CS 4013: Compiler Construction: Projects 3 & 4

Nate Beckemeyer

December 2016

Introduction

For Projects 3 and 4, I decorated the LL(1) grammar created in project 2 to the static semantics of our modified Pascal language. Then, using the synthesized and inherited attributes, I folded the decorations into the recursive descent parser.

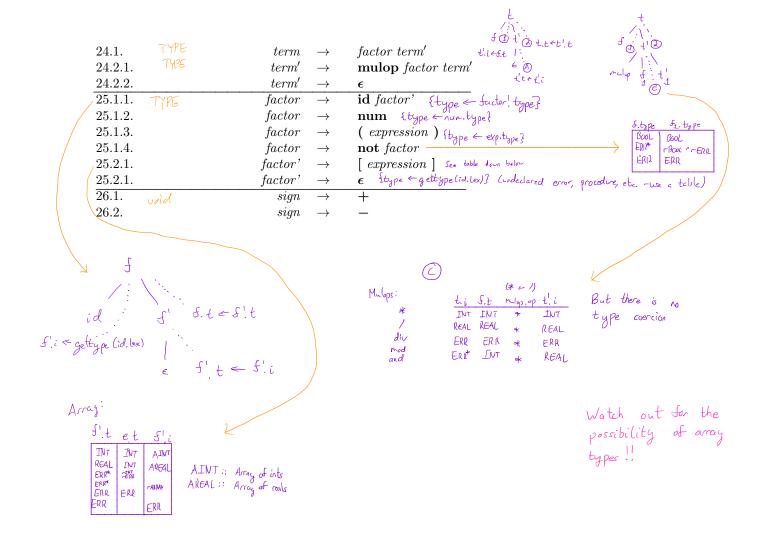
The compiler detects any lexical, syntax, and semantic errors that occur, and reports them in the listing file.

1 Methodology

By paying attention to when certain events should happen in the productions, I managed to modify the RDP to enforce the static semantics, such as type-checking. See the included L-Attributed Definition for more information.

```
true or succeed
Salse otherwise
Grammar with Decorations
                                                                                               check Add Green Node (id: lex, PGNNAME)
                                                            Global
                                                           Synthesized
                                                             inherited for other
                                                            semantic action)
                 1. Void
                                                                    program id ( identifier_list ) ;
                                                program
                                                                    declarations
                                                                    subprogram\_declarations
                                                                    compound\_statement
                                                                                                  >> chedc/Add Blue Nude (id: lex, P6 hNAMi)
                  2.1. void
                                           identifier\_list
                                                                    id identifier_list
                 2.2.1. Void
                                          identifier\_list'
                                                                    , id identifier_list'
                                                             \rightarrow
                 2.2.2.
                                          identifier\_list'
                                                                                                            > check/fdd Blue Node (:d: (ex, type.type)
                                                             \rightarrow
                                                                    var id: type; declarations
                  3.1.
                                            declarations
                        Void
                  3.2.
                                            declarations
                                                             \rightarrow
                                                                    standard_type > type < stdtype.type width & strwickth
                 4.1.
                                                     type
                                                             \rightarrow
                                                                                                                    type < AINT ; S INT
                                                                    array [ num .. num ] of standard_type
                 4.2.
                                                     type
                                                             \rightarrow
                                                                                                                                   AREAL IF REAL
                                                                                                                           width e(nun-nun+1) st. with
E12R if ERR,
                 5.1.
                                          standard\_type
                                                                    integer type < INT width < 4
                        TYPE
                                                                                                 width €8
                  5.2.
                                          standard\_type
                                                                               type - REAL
                                                                                                                              ERIL* otherise
                 6.1. vad
                                                                    subprogram_declaration ; subprogram_declarations
                              subprogram\_declarations
                                                             \rightarrow
                                                                                                                                  offset += width
                              subprogram\_declarations
                 6.2.
                                                             \rightarrow
                                                                                                                         pop From Green Stack
                                                                    subprogram_head declarations
                               subprogram\_declaration
                       void
                                                                    subprogram\_declarations\ compound\_statement
                                                                                                          > check Add Green Node (iv: lex, PROC)
                  8.
                        Void
                                      subprogram\_head
                                                                    procedure id arguments;
                                                             \rightarrow
                                                                                                            offset = 0
                 9.1. void
                                                                    ( parameter_list )
                                              arguments
                                                             \rightarrow
                 9.2.
                                              arguments
                                                                                                   -> checkAddBlue Nadelidillos 'pp"++ type
                  10.1. void
                                                                    id: type parameter_list'
                                         parameter\_list
                  10.2.1. void
                                                                    ; id : type parameter_list'
                                         parameter_list'
                                                             \rightarrow
                  10.2.2.
                                         parameter_list'
                  11. void
                                  compound\_statement
                                                                    begin
                                                                    optional\_statements
                                                                    end
                  12.1. void
                                   optional\_statements
                                                                    statement\_list
                  12.2.
                                   optional\_statements
                  13.1. vad
                                          statement\_list
                                                                    statement statement_list'
                  13.2.1. Void
                                         statement\_list'
                                                                    ; statement statement_list'
                  13.2.2.
                                         statement\_list'
                  14.1.
                                              statement
                                                                    variable assignop expression
                  14.2.
                                                                    procedure\_statement
                                              statement
                  14.3.
                                                                    compound\_statement
                                              statement
                  14.4.
                                              statement
                                                                    while expression do statement
                                                                                                              Check type 1300L
                  14.5.
                                              statement
                                                                    if expression then statement else'
                  15.1.
                           void
                                                     else'
                                                                    else statement
                                                     else'
                  15.2.
                                                             \rightarrow
                                                                    \epsilon
                  16.
                                                variable
                                                                    id array_access
                                                             \rightarrow
                                                                                           Same as factor - id factor
                  17.1.
                                            array\_access
                                                             \rightarrow
                                                                    [ expression ]
                  17.2.
                                            array\_access
                                                                                                      > Must exist & be a procedure points to the first item
                                                                    call id optional_expressions
                                   procedure\_statement
                        void
                                                                    ( expression_list ) ~ ( ~
                  19.1. void
                                  optional\_expressions
                                                            \rightarrow
                  19.2.
                                  optional\_expressions
                 20.1. Void
                                         expression\_list
                                                                    expression expression_list'
                                                                                                        i < points to the right type - iI natch, continue; else, Suil
                 20.2.1. void
                                                                    , expression\_list'
                                         expression\_list'
                                                             \rightarrow
                  20.2.2.
                                         expression\_list'
                  \overline{21}.
                                              expression
                                                                    simple\_expression\ related\_expression
                  22.1. TY (C
                                                                                                              Looks exactly like the table
                                     related\_expression
                                                                    relop simple_expression
                                                                                                              for * and / for factor !
                  22.2.
                                     related\_expression
                  23.1.1. TYPE
                                      simple\_expression
                                                                    term simple_expression'
                 23.1.2.
                                     simple\_expression
                                                            \rightarrow
                                                                    sign\ term\ simple\_expression'
                 23.2.1. TYPE
                                                                    addop term simple_expression'
                                     simple\_expression'
                 23.2.2.
                                     simple\_expression'
```

or



2 Implementation

I merely modified the productions to enforce the rules, according to the decorated grammar given above.

The declarations processing was interesting—I used a binary tree with left-child, right-sibling notation. I a pointer to the bottom of the tree at all times. Whenever anything was added to the tree, I updated the bottom pointer to point to the new data. Then, whenever a new scope was declared, I added a pointer to that tree node to the stack; whenever the new scope ended, I set the bottom pointer to the value popped from the stack, and set a flag to add to the right of the child.

If any errors were encountered while parsing, the error is added to the error queue. Then, the error is printed before the next token is collected. In this situation, multiple semantic errors could happen on the same token, so I had to create a separate message for each one and add it to a queue.

3 Discussion & Conclusions

Implementing this project definitely taught me about the importance of an LL(1) grammar, and how neat the recursive descent parser is. Decorating the grammar is a really cool way of implementing the compiler.

I wrote this compiler in C, with no external code of any kind. It was compiled with clang on macOS Sierra.

Appendix 1: Sample Inputs and Outputs

3.1 Error-Filled

Listing 1: Error-Full Source Code

```
program fib(input; output):
     var a: int; var p: integer;
     var numsArray : array [6..12] on integer;
     var q: real;
     procedure fib1(aReallyLongInt : integer; b : real, c
          : real);
       begin
            if a \leq 1.20 then fib := c
            else call fib (a - 01, c, b + c)
9
       end;
11
     procedure fib2(a : integer);
       var b : real; var c : real; var sum : ;
13
       var b : real;
       procedure rawr3(b : real);
         var q : real;
16
         begin
17
           q := b + 2.0;
18
           call fib2(q).
         end;
20
21
       begin
22
         a := a - 1;
         fib1(3.00);
24
         sum := 1;
         c := b;
26
         while a > 0) do
27
           call 3;
28
           begin
              a := a - 1;
30
             b := sum;
31
              sum := c + sum;
32
              c := b
33
           end;
34
         fib2 := sum
35
       end;
36
37
     procedure init;
       begin
39
```

```
n := 12;
40
         if (1 and 2) or 3 then p := 12
41
          else p := 14;
42
         numsArray[3] := 15.560;
         q := q[4];
44
         q[4] := 12
       end;
46
47
       begin*
48
         call init;
         call fib2;
50
         call rawr3(34, 56);
51
       end.
52
53
54 a
```

Listing 2: Error-Full Listing File

```
1 program fib(input; output):
  SYNERR: Found ';'; expected ')', ',' instead.
  SYNERR: Found ':'; expected ';' instead.
        var a: int; var p: integer;
  SYNERR: Found 'ID'; expected 'array', 'real', 'integer
      ' instead.
  SYNERR: Found 'integer'; expected ';' instead.
  SYNERR: Found ';'; expected 'begin', 'procedure', 'var
      ' instead.
            var numsArray : array [6..12] on integer;
        4
            var q: real;
        5
            procedure fib1(aReallyLongInt : integer; b :
             real, c : real);
  LEXERR:
                           ID length exceeded 10
                       aReallyLongInt
      characters:
  SYNERR: Found ','; expected ')', ';' instead.
13
        7
           begin
14
        8
                   if a \leq= 1.20 then fib := c
  SEMERR: No variable 'a' is defined in the local scope!
  LEXERR:
                                         Trailing 0 in
      real:
                           1.20
  SEMERR: Cannot assign to ID 'fib' of type 'PROGRAM'!
                   else call fib (a - 01, c, b + c)
  SEMERR: No variable ' ' is defined in the local scope!
  SEMERR: Procedure 'fib' not in scope!
22 SEMERR: No variable 'a' is defined in the local scope!
LEXERR:
                                           Leading 0 in
```

```
int:
                              01
  SEMERR: No variable 'c' is defined in the local scope!
   SEMERR: No variable 'c' is defined in the local scope!
        10
               end;
        11
27
        12
             procedure fib2(a : integer);
                var b : real; var c : real; var sum : ;
        13
29
   SYNERR: Found ';'; expected 'array', 'real', 'integer'
       instead.
        14
               var b : real;
  SYNERR: Found 'real'; expected ';' instead.
  SYNERR: Found ';'; expected 'begin', 'procedure', 'var
      ' instead.
        15
               procedure rawr3(b : real);
34
        16
                  var q : real;
35
        17
                  begin
36
        18
                    q := b + 2.0;
        19
                    call fib2(q).
  SEMERR: Expected type INT, not REAL!
   SYNERR: Found '.'; expected 'end', ';' instead.
40
        20
                  end;
        21
42
        22
               begin
        23
                  a := a - 1;
44
        24
                  fib1(3.00);
  SYNERR: Found '('; expected '[', ':=' instead.
  LEXERR:
                                            Trailing 0 in
                             3.00
      real:
        25
                  sum := 1;
        26
                  c := b;
49
        27
                  while a > 0) do
50
  SYNERR: Found ')'; expected 'do' instead.
   SYNERR: Found 'do'; expected 'if', 'while', 'begin', '
      call', 'ID' instead.
        28
                    call 3;
53
        29
                    begin
54
        30
                      a := a - 1;
55
                      b := sum;
        31
   SEMERR: No variable 'sum' is defined in the local
57
      scope!
        32
                      sum := c + sum;
58
  SEMERR: ID 'sum' not in scope!
   SEMERR: No variable 'sum' is defined in the local
      scope!
        33
                      c := b
61
        34
                    end;
```

```
fib2 := sum
  SEMERR: Cannot assign to ID 'fib2' of type 'PROCEDURE
      '!
        36
                end;
   SEMERR: No variable 'sum' is defined in the local
66
      scope!
        37
67
        38
             procedure init;
68
        39
               begin
69
        40
                 n := 12;
   SEMERR: ID 'n' not in scope!
71
                if (1 \text{ and } 2) or 3 \text{ then } p := 12
        41
   SEMERR: Expected BOOL and BOOL for use with 'and',
      received INT and INT!
   SEMERR: ID 'p' not in scope!
                 else p := 14;
75
  SEMERR: ID 'p' not in scope!
        43
                 numsArray[3] := 15.560;
77
  LEXERR:
                                            Trailing 0 in
      real:
                          15.560
  SEMERR: ID 'numsArray' not in scope!
                  q := q[4];
80
  SEMERR: ID 'q' not in scope!
   SEMERR: No variable 'q' is defined in the local scope!
                q[4] := 12
83
   SEMERR: ID 'q' not in scope!
84
        46
                end;
        47
86
        48
                begin*
   SYNERR: Found '*'; expected 'array', 'end', 'if', '
      while', 'begin', 'call', 'ID' instead.
        49
                  call init;
89
        50
                 call fib2;
90
        51
                  call rawr3(34, 56);
91
        52
                end.
92
        53
        54 a
94
  SYNERR: Found 'ID'; expected 'EOF' instead.
```

