;

                                        if(Character.isDigit(buffer[f])) {
                                                numString += buffer[f];
                                                f++;
                                                c++;
                                                while(Character.isDigit(buffer[f])) {
                                                        numString += buffer[f];
                                                        f++;
                                                        c++;
                                                }

                                                if(c > 2) {
                                                        // TODO I fixed this.
                                                        //LEXERR: Digits after decimal point
too long
                                                        tokenWriter.printf(lineCounter +
"\t\t\t" + numString
                                                                + "\t\t99 (LEXERR)\t"
                                                                + "4 (EXPLONG)\t" +
"\n");
                                                        tokenList.add(new Token(99, 4,
lineCounter));
                                                        listingList.set(lineCounter-1,
listingList.get(lineCounter-1)
                                                                + "LEXERR:\tExp long"
                                                                + ":\t"+ numString
+"\n");
                                                        listingWriter.printf("LEXERR:\tExp
//
long"
                                                                + ":\t"+ numString
//
+"\n");
                                                        numString = "";
                                                        getNextToken();

                                                }
                                                else {
                                                        // LONGREAL
                                                        tokenWriter.printf(lineCounter +
"\t\t\t"
                                                                + numString
                                                                + "\t\t28
(LONGREAL)\t"
```

```
                                                                        + "0 (NULL)\n");
                                                                tokenList.add(new Token(28, 0,
lineCounter));
                                                                numString = "";
                                                                getNextToken();

                                                        }
                                                }
                                        }
                                }
                                else if(Character.isDigit(buffer[f])) {

                                }


                        }
                        numString = "";
                        c = 0;
                        f = b;
                }

                // REAL machine only
                if(Character.isDigit(buffer[f])) {
                        int z = 1;
                        numString += buffer[f];
                        //                              if(buffer[f] == '0' &&
Character.isDigit(buffer[f+1])) {
                        //                                      // LEXERR: leading zero
                        //                                      tokenWriter.printf(lineCounter +
"\t\t\t" + numString
                        //                                                      + "\t\t99 (LEXERR)\t"
                        //                                                      + "5 (LEADZERO)\t" +
"\n");
                        //                                      tokenList.add(new Token(99, 5));
                        //
        listingWriter.printf("LEXERR:\tLeading zero"
                        //                                                      + ":\t"+ numString
+"\n");
                        //                              }
                        f++;
                        c++;
                        while(Character.isDigit(buffer[f])) {// && c <= 5) {
                                numString += buffer[f];
                                f++;
                                c++;
                                z++;
                        }

                        if(buffer[f] == '.') {



                                numString += buffer[f];
                                f++;
                                c = 0;
                                // TODO changing
                                while(Character.isDigit(buffer[f])) {// && c <= 5) {
                                        numString += buffer[f];
                                        f++;
                                        c++;
                                }
                                if (c > 5) {
                                        tokenWriter.printf(lineCounter + "\t\t\t" + numString
                                                        + "\t\t99 (LEXERR)\t"
```

```
                                                + "3 (RL2NDLONG)\t" + "\n");
                                        tokenList.add(new Token(99, 3, lineCounter));
                                        listingList.set(lineCounter-1,
listingList.get(lineCounter-1)
                                                + "LEXERR:\tReal 2nd part too long"
                                                + ":\t"+ numString +"\n");
//                                      listingWriter.printf("LEXERR:\tReal 2nd part too long"
//                                              + ":\t"+ numString +"\n");
                                        numString = "";
                                        c = 0;
                                        b = f;
                                        getNextToken();
                                }
                                else if(z > 5) {
                                        // LEXERR: real first part too long
                                        tokenWriter.printf(lineCounter + "\t\t\t" + numString
                                                + "\t\t99 (LEXERR)\t"
                                                + "10 (RL1STLONG)\t" + "\n");
                                        tokenList.add(new Token(99, 3, lineCounter));
                                        listingList.set(lineCounter-1,
listingList.get(lineCounter-1)
                                                + "LEXERR:\tReal 1st part too long"
                                                + ":\t"+ numString +"\n");
//                                      listingWriter.printf("LEXERR:\tReal 1st part too long"
//                                              + ":\t"+ numString +"\n");
                                        numString = "";
                                        z = 0;
                                        b = f;
                                        getNextToken();
                                }
                                // Check if buffer[f-1] isn't decimal
                                else if(Character.isDigit(buffer[f-1])) {
                                        c = 0;
                                        b = f;
                                        tokenWriter.printf(lineCounter + "\t\t\t"
                                                + numString
                                                + "\t\t27 (REAL)\t\t"
                                                + "0 (NULL)\n");
                                        tokenList.add(new Token(27, 0, lineCounter));
                                        numString = "";
                                        getNextToken();
                                }



                        }

                        else if(Character.isDigit(buffer[f])) {

                        }
                        numString = "";
                        c = 0;
                        f = b;
                }

                // INT machine only
                if(Character.isDigit(buffer[f])) {
                        numString += buffer[f];
                        //                      if(buffer[f] == '0' &&
Character.isDigit(buffer[f+1])) {
                        //                              // LEXERR: leading zero
                        //                              tokenWriter.printf(lineCounter +
"\t\t\t" + numString
                        //                                              + "\t\t99 (LEXERR)\t"
```

```
                                //                                                      + "5 (LEADZERO)\t" +
"\n");
                                //                                          tokenList.add(new Token(99, 5));
                                //
        listingWriter.printf("LEXERR:\tLeading zero"
                                //                                                      + ":\t"+ numString
+"\n");
                                //                                }
                                f++;
                                c++;
                                while(Character.isDigit(buffer[f]) && c <= 10) {
                                        numString += buffer[f];
                                        f++;
                                        c++;
                                }
                                if(c > 10) {
                                        // TODO: LEXERR: Int too long
                                        tokenWriter.printf(lineCounter + "\t\t\t" + numString
                                                        + "\t\t99 (LEXERR)\t"
                                                        + "7 (INTLONG)\t" + "\n");
                                        tokenList.add(new Token(99, 7, lineCounter));
                                        listingList.set(lineCounter-1, listingList.get(lineCounter-1)
                                                        + "LEXERR:\tInt too long"
                                                        + ":\t"+ numString +"\n");
//                                      listingWriter.printf("LEXERR:\tInt too long"
//                                                      + ":\t"+ numString +"\n");
                                }
                                else {
                                        if(numString.startsWith("0") && numString.length()
                                                        > 1) {
                                                // LEXERR: leading zero
                                                tokenWriter.printf(lineCounter + "\t\t\t" + numString
                                                                + "\t\t99 (LEXERR)\t"
                                                                + "5 (LEADZERO)\t" + "\n");
                                                tokenList.add(new Token(99, 5, numString, lineCounter));
                                                listingList.set(lineCounter-1,
listingList.get(lineCounter-1)
                                                                + "LEXERR:\tLeading zero"
                                                                + ":\t"+ numString +"\n");
//                                              listingWriter.printf("LEXERR:\tLeading zero"
//                                                              + ":\t"+ numString +"\n");
                                                numString = "";
                                                b = f;
                                                getNextToken();
                                        }
                                        else {
                                                tokenWriter.printf(lineCounter + "\t\t\t"
                                                                + numString
                                                                + "\t\t\t26 (INT)\t"
                                                                + "0 (NULL)\n");
                                                tokenList.add(new Token(26, 0, lineCounter));
                                                numString = "";
                                                b = f;
                                                getNextToken();
                                        }

