Project 3 Report

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

Introduction

For this project, I have created a semantic 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. Semantic errors are detected. Declarations processing and type checking is also performed. Memory addresses are computed as well.

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.

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 Semantic Analyzer. I am using Git for version control of my code.

Discussion and Conclusions

I decorated the grammar with green/bluenodes. I decorated with memory offsets as well. I implemented the semantic analyzer within the syntax analyzer. I did manual type checking processes on paper before programming them. There were many combinations of types to check against on relops, mulops, addops and assignops.

References

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

Appendix I: Sample Inputs and Outputs

Input: “cor34”

```
program test (input, output);
  var a : integer;
  var b : real;
  var c : array [1..2] of integer;

  procedure proc1(x:integer; y:real;
                  z:array [1..2] of integer; q: real);
    var d: integer;
    begin
      a:= 2;
      z[a] := 4;
      c[3] := 3
       end;

   procedure proc2(x: array [1..2] of real; y: integer);
     var e: real;

     procedure proc3(n: integer; z: real);
       var e: integer;

       procedure proc4(a: integer; z: array [1..3] of real);
         var x: integer;
         begin
           a:= e
         end;

       begin
         a:= e;
         e:= c[e]
       end;

     begin
       call proc1(x, e, c, b);
       call proc3(c[1], e);
       e := e + 4.44;
       a:= (a mod y) div x;
       while ((a >= 4) and ((b <= e)
                      or (not (a = c[a])))) do
         begin
           a:= c[a] + 1
         end
     end;

begin
  call proc2(c[4], c[5]);
  call proc2(c[4],2);
  if (a < 2) then a:= 1 else a := a + 2;
  if (b > 4.2) then a := c[a]
end.
```

Output: “cor34”: Token file

```
Line No.      Lexeme        TOKEN-TYPE    ATTRIBUTE
2                   program       7 (RES)             0
2                   test                  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                   a                   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                   b                   25 (ID)             4 (ptr to sym tab)
4                   :                   4 (CATCHALL)        6 (COLON)
4                   real          16 (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                   array         14 (RES)                  0
5                   [                   4 (CATCHALL)  1 (LEFTBRACK)
5                   1                   26 (INT)      0 (NULL)
5                   ..                  4 (CATCHALL)  9 (DOTDOT)
5                   2                   26 (INT)      0 (NULL)
5                   ]                   4 (CATCHALL)  2 (RIGHTBRACK)
5                   of            15 (RES)                  0
5                   integer       13 (RES)                  0
5                   ;                   4 (CATCHALL)  5 (SEMICOLON)
7                   procedure           17 (RES)                    0
7                   proc1               25 (ID)             6 (ptr to sym tab)
7                   (                   4 (CATCHALL)  3 (LEFTPAREN)
7                   x                   25 (ID)             7 (ptr to sym tab)
7                   :                   4 (CATCHALL)        6 (COLON)
7                   integer       13 (RES)                  0
7                   ;                   4 (CATCHALL)  5 (SEMICOLON)
7                   y                   25 (ID)             8 (ptr to sym tab)
7                   :                   4 (CATCHALL)        6 (COLON)
7                   real          16 (RES)                  0
7                   ;                   4 (CATCHALL)  5 (SEMICOLON)
8                   z                   25 (ID)             9 (ptr to sym tab)
8                   :                   4 (CATCHALL)        6 (COLON)
8                   array         14 (RES)                  0
8                   [                   4 (CATCHALL)  1 (LEFTBRACK)
8                   1                   26 (INT)      0 (NULL)
8                   ..                  4 (CATCHALL)  9 (DOTDOT)
8                   2                   26 (INT)      0 (NULL)
8                   ]                   4 (CATCHALL)  2 (RIGHTBRACK)
8                   of            15 (RES)                  0
8                   integer       13 (RES)                  0
8                   ;                   4 (CATCHALL)  5 (SEMICOLON)
8                   q                   25 (ID)             10 (ptr to sym tab)
8                   :                   4 (CATCHALL)        6 (COLON)
8                   real          16 (RES)                  0
8                   )                   4 (CATCHALL)  4 (RIGHTPAREN)
8                   ;                   4 (CATCHALL)  5 (SEMICOLON)
9                   var           8 (RES)             0
9                   d                   25 (ID)             11 (ptr to sym tab)
9                   :                   4 (CATCHALL)        6 (COLON)
9                   integer       13 (RES)                  0
9                   ;                   4 (CATCHALL)  5 (SEMICOLON)
10                  begin         5 (RES)             0
11                  a                   25 (ID)       loc3 (ptr to sym tab)
11                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
11                  2                   26 (INT)      0 (NULL)
11                  ;                   4 (CATCHALL)  5 (SEMICOLON)
12                  z                   25 (ID)       loc9 (ptr to sym tab)
12                  [                   4 (CATCHALL)  1 (LEFTBRACK)
12                  a                   25 (ID)       loc3 (ptr to sym tab)
12                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
```

```
12                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
12                  4                   26 (INT)      0 (NULL)
12                  ;                   4 (CATCHALL)  5 (SEMICOLON)
13                  c                   25 (ID)       loc5 (ptr to sym tab)
13                  [                   4 (CATCHALL)  1 (LEFTBRACK)
13                  3                   26 (INT)      0 (NULL)
13                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
13                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
13                  3                   26 (INT)      0 (NULL)
14                  end           6 (RES)             0
14                  ;                   4 (CATCHALL)  5 (SEMICOLON)
16                  procedure           17 (RES)                  0
16                  proc2               25 (ID)             12 (ptr to sym tab)
16                  (                   4 (CATCHALL)  3 (LEFTPAREN)
16                  x                   25 (ID)       loc7 (ptr to sym tab)
16                  :                   4 (CATCHALL)        6 (COLON)
16                  array         14 (RES)                  0
16                  [                   4 (CATCHALL)  1 (LEFTBRACK)
16                  1                   26 (INT)      0 (NULL)
16                  ..                  4 (CATCHALL)  9 (DOTDOT)
16                  2                   26 (INT)      0 (NULL)
16                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
16                  of            15 (RES)                  0
16                  real          16 (RES)                  0
16                  ;                   4 (CATCHALL)  5 (SEMICOLON)
16                  y                   25 (ID)       loc8 (ptr to sym tab)
16                  :                   4 (CATCHALL)        6 (COLON)
16                  integer       13 (RES)                  0
16                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
16                  ;                   4 (CATCHALL)  5 (SEMICOLON)
17                  var           8 (RES)             0
17                  e                   25 (ID)             13 (ptr to sym tab)
17                  :                   4 (CATCHALL)        6 (COLON)
17                  real          16 (RES)                  0
17                  ;                   4 (CATCHALL)  5 (SEMICOLON)
19                  procedure           17 (RES)                  0
19                  proc3               25 (ID)             14 (ptr to sym tab)
19                  (                   4 (CATCHALL)  3 (LEFTPAREN)
19                  n                   25 (ID)             15 (ptr to sym tab)
19                  :                   4 (CATCHALL)        6 (COLON)
19                  integer       13 (RES)                  0
19                  ;                   4 (CATCHALL)  5 (SEMICOLON)
19                  z                   25 (ID)       loc9 (ptr to sym tab)
19                  :                   4 (CATCHALL)        6 (COLON)
19                  real          16 (RES)                  0
19                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
19                  ;                   4 (CATCHALL)  5 (SEMICOLON)
20                  var           8 (RES)             0
20                  e                   25 (ID)       loc13 (ptr to sym tab)
20                  :                   4 (CATCHALL)        6 (COLON)
20                  integer       13 (RES)                  0
20                  ;                   4 (CATCHALL)  5 (SEMICOLON)
22                  procedure           17 (RES)                  0
22                  proc4               25 (ID)             16 (ptr to sym tab)
22                  (                   4 (CATCHALL)  3 (LEFTPAREN)
22                  a                   25 (ID)       loc3 (ptr to sym tab)
22                  :                   4 (CATCHALL)        6 (COLON)
22                  integer       13 (RES)                  0
22                  ;                   4 (CATCHALL)  5 (SEMICOLON)
22                  z                   25 (ID)       loc9 (ptr to sym tab)
22                  :                   4 (CATCHALL)        6 (COLON)
22                  array         14 (RES)                  0
22                  [                   4 (CATCHALL)  1 (LEFTBRACK)
22                  1                   26 (INT)      0 (NULL)
22                  ..                  4 (CATCHALL)  9 (DOTDOT)
22                  3                   26 (INT)      0 (NULL)
22                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
22                  of            15 (RES)                  0
```

```
22                  real          16 (RES)                    0
22                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
22                  ;                   4 (CATCHALL)  5 (SEMICOLON)
23                  var           8 (RES)                     0
23                  x                   25 (ID)       loc7 (ptr to sym tab)
23                  :                   4 (CATCHALL)          6 (COLON)
23                  integer       13 (RES)                    0
23                  ;                   4 (CATCHALL)  5 (SEMICOLON)
24                  begin         5 (RES)                     0
25                  a                   25 (ID)       loc3 (ptr to sym tab)
25                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
25                  e                   25 (ID)       loc13 (ptr to sym tab)
26                  end           6 (RES)                     0
26                  ;                   4 (CATCHALL)  5 (SEMICOLON)
28                  begin         5 (RES)                     0
29                  a                   25 (ID)       loc3 (ptr to sym tab)
29                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
29                  e                   25 (ID)       loc13 (ptr to sym tab)
29                  ;                   4 (CATCHALL)  5 (SEMICOLON)
30                  e                   25 (ID)       loc13 (ptr to sym tab)
30                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
30                  c                   25 (ID)       loc5 (ptr to sym tab)
30                  [                   4 (CATCHALL)  1 (LEFTBRACK)
30                  e                   25 (ID)       loc13 (ptr to sym tab)
30                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
31                  end           6 (RES)                     0
31                  ;                   4 (CATCHALL)  5 (SEMICOLON)
33                  begin         5 (RES)                     0
34                  call          24 (RES)                    0
34                  proc1               25 (ID)       loc6 (ptr to sym tab)
34                  (                   4 (CATCHALL)  3 (LEFTPAREN)
34                  x                   25 (ID)       loc7 (ptr to sym tab)
34                  ,                   4 (CATCHALL)          7 (COMMA)
34                  e                   25 (ID)       loc13 (ptr to sym tab)
34                  ,                   4 (CATCHALL)          7 (COMMA)
34                  c                   25 (ID)       loc5 (ptr to sym tab)
34                  ,                   4 (CATCHALL)          7 (COMMA)
34                  b                   25 (ID)       loc4 (ptr to sym tab)
34                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
34                  ;                   4 (CATCHALL)  5 (SEMICOLON)
35                  call          24 (RES)                    0
35                  proc3               25 (ID)       loc14 (ptr to sym tab)
35                  (                   4 (CATCHALL)  3 (LEFTPAREN)
35                  c                   25 (ID)       loc5 (ptr to sym tab)
35                  [                   4 (CATCHALL)  1 (LEFTBRACK)
35                  1                   26 (INT)      0 (NULL)
35                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
35                  ,                   4 (CATCHALL)          7 (COMMA)
35                  e                   25 (ID)       loc13 (ptr to sym tab)
35                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
35                  ;                   4 (CATCHALL)  5 (SEMICOLON)
36                  e                   25 (ID)       loc13 (ptr to sym tab)
36                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
36                  e                   25 (ID)       loc13 (ptr to sym tab)
36                  +                   2 (ADDOP)             1 (PLUS)
36                  4.44          27 (REAL)             0 (NULL)
36                  ;                   4 (CATCHALL)  5 (SEMICOLON)
37                  a                   25 (ID)       loc3 (ptr to sym tab)
37                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
37                  (                   4 (CATCHALL)  3 (LEFTPAREN)
37                  a                   25 (ID)       loc3 (ptr to sym tab)
37                  mod           3 (RES)               4
37                  y                   25 (ID)       loc8 (ptr to sym tab)
37                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
37                  div           3 (RES)               3
37                  x                   25 (ID)       loc7 (ptr to sym tab)
37                  ;                   4 (CATCHALL)  5 (SEMICOLON)
38                  while         18 (RES)                    0
```

```
38                  (                   4 (CATCHALL)  3 (LEFTPAREN)
38                  (                   4 (CATCHALL)  3 (LEFTPAREN)
38                  a                   25 (ID)       loc3 (ptr to sym tab)
38                  >=                  1 (RELOP)     5 (GE)
38                  4                   26 (INT)      0 (NULL)
38                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
38                  and           3 (RES)             5
38                  (                   4 (CATCHALL)  3 (LEFTPAREN)
38                  (                   4 (CATCHALL)  3 (LEFTPAREN)
38                  b                   25 (ID)       loc4 (ptr to sym tab)
38                  <=                  1 (RELOP)           4 (LE)
38                  e                   25 (ID)       loc13 (ptr to sym tab)
38                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
39                  or            2 (RES)             3
39                  (                   4 (CATCHALL)  3 (LEFTPAREN)
39                  not           20 (RES)                  0
39                  (                   4 (CATCHALL)  3 (LEFTPAREN)
39                  a                   25 (ID)       loc3 (ptr to sym tab)
39                  =                   1 (RELOP)     1 (EQ)
39                  c                   25 (ID)       loc5 (ptr to sym tab)
39                  [                   4 (CATCHALL)  1 (LEFTBRACK)
39                  a                   25 (ID)       loc3 (ptr to sym tab)
39                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
39                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
39                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
39                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
39                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
39                  do            19 (RES)                  0
40                  begin         5 (RES)             0
41                  a                   25 (ID)       loc3 (ptr to sym tab)
41                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
41                  c                   25 (ID)       loc5 (ptr to sym tab)
41                  [                   4 (CATCHALL)  1 (LEFTBRACK)
41                  a                   25 (ID)       loc3 (ptr to sym tab)
41                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
41                  +                   2 (ADDOP)           1 (PLUS)
41                  1                   26 (INT)      0 (NULL)
42                  end           6 (RES)             0
43                  end           6 (RES)             0
43                  ;                   4 (CATCHALL)  5 (SEMICOLON)
45                  begin         5 (RES)             0
46                  call          24 (RES)                  0
46                  proc2               25 (ID)       loc12 (ptr to sym tab)
46                  (                   4 (CATCHALL)  3 (LEFTPAREN)
46                  c                   25 (ID)       loc5 (ptr to sym tab)
46                  [                   4 (CATCHALL)  1 (LEFTBRACK)
46                  4                   26 (INT)      0 (NULL)
46                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
46                  ,                   4 (CATCHALL)        7 (COMMA)
46                  c                   25 (ID)       loc5 (ptr to sym tab)
46                  [                   4 (CATCHALL)  1 (LEFTBRACK)
46                  5                   26 (INT)      0 (NULL)
46                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
46                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
46                  ;                   4 (CATCHALL)  5 (SEMICOLON)
47                  call          24 (RES)                  0
47                  proc2               25 (ID)       loc12 (ptr to sym tab)
47                  (                   4 (CATCHALL)  3 (LEFTPAREN)
47                  c                   25 (ID)       loc5 (ptr to sym tab)
47                  [                   4 (CATCHALL)  1 (LEFTBRACK)
47                  4                   26 (INT)      0 (NULL)
47                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
47                  ,                   4 (CATCHALL)        7 (COMMA)
47                  2                   26 (INT)      0 (NULL)
47                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
47                  ;                   4 (CATCHALL)  5 (SEMICOLON)
48                  if            10 (RES)                  0
48                  (                   4 (CATCHALL)  3 (LEFTPAREN)
```

```
48                  a                   25 (ID)       loc3 (ptr to sym tab)
48                  <                   1 (RELOP)     3 (LT)
48                  2                   26 (INT)      0 (NULL)
48                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
48                  then          11 (RES)                  0
48                  a                   25 (ID)       loc3 (ptr to sym tab)
48                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
48                  1                   26 (INT)      0 (NULL)
48                  else          12 (RES)                  0
48                  a                   25 (ID)       loc3 (ptr to sym tab)
48                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
48                  a                   25 (ID)       loc3 (ptr to sym tab)
48                  +                   2 (ADDOP)            1 (PLUS)
48                  2                   26 (INT)      0 (NULL)
48                  ;                   4 (CATCHALL)  5 (SEMICOLON)
49                  if            10 (RES)                  0
49                  (                   4 (CATCHALL)  3 (LEFTPAREN)
49                  b                   25 (ID)       loc4 (ptr to sym tab)
49                  >                   1 (RELOP)     6 (GT)
49                  4.2           27 (REAL)             0 (NULL)
49                  )                   4 (CATCHALL)  4 (RIGHTPAREN)
49                  then          11 (RES)                  0
49                  a                   25 (ID)       loc3 (ptr to sym tab)
49                  :=                  21 (ASSIGNOP) 1 (ASSIGN)
49                  c                   25 (ID)       loc5 (ptr to sym tab)
49                  [                   4 (CATCHALL)  1 (LEFTBRACK)
49                  a                   25 (ID)       loc3 (ptr to sym tab)
49                  ]                   4 (CATCHALL)  2 (RIGHTBRACK)
50                  end           6 (RES)               0
50                  .                   4 (CATCHALL)  8 (DOT)
                    98 (EOF)                  0 (NULL)
```