Listing 3: Error-Full Semantic Mem File

1	ID	Memory Offset
2	a	0
3	b	0
4	С	8
5	sum	16

q 0

Listing 4: Error-Full Token File

```
FILEEND
1 1
         ASSIGNOP
         RELOP
   3
         ID
   5
         CONTROL
   6
         ADDOP
   7
         MULOP
   8
         WS
   9
         ARRAY
   10
         TYPE
         VAR
   11
   12
         NUM
         PUNC
   13
   14
         GROUP
   15
         INVERSE
   16
         LEXERR
   17
         SYNERR
   18
         SEMERR
                                                Token Type
         LineLexeme
                           Token Attribute
19
     1
                           5
                                  7
             program
20
                           0x7ff6d6600fd0
     1
          fib
                   4
21
                  14
     1
                           0x7ff6d66012e0
     1 input
                  4
23
     1
                  13
     1
                  17
25
     1output
                           0x7ff6d6601760
                   4
            )
                  14
     1
27
                          0
                  10
     1
     1
                  17
                          0
29
     2
                  11
          var
     2
                   4
                           0x7ff6d6601f00
            а
     2
                  10
32
     2
                           0x7ff6d66021f0
          int
                   4
     2
                  17
          int
34
     2
                  13
                          1
     2
                  11
          var
36
     2
                           0x7ff6d6602780
                   4
            р
37
     2
                  10
38
     2
            integer
                          10
                                  1
     2
            integer
                          17
40
     2
                  13
                          1
     2
                  17
            ;
42
```

```
var 11
                    0
43
                         0x7ff6d6700a60
    3
      numsArray
                    4
        :
             10
45
              9
    3 array
                    0
    3
      [
             14
                    2
47
    3
        6
             12
                    0
    3
             9
        . .
49
      12
             12
    3
50
    3
        ]
              14
51
             4
    3
       on
                    0x7ff6d67016c0
       integer
; 13
    3
                   10 1
53
    3
54
             11
    4
      var
55
             4
                    0x7ff6d6701eb0
        q
56
             10
    4
        :
              10
      real
58
    4
              13
                    1
59
                   5 6
        procedure
60
                    0x7ff6d67028f0
      fib1
             4
    6
       (
                    0
    6
              14
62
        {\tt aReallyLongInt}
                          4
    6
                                0x7ff6d6702d00
         aReallyLongInt 16 1
    6
64
                    0
    6
             10
65
    6
         integer
                    10 1
66
        ; 13
                   1
67
                    0x7ff6d6703490
    6
        b
             4
68
             10
                    0
    6
      real
    6
              10
70
              13
                    0
71
              17
                    0
    6
72
                    0x7ff6d6703bf0
    6
         С
             4
73
    6
         :
              10
74
              10
      real
75
              14
    6
      )
                    1
76
              13
                    1
77
    7 begin
              5
                    0
      if
             5
79
             4
                   0x7ff6d6601f00
    8
        a
             3
    8
       <=
81
    8
      18
              0
    8
      1.20
             12
83
              16
    8
      1.20
    8
      then
              5
85
      fib
             4
                    0x7ff6d6600fd0
    8
    8
       :=
              2
87
    8
        С
             4 0x7ff6d6703bf0
```

```
18
89
                   5
         else
                         2
90
         18
                  0
91
                  5
                         10
      9
         call
      9
          fib
                   4
                          0x7ff6d6600fd0
93
      9
                  14
           (
94
      9
          18
                  0
95
          a
-
                  4
                           0x7ff6d6601f00
      9
96
      9
                    6
97
      9
          18
                  0
      9
          01
                  12
                           0
99
                           7
      9
           01
                  16
100
      9
                  13
101
      9
                   4
                            0x7ff6d6703bf0
102
      9
                  13
103
104
      9
          18
                  4
                            0x7ff6d6703490
      9
           b
105
      9
106
                            0x7ff6d6703bf0
      9
                  4
             С
107
      9
                  14
108
      9
          18
                  0
     10
          end
                   5
110
     10
                   13
                           1
111
           ;
                           5 6
     12
           procedure
112
                  4
                            0x7ff6d67088f0
113
     12
         fib2
     12
           (
                  14
114
     12
                  4
                            0x7ff6d6601f00
             a
     12
                  10
                           0
116
     12
             integer
                           10 1
117
           )
                  14
     12
                           1
118
     12
           ;
                  13
                           1
119
     13
                  11
         var
120
                  4
     13
           b
                            0x7ff6d6703490
121
                  10
     13
122
                  10
                           2
     13
         real
123
     13
                  13
                           1
124
     13
         var
                  11
125
                            0x7ff6d6703bf0
     13
                   4
             С
     13
                   10
127
     13
         real
                   10
128
     13
                  13
129
     13
                  11
130
          var
    13
         sum
                   4
                           0x7ff6d670a770
131
                  10
                           0
     13
           :
                  13
     13
                           1
133
     13
                  17
                           0
134
```

```
14
                 11
        var
135
        b
                         0x7ff6d6703490
    14
                 4
136
                 10
137
                 10
                        2
    14 real
    14
        real
                 17
                        0
139
    14
                 13
                        1
140
    14
                 17
141
                        5 6
    15
           procedure
142
    15 rawr3
                4
                         0x7ff6d670bf00
143
    15
        (
                 14
    15
          b
                 4
                         0x7ff6d6703490
145
                 10
    15
146
    15
                 10
                        2
       real
147
    15
         )
                 14
                        1
148
    15
                 13
                        1
149
    16
150
         var
                 11
                         0x7ff6d6701eb0
    16
                 4
        q
151
                 10
    16
152
    16
                 10
                        2
153
       real
    16
                 13
154
    17 begin
                 5
155
                         0x7ff6d6701eb0
    18
        q
156
    18
          :=
                  2
157
         b
    18
                 4
                        0x7ff6d6703490
158
                 6
    18
159
    18
        2.0
                 12
160
                 13
    18
                        1
    19
        call
                 5
                       10
162
                         0x7ff6d67088f0
    19
        fib2
                 4
163
          (
                 14
    19
164
    19
           q
                 4
                        0x7ff6d6701eb0
165
    19
          )
                 14
166
    19
        18
                 0
167
                 13
                        2
    19
168
                 17
                        0
    19
169
                        3
    20
        end
                  5
170
        ;
    20
                 13
171
    22 begin
                 5
172
    23
                  4
                         0x7ff6d6601f00
173
        a
          :=
                  2
    23
    23
                4
                         0x7ff6d6601f00
          a
175
    23
                  6
176
    23
          1
                 12
177
    23
                 13
    24
       fib1
                 4
                         0x7ff6d67028f0
179
    24
        (
                 14
180
```

```
24
         (
                   17
181
         3.00
     24
                   12
                           0
182
         3.00
                   16
183
                   14
     24
             )
                           1
     24
                   13
185
     25
                   4
                            0x7ff6d670a770
         sum
186
     25
           :=
                   2
187
                           0
     25
            1
                   12
188
     25
                   13
189
     26
                   4
                            0x7ff6d6703bf0
190
            С
     26
            :=
                    2
191
                            0x7ff6d6703490
     26
             b
                    4
192
                   13
     26
                           1
193
     27 while
                    5
194
                            0x7ff6d6601f00
     27
             a
                    4
195
     27
                    3
             >
196
             0
     27
                   12
                           0
197
     27
             )
                   14
198
            )
     27
                   17
                           0
199
     27
           do
                   5
200
     27
            do
                   17
                           0
     28
         call
                   5
                          10
202
     28
                   12
             3
203
     28
                   13
204
     29 begin
                    5
205
     30
                    4
                             0x7ff6d6601f00
206
             a
                    2
     30
            :=
     30
                    4
                            0x7ff6d6601f00
             a
208
     30
                    6
                           1
209
     30
            1
                   12
210
     30
             ;
                   13
211
     31
                    4
                            0x7ff6d6703490
            b
212
     31
                    2
           :=
213
                             0x7ff6d670a770
     31
                    4
         sum
214
     31
                   13
215
     31
         18
                   0
216
     32
          sum
                    4
                             0x7ff6d670a770
217
     32
                    2
           :=
218
                             0x7ff6d6703bf0
     32
           С
219
     32
           18
                   0
     32
                    6
221
                            0x7ff6d670a770
     32
                    4
222
           sum
     32
                   13
223
     32
         18
                   0
     33
                    4
                            0x7ff6d6703bf0
           С
225
     33
           :=
                    2
                           0
226
```

```
Ъ
    33
                       0x7ff6d6703490
227
         end
    34
               5
228
    34
        ;
               13
229
    35
                       0x7ff6d67088f0
       fib2
               4
230
    35
        :=
                2
231
    35
               4
                      0x7ff6d670a770
232
       sum
    35
       18
               0
233
    36
               5
       end
234
       18
    36
235
    36
       ;
               13
                       1
                      5 6
         procedure
    38
237
                       0x7ff6d671a030
    38
       init
               4
238
               13
    38
239
               5
240
    39 begin
    40
        n
               4
                      0x7ff6d671aae0
^{241}
               2
242
    40
         :=
    40
         12
               12
                       0
243
    40
               0
       18
244
               13
    40
                       1
^{245}
        if
    41
               5
                       5
246
         (
    41
               14
                       0
247
    41
         1
               12
248
    41
               7
                       2
       and
249
       2
    41
               12
250
    41
               14
251
    41
       18
252
                6
                       2
    41
        or
    41
         3
               12
                       0
254
    41
       then
               5
255
               4
                      0x7ff6d6602780
    41
         р
256
         :=
               2
    41
257
    41
         12
               12
                      0
258
    41
       18
               0
259
               5
    42
       else
260
    42
               4
                      0x7ff6d6602780
        p
261
               2
                       0
    42
        :=
262
        14
    42
               12
                       0
263
    42
       18
               0
264
        ;
    42
               13
                       1
265
                              0x7ff6d6700a60
    43
        numsArray
                      4
    43
         [
               14
                       2
267
          3
               12
    43
                       0
268
          ]
    43
               14
                       3
269
        :=
               2
    43
                       0
    4315.560
               12
                       0
271
  4315.560
             16
272
```

```
43
           18
                    0
273
     43
                    13
                             1
274
                              0x7ff6d6701eb0
275
             q
                     2
     44
            : =
     44
                     4
                              0x7ff6d6701eb0
              q
277
     44
           18
                    0
278
     44
              14
279
     44
                    0
           18
280
     44
              4
                    12
281
                    14
                             3
     44
              ]
     44
                    13
283
     45
                     4
                              0x7ff6d6701eb0
              q
284
     45
              [
                    14
                             2
285
     45
              4
                    12
                             0
286
             ]
                    14
                             3
     45
287
     45
                    2
            :=
                             0
288
     45
            12
                    12
                             0
289
     45
                    0
           18
290
                     5
                             3
     46
           end
291
     46
                    13
                             1
292
     48 begin
                     5
                             0
                     7
     48
294
     48
                    17
                             0
295
     49
          call
                     5
                            10
296
     49
                     4
                              0x7ff6d671a030
          init
297
     49
                    13
298
     50
                     5
                            10
          call
     50
          fib2
                     4
                              0x7ff6d67088f0
300
     50
                    13
                            1
301
     51
         call
                     5
                            10
302
                              0x7ff6d670bf00
     51 rawr3
                     4
303
     51
             (
                    14
304
     51
             34
                    12
                             0
305
                    13
                             0
     51
306
     51
             56
                    12
                             0
307
            )
     51
                    14
                             1
308
     51
                    13
309
     52
                    5
                             3
           end
310
     52
                    13
311
                              0x7ff6d6601f00
     54
                     4
312
              а
     54
                    17
                             0
            a
313
                             0
     55
           EOF
                     1
314
```

3.2 Just Semantic Errors

Listing 5: Just Semantic Source Code

```
program fib(input, output);
     var a: integer; var p: integer;
     var numsArray : array [6..12] of integer;
     var q: real;
     procedure fib1(aLongInt : integer; b : real;
                     c : integer; d: integer);
       begin
            if a <= 1 then fib := c
            else call fib (a - 1, c, b + c)
10
       end;
12
     procedure fib2(a : integer);
13
       var b : real; var c : real; var sum : integer;
14
       var b : real;
15
       procedure rawr3(b : real);
16
         var q : real;
         begin
            q := b + sum;
19
            call fib2(q)
20
         end;
21
22
       begin
23
         q := not 3;
         q := (3 < 4) \text{ and } (3.6 < p);
25
         a := a - 1;
         call fib1(3.2, 1);
27
         sum := 1;
         c := b;
29
         while not (a > 0) do
30
            call rawr3(b * 4);
31
         begin
32
           a := a - 1;
33
            b := sum;
34
            sum := c + sum;
35
            c := b
36
         end;
37
         fib2 := sum
38
       end;
39
40
     procedure init;
       begin
42
```

```
n := 12;
43
         if not (1 < 74) or 3 then p:=(1 > 2) and (3 < n)
44
            else p:=not 2;
45
         numsArray[3] := 15.56;
         q := q[4];
47
         q[4] := 12
       end;
49
50
     begin
51
       call init;
       call fib2;
53
       call rawr3(34, 56)
     end.
```

Listing 6: Just Semantic Listing File

```
1 program fib(input, output);
         2
             var a: integer; var p: integer;
         3
             var numsArray : array [6..12] of integer;
         4
             var q: real;
         5
         6
             procedure fib1(aLongInt : integer; b : real;
         7
                            c : integer; d: integer);
         8
               begin
                   if a \leq 1 then fib := c
  SEMERR: Cannot assign to ID 'fib' of type 'PROGRAM'!
                   else call fib (a - 1, c, b + c)
11
  SEMERR: Procedure 'fib' not in scope!
  SEMERR: Attempt to add incompatible types REAL and INT
        11
               end;
14
        12
        13
             procedure fib2(a : integer);
16
        14
               var b : real; var c : real; var sum :
           integer;
        15
               var b : real;
18
  SEMERR: A variable named 'b' is already defined in the
       local scope!
               procedure rawr3(b : real);
        16
20
        17
                 var q : real;
21
        18
                 begin
        19
                   q := b + sum;
  SEMERR: Attempt to add incompatible types REAL and INT
                   call fib2(q)
        20
  SEMERR: Expected type INT, not REAL!
```

```
21
                  end;
27
        22
        23
                begin
29
        24
                  q := not 3;
   SEMERR: Expected BOOL use with 'not', received INT!
31
                  q := (3 < 4) \text{ and } (3.6 < p);
32
   SEMERR: Attempt to compare incompatible types REAL and
33
       INT!
        26
                  a := a - 1;
34
        27
                  call fib1(3.2, 1);
   SEMERR: Expected type INT, not REAL!
36
   SEMERR: Expected type REAL, not INT!
   SEMERR: Expected INT, not the end of the parameters!
        28
                  sum := 1;
39
        29
                  c := b;
40
        30
                  while not (a > 0) do
41
        31
                    call rawr3(b * 4);
42
   SEMERR: Attempt to multiply or divide incompatible
43
      types REAL and INT!
        32
                  begin
44
        33
                    a := a - 1;
45
                    b := sum;
46
   SEMERR: Attempt to convert type REAL into type INT in
      assignment!
        35
                    sum := c + sum;
48
   SEMERR: Attempt to add incompatible types REAL and INT
49
        36
                    c := b
50
        37
                  end;
51
        38
                  fib2 := sum
52
   SEMERR: Cannot assign to ID 'fib2' of type 'PROCEDURE
53
      '!
        39
                end;
54
        40
55
        41
              procedure init;
56
        42
                begin
        43
                  n := 12;
58
   SEMERR: ID 'n' not in scope!
                  if not (1 < 74) or 3 then p := (1 > 2) and
60
            (3 < n)
   SEMERR: Expected BOOL and BOOL for use with 'or',
61
      received BOOL and INT!
   SEMERR: No variable 'n' is defined in the local scope!
62
        45
                    else p:=not 2;
   SEMERR: Expected BOOL use with 'not', received INT!
        46
                  numsArray[3] := 15.56;
```

```
SEMERR: Attempt to convert type INT into type REAL in
      assignment!
        47
                 q := q[4];
67
  SEMERR: Attempt to index variable of type REAL!
        48
                 q[4] := 12
69
   SEMERR: Attempt to index variable of type REAL!
        49
               end;
71
        50
72
        51
             begin
73
        52
                call init;
        53
               call fib2;
  SEMERR: Expected an argument of type INT!
76
               call rawr3(34, 56)
        54
77
   SEMERR: Procedure 'rawr3' not in scope!
78
        55
             end.
```

Listing 7: Just Semantic Mem File

		<u> </u>	
1	ID	Memory Offset	
2	a	0	
3	р	4	
4	numsArray	8	
5	q	36	
6	b	0	
7	С	8	
8	sum	16	
9	q	0	

Listing 8: Just Semantic Token File

```
1
      FILEEND
2
      ASSIGNOP
3
      RELOP
4
      ID
5
      CONTROL
6
      ADDOP
7
      MULOP
8
      WS
9
      ARRAY
      TYPE
10
11
      VAR
12
      NUM
      PUNC
13
      GROUP
14
15
      INVERSE
16
      LEXERR
```

```
17 SYNERR
  18
     SEMERR
                    Token Attribute Token Type
      LineLexeme
       program
    1
      fib 4
                    0x7fba52500fd0
21
    1 (
             14
             4
                     0x7fba525012e0
    1 input
23
             13
^{24}
    1output
             4
                     0x7fba52501660
25
      )
             14
             13
    2
             11
      var
28
    2
       a
              4
                    0x7fba52501d30
29
             10
        :
30
                    10 1
    2
         integer
31
       ; 13
    2
32
             11
    2
      var
             4
                     0x7fba52502530
       p
34
            10
    2
        :
                    0
    2
         integer
                    10 1
36
    2
             13
                    1
            11
                    0
      var
38
                    4
                         0x7fba52502f50
    3
        numsArray
39
    3
        :
             10
40
             9
    3 array
                    0
41
      [
             14
42
        6
             12
                    0
    3
    3
              9
       . .
                    1
44
    3
      12
             12
45
        ]
             14
                    3
    3
46
    3
        of
             9
                    2
47
       integer
    3
                    10 1
48
        ;
            13
49
             11
       var
50
             4
                    0x7fba525043b0
    4
        q
51
    4
             10
      real
             10
53
             13
                   5 6
    6
        procedure
55
      fib1 4
                    0x7fba52504e20
    6
             14
        (
57
                         0x7fba52505170
    6
         aLongInt
                    4
    6
        : 10
                    0
59
                    10 1
    6
         integer
        ; 13
    6
61
        b 4
                  0x7fba52505830
```

```
10
        :
63
        real
                10
                13
65
     7
                4
                        0x7fba52506790
           С
     7
                10
                        0
67
     7
                        10 1
           integer
68
     7
                13
                        1
69
     7
                        0x7fba52506e50
           d
                 4
70
     7
                10
           :
71
     7
           integer
                        10 1
     7
           )
                14
                        1
73
     7
                 13
          ;
74
                 5
     8 begin
                        0
75
          if
                 5
76
     9
           a
                 4
                        0x7fba52501d30
77
                 3
     9
           <=
78
     9
           1
                 12
                        0
79
                5
        then
80
                       0x7fba52500fd0
     9
                 4
        fib
                 2
     9
         :=
82
     9
                 4
                        0x7fba52506790
          С
     9
        18
84
    10
                5
                      2
        else
    10
        call
                 5
                       10
86
                4
                      0x7fba52500fd0
    10
        fib
    10
         (
                14
                0
    10
        18
    10
                 4
                        0x7fba52501d30
          a
90
    10
                 6
91
           1
                12
    10
    10
                 13
93
                        0x7fba52506790
    10
                4
           С
                13
    10
95
                        0x7fba52505830
    10
           b
                4
96
                6
    10
           +
97
                         0x7fba52506790
    10
                4
           С
    10
                14
99
                 0
    10
        18
                5
    11
         end
101
    11
                 13
                        1
102
                        5 6
    13
          procedure
103
                        0x7fba5250b340
    13
        fib2
                4
104
    13
          (
                14
105
                         0x7fba52501d30
    13
          a
                 4
106
    13
                10
                        0
107
        integer
    13
                        10 1
108
```

```
);
    13
              14
109
    13
               13
                     1
110
       var
               11
111
        b
:
               4
                      0x7fba52505830
    14
112
    14
               10
                      0
113
               10
                      2
    14
       real
114
    14
               13
                      1
115
    14
                11
116
         var
       C
    14
                4
                      0x7fba52506790
117
    14
               10
                      0
    14 real
               10
                      2
119
    14
               13
                      1
        ;
120
               11
    14
       var
121
    14
               4
                      0x7fba5250d1c0
122
       sum
    14
         :
               10
                      0
123
          integer
                      10 1
124
    14
    14
               13
                      1
125
               11
    15
       var
126
        b
:
               4
                      0x7fba52505830
    15
127
               10
                      0
    15
128
    15 real
               10
                      2
129
       ;
    15
               13
130
    15
       18
               0
131
                      5 6
    16
        procedure
132
    16 rawr3 4
                       0x7fba5250e950
133
    16
       (
               14
134
         b
               4
                       0x7fba52505830
    16
        :
    16
               10
                      0
136
    16 real
               10
                      2
137
         )
               14
    16
                      1
138
    16
         ;
               13
139
    17
       var
               11
140
               4
    17
       q
                      0x7fba525043b0
141
    17
               10
142
                10
    17
       real
143
    17
               13
                      1
144
    18 begin
               5
145
                      0x7fba525043b0
    19
               4
146
        q
    19
                2
147
    19
         b
               4
                      0x7fba52505830
    19
               6
149
                      0x7fba5250d1c0
    19
       sum
                4
150
    19
               13
151
    19
               0
       18
    20 call
               5
                    10
153
               4
    20 fib2
                    0x7fba5250b340
```

```
20
           (
                  14
155
                  4
                            0x7fba525043b0
     20
           q
156
     20
                  14
157
     20
                  0
         18
158
     21
          end
                  5
                          3
159
     21
                  13
                          1
160
     23 begin
                  5
161
     24
                   4
                          0x7fba525043b0
162
            q
                   2
     24
           :=
163
                  15
     24
          not
                          0
164
     24
          3
                  12
165
                  13
     24
                          1
166
     24
         18
                  0
167
           q
                  4
     25
                            0x7fba525043b0
168
                  2
     25
           :=
                          0
169
     25
            (
                  14
170
                          0
     25
             3
                  12
                          0
171
     25
             <
                  3
                          0
172
             4
                  12
                          0
     25
173
     25
            )
                  14
                          1
174
         and
                  7
     25
                          2
175
     25
          (
                  14
176
     25
          3.6
                  12
                          1
     25
          <
                  3
178
                            0x7fba52502530
     25
                  4
179
             р
     25
                  14
180
     25
                  0
          18
     25
                  13
182
     26
                  4
                           0x7fba52501d30
            a
183
     26
           :=
                   2
184
                  4
     26
            a
                           0x7fba52501d30
185
    26
                  6
186
     26
                  12
187
                  13
     26
                         1
188
     27
                  5
                         10
         call
189
                  4
                            0x7fba52504e20
     27
         fib1
190
     27
          (
                  14
191
     27
          3.2
                  12
                          1
192
     27
                  13
193
     27
          18
                  0
          1
     27
                  12
195
           )
     27
                  14
                          1
196
     27
          18
                  0
197
          18
                  0
    27
198
    27
          ;
                  13
199
    28
                  4
                          0x7fba5250d1c0
200
          sum
```

```
28
           :=
                   2
                            0
201
     28
            1
                   12
202
                   13
203
                             0x7fba52506790
     29
             С
                    4
     29
            :=
                     2
205
     29
             b
                    4
                             0x7fba52505830
206
     29
                   13
207
     30 while
                    5
208
     30
          not
                    15
209
     30
           (
                    14
     30
                   4
                             0x7fba52501d30
             a
211
                            3
     30
             >
                     3
212
     30
             0
                   12
                            0
213
     30
             )
                    14
                            1
214
     30
           do
                    5
                           1
215
                     5
                           10
216
     31
         call
                             0x7fba5250e950
     31 rawr3
                    4
^{217}
     31
              (
                   14
218
     31
                    4
                             0x7fba52505830
219
             b
                    7
     31
220
     31
             4
                   12
                            0
     31
                    14
222
                    0
     31
           18
223
     31
                    13
^{224}
     32 begin
                     5
225
     33
                             0x7fba52501d30
226
                     2
     33
     33
                     4
                             0x7fba52501d30
             a
228
     33
                     6
229
     33
             1
                   12
230
     33
             ;
                    13
231
     34
                    4
                             0x7fba52505830
             b
232
     34
           :=
233
                             0x7fba5250d1c0
     34
                     4
          sum
234
     34
                    13
235
     34
          18
                    0
236
     35
                             0x7fba5250d1c0
          sum
237
                     2
     35
           :=
     35
                     4
                             0x7fba52506790
             С
239
     35
                     6
     35
                     4
                             0x7fba5250d1c0
           sum
^{241}
     35
                    13
242
     35
          18
243
     36
                     4
                             0x7fba52506790
            С
     36
           :=
                     2
245
     36
            b
                             0x7fba52505830
246
```

```
37
        end
                5
                        3
247
    37
         ;
                 13
                       1
248
                        0x7fba5250b340
    38
        fib2
                4
249
                2
                        0
    38
         :=
250
                4
    38
        sum
                        0x7fba5250d1c0
251
    38
        18
                 0
252
    39
        end
                 5
253
        ;
    39
                 13
                        1
254
                        5 6
         procedure
    41
255
                4
    41
        init
                         0x7fba5260f370
    41
                 13
                        1
257
    42 begin
                 5
258
    43
                4
                        0x7fba5260fe20
        n
259
                2
    43
         :=
                        0
260
         12
    43
                12
                        0
261
    43
262
         18
                 0
    43
                 13
                        1
263
    44
         if
                5
                        5
264
    44
                 15
                        0
        not
265
    44
         (
                 14
                        0
266
    44
          1
                12
                        0
267
    44
          <
                 3
268
         74
    44
                12
                        0
269
    44
          )
                 14
                        1
270
                6
    44
          or
271
    44
          3
                 12
272
    44
                 5
                        8
        then
        18
    44
                 0
274
    44
                4
                        0x7fba52502530
275
         р
    44
                2
         :=
276
           (
    44
                 14
                        0
277
    44
           1
                 12
                        0
278
    44
                3
                        3
          >
279
           2
                 12
                        0
    44
280
           )
                 14
    44
                        1
281
        and
                 7
                        2
    44
282
    44
          (
                 14
                        0
283
    44
           3
                 12
                        0
284
                3
    44
           <
285
    44
                4
          n
                        0x7fba5260fe20
    44
          )
                 14
287
                 0
    44
        18
288
    45
        else
                 5
289
    45
                  4
                        0x7fba52502530
         р
         :=
    45
                  2
291
    45
         not
                 15
                        0
292
```

```
45
           2
                  12
                           0
293
     45
                  13
                           1
294
     45
                   0
         18
295
                                    0x7fba52502f50
                            4
     46
          numsArray
     46
           [
                  14
                           2
297
     46
             3
                  12
                           0
298
            ]
     46
                  14
                           3
299
     46
                  2
                           0
           :=
300
     46 15.56
                  12
301
                  13
     46
     46
          18
                  0
303
                  4
                            0x7fba525043b0
     47
           q
304
     47
                   2
           :=
305
     47
                  4
                            0x7fba525043b0
306
            q
            [
                  14
     47
307
     47
                  12
                           0
308
             4
     47
             ]
                  14
                           3
309
     47
                  13
310
     47
          18
                  0
311
                   4
                            0x7fba525043b0
     48
           q
312
     48
             [
                  14
313
     48
             4
                  12
                           0
314
             ]
     48
                  14
                           3
315
     48
          :=
                   2
                           0
316
                  0
     48
         18
317
     48
          12
                  12
                           0
318
     49
          end
                   5
                           3
319
     49
                  13
                           1
320
     51 begin
                  5
                           0
321
     52
         call
                   5
                          10
322
     52
         init
                  4
                            0x7fba5260f370
323
     52
                  13
                          1
324
     53
                  5
                          10
         call
325
     53
                  4
                            0x7fba5250b340
         fib2
326
     53
                   13
327
     53
         18
                  0
328
                  5
     54
        call
                          10
329
     54 rawr3
                  4
                            0x7fba5250e950
330
     54
         (
                   14
331
                  0
     54
          18
     54
          34
                  12
333
                   13
                           0
     54
334
     54
          56
                   12
                           0
335
           )
                  14
     54
                           1
     55
           end
                   5
                           3
337
     55
                   13
                           2
338
```