                                }
                        }


                        // End LONGREAL, REAL, INT machine code
```

```
                        // Begin CATCHALL machine: ., .., [], (), ;, :, +, -, *, /, ,
                        if(buffer[f] == '.') {
                                f++;
                                if(buffer[f] == '.') {
                                        tokenWriter.printf(lineCounter + "\t\t\t.."
                                                        + "\t\t\t4 (CATCHALL)\t9 (DOTDOT)\n");
                                        tokenList.add(new Token(4, 9, lineCounter));
                                        f++;
                                        b = f;
                                        getNextToken();
                                }
                                else {
                                        tokenWriter.printf(lineCounter + "\t\t\t."
                                                        + "\t\t\t4 (CATCHALL)\t8 (DOT)\n");
                                        tokenList.add(new Token(4, 8, lineCounter));
                                        b = f;
                                        getNextToken();
                                }
                        }
//                      if(buffer[f] == '4') {
//                              tokenWriter.printf(lineCounter + "\t\t\t"
//                                              + numString
//                                              + "\t\t\t26 (INT)\t"
//                                              + "0 (NULL)\n");
//                              tokenList.add(new Token(26, 0));
//                              numString = "";
//                              b = f;
//                              getNextToken();
//                      }
                        if(buffer[f] == '[') {
                                tokenWriter.printf(lineCounter + "\t\t\t["
                                                + "\t\t\t4 (CATCHALL)\t1 (LEFTBRACK)\n");
                                tokenList.add(new Token(4, 1, lineCounter));
                                f++;
                                b = f;
                                getNextToken();
                        }
                        if(buffer[f] == ']') {
                                tokenWriter.printf(lineCounter + "\t\t\t]"
                                                + "\t\t\t4 (CATCHALL)\t2 (RIGHTBRACK)\n");
                                tokenList.add(new Token(4, 2, lineCounter));
                                f++;
                                b = f;
                                getNextToken();
                        }
                        if(buffer[f] == '(') {
                                tokenWriter.printf(lineCounter + "\t\t\t("
                                                + "\t\t\t4 (CATCHALL)\t3 (LEFTPAREN)\n");
                                tokenList.add(new Token(4, 3, lineCounter));
                                f++;
                                b = f;
                                getNextToken();
                        }
                        if(buffer[f] == ')') {
                                tokenWriter.printf(lineCounter + "\t\t\t)"
                                                + "\t\t\t4 (CATCHALL)\t4 (RIGHTPAREN)\n");
                                tokenList.add(new Token(4, 4, lineCounter));
                                f++;
                                b = f;
                                getNextToken();
                        }
                        if(buffer[f] == ';') {
                                tokenWriter.printf(lineCounter + "\t\t\t;"
                                                + "\t\t\t4 (CATCHALL)\t5 (SEMICOLON)\n");
```

```
        tokenList.add(new Token(4, 5, lineCounter));
        f++;
        b = f;
        getNextToken();
}
if(buffer[f] == ':') {
        tokenWriter.printf(lineCounter + "\t\t\t:"
                        + "\t\t\t4 (CATCHALL)\t\t6 (COLON)\n");
        tokenList.add(new Token(4, 6, lineCounter));
        f++;
        b = f;
        getNextToken();
}
if(buffer[f] == ',') {
        tokenWriter.printf(lineCounter + "\t\t\t,"
                        + "\t\t\t4 (CATCHALL)\t\t7 (COMMA)\n");
        tokenList.add(new Token(4, 7, lineCounter));
        f++;
        b = f;
        getNextToken();
}
if(buffer[f] == '+') {
        tokenWriter.printf(lineCounter + "\t\t\t+"
                        + "\t\t\t2 (ADDOP)\t\t1 (PLUS)\n");
        tokenList.add(new Token(2, 1, lineCounter));
        f++;
        b = f;
        getNextToken();
}
if(buffer[f] == '-') {
        tokenWriter.printf(lineCounter + "\t\t\t-"
                        + "\t\t\t2 (ADDOP)\t\t2 (MINUS)\n");
        tokenList.add(new Token(2, 2, lineCounter));
        f++;
        b = f;
        getNextToken();
}
if(buffer[f] == '*') {
        f++;
        b = f;
        tokenWriter.printf(lineCounter + "\t\t\t*"
                        + "\t\t\t3 (MULOP)\t\tMULT\n");
        tokenList.add(new Token(3, 1, lineCounter));
        getNextToken();
}
if(buffer[f] == '/') {
        f++;
        b = f;
        tokenWriter.printf(lineCounter + "\t\t\t/"
                        + "\t\t\t3 (MULOP)\t\t2 (DIV)\n");
        tokenList.add(new Token(3, 2, lineCounter));
        getNextToken();
}


// Begin source file newline
// If a newline character is encountered, break/move to next line
if(buffer[f] == '\u0000') {
        f = 0;
        b = 0;
        break;
}
// TODO i'm changing this
if(buffer[f] == 'E' && buffer[f+1] == '4') {
```

```
	if(symbolTable.contains("E4")) {
		tokenWriter.printf(lineCounter + "\t\t\t" + "E4"
				+ "\t\t\t25 (ID)\t\t"
				+ "loc" + symbolTable.indexOf("E4")
				+ " (ptr to sym tab)"
				+ "\n");
		tokenList.add(new Token(25,
				symbolTable.indexOf("E4"), lineCounter));
	}
	// Else the ID is not in the symbol table
	else {
		symbolTable.add("E4");
		tokenWriter.printf(lineCounter + "\t\t\t" + "E4"
				+ "\t\t\t25 (ID)\t\t\t"
				+ symbolTable.indexOf("E4")
				+ " (ptr to sym tab)"
				+ "\n");
		tokenList.add(new Token(25, 0, lineCounter));
	}
	f++;
	b = f;
}

// E
if(buffer[f] == 'E') {
	if(symbolTable.contains("E")) {
		tokenWriter.printf(lineCounter + "\t\t\t" + "E"
				+ "\t\t\t25 (ID)\t\t"
				+ "loc" + symbolTable.indexOf("E")
				+ " (ptr to sym tab)"
				+ "\n");
		tokenList.add(new Token(25,
				symbolTable.indexOf("E"), lineCounter));
	}
	// Else the ID is not in the symbol table
	else {
		symbolTable.add("E");
		tokenWriter.printf(lineCounter + "\t\t\t" + "E"
				+ "\t\t\t25 (ID)\t\t\t"
				+ symbolTable.indexOf("E")
				+ " (ptr to sym tab)"
				+ "\n");
		tokenList.add(new Token(25, 0, lineCounter));
	}
	f++;
	b = f;
}

// After all of the machines are through, if we haven't found
// a token that matches what's on the buffer, return an error
// Write to listing file and token file
if(!(Character.isDigit(buffer[f])) && !(Character.isWhitespace(buffer[f])
		|| buffer[f] == '\n'
		|| buffer[f] == '\t'
		|| buffer[f] == '\r'
		|| Character.isSpaceChar(buffer[f])
		|| buffer[f] == '%'
		|| buffer[f] == 'E'
		)) {
	tokenWriter.printf(lineCounter + "\t\t\t" + buffer[f] + "\t\t\t"
			+"99"
			+ " (LEXERR)\t" +  "1 (UNRECOGSYM)" +"\n");
	listingList.set(lineCounter-1, listingList.get(lineCounter-1)
			+ "LEXERR:\tUnrecognized Symbol:\t"
```

```
                                        + buffer[f] + "\n");
                        String x = buffer[f] + "";
                        tokenList.add(new Token(99, 1, x, lineCounter));
//                      listingWriter.printf("LEXERR:\tUnrecognized Symbol:\t"
//                                      + buffer[f] + "\n");
                }
//                 TODO WHAT IS THIS?
//                IF(CHARACTER.ISDIGIT(BUFFER[F])) {
//                      TOKENWRITER.PRINTF(LINECOUNTER + "\T\T\T"
//                                      + BUFFER[F]
//                                                      + "\T\T\T26 (INT)\T"
//                                                      + "0 (NULL)\N");
//                      TOKENLIST.ADD(NEW TOKEN(26, 0));
//
//                }
                if(buffer[f] == '%') {
                        tokenWriter.print(lineCounter + "\t\t\t" + "%" + "\t\t\t"
                                        +"99"
                                        + " (LEXERR)\t" +  "1 (UNRECOGSYM)" +"\n");
                        tokenList.add(new Token(99, 1, lineCounter));
                        listingList.set(lineCounter-1, listingList.get(lineCounter-1)
                                        + "LEXERR:\tUnrecognized Symbol:\t"
                                        + "%" + "\n");
//                      listingWriter.print("LEXERR:\tUnrecognized Symbol:\t"
//                                      + "%" + "\n");