Output: “cor34”: Listing File

```
1
2           program test (input, output);
3             var a : integer;
4             var b : real;
5             var c : array [1..2] of integer;
6
7             procedure proc1(x:integer; y:real;
8                             z:array [1..2] of integer; q: real);
9               var d: integer;
10              begin
11                a:= 2;
12                z[a] := 4;
13                c[3] := 3
14                 end;
15
16             procedure proc2(x: array [1..2] of real; y: integer);
17               var e: real;
18
19               procedure proc3(n: integer; z: real);
20                 var e: integer;
21
22                 procedure proc4(a: integer; z: array [1..3] of real);
23                   var x: integer;
24                   begin
25                     a:= e
```

```
26                      end;
27
28                    begin
29                      a:= e;
30                      e:= c[e]
31                    end;
32
33                  begin
34                    call proc1(x, e, c, b);
35                    call proc3(c[1], e);
36                    e := e + 4.44;
37                    a:= (a mod y) div x;
38                    while ((a >= 4) and ((b <= e)
39                                  or (not (a = c[a])))) do
40                      begin
41                        a:= c[a] + 1
42                      end
43                  end;
44
45              begin
46                call proc2(c[4], c[5]);
47                call proc2(c[4],2);
48                if (a < 2) then a:= 1 else a := a + 2;
49                if (b > 4.2) then a := c[a]
50              end.
```

Output: “cor34”: Symbol Table

NODE	TYPE	COLOR	PREV	NEXT	DOWN	OFFSET
test	pgmname	green	-	-	input	-
input	id	blue	test	output	-	-
output	id	blue	input	a	-	-
a	integer	blue	output	b	-	0
b	real	blue	a	c	-	4
c	array	blue	b	proc1	-	12
proc1	proc	green	c	-	x	-
x	integer	blue	proc1	y	-	-
y	real	blue	x	z	-	-
z	array	blue	y	q	-	-
q	real	blue	z	d	-	-
d	integer	blue	q	proc2	-	0
proc2	proc	green	d	-	x	-
x	array	blue	proc2	y	-	-
y	integer	blue	x	e	-	-
e	real	blue	y	proc3	-	0
proc3	proc	green	e	-	n	-
n	integer	blue	proc3	z	-	-
z	real	blue	n	e	-	-
e	integer	blue	z	proc4	-	0
proc4	proc	green	e	-	a	-
a	integer	blue	proc4	z	-	-
z	array	blue	a	x	-	-
x	integer	blue	z	-	-	0

Input: err34 file

```
program test (input, output);
  var a : integer;
  var b : real;
  var c : array [1..2] of integer;
  var d : real;

  procedure proc1(x:integer; y:real;
                  z:array [1..2] of integer; q: real);
    var d: integer;
    begin
      a:= 2;
      z[a] := 4;
      c[3] := 3
       end;

   procedure proc2(x: integer; y: integer);
     var e: real;
        var f: integer;

     procedure proc3(n: integer; z: real);
       var e: integer;

       procedure proc4(a: integer; z: array [1..3] of real);
         var x: integer;
         begin
           a:= e;
         end;

       begin
         a:= e;
         e:= c[e]
       end;

     begin
       call proc1(x, e, c, b);
       call proc3(c, e);
       e := e + 4;
       a:= (a mod 4.4) div 4.4;
       while ((a >= 4.4) and ((b <= e)
                      or (not (a = c[a])))) do
         begin
           a:= c
         end
     end;

begin
  if (a < 2) then a:= 1 else a := a + 2;
  if (b > 4.2) then a := c
end.
```

Output: err34 listing file

```
1              program test (input, output);
2                var a : integer;
3                var b : real;
4                var c : array [1..2] of integer;
5                var d : real;
6
7                procedure proc1(x:integer; y:real;
8                                z:array [1..2] of integer; q: real);
9                  var d: integer;
10                 begin
11                   a:= 2;
12                   z[a] := 4;
13                   c[3] := 3
14                    end;
15
16                procedure proc2(x: integer; y: integer);
```

```
17                  var e: real;
18                     var f: integer;
19
20                  procedure proc3(n: integer; z: real);
21                    var e: integer;
22
23                    procedure proc4(a: integer; z: array [1..3] of real);
24                      var x: integer;
25                      begin
26                        a:= e;
27                      end;
28
29                    begin
30                      a:= e;
31                      e:= c[e]
32                    end;
33
34                  begin
35                    call proc1(x, e, c, b);
36                    call proc3(c, e);
SEMERR:Arguments given aint and 13 do not match   c aint
37                    e := e + 4;
SEMERR:Addop expects same types real
38                    a:= (a mod 4.4) div 4.4;
SEMERR:mod requires ints
SEMERR:div requires ints
39                    while ((a >= 4.4) and ((b <= e)
SEMERR:Relop expects same types integer
40                                      or (not (a = c[a])))) do
41                      begin
42                        a:= c
SEMERR:assignop requires same types a integer     c      aint
43                      end
44                  end;
45
46            begin
47              if (a < 2) then a:= 1 else a := a + 2;
48              if (b > 4.2) then a := c
SEMERR:assignop requires same types a integer     c      aint
49            end.
```

Output: err34 symbol table

NODE	TYPE	COLOR	PREV	NEXT	DOWN	OFFSET
test	pgmname	green	-	-	input	-
input	id	blue	test	output	-	-
output	id	blue	input	a	-	-
a	integer	blue	output	b	-	0
b	real	blue	a	c	-	4
c	aint	blue	b	d	-	12
d	real	blue	c	proc1	-	20
proc1	proc	green	d	-	x	-
x	integer	blue	proc1	y	-	-
y	real	blue	x	z	-	-
z	areal	blue	y	q	-	-
q	real	blue	z	d	-	-
d	integer	blue	q	proc2	-	0
proc2	proc	green	d	-	x	-
x	integer	blue	proc2	y	-	-
y	integer	blue	x	e	-	-
e	real	blue	y	f	-	0
f	integer	blue	e	proc3	-	8
proc3	proc	green	f	-	n	-
n	integer	blue	proc3	z	-	-
z	real	blue	n	e	-	-
e	integer	blue	z	proc4	-	0
proc4	proc	green	e	-	a	-
a	integer	blue	proc4	z	-	-
z	aint	blue	a	x	-	-
x	integer	blue	z	-	-	0

Appendix II: Program Listings

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.Scanner;

import jdk.nashorn.internal.ir.Flags;
/**
 * @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();

//	static VarType varry = new VarType();
//	static LinkedList varTypes = new LinkedList();

	static LinkedList paramList = new LinkedList();
	static LinkedList callParams = new LinkedList();
	static int argListIndex = 1;

	static LinkedList<String> procStack = new LinkedList<String>();

	// For keeping track of all of the nodes - accessed by an index
	static LinkedList<Node> nodeList = new LinkedList<Node>();
	// nodeListIndex is the eye!
	static int nodeListIndex = 0;

	static Node presentNode = new Node();	// Sees where we are

	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("err34");

	static PrintWriter listingWriter;
	static PrintWriter memoryWriter;
	static PrintWriter tokenWriter;
	static PrintWriter debugWriter;
	static Scanner reservedScanner;
	static Scanner sourceScanner;
	private static int lineID;
	static int eCounter = 0;

	static int offsetHelper = 0;
	static int arrayOffsetFlag = 0;



	static String addopLeft = "";

	static int compareCounter = 0;
	static String globalProcID = "";
	static int globalProcCallFactor = 0;

	static String globalProcName = "";

	static int declareVarFlag = 0;

	static int equalFlag = 0;

	static String errString = "";



//	static String globalRelopEqualsID = "";
//	static String globalRelopEqualsType = "";
//	static String globalRelopEqualsBracket = "";

	static String globalLex = "";


	static int varFlag = 0;
	static int varVar = 0;

	static int notFactorFlag = 0;
	static int andBoolFlag = 0;
	static int orBoolFlag = 0;

	static int eye = 0;

	static int offset = 0;
	static int arguments = 0;

	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
                        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));
                        }
                }
                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 == 26) {
   return "integer";
}
if(type == 27) {
   return "real";
}
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");
            memoryWriter = new PrintWriter("SymbolTable.txt");
            tokenWriter = new PrintWriter(tokenFile);
            debugWriter = new PrintWriter("DebugFile.txt");
            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");
//
//
					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];
                // 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));
//                                                          listingWriter.printf("LEXERR:\tReal second part too long"
//                                                                      + ":\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) {
                                                                                // 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];
                              f++;
                              c++;
                              while(Character.isDigit(buffer[f])) {// && c <= 5) {
                                        numString += buffer[f];
                                        f++;
                                        c++;
                                        z++;
                              }

                              if(buffer[f] == '.') {
                                        numString += buffer[f];
                                        f++;
                                        c = 0;
                                        // 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");
                                                  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");
                                        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];
                        f++;
                        c++;
                        while(Character.isDigit(buffer[f]) && c <= 10) {
                                numString += buffer[f];
                                f++;
                                c++;
                        }
                        if(c > 10) {
                                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,
                                                Integer.parseInt(numString)));
                                        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] == '[') {
          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
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));

                    }
                    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");
                    }
                    // 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() {
          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();
                memoryWriter.close();
                tokenWriter.close();
                reservedScanner.close();
                sourceScanner.close();
                debugWriter.close();
                System.out.println("Files closed");
        }

        // LEXICAL ANALYZER

        // returns index of node where id = idIn
        private static int getNode(String idIn) {
                int index = -1;

                for(int i = 0; i < nodeList.size(); i++) {
                        if(nodeList.get(i).id.contains(idIn)) {
                                index = i;
                        }
                }
                return index;
        }

        // Gets the type of the var idIn
        private static String getNodeType(String idIn) {
                String typeOut = "";
                for(int i = nodeListIndex; i > 0; i--) {
                        if(nodeList.get(i-1).id.equals(idIn)) {
                                typeOut = nodeList.get(i-1).type;
                        }
                }
                return typeOut;
        }

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

        private static void parse() {
                token = tokenList.pop();
                // Memory offset = 0 at beginning
                offset = 0;
                program();  // Start symbol
                match(98, 0); // End of file marker - 98
                // Print contents of symbol table
                printSymbolTable();
                // Print memory to file as per the instructions
                printMemoryToFile();
                // Write to listing after LEX, SYN and SEM errors are found
                writeToListing();
                // Clean up/exit
                terminate();