339 56 EOF 1 0

3.3 Error-Free

Listing 9: Error-Free Source Code

```
1
   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
10
          a := 2;
          z[a] := 4;
12
          c[3] := 3
13
         end;
14
15
      procedure proc2(x: integer; y: integer);
16
         var e: real;
18
         procedure proc3(n: integer; z: real);
19
           var e: integer;
20
21
           procedure proc4(a: integer; z: array [1..3] of
               real);
             var x: integer;
23
             begin
24
               a:= e
             end;
26
27
           begin
28
             a:= e;
29
             e:= c[e]
30
31
           end;
32
         begin
33
           call proc1(x, e, c, b);
34
           call proc3(c[1], e);
35
           e := e + 4.44;
36
           a := (a \mod y) \operatorname{div} x;
37
           while ((a \ge 4) \text{ and } ((b \le e)
38
                             or (not (a = c[a]))) do
39
             begin
40
                a := c[a] + 1
41
```

```
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.
```

Listing 10: Error-Free Listing File

```
1
          2 program test (input, output);
2
              var a : integer;
          3
              var b : real;
          4
          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;
9
        10
                begin
10
        11
                   a := 2;
11
        12
                   z[a] := 4;
12
        13
                   c[3] := 3
        14
                  end;
14
        15
        16
               procedure proc2(x: integer; y: integer);
16
        17
                 var e: real;
17
        18
18
                 procedure proc3(n: integer; z: real);
        19
19
        20
                    var e: integer;
20
        21
21
        22
                    procedure proc4(a: integer; z: array
22
            [1..3] of real);
        23
                      var x: integer;
23
        24
                      begin
24
        25
                        a:= e
        26
                      end;
26
        27
27
        28
                    begin
28
        29
                      a:=e;
        30
                      e:= c[e]
30
        31
                    end;
        32
32
```

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

Listing 11: Error-Free Mem File

1	ID	Memory Offset	
2	a	0	
3	Ъ	4	
4	С	12	
5	d	0	
6	е	0	
7	е	0	
8	Х	0	

Listing 12: Error-Free Token File

```
1
      FILEEND
2
      ASSIGNOP
3
      RELOP
4
      ID
5
      CONTROL
6
      ADDOP
7
      MULOP
8
      WS
9
      ARRAY
10
      TYPE
11
      VAR
12
      NUM
13
      PUNC
```

```
14 14
       GROUP
  15
      INVERSE
      LEXERR
      SYNERR
  17
  18
     SEMERR
18
                    Token Attribute Token Type
       LineLexeme
19
    2 program
                     5 7
20
    2
                      0x7f8012600dc0
      test 4
21
    2
      (
              14
22
    2 input
              4
                     0x7f8012601160
    2 ,
              13
24
    2output
              4
                     0x7f80126014e0
25
              14
    2 )
                     1
26
27
        ;
              13
      var
    3
              11
       a
                     0x7f8012601bb0
    3
              4
29
        :
    3
              10
                     0
30
      integer; 13
    3
                     10 1
31
    3
                     1
       var
    4
              11
33
    4
        b
              4
                     0x7f8012602560
        :
              10
35
              10
      real
36
    4
              13
                     1
37
        var
              11
38
    5
        С
              4
                    0x7f8012602eb0
39
              10
                     0
    5
40
    5 array
               9
                     0
41
              14
                     2
      [
42
    5
        1
              12
                     0
43
    5
        . .
              9
                     1
44
    5
      2
              12
                     0
45
    5
        ]
             14
                     3
46
    5
       of
              9
                     2
47
       integer; 13
                     10 1
    5
48
    5
                     1
49
    7
      procedure
                     5 6
50
    7 proc1
             4
                     0x7f8012604480
51
    7
        (
              14
52
    7
                     0x7f80126046f0
              4
         X
    7
             10
54
                     10 1
    7
         integer
    7
        ; 13
56
                     0x7f8012604c90
    7
              4
        У
    7
             10
        :
58
    7 real
             10
                     2
```

```
7 ; 13 1
8 z 4 0x7f8012605bf0
60
             10
62
              9
    8 array
                    0
    8
       ]
             14
                    2
64
    8
        1
             12
                    0
    8
              9
                    1
        . .
66
        2
             12
                    0
    8
67
    8
         ]
              14
                    3
68
             9
    8
       of
                    2
         integer
                   10 1
    8
70
        ;
             13
    8
                   1
71
              4
                    0x7f8012606be0
    8
        q
72
    8
             10
73
    8
      real
             10
                    2
74
       )
    8
              14
75
    8
             13
                    1
76
             11
      var
77
       d
              4
                    0x7f80126076c0
    9
             10
                    0
    9
        :
79
      integer
; 13
    9
                   10 1
    9
                    1
81
             5
                    0
   10 begin
82
                    0x7f8012601bb0
   11
              4
      a
83
              2
   11
        :=
   11
         2
             12
85
             13
   11
         ;
   12
              4
                    0x7f8012605bf0
        Z
87
        [
   12
             14
             4
                    0x7f8012601bb0
   12
        a
89
        ]
   12
              14
90
   12
        :=
              2
                    0
   12
        4
             12
92
   12
             13
                   1
   13
              4
                    0x7f8012602eb0
         С
94
        [
             14
   13
                    2
   13
         3
             12
96
   13
        ]
              14
                    3
        :=
              2
   13
98
                    0
   13
        3
             12
   14
      end
              5
                    3
100
   14
              13
                    1
         procedure
   16
                    5 6
102
   16 proc2 4
                    0x7f801260ad50
103
   16
        (
             14
104
        x
   16
             4 0x7f80126046f0
105
```

```
: 10
    16
106
                    10 1
    16
         integer
107
         ; 13
                    1
108
         У
              4
                     0x7f8012604c90
    16
         : 10
    16
                     0
110
                     10 1
    16
          integer
111
    16
         ) 14
                     1
112
    16
              13
                     1
113
         ;
    17
              11
114
        var
              4
    17
                     0x7f801260c180
          е
    17
               10
116
               10
    17
117
       real
               13
                     1
    17
118
                     5 6
        procedure
    19
119
    19 proc3 4
                     0x7f801260cdc0
120
               14
121
    19
        (
                     0x7f801260d030
              4
    19
         n
122
    19
              10
                     0
         :
123
                     10 1
          integer
    19
124
    19
         ; 13
125
                     0x7f8012605bf0
    19
              4
         Z
126
    19
         :
              10
127
               10
                     2
    19
       real
       )
    19
               14
                     1
129
              13
    19
                     1
130
    20
       var
         7ar
e <del>1</del>0
              11
131
                     0x7f801260c180
    20
    20
                     0
133
    20
         integer
                     10 1
134
    20
         ; 13
                     1
135
                     5 6
    22
         procedure
136
    22 proc4 4
                      0x7f801260f070
137
       (
              14
    22
138
                      0x7f8012601bb0
              4
    22
         a
139
         :
             10
                     0
    22
140
         integer
                     10 1
    22
141
    22
         ; 13
142
    22
              4
                      0x7f8012605bf0
          Z
143
    22
          :
               10
144
    22 array
              9
                     0
    22
       [
              14
                     2
146
               12
    22
          1
                     0
147
    22
        . .
               9
                     1
148
               12
    22
         3
                     0
149
    22
         ]
               14
                     3
150
       of
    22
              9
                     2
151
```

```
22
       real
               10
152
       )
    22
                14
153
               13
154
    23
                11
       var
        х
    23
                4
                      0x7f80126046f0
156
               10
                       0
    23
157
          integer
    23
                      10 1
158
       ; 13
    23
159
                       1
    24 begin
                5
160
    25
                4
                      0x7f8012601bb0
161
          a
         :=
               2
    25
162
    25
               4
                       0x7f801260c180
          е
163
       end
                5
    26
164
               13
165
    26
       ;
    28 begin
               5
166
                      0x7f8012601bb0
    29
         a
                4
167
    29
          :=
                2
168
    29
                        0x7f801260c180
         е
               4
169
    29
               13
170
                       0x7f801260c180
    30
          е
                4
171
         :=
    30
               2
172
    30
         С
               4
                       0x7f8012602eb0
173
         [
               14
    30
    30
               4
                       0x7f801260c180
          е
175
    30
          ]
               14
176
    31
       end
               5
177
               13
    31
                      1
    33 begin
                5
179
    34 call
               5
                    10
180
                      0x7f8012604480
    34 proc1
               4
181
    34
           (
                14
182
    34
               4
                      0x7f80126046f0
           х
183
                13
    34
184
                       0x7f801260c180
    34
                4
           е
185
    34
               13
186
                       0x7f8012602eb0
    34
                4
         С
187
    34
               13
188
         b
                       0x7f8012602560
    34
               4
    34
                14
190
    34
                13
                      1
    35 call
               5
192
                        0x7f801260cdc0
    35 proc3
               4
193
    35
        (
               14
194
                      0x7f8012602eb0
                4
    35
         С
    35
               14
196
       1
    35
                12
                       0
197
```

```
35
             ]
                   14
198
     35
                   13
199
                             0x7f801260c180
                   4
200
             )
     35
                   14
     35
                   13
202
                             0x7f801260c180
     36
                    4
             е
203
     36
            :=
                   2
204
                             0x7f801260c180
     36
                    4
205
             е
     36
                    6
206
     36
          4.44
                   12
                            1
     36
                   13
208
     37
                    4
                             0x7f8012601bb0
             a
209
     37
                    2
           :=
210
     37
             (
                   14
211
     37
                    4
                             0x7f8012601bb0
            a
212
                    7
213
     37
           mod
                             0x7f8012604c90
     37
                    4
             у
214
     37
                   14
215
                    7
     37
          div
216
                             0x7f80126046f0
     37
             X
                    4
217
     37
                   13
                            1
     38 while
                    5
219
                    14
                            0
     38
              (
220
     38
             (
                   14
221
                             0x7f8012601bb0
     38
             a
                   4
222
     38
            >=
                   3
223
             4
                   12
                            0
     38
     38
             )
                   14
                            1
225
     38
           and
                    7
                            2
226
                   14
     38
             (
227
     38
             (
                    14
228
     38
             b
                    4
                             0x7f8012602560
229
     38
                    3
            <=
230
                             0x7f801260c180
     38
             е
                    4
231
     38
             )
                   14
232
     39
                     6
                            2
            or
233
     39
             (
                    14
                            0
234
                    15
                            0
     39
          not
     39
             (
                    14
236
                             0x7f8012601bb0
     39
                   4
     39
                   3
238
                             0x7f8012602eb0
     39
             С
                   4
239
     39
             Ε
                   14
240
                             0x7f8012601bb0
                    4
     39
             a
     39
             ]
                   14
242
     39
             )
                   14
                            1
243
```

```
39
            )
                   14
244
     39
             )
                   14
                            1
^{245}
     39
                   14
             )
246
                    5
     39
            do
^{247}
     40 begin
                    5
248
                   4
                             0x7f8012601bb0
     41
             а
^{249}
     41
            :=
                   2
250
                   4
                             0x7f8012602eb0
     41
             С
251
     41
             Ε
                   14
252
                   4
     41
                           0x7f8012601bb0
            a
     41
             ]
                   14
254
                    6
                            0
     41
             +
255
     41
            1
                   12
                            0
256
                            3
     42
         end
                   5
257
                            3
     43
           end
                   5
258
     43
259
                   13
                           0
     45 begin
                   5
260
                   5
                          10
     46 call
261
     46 proc2
                   4
                             0x7f801260ad50
262
     46
             (
                   14
263
                             0x7f8012602eb0
     46
                   4
             С
264
     46
             [
                   14
265
     46
             4
                   12
                            0
266
     46
             ]
                   14
                            3
267
     46
                   13
268
     46
             С
                   4
                             0x7f8012602eb0
269
     46
             [
                   14
                            2
     46
             5
                   12
                            0
271
             ]
     46
                   14
                            3
272
     46
             )
                   14
                           1
273
     46
                   13
                           1
274
                  5
     47
        call
                          10
275
     47 proc2
                             0x7f801260ad50
                   4
276
     47
             (
                   14
277
     47
                   4
                             0x7f8012602eb0
             С
278
                   14
                            2
     47
             [
^{279}
     47
             4
                   12
                            0
280
     47
             ]
                   14
                            3
     47
                   13
                            0
282
             2
                   12
                            0
     47
     47
             )
                   14
                            1
284
                   13
     47
                            1
285
     48
            if
                    5
286
                   14
     48
            (
     48
                   4
                             0x7f8012601bb0
             a
288
     48
             <
                    3
```

```
48
           2
                 12
                         0
290
     48
            )
                  14
                         1
291
     48
        then
292
                 4
                         0x7f8012601bb0
     48
            а
    48
          :=
                 2
294
     48
           1
                 12
                          0
295
     48
         else
                 5
296
                  4
                         0x7f8012601bb0
     48
297
          a
     48
           :=
                   2
298
                 4
     48
                         0x7f8012601bb0
            a
    48
                 6
300
            2
                 12
                          0
     48
301
    48
                  13
                         1
302
     49
          if
                 5
303
     49
          (
                  14
304
     49
                 4
                         0x7f8012602560
            b
305
    49
            >
                  3
306
     49
          4.2
                 12
307
                  14
     49
           )
                          1
308
    49
                  5
        then
309
                           0x7f8012601bb0
    49
                 4
            a
310
                 2
    49
           :=
311
                 4
     49
            С
                           0x7f8012602eb0
312
    49
            [
                  14
313
                           0x7f8012601bb0
    49
                  4
314
            a
            ]
    49
                  14
315
                  5
                          3
    50
          end
316
    50
                  13
                          2
317
                          0
    51
          EOF
                 1
318
```