                }
                // Move on in the buffer after unrecognized symbol
                f++;
                b = f;
        }
        // Out of machine codes, so set buffer indices to zero
        f = 0;
        b = 0;
        // Clear the buffer once all tokens have been gotten from it
        clearBuffer();
        return null;
    }

    /**
     * This method will write the buffer's contents to the listing file
     * with line number prefixed
     *
     */
    private static void writeBufferToListing() {
        // TODO delete line
        listingWriter.printf(lineCounter + "\t\t");
//      listingList.add(lineCounter + "\t\t");
        for(int i = 0; i < buffer.length; i++) {
                // Check to see we haven't gone past our string
                if(buffer[i] != '\u0000') {
                        listingWriter.print(buffer[i]);
                }
        }
        listingWriter.println();
    }

    /**
     * This method is called once the source line has been tokenized
     */
    private static void clearBuffer() {
        for(int i = 0; i < buffer.length; i++) {
                buffer[i] = '\u0000';
        }
```

```
}

/**
 * This method will take a line from the source file and put it into the
 * character buffer
 * @return
 */
private static String getNextLine() {
	String line = sourceScanner.nextLine();
	listingList.add(lineCounter + "\t\t" + line + "\n");
	if (line.length() > 72) {
		listingWriter.printf("Source line too long: max 72 chars\n");
	}

	for(int i = 0; i < line.length(); i++) {
		buffer[i] = line.charAt(i);
	}
	return null;
}

private static void terminate() {
	listingWriter.close();
	tokenWriter.close();
	reservedScanner.close();
	sourceScanner.close();
	System.out.println("Files closed");
}

/*
 *
 *
 * TODO lexeme list for token error reporting?
 *
 *
 *
 */

// LEXICAL ANALYZER

// Main driver
private static void lexicalAnalyzer() {
	parse();
}

private static void parse() {
	token = tokenList.pop();
	program();  // Start symbol
	match(98, 0); // End of file marker - 98
	// Close all open files and clean up
	terminate();
	// Write listing nodes to listing file

}

private static void match(int type, int attribute) {
	System.out.println("Match " + type + " " + attribute +
			" ,  given " + token.tokenType + " " + token.attribute
			+ "\t\t" + token.lexeme);
	// TODO attributes too!
	if((type == token.tokenType && attribute == token.attribute)
			&& token.tokenType == 98) {
		System.out.println("Success - reached end of file");
		for(int i = 0; i < listingList.size(); i++) {
			listingWriter.print(listingList.get(i));
```

```
                }
//                  System.exit(0);
            }
            // IDs
            else if(type == 25 && token.tokenType == 25
                        && token.tokenType != 98) {
                token = tokenList.pop();
            }
            else if((type == token.tokenType && attribute == token.attribute)
                        && token.tokenType != 98) {
                //&& attribute == token.attribute) {
                //                  System.out.println("Match of token and type");
                token = tokenList.pop();
//                  System.out.println(token.lexeme);
            }
            else if(type != token.tokenType
                        || attribute != token.attribute) {
                System.out.println("SYNTAX ERROR: Expected "
                                + getTokenReference(token.tokenType, token.attribute)
                                + ", received " +
                                token.lexeme + " " + token.tokenType + " "
                                + token.attribute);
                System.out.println(lineCounter);
                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SYNTAX ERROR: Expected "
                                + getTokenReference(type, attribute)
                                + ", received " +
                                token.lexeme + "\n");
                token = tokenList.pop();
            }
            else {
                token = tokenList.pop(); // Make sure you don't do this twice
            }
        }

        private static void program() {
            System.out.println("program(), " + token.tokenType);
            if(token.tokenType == 7) {   // program
                match(7, 0);          // program
                match(25, 0); // id
                match(4, 3);          // (
                identifier_list();
                match(4, 4);  // )
                match(4, 5);  // ;
                program_tail();
            }
            else {
                System.out.println("SYNTAX ERROR: Expecting one of 'program'"
                                + " given " + token.lexeme );
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'program'"
                                + " given " + token.lexeme + "\n");

                 while(token.tokenType != 98) {   // Error recovery
                                token = tokenList.pop();
                 }
            }
        }



        private static void program_tail() {
            // program' -> sub_decs cmpd_stmt .
            // program' -> cmpd_stmt .
```

```
	// program' -> declarations program''
	System.out.println("program_tail(), " + token.tokenType);
	if(token.tokenType == 17) {  // procedure
		subprogram_declarations();
		compound_statement();
		match(4, 8);  // .
	}
	else if(token.tokenType == 5) {     // begin
		compound_statement();
		match(4, 8);  // .
	}
	else if(token.tokenType == 8) {     // var
		declarations();
		program_tail_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'procedure',"
				+ "'begin', 'var'"
				+ " given " + token.lexeme );
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'procedure',"
				+ "'begin', 'var'"
				+ " given " + token.lexeme + "\n");
		 while(token.tokenType != 98) {   // Error recovery
				token = tokenList.pop();
		 }
	}
}



private static void program_tail_tail() {
	// program'' -> subprog_decs cmpd_stmt .
	// program'' -> cmpd_stmt .
	System.out.println("program_tail_tail(), " + token.tokenType);
	if(token.tokenType == 17) {  // procedure
		subprogram_declarations();
		compound_statement();
		match(4, 8);  // .
	}
	else if(token.tokenType == 5) {     //begin
		compound_statement();
		match(4, 8);  // .
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
				+ ", 'begin'"
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'procedure'"
				+ ", 'begin'"
				+ " given " + token.lexeme + "\n");
		while(token.tokenType != 98) {
			token = tokenList.pop();
		}
	}


}



private static void declarations() {
	// declarations -> var id : type ; declarations'
```

```
        System.out.println("declarations(), " + token.tokenType);
        if(token.tokenType == 8) {   // var
                match(8, 0);  // var
                match(25, 0); // id
                match(4, 6);  // :
                type();
                match(4, 5);  // ;
                declarations_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'var'"
                                + " given " + token.lexeme);
                listingList.set(token.line-1, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'var'"
                                + " given " + token.lexeme + "\n");
                // procedure, begin, $: follows - 17, 5, 98
         while(token.tokenType != 17
                        && token.tokenType != 5
                        && token.tokenType != 98) {   // Error recovery
                    token = tokenList.pop();
         }
        }
}



private static void declarations_tail() {
        // declarations_tail -> var id : type ; declarations' | e
        System.out.println("declarations_tail(), " + token.tokenType);
        if(token.tokenType == 8) {   // var
                match(8, 0);  // var
                match(25, 0); // id
                match(4, 6);  // :
                type();
                match(4, 5);  // ;
                declarations_tail();
        }
        else if(token.tokenType == 17 ||
                        token.tokenType == 5) { // procedure, begin
                // NoOp, epsilon
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'var'"
                                + "'procedure', 'begin'"
                                + " given " + token.lexeme);
                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SYNTAX ERROR: Expecting one of 'var'"
                                + "'procedure', 'begin'"
                                + " given " + token.lexeme + "\n");
                // procedure, begin, $: follows - 17, 5, 98
         while(token.tokenType != 17
                        && token.tokenType != 5
                        && token.tokenType != 98) {   // Error recovery
                    token = tokenList.pop();
         }
        }