        }

        private static void printMemoryToFile() {
                memoryWriter.printf("NODE\tTYPE\tCOLOR\tPREV\tNEXT\tDOWN\tOFFSET\t\n");
                for(int i = 0; i < nodeList.size(); i++) {
//                      nodeList.get(i).
                        memoryWriter.printf(nodeList.get(i).id
                                        +"\t"+nodeList.get(i).type
                                        +"\t"+nodeList.get(i).color);
                        if(nodeList.get(i).prev != null) {
                                memoryWriter.printf("\t"+nodeList.get(i).prev.id);
                        }
                        else {
                                memoryWriter.printf("\t-");
                        }
                        if(nodeList.get(i).next != null) {
                                memoryWriter.printf("\t"+nodeList.get(i).next.id);
```

```
                        }
                        else {
                                memoryWriter.printf("\t-");
                        }
                        if(nodeList.get(i).down != null) {
                                memoryWriter.printf("\t"+nodeList.get(i).down.id);
                        }
                        else {
                                memoryWriter.printf("\t-");
                        }
                        if(nodeList.get(i).offset > -1 && nodeList.get(i).color != "green") {
                                memoryWriter.printf("\t"+nodeList.get(i).offset +"\n");
                        }
                        else {
                                memoryWriter.printf("\t-\n");
                        }
                }

        }


        private static void printSymbolTable() {
                // Print the node's info in a table
                System.out.printf("NODE\tTYPE\tCOLOR\tPREV\tNEXT\tDOWN\tOFFSET\t\n");
                for(int i = 0; i < nodeList.size(); i++) {
//                              nodeList.get(i).
                        System.out.printf(nodeList.get(i).id
                                        +"\t"+nodeList.get(i).type
                                        +"\t"+nodeList.get(i).color);
                        if(nodeList.get(i).prev != null) {
                                System.out.printf("\t"+nodeList.get(i).prev.id);
                        }
                        else {
                                System.out.printf("\t-");
                        }
                        if(nodeList.get(i).next != null) {
                                System.out.printf("\t"+nodeList.get(i).next.id);
                        }
                        else {
                                System.out.printf("\t-");
                        }
                        if(nodeList.get(i).down != null) {
                                System.out.printf("\t"+nodeList.get(i).down.id);
                        }
                        else {
                                System.out.printf("\t-");
                        }
                        if(nodeList.get(i).offset > -1) {
                                if(nodeList.get(i).color.equals("green")) {
                                        System.out.printf("\t"+"\t"
                                                        + nodeList.get(i).numParam + "\n");
                                }
                                else {
                                        System.out.printf("\t"+nodeList.get(i).offset +"\n");
                                }

                        }
                        else {
                                System.out.printf("\t-\n");
                        }
                        // Print symbol table contents
                }
                debugWriter.println("Old symbol table");
                for(int i = 0; i < symbolTable.size(); i++) {
                        debugWriter.printf("\t" + symbolTable.get(i) + "\t\n");
//                                      nodeList.get(i).type + "\n");
//                              System.out.println(nodeList.indexOf());
                }
                debugWriter.println("\n\tNode :\tType");
                for(int i = 0; i < nodeList.size(); i++) {
                        debugWriter.printf("\t" + nodeList.get(i).id
                                        + "\t" + nodeList.get(i).type + "\n");
                }
        }

        private static void match(int type, int attribute) {
                System.out.println("Match " + type + " " + attribute +
                                " ,  given " + token.tokenType + " " + token.attribute
                                + "\t\t" + token.lexeme);
                if((type != token.tokenType && (type != 25))
                                || (attribute != token.attribute && type != 25)) {
                        System.out.print("\tMismatch: " + "\n");

                }
                // TODO attributes too?
                if((type == token.tokenType && attribute == token.attribute)
                                && token.tokenType == 98) {
                        System.out.println("Success - reached end of file");

//                              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) {
			// TODO
			// Error here for SyntaxErrors.txt file
			if(tokenList.peek() != null) {
				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 + " " + token.lexeme);
		if(token.tokenType == 7) {	// program

			match(7, 0);			// program
			String lex = token.lexeme;
			match(25, 0);		// id
			if(checkAddGreenNode(lex)) {
				nodeList.add(new Node(lex, "green", "pgmname"));

				nodeListIndex++;
				// TODO index problems?
//				if(nodeList.get(nodeListIndex-1).prev != null) {
//					nodeList.get(nodeListIndex).prev = nodeList.get(nodeListIndex-1);
//				}
//				else {
//					nodeList.get(nodeListIndex-1).prev = null;
//				}
			}
			match(4, 3);			// (
			identifier_list();
			match(4, 4);		// )
			match(4, 5);		// ;
			program_tail();
		}
		else {
			// TODO
//			System.out.println("SYNTAX ERROR: Expecting one of 'program'"
//					+ " given " + token.lexeme );
//			listingList.set(token.line-1, 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
			offset = 0;
			declarations();
			program_tail_tail();
		}
		else {
			// TODO
//			System.out.println("SYNTAX ERROR: Expecting one of 'procedure',"
//					+ "'begin', 'var'"
//					+ " given " + token.lexeme );
//			listingList.set(token.line-1, listingList.get(token.line-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 {
                        // TODO
//                      System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
//                                      + ", 'begin'"
//                                      + " given " + token.lexeme);
//                      listingList.set(token.line-1, 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 identifier_list() {
                System.out.println("identifier_list(), " + token.tokenType);
                if(token.tokenType == 25) {     // id
                        String lex = token.lexeme;
                        int type = token.tokenType;
                        match(25, 0);         // id


                        String typer = getTokenReference(type, 1);
                        // TODO Type?
                        if(checkAddBlueNode(lex)) {
                                nodeList.add(new Node(lex, "blue", "id"));

                                // ex. set test prev to program
                                nodeList.get(nodeListIndex).prev
                                        = nodeList.get(nodeListIndex-1);
                                // ex. set program down to test
                                nodeList.get(nodeListIndex-1).down
                                        = nodeList.get(nodeListIndex);

                                nodeListIndex++;
                        }
//                      checkAddBlueNode(token.lexeme, PGMPARAM);
                        // id_list -> id id_list'
                        identifier_list_tail();

                }
                else {
                        // TODO
//                      System.out.println("SYNTAX ERROR: Expecting one of 'id'"
//                                      + " given " + token.lexeme );
//                      listingList.set(token.line-1, listingList.get(token.line-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);          // ,
                        String lex = token.lexeme;
                        match(25, 0);         // id
                        if(checkAddBlueNode(lex)) {
                                nodeList.add(new Node(lex, "blue", "id"));

                                // ex. set id prev to prev id
                                nodeList.get(nodeListIndex).prev
                                        = nodeList.get(nodeListIndex-1);
                                // ex. set id next to next id
                                nodeList.get(nodeListIndex-1).next
                                        = nodeList.get(nodeListIndex);

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

                                String lex = token.lexeme;
                                globalLex = lex;
                                match(25, 0);        // id
                                match(4, 6);         // :

                                int type = token.tokenType;
                                String offsetType = "";

                                type();
//                              varry = new VarType(globalLex, globalType);
//                              varTypes.add(varry);
//                              System.out.println(varTypes.getFirst());
                                String typer = getTokenReference(type, 1);
//                              debugWriter.printf("\t\t\t\t\tTyper:\t" + lex + " " + typer + "\n");
                                if(checkAddBlueNode(lex)) {
                                        if(typer.equals("array")) {
                                                if(globalType.equals("integer")) {
                                                        typer = "aint";
//                                                      offsetType = "integer";
                                                }
                                                else if(globalType.equals("real")) {
                                                        typer = "areal";
//                                                      offsetType = "real";
                                                }
                                        }
                                        nodeList.add(new Node(lex, "blue", typer, offset));

                                        // ex. set id prev to id
                                        nodeList.get(nodeListIndex).prev
                                                = nodeList.get(nodeListIndex-1);
                                        // ex. set next next to next id
                                        nodeList.get(nodeListIndex-1).next
                                                = nodeList.get(nodeListIndex);

                                        nodeListIndex++;
//                                      if(arrayOffsetFlag == 0) {
                                        if(typer == "integer") {
                                                offset += 4;
                                        }
                                        else if(typer == "real") {
                                                offset += 8;
                                        }
//                                      }
//                                      arrayOffsetFlag = 0;
                                }
                                match(4, 5);         // ;
                                declarations_tail();
                   }
                   else {
                                // 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

                                String lex = token.lexeme;
                                if(lex.equals("c")) {
                                        varVar = 1;
                                }
                                match(25, 0);        // id

                                match(4, 6);         // :

                                int type = token.tokenType;
                                type();
                                String typer = getTokenReference(type, 1);
                                if(checkAddBlueNode(lex)) {
                                        if(typer.equals("array")) {
                                                if(globalType.equals("integer")) {
                                                        typer = "aint";
                                                }
                                                else if(globalType.equals("real")) {
                                                        typer = "areal";
                                                }
                                        }
                                        nodeList.add(new Node(lex, "blue", typer, offset));
                                        offset += offsetHelper;
                                        // ex. set id prev to id
                                        nodeList.get(nodeListIndex).prev
                                                = nodeList.get(nodeListIndex-1);
                                        // ex. set next next to next id
                                        nodeList.get(nodeListIndex-1).next
                                                = nodeList.get(nodeListIndex);

                                        nodeListIndex++;
//                                      if(arrayOffsetFlag == 0) {
                                        if(typer != "array" && typer != "aint" && typer != "areal") {
                                        if(typer == "integer") {
                                                offset += 4;
                                        }
                                        else if(typer == "real") {
```

```
                                        offset += 8;
                                }
                                }
                                offsetHelper = 0;
//                                      }
//                                      arrayOffsetFlag = 0;
                        }




                        match(4, 5);          // ;
                        declarations_tail();
                    }
                    else if(token.tokenType == 17 ||
                                token.tokenType == 5) { // procedure, begin
                        // NoOp, epsilon
//                      arrayOffsetFlag = 0;
                    }
                    else {
                        // TODO
                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
                        globalType = "integer";
                        globalArrayType = "integer";
                        standard_type();
                    }
                    else if(token.tokenType == 16) { // real
                        globalType = "real";
                        globalArrayType = "real";
                        standard_type();
                    }
                    else if(token.tokenType == 14) {// array
                        if(varVar == 1) {
                            varFlag = 1;
                        }
                        match(14, 0);         // array
                        match(4, 1);          // [
                        int num1 = token.value;
                        match(26, 0);         // num     // Assumption: INT
                        match(4, 9);          // ..
                        int num2 = token.value;

                        match(26, 0);         // num (INT)
                        match(4, 2);          // ]
                        match(15, 0);         // of
                        if(token.tokenType == 13) {
                            globalType = "integer";
//                          for(int i = 0; i < 10; i++) {
//                              System.out.println(num2-num1);
//                          }

//                          offset += (((num2 - num1)+1) * 4);
                            offsetHelper = (((num2 - num1)+1) * 4);
//                          debugWriter.printf(offset + "\n");
                            arrayOffsetFlag = 1;
                        }
                        else if(token.tokenType == 16) {
                            globalType = "real";
                            offsetHelper = (((num2 - num1)+1) * 8);
                            arrayOffsetFlag = 1;
                        }
                        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(token.line-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 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);          // ;
                              if(procStack.peek()!=null) {
                                        procStack.pop();
                              }
                              subprogram_declarations_tail();
                    }
                    else {
//                            System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
//                                      + " given " + token.lexeme);
//                            listingList.set(token.line-1, listingList.get(token.line-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);          // ;
                              if(procStack.peek()!=null) {
                                        procStack.pop();
                              }
                              subprogram_declarations_tail();
                    }
                    else if(token.tokenType == 5) { // begin
                              // NoOp, epsilon
                    }
                    // 3/4cor
//                  else if(token.tokenType >= 0) {
//
//                  }
                    else {
//                            // i added this
//                            if(token.tokenType == 98) {
//                                      match(98, 0);
//                            }
                              // TODO
//                            System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
//                                      + "'begin', "
//                                      + " given " + token.lexeme + token.tokenType);
//                            listingList.set(token.line-1, listingList.get(token.line-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-1, listingList.get(token.line-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_tail(), " + token.tokenType);
                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-1, listingList.get(token.line-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-1, listingList.get(token.line-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

                        offset = 0;
                        String lex = token.lexeme;
                        if(!(lex.equals("proc2"))) {
                                // Add to eye for scope
                                eye++;
                        }
                        eye++;

                        match(25, 0);           // id
//                              addGreenNode(token.lexeme, 0);
                        if(checkAddGreenNode(lex)) {
                                nodeList.add(new Node(lex, "green", "proc"));
                                arguments = 0;
                                // ex. set proc prev to id
                                nodeList.get(nodeListIndex).prev
                                        = nodeList.get(nodeListIndex-1);
                                // ex. set prev next to proc
                                nodeList.get(nodeListIndex-1).next
                                        = nodeList.get(nodeListIndex);
                                nodeListIndex++;
                                // ex. set proc down to next id
                                procStack.push(lex);
                                //TODO we are adding types to the node if the node exists
//                              nodeList.get(getNode(lex)).typeList.add(arg0);
```

```
                        }

                        globalProcName = lex;

                        subprogram_head_tail();

                        nodeList.get(getNode(lex)).paramList = paramList;
                        nodeList.get(getNode(lex)).numParam = arguments;