Appendix 2: Program 30 Listings

Listing 13: compiler.c

```
#include<stdio.h>
   #include<stdlib.h>
   #include<stdbool.h>
   #include "dataStructures/
       linkedList/linkedlist.h"
   #include "errorHandler/
       errorHandler.h"
   #include "globals/globals.h"
   #include "handler/handler.h"
   #include "parser/parser.h"
   // Global file constants
   static const char TOKEN_PATH[]
        = "out/tokens.dat";
   static const char LISTING_PATH
       [] = "out/listing.txt";
   static const char MEM_PATH[] =
        "out/mem.txt";
   static const char RESWORD_PATH
       [] = "compiler/data/
       reswords.dat";
16
   // Returns 1 on failure, 0 on
       success.
   int init(char* sourcePath) {
   return initializeGlobals() &&
       initializeErrorHandler() &&
   initializeHandler(sourcePath,
       RESWORD_PATH, LISTING_PATH,
        TOKEN_PATH, MEM_PATH)
   ? 0 : 1;
22
   int run()
24
25
   generateParseTree();
27
   return 0;
```

```
int main(int argc, char *argv
       []) {
   if (argc != 2) {
   fprintf(stderr, "%s\n", "
       Expected exactly one file
       to compile!");
   } else {
   if (init(argv[1]) == 0) {
   if (run() != 0)
   fprintf(stderr, "%s\n", "Run
       failed. Could not terminate
        properly.");
   } else {
   fprintf(stderr, "%s\n", "
       Initialization process
       failed in tokenizer.");
40
   return 0;
```

Listing 14: declarationsTree.h

```
#ifndef DECLARATIONS_TREE_H
#define DECLARATIONS_TREE_H
#include <stdbool.h>
#include "../../tokenizer/
    tokens.h"
typedef struct tree_node {
char* lex; // The lexeme
LangType type; // The type
union {
bool param; // True if param
bool add_right; // True if add
     right to green node
};
struct tree_node* left;
struct tree_node* right;
struct tree_node* parent;
} tree_node;
```