}

private static void type() {
        // type -> standard_type
        // type -> standard_type
        // type -> array [ num .. num ] of standard_type
        System.out.println("type(), " + token.tokenType);
```

```
        if(token.tokenType == 13) {  // integer
               standard_type();
        }
        else if(token.tokenType == 16) { // real
               standard_type();
        }
        else if(token.tokenType == 14) {    // array
               match(14, 0);  // array
               match(4, 1);   // [
               match(13, 0);  // num // TODO Assumption: integer
               match(4, 9);   // ..
               match(13, 0);  // num (integer)
               match(4, 2);   // ]
               match(15, 0);  // of
               standard_type();
        }
        else {
               System.out.println("SYNTAX ERROR: Expecting one of 'integer'"
                             + ", 'real', 'array'"
                             + ",  given: " + token.tokenType + " "
                             + token.attribute);
               listingList.set(token.line-1, listingList.get(token.line-1)
                             + "SYNTAX ERROR: Expecting one of 'integer'"
                             + ", 'real', 'array'"
                             + ",  given: " + token.lexeme
                             + "\n");
               // follows: ;, ), $
         while(token.tokenType != 98   // Error recovery
                      && (token.tokenType != 4 && token.attribute != 5)
                      && (token.tokenType != 4 && token.attribute != 4)) {
             token = tokenList.pop();
         }
        }

}



private static void standard_type() {
        // standard_type -> integer
        // standard_type -> real
        System.out.println("standard_type(), " + token.tokenType);
        if(token.tokenType == 13) {  // integer
               match(13, 0);
        }
        else if(token.tokenType == 16) { // real
               match(16, 0);
        }
        else {
               System.out.println("SYNTAX ERROR: Expecting one of 'integer'"
                             + ", real "
                             + " given " + token.lexeme);
               listingList.set(token.line-1, listingList.get(lineCounter-1)
                             + "SYNTAX ERROR: Expecting one of 'integer'"
                             + ", real "
                             + " given " + token.lexeme + "\n");
               // follows: ;, ), $
         while(token.tokenType != 98   // Error recovery
                      && (token.tokenType != 4 && token.attribute != 5)
                      && (token.tokenType != 4 && token.attribute != 4)) {
             token = tokenList.pop();
         }
        }
}
```

```
private static void compound_statement() {
	// cmpd_stmt -> begin cmpd_stmt'
	System.out.println("compdound_statement(), " + token.tokenType);
	if(token.tokenType == 5) {   // begin
		match(5, 0);  // begin
		compound_statement_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'begin'"
				+ " given " + token.lexeme);
		listingList.set(token.line-1, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'begin'"
				+ " given " + token.lexeme + "\n");
		// . ; end else $
	 while(token.tokenType != 98   // Error recovery
			&& (token.tokenType != 4 && token.attribute != 8)
			&& (token.tokenType != 4 && token.attribute != 5)
			&& token.tokenType != 12
			&& token.tokenType != 6) {
		token = tokenList.pop();
	 }
	}
}

private static void compound_statement_tail() {
	// cmpd_stmt' -> opt_stmts end
	// cmpd_stmt' -> opt_stmts end
	// cmpd_stmt' -> opt_stmts end
	// cmpd_stmt' -> opt_stmts end
	// cmpd_stmt' -> opt_stmts end
	// cmpd_stmt' -> end         // TODO is this right?

	System.out.println("compdound_statement_tail(), " + token.tokenType);
	if(token.tokenType == 25) {  // id
		optional_statements();
		match(6, 0);  // end
	}
	else if(token.tokenType == 24) {    // call
		optional_statements();
		match(6, 0);  // end
	}
	else if(token.tokenType == 5) {     // begin
		optional_statements();
		match(6, 0);  // end
	}
	else if(token.tokenType == 18) {    // while
		optional_statements();
		match(6, 0);  // end
	}
	else if(token.tokenType == 10) {    // if
		optional_statements();
		match(6, 0);  // end
	}
	else if(token.tokenType == 6) {     // end
		match(6, 0);  // end
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'begin'"
				+ "'call', 'id', 'while', 'if', 'end'"
				+ " given " + token.lexeme);
		listingList.set(token.line-1, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'begin'"
				+ "'call', 'id', 'while', 'if', 'end'"
```

```
                        + " given " + token.lexeme + "\n");
            // . ; end else $
         while(token.tokenType != 98   // Error recovery
                     && (token.tokenType != 4 && token.attribute != 8)
                     && (token.tokenType != 4 && token.attribute != 5)
                     && token.tokenType != 12
                     && token.tokenType != 6) {
             token = tokenList.pop();
         }
        }
}

private static void optional_statements() {
        // optional_statements -> statement_list
        // optional_statements -> statement_list
        // optional_statements -> statement_list
        // optional_statements -> statement_list
        // optional_statements -> statement_list

        System.out.println("optional_statements(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
                statement_list();
        }
        if(token.tokenType == 24) {  // call
                statement_list();
        }
        if(token.tokenType == 5) {   // begin
                statement_list();
        }
        if(token.tokenType == 18) {  // while
                statement_list();
        }
        if(token.tokenType == 10) {  // if
                statement_list();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'id'"
                        + "'call', 'begin', 'while', 'if'"
                        + " given " + token.lexeme);
                listingList.set(token.line-1, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of 'id'"
                        + "'call', 'begin', 'while', 'if'"
                        + " given " + token.lexeme + "\n");
                // end $
         while(token.tokenType != 98
                        && token.tokenType != 6) {
                token = tokenList.pop();
         }
        }
}

private static void statement_list() {
        // Stmt_list -> stmt stmt_list'
        System.out.println("statement_list(), " + token.tokenType);
        if(token.tokenType == 5) {   // begin
                statement();
                statement_list_tail();
        }
        else if(token.tokenType == 25) {    // id
                statement();
                statement_list_tail();
        }
        else if(token.tokenType == 24) {    // call
                statement();
```

```
                statement_list_tail();
        }
        else if(token.tokenType == 18) {    // while
                statement();
                statement_list_tail();
        }
        else if(token.tokenType == 10) {    // if
                statement();
                statement_list_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'id'"
                                + "'call', 'begin', 'while', 'if'"
                                + " given " + token.lexeme );
                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SYNTAX ERROR: Expecting one of 'id'"
                                + "'call', 'begin', 'while', 'if'"
                                + " given " + token.lexeme + "\n");
                // end, $
                while(token.tokenType != 6
                                && token.tokenType != 98) {
                        token = tokenList.pop();
                }
        }
}

private static void statement_list_tail() {
        // Stmt_list' -> ; stmt stmt_list' - ;
        // Stmt_list' -> e - end

        System.out.println("statement_list_tail(), " + token.tokenType);
        if(token.tokenType == 4 && token.attribute == 5) {// ;
                match(4, 5);  // ;
                statement();
                statement_list_tail();
        }
        // TODO I added this, for the . error
        else if(token.tokenType == 4 && token.attribute == 8) {
                match(4, 8);  // .
                statement();
                statement_list_tail();
        }
        else if(token.tokenType == 6) {     // end
                // NoOp, epsilon
                return;
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of ';', 'end'"
                                + " given " + token.lexeme );
                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SYNTAX ERROR: Expecting one of ';', 'end'"
                                + " given " + token.lexeme + "\n");
                // end, $
                while(token.tokenType != 6
                                && token.tokenType != 98) {
                        token = tokenList.pop();
                }
        }
}

private static void statement() {
        // statement -> compound_statement - begin
        // Stmt -> variable assignop expression - id
        // Stmt -> procedure_statement - call
```

```
	// Stmt -> while exp do stmt - while
	// Stmt -> if expression then stmt stmt' - if