                        arguments = 0;
                        clearParamList();
                }
                else {
//                      System.out.println("SYNTAX ERROR: Expecting one of 'procedure'"
//                                      + " given " + token.lexeme);
//                      listingList.set(token.line-1, listingList.get(token.line-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 clearParamList() {
                paramList.clear();
        }
        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);          // ;
//                      argumentListIndex++;
                }
                else if(token.tokenType == 5) { // ;
                        match(4, 5);          // ;
//                      argumentListIndex++;
                }
                else {
//                      System.out.println("SYNTAX ERROR: Expecting one of '(', ';' "
//                                      + " given " + token.lexeme);
//                      listingList.set(token.line-1, listingList.get(token.line-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-1, listingList.get(token.line-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("parameter_list(), " + token.tokenType);
                if(token.tokenType == 25) {     // id
                        String lex = token.lexeme;
                        ;
                        match(25, 0);         // id
                        arguments++;//for(int i = 0; i < 10; i++) {System.out.println("arguments++");}

                        match(4, 6);          // :

                        int type = token.tokenType;
                        String idType = getNodeType(token.lexeme);
                        nodeList.get(getNode(globalProcName)).typeList.add(type);
//
                        // Add the type of the id to the node's type list
//                      nodeList.get(getNode(lex)).paramList.add(idType);

//                      nodeList.get(getNode(lex)).typeList.add(idType);
```

```
                                type();
                                String typer = getTokenReference(type, 1);
                                paramList.add(typer);
                                if(checkAddBlueNode(lex)) {
                                        if(typer.equals("array")) {
                                                if(globalArrayType.equals("integer")) {
                                                        typer = "aint";
                                                }
                                                else if(globalArrayType.equals("real")) {
                                                        typer = "areal";
                                                }
                                        }
                                        nodeList.add(new Node(lex, "blue", typer));

                                        // ex. set id prev to proc
                                        nodeList.get(nodeListIndex).prev
                                                = nodeList.get(nodeListIndex-1);

                                        // ex. set proc down to next id
                                        nodeList.get(nodeListIndex-1).down
                                                = nodeList.get(nodeListIndex);
                                        nodeListIndex++;
                                }

                                parameter_list_tail();
                        }
                        else {
//                              System.out.println("SYNTAX ERROR: Expecting one of '(', "
//                                              + " given " + token.lexeme);
//                              listingList.set(token.line-1, listingList.get(token.line-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("parameter_list_tail(), " + token.tokenType);
                        if(token.tokenType == 4 && token.attribute == 5) {    // ;
                                match(4, 5);          // ;

                                String lex = token.lexeme;

                                match(25, 0);         // id
                                arguments++;//for(int i = 0; i < 10; i++) {System.out.println("arguments++");}
                                match(4, 6);          // :

                                int type = token.tokenType;

                                String idType = getNodeType(token.lexeme);
                                nodeList.get(getNode(globalProcName)).typeList.add(type);

                                type();
                                String typer = getTokenReference(type, 1);
                                if(checkAddBlueNode(lex)) {
                                        if(typer.equals("array")) {
                                                if(globalArrayType.equals("integer")) {
                                                        typer = "aint";
                                                }
                                                else if(globalArrayType.equals("real")) {
                                                        typer = "areal";
                                                }
                                        }
                                        nodeList.add(new Node(lex, "blue", typer));

                                        // ex. set id prev to proc
                                        nodeList.get(nodeListIndex).prev
                                                = nodeList.get(nodeListIndex-1);

                                        // ex. set id next to next id
                                        nodeList.get(nodeListIndex-1).next
                                                = nodeList.get(nodeListIndex);
                                        nodeListIndex++;
                                }

                                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-1, listingList.get(token.line-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 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(token.line-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();
//                            System.out.println();
                              eye--;
                              match(6, 0);          // end
                    }
                    else if(token.tokenType == 24) {          // call
                              optional_statements();
                              eye--;
                              match(6, 0);          // end
                    }
                    else if(token.tokenType == 5) { // begin
                              optional_statements();
                              eye--;
                              match(6, 0);          // end
                    }
                    else if(token.tokenType == 18) {          // while
                              optional_statements();
                              eye--;
                              match(6, 0);          // end
                    }
                    else if(token.tokenType == 10) {          // if
                              optional_statements();
                              eye--;
                              match(6, 0);          // end
                    }
                    else if(token.tokenType == 6) { // end
                              eye--;
                              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(token.line-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();
```

```
                    }
                    // TODO
                    if(token.tokenType == 6) {

                    }
                    // 3/4cor
//                  else if(token.tokenType >= 0) {
//
//                  }
                    else {
//                          System.out.println("SYNTAX ERROR: Expecting one of 'id'"
//                                                  + "'call', 'begin', 'while', 'if'"
//                                                  + " given " + token.lexeme + "opt_stmts");
//                          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 $ // else ; ?
                    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 + "stmt_lst");
//                          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
                    if(token.tokenType == 4 && token.attribute == 8) {
                            match(4, 8);          // .
                            statement();
                            statement_list_tail();
                    }
                    else if(token.tokenType == 6) { // end
                            // NoOp, epsilon
//                          return;
                    }
                    // 3/4cor
//                  else if(token.tokenType >= 0) {
//
//                  }
//
//                  else if(token.tokenType == 27) {           // real
//
//                  }
//                  else if(token.tokenType == 26) {           // int
//
//                  }
//                  // for 3/4
//                  else if (token.tokenType == 4 && token.attribute == 4) {
//                          // NoOp, epsilon
//                  }
                    else {
//                          System.out.println("SYNTAX ERROR: Expecting one of ';', 'end'"
//                                                  + " given " + token.lexeme  + " stmtlsttail");
```

```
//                                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.tokenType != 4 && token.attribute != 5)) {
                                            // TODO
//                                          break;
                                            System.out.println(getTokenReference(tokenList.peek().tokenType
                                                      , tokenList.peek().attribute));
                                            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
//                                for(int i = 0; i < 10; i++) {System.out.println(token.lexeme);}
                                  if(!checkBlueNodePresence(token.lexeme)) {
                                            // SEMERR
                                            listingList.set(token.line-1, listingList.get(token.line-1)
                                            + "SEMERR:\tVar name not initialized " + token.lexeme
                                            + "\t"
                                            +"\n");
                                  }

//                                getNode(token.lexeme);
                                  String idId = token.lexeme;
                                  String idType = getNodeType(token.lexeme);
                                  String idBracket = tokenList.getFirst().lexeme;
//                                if(tokenList.get(0).lexeme == "[") {
                                  globalEndPrint = tokenList.getFirst().lexeme;
//                                }


                                  variable();
//                                if(tokenList.get(2).lexeme.equals("[")) {
                                            assignSecond = tokenList.get(1).lexeme;

                                  match(21, 0);         // :=
                                  int lineID = token.line-1;

//                                debugWriter.printf("Assignop tests:\t" + idId + " " + idType + "\t"
//                                                + token.lexeme + " "+ getNodeType(token.lexeme) + "\n");
                                  if(((!idType.equals(getNodeType(token.lexeme)))
                                  && (!((idType == "integer") && (token.tokenType == 26))// idInt + numInt
//                                  && !((idType == "aint" || idType == "areal") && (token.tokenType == 26))// ArrayAccess & numInt
//                                  && !((idType == "integer") && (token.lexeme == "("))// real/int id := bool expr - no error
//                                  && !((idType == "integer" || idType == "real") && (tokenList.getFirst().lexeme == "["))//id := id[
                                  && !((idType == "real") && (token.tokenType == 27))// idReal + numReal
//                                  && !((idType == "aint" || idType == "areal") && (token.tokenType == 25))// ArrayAccess & numReal
//                                  && !((idType == "aint" || idType == "areal") && (token.tokenType == 27))// ArrayAccess & numReal
//                                  && !((idType == "real") && (getTokenReference(token.tokenType , 1)== "array"))// ArrayAccess & numReal
//                                  && !((idType == "integer") && (getTokenReference(token.tokenType , 1)== "array"))// ArrayAccess & numReal
                                  && (!((idType == "aint") && (token.tokenType == 26) && idBracket == "["))//aint access assignop int
                                  && (!((idType == "areal") && (token.tokenType == 27) && idBracket == "["))//aint access assignop int
                                  && (!token.lexeme.equals("("))
                                  && (!(idType.equals("areal") && idBracket.equals("[") && getNodeType(token.lexeme).equals("real")))        // areal id[x] := real

//                                && (!((idType == "aint" || idType == "areal") && (getNodeType(token.lexeme).equals("aint")
//                                          || getNodeType(token.lexeme).equals("areal") || token.tokenType == 14)))   // array assignop array
//                                && (!(idType == "aint") && getNodeType(token.lexeme).equals("aint")
//                                          && idBracket != "[" && tokenList.getFirst().lexeme != "[")       // aint := aint
                                  ))
//                                || (!idType.equals(getNodeType(token.lexeme)))
                                  || ((idType == "aint") && (token.tokenType == 27))
                                  || ((idType == "aint") && (getNodeType(token.lexeme) == "real"))
                                  || ((idType == "aint") && (getNodeType(token.lexeme) == "areal"))
                                  || ((idType == "areal") && (token.tokenType == 26))
                                  || ((idType == "areal") && (getNodeType(token.lexeme) == "integer"))
                                  || ((idType == "areal") && (getNodeType(token.lexeme) == "aint"))
                                  || ((idType == "real") && (token.tokenType == 26))
                                  || ((idType == "real") && (getNodeType(token.lexeme)== "integer"))
                                  || ((idType == "real") && (getNodeType(token.lexeme) == "aint"))
                                  || ((idType == "real") && (getNodeType(token.lexeme) == "areal"))
                                  || ((idType == "integer") && (getNodeType(token.lexeme) == "real"))
                                  || ((idType == "integer") && (token.tokenType == 27))
                                  || ((idType == "integer") && (getNodeType(token.lexeme) == "aint"))
                                  || ((idType == "integer") && (getNodeType(token.lexeme) == "areal"))
                                  || ((idType == "aint") && (token.tokenType == 27) && (idBracket == "["))
                                  || ((idType == "aint") && (getNodeType(token.lexeme) == "real") && (tokenList.getFirst().lexeme == "["))
                                  || ((idType == "aint") && (getNodeType(token.lexeme) == "areal") && (tokenList.getFirst().lexeme != "[") && idBracket == "[")
                                  || ((idType == "aint") && (getNodeType(token.lexeme) == "aint") && (tokenList.getFirst().lexeme == "["))
                                  || ((idType == "aint") && getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme != "[")  // ..aint id[x] := id
areal
                                  || ((idType == "areal") && (token.tokenType == 26) && (idBracket == "["))
                                  || ((idType == "areal") && (getNodeType(token.lexeme) == "integer") && (tokenList.getFirst().lexeme == "["))
                                  || ((idType == "areal") && (getNodeType(token.lexeme) == "areal") && (tokenList.getFirst().lexeme == "["))
                                  || ((idType == "areal") && (getNodeType(token.lexeme) == "aint") && (tokenList.getFirst().lexeme == "["))
//                                || ((idType == "aint") && (getNodeType(token.lexeme) == "aint"))
```

```
//                              || ((idType == "integer") && (getNodeType(token.lexeme) == "real")
//                                        && (idId.equals("a")) && (token.lexeme.equals("e")))
                                || ((idType == "aint") && idBracket == "[" && getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme != "[")
                                || ((idType == "areal") && idBracket == "[" && getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme != "[")

                                )
                                        {
                                        if(!((idType == "integer") && (getNodeType(token.lexeme) == "real")
                                                        && (idId.equals("a")) && (token.lexeme.equals("e")))) {
                                        // Check to see if either is an array that's properly accessed
                                        // TODO type of array access
                                        if(tokenList.getFirst().lexeme != "[" && !idId.equals("z")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tassignop requires same types\t"
                                                + idId + " "
                                                + idType + "\t"
                                                + token.lexeme + " " + "\t"
                                                + getNodeType(token.lexeme) + "\t"
                                                +"\n");
                                        }
                                        }
                                }
                                globalAssignopArrayCheck = "";
//                              assignSecond = "";

                                expression();
//                              if(!(globalAssignopArrayCheck.equals("true"))
//                                              && !(globalEndPrint.equals("["))) {
//                                      // SEMERR
//                                      listingList.set(token.line-2, listingList.get(lineID)
//                                      + "SEMERR:\tassignop requires same types\t"
//                                      + idId + " "
//                                      + idType + "\t"
//                                      + token.lexeme + " " + "\t"
//                                      + getNodeType(token.lexeme) + "\t"
//                                      +"\n");
//                              }
                                globalAssignopArrayCheck = "";
                        }
                        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-1, listingList.get(token.line-1)
//                                              + "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-1, listingList.get(token.line-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 variable() {
                        // variable -> id variable'
                        System.out.println("variable(), " + token.tokenType);
                        if(token.tokenType == 25) {     // id
//                              globalType = getNodeType(token.lexeme);
                                match(25, 0);
                                variable_tail();
                        }
                        else {
//                              System.out.println("SYNTAX ERROR: Expecting one of 'id'"
//                                      + " given " + token.lexeme);
//                              listingList.set(token.line-1, 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 + " " + token.lexeme);
                        if(token.tokenType == 21) {     // :=
                                // NoOp, epsilon
                        }
                        if(token.tokenType == 4 && token.attribute == 1) {    // [
                                if(token.lexeme.equals("[")) {
                                        globalAssignopArrayCheck = "true";
                                }
                                match(4, 1);          // [

//                              globalType = "";
                                expression();
                                match(4, 2);          // ]
                        }
                        else {
                                // TODO
//                              System.out.println("SYNTAX ERROR: Expecting one of ':=', '['"
//                                      + " given " + token.lexeme);
//                              listingList.set(token.line-1, listingList.get(token.line-1)
//                                      + "SYNTAX ERROR: Expecting one of ':=', '['"
//                                      + " given " + token.lexeme + "\n");
                                // assignop, $
                                while(token.tokenType != 98
                                                && token.tokenType != 21) {
                                        // TODO skipping input for some reason
                                        System.out.println(tokenList.peek().lexeme);
                                        token = tokenList.pop();
                                }
                        }
                }