20

14

15

16

```
typedef struct LinkedTree {
                                           of the linked list
   struct node* head;
                                       void* pop(LinkedList* list);
   } DeclarationsTree;
                                    23
                                       #endif // LINKED_H_
   // Green nodes designate
       scopes, and blue nodes
                                            Listing 16: errorHandler.h
       designate variables
                                       #ifndef ERROR_HANDLER_H
   bool check_add_node(Token*
                                       #define ERROR_HANDLER_H
       decl);
                                       #include "../tokenizer/tokens.
   tree_node* get_last_green_node
                                           h"
       ():
   tree_node*
                                       extern const char* lexErrs[];
       start_param_matching(Token*
                                       char* synErr;
                                       LinkedList* semErrs;
   void reached_end_of_scope();
   LangType get_type(Token* id);
30
                                       void throw_sem_error(char* msg
31
                                           );
   #endif // DECLARATIONS_TREE_H
                                       void throw_syn_error(Token*
                                    10
                                           received, const Token**
         Listing 15: linkedlist.h
                                           expected, int exp_size);
   #ifndef LINKED_H_
                                       void throw_lex_error(enum
   #define LINKED_H_
                                           TokenType attribute, int
                                           aspect, int start, int
   // Behaves like a stack
                                           length);
   struct node {
                                       int initializeErrorHandler();
   void* data;
                                    13
   struct node* next;
                                       Token* getNextErrorToken();
                                    15
                                       #endif // ERROR_HANDLER_H
   typedef struct LinkedNodes {
   struct node* head;
                                              Listing 17: globals.h
   int size;
                                       #ifndef GLOBALS_H
   } LinkedList;
                                       #define GLOBALS_H
   // Add an item to the front of
                                       extern int START;
        the linked list
                                       extern int LINE;
   int add(LinkedList* list, void
                                       extern char* BUFFER;
       * data, int size);
   // For use as a queue; slow,
                                       void updateLine(char* line);
       do not use
                                       int initializeGlobals();
   int addLast(LinkedList* list,
       void *data, int size);
                                    10
                                       #endif // GLOBALS_H
19
   // Pop an item from the front
```

```
#include "../../tokenizer/
          Listing 18: handler.h
                                           tokens.h"
   #ifndef HANDLER_H
                                       #include "../../dataStructures
   #define HANDLER_H
                                           /declarationsTree/
                                           declarationsTree.h"
   #include<stdbool.h>
                                       #include "../../errorHandler/
   #include "../tokenizer/tokens.
                                           errorHandler.h"
       h"
                                    10
                                       extern Token* current_tok;
                                    11
   int initializeHandler(const
       char* sourcePath, const
                                       // All of these must have
       char* resPath,
                                           their follows added to the
   const char* listingPath, const
                                           sync set
        char* tokenPath,
                                       void program();
   const char* memPath);
                                       void id_list();
   bool handleToken(Token* token)
                                       void id_list_tail();
                                       void declarations();
   void outputWidth(char* lex,
11
                                       LangType type();
       int width);
                                       LangType standard_type();
12
                                       void subprogram_declarations()
   #endif // HANDLER_H
                                       void subprogram_declaration();
          Listing 19: parser.h
                                       bool subprogram_head();
                                       void arguments();
   #ifndef PARSER H
                                    23
                                       void parameter_list();
   #define PARSER H
                                       void parameter_list_tail();
   #include<stdbool.h>
                                       void compound_statement();
                                       void optional_statements();
   int generateParseTree();
                                       void statement_list();
   Token* match(const Token*
                                       void statement_list_tail();
       source, bool strict);
                                    29
                                       void statement();
                                    30
   void require_sync(const Token*
                                       void else_tail();
        sync_set[], int size,
                                       LangType variable();
   const Token* first_set[], int
                                    32
                                       LangType array_access(LangType
       first_size);
                                            id_type);
                                       void procedure_statement();
   #endif // PARSER_H
                                       void optional_expressions(
                                    35
                                           tree_node* to_match, bool
        Listing 20: productions.h
                                           should_error);
   #ifndef voidS_H
                                       void expression_list(tree_node
   #define voidS_H
                                           * to_match, bool
   #include <stdio.h>
                                           should_error);
   #include <stdlib.h>
                                       void expression_list_tail(
                                           tree_node* to_match, bool
   #include "../../globals/
                                           should_error);
                                       LangType expression();
       globals.h"
```

```
LangType related_expression();
                                       int mulop(Token* storage, char
   LangType simple_expression();
                                           * str, int start);
   LangType
                                       int addop(Token* storage, char
       simple_expression_tail();
                                           * str, int start);
   LangType term();
                                       int whitespace(Token* storage,
42
                                    15
   LangType term_tail();
                                            char* str, int start);
                                       int relop(Token* storage, char
   LangType factor();
   LangType factor_tail();
                                           * str, int start);
   void sign();
                                    17
                                       int idres(Token* storage, char
   #endif // voidS_H
                                           * str, int start);
                                       int initIDResMachine(FILE*
                                           resFile);
        Listing 21: symbolTable.h
   #ifndef SYMBOL_TABLE_H
                                       extern const machine machines
   #define SYMBOL_TABLE_H
                                       #endif // MACHINES_H
   int initSymbolTable();
   char* checkSymbolTable(char*
                                             Listing 23: tokenizer.h
       name);
                                       #ifndef PROCESSOR_H_
   char* pushToSymbolTable(char*
                                       #define PROCESSOR_H_
       name, size_t length);
                                       #include<stdio.h>
                                       #include "tokens.h"
   #endif // SYMBOL_TABLE_H
                                       Token* getNextToken();
         Listing 22: machines.h
                                       int initializeTokens(FILE*
                                           resFile);
   #ifndef MACHINES_H
   #define MACHINES_H
                                       #endif // PROCESSOR_H_
   #include <stdio.h>
   #include "../tokens.h"
                                               Listing 24: tokens.h
   typedef int (*machine)(Token*,
                                       #ifndef TOKENS_H
        char*, int);
                                       #define TOKENS_H
   int intMachine(Token* storage,
                                       #include<stdbool.h>
        char* str, int start);
   int realMachine(Token* storage
                                       #include "../dataStructures/
       , char* str, int start);
                                           linkedList/linkedList.h"
   int longRealMachine(Token*
       storage, char* str, int
                                       // Must have a boolean
       start);
                                           indicating whether it is a
   int grouping(Token* storage,
                                           parameter or not
       char* str, int start);
                                       typedef enum LangType {ERR,
  int catchall(Token* storage,
                                           REAL, INT, BOOL, PGNAME,
       char* str, int start);
                                           PPNAME,
```

```
PROC, AINT, AREAL} LangType;
                                        extern const Token semic_tok;
10
                                        extern const Token colon_tok;
   enum TokenType {NOOP, FILEEND,
                                       extern const Token period_tok;
                                    48
12
        ASSIGNOP, RELOP, ID,
                                       extern const Token dotdot_tok;
   CONTROL, ADDOP, MULOP, WS,
                                       extern const Token lbrac_tok;
13
       ARRAY, TYPE,
                                       extern const Token rbrac_tok;
   VAR, NUM, PUNC, GROUP, INVERSE
                                       extern const Token addop_tok;
14
                                       extern const Token array_tok;
   LEXERR, SYNERR, SEMERR);
                                       extern const Token
15
                                           assignop_tok;
   // The token data type
                                       extern const Token begin_tok;
   typedef struct T_Type {
                                       extern const Token call_tok;
   enum TokenType attribute; //
                                       extern const Token do_tok;
       Attribute
                                       extern const Token else_tok;
                                       extern const Token end_tok;
20
   union { // Aspect or character
                                        extern const Token id_tok;
                                    60
        pointer
                                        extern const Token if_tok;
   int aspect;
                                        extern const Token integer_tok
                                    62
22
   char* id;
                                        extern const Token
24
                                           integer_val_tok;
   int start; // Start in the
                                       extern const Token of_tok;
       1. i.n.e
                                       extern const Token
   int length; // Length of the
                                           real_val_tok;
27
       lexeme
                                       extern const Token mulop_tok;
                                       extern const Token not_tok;
28
   union { // Value of the number
                                       extern const Token num_tok;
       , or length of the array
                                        extern const Token
                                    69
   int int_val;
                                           procedure_tok;
   double real_val;
                                       extern const Token program_tok
   int array_length;
                                            ;
   };
                                       extern const Token real_tok;
                                       extern const Token relop_tok;
34
   LangType type; // The type of
                                       extern const Token then_tok;
       the token
                                        extern const Token var_tok;
   bool param; // Whether the
                                        extern const Token while_tok;
       token is a parameter or not 76
                                       extern const char* catNames
   } Token;
                                            [19];
38
                                        extern const char* typeNames
   extern const Token eof_tok;
                                            [9];
   extern const Token lparen_tok;
                                       const Token* getTokenFromLex(
   extern const Token rparen_tok;
                                           char* lex);
   extern const Token plus_tok;
                                       const char* getLexFromToken(
                                           Token* token, bool strict);
   extern const Token comma_tok;
   extern const Token minus_tok;
```

```
// The type; else null if
                                       static void initialize_d_tree
       impossible
                                            () {
   LangType convert_to_array(
                                       d_tree = malloc(sizeof(*d_tree
       LangType type);
                                           ));
   LangType convert_from_array(
                                       green_node_stack = malloc(
84
                                    18
                                            sizeof(*green_node_stack));
       LangType type);
85
                                    19
   // Returns the type produced
                                    20
       by the operation
                                       static int get_width(Token*
                                    21
   LangType type_lookup(LangType
                                           val) {
       first, LangType second,
                                       switch (val -> type) {
                                    22
       Token* op);
                                        case INT: return 4;
                                        case REAL: return 8;
   // Returns true if the tokens
89
                                    25
       are equivalent, false
                                       case AINT: return 4*(val ->
       otherwise
                                            array_length);
   bool tokens_equal(const Token*
                                       case AREAL: return 8*(val ->
                                    27
        p1, Token* p2, bool strict
                                            array_length);
       );
                                       default: return 1000000;
91
                                       }
92
                                    30
   #endif // TOKENS_H
                                    31
93
                                       static bool check_node(char*
      Listing 25: declarationsTree.c
                                            id, bool green) {
   #include <stdlib.h>
                                       tree_node* current_node =
   #include <stdio.h>
                                           bottom_node;
                                       while (current_node != NULL) {
   #include "../../handler/
                                       // Already exists
       handler.h"
                                       if (id == current_node -> lex)
   #include "../../errorHandler/
                                       return true;
       errorHandler.h"
   #include "../../globals/
                                       if (!green && (current_node ->
       globals.h"
                                            type == PROC ||
   #include "declarationsTree.h"
                                            current_node -> type ==
                                           PGNAME))
   static int offset = 0;
                                       return false;
10
   static DeclarationsTree*
                                       // We've passed the most
       d_tree = NULL;
                                            recent green node, and this
   static tree_node* bottom_node
                                             is a blue one
       = NULL;
                                       if (!green && (current_node ->
                                            type == PROC ||
   static LinkedList*
                                            current_node -> type ==
14
       green_node_stack = NULL;
                                           PGNAME))
15
                                       break;
```

```
tree_node* addition = malloc(
46
   current_node = current_node ->
                                            sizeof(*addition));
                                       addition -> lex = decl -> id;
        parent;
                                    82
                                       addition -> type = decl ->
48
                                            type;
49
   return false;
                                       addition -> add_right = false;
   }
51
                                    85
                                       // Add it to the top of the
52
   static bool
                                            stack
       check_add_green_node(Token* 87
                                       add(green_node_stack, &
                                           addition, sizeof(&addition)
        decl) {
   if (d_tree == NULL)
   initialize_d_tree();
                                       addition -> left = NULL;
   if (bottom_node == NULL)
                                       addition -> right = NULL;
                                       addition -> parent =
58
   tree_node* addition = malloc(
                                           bottom_node;
       sizeof(*addition));
   addition -> lex = decl -> id;
                                       // Make it the new bottom node
   addition -> type = decl ->
                                       if (bottom_node -> add_right
                                            == true)
       type;
   addition -> add_right = false;
                                       bottom_node -> right =
                                    95
62
                                            addition;
   // Add it to the top of the
                                       else
       stack
                                       bottom_node -> left = addition
   add(green_node_stack, &
       addition, sizeof(&addition) 98
                                       bottom_node = addition;
   addition -> left = NULL;
                                       return true;
67
                                    101
   addition -> right = NULL;
                                       }
                                   102
   addition -> parent = NULL;
   bottom_node = addition;
                                   104
   return true;
                                       static bool
                                    105
                                            check_add_blue_node(Token*
72
                                            decl) {
   // Check if it's been declared 106
                                       // If there's no scope, that's
        at all
                                             an error!
   if (check_node(decl -> id,
                                       if (d_tree == NULL)
       true))
                                       return false;
   return false;
76
                                    109
                                       // It's been declared in the
   offset = 0;
                                            scope already
78
                                       if (check_node(decl -> id,
   // It hasn't been declared;
                                           false))
       create it
                                    112 return false;
```

```
return current_node;
113
    // It hasn't been declared;
                                          current_node = current_node ->
        create it
                                               parent;
    tree_node* addition = malloc(
                                     145
        sizeof(*addition));
                                      146
    addition -> lex = decl -> id;
                                          return NULL;
    //printf("%s\n", addition ->
                                          }
117
                                      148
        lex);
                                      149
    addition -> type = decl ->
                                          bool check_add_node(Token*
118
                                      150
                                              decl) {
                                          char* errorMessage ;
    addition -> param = decl ->
119
                                      151
        param;
                                          switch (decl -> type) {
                                          case PGNAME:
120
    if (!addition -> param)
                                          case PROC: if (!
121
                                              check_add_green_node(decl))
122
    outputWidth(addition -> lex,
123
        offset);
                                           errorMessage = calloc(100,
                                      155
    offset += get_width(decl);
                                               sizeof(*errorMessage));
124
                                           sprintf(errorMessage,
125
                                      156
                                          "A program or procedure named
126
    addition -> left = NULL;
                                              '%.*s' is already defined
127
    addition -> right = NULL;
                                              in this scope!",
128
    addition -> parent =
                                          decl -> length, &BUFFER[decl
        bottom_node;
                                              -> start]);
                                          throw_sem_error(errorMessage);
130
    bottom_node -> left = addition 160
                                          return false;
131
    bottom_node = addition;
                                           return true;
132
                                      162
133
                                      163
    //printf("(%s, %s)\n",
                                          default: if (!
134
                                      164
        bottom_node -> lex,
                                              check_add_blue_node(decl))
        bottom_node -> parent ->
        lex);
                                           errorMessage = calloc(100,
                                      165
    return true;
                                               sizeof(*errorMessage));
135
    }
                                           sprintf(errorMessage,
136
                                      166
                                          "A variable named '%.*s' is
137
                                              already defined in the
    tree_node*
138
                                              local scope!",
        start_param_matching(Token*
                                          decl -> length, &BUFFER[decl
         id) {
                                      168
    tree_node* current_node =
                                              -> start]);
        bottom_node;
                                          throw_sem_error(errorMessage);
                                      169
   while (current_node != NULL)
                                          return false;
140
                                      170
                                      171
141
   if (current_node -> type ==
                                           return true;
        PROC && current_node -> lex<sub>173</sub>
                                          }
         == id -> id)
                                         }
```

```
// Do a byte-by-byte copy of
175
   void reached_end_of_scope() {
                                             the data
   bottom_node = (*(tree_node**)
                                     13 for (int i = 0; i < size; i++)
                                        *(char *) (addition -> data +
        pop(green_node_stack));
   bottom_node -> add_right =
                                             i) = *(char *) (data + i);
178
        true;
                                        list -> size++;
179
180
                                        list -> head = addition;
181
   LangType get_type(Token* id) {
                                        return list -> size;
   char* errorMessage;
                                     20
   if (id == NULL)
                                        // For use as a queue; slow,
   return ERR;
185
                                             do not use
186
   tree_node* current_node =
                                        int addLast(LinkedList* list,
        bottom_node;
                                             void *data, int size)
   while (current_node != NULL)
188
                                         struct node* addition = malloc
189
   if (current_node -> lex == id
                                             (sizeof(*addition));
190
                                         addition -> data = malloc(size
   return current_node -> type;
                                            );
                                        addition -> next = NULL;
192
   current_node = current_node ->
193
         parent;
                                        for (int i = 0; i < size; i++)
                                         *(char *) (addition -> data +
194
                                             i) = *(char *) (data + i);
195
   return NULL;
                                        struct node* current = list ->
197
                                     32
                                             head;
          Listing 26: linkedlist.c
                                        if (list \rightarrow size == 0)
   #include<stdlib.h>
                                        list -> head = addition;
    #include<stdio.h>
                                        else {
                                        while (current -> next != NULL
   #include "linkedlist.h"
                                         current = current -> next;
    int add(LinkedList* list, void
                                         current -> next = addition;
         *data, int size)
                                        list -> size++;
   struct node* addition = malloc
        (sizeof(*addition));
   addition -> data = malloc(size
                                        void* pop(LinkedList* list)
   addition -> next = (list ->
                                     47
                                        {
       head);
                                        struct node* head = list ->
```

```
head;
                                    23
   struct node* next = head ->
                                       int initializeErrorHandler()
                                    25
   void* data = head -> data;
                                       errorList = malloc(sizeof(*
51
   list -> head = next;
                                           errorList));
   list -> size--;
                                       semErrs = malloc(sizeof(*
                                           semErrs));
   //free(head); // TODO this is
                                       return errorList != NULL &&
                                    29
       necessary; should fix
                                           semErrs != NULL;
   return data;
56
                                    30
                                       void throw_syn_error(Token*
                                    32
                                           received, const Token **
        Listing 27: errorHandler.c
                                           expected, int exp_size)
   #include<string.h>
   #include<stdlib.h>
                                       // Generate token
                                       Token* errToken = malloc(
                                           sizeof(*errToken));
   #include "errorHandler.h"
                                        errToken -> attribute = SYNERR
   static LinkedList* errorList;
                                        errToken -> aspect = 0;
                                    37
                                       errToken -> start = received
   const char* lexErrs[] = {"
                                           -> start;
       Unrecognized symbol:",
                                       errToken -> length = received
    "ID length exceeded 10
                                           -> length;
        characters:",
    "Int length exceeded 10
                                       add(errorList, errToken,
        characters:",
                                           sizeof(*errToken));
    "Integer part of real exceeded _{42}
         5 characters:",
                                       // Generate error message
    "Fractional part of real
                                    44 // Calculate space needed
13
        exceeded 5 characters:",
                                      int size = strlen("Found '';
    "Exponent part of long real
                                           expected ");
14
        exceeded 2 characters:",
                                       size += strlen(getLexFromToken
                                    46
    "Missing exponent part of long
                                            (received, true));
15
         real:",
                                       for (int i = exp_size - 1; i
    "Leading 0 in int:",
                                           >= 0; i--) {
16
    "Excessive leading 0 in real:" _{48}
                                       size += strlen("'');
                                       size += strlen(getLexFromToken
    "Trailing 0 in real:",
                                            (expected[i], expected[i]
    "Leading 0 in exponent:",
                                           -> start));
19
    "Attempt to use real exponent: _{50}
                                       if (i > 0)
        "};
                                       size += strlen(", ");
   char* synErr;
                                    52
   LinkedList* semErrs;
                                    size += strlen(" instead.");
```

```
size += 1; // Null terminator
                                       errToken -> aspect = 0;
   synErr = malloc(sizeof(*synErr
                                       errToken -> start = 0;
56
                                    86
       ) * size);
                                        errToken -> length = 0;
   synErr[size - 1] = '\0';
57
                                    88
   strcpy(synErr, "Found '");
                                        addLast(errorList, errToken,
   int current = 7;
                                           sizeof(*errToken));
   int len = strlen(
       getLexFromToken(received,
                                       // Set the msq
                                    91
       true));
                                        addLast(semErrs, &msg, sizeof
   strcpy(&synErr[current],
                                            (&msg));
       getLexFromToken(received,
       true)):
   current += len;
                                       void throw_lex_error(enum
   strcpy(&synErr[current], "';
                                           TokenType attribute, int
       expected ");
                                           aspect, int start, int
   current += 12;
                                           length)
64
   for (int i = exp_size - 1; i
65
       >= 0; i--) {
                                       Token* errToken = malloc(
   strcpy(&synErr[current], "'");
                                           sizeof(*errToken));
66
   current += 1;
                                       errToken -> attribute =
   len = strlen(getLexFromToken(
                                           attribute;
       expected[i], expected[i] -> 99
                                       errToken -> aspect = aspect;
        start));
                                        errToken -> start = start;
   strcpy(&synErr[current],
                                        errToken -> length = length;
       getLexFromToken(expected[i 102
       ], expected[i] -> start)); 103
                                       add(errorList, errToken,
                                           sizeof(*errToken));
   current += len;
   strcpy(&synErr[current], "'"); 104
   current += 1;
   if (i > 0) {
                                       Token* getNextErrorToken()
   strcpy(&synErr[current], ", ") 107
                                       if (errorList -> size > 0)
                                    108
   current += 2;
                                       return (Token *) pop(errorList
75
                                    109
76
                                   110
77
   strcpy(&synErr[current], "
                                       return NULL;
                                   111
       instead.");
                                       }
                                   112
79
                                               Listing 28: globals.c
   void throw_sem_error(char* msg
                                       #include<string.h>
                                       #include<stdlib.h>
   // Generate error token
                                       #include<stdbool.h>
   Token* errToken = malloc(
                                     4 #include<stdio.h>
       sizeof(*errToken));
   errToken -> attribute = SEMERR 5 #include "globals.h"
```

```
ListingLineSpace = 7;
                                       static const int
   char* BUFFER;
   int LINE = 0;
                                           ListingErrSpace = 50;
   int START = 0;
                                       static const int
                                           ListingLexSpace = 20;
10
   int initializeGlobals()
                                       static const int MemNameSpace
12
   BUFFER = malloc(sizeof(char*)
                                           = 10;
                                       static const int MemValSpace =
       *73);
   return (BUFFER != NULL);
                                            20;
15
                                       void writeEOFToken()
16
   void updateLine(char* line)
                                       {
17
                                    26
                                       fprintf(tokenFile, "%*d%*.*s%*
   START = 0;
                                           d%*d\n", TokenLineSpace,
   LINE++;
                                           LINE, TokenLexSpace,
   strcpy(BUFFER, line);
                                       3, "EOF", TokenTypeSpace,
                                           FILEEND, TokenAttrSpace, 0)
22
                                       }
                                    29
          Listing 29: handler.c
   #include<stdio.h>
                                       int initializeHandler(const
                                           char* sourcePath, const
   #include "handler.h"
                                           char* resPath,
   #include "../globals/globals.h
                                       const char* listingPath, const
                                            char* tokenPath,
   #include "../tokenizer/
                                       const char* memPath)
       tokenizer.h"
   #include "../errorHandler/
                                       if ((sourceFile = fopen(
       errorHandler.h"
                                           sourcePath, "r")) == NULL)
                                    36
   static FILE* listingFile;
                                       fprintf(stderr, "%s\n", "
                                    37
   static FILE* tokenFile;
                                           Source was null?");
   static FILE* sourceFile;
                                       return 0;
   static FILE* memFile;
12
                                    40
   static const int
                                       FILE* resFile = fopen(resPath,
                                    41
       TokenLineSpace = 10;
                                             "r");
   static const int
                                       initializeTokens(resFile);
       TokenTypeSpace = 20;
                                       fclose(resFile);
   static const int
       TokenAttrSpace = 20;
                                       if ((listingFile = fopen(
   static const intTokenLexSpace
                                           listingPath, "w+")) == NULL
       = 20;
                                        (tokenFile = fopen(tokenPath,
   static const int
                                            "w+")) == NULL ||
```

```
(memFile = fopen(memPath, "w+"
       )) == NULL)
                                        fprintf(tokenFile, "%*d%*.*s%*
   return 0;
                                            d%*d\n", TokenLineSpace,
48
                                            LINE,
   for (size_t i = FILEEND; i <=</pre>
                                        TokenLexSpace, description ->
50
                                            length, &BUFFER[description
       SEMERR; i++) {
   fprintf(tokenFile, "%-5zu%s\n"
                                             -> start],
       , i, catNames[i]);
                                        TokenTypeSpace, description ->
                                             attribute, TokenAttrSpace,
52
                                        description -> aspect);
53
                                        if (description -> attribute
                                            == LEXERR)
   char line[72];
55
   if (fgets(line, sizeof(line),
                                        fprintf(listingFile, "%*s:%*s
                                            %*.*s\n", ListingLineSpace
       sourceFile) != NULL)
                                            - 1.
57
   updateLine(line);
                                        catNames[description ->
   fprintf(listingFile, "%*d\t%s"
                                            attribute], ListingErrSpace
       , ListingLineSpace, LINE,
                                        lexErrs[description -> aspect
       line);
   } else {
                                            ], ListingLexSpace,
   writeEOFToken();
                                        description -> length, &BUFFER
   }
                                            [description -> start]);
62
                                        else if (description ->
   fprintf(tokenFile, "%*s%*s%*s
                                            attribute == SYNERR)
       %*s\n", TokenLineSpace, "
                                        fprintf(listingFile, "%*s: %s\
       Line",
                                            n", ListingLineSpace - 1,
   TokenLexSpace, "Lexeme",
                                        catNames[description ->
   TokenAttrSpace, "Token
                                            attribute], synErr);
66
       Attribute",
                                        else if (description ->
   TokenTypeSpace, "Token Type");
                                            attribute == SEMERR)
67
                                        fprintf(listingFile, "%*s: %s\
68
   fprintf(memFile, "%*s%*s\n",
                                            n", ListingLineSpace - 1,
       MemNameSpace, "ID",
                                        catNames[description ->
   MemValSpace, "Memory Offset");
                                            attribute], *(char**) pop(
   return 1;
                                            semErrs));
71
   }
                                        }
72
                                    95
                                    96
   void outputWidth(char* lex,
                                        void writeToken(Token* token)
       int width) {
                                        {
                                    98
   fprintf(memFile, "%*s%*d\n",
                                        // Don't bother including in
       MemNameSpace, lex,
                                            the output file.
       MemValSpace, width);
                                        if (token -> attribute == WS
   }
                                            || token -> attribute ==
76
                                            NOOP)
   void writeError(Token*
                                        return;
                                    101
       description)
                                    102
```

```
if (token -> attribute >=
                                             file (assume the latter)
        LEXERR)
                                     136 LINE++;
                                         writeEOFToken();
104
                                     137
    writeError(token);
                                         return false; // Terminate
    return;
                                     139
106
                                         }
    }
107
                                         return true; // Continue
                                     141
108
                                     142
                                         }
109
    fprintf(tokenFile, "%*d%*.*s%*
110
        d", TokenLineSpace, LINE,
                                                 Listing 30: parser.c
        TokenLexSpace,
                                         #include<stdlib.h>
    token -> length, &BUFFER[token
                                         #include<stdbool.h>
         -> start], TokenTypeSpace, 2
    token -> attribute);
                                         #include "../tokenizer/tokens.
    switch (token -> attribute) {
                                             h"
    case ID:
                                         #include "productions/
    fprintf(tokenFile, "%*p",
                                             productions.h"
        TokenAttrSpace, token -> id
                                         #include "../tokenizer/
                                             tokenizer.h"
    break;
116
                                         #include "../handler/handler.h
117
    default:
118
                                         #include "../errorHandler/
    fprintf(tokenFile, "%*d",
                                             errorHandler.h"
        TokenAttrSpace, token ->
        aspect);
                                         Token* current_tok = NULL;
   break;
120
                                      11
    fprintf(tokenFile, "\n");
                                         static bool sequence_running =
122
                                              true:
123
124
                                         Token* get_next_relevant_token
    bool handleToken(Token* token)
                                              ()
126
                                         {
    writeToken(token);
                                      15
127
                                         const Token* next = malloc(
    if (token -> attribute == WS
                                      16
                                             sizeof(*next));
        && token -> aspect == 1) //
                                         if (sequence_running)
         A newline
129
                                         do {
    char line[72];
                                      19
                                         next = getNextToken();
    if (fgets(line, sizeof(line),
                                      20
131
                                         if (!handleToken(next))
                                      21
        sourceFile) != NULL)
132
                                         sequence_running = false;
    updateLine(line);
                                      23
    fprintf(listingFile, "\%*d\t\%s" \ \ensuremath{^{24}} \ next = \&eof\_tok;
                                         break;
        , ListingLineSpace, LINE,
                                      25
        line):
                                         } while (next -> attribute ==
135 } else { // Error or end of
                                      27
```

```
WS || next -> attribute == 58
                                       current_tok =
       NOOP
                                           get_next_relevant_token();
    || next -> attribute >= LEXERR 59
                                       return prev_tok;