	System.out.println("statement(), " + token.tokenType);
	if(token.tokenType == 5) {   // begin
		compound_statement();
	}
	else if(token.tokenType == 25) {    // id
		variable();
		match(21, 0); // :=
		expression();
	}
	else if(token.tokenType == 24) {    // call
		procedure_statement();
	}
	else if(token.tokenType == 18) {    // while
		match(18, 0); // while
		expression();
		match(19, 0); // do
		statement();
	}
	else if(token.tokenType == 10) {    // if
		match(10, 0); // if
		expression();
		match(11, 0); // then
		statement();
		statement_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'id'"
				+ "'call', 'begin', 'while', 'if'"
				+ " given " + token.lexeme );
		listingList.set(token.line, listingList.get(token.line)
				+ "SYNTAX ERROR: Expecting one of 'id'"
				+ "'call', 'begin', 'while', 'if'"
				+ " given " + token.lexeme + "\n");
		// else, end, ;, $
		while(token.tokenType != 98
				&& token.tokenType != 6
				&& token.tokenType != 12
				&& (token.tokenType != 4 && token.attribute != 5)) {
			token = tokenList.pop();
		}
	}
}

private static void statement_tail() {
	// stmt' -> else statement - else
	// stmt' -> e - else, ;, end

	System.out.println("statement_tail(), " + token.tokenType);
	// TODO parse table anomaly
	if(token.tokenType == 12) {  // else
		match(12, 0); // else
		statement();
	}
	else if(token.tokenType == 25
			|| token.tokenType == 6
			|| (token.tokenType == 4 && token.attribute == 5)) {
		// else, ;, end
		// NoOp, epsilon
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'else'"
```

```
                        + "';', 'end'"
                        + " given " + token.lexeme );
                listingList.set(token.line, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of 'else'"
                        + "';', 'end'"
                        + " given " + token.lexeme + "\n");
                // else, end, ;, $
                while(token.tokenType != 98
                        && token.tokenType != 6
                        && token.tokenType != 12
                        && (token.tokenType != 4 && token.attribute != 5)) {
                        token = tokenList.pop();
                }
        }
}



private static void procedure_statement() {
        // procedure_statement -> call id procedure_statement'
        System.out.println("procedure_statement(), " + token.tokenType);
        if(token.tokenType == 24) {  // call
                match(24, 0);  // call
                match(25, 0);  // id
                procedure_statement_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'call'"
                        + " given " + token.lexeme );
                listingList.set(token.line, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of 'call'"
                        + " given " + token.lexeme + "\n");
                // else, end, ;, $
                while(token.tokenType != 98
                        && token.tokenType != 6
                        && token.tokenType != 12
                        && (token.tokenType != 4 && token.attribute != 5)) {
                        token = tokenList.pop();
                }
        }
}

private static void procedure_statement_tail() {
        // procedure_statement_tail -> ( expression_list ) - (
        // procedure_statement_tail -> e - else, ;, end
        System.out.println("procedure_statement_tail(), " + token.tokenType);
        if(token.tokenType == 4 && token.attribute ==3) { // (
                match(4, 3);  // (
                expression_list();
                match(4, 4);
        }
        else if(token.tokenType == 12) {    // else
                // NoOp, epsilon
        }
        else if(token.tokenType == 4 && token.attribute == 5) {  // ;
                // NoOp, epsilon
        }
        else if(token.tokenType == 6) {     // end
                // NoOp, epsilon
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of '(', 'else'"
                        + ", ';', 'end'"
                        + " given " + token.lexeme );
```

```
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of '(', 'else'"
                                + ", ';', 'end'"
                                + " given " + token.lexeme + "\n");
                // else, end, ;, $
                while(token.tokenType != 98
                                && token.tokenType != 6
                                && token.tokenType != 12
                                && (token.tokenType != 4 && token.attribute != 5)) {
                        token = tokenList.pop();
                }
        }
}

private static void expression_list() {
        // expression_list -> expression expression_list'
        System.out.println("expression_list(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 13 || token.tokenType == 16
                        || token.tokenType == 23) {  // integer, real, longreal
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 20) {    // not
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 1) {   // +
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 2) {   // -
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 4 && token.attribute == 3) {   // (
                expression();
                expression_list_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'id', 'num'"
                                + "'not', '+', '-', '('"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'id', 'num'"
                                + "'not', '+', '-', '('"
                                + " given " + token.lexeme + "\n");
                // ), $
                while(token.tokenType != 98
                                && (token.tokenType != 4 && token.attribute != 4)) {
                        token = tokenList.pop();
                }
        }
}



private static void expression_list_tail() {
        // expression_list_tail -> , expression expression_list' - ,
        // expression_list_tail -> e - )
```

```
        System.out.println("expression_list_tail(), " + token.tokenType);
        if(token.tokenType == 4 && token.attribute == 7) {       // ,
                match(4, 7);  // ,
                expression();
                expression_list_tail();
        }
        else if(token.tokenType == 4 && token.attribute == 4) {  // )
                // NoOp, epsilon
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of ',', ')'"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of ',', ')'"
                                + " given " + token.lexeme + "\n");
                // ), $
                while(token.tokenType != 98
                                && (token.tokenType != 4 && token.attribute != 4)) {
                        token = tokenList.pop();
                }
        }
}

private static void variable() {
        // variable -> id variable'
        System.out.println("variable(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
                match(25, 0);
                variable_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'id'"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'id'"
                                + " given " + token.lexeme + "\n");
                // assignop, $
                while(token.tokenType != 98
                                && token.tokenType != 21) {
                        token = tokenList.pop();
                }
        }
}

private static void variable_tail() {
        // variable' -> e - assignop
        // variable' -> [ expression ] - [
        System.out.println("variable_tail(), " + token.tokenType);
        if(token.tokenType == 21) {  // :=
                // NoOp, epsilon
        }
        else if(token.tokenType == 4 && token.attribute == 1) {  // [
                match(4, 1);  // [
                expression();
                match(4, 2);  // ]
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of ':=', '['"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of ':=', '['"
                                + " given " + token.lexeme + "\n");
                // assignop, $
                while(token.tokenType != 98
```

```
                        && token.tokenType != 21) {
                token = tokenList.pop();
            }
        }
}

private static void expression() {
        // expression -> simple_expression expression'
        System.out.println("expression(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
            simple_expression();
            expression_tail();
        }
        else if(token.tokenType == 26 || token.tokenType == 27
                || token.tokenType == 28) {  // int, real, longreal
            simple_expression();
            expression_tail();
        }
        else if(token.tokenType == 20) {    // not
            simple_expression();
            expression_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 1) {   // +
            simple_expression();
            expression_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 2) {   // -
            simple_expression();
            expression_tail();
        }
        else if(token.tokenType == 4 && token.attribute == 3) {   // (
            simple_expression();
            expression_tail();
        }
        else {
            System.out.println("SYNTAX ERROR: Expecting one of 'id', 'num'"
                        + "'not', '+', '-', '('"
                        + " given " + token.lexeme + " " + token.tokenType);
            listingList.set(token.line, listingList.get(token.line)
                        + "SYNTAX ERROR: Expecting one of 'id', 'num'"
                        + "'not', '+', '-', '('"
                        + " given " + token.lexeme + "\n");
            // do, then, else, end, $
            // ;, ,, ), ]
            while(token.tokenType != 98
                        && token.tokenType != 6
                        && token.tokenType != 12
                        && token.tokenType != 11
                        && token.tokenType != 19
                        && (token.tokenType != 4 && token.tokenType != 5)
                        && (token.tokenType != 4 && token.tokenType != 4)
                        && (token.tokenType != 4 && token.tokenType != 2)
                        && (token.tokenType != 4 && token.tokenType != 7)) {
                token = tokenList.pop();
            }
        }
}

private static void expression_tail() {
        // expression' -> relop simple_expression - relop
        // expression' -> e - ), ], do, then, ,, else, ;, end
        System.out.println("expression_tail(), " + token.tokenType);
        if(token.tokenType == 1 && token.attribute == 1) {       // relop
            match(1, 1);
```