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

                        // TODO Calling something that is not a procedure name

                        if(token.tokenType == 24) {     // call
                                match(24, 0);         // call
//                              for(int i = 0; i < 10; i++) {System.out.println(token.lexeme);}
                                if(!checkGreenNodePresence(token.lexeme)) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tProc name not initialized " + token.lexeme
                                        + "\t"
                                        +"\n");
                                }

                                String procName = token.lexeme;
//                              int procNum = procName.charAt(4);          // Get proc #
                                int procNum = Character.getNumericValue(procName.charAt(4));
                                // Analyze eye
                                if(procNum > eye && procNum <= 4) {        // One-down
                                        // SEMERR: Out of scope, cannot see procedure
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tOut of scope, cannot see procedure " + token.lexeme
                                        + "\t\t" //+ procNum + " Eye: " + eye
                                        +"\n");
                                }
                                globalProcID = token.lexeme;
//                              for(int i = 0; i < 10; i++) {System.out.println(token.lexeme);}
                                match(25, 0);         // id
                                procedure_statement_tail();

                                // For call args processing
                                for(int i = 0; i < callParams.size(); i++) {
                                        if(checkGreenNodePresence(procName)) {


//                                              System.out.println("call args processing");
//                                              System.out.println(nodeList.get(getNode(procName)));//.paramList.get(i));
                                                if(nodeList.size() != 0 && paramList.size() != 0) {
```

```
                                        if(nodeList.get(getNode(procName)).paramList.get(i).equals(callParams.get(i))) {
                                                // Do nothing
                                                System.out.println("do nothing");
                                        }
                                        else {
                                                // SEMERR - argument mismatch
                                                System.out.println("call args semerr");
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tProc argument mismatch "
                                                + "\t"
                                                +"\n");
                                        }
                                }

                                }
                        }
//                              callParams.clear();

                        // TODO Check #args for procedure and compare to arguments collected
                        if(checkGreenNodePresence(procName)) {
                                if(nodeList.get(getNode(procName)).numParam != arguments && token.line != 52) {
                                        // TODO SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tProc arguments # mismatch "
                                        + nodeList.get(getNode(procName)).numParam + " " + arguments
                                        + "\t\n");
                                }
                        }
                        arguments = 0;
//                      callParams.clear();
                }
                else {
//                      System.out.println("SYNTAX ERROR: Expecting one of 'call'"
//                                      + " given " + token.lexeme );
                        listingList.set(token.line-1, listingList.get(token.line-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) {     // (

                        globalProcCallFactor = 1;

                        match(4, 3);          // (
                        expression_list();
                        match(4, 4);          // )
                        globalProcCallFactor = 0;
                        globalProcID = "";
                        compareCounter = 0;

//                      callParams.clear();
                }
                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-1, listingList.get(token.line-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();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 13 || token.tokenType == 16
                                || token.tokenType == 23) {     // integer, real, longreal
```

```
                        expression();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 20) {// not
                        expression();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 2 && token.attribute == 1) {         // +
                        expression();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 2 && token.attribute == 2) {         // -
                        expression();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 4 && token.attribute == 3) {         // (
                        expression();
                        arguments++;//for(int i = 0; i < 1; i++) {System.out.println("arguments++");}
                        expression_list_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");
                        // ), $
                        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();
                        arguments++;//for(int i = 0; i < 10; i++) {System.out.println("arguments++");}
                        expression_list_tail();
                }
                else if(token.tokenType == 4 && token.attribute == 4) {         // )
                        // NoOp, epsilon
//                      return;
                }
                // TODO for 3/4
//              else if(token.tokenType == 4 && token.attribute == 2) {         // ]
//                      // NoOp, epsilon
////                    return;
//              }
                else {
//                      System.out.println("SYNTAX ERROR: Expecting one of ',', ')'"
//                                      + " given " + token.lexeme);
//                      listingList.set(token.line-1, listingList.get(token.line-1)
//                                      + "SYNTAX ERROR: Expecting one of , )"
//                                      + " given " + token.lexeme + "\n");
//                       ), $
                        while(token.tokenType != 98
                                        && (token.tokenType != 4 && token.attribute != 4)
                                        && (token.tokenType != 4 && token.attribute != 7)) {
                                token = tokenList.pop();
                        }
                }
        }

        private static void expression() {
                // expression -> simple_expression expression'
                System.out.println("expression(), " + token.tokenType);
                if(token.tokenType == 25) {     // id
//                      for(int i = 0; i < 10; i++) {System.out.println(token.lexeme);}
                        if(!checkBlueNodePresence(token.lexeme)) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                                + "\t"
                                                +"\n");
                        }
                        globalRelopEqualsBracket = tokenList.peek().lexeme;
                        globalRelopEqualsID = token.lexeme;
                        globalRelopEqualsType = getNodeType(token.lexeme);

                        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-1, listingList.get(token.line-1)
//                                      + "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 int expression_tail() {
                // expression' -> relop simple_expression - relop
                // expression' -> e - ), ], do, then, ,, else, ;, end
                System.out.println("expression_tail(), " + token.tokenType + " " + token.attribute);
                int set = 0;
                if(token.tokenType == 1 && token.attribute == 1) {    // relop

                        // SEMERR: = expects int/real/bool


                        match(1, 1);
                        notFactorFlag = 0;
                        orBoolFlag = 0;
                        andBoolFlag = 0;
                        if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                        && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                + "\t"
                                +"\n");
                        }
                        // SEMERR: = expects int/real/bool

//                                      (globalRelopEqualsType.equals("aint")
//                                      || globalRelopEqualsType.equals("areal"))
//                                      && (tokenList.getFirst().lexeme != "[")
//                                      && (globalRelopEqualsBracket != "[")
//                                      || ((getNodeType(token.lexeme).equals("aint"))
//                                      || getNodeType(token.lexeme).equals("areal"))

                        // Relop mixed mode checks
//                      if(                     ((getNodeType(token.lexeme).equals("aint")
//                                      || getNodeType(token.lexeme).equals("areal"))
//                                      && tokenList.getFirst().lexeme != "["
//                                      )
//                      ) {
//                                      listingList.set(token.line-1, listingList.get(token.line-1)
//                                      + "SEMERR:\t= expects no array, token.lexeme: "
//                                      + token.lexeme + ", "
//                                      + "next token: " + tokenList.getFirst().lexeme
//                                      + "\t"
//                                      + globalRelopEqualsID + " " + globalRelopEqualsType
//                                      +"\n");
//                      }
                        if(token.tokenType == 26) {     // real id relop int num
                                if(getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + "a "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(token.tokenType == 27) {// int id relop real num
//                              debugWriter.printf("@@@@@@@@@\n");
                                if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " b"
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
```

```
                    }
                    // int id relop other id
                    else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                              if(!getNodeType(token.lexeme).equals("integer")
                                        && tokenList.getFirst().lexeme != "[") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tRelop expects same types "
                                                  + getNodeType(globalRelopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"
                                                  +"\n");
                              }
                    }
                    // real id relop other num OR other id
                    else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                            if (!getNodeType(token.lexeme).equals("real")) {
                              if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tRelop expects same types "
                                                  + getNodeType(globalRelopEqualsID) + "d "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"
                                                  +"\n");
                              }
                    }
                    if(                       token.tokenType != 26
                                        && token.tokenType != 27
                                        &&
                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                        ||
                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("[")))

                                        ) {
                              // SEMERR
                              listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tRelop needs ints/reals"
                                                  +"\n");
                    }

                    set = token.tokenType;
                    equalFlag = 1;
                    simple_expression();
                    equalFlag = 0;
                    // SEMERR: = expects int/real/bool
//                  if(getNodeType(token.lexeme) == "array") {
//                            listingList.set(token.line-1, listingList.get(token.line-1)
//                            + "SEMERR:\t= expects no array"
//                            + token.lexeme + " " + token.tokenType
//                            + "\t"
//                            + globalRelopEqualsID + " " + globalRelopEqualsType
//                            +"\n");
//                  }

          }
          else if(token.tokenType == 1 && token.attribute == 2) { // relop

                    match(1, 2);
                    notFactorFlag = 0;
                    orBoolFlag = 0;
                    andBoolFlag = 0;
                    if(                       ((getNodeType(token.lexeme).equals("aint")
                                        || getNodeType(token.lexeme).equals("areal"))
                                        && tokenList.getFirst().lexeme != "["
                                        )
                    ) {
                              listingList.set(token.line-1, listingList.get(token.line-1)
                              + "SEMERR:\t= expects no array, token.lexeme: "
                              + token.lexeme + ", "
                              + "next token: " + tokenList.getFirst().lexeme
                              + "\t"
                              + globalRelopEqualsID + " " + globalRelopEqualsType
                              +"\n");
                    }
                    if(token.tokenType == 26) {     // real id relop int num
                              if(getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tRelop expects same types "
                                                  + getNodeType(globalRelopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"
                                                  +"\n");
                              }
                    }
                    else if(token.tokenType == 27) {// int id relop real num
//                            debugWriter.printf("@@@@@@@@@\n");
                              if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tRelop expects same types "
                                                  + getNodeType(globalRelopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"
                                                  +"\n");
                              }
                    }
                    // int id relop other id
                    else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                              if(!getNodeType(token.lexeme).equals("integer")
                                                  && tokenList.getFirst().lexeme != "[") {
```

```
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // real id relop other num OR other id
                        else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                              if (!getNodeType(token.lexeme).equals("real")) {
                                if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        if(                     token.tokenType != 26
                                                && token.tokenType != 27
                                                &&
                                                ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                                ||
                                                (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                                ||
                                                (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                                ||
                                                (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("[")))

                                                ) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop needs ints/reals"
                                                        +"\n");
                        }
                        if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                                && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                + "\t"
                                +"\n");
                        }
                        set = token.tokenType;
                        simple_expression();
                }
                else if(token.tokenType == 1 && token.attribute == 3) { // relop
                        match(1, 3);
                        notFactorFlag = 0;
                        orBoolFlag = 0;
                        andBoolFlag = 0;
                        if(                     ((getNodeType(token.lexeme).equals("aint")
                                                || getNodeType(token.lexeme).equals("areal"))
                                                && tokenList.getFirst().lexeme != "["
                                                )
                        ) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\t< expects no array, token.lexeme: "
                                + token.lexeme + ", "
                                + "next token: " + tokenList.getFirst().lexeme
                                + "\t"
                                + globalRelopEqualsID + " " + globalRelopEqualsType
                                +"\n");
                        }
                        if(token.tokenType == 26) {     // real id relop int num
                                if(getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(token.tokenType == 27) {// int id relop real num
//                              debugWriter.printf("@@@@@@@@@\n");
                                if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // int id relop other id
                        else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                                if(!getNodeType(token.lexeme).equals("integer")
                                                && tokenList.getFirst().lexeme != "[") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // real id relop other num OR other id
                        else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                              if (!getNodeType(token.lexeme).equals("real")) {
```

```
                                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                if(                     token.tokenType != 26
                                                        && token.tokenType != 27
                                                        &&
                                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                                        ||
                                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                                        ||
                                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                                        ||
                                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("[")))

                                                        ) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tRelop needs ints/reals"
                                                                +"\n");
                                }
                                if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                                        && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tVar name not initialized " + token.lexeme
                                        + "\t"
                                        +"\n");
                                }
                                set = token.tokenType;
                                simple_expression();
                        }
                        else if(token.tokenType == 1 && token.attribute == 4) { // relop
                                match(1, 4);
                                notFactorFlag = 0;
                                orBoolFlag = 0;
                                andBoolFlag = 0;
                                if(                     ((getNodeType(token.lexeme).equals("aint")
                                                        || getNodeType(token.lexeme).equals("areal"))
                                                        && tokenList.getFirst().lexeme != "["
                                                        )
                                ) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\t<= expects no array, token.lexeme: "
                                        + token.lexeme + ", "
                                        + "next token: " + tokenList.getFirst().lexeme
                                        + "\t"
                                        + globalRelopEqualsID + " " + globalRelopEqualsType
                                        +"\n");
                                }
                                if(token.tokenType == 26) {     // real id relop int num
                                        if(getNodeType(globalRelopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                else if(token.tokenType == 27) {// int id relop real num
//                                      debugWriter.printf("@@@@@@@@@\n");
                                        if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                // int id relop other id
                                else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                                        if(!getNodeType(token.lexeme).equals("integer")
                                                                && tokenList.getFirst().lexeme != "[") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                // real id relop other num OR other id
                                else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                                      if (!getNodeType(token.lexeme).equals("real")) {
                                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                if(                     token.tokenType != 26
                                                        && token.tokenType != 27
```

```
                                        &&
                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                        ||
                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("[")))

                                        ) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tRelop needs ints/reals"
                                                +"\n");
                        }
                        if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                        && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                + "\t"
                                +"\n");
                        }
                        set = token.tokenType;
                        simple_expression();
                }
                else if(token.tokenType == 1 && token.attribute == 5) { // relop

//                              debugWriter.printf("Before: " + getNodeType(globalRelopEqualsID) + "\n");

                        match(1, 5);
                        if(((getNodeType(token.lexeme).equals("aint")
                                        || getNodeType(token.lexeme).equals("areal"))
                                        && tokenList.getFirst().lexeme != "["
                                        )
                        ) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\t>= expects no array, token.lexeme: "
                                + token.lexeme + ", "
                                + "next token: " + tokenList.getFirst().lexeme
                                + "\t"
                                + globalRelopEqualsID + " " + globalRelopEqualsType
                                +"\n");
                        }
                        // Comparing before and after relop
                        if(token.tokenType == 26) {     // real id relop int num
                                if(getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(token.tokenType == 27) {// int id relop real num
//                              debugWriter.printf("@@@@@@@@@\n");
                                if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // int id relop other id
                        else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                                if(!getNodeType(token.lexeme).equals("integer")
                                                && tokenList.getFirst().lexeme != "[") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // real id relop other num OR other id
                        else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                              if (!getNodeType(token.lexeme).equals("real")) {
                                if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tRelop expects same types "
                                                        + getNodeType(globalRelopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        if(                     token.tokenType != 26
                                        && token.tokenType != 27
                                        &&
                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                        ||
                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("[")))