        );
   } else {
                                       else
29
   next = &eof_tok;
                                       throw_syn_error(current_tok, &
31
                                           source, 1);
   return next;
                                       current_tok =
33
                                           get_next_relevant_token();
   }
                                       return NULL;
35
   void require_sync(const Token*
36
        sync_set[], int size,
   const Token* first_set[], int
       first_size)
                                       bool generateParseTree()
                                    69
                                    70
38
   throw_syn_error(current_tok,
                                       current_tok = malloc(sizeof(*
       first_set, first_size);
                                           current_tok));
                                       current_tok =
40
   while (true) {
                                           get_next_relevant_token();
41
   for (int i = 0; i < size; i++) 73
                                       program();
   if (tokens_equal(sync_set[i],
                                       return match(&eof_tok, false);
                                    74
       current_tok, sync_set[i] -> 75
        start))
    return;
44
                                             Listing 31: arguments.c
45
                                       #include<stdbool.h>
   current_tok =
                                       #include<stdlib.h>
       get_next_relevant_token();
   }
                                       #include "productions.h"
   }
48
                                       #include "../parser.h"
                                       #include "../../tokenizer/
   // Attempts to match the
                                           tokens.h"
       source token with the
       current token;
                                       static const Token* first_set
   // if it is found, it returns
                                            [] = {&lparen_tok, &
       the matched token (for use
                                           semic_tok};
       in the RDP).
  // If it is not found, then
                                       static const int first_size =
                                           sizeof(first_set)/sizeof(
       match returns null.
                                           first_set[0]);
   Token* match(const Token*
       source, bool strict)
                                    10
                                       static const Token* sync_set[]
                                    11
                                             = {&eof_tok, &semic_tok};
   if (tokens_equal(source,
                                       static const int sync_size =
       current_tok, strict))
                                           sizeof(sync_set)/sizeof(
   {
56
                                           sync_set[0]);
   Token* prev_tok = current_tok;
```

```
static const Token* sync_set[]
13
   static void synch()
                                             = {&eof_tok, &assignop_tok
                                            };
15
   require_sync(sync_set,
                                        static const int sync_size =
       sync_size, first_set,
                                            sizeof(sync_set)/sizeof(
       first_size);
                                            sync_set[0]);
17
                                    13
                                        static void synch()
18
   // Needs implementing: None
19
                                    15
   void arguments()
                                        require_sync(sync_set,
                                            sync_size, first_set,
   // Production 9.1
                                            first_size);
   if (tokens_equal(&lparen_tok,
                                    17
       current_tok, true))
                                    18
   {
                                        static LangType array_compare(
24
   match(&lparen_tok, true);
                                            LangType a_vals, LangType
25
   parameter_list();
                                            e_type) {
   match(&rparen_tok, true);
                                        if ((a_vals == INT || a_vals
                                            == REAL) && e_type == INT)
   return;
                                        return a_vals;
29
   // Production 9.2
                                        if (a_vals != ERR)
   } else if (tokens_equal(&
                                        {
                                    23
       semic_tok, current_tok,
                                        char* errorMessage = calloc
       true))
                                            (100, sizeof(*errorMessage)
   return; // Epsilon
                                            );
                                        sprintf(errorMessage, "Attempt
33
                                             to index variable of type
34
   synch();
                                            %s!", typeNames[a_vals]);
   }
35
                                        throw_sem_error(errorMessage);
                                        }
                                    27
        Listing 32: array_access.c
   #include<stdbool.h>
                                        return ERR;
   #include<stdlib.h>
                                    30
                                    31
   #include "productions.h"
                                        // Needs implementing: None
                                    32
   #include "../parser.h"
                                        LangType array_access(LangType
   #include "../../tokenizer/
                                             id_type)
       tokens.h"
                                        // Production 17.1
   static const Token* first_set
                                        if (tokens_equal(&lbrac_tok,
       [] = {&assignop_tok, &
                                            current_tok, true))
       lbrac_tok};
   static const int first_size =
                                        match(&lbrac_tok, true);
       sizeof(first_set)/sizeof(
                                       LangType e_type = expression()
       first_set[0]);
                                        match(&rbrac_tok, true);
10
```

```
LangType n_type =
       convert_from_array(id_type) 20
                                        // Needs implementing: None
   return array_compare(n_type,
                                        void compound_statement()
       e_type);
                                    23
                                       // Production 11
   // Production 17.2
                                       if (tokens_equal(&begin_tok,
   } else if (tokens_equal(&
                                            current_tok, true))
       assignop_tok, current_tok,
                                    26
       true))
                                       match(&begin_tok, true);
                                       optional_statements();
   return id_type; // epsilon
46
                                       match(&end_tok, true);
                                       return;
   synch();
   return ERR;
                                       }
   }
50
                                    32
                                       synch();
                                       }
    Listing 33: compound<sub>s</sub>tatement.c
   #include<stdbool.h>
                                             Listing 34: declarations.c
   #include<stdlib.h>
                                        #include<stdbool.h>
   #include "productions.h"
                                        #include<stdlib.h>
   #include "../parser.h"
   #include "../../tokenizer/
                                       #include "productions.h"
       tokens.h"
                                       #include "../parser.h"
                                       #include "../../tokenizer/
   static const Token* first_set
                                            tokens.h"
       [] = {&begin_tok};
   static const int first_size =
                                       static const Token* first set
       sizeof(first_set)/sizeof(
                                            [] = {&var_tok, &
       first_set[0]);
                                            procedure_tok, &begin_tok};
                                       static const int first_size =
10
                                            sizeof(first_set)/sizeof(
   static const Token* sync_set[]
        = {&eof_tok, &semic_tok, &
                                            first_set[0]);
       period_tok,
   &end_tok, &else_tok};
                                       static const Token* sync_set[]
   static const int sync_size =
                                             = {&eof_tok, &
       sizeof(sync_set)/sizeof(
                                            procedure_tok, &begin_tok};
       sync_set[0]);
                                       static const int sync_size =
                                            sizeof(sync_set)/sizeof(
14
   static void synch()
                                            sync_set[0]);
16
   require_sync(sync_set,
                                       static void synch()
                                    14
       sync_size, first_set,
                                    15
       first_size);
                                       require_sync(sync_set,
18 }
                                            sync_size, first_set,
```

```
first_size);
                                        static const Token* first_set
   }
                                            [] = {&else_tok, &semic_tok
17
                                            , &end_tok, &else_tok};
18
   // Needs implementing: None
                                        static const int first_size =
   void declarations()
                                            sizeof(first_set)/sizeof(
                                            first_set[0]);
   // Production 3.1
                                    10
   if (tokens_equal(&var_tok,
                                        static const Token* sync_set[]
       current_tok, true))
                                             = {&eof_tok, &semic_tok, &
                                            end_tok, &else_tok};
                                        static const int sync_size =
   match(&var_tok, true);
   Token* id_ref = match(&id_tok,
                                            sizeof(sync_set)/sizeof(
        false);
                                            sync_set[0]);
   match(&colon_tok, true);
   if (id_ref != NULL) {
                                        static void synch()
                                    14
   id_ref -> type = type(id_ref);
                                    15
   id_ref -> param = false;
                                        require_sync(sync_set,
   check_add_node(id_ref);
                                            sync_size, first_set,
   } else {
                                            first_size);
   type(NULL);
                                        }
                                    17
33
   match(&semic_tok, true);
                                        // Needs implementing: None
                                    19
35
   declarations();
                                        void else_tail()
   return;
                                        {
37
                                        // Production 15.1
   // Production 3.2
                                        if (tokens_equal(&else_tok,
39
                                            current_tok, true)) // else
   } else if (tokens_equal(&
       begin_tok, current_tok,
                                    24
                                        match(&else_tok, true);
       true)
    || tokens_equal(&procedure_tok 26
                                        statement();
41
        , current_tok, true))
                                        return;
    return; // epsilon
42
                                        // Production 15.2
43
   synch();
                                        } else if (tokens_equal(&
44
                                            end_tok, current_tok, true)
45
                                             // end
                                         || tokens_equal(&semic_tok,
                                    31
          Listing 35: {\it else}_t ail.c
                                             current_tok, true)) // ;
   #include<stdbool.h>
                                        return; // epsilon
   #include<stdlib.h>
                                        synch();
                                    34
   #include "productions.h"
                                        }
                                    35
   #include "../parser.h"
   #include "../../tokenizer/
                                              Listing 36: expression.c
       tokens.h"
                                        #include<stdbool.h>
```

```
#include<stdlib.h>
                                       LangType s_type =
   #include "productions.h"
                                            simple_expression();
   #include "../parser.h"
                                       return related_expression(
   #include "../../tokenizer/
                                            s_type);
       tokens.h"
                                       }
   static const Token* first_set
                                       synch();
       [] = {&id_tok, &num_tok, &
                                    37
                                       return ERR:
       lparen_tok, &not_tok,
                                       }
    &plus_tok, &minus_tok};
   static const int first_size =
                                            Listing 37: expression_list.c
       sizeof(first_set)/sizeof(
                                       #include<stdbool.h>
       first_set[0]);
                                        #include<stdlib.h>
11
   static const Token* sync_set[]
                                       #include "productions.h"
        = {&eof_tok, &semic_tok, & 4
                                       #include "../parser.h"
       end_tok, &else_tok,
                                        #include "../../tokenizer/
   &do_tok, &then_tok, &rbrac_tok 6
                                            tokens.h"
       , &rparen_tok,
   &comma_tok};
                                       static const Token* first_set
   static const int sync_size =
                                            [] = {&id_tok, &num_tok, &
       sizeof(sync_set)/sizeof(
                                           lparen_tok, &not_tok,
       sync_set[0]);
                                        &plus_tok, &minus_tok};
16
                                        static const int first_size =
   static void synch()
17
                                            sizeof(first_set)/sizeof(
                                           first_set[0]);
   require_sync(sync_set,
       sync_size, first_set,
                                    11
                                        static const Token* sync_set[]
       first_size);
                                             = {&eof_tok, &rparen_tok};
   }
20
                                        static const int sync_size =
21
                                            sizeof(sync_set)/sizeof(
   // Needs implementing: None
                                            sync_set[0]);
   LangType expression()
                                    14
24
                                       static void synch()
   // Production 21
                                    15
   if (tokens_equal(&lparen_tok,
                                    16
                                       require_sync(sync_set,
       current_tok, true)
                                    17
                                            sync_size, first_set,
   || tokens_equal(&addop_tok,
                                            first_size);
       current_tok, false)
                                       }
                                    18
   || tokens_equal(&id_tok,
       current_tok, false)
                                    19
   || tokens_equal(&not_tok,
                                       // Needs implementing: None
                                       void expression_list(tree_node
       current_tok, true)
                                            * to_match, bool
   || tokens_equal(&num_tok,
                                           should_error)
       current_tok, false))
```

```
{
                                        }
   // Production 20.1
                                        synch();
   if (tokens_equal(&lparen_tok,
                                    49
       current_tok, true)
                                        }
                                    50
   || tokens_equal(&addop_tok,
25
       current_tok, false) // + OR
                                           Listing 38: expression<sub>l</sub>ist_tail.c
                                        #include<stdbool.h>
   || tokens_equal(&id_tok,
                                        #include<stdlib.h>
       current_tok, false) // ID
   || tokens_equal(&not_tok,
                                        #include "productions.h"
       current_tok, true)
                                        #include "../parser.h"
   || tokens_equal(&num_tok,
28
                                        #include "../../tokenizer/
       current_tok, false)) // num 6
                                            tokens.h"
29
   char* errorMessage;
                                        static const Token* first_set
   if (to_match == NULL &&
                                            [] = {&comma_tok, &
       should_error)
                                            rparen_tok};
32
                                        static const int first_size =
   errorMessage= calloc(100,
                                            sizeof(first_set)/sizeof(
       sizeof(*errorMessage));
                                            first_set[0]);
   sprintf(errorMessage, "Attempt
        to pass extraneous
                                        static const Token* sync_set[]
       parameter!");
                                             = {&eof_tok, &rparen_tok};
   throw_sem_error(errorMessage);
35
                                        static const int sync_size =
   }
                                            sizeof(sync_set)/sizeof(
   LangType e_type = expression()
37
                                            sync_set[0]);
   if (should_error && to_match
                                        static void synch()
       != NULL && to_match ->
                                    14
       param && e_type != ERR &&
                                    15
                                        require_sync(sync_set,
       e_type != to_match -> type)
                                    16
        {
                                            sync_size, first_set,
                                            first_size);
   errorMessage= calloc(100,
39
       sizeof(*errorMessage));
   sprintf(errorMessage, "
40
                                        // Needs implementing: None
       Expected type %s, not %s!",
                                        void expression_list_tail(
   typeNames[to_match -> type],
41
                                            tree_node* to_match, bool
       typeNames[e_type]);
                                            should_error)
   throw_sem_error(errorMessage);
42
   }
                                    21
                                        char* errorMessage;
   expression_list_tail(to_match
                                        // Production 20.2.1
       == NULL || !to_match ->
                                    23
                                        if (tokens_equal(&comma_tok,
       param ? NULL :
                                            current_tok, true))
   to_match -> left, e_type !=
       ERR && should_error);
                                    25
                                        match(&comma_tok, true);
  return;
```

```
if (to_match == NULL &&
       should_error)
                                       return; // epsilon
   errorMessage = calloc(100,
       sizeof(*errorMessage));
                                       synch();
                                    56
   sprintf(errorMessage, "Attempt
                                       }
        to pass extraneous
       parameters!");
                                               Listing 39: factor.c
   throw_sem_error(errorMessage);
31
                                       #include<stdbool.h>
   }
   LangType e_type = expression()
                                       #include<stdlib.h>
                                       #include "productions.h"
   if (should_error && to_match
                                       #include "../parser.h"
       != NULL && e_type != ERR && 5
                                       #include "../../tokenizer/
        e_type != to_match -> type 6
                                           tokens.h"
   errorMessage = calloc(100,
                                       static const Token* first_set
       sizeof(*errorMessage));
                                            [] = {&id_tok, &num_tok, &
   sprintf(errorMessage, "
                                           lparen_tok, &not_tok);
       Expected type %s, not %s!",
                                       static const int first_size =
   typeNames[to_match -> type],
                                           sizeof(first_set)/sizeof(
       typeNames[e_type]);
                                           first_set[0]);
   throw_sem_error(errorMessage);
   }
39
                                       static const Token* sync_set[]
   expression_list_tail(to_match
                                            = {&eof_tok, &mulop_tok, &
       == NULL || !to_match ->
                                           addop_tok, &relop_tok,
       param ? NULL :
                                       &semic_tok, &end_tok, &
   to_match -> left, e_type !=
41
                                           else_tok, &do_tok,
       ERR && should_error);
                                       &then_tok, &rbrac_tok, &
   return;
42
                                           rparen_tok,
43
   // Production 20.2.2
                                       &comma_tok};
                                       static const int sync_size =
   } else if (tokens_equal(&
                                           sizeof(sync_set)/sizeof(
       rparen_tok, current_tok,
                                           sync_set[0]);
       true))
   {
                                    16
                                       static void synch()
   if (to_match != NULL &&
                                    17
47
       to_match -> param &&
                                    18
                                       require_sync(sync_set,
       should_error) {
                                           sync_size, first_set,
   errorMessage = calloc(100,
                                           first_size);
       sizeof(*errorMessage));
   sprintf(errorMessage, "
                                    20
       Expected %s, not the end of 21
                                       // Needs implementing: 25.1.2
        the parameters!",
                                       LangType factor()
   typeNames[to_match -> type]);
   throw_sem_error(errorMessage);
                                    24
```

```
// Production 25.1.1
                                       match(&not_tok, true);
   if (tokens_equal(&id_tok,
                                       LangType f_type = factor();
       current_tok, false)) { //
                                       return type_lookup(f_type, INT
                                    57
       id
                                            , &not_tok);
   char* errorMessage;
27
                                    58
   Token* id_ref = match(&id_tok,
        false); // id
   LangType id_type = get_type(
                                        synch();
       id_ref);
                                       return ERR;
   if (id_type == NULL) {
                                       }
   errorMessage = calloc(100,
31
       sizeof(*errorMessage));
                                              Listing 40: factor<sub>t</sub> ail.c
   sprintf(errorMessage,
32
                                       #include<stdbool.h>
    "No variable '%.*s' is defined 1
33
         in the local scope!",
                                       #include "productions.h"
    id_ref -> length, &BUFFER[
34
                                       #include "../parser.h"
        id_ref -> start]);
                                        #include "../../tokenizer/
    throw_sem_error(errorMessage); 5
35
                                            tokens.h"
    id_type = ERR;
37
                                       static const Token* first_set
   return factor_tail(id_type);
                                            [] = {&lbrac_tok, &
39
                                           mulop_tok, &addop_tok, &
   // Production 25.1.2
                                           relop_tok,
   } else if (tokens_equal(&
                                        &semic_tok, &end_tok, &
       num_tok, current_tok, false 8
                                             else_tok, &do_tok,
       )) { // num
                                        &then_tok, &rbrac_tok, &
   Token* num_type;
                                            rparen_tok,
   num_type = match(&num_tok,
                                        &comma_tok};
       false);
                                        static const int first_size =
   return num_type -> aspect == 0
                                    11
                                            sizeof(first_set)/sizeof(
        ? INT : REAL;
                                            first_set[0]);
45
   // Production 25.1.3
                                       static const Token* sync_set[]
   } else if (tokens_equal(&
                                    13
                                             = {&eof_tok, &mulop_tok, &
       lparen_tok, current_tok,
                                            addop_tok, &relop_tok,
       true)) { // (
                                        &semic_tok, &end_tok, &
   match(&lparen_tok, true);
                                            else_tok, &do_tok,
   LangType e_type = expression()
                                       &then_tok, &rbrac_tok, &
                                           rparen_tok,
   match(&rparen_tok, true); // )
                                       &comma_tok};
   return e_type;
51
                                        static const int sync_size =
                                            sizeof(sync_set)/sizeof(
   // Production 25.1.4
                                            sync_set[0]);
   } else if (tokens_equal(&
       not_tok, current_tok, true) 18
                                       static void synch()
       ) { // not
```

```
{
                                        || tokens_equal(&comma_tok,
20
   require_sync(sync_set,
                                            current_tok, true)
                                        || tokens_equal(&semic_tok,
       sync_size, first_set,
       first_size);
                                            current_tok, true)
                                        || tokens_equal(&rbrac_tok,
22
                                            current_tok, true)
   static LangType array_compare(
                                        || tokens_equal(&addop_tok,
       LangType a_vals, LangType
                                            current_tok, false)
       e_type) {
                                        || tokens_equal(&do_tok,
   if ((a_vals == INT || a_vals
                                            current_tok, true)
       == REAL) && e_type == INT)
                                           tokens_equal(&else_tok,
   return a_vals;
                                            current_tok, true)
                                        || tokens_equal(&end_tok,
   if (a_vals != ERR)
27
                                            current_tok, true)
28
   char* errorMessage = calloc
                                        || tokens_equal(&mulop_tok,
       (100, sizeof(*errorMessage)
                                            current_tok, false)
       );
                                        || tokens_equal(&relop_tok,
   sprintf(errorMessage, "Attempt
                                            current_tok, false)
        to index variable of type
                                        || tokens_equal(&then_tok,
       %s!", typeNames[a_vals]);
                                            current_tok, true))
   throw_sem_error(errorMessage);
                                        return id_type; // epsilon
   }
32
                                        synch();
   return ERR;
                                       return ERR;
34
                                       }
35
   // Needs implementing: None
                                                Listing 41: id_l ist.c
   LangType factor_tail(id_type)
38
                                        #include<stdbool.h>
   {
                                        #include<stdlib.h>
   // Production 25.2.1
   if (tokens_equal(&lbrac_tok,
                                        #include "productions.h"
       current_tok, true)) {
                                        #include "../parser.h"
   match(&lbrac_tok, true);
42
                                        #include "../../tokenizer/
   LangType e_type = expression()
                                            tokens.h"
   match(&rbrac_tok, true);
                                        static const Token* first_set
   LangType n_type =
45
                                            [] = {&id_tok};
       convert_from_array(id_type)
                                        static const int first_size =
                                            sizeof(first_set)/sizeof(
   return array_compare(n_type,
                                           first_set[0]);
       e_type);
                                    10
   // Production 25.2.2
                                        static const Token* sync_set[]
                                             = {&eof_tok, &rparen_tok};
   } else if (tokens_equal(&
                                        static const int sync_size =
       rparen_tok, current_tok,
                                            sizeof(sync_set)/sizeof(
       true)
```

```
sync_set[0]);
                                     10
                                        static const Token* sync_set[]
13
   static void synch()
                                              = {&eof_tok, &rparen_tok};
14
                                         static const int sync_size =
   require_sync(sync_set,
                                             sizeof(sync_set)/sizeof(
16
                                             sync_set[0]);
       sync_size, first_set,
       first_size);
   }
                                         static void synch()
17
18
                                     15
   // Needs implementing: None
                                         require_sync(sync_set,
   void id_list()
                                             sync_size, first_set,
20
                                             first_size);
   Token* id_ref;
                                        }
                                     17
   // Production 2.1
                                     18
   if (tokens_equal(&id_tok,
                                        // Needs implementing: None
                                     19
       current_tok, false)) {
                                     20
                                         void id_list_tail()
   id_ref = match(&id_tok, false)
                                         {
                                     21
                                         // Production 2.2.1
   if (id_ref != NULL) {
                                         if (tokens_equal(&comma_tok,
   id_ref -> type = PPNAME;
                                             current_tok, true))
   id_ref -> param = true;
                                     24
   check_add_node(id_ref);
                                        match(&comma_tok, true);
29
                                     25
   }
                                        Token* id_ref;
   id_list_tail();
                                     27
   return;
                                         id_ref = match(&id_tok, false)
33
                                         if (id_ref != NULL) {
                                         id_ref -> type = PPNAME;
   synch();
35
                                         id_ref -> param = true;
   }
36
                                         check_add_node(id_ref);
                                     32
                                        }
                                     33
          Listing 42: id_l ist_t ail.c
   #include<stdbool.h>
                                        id_list_tail();
                                     35
   #include<stdlib.h>
                                        return;
                                     36
   #include "productions.h"
                                        // Production 2.2.2
   #include "../parser.h"
                                        } else if (tokens_equal(&
   #include "../../tokenizer/
                                             rparen_tok, current_tok,
       tokens.h"
                                             true))
                                         return; // Epsilon
   static const Token* first_set
                                     41
       [] = {&comma_tok, &
                                         synch();
                                     42
       rparen_tok};
                                        }
                                     43
   static const int first_size =
       sizeof(first_set)/sizeof(
                                          Listing 43: optional<sub>e</sub>xpressions.c
       first_set[0]);
```

```
#include<stdbool.h>
                                        // Production 19.2
                                        } else if (tokens_equal(&
   #include<stdlib.h>
                                            semic_tok, current_tok,
   #include "productions.h"
                                            true)
   #include "../parser.h"
                                        || tokens_equal(&else_tok,
   #include "../../tokenizer/
                                            current_tok, true)
       tokens.h"
                                        || tokens_equal(&end_tok,
                                            current_tok, true))
   static const Token* first_set
                                        {
                                    36
       [] = {&lparen_tok, &
                                        if (to_match != NULL &&
                                            should_error) {
       semic_tok, &end_tok,
                                        errorMessage= calloc(100,
    &else_tok};
   static const int first_size =
                                            sizeof(*errorMessage));
                                        sprintf(errorMessage, "
       sizeof(first_set)/sizeof(
       first_set[0]);
                                            Expected an argument of
                                            type %s!",
11
   static const Token* sync_set[]
                                        typeNames[to_match -> type]);
        = {&eof_tok, &semic_tok, & 41
                                        throw_sem_error(errorMessage);
       end_tok, &else_tok};
   static const int sync_size =
                                    43
       sizeof(sync_set)/sizeof(
                                       return; // epsilon
       sync_set[0]);
                                        }
                                    45
14
   static void synch()
                                        synch();
15
                                    47
                                        }
   require_sync(sync_set,
       sync_size, first_set,
                                          Listing 44: optional<sub>s</sub> tatements.c
       first_size);
                                        #include<stdbool.h>
   }
18
                                        #include<stdlib.h>
19
   // Needs implementing: None
                                        #include "productions.h"
   void optional_expressions(
                                        #include "../parser.h"
       tree_node* to_match, bool
                                        #include "../../tokenizer/
       should_error)
                                            tokens.h"
   {
22
   char* errorMessage;
                                        static const Token* first_set
   // Production 19.1
                                            [] = {&id_tok, &call_tok, &
   if (tokens_equal(&lparen_tok,
                                            begin_tok, &while_tok,
       current_tok, true))
                                         &if_tok, &end_tok, &array_tok
                                             };
   match(&lparen_tok, true);
27
                                        static const int first_size =
   expression_list(to_match,
                                    10
                                            sizeof(first_set)/sizeof(
       should_error);
                                            first_set[0]);
   match(&rparen_tok, true);
   return;
30
                                        static const Token* sync_set[]
31
```

```
= {&eof_tok, &end_tok};
                                        #include "../parser.h"
                                        #include "../../tokenizer/
   static const int sync_size =
       sizeof(sync_set)/sizeof(
                                            tokens.h"
       sync_set[0]);
                                       static const Token* first_set
14
   static void synch()
                                            [] = {&id_tok};
15
   {
                                       static const int first_size =
16
                                            sizeof(first_set)/sizeof(
   require_sync(sync_set,
       sync_size, first_set,
                                            first_set[0]);
       first_size);
   }
                                       static const Token* sync_set[]
18
                                             = {&eof_tok, &rparen_tok};
19
   // Needs implementing: None
                                        static const int sync_size =
   void optional_statements()
                                            sizeof(sync_set)/sizeof(
                                            sync_set[0]);
   {
22
   // Production 12.1
                                    13
   if (tokens_equal(&begin_tok,
                                       static void synch()
       current_tok, true) // begin 15
   || tokens_equal(&call_tok,
                                       require_sync(sync_set,
                                    16
       current_tok, true) // call
                                            sync_size, first_set,
                                            first_size);
   || tokens_equal(&id_tok,
       current_tok, false) // ID
                                    17
   || tokens_equal(&if_tok,
                                       // Needs implementing: None
       current_tok, true) // if
   || tokens_equal(&while_tok,
                                        void parameter_list()
       current_tok, true)) //
                                       {
                                    21
       while
                                       // Production 10.1
   {
                                        if (tokens_equal(&id_tok,
29
   statement_list();
                                            current_tok, false)) {
   return;
                                       Token* id_ref;
31
                                       id_ref = match(&id_tok, false)
   // Production 12.2
                                            ; // ID
   } else if (tokens_equal(&
                                       match(&colon_tok, true);
                                    26
       end_tok, current_tok, true) 27
                                       if (id_ref != NULL) {
       ) // end
                                        id_ref -> param = true;
   return; // epsilon
                                       id_ref -> type = type(id_ref);
35
                                       check_add_node(id_ref);
36
                                       } else {
   synch();
                                        type(NULL);
38
                                    32
                                       }
                                       parameter_list_tail();
                                    34
       Listing 45: parameter list.c
                                    35
                                       return;
   #include<stdbool.h>
                                    36
   #include<stdlib.h>
                                       synch();
                                    38
   #include "productions.h"
                                       }
```

```
check_add_node(id_ref);
                                        } else {
                                        type(NULL);
      Listing 46: parameter list_t ail.c
                                    33
   #include<stdbool.h>
                                        parameter_list_tail();
                                    35
   #include<stdlib.h>
                                        return;
                                    37
   #include "productions.h"
                                        // Production 10.2.2
   #include "../parser.h"
                                        } else if (tokens_equal(&
   #include "../../tokenizer/
                                            rparen_tok, current_tok,
       tokens.h"
                                            true)) // )
                                        return; // epsilon
                                    40
   static const Token* first_set
                                    41
       [] = {\&semic\_tok, \&}
                                    42
                                        synch();
       rparen_tok};
                                        }
                                    43
   static const int first_size =
       sizeof(first_set)/sizeof(
       first_set[0]);
                                         Listing 47: procedure statement.c
                                        #include<stdbool.h>
   static const Token* sync_set[]
11
                                        #include<stdlib.h>
        = {&eof_tok, &rparen_tok};
   static const int sync_size =
                                        #include "productions.h"
       sizeof(sync_set)/sizeof(
                                        #include "../parser.h"
       sync_set[0]);
                                        #include "../../tokenizer/
13
                                            tokens.h"
   static void synch()
14
                                        static const Token* first_set
   require_sync(sync_set,
                                            [] = {&call_tok};
       sync_size, first_set,
                                        static const int first_size =
       first_size);
                                            sizeof(first_set)/sizeof(
   }
17
                                            first_set[0]);
18
   // Needs implementing: None
                                        static const Token* sync_set[]
   void parameter_list_tail()
                                             = {&eof_tok, &semic_tok, &
   {
21
                                            end_tok, &else_tok};
   // Production 10.2.1
                                        static const int sync_size =
   if (tokens_equal(&semic_tok,
                                            sizeof(sync_set)/sizeof(
       current_tok, true)) //;
                                            sync_set[0]);
24
   match(&semic_tok, true); // ;
                                        static void synch()
   Token* id_ref = match(&id_tok,
                                       {
        false); // ID
                                        require_sync(sync_set,
                                    16
   match(&colon_tok, true); // :
                                            sync_size, first_set,
   if (id_ref != NULL) {
                                            first_size);
   id_ref -> param = true;
                                       }
   id_ref -> type = type(id_ref);
```

```
// Needs implementing: None
                                       #include<stdlib.h>
   void procedure_statement()
                                       #include<stdio.h>
21
                                       #include "productions.h"
   char* errorMessage;
   // Production 18
                                       #include "../parser.h"
                                       #include "../../tokenizer/
   if (tokens_equal(&call_tok,
                                           tokens.h"
       current_tok, true)) // call
25
   Token* id_ref;
                                       static const Token* first_set