```
        simple_expression();
    }
    else if(token.tokenType == 1 && token.attribute == 2) { // relop
        match(1, 2);
        simple_expression();
    }
    else if(token.tokenType == 1 && token.attribute == 3) { // relop
        match(1, 3);
        simple_expression();
    }
    else if(token.tokenType == 1 && token.attribute == 4) { // relop
        match(1, 4);
        simple_expression();
    }
    else if(token.tokenType == 1 && token.attribute == 5) { // relop
        match(1, 5);
        simple_expression();
    }
    else if(token.tokenType == 1 && token.attribute == 6) { // relop
        match(1, 6);
        simple_expression();
    }
    else if(token.tokenType == 4 && token.attribute == 4) {  // )
        // NoOp, epsilon
    }
    else if(token.tokenType == 4 && token.attribute == 2) {  // ]
        // NoOp, epsilon
    }
    else if(token.tokenType == 19) {    // do
        // NoOp, epsilon
    }
    else if(token.tokenType == 11) {    // then
        // NoOp, epsilon
    }
    else if(token.tokenType == 4 && token.attribute == 7) {  // ,
        // NoOp, epsilon
    }
    else if(token.tokenType == 12) {    // else
        // NoOp, epsilon
    }
    else if(token.tokenType == 4 && token.attribute == 5) {  // ;
        // NoOp, epsilon
    }
    else if(token.tokenType == 6) {     // end
        // NoOp, epsilon
    }
    else {
        System.out.println("SYNTAX ERROR: Expecting one of "
                        + "'relop', '), ']', 'do', 'then', ',', 'else', ';', "
                        + "'end' "
                        + " given " + token.lexeme);
        listingList.set(token.line, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of "
                        + "'relop', '), ']', 'do', 'then', ',', 'else', ';', "
                        + "'end' "
                        + " given " + token.lexeme + "\n");
        // do, then, else, end, $
        // ;, ,, ), ]
        while(token.tokenType != 98
                        && token.tokenType != 6
                        && token.tokenType != 12
                        && token.tokenType != 11
                        && token.tokenType != 19
                        && (token.tokenType != 4 && token.tokenType != 5)
```

```
                        && (token.tokenType != 4 && token.tokenType != 4)
                        && (token.tokenType != 4 && token.tokenType != 2)
                        && (token.tokenType != 4 && token.tokenType != 7)) {
                token = tokenList.pop();
            }
        }
}

private static void simple_expression() {
        // simple_expression -> term simple_expression' - id ( num not
        // simple_expression -> sign term simple_expression' - + -
        System.out.println("simple_expression(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
            term();
            simple_expression_tail();
        }
        // These are NUMs
        else if(token.tokenType == 26 || token.tokenType == 27
                || token.tokenType == 28) {  // integer, real, longreal
            term();
            simple_expression_tail();
        }
        else if(token.tokenType == 20) {    // not
            term();
            simple_expression_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 1) {  // +
            sign();
            term();
            simple_expression_tail();
        }
        else if(token.tokenType == 2 && token.attribute == 2) {  // -
            sign();
            term();
            simple_expression_tail();
        }
        else if(token.tokenType == 4 && token.attribute == 3) {  // (
            term();
            simple_expression_tail();
        }
        else {
            System.out.println("SYNTAX ERROR: Expecting one of 'id', 'num'"
                        + "'not', '+', '-', '('"
                        + " given " + token.lexeme);
            listingList.set(token.line-1, listingList.get(token.line-1)
                        + "SYNTAX ERROR: Expecting one of 'id', 'num'"
                        + "'not', '+', '-', '('"
                        + " given " + token.lexeme + "\n");
            // do, then, else, end, $
            // ;, ,, ), ]
            // relop
            while(token.tokenType != 98
                        && token.tokenType != 6
                        && token.tokenType != 12
                        && token.tokenType != 11
                        && token.tokenType != 19
                        && (token.tokenType != 4 && token.tokenType != 5)
                        && (token.tokenType != 4 && token.tokenType != 4)
                        && (token.tokenType != 4 && token.tokenType != 2)
                        && (token.tokenType != 4 && token.tokenType != 7)
                        && token.tokenType != 1) {
                token = tokenList.pop();
            }
        }
```

```
}

private static void simple_expression_tail() {
	// simple_expression' -> addop term simple_expression - addop
	// simple_expression' -> e - relop, ), ], do, then, else, ;, end
	System.out.println("simple_expression_tail(), " + token.tokenType);
	if(token.tokenType == 2) {   // addop
		if(token.attribute == 1) {
			match(2, 1);  // 3 1
		}
		if(token.attribute == 2) {
			match(2, 2);
		}
		if(token.attribute == 3) {
			match(2, 3);
		}
	}
	else if(token.tokenType == 1
			// relop, end, ), ], do, then, ,, else, ;

			|| token.tokenType == 6
			|| (token.tokenType == 4 && token.attribute == 2)
			|| (token.tokenType == 4 && token.attribute == 4)
			|| token.tokenType == 19
			|| token.tokenType == 11
			|| (token.tokenType == 4 && token.attribute == 7)
			|| token.tokenType == 12
			|| (token.tokenType == 4 && token.attribute == 5)) {
		// NoOp, epsilon
	}

	else {
		System.out.println("SYNTAX ERROR: Expecting one of "
				+ "addop, relop, ), ], do, then, ,, else, ;, end"
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of "
				+ "addop, relop, ), ], do, then, ,, else, ;, end"
				+ " given " + token.lexeme + "\n");
		// do, then, else, end, $
		// ;, ,, ), ]
		// relop
		while(token.tokenType != 98
				&& token.tokenType != 6
				&& token.tokenType != 12
				&& token.tokenType != 11
				&& token.tokenType != 19
				&& (token.tokenType != 4 && token.tokenType != 5)
				&& (token.tokenType != 4 && token.tokenType != 4)
				&& (token.tokenType != 4 && token.tokenType != 2)
				&& (token.tokenType != 4 && token.tokenType != 7)
				&& token.tokenType != 1) {
			token = tokenList.pop();
		}
	}
}

private static void sign() {
	// sign -> + | -
	System.out.println("sign(), " + token.tokenType);
	if(token.tokenType == 24) {  // call
		match(24, 0); // call
		match(25, 0); // id
		procedure_statement_tail();
```

```
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of '+', '-'"
				+ ", given " + token.lexeme );
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of '+', '-'"
				+ ", given " + token.lexeme + "\n");
		// id, num, (, not, $
		while(token.tokenType != 98
				&& token.tokenType != 20
				&& (token.tokenType != 4 && token.attribute != 3)
				&& token.tokenType != 26
				&& token.tokenType != 27
				&& token.tokenType != 28
				&& token.tokenType != 25){
			token = tokenList.pop();
		}
	}
}

private static void term() {
	// term -> factor term' - num ( not id
	System.out.println("term(), " + token.tokenType);
	if(token.tokenType == 4 && token.attribute == 3) {	// (
		factor();
		term_tail();
	}
	else if(token.tokenType == 26 || token.tokenType == 27
			|| token.tokenType == 28) {  // integer, real, longreal
		factor();
		term_tail();
	}
	else if(token.tokenType == 20) {    // not
		factor();
		term_tail();
	}
	else if(token.tokenType == 25) {    // id
		factor();
		term_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of ')', 'num'"
				+ ", 'not', 'id' "
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of ')', 'num'"
				+ ", 'not', 'id' "
				+ " given " + token.lexeme + "\n");
		// do, then, else, end, $
		// ;, ,, ), ]
		// relop
		// addop
		while(token.tokenType != 98
				&& token.tokenType != 6
				&& token.tokenType != 12
				&& token.tokenType != 11
				&& token.tokenType != 19
				&& (token.tokenType != 4 && token.tokenType != 5)
				&& (token.tokenType != 4 && token.tokenType != 4)
				&& (token.tokenType != 4 && token.tokenType != 2)
				&& (token.tokenType != 4 && token.tokenType != 7)
				&& token.tokenType != 1
				&& token.tokenType != 2) {
			token = tokenList.pop();
```