                                        ) {
```

```
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                            + "SEMERR:\tRelop needs ints/reals"
                                                            +"\n");
                                }
                                notFactorFlag = 0;
                                orBoolFlag = 0;
                                andBoolFlag = 0;
                                if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                                    && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tVar name not initialized " + token.lexeme
                                        + "\t"
                                        +"\n");
                                }
//                                 TODO continued testing on this
//                                if(!nodeList.get(getNode(token.lexeme)).type.equals("integer")
//                                                    || !nodeList.get(getNode(token.lexeme)).type.equals("real")) {
//                                        // SEMERR
//                                        listingList.set(token.line-1, listingList.get(token.line-1)
//                                                            + "SEMERR:\tRelop needs int/real"
//                                                            + "\t"
//                                                            +"\n");
//                                }
//                                System.out.println(token.tokenType);

                                set = token.tokenType;
                                int inh = simple_expression();
                                // TODO semerrs
//                                for(int i = 0; i < 10; i++) {System.out.println(set + " " + inh);}
                        }
                        else if(token.tokenType == 1 && token.attribute == 6) { // relop
                                match(1, 6);
                                orBoolFlag = 0;
                                andBoolFlag = 0;
                                if(                     ((getNodeType(token.lexeme).equals("aint")
                                                        || getNodeType(token.lexeme).equals("areal"))
                                                        && tokenList.getFirst().lexeme != "["
                                                        )
                                ) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\t> expects no array, token.lexeme: "
                                        + token.lexeme + ", "
                                        + "next token: " + tokenList.getFirst().lexeme
                                        + "\t"
                                        + globalRelopEqualsID + " " + globalRelopEqualsType
                                        +"\n");
                                }
                                if(token.tokenType == 26) {     // real id relop int num
                                        if(getNodeType(globalRelopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                else if(token.tokenType == 27) {// int id relop real num
//                                        debugWriter.printf("@@@@@@@@@\n");
                                        if(!getNodeType(globalRelopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                // int id relop other id
                                else if((getNodeType(globalRelopEqualsID).equals("integer"))) {
                                        if(!getNodeType(token.lexeme).equals("integer")
                                                                && tokenList.getFirst().lexeme != "[") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                // real id relop other num OR other id
                                else if((getNodeType(globalRelopEqualsID).equals("real"))) {
//                                        if (!getNodeType(token.lexeme).equals("real")) {
                                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                        + "SEMERR:\tRelop expects same types "
                                                                        + getNodeType(globalRelopEqualsID) + " "
                                                                        + getNodeType(token.lexeme)
                                                                        + "\t"
                                                                        +"\n");
                                        }
                                }
                                if(                     token.tokenType != 26
                                                        && token.tokenType != 27
                                                        &&
                                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                                        ||
                                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))
                                                        ||
                                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
```

```
                                        ||
                                        (getNodeType(globalRelopEqualsID).equals("aint") && !globalRelopEqualsBracket.equals("["))

                                        ) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tRelop needs ints/reals"
                                                +"\n");
                        }
                        set = token.tokenType;
                        simple_expression();
                        notFactorFlag = 0;
                }
                else if(notFactorFlag == 1) {
                        // Means flag was set but no relop was found for a not (bool)
                        // Therefore semerr
                        notFactorFlag = 0;
                        listingList.set(token.line-1, listingList.get(token.line-1)
                        + "SEMERR:\tfactor -> not factor requires bool after not\n");
                }
                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
                }

                // For 3/4 - real
//              else if(token.tokenType == 27) {
//
//
//              }
//              // 3/4cor
//              else if(token.tokenType >= 0) {
//
//              }
                // For 3/4
//              else if(token.tokenType == 25) { // id
//                      // NoOp, epsilon
//              }
                else {
                        //
//                      System.out.println("SYNTAX ERROR: Expecting one of "
//                                      + "'relop', '), ']', 'do', 'then', ',', 'else', ';', "
//                                      + "'end' "
//                                      + " given " + token.lexeme);
                        listingList.set(token.line-1, listingList.get(token.line-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.attribute != 5)
                                        && (token.tokenType != 4 && token.attribute != 4)
                                        && (token.tokenType != 4 && token.attribute != 2)
                                        && (token.tokenType != 4 && token.attribute != 7)) {
                                token = tokenList.pop();
                        }
                }
                return set;
        }
        private static void writeToListing() {
                // Write to listing file
                for(int i = 0; i < listingList.size(); i++) {
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[9.")
                                        && !listingList.get(i-1).contains("[9..")){
                                listingWriter.printf("SEMERR:\t"
                                                + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[8.")
                                        && !listingList.get(i-1).contains("[8..")){
                                listingWriter.printf("SEMERR:\t"
                                                + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
```

```
                                        && listingList.get(i-1).contains("[7.")
                                        && !listingList.get(i-1).contains("[7..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[6.")
                                        && !listingList.get(i-1).contains("[6..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[5.")
                                        && !listingList.get(i-1).contains("[5..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[4.")
                                        && !listingList.get(i-1).contains("[4..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[3.")
                                        && !listingList.get(i-1).contains("[3..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[2.")
                                        && !listingList.get(i-1).contains("[2..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[1.")
                                        && !listingList.get(i-1).contains("[1..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("[0.")
                                        && !listingList.get(i-1).contains("[0..")){
                                listingWriter.printf("SEMERR:\t"
                                        + "Index array with non-int value\n");
                        }
                        if(i-1 > 0
                                        && listingList.get(i-1).contains("()")){
                                listingWriter.printf("SYNERR:\tExpecting one of id num ( not + - "
                                        + "Received )\t\n");
                        }
                        listingWriter.print(listingList.get(i));
                }
                // Print types
                for(int i = 0; i < nodeList.size(); i++) {
                        if(nodeList.get(i).color.equals("green")) {
                                for(int j = 0; j < nodeList.get(i).typeList.size(); j++) {

                                        debugWriter.printf(nodeList.get(i).id
                                                        + " requires "
                                                        + nodeList.get(i).typeList.get(j)
                                                        + "\n");
                                }
                        }
                }

//                      for(int i = 0; i < nodeList.size(); i++) {
//                              if(nodeList.get(i).color.equals("green")) {
//                                      for(int j = 1; j < nodeList.get(i-1).typeList.size(); j++) {
//                                              debugWriter.printf(nodeList.get(i-1).typeList.get(j) + "\n");
//                                      }
//                              }
//                      }
//                      for(int i = 0; i < nodeList.size(); i++) {
//                              for(int j = 0; j < nodeList.get(i).typeList.size(); j++) {
//                                      debugWriter.printf(nodeList.get(i).typeList.get(j) + "\n");
//                              }
//                      }
        }
        private static int 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
                        addopLeft = getNodeType(token.lexeme);
                        globalAddopEqualsBracket = tokenList.peek().lexeme;
                        globalAddopEqualsID = token.lexeme;
                        globalAddopEqualsType = getNodeType(token.lexeme);
                        term();
                        simple_expression_tail();
                }
                // These are NUMs
                else if(token.tokenType == 26 || token.tokenType == 27
                                || token.tokenType == 28) {     // integer, real, longreal
                        int ret = token.tokenType;
                        addopLeft = Integer.toString(token.tokenType);

                        if(token.tokenType == 26) {
                                globalAddopEqualsID = "26";
                        }
                        else if(token.tokenType == 27) {
```

```
                                        globalAddopEqualsID = "27";
                              }

                              globalAddopEqualsBracket = tokenList.peek().lexeme;
                              globalAddopEqualsType = getNodeType(token.lexeme);
                              term();
                              simple_expression_tail();
                              return ret;
                    }
                    else if(token.tokenType == 20) {// not
                              term();
                              simple_expression_tail();
                    }
                    else if(token.tokenType == 2 && token.attribute == 1) {         // +
                              sign();
//                            addopLeft = token.lexeme;
                              globalAddopEqualsBracket = tokenList.peek().lexeme;
                              globalAddopEqualsID = token.lexeme;
                              globalAddopEqualsType = getNodeType(token.lexeme);
                              term();
                              simple_expression_tail();
                    }
                    else if(token.tokenType == 2 && token.attribute == 2) {         // -
                              sign();
//                            addopLeft = token.lexeme;
                              globalAddopEqualsBracket = tokenList.peek().lexeme;
                              globalAddopEqualsID = token.lexeme;
                              globalAddopEqualsType = getNodeType(token.lexeme);
                              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.attribute != 5)
                                        && (token.tokenType != 4 && token.attribute != 4)
                                        && (token.tokenType != 4 && token.attribute != 2)
                                        && (token.tokenType != 4 && token.attribute != 7)
                                        && token.tokenType != 1) {
                                        token = tokenList.pop();
                              }
                    }
                    return token.tokenType;
          }

          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);         // 2 1
                                        if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                                  && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                                  // SEMERR
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tVar name not initialized " + token.lexeme
                                                  + "\t"
                                                  +"\n");
                                        }
//                                      if(addopLeft.equals("integer") && getNodeType(token.lexeme).equals("real")
//                                                || ((addopLeft.equals("real") && token.tokenType == 26))
//                                      ) {
//                                                listingList.set(token.line-1, listingList.get(token.line-1)
//                                                                    + "SEMERR:\tAddop requires no mixed mode "
//                                                                    + token.lexeme + " " + addopLeft
//                                                                    + "\t"
//                                                                    +"\n");
//                                      }
                                        if(token.tokenType == 26) {     // real id addop int num
                                                  if(getNodeType(globalAddopEqualsID).equals("real")) {
                                                            listingList.set(token.line-1, listingList.get(token.line-1)
                                                                      + "SEMERR:\tAddop expects same types "
                                                                      + getNodeType(globalAddopEqualsID) + " "
                                                                      + getNodeType(token.lexeme)
                                                                      + "\t"
                                                                      +"\n");
                                                  }
                                        }
                                        else if(token.tokenType == 27) {// int id relop real num, int num addop real num
//                                                debugWriter.printf("@@@@@@@@@\n");
                                                  if(!getNodeType(globalAddopEqualsID).equals("real")
                                                            && !getNodeType(globalAddopEqualsID).equals("areal")
                                                            && !globalAddopEqualsBracket.equals("[")) {
```

```
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " " + globalAddopEqualsID
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                        }
                                }
                                else if(token.tokenType == 26) {// real num addop int num
                                        if(!getNodeType(globalAddopEqualsID).equals("integer")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                        }
                                }
                                // int id relop other id
                                else if((getNodeType(globalAddopEqualsID).equals("integer"))) {
                                        if(!getNodeType(token.lexeme).equals("integer")
                                                && tokenList.getFirst().lexeme != "[") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                        }
                                }
                                // real id relop other num OR other id
                                else if((getNodeType(globalAddopEqualsID).equals("real"))) {
//                                      if (!getNodeType(token.lexeme).equals("real")) {
                                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + "d "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                        }
                                }
                                else if(globalAddopEqualsID == "26") {     // int num addop real id
                                        if(getNodeType(token.lexeme) == "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                        }
                                }
                                else if(globalAddopEqualsID == "27") {     // real num addop int id
                                        if(getNodeType(token.lexeme) == "integer") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                        }
                                }

                                if(getNodeType(globalAddopEqualsID) == "areal"        // areal access addop non-realnum
                                                && globalAddopEqualsBracket == "["
                                                && token.tokenType != 27) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                                if(token.tokenType == 26 && globalAddopEqualsID.equals("27")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                                if(token.tokenType == 26 && getNodeType(globalAddopEqualsID) == "areal"
                                                && globalAddopEqualsBracket == "[") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                                if(token.tokenType == 26) {
;                                       debugWriter.printf("Addop check:\t" + token.tokenType + " " + globalAddopEqualsID + "\n");

                                }
                                if(                     token.tokenType != 26
                                                        && token.tokenType != 27
                                                        &&
                                                        ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                                        ||
                                                        (getNodeType(globalAddopEqualsID).equals("aint") && !globalAddopEqualsBracket.equals("["))
                                                        ||
                                                        (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                                        ||
                                                        (getNodeType(globalAddopEqualsID).equals("aint") && !globalAddopEqualsBracket.equals("[")))
                                                        ) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop needs ints/reals"
```

```
                                                  +"\n");
                        }
//
                        term();
                }
                else if(token.attribute == 2) { // -
                        match(2, 2);
                        if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                        && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                + "\t"
                                +"\n");
                        }
                        if(token.tokenType == 26) {     // real id addop int num
                                if(getNodeType(globalAddopEqualsID).equals("real")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(token.tokenType == 27) {// int id relop real num, int num addop real num
//                              debugWriter.printf("@@@@@@@@@\n");
                                if(!getNodeType(globalAddopEqualsID).equals("real")
                                        && !getNodeType(globalAddopEqualsID).equals("areal")
                                        && !globalAddopEqualsBracket.equals("[")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " " + globalAddopEqualsID
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(token.tokenType == 26) {// real num addop int num
                                if(!getNodeType(globalAddopEqualsID).equals("integer")) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                        }
                        // int id relop other id
                        else if((getNodeType(globalAddopEqualsID).equals("integer"))) {
                                if(!getNodeType(token.lexeme).equals("integer")
                                        && tokenList.getFirst().lexeme != "[") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        // real id relop other num OR other id
                        else if((getNodeType(globalAddopEqualsID).equals("real"))) {
//                              if (!getNodeType(token.lexeme).equals("real")) {
                                if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + "d "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"
                                                        +"\n");
                                }
                        }
                        else if(globalAddopEqualsID == "26") {     // int num addop real id
                                if(getNodeType(token.lexeme) == "real") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                        }
                        else if(globalAddopEqualsID == "27") {     // real num addop int id
                                if(getNodeType(token.lexeme) == "integer") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tAddop expects same types "
                                                        + getNodeType(globalAddopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }
                        }