26
   match(&call_tok, true); //
                                            [] = {&program_tok};
       call
                                       static const int first_size =
                                           sizeof(first_set)/sizeof(
   id_ref = match(&id_tok, false)
28
                                           first_set[0]);
   if (id_ref != NULL) {
                                    11
   tree_node* addition =
                                       static const Token* sync_set[]
       start_param_matching(id_ref
                                            = {&eof_tok};
       );
                                       static const int sync_size =
   if (addition == NULL) {
                                           sizeof(sync_set)/sizeof(
   errorMessage= calloc(100,
                                           sync_set[0]);
       sizeof(*errorMessage));
                                    14
   sprintf(errorMessage, "
                                       static void synch()
       Procedure '%s' not in scope 16
                                       {
       !", id_ref -> id);
                                       require_sync(sync_set,
   throw_sem_error(errorMessage);
                                           sync_size, first_set,
34
                                           first_size);
35
   optional_expressions(NULL,
36
                                    18
       false);
   } else
                                       // Needs implementing: None
                                    20
37
                                       void program()
    optional_expressions(addition
        -> left == NULL ? NULL :
                                       {
                                    22
        addition -> left -> param
                                    23
                                       Token* id_ref;
        ? addition -> left : NULL, 24
                                       // Production 1
         true);
                                       if (tokens_equal(&program_tok,
   } else {
                                            current_tok, true)) {
   optional_expressions(NULL,
                                       match(&program_tok, true); //
                                    26
       false);
                                           program
                                       id_ref = match(&id_tok, false)
41
                                            ; // id
   return;
                                       if (id_ref != NULL) {
   }
43
                                        id_ref -> type = PGNAME;
   synch();
                                       id_ref -> param = false;
45
                                    30
                                       check_add_node(id_ref);
                                    32
                                       match(&lparen_tok, true); // (
          Listing 48: program.c
                                       id_list();
                                    34
   #include<stdbool.h>
                                       match(&rparen_tok, true); // )
```

```
match(&semic_tok, true); //;
                                            first_size);
   declarations();
   subprogram_declarations();
                                    23
   compound_statement();
                                       // Needs implementing: None
   match(&period_tok, true); // .
                                        LangType related_expression(
                                    25
   return;
                                            LangType s_type)
   }
42
                                    26
                                        // Production 22.1
43
                                        if (tokens_equal(&relop_tok,
   synch();
44
                                            current_tok, false)) {
   }
                                       Token* relop_op;
                                    29
                                        relop_op = match(&relop_tok,
      Listing 49: related expression.c
                                            false);
   #include<stdbool.h>
                                       LangType s1_type =
   #include<stdlib.h>
                                            simple_expression();
                                       return type_lookup(s_type,
   #include "productions.h"
                                            s1_type, relop_op);
   #include "../parser.h"
   #include "../../tokenizer/
                                       // Production 22.2
       tokens.h"
                                       } else if (tokens_equal(&
                                            rparen_tok, current_tok,
   static const Token* first_set
                                            true)
       [] = {\&relop\_tok,}
                                        || tokens_equal(&comma_tok,
    &semic_tok, &end_tok, &
                                            current_tok, true)
        else_tok, &do_tok,
                                        || tokens_equal(&semic_tok,
    &then_tok, &rbrac_tok, &
                                            current_tok, true)
        rparen_tok,
                                        || tokens_equal(&rbrac_tok,
    &comma_tok};
                                            current_tok, true)
   static const int first_size =
                                        || tokens_equal(&do_tok,
       sizeof(first_set)/sizeof(
                                            current_tok, true)
       first_set[0]);
                                        || tokens_equal(&else_tok,
                                            current_tok, true)
13
   static const Token* sync_set[]
                                        || tokens_equal(&end_tok,
        = {&eof_tok, &semic_tok, &
                                            current_tok, true)
       end_tok,
                                        || tokens_equal(&then_tok,
   &else_tok, &do_tok, &then_tok,
                                            current_tok, true))
        &rbrac_tok,
                                       return s_type; // epsilon
                                    43
   &rparen_tok, &comma_tok};
                                    44
   static const int sync_size =
                                        synch();
       sizeof(sync_set)/sizeof(
                                       return ERR;
       sync_set[0]);
                                    47
                                       }
18
   static void synch()
19
                                                Listing 50: sign.c
20
   require_sync(sync_set,
                                        #include<stdbool.h>
       sync_size, first_set,
                                       #include<stdlib.h>
```

```
#include "productions.h"
   #include "../parser.h"
                                          Listing 51: simple_expression.c
   #include "../../tokenizer/
                                       #include<stdbool.h>
       tokens.h"
                                        #include<stdlib.h>
   static const Token* first_set
                                        #include "productions.h"
       [] = {&plus_tok, &minus_tok
                                        #include "../parser.h"
                                        #include "../../tokenizer/
   static const int first_size =
                                           tokens.h"
       sizeof(first_set)/sizeof(
       first_set[0]);
                                        static const Token* first_set
10
                                            [] = {&id_tok, &num_tok, &
   static const Token* sync_set[]
                                           lparen_tok, &not_tok,
        = {&eof_tok, &id_tok, &
                                        &plus_tok, &minus_tok);
       num_tok,
                                        static const int first_size =
   &not_tok, &rparen_tok};
                                            sizeof(first_set)/sizeof(
   static const int sync_size =
13
                                           first_set[0]);
       sizeof(sync_set)/sizeof(
                                    11
       sync_set[0]);
                                       static const Token* sync_set[]
                                             = {&eof_tok, &relop_tok, &
   static void synch()
15
                                            semic_tok,
   {
                                        &end_tok, &else_tok, &do_tok,
   require_sync(sync_set,
                                            &then_tok,
       sync_size, first_set,
                                       &rbrac_tok, &rparen_tok, &
       first_size);
                                            comma_tok};
   }
18
                                        static const int sync_size =
19
                                            sizeof(sync_set)/sizeof(
   // Needs implementing: None
                                            sync_set[0]);
   void sign()
                                    16
   {
                                       static void synch()
                                    17
   // Production 24.2.1
                                    18
   if (tokens_equal(&plus_tok,
                                       require_sync(sync_set,
       current_tok, true)) {
                                            sync_size, first_set,
   match(&plus_tok, true);
25
                                           first_size);
   return;
26
                                    20
27
                                    21
   // Production 24.2.2
                                       // Needs implementing: None
                                    22
   } else if (tokens_equal(&
                                       LangType simple_expression()
       minus_tok, current_tok,
                                    24
       true)) {
                                        char* errorMessage;
   match(&minus_tok, true);
                                        // Production 23.1.1
   return; // epsilon
                                        if (tokens_equal(&lparen_tok,
   }
                                            current_tok, true)
   synch();
                                        || tokens_equal(&id_tok,
   }
```

```
current_tok, false)
   || tokens_equal(&not_tok,
                                       static const Token* first_set
       current_tok, true)
                                            [] = {&addop_tok, &
   || tokens_equal(&num_tok,
                                           relop_tok,
       current_tok, false))
                                        &semic_tok, &end_tok, &
                                            else_tok, &do_tok,
31
   LangType t_type = term();
                                         &then_tok, &rbrac_tok, &
32
   return simple_expression_tail(
                                            rparen_tok,
       t_type);
                                        &comma_tok};
                                        static const int first_size =
   // Production 23.1.2
                                            sizeof(first_set)/sizeof(
   } else if (tokens_equal(&
                                            first_set[0]);
       plus_tok, current_tok, true 13
                                       static const Token* sync_set[]
                                            = {&eof_tok, &relop_tok, &
    || tokens_equal(&minus_tok,
        current_tok, true)) {
                                            semic_tok,
   sign();
                                       &end_tok, &else_tok, &do_tok,
38
   LangType t_type = term();
                                            &then_tok,
   if (t_type != INT && t_type !=
                                       &rbrac_tok, &rparen_tok, &
        REAL && t_type != ERR)
                                            comma_tok};
                                       static const int sync_size =
41
   errorMessage= calloc(100,
                                            sizeof(sync_set)/sizeof(
       sizeof(*errorMessage));
                                            sync_set[0]);
   sprintf(errorMessage, "
                                    18
       Expected number for use
                                       static void synch()
       with sign, not %s!",
                                    20
   typeNames[t_type]);
                                       require_sync(sync_set,
   throw_sem_error(errorMessage);
                                           sync_size, first_set,
                                            first_size);
   return simple_expression_tail( 22
       t_type);
   }
                                       // Needs implementing: None
48
                                       LangType
49
   synch();
                                            simple_expression_tail(
   return ERR;
                                            LangType t_type)
51
                                       {
   }
                                    26
52
                                       // Production 23.2.1
                                    27
                                        if (tokens_equal(&addop_tok,
    Listing 52: simple_expression_tail.c
                                            current_tok, false)) {
   #include<stdbool.h>
                                       Token* addop_op;
   #include<stdlib.h>
                                        addop_op = match(&addop_tok,
                                           false);
   #include "productions.h"
                                       LangType t_type2 = term();
   #include "../parser.h"
                                       return simple_expression_tail(
   #include "../../tokenizer/
                                            type_lookup(t_type, t_type2
       tokens.h"
                                            , addop_op));
```

```
= {&eof_tok, &semic_tok, &
33
                                            rparen_tok};
34
   // Production 23.2.2
                                        static const int sync_size =
   } else if (tokens_equal(&
                                            sizeof(sync_set)/sizeof(
       rparen_tok, current_tok,
                                            sync_set[0]);
       true)
   || tokens_equal(&comma_tok,
                                        static void synch()
37
                                    14
       current_tok, true)
                                        require_sync(sync_set,
   || tokens_equal(&semic_tok,
                                    16
                                            sync_size, first_set,
       current_tok, true)
   || tokens_equal(&rbrac_tok,
                                            first_size);
39
                                        }
       current_tok, true)
                                     17
   || tokens_equal(&do_tok,
40
                                        // Needs implementing: None
       current_tok, true)
                                    19
   || tokens_equal(&else_tok,
                                        LangType standard_type()
                                    20
       current_tok, true)
   || tokens_equal(&end_tok,
                                       // Production 5.1
       current_tok, true)
                                        if (tokens_equal(&integer_tok,
                                             current_tok, true)) //
   || tokens_equal(&relop_tok,
       current_tok, false)
                                            integer
   || tokens_equal(&then_tok,
       current_tok, true))
                                        match(&integer_tok, true);
                                    25
   return t_type; // epsilon
                                        return INT;
45
46
                                    27
   synch();
                                        // Production 5.2
   return ERR;
                                        } else if (tokens_equal(&
                                            real_tok, current_tok, true
                                            )) { // real
                                        match(&real_tok, true);
        Listing 53: standard<sub>t</sub>ype.c
                                        return REAL;
                                    31
   #include<stdbool.h>
                                        }
                                    32
   #include<stdlib.h>
                                    33
                                        synch();
                                    34
   #include "productions.h"
                                        return ERR;
                                    35
   #include "../parser.h"
   #include "../../tokenizer/
       tokens.h"
                                              Listing 54: statement.c
   static const Token* first_set
                                        #include<stdbool.h>
       [] = {&integer_tok, &
                                        #include<stdlib.h>
       real_tok};
   static const int first_size =
                                        #include "productions.h"
                                        #include "../parser.h"
       sizeof(first_set)/sizeof(
       first_set[0]);
                                        #include "../../tokenizer/
                                            tokens.h"
   static const Token* sync_set[]
```

```
static const Token* first_set
                                       } else if (v_type != ERR &&
                                   38
       [] = {&id_tok, &call_tok, &
                                           v_type != INT && v_type !=
       begin_tok, &while_tok,
                                           REAL)
    &if_tok};
                                       {
   static const int first_size =
                                       errorMessage = calloc(100,
                                   40
       sizeof(first_set)/sizeof(
                                           sizeof(*errorMessage));
       first_set[0]);
                                       sprintf(errorMessage, "Cannot
                                           assign to ID '%s' of type
11
   static const Token* sync_set[]
                                           '%s'!",
        = {&eof_tok, &semic_tok, & 42
                                       id_ref -> id, typeNames[v_type
       end_tok, &else_tok};
                                           ]);
   static const int sync_size =
                                       throw_sem_error(errorMessage);
13
       sizeof(sync_set)/sizeof(
                                       expression();
       sync_set[0]);
                                       } else {
                                       LangType e_type = expression()
   static void synch()
15
   {
                                       type_lookup(v_type, e_type, &
16
   require_sync(sync_set,
                                           assignop_tok);
       sync_size, first_set,
                                       }
       first_size);
                                       return;
                                    49
   }
18
                                       // Production 14.2
                                    51
19
                                       } else if (tokens_equal(&
   // Needs implementing: None
   void statement()
                                           call_tok, current_tok, true
                                           )) { // call
   char* errorMessage;
                                       procedure_statement();
   // Production 14.1
                                       return;
   if (tokens_equal(&id_tok,
       current_tok, false)) { //
                                       // Production 14.3
       id
                                       } else if (tokens_equal(&
   Token* id_ref = current_tok;
                                           begin_tok, current_tok,
   LangType v_type = variable();
                                           true)) { // begin
   match(&assignop_tok, true);
                                       compound_statement();
                                       return;
   if (get_type(id_ref) == ERR)
30
   // The only way for this to
                                       // Production 14.4
       error is an undeclared
                                       } else if (tokens_equal(&
       variable
                                           while_tok, current_tok,
                                           true)) { // while
32
   errorMessage = calloc(100,
                                       match(&while_tok, true); //
       sizeof(*errorMessage));
                                           while
   sprintf(errorMessage, "ID '%s'
                                       LangType e_type = expression()
        not in scope!",
   id_ref -> id);
                                       if (e_type != BOOL && e_type
   throw_sem_error(errorMessage);
                                           !=ERR)
   expression();
```

```
errorMessage= calloc(100,
                                       #include "productions.h"
       sizeof(*errorMessage));
                                       #include "../parser.h"
   sprintf(errorMessage, "
                                       #include "../../tokenizer/
       Expression in while must be
                                           tokens.h"
        boolean, not %s!",
                                       static const Token* first_set
   typeNames[e_type]);
                                            [] = {&id_tok, &call_tok, &
   throw_sem_error(errorMessage);
                                           begin_tok, &while_tok,
   }
                                        &if_tok};
   match(&do_tok, true);
                                        static const int first_size =
   statement();
                                           sizeof(first_set)/sizeof(
   return;
                                           first_set[0]);
   // Production 14.5
                                       static const Token* sync_set[]
   } else if (tokens_equal(&
                                             = {&eof_tok, &end_tok};
       if_tok, current_tok, true)) 12 static const int sync_size =
        \{ // if
                                           sizeof(sync_set)/sizeof(
   match(&if_tok, true); // if
                                           sync_set[0]);
   LangType e_type = expression()
                                       static void synch()
   if (e_type != BOOL && e_type
                                    15
80
       ! = ERR)
                                       require_sync(sync_set,
                                           sync_size, first_set,
81
   errorMessage= calloc(100,
                                           first_size);
       sizeof(*errorMessage));
   sprintf(errorMessage, "If
       clause must be a boolean
                                       // Needs implementing: None
                                    19
       expression, not %s!",
                                       void statement_list()
   typeNames[e_type]);
   throw_sem_error(errorMessage);
                                       // Production 13.1
                                       if (tokens_equal(&begin_tok,
86
   match(&then_tok, true); //
                                            current_tok, true)
       then
                                       || tokens_equal(&call_tok,
   statement();
                                           current_tok, true)
   else_tail();
                                       || tokens_equal(&id_tok,
   return;
                                           current_tok, false)
                                        || tokens_equal(&if_tok,
   }
91
                                           current_tok, true)
92
                                        || tokens_equal(&while_tok,
                                           current_tok, true))
   synch();
94
   }
                                       statement();
                                    29
                                       statement_list_tail();
        Listing 55: statement list.c
                                       return;
                                    31
   #include<stdbool.h>
                                       }
                                    32
   #include<stdlib.h>
                                       synch();
```

```
}
                                        } else if (tokens_equal(&
35
                                             end_tok, current_tok, true)
                                             ) // end
      Listing 56: statement<sub>l</sub> ist_tail.c
                                        return; // epsilon
   #include<stdbool.h>
                                     34
   #include<stdlib.h>
                                        synch();
                                        }
                                     36
   #include "productions.h"
   #include "../parser.h"
                                        Listing 57: subprogram<sub>d</sub> eclaration.c
   #include "../../tokenizer/
       tokens.h"
                                        #include<stdbool.h>
                                        #include<stdlib.h>
   static const Token* first_set
                                        #include "productions.h"
       [] = {&semic_tok, &end_tok
       };
                                        #include "../parser.h"
                                        #include "../../tokenizer/
   static const int first_size =
       sizeof(first_set)/sizeof(
                                            tokens.h"
       first_set[0]);
                                        static const Token* first_set
10
                                             [] = {&procedure_tok};
   static const Token* sync_set[]
        = {&eof_tok, &end_tok};
                                        static const int first_size =
   static const int sync_size =
                                            sizeof(first_set)/sizeof(
       sizeof(sync_set)/sizeof(
                                            first_set[0]);
       sync_set[0]);
                                        static const Token* sync_set[]
13
   static void synch()
                                              = {&procedure_tok};
                                        static const int sync_size =
15
   require_sync(sync_set,
                                             sizeof(sync_set)/sizeof(
       sync_size, first_set,
                                            sync_set[0]);
       first_size);
   }
                                        static void synch()
17
                                     15
   // Needs implementing: None
                                        require_sync(sync_set,
   void statement_list_tail()
                                             sync_size, first_set,
20
                                            first_size);
21
   // Production 13.2.1
                                        }
                                     17
   if (tokens_equal(&semic_tok,
       current_tok, true))
                                        // Needs implementing: None
                                     19
                                        void subprogram_declaration()
24
   match(&semic_tok, true);
                                        {
                                     21
25
                                        // Production 7
   statement();
   statement_list_tail();
                                        if (tokens_equal(&
                                            procedure_tok, current_tok,
   return;
                                              true)) // procedure
29
                                        {
   // Production 13.2.2
                                        bool declared =
                                     25
```

```
subprogram_head();
                                        void subprogram_declarations()
   declarations();
   subprogram_declarations();
                                       // Production 6.1
   compound_statement();
                                        if (tokens_equal(&
                                            procedure_tok, current_tok,
29
   if (declared)
                                             true)) // procedure
   reached_end_of_scope(); // pop 24
                                        subprogram_declaration();
        from stack
                                        match(&semic_tok, true); // ;
   return;
32
   }
                                        subprogram_declarations();
                                        return;
   synch();
35
                                        // Production 6.2
36
                                        } else if (tokens_equal(&
                                            begin_tok, current_tok,
                                 58:
   Listing
                                            true)) // begin
   subprogram_declarations.c
                                        return; // Epsilon
                                    32
   #include<stdbool.h>
                                    33
   #include<stdlib.h>
                                    34
                                        synch();
                                    35
   #include "productions.h"
   #include "../parser.h"
                                           Listing 59: subprogram<sub>h</sub> ead.c
   #include "../../tokenizer/
       tokens.h"
                                        #include<stdbool.h>
                                        #include<stdlib.h>
   static const Token* first_set
       [] = {&procedure_tok, &
                                        #include "productions.h"
                                        #include "../parser.h"
       begin_tok};
   static const int first_size =
                                        #include "../../tokenizer/
                                            tokens.h"
       sizeof(first_set)/sizeof(
       first_set[0]);
                                        static const Token* first_set
10
   static const Token* sync_set[]
                                            [] = {&procedure_tok};
11
        = {&eof_tok, &begin_tok};
                                        static const int first_size =
   static const int sync_size =
                                            sizeof(first_set)/sizeof(
       sizeof(sync_set)/sizeof(
                                            first_set[0]);
       sync_set[0]);
                                        static const Token* sync_set[]
13
   static void synch()
                                             = {&eof_tok, &var_tok, &
                                            procedure_tok,
15
   require_sync(sync_set,
                                        &begin_tok};
       sync_size, first_set,
                                        static const int sync_size =
                                    13
       first_size);
                                            sizeof(sync_set)/sizeof(
   }
                                            sync_set[0]);
17
   // Needs implementing: None
                                        static void synch()
                                    15
```

```
{
                                            [] = {&id_tok, &num_tok, &
16
   require_sync(sync_set,
                                           lparen_tok, &not_tok};
       sync_size, first_set,
                                       static const int first_size =
       first_size);
                                           sizeof(first_set)/sizeof(
                                           first_set[0]);
18
19
   // Needs implementing: None
                                       static const Token* sync_set[]
20
   bool subprogram_head()
                                            = {&eof_tok, &addop_tok, &
21
                                           relop_tok, &semic_tok,
22
   bool result = false;
                                       &end_tok, &else_tok, &do_tok,
   // Production 8
                                           &then_tok,
   if (tokens_equal(&
                                       &rbrac_tok, &rparen_tok, &
       procedure_tok, current_tok,
                                           comma_tok};
        true)) // procedure
                                       static const int sync_size =
   {
                                           sizeof(sync_set)/sizeof(
26
   Token* id_ref;
                                           sync_set[0]);
   match(&procedure_tok, true);
       // procedure
                                       static void synch()
                                    16
   id_ref = match(&id_tok, false)
                                    17
                                       require_sync(sync_set,
   if (id_ref != NULL) {
                                           sync_size, first_set,
   id_ref -> type = PROC;
                                           first_size);
   id_ref -> param = false;
   result = check_add_node(id_ref
       );
                                       // Needs implementing: None
                                       LangType term()
34
   arguments();
   match(&semic_tok, true); //;
                                       // Production 24.1
   return result;
                                       if (tokens_equal(&lparen_tok,
   }
                                           current_tok, true) // (
38
                                        || tokens_equal(&id_tok,
                                           current_tok, false) // ID
   synch();
   return result;
                                        || tokens_equal(&not_tok,
41
                                           current_tok, true) // not
   }
42
                                        || tokens_equal(&num_tok,
                                           current_tok, false)) { //
           Listing 60: term.c
   #include<stdbool.h>
                                        LangType f_type = factor();
   #include<stdlib.h>
                                        return term_tail(f_type);
                                       }
   #include "productions.h"
                                    32
   #include "../parser.h"
                                       synch();
   #include "../../tokenizer/
                                       return ERR;
                                    34
       tokens.h"
                                       }
   static const Token* first_set
```

```
Listing 61: term_t ail.c
                                            current_tok, false)) { //
                                            MULOP
   #include<stdbool.h>
                                        Token* mulop_op = match(&
   #include<stdlib.h>
                                           mulop_tok, false);
                                       LangType f2_type = factor();
                                    30
   #include "productions.h"
                                        return term_tail(type_lookup(
   #include "../parser.h"
                                            f_type, f2_type, mulop_op))
   #include "../../tokenizer/
       tokens.h"
                                        // Production 24.2.2
   static const Token* first_set
                                        } else if (tokens_equal(&
       [] = {&mulop_tok, &
                                           rparen_tok, current_tok,
       addop_tok, &relop_tok,
                                            true)
    &semic_tok, &end_tok, &
                                        || tokens_equal(&comma_tok,
        else_tok, &do_tok,
                                            current_tok, true)
    &then_tok, &rbrac_tok, &
10
                                        || tokens_equal(&semic_tok,
        rparen_tok,
                                            current_tok, true)
    &comma_tok};
11
                                        || tokens_equal(&rbrac_tok,
   static const int first_size =
                                            current_tok, true)
       sizeof(first_set)/sizeof(
                                        || tokens_equal(&addop_tok,
       first_set[0]);
                                            current_tok, false)
13
                                        || tokens_equal(&do_tok,
   static const Token* sync_set[]
                                            current_tok, true)
        = {&eof_tok, &addop_tok, &
                                        || tokens_equal(&else_tok,
       relop_tok, &semic_tok,
                                            current_tok, true)
   &end_tok, &else_tok, &do_tok,
15
                                        || tokens_equal(&end_tok,
       &then_tok,
                                            current_tok, true)
   &rbrac_tok, &rparen_tok, &
                                        || tokens_equal(&relop_tok,
       comma_tok};
                                            current_tok, false)
   static const int sync_size =
                                        || tokens_equal(&then_tok,
       sizeof(sync_set)/sizeof(
                                            current_tok, true))
       sync_set[0]);
                                       return f_type; // epsilon
18
                                    45
   static void synch()
19
                                       synch();
20
                                       return ERR;
   require_sync(sync_set,
                                       }
       sync_size, first_set,
       first_size);
   }
22
                                                Listing 62: type.c
                                        #include<stdbool.h>
   // Needs implementing: None
24
                                        #include<stdlib.h>
   LangType term_tail(LangType
       f_type)
                                        #include "productions.h"
   {
                                       #include "../parser.h"
   // Production 24.2.1
                                        #include "../../tokenizer/
   if (tokens_equal(&mulop_tok,
                                            tokens.h"
```

```
== 0 ? INT : REAL, numF ->
   static const Token* first_set
                                            aspect == 0 ? INT : REAL,
       [] = {&integer_tok, &
                                           &dotdot_tok) != ERR) {
       real_tok, &array_tok};
                                       if (numI -> int_val >= numF ->
   static const int first_size =
                                            int_val) {
       sizeof(first_set)/sizeof(
                                       errorMessage= calloc(100,
       first_set[0]);
                                           sizeof(*errorMessage));
                                       sprintf(errorMessage, "
10
   static const Token* sync_set[]
                                           Expected array end index %d
        = {&array_tok, &
                                            to be strictly greater
       integer_tok, &real_tok};
                                           than start %d", numF ->
   static const int sync_size =
                                           int_val, numI -> int_val);
       sizeof(sync_set)/sizeof(
                                       throw_sem_error(errorMessage);
       sync_set[0]);
                                    41
                                       id_ref -> array_length = numF
13
   static void synch()
                                           -> int_val - numI ->
14
   {
                                           int_val + 1;
15
   require_sync(sync_set,
                                       }
                                    43
       sync_size, first_set,
                                       return convert_to_array(
                                    44
       first_size);
                                           standard_type());
17
                                       // Production 4.1
                                    46
18
                                       } else if (tokens_equal(&
   // Needs implementing: None
   LangType type(Token* id_ref)
                                           integer_tok, current_tok,
                                           true) // int
   // Production 4.2
                                        || tokens_equal(&real_tok,
                                            current_tok, true)) //
   if (tokens_equal(&array_tok,
       current_tok, true))
                                            real
24
   char* errorMessage;
                                       return standard_type();
   Token* numI;
   Token* numF;
   match(&array_tok, true); //
                                       synch();
                                    53
                                       return ERR;
       array
   match(&lbrac_tok, true); // [
29
   numI = match(&num_tok, false);
        // num
                                              Listing 63: variable.c
   match(&dotdot_tok, true); //
                                       #include<stdbool.h>
                                       #include<stdlib.h>
   numF = match(&num_tok, false);
        // num
                                       #include "productions.h"
   match(&rbrac_tok, true); // ]
                                       #include "../parser.h"
   match(&of_tok, true); // of
                                       #include "../../tokenizer/
   if (numI != NULL && numF !=
                                           tokens.h"
       NULL && id_ref != NULL)
   if (type_lookup(numI -> aspect
```

```
static const Token* first_set
                                           linkedList/linkedList.h"
       [] = {&id_tok};
                                       #include "symbolTable.h"
   static const int first_size =
       sizeof(first_set)/sizeof(
                                       LinkedList* symbolTable;
       first_set[0]);
                                       int initSymbolTable()
   static const Token* sync_set[]
                                   11
        = {&eof_tok, &assignop_tok 12
                                       symbolTable = malloc(sizeof(*
                                           symbolTable));
   static const int sync_size =
                                       symbolTable -> head = 0;
       sizeof(sync_set)/sizeof(
                                       return 0;
                                    14
       sync_set[0]);
13
   static void synch()
                                       char* pushToSymbolTable(char*
14
                                    17
                                           name, size_t length)
15
   require_sync(sync_set,
                                    18
       sync_size, first_set,
                                       add(symbolTable, name, sizeof(
       first_size);
                                           char)*length);
   }
                                       return (char *)(symbolTable ->
17
                                            head -> data);
18
   // Needs implementing: None
   LangType variable()
                                    22
                                       char* checkSymbolTable(char*
   // Production 16
                                           word)
   if (tokens_equal(&id_tok,
       current_tok, false)) // id 25
                                       // Then check the symbol table
                                       struct node* node =
   Token* id_ref;
                                           symbolTable -> head;
25
   id_ref = match(&id_tok, false)
                                       while (node)
   return array_access(get_type(
                                       if (strcmp((char *) node ->
                                           data, word) == 0) // Match
       id_ref));
   }
                                       return (char *) (node -> data);
28
                                       node = node -> next;
29
   synch();
30
   return ERR;
                                       return NULL;
                                    34
32
                                       }
                                    35
        Listing 64: symbolTable.c
                                               Listing 65: addop.c
   #include<stdlib.h>
   #include<string.h>
                                       #include "../tokens.h"
   #include<stdio.h> // TODO
                                       #include "machines.h"
       Remove
                                       int addop(Token* storage, char
   #include "../dataStructures/
                                           * str, int start)
```

```
start++;
  storage -> attribute = ADDOP;
                                     } else if (str[start] == ',')
7 switch (str[start])
                                  23
                                  storage -> attribute = PUNC;
   case '+':
                                  storage -> aspect = 0;
   storage -> aspect = 0;
                                  26 start++;
                                  27 } else if (str[start] == ';')
   start++;
   return start;
                                  storage -> attribute = PUNC;
13
  case '-':
                                  storage -> aspect = 1;
  storage -> aspect = 1;
                                  31 start++;
                                  32 } else if (str[start] == '.')
   start++;
   return start;
                                  storage -> attribute = PUNC;
   default: break;
                                     storage -> aspect = 2;
                                  35
   }
                                  36
                                      start++;
20
                                     }
21
                                  37
   return start;
22
                                  39 return start;
                                  40
```