```
		}
	}
}

private static void term_tail() {
	// term' -> mulop factor term' - mulop
	// term' -> e - addop, relop, ), ], do, then, ,, else, ;, end
	System.out.println("term_tail(), " + token.tokenType);
	if(token.tokenType == 3) {   // mulop
		if(token.attribute == 1) {
			match(3, 1);  // 3 1
		}
		if(token.attribute == 2) {
			match(3, 2);
		}
		if(token.attribute == 3) {
			match(3, 3);
		}
		if(token.attribute == 4) {
			match(3, 4);
		}
		if(token.attribute == 5) {
			match(3, 5);
		}

	}
	else if(token.tokenType == 1
			|| token.tokenType == 2             // addop, relop, end,
			// ), ], do, then, ,, else, ;,
			|| token.tokenType == 6
			|| (token.tokenType == 4 && token.attribute == 2)
			|| (token.tokenType == 4 && token.attribute == 4)
			|| token.tokenType == 19
			|| token.tokenType == 11
			|| (token.tokenType == 4 && token.attribute == 7)
			|| token.tokenType == 12
			|| (token.tokenType == 4 && token.attribute == 5)) {
		// NoOp, epsilon
	}

	else {
		System.out.println("SYNTAX ERROR: Expecting one of "
				+ "addop, relop, mulop, ), ], do, then, ,, else, ;, end"
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of "
				+ "addop, relop, mulop, ), ], do, then, ,, else, ;, end"
				+ " given " + token.lexeme + "\n");
		// do, then, else, end, $
		// ;, ,, ), ]
		// relop
		// addop
		while(token.tokenType != 98
				&& token.tokenType != 6
				&& token.tokenType != 12
				&& token.tokenType != 11
				&& token.tokenType != 19
				&& (token.tokenType != 4 && token.tokenType != 5)
				&& (token.tokenType != 4 && token.tokenType != 4)
				&& (token.tokenType != 4 && token.tokenType != 2)
				&& (token.tokenType != 4 && token.tokenType != 7)
				&& token.tokenType != 1
				&& token.tokenType != 2) {
			token = tokenList.pop();
```

```
                }
        }
}

private static void factor() {
        // factor -> num | ( exp ) | not factor | id factor'
        System.out.println("factor(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
                match(25, 0); // id
                factor_tail();
        }
        else if(token.tokenType == 26) {    // int
                match(26, 0); // int
        }
        else if(token.tokenType == 28) {    // longreal
                match(28, 0); // longreal
        }
        else if(token.tokenType == 16) {    // real
                match(27, 0); // real
        }

        else if(token.tokenType == 20) {    // not
                match(20, 0); // not
                factor();
        }
        else if(token.tokenType == 4 && token.attribute == 3) {  // (
                match(4, 3);  // (
                expression();
                match(4, 4);  // )
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'id', 'num'"
                                + "'not', '('"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'id', 'num'"
                                + "'not', '('"
                                + " given " + token.lexeme + "\n");
                // do, then, else, end, $, ;, ,, ), ], relop, addop, mulop
                while(token.tokenType != 98
                                && token.tokenType != 6
                                && token.tokenType != 12
                                && token.tokenType != 11
                                && token.tokenType != 19
                                && (token.tokenType != 4 && token.tokenType != 5)
                                && (token.tokenType != 4 && token.tokenType != 4)
                                && (token.tokenType != 4 && token.tokenType != 2)
                                && (token.tokenType != 4 && token.tokenType != 7)
                                && token.tokenType != 1
                                && token.tokenType != 2
                                && token.tokenType != 3) {
                        token = tokenList.pop();
                }
        }
}

private static void factor_tail() {
        // factor' -> [ exp ] - [
        // factor' -> e - mulop, addop, relop, ), ], do, then, ,, else, ;,
        // end
        System.out.println("factor_tail(), " + token.tokenType + " "
                        + token.attribute);
        if(token.tokenType == 4 && token.attribute == 1) {       // [
                match(4, 1);  // [
```

```
                expression();
                match(4, 2);  // ]
        }
        else if(token.tokenType == 1
                        || token.tokenType == 2             // mulop, addop, relop, end,
                        || token.tokenType == 3             // ), ], do, then, ,, else, ;,
                        || token.tokenType == 6
                        || (token.tokenType == 4 && token.attribute == 2)
                        || (token.tokenType == 4 && token.attribute == 4)
                        || token.tokenType == 19
                        || token.tokenType == 11
                        || (token.tokenType == 4 && token.attribute == 7)
                        || token.tokenType == 12
                        || (token.tokenType == 4 && token.attribute == 5)) {
                // NoOp, epsilon
                return;
        }

        else {
                System.out.println("SYNTAX ERROR: Expecting one of "
                                + "addop, relop, mulop, ), ], do, then, ,, else, ;, end"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of "
                                + "addop, relop, mulop, ), ], do, then, ,, else, ;, end"
                                + " given " + token.lexeme + "\n");
                // do, then, else, end, $, ;, ,, ), ], relop, addop, mulop
                while(token.tokenType != 98
                                && token.tokenType != 6
                                && token.tokenType != 12
                                && token.tokenType != 11
                                && token.tokenType != 19
                                && (token.tokenType != 4 && token.tokenType != 5)
                                && (token.tokenType != 4 && token.tokenType != 4)
                                && (token.tokenType != 4 && token.tokenType != 2)
                                && (token.tokenType != 4 && token.tokenType != 7)
                                && token.tokenType != 1
                                && token.tokenType != 2
                                && token.tokenType != 3) {
                        token = tokenList.pop();
                }
        }
}



private static void subprogram_declarations() {
        // subprgm_decs -> subprgm_dec ; subprgm_decs'
        System.out.println("subprogram_declarations(), " + token.tokenType);
        if(token.tokenType == 17) {  // procedure
                subprogram_declaration();
                match(4, 5);  // ;
                subprogram_declarations_tail();
        }
        else {
                System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
                                + " given " + token.lexeme);
                listingList.set(token.line, listingList.get(lineCounter-1)
                                + "SYNTAX ERROR: Expecting one of 'procedure'"
                                + " given " + token.lexeme + "\n");
                // begin, $
                while(token.tokenType != 98
                                && token.tokenType != 5) {
                        token = tokenList.pop();
```