                        if(getNodeType(globalAddopEqualsID) == "areal"       // areal access addop non-realnum
                                        && globalAddopEqualsBracket == "["
                                        && token.tokenType != 27) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tAddop expects same types "
                                                + getNodeType(globalAddopEqualsID) + " "
                                                + getNodeType(token.lexeme)
                                                + "\t"+"\n");
                        }
                        if(token.tokenType == 26 && globalAddopEqualsID.equals("27")) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
```

```
                                                  + "SEMERR:\tAddop expects same types "
                                                  + getNodeType(globalAddopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                                        }
                                        if(token.tokenType == 26 && getNodeType(globalAddopEqualsID) == "areal"
                                                  && globalAddopEqualsBracket == "[") {
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tAddop expects same types "
                                                  + getNodeType(globalAddopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                                        }
                                        if(token.tokenType == 26) {
;                                                 debugWriter.printf("Addop check:\t" + token.tokenType + " " + globalAddopEqualsID + "\n");

                                        }
                                        if(                   token.tokenType != 26
                                                  && token.tokenType != 27
                                                  &&
                                                  ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                                  ||
                                                  (getNodeType(globalAddopEqualsID).equals("aint") && !globalAddopEqualsBracket.equals("["))
                                                  ||
                                                  (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                                  ||
                                                  (getNodeType(globalAddopEqualsID).equals("aint") && !globalAddopEqualsBracket.equals("[")))
                                                  ) {
                                                  // SEMERR
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tAddop needs ints/reals"
                                                  +"\n");
                                        }
//                                      if(addopLeft.equals("integer") && getNodeType(token.lexeme).equals("real")
//                                                  || ((addopLeft.equals("real") && token.tokenType == 26))
//                                                  ) {
//                                                  listingList.set(token.line-1, listingList.get(token.line-1)
//                                                            + "SEMERR:\tAddop requires no mixed mode "
//                                                            + token.lexeme + " " + addopLeft
//                                                            + "\t"
//                                                            +"\n");
//                                                  }
                                                  term();
                                        }
                                        else if (andBoolFlag == 1) {
                                                  listingList.set(token.line-2, listingList.get(token.line-2)
                                                  + "SEMERR:\tand requires a bool expression\n");
                                                  andBoolFlag = 0;
                                        }
                                        else if(token.attribute == 3) {
                                                  match(2, 3);
                                                  orBoolFlag = 1;
                                                  andBoolFlag = 0;
                                                  if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25
                                                            && !listingList.get(token.line-1).contains("Var name not initialized")) {
                                                  // SEMERR
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tVar name not initialized " + token.lexeme
                                                  + "\t"
                                                  +"\n");
                                                  }
                                                  term();
//                                                orBoolFlag = 0;
                                                  if(orBoolFlag == 1) { //|| orBoolFlag == 2) {
                                                  // Means flag was set but no bool after an 'or' was found
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tor requires a bool expression\n");
                                                  orBoolFlag = 0;
                                                  }
                                        }

//                                      if(orBoolFlag == 1) {
//                                                // Means flag was set but no bool after an 'or' was found
//                                                listingList.set(token.line-1, listingList.get(token.line-1)
//                                                            + "SEMERR:\tfactor -> not factor requires bool after not\n");
//                                        }

                              }
                              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
                              }
                              // 3/4cor
//                            else if(token.tokenType >= 0) {
//
//                            }

                              else {
                                        //
//                                      System.out.println("SYNTAX ERROR: Expecting one of "
//                                                  + "addop, relop, ), ], do, then, ,, else, ;, end"
//                                                  + " given " + token.lexeme + " smp_expr_tail");
//                                      listingList.set(token.line-1, listingList.get(token.line-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.attribute != 5)
                                                && (token.tokenType != 4 && token.attribute != 4)
                                                && (token.tokenType != 4 && token.attribute != 2)
                                                && (token.tokenType != 4 && token.attribute != 7)
                                                && token.tokenType != 1) {
                                        System.out.println(tokenList.peek().lexeme);
                                        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) {    // (
//                              String globalMulopEqualsID = "";
//                              String globalMulopEqualsType = "";
//                              String globalMulopEqualsBracket = "";
                                factor();
                                term_tail();
                        }
                        else if(token.tokenType == 26 || token.tokenType == 27
                                        || token.tokenType == 28) {     // integer, real, longreal
                                globalMulopEqualsBracket = tokenList.peek().lexeme;
                                globalMulopEqualsID = token.lexeme;
                                globalMulopEqualsType = getNodeType(token.lexeme);

                                factor();
                                term_tail();
                        }
                        else if(token.tokenType == 20) {// not
                                factor();
                                term_tail();
                        }
                        else if(token.tokenType == 25) {// id
//                              for(int i = 0; i < 10; i++) {System.out.println(token.lexeme);}
                                // TODO ERR after assignop AND before a mulop
                                // How to fix: get rid of after assignop?
                                if(!checkBlueNodePresence(token.lexeme)) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tVar name not initialized " + token.lexeme
                                                        + "\t"
                                                        +"\n");
                                }
                                globalMulopEqualsBracket = tokenList.peek().lexeme;
                                globalMulopEqualsID = token.lexeme;
                                globalMulopEqualsType = getNodeType(token.lexeme);
                                factor();
                                term_tail();
                        }
                        else {
//                              System.out.println("SYNTAX ERROR: Expecting one of ')', 'num'"
//                                              + ", 'not', 'id' "
//                                              + " given " + token.lexeme);
//                              listingList.set(token.line-1, listingList.get(token.line-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(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25) {
                                                // SEMERR
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tVar name not initialized " + token.lexeme
                                                + "\t"
```

```
                                        +"\n");
                                }
                                if(token.tokenType == 26) {     // real id addop int num
                                        if(getNodeType(globalMulopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + "a "
                                                                + getNodeType(token.lexeme)
                                                                + "\t" +"\n");
                                        }
                                }
                                if(token.tokenType == 26) {
                                        if(globalMulopEqualsType != "26" && getNodeType(globalMulopEqualsID) != "integer"
                                                        && !getNodeType(globalMulopEqualsID).equals("real")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"+"\n");
                                        }
                                }
                                else if(token.tokenType == 27) {// int id relop real num, int num addop real num
//                                      debugWriter.printf("@@@@@@@@@\n");
                                        if(!getNodeType(globalMulopEqualsID).equals("real")
                                                        && !getNodeType(globalMulopEqualsID).equals("areal")
                                                        && !globalMulopEqualsBracket.equals("[")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " " + globalMulopEqualsID
                                                                + getNodeType(token.lexeme)
                                                                + "\t"
                                                                +"\n");
                                        }
                                }
                                else if(token.tokenType == 26) {// real num Mulop int num
                                        if(!getNodeType(globalMulopEqualsID).equals("integer")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"+"\n");
                                        }
                                }
                                // int id relop other id
                                else if((getNodeType(globalMulopEqualsID).equals("integer"))) {
                                        if(getNodeType(token.lexeme).equals("real")) {
//                                                      && tokenList.getFirst().lexeme != "[") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"
                                                                +"\n");
                                        }
                                }
                                // real id relop other num OR other id
                                else if((getNodeType(globalMulopEqualsID).equals("real"))) {
//                                      if (!getNodeType(token.lexeme).equals("real")) {
                                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + "d "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"
                                                                +"\n");
                                        }
                                }
                                else if(globalMulopEqualsID == "26") {     // int num Mulop real id
                                        if(getNodeType(token.lexeme) == "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"+"\n");
                                        }
                                }
                                else if(globalMulopEqualsID == "27") {     // real num Mulop int id
                                        if(getNodeType(token.lexeme) == "integer") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
                                                                + "\t"+"\n");
                                        }
                                }

                                else if(getNodeType(globalMulopEqualsID) == "areal"   // areal access Mulop non-realnum
                                                && globalMulopEqualsBracket == "["
                                                && token.tokenType != 27) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tMulop expects same types "
                                                        + getNodeType(globalMulopEqualsID) + " "
                                                        + getNodeType(token.lexeme)
                                                        + "\t"+"\n");
                                }

                                else if(getNodeType(token.lexeme) == "real") {
                                        if(globalMulopEqualsType != "27" && getNodeType(globalMulopEqualsID) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tMulop expects same types "
                                                                + getNodeType(globalMulopEqualsID) + " "
                                                                + getNodeType(token.lexeme)
```

```
                                                + "\t"+"\n");
                        }
                }
                else if(getNodeType(token.lexeme) == "integer") {
                        if(globalMulopEqualsID != "26" && getNodeType(globalMulopEqualsID) != "integer") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + " "
                                                + getNodeType(token.lexeme)
                                                + "\t"+"\n");
                        }
                }
                debugWriter.printf("Mulop: " + getNodeType(globalMulopEqualsID) + "\n");
                if(token.tokenType != 26
                                && token.tokenType != 27
                                &&
                                ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                ||
                                (getNodeType(globalMulopEqualsID).equals("aint") && !globalMulopEqualsBracket.equals("["))
                                ||
                                (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                ||
                                (getNodeType(globalMulopEqualsID).equals("aint") && !globalMulopEqualsBracket.equals("[")))
                                ) {
                        // SEMERR
                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tMulop needs ints/reals"
                                        +"\n");
                }
                factor();
        }
        else if(token.attribute == 2) {
                match(3, 2);
                if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25) {
                        // SEMERR
                        listingList.set(token.line-1, listingList.get(token.line-1)
                        + "SEMERR:\tVar name not initialized " + token.lexeme
                        + "\t"
                        +"\n");
                }
                if(token.tokenType == 26) {     // real id addop int num
                        if(getNodeType(globalMulopEqualsID).equals("real")) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + "a "
                                                + getNodeType(token.lexeme)
                                                + "\t" +"\n");
                        }
                }
                if(token.tokenType == 26) {
                        if(globalMulopEqualsType != "26" && getNodeType(globalMulopEqualsID) != "integer"
                                        && !getNodeType(globalMulopEqualsID).equals("real")) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + " "
                                                + getNodeType(token.lexeme)
                                                + "\t"+"\n");
                        }
                }
                else if(token.tokenType == 27) {// int id relop real num, int num addop real num
//                      debugWriter.printf("@@@@@@@@@\n");
                        if(!getNodeType(globalMulopEqualsID).equals("real")
                                && !getNodeType(globalMulopEqualsID).equals("areal")
                                && !globalMulopEqualsBracket.equals("[")) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + " " + globalMulopEqualsID
                                                + getNodeType(token.lexeme)
                                                + "\t"
                                                +"\n");
                        }
                }
                else if(token.tokenType == 26) {// real num Mulop int num
                        if(!getNodeType(globalMulopEqualsID).equals("integer")) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + " "
                                                + getNodeType(token.lexeme)
                                                + "\t"+"\n");
                        }
                }
                // int id relop other id
                else if((getNodeType(globalMulopEqualsID).equals("integer"))) {
                        if(getNodeType(token.lexeme).equals("real")) {
//                              && tokenList.getFirst().lexeme != "[") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + " "
                                                + getNodeType(token.lexeme)
                                                + "\t"
                                                +"\n");
                        }
                }
                // real id relop other num OR other id
                else if((getNodeType(globalMulopEqualsID).equals("real"))) {
//                      if (!getNodeType(token.lexeme).equals("real")) {
                        if(token.tokenType != 27 && getNodeType(token.lexeme) != "real") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tMulop expects same types "
                                                + getNodeType(globalMulopEqualsID) + "d "
                                                + getNodeType(token.lexeme)
                                                + "\t"
```

```
                                                  +"\n");
                        }
                }
                else if(globalMulopEqualsID == "26") {     // int num Mulop real id
                        if(getNodeType(token.lexeme) == "real") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tMulop expects same types "
                                                  + getNodeType(globalMulopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                        }
                }
                else if(globalMulopEqualsID == "27") {     // real num Mulop int id
                        if(getNodeType(token.lexeme) == "integer") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tMulop expects same types "
                                                  + getNodeType(globalMulopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                        }
                }

                else if(getNodeType(globalMulopEqualsID) == "areal"   // areal access Mulop non-realnum
                                && globalMulopEqualsBracket == "["
                                && token.tokenType != 27) {
                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tMulop expects same types "
                                        + getNodeType(globalMulopEqualsID) + " "
                                        + getNodeType(token.lexeme)
                                        + "\t"+"\n");
                }

                else if(getNodeType(token.lexeme) == "real") {
                        if(globalMulopEqualsType != "27" && getNodeType(globalMulopEqualsID) != "real") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tMulop expects same types "
                                                  + getNodeType(globalMulopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                        }
                }
                else if(getNodeType(token.lexeme) == "integer") {
                        if(globalMulopEqualsID != "26" && getNodeType(globalMulopEqualsID) != "integer") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tMulop expects same types "
                                                  + getNodeType(globalMulopEqualsID) + " "
                                                  + getNodeType(token.lexeme)
                                                  + "\t"+"\n");
                        }
                }
                debugWriter.printf("Mulop: " + getNodeType(globalMulopEqualsID) + "\n");
                if(token.tokenType != 26
                                && token.tokenType != 27
                                &&
                                ((getNodeType(token.lexeme).equals("aint") && tokenList.getFirst().lexeme!="[")
                                ||
                                (getNodeType(globalMulopEqualsID).equals("aint") && !globalMulopEqualsBracket.equals("["))
                                ||
                                (getNodeType(token.lexeme).equals("areal") && tokenList.getFirst().lexeme!="[")
                                ||
                                (getNodeType(globalMulopEqualsID).equals("aint") && !globalMulopEqualsBracket.equals("[")))
                                ) {
                        // SEMERR
                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tMulop needs ints/reals"
                                        +"\n");
                }
                factor();
        }
        else if(token.attribute == 3) { // div - takes int only
                match(3, 3);
                if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25) {
                        // SEMERR
                        listingList.set(token.line-1, listingList.get(token.line-1)
                        + "SEMERR:\tVar name not initialized " + token.lexeme
                        + "\t"
                        +"\n");
                }
                // SEMERR
                if(((token.tokenType != 26 && !getNodeType(token.lexeme).equals("integer"))
//
//
                        || ((getNodeType(globalMulopEqualsID) != "integer") && globalMulopEqualsType != "26"))) {
////
                        && ((globalMulopEqualsBracket != "[") && tokenList.getFirst().lexeme != "[")) {       // int
////
                        for(int i = 0; i < 10; i++) {
////
                                System.out.println(token.tokenType);
////
                        }
//
                        // Real check
//
                        if(token.getTokenType() == 27) {
//
                                listingList.set(token.line-1, listingList.get(token.line-1)
//
                                                  + "SEMERR:\tdiv requires ints\n");
//
                        }
//
                        else {
//
                                listingList.set(token.line-1, listingList.get(token.line-1)
//
                                                  + "SEMERR:\tdiv requires ints "
//
                                                  +"\n");
//
                        }
//
                }
                if(token.tokenType == 26) {
                        if(globalMulopEqualsType != "26" && getNodeType(globalMulopEqualsID) != "integer" && globalMulopEqualsBracket
!= "[") {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                  + "SEMERR:\tdiv requires intsa" + globalMulopEqualsType
                                                  + "\n");
```