Listing 66: catchall.c

```
Listing 67: grouping.c
   #include<string.h>
                                     #include "../tokens.h"
   #include "../tokens.h"
                                      #include "machines.h"
   #include "machines.h"
                                      int grouping(Token* storage,
   int catchall(Token* storage,
                                         char* str, int start)
       char* str, int start)
                                     storage -> attribute = GROUP;
   if (strncmp(&str[start], ":=",
                                     switch (str[start])
        2) == 0)
                                      {
                                   9 case '(':
9
   storage -> attribute =
                                  storage -> aspect = 0;
                                  11 start++;
       ASSIGNOP;
   storage -> aspect = 0;
                                  12 break;
11
   start += 2;
   } else if (strncmp(&str[start
                                  14 case ')':
       ], "..", 2) == 0)
                                  storage -> aspect = 1;
                                  16 start++;
14
storage -> attribute = ARRAY;
                                  17 break;
storage -> aspect = 1;
                                  19 case '[':
17 start += 2;
                                     storage -> aspect = 2;
18 } else if (str[start] == ':'){ 20
storage -> attribute = TYPE;
                                     start++;
20 storage -> aspect = 0;
                                  22 break;
```

```
23
   case ']':
                                       return -1;
   storage -> aspect = 3;
                                    27
   start++;
   break;
                                       static int initResWords(FILE*
27
                                    29
                                           resFile)
   default:
                                    30
   break;
                                       static const int length = 11;
                                       LinkedList* resWords = malloc(
31
                                           sizeof(*resWords));
                                       LinkedList* cats = malloc(
   return start;
33
                                           sizeof(*cats));
                                       LinkedList* attrs = malloc(
                                           sizeof(*attrs));
           Listing 68: idres.c
   #include<string.h>
                                       char word[length] = {0};
   #include<stdlib.h>
                                       char attribute[length] = {0};
   #include<ctype.h>
                                       int attr = 0;
   #include<stdio.h>
                                       //while (fgets(word, length,
   #include<stdbool.h>
                                           resFile))
                                       while (true)
   #include "machines.h"
                                    41
   #include "../../errorHandler/
                                       fscanf(resFile, "%s", word);
       errorHandler.h"
                                      if (feof(resFile))
   #include "../../symbolTable/
                                    44 break;
       symbolTable.h"
                                    45 fscanf(resFile, "%s",
   #include "../../dataStructures
                                           attribute); // The actual
       /linkedList/linkedList.h"
                                           name.
   #include "../tokens.h"
                                       fscanf(resFile, "%d", &attr);
11
                                       numReserved = add(resWords, &
   static char** reservedWords;
                                           word, length*sizeof(char));
   static int numReserved;
                                       add(cats, &attribute, length*
   static enum TokenType*
                                           sizeof(char));
       categories;
                                       add(attrs, &attr, sizeof(int))
   static int* attributes;
16
17
   static int getIndex(const char
       ** array, size_t arr_size,
                                       // Initialize the lexeme table
       char* item)
                                       reservedWords = malloc(
                                           numReserved*sizeof(char*));
   while (arr_size > 0)
                                       struct node* node = resWords
21
                                           -> head;
   if (strcmp(array[arr_