```
                }
        }
    }

    private static void subprogram_declarations_tail() {
        // subprogram_declarations' -> subprgm_dec ; subprgm_decs' - proc.
        // subprgm_decs' -> e - begin
        if(token.tokenType == 98) {
            match(98, 0);
        }
        System.out.println("subprogram_declarations_tail(), "
                + token.tokenType);
        if(token.tokenType == 17) {  // procedure
            subprogram_declaration();
            match(4, 5);  // ;
            subprogram_declarations_tail();
        }
        else if(token.tokenType == 5) {     // begin
            // NoOp, epsilon
        }
        else {
//          // TODO i added this
//          if(token.tokenType == 98) {
//              match(98, 0);
//          }
            System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
                    + "'begin', "
                    + " given " + token.lexeme + token.tokenType);
            listingList.set(token.line, listingList.get(lineCounter-1)
                    + "SYNTAX ERROR: Expecting one of 'procedure'"
                    + "'begin', "
                    + " given " + token.lexeme + "\n");
            // begin, $
            while(token.tokenType != 98
                    && token.tokenType != 5) {
                token = tokenList.pop();
            }
        }
    }

    private static void subprogram_declaration() {
        // subprogram_declaration -> subprogram_head -> subprogram_dec'
        System.out.println("subprogram_declaration(), " + token.tokenType);
        if(token.tokenType == 17) {  // procedure
            subprogram_head();
            subprogram_declaration_tail();
        }
        else {
            System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
                    + " given " + token.lexeme);
            listingList.set(token.line, listingList.get(lineCounter-1)
                    + "SYNTAX ERROR: Expecting one of 'procedure'"
                    + " given " + token.lexeme + "\n");
            // ;, $
            while(token.tokenType != 98
                    && (token.tokenType != 4 && token.attribute != 5)) {
                token = tokenList.pop();
            }
        }
    }

    private static void subprogram_declaration_tail() {
        // subprogram_declaration' -> cmpd_stmt - begin
        // subprgm_dec' -> subprgm_decs cmpd_stmt - procedure
```

```
	// subprgm_dec' -> decs subprgm_dec'' - var
	System.out.println("subprogram_declaration(), " + token.tokenType);
	// TODO
	if(token.tokenType == 98) {
		match(98, 0);
	}
	if(token.tokenType == 17) {  // procedure
		subprogram_declarations();
		compound_statement();
	}
	else if(token.tokenType == 5) {     // begin
		compound_statement();
	}
	else if(token.tokenType == 8) {     // var
		declarations();
		subprogram_declaration_tail_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
				+ "'begin', 'var'"
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'procedure'"
				+ "'begin', 'var'"
				+ " given " + token.lexeme + "\n");
		// ;, $
		while(token.tokenType != 98
				&& (token.tokenType != 4 && token.attribute != 5)) {
			token = tokenList.pop();
		}
	}
}

private static void subprogram_declaration_tail_tail() {
	// subprogram_declaration'' -> subprogram_decs cmpd_stmt
	// subprgm_dec'' -> compound_statement - begin
	System.out.println("subprogram_declaration_tail_tail(), "
			+ token.tokenType);
	if(token.tokenType == 17) {  // procedure
		subprogram_declarations();
		compound_statement();
	}
	else if(token.tokenType == 5) {     // begin
		compound_statement();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
				+ "'begin', "
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'procedure'"
				+ "'begin', "
				+ " given " + token.lexeme + "\n");
		// ;, $
		while(token.tokenType != 98
				&& (token.tokenType != 4 && token.attribute != 5)) {
			token = tokenList.pop();
		}
	}
}

private static void subprogram_head() {
	// subprogram_head -> procedure id subprogram_head'
	System.out.println("subprogram_head(), " + token.tokenType);
```

```
	if(token.tokenType == 17) {  // procedure
		match(17, 0);  // procedure
		match(25, 0);  // id
		subprogram_head_tail();
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of 'procedure'"
				+ " given " + token.lexeme + "\n");
		// begin, procedure, var, $
		while(token.tokenType != 98
				&& token.tokenType != 5
				&& token.tokenType != 8
				&& token.tokenType != 17) {
			token = tokenList.pop();
		}
	}
}

private static void subprogram_head_tail() {
	// subprogram_head' -> arguments ; - (
	// subprgm_head' -> ; - ;
	System.out.println("subprogram_head_tail(), " + token.tokenType);
	if(token.tokenType == 4 && token.attribute == 3) {	// (
		arguments();
		match(4, 5);  // ;
	}
	else if(token.tokenType == 5) {	// ;
		match(4, 5);  // ;
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of '(', ';' "
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of '(', ';' "
				+ " given " + token.lexeme + "\n");
		// begin, procedure, var, $
		while(token.tokenType != 98
				&& token.tokenType != 5
				&& token.tokenType != 8
				&& token.tokenType != 17) {
			token = tokenList.pop();
		}
	}
}

private static void arguments() {
	// arguments -> ( parameter_list )
	System.out.println("arguments(), " + token.tokenType);
	if(token.tokenType == 4 && token.attribute == 3) {	// (
		match(4, 3);  // (
		parameter_list();
		match(4, 4);  // )
	}
	else {
		System.out.println("SYNTAX ERROR: Expecting one of '(', "
				+ " given " + token.lexeme);
		listingList.set(token.line, listingList.get(lineCounter-1)
				+ "SYNTAX ERROR: Expecting one of '(', "
				+ " given " + token.lexeme + "\n");
		// ;, $
		while(token.tokenType != 98
```

```
                        && (token.tokenType != 4 && token.tokenType != 5)) {
                token = tokenList.pop();
            }
        }
}

private static void parameter_list() {
        // parameter_list() -> id : type parameter_list'
        System.out.println("arguments(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
            match(25, 0);  // id
            match(4, 6);   // :
            type();
            parameter_list_tail();
        }
        else {
            System.out.println("SYNTAX ERROR: Expecting one of '(', "
                        + " given " + token.lexeme);
            listingList.set(token.line, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of '(', "
                        + " given " + token.lexeme + "\n");
            // ), $
            while(token.tokenType != 98
                        && (token.tokenType != 4 && token.attribute != 4)) {
                token = tokenList.pop();
            }
        }
}

private static void parameter_list_tail() {
        // parameter_list() -> id : type parameter_list'
        System.out.println("arguments(), " + token.tokenType);
        if(token.tokenType == 4 && token.attribute == 5) {       // ;
            match(4, 5);   // ;
            match(25, 0);  // id
            match(4, 6);   // :
            type();
            parameter_list_tail();

        }
        else if(token.tokenType == 4 && token.attribute == 4) {   // )
            // NoOp, epsilon
        }
        else {
            System.out.println("SYNTAX ERROR: Expecting one of '(', "
                        + " given " + token.lexeme);
            listingList.set(token.line, listingList.get(lineCounter-1)
                        + "SYNTAX ERROR: Expecting one of '(', "
                        + " given " + token.lexeme + "\n");
            // ), $
            while(token.tokenType != 98
                        && (token.tokenType != 4 && token.attribute != 4)) {
                token = tokenList.pop();
            }
        }
}

private static void identifier_list() {
        System.out.println("identifier_list(), " + token.tokenType);
        if(token.tokenType == 25) {  // id
            match(25, 0);  // id
            // id_list -> id id_list'
            identifier_list_tail();
```

```
		}
		else {
			System.out.println("SYNTAX ERROR: Expecting one of 'id'"
					+ " given " + token.lexeme );
			listingList.set(token.line, listingList.get(lineCounter-1)
					+ "SYNTAX ERROR: Expecting one of 'id'"
					+ " given " + token.lexeme + "\n");
			// ), $
			while(token.tokenType != 98
					&& (token.tokenType != 4 && token.attribute != 4)) {
				token = tokenList.pop();
			}
		}
	}

	private static void identifier_list_tail() {
		// id_list' -> e | , id id_list'

		System.out.println("identifier_list_tail(), " + token.tokenType);
		if(token.tokenType == 4 && token.attribute == 4) {	// )
			// Epsilon - NoOp, do nothing
		}
		else if (token.tokenType == 4 && token.attribute == 7) { // ,
			match(4, 7);  // ,
			match(25, 0); // id
			identifier_list_tail();
		}
		else {
			System.out.println("SYNTAX ERROR: Expecting one of 'e', ','"
					+ " given " + token.lexeme );
			listingList.set(token.line, listingList.get(lineCounter-1)
					+ "SYNTAX ERROR: Expecting one of 'e', ','"
					+ " given " + token.lexeme + "\n");
			// ), $
			while(token.tokenType != 98
					&& (token.tokenType != 4 && token.attribute != 4)) {
				token = tokenList.pop();
			}
		}

	}


}
```