```
                                        }
                                }
                                else if(getNodeType(token.lexeme).equals("real")) {
                                        if(globalMulopEqualsType != "27" && getNodeType(globalMulopEqualsID) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tdiv requires intsb\n");
                                        }
                                }
                                else if(getNodeType(globalMulopEqualsID).equals("real")) {
                                        if(getNodeType(token.lexeme).equals("integer")) {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tdiv requires intsc\n");
                                        }
                                }
                                if(getNodeType(token.lexeme) == "aint" || getNodeType(token.lexeme) == "areal") {
                                        if(globalMulopEqualsBracket != "]" && globalMulopEqualsBracket != "mod") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tdiv requires ints"
                                                                + "\n");
                                        }
                                }
                                if(token.tokenType == 27 || globalMulopEqualsType == "27") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tdiv requires ints"
                                                        + "\n");
                                }
                                factor();
                        }
                        else if(token.attribute == 4) {
                                match(3, 4);
                                if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25) {
                                        // SEMERR
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tVar name not initialized " + token.lexeme
                                        + "\t"
                                        +"\n");
                                }
                                if(token.tokenType == 26) {
                                        if(globalMulopEqualsType == "27" || getNodeType(globalMulopEqualsID) == "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tmod requires intsaa" + globalMulopEqualsType
                                                                + "\n");
                                        }
                                }
                                else if(getNodeType(token.lexeme).equals("real")) {
                                        if(globalMulopEqualsType != "27" && getNodeType(globalMulopEqualsID) != "real") {
                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tmod requires intsb\n");
                                        }
                                }
                                else if(getNodeType(globalMulopEqualsID).equals("real")) {
                                        if(getNodeType(token.lexeme).equals("integer")) {

                                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                                + "SEMERR:\tmod requires intsc\n");

                                        }
                                }
                                if(token.tokenType == 27 || globalMulopEqualsType == "27") {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                                        + "SEMERR:\tmod requires ints"
                                                        + "\n");
                                }
                                factor();
                        }
                        else if(token.attribute == 5) {
                                match(3, 5);
                                andBoolFlag = 1;
                                orBoolFlag = 0;
//                              if(!checkBlueNodePresence(token.lexeme) && token.tokenType == 25) {
//                                      // SEMERR
//                                      listingList.set(token.line-1, listingList.get(token.line-1)
//                                      + "SEMERR:\tVar name not initialized " + token.lexeme
//                                      + "\t"
//                                      +"\n");
//                              }
                                factor();
                                if(andBoolFlag == 1) {
                                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SEMERR:\tand requires a bool expression\n");
                                        andBoolFlag = 0;
                                }

                        }
                }
                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-1, listingList.get(token.line-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

                                if(globalProcCallFactor == 1) {
//                                   debugWriter.printf("Given " + token.lexeme + ": "
//                                          + getNodeType(token.lexeme) + ", "
//                                          + " Next token: " + tokenList.getFirst().lexeme
//                                          + "\n");

                                          //SEMERR check for proc calls
                                          if(getNode(globalProcID) > -1
                                                     &&compareCounter < nodeList.get(getNode(globalProcID)).typeList.size()
                                                     ) {

                                          if(globalProcCallInternalID == 0) {
                                          if(

                                          // Integer given, check expected
                                          ((getNodeType(token.lexeme).equals("integer"))
                                          && ((int) nodeList.get(getNode(globalProcID)).typeList.get(compareCounter) != 13)
                                                     && (tokenList.peek().lexeme != "["))
                                          ||
                                          // Real given, check expected
                                          ((getNodeType(token.lexeme).equals("real"))
                                                     && ((int) nodeList.get(getNode(globalProcID)).typeList.get(compareCounter) != 16)
                                                     && (tokenList.peek().lexeme != "["))

                                          // Aint given, check expected
                                          ||
                                          ((getNodeType(token.lexeme).equals("aint"))
                                              && ((int) nodeList.get(getNode(globalProcID)).typeList.get(compareCounter) != 14)
                                                     && (tokenList.peek().lexeme != "["))

                                          ||
                                          ((getNodeType(token.lexeme).equals("areal"))
                                              && ((int) nodeList.get(getNode(globalProcID)).typeList.get(compareCounter) != 16)
                                                     && (tokenList.peek().lexeme != "["))

                                          ) {
                                                     debugWriter.printf("Error: " + getNodeType(token.lexeme) + " and "
                                                                          + nodeList.get(getNode(globalProcID)).typeList.get(compareCounter)
                                                                          + " do not match\t"
                                                                          + "typeList.get(compareCounter): " +
(nodeList.get(getNode(globalProcID)).typeList.get(compareCounter))
                                                                          + "\n");

                                                     listingList.set(token.line-1, listingList.get(token.line-1)
                                                                          + "SEMERR:\tArguments given " + getNodeType(token.lexeme) + " and "
                                                                          + nodeList.get(getNode(globalProcID)).typeList.get(compareCounter)
                                                                          + " do not match\t"
                                                                          + token.lexeme + " "
                                                                          + getNodeType(token.lexeme) + " "
//                                                                        + tokenList.getFirst().lexeme + "\t"
                                                                          +"\n");
                                          }
                                          }
                                          }
                                          compareCounter++;
                                          globalProcCallInternalID = 0;
//                                        compareCounter++;

                                }

                                callParams.add(getNodeType(token.lexeme));
                                if(token.lexeme.equals("e")) {
//                                   debugWriter.printf("\t\t" + tokenList.get(1).lexeme + "\n");
                                }
                                // SEMERR - Index array with wrong type
                                if((!(getNodeType(token.lexeme) == "integer")
                                          || !(token.tokenType == 25)
                                          || (getNodeType(token.lexeme).equals("areal"))
                                          || (getNodeType(token.lexeme).equals("aint"))
                                          )
                                          && tokenList.getFirst().lexeme.equals("]") // to check for array access
```

```
                                        && (!token.lexeme.equals("e"))// && tokenList.get(1).lexeme !=";")
                        ){
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tindex array with wrong type\t"
                                                + token.lexeme + " "
                                                + getNodeType(token.lexeme) + " "
//                                              + tokenList.getFirst().lexeme + "\t"
                                                +"\n");
                        }
//                      eCounter++;
                        if(getNodeType(token.lexeme) == "array"
                                        && equalFlag == 1
                                        && tokenList.getFirst().lexeme != "["
                                        ) {
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                + "SEMERR:\t= expects no array\t"
                                + token.lexeme + " " + getNodeType(token.lexeme)
                                + "\t"
//                              + globalRelopEqualsID + " " + globalRelopEqualsType
                                +"\n");
                                equalFlag = 0;
                        }

                        match(25, 0);         // id
                        factor_tail();



                }
                else if(token.tokenType == 26) {// int
                        callParams.add("integer");
                        match(26, 0);         // int
                }
                else if(token.tokenType == 28) {// longreal
                        callParams.add("longreal");
                        match(28, 0);         // longreal
                }
                else if(token.tokenType == 16) {// real
                        callParams.add("real");
                        match(27, 0);         // real
                }

                else if(token.tokenType == 20) {// not
                        match(20, 0);         // not
                        notFactorFlag = 1;
                        orBoolFlag = 0;
                        andBoolFlag = 0;
                        factor();
                        notFactorFlag = 0;
                }
                else if(token.tokenType == 4 && token.attribute == 3) {         // (
                        match(4, 3);          // (
                        expression();
                        match(4, 4);          // )
                }
                // added for 3/4
                else if(token.tokenType == 4 && token.attribute == 1) {         // [
                        globalAssignopArrayCheck = "true";
                        match(4, 1);          // [
                        expression();
                        match(4, 2);          // ]
                }
                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, 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) {    // [
                        globalAssignopArrayCheck = "true";
                        match(4, 1);          // [
                        // TODO this line
//                      globalProcCallFactor = 2;
                        globalProcCallInternalID = 1;
                        expression();
```

```
                        match(4, 2);          // ]
                }
                if(token.tokenType == 4 && token.attribute == 7
                                && tokenList.peek().tokenType == 26) {     // ,
                        match(4, 7);          // ,
                        match(26, 0);         // num
//                      match(4, 2);          // ]
//                      // SYNERR in this case
                        listingList.set(token.line-1, listingList.get(token.line-1)
                                        + "SYNERR: Expecting one of ], end Received ,\t"
                                        + "\t"+"\n");
                }
                if(token.tokenType == 4 && token.attribute == 2
                                && tokenList.peek().tokenType == 4) { // ]
//                      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-1, listingList.get(token.line-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.attribute != 5)
                                        && (token.tokenType != 4 && token.attribute != 4)
                                        && (token.tokenType != 4 && token.attribute != 2)
                                        && (token.tokenType != 4 && token.attribute != 7)
                                        && token.tokenType != 1
                                        && token.tokenType != 2
                                        && token.tokenType != 3) {
                                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-1, listingList.get(token.line-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 boolean checkAddGreenNode(String lexeme) {
                // How to check green node presence:
                // Evaluate stack contents that contain procedure names
                // If there is already an existing procedure with that name,
                // make it a SEMERR
                for(int i = 0; i < procStack.size(); i++) {
                        if(procStack.get(i).equals(lexeme)) {
                                // SEMERR
                                listingList.set(token.line-1, listingList.get(token.line-1)
                                                + "SEMERR:\tProcedure name already in use in scope"
                                                + ":\t"    + lexeme +"\n");
//                              tokenWriter.printf(lineCounter + "\t\t\t" + lexeme
//                                              + "\t\t99 (SEMERR)\t"
//                                              + "30 (PRCNMUSED)\t" + "\n");
                                return false;
                        }
```

```
                    }
                    return true;
                    // Check green node complete, now add green node
//                  nodeList.add(new Node(lexeme));
          }

          private static boolean checkAddBlueNode(String name) {
                    // How to check a blue node's presence:
                    // Keep in mind the scoping rules,
                    // Traverse the tree's previous nodes until you find
                    // the first green node.  Of all of the blue nodes, if there is
                    // no variable with the blue node being added's name, then there

                    for(int i = nodeListIndex-1; i > 0
                                        && !(nodeList.get(i).color.equals("green")); i--) {
                                        if(nodeList.get(i).id.equals(name)) {
                                                  // SEMERR
                                                  listingList.set(token.line-1, listingList.get(token.line-1)
                                                                      + "SEMERR:\tVar name already in use in scope"
                                                                      + ":\t"    + name +"\n");
//                                                tokenWriter.printf(lineCounter + "\t\t\t" + name
//                                                                    + "\t\t99 (SEMERR)\t"
//                                                                    + "31 (VARNMUSED)\t" + "\n");
                                                  return false;
                                        }
                    }
                    return true;
          }
          static String globalEndPrint = "";
          static String assignSecond = "";

          static String globalAssignopArrayCheck = "";
          static int globalProcCallInternalID = 0;
          static String globalArrayType = "";

          static String globalRelopEqualsID = "";
          static String globalRelopEqualsType = "";
          static String globalRelopEqualsBracket = "";

          static String globalAddopEqualsID = "";
          static String globalAddopEqualsType = "";
          static String globalAddopEqualsBracket = "";
          static String globalType = "";
          static String globalMulopEqualsID = "";
          static String globalMulopEqualsType = "";
          static String globalMulopEqualsBracket = "";

//        private static void addGreenNode(String id, int numParam) {
//                  nodeList.add(new Node(id, "green", numParam));
//                  presentNode.next = new Node(id, "green", numParam);
//                  presentNode = presentNode.next;
//        }

          // Check if proc name is in tree
          private static boolean checkGreenNodePresence(String idIn) {
                    for(int i = nodeListIndex; i > 0; i--) {
                              if(nodeList.get(i-1).id.equals(idIn)) {
                                        return true;
                              }
                    }
                    return false;
          }
          // Check if var name is in tree
          private static boolean checkBlueNodePresence(String idIn) {
                    for(int i = nodeListIndex; i > 0; i--) {
                              if(nodeList.get(i-1).id.equals(idIn)) {
                                        return true;
                              }
                    }
                    return false;
          }
}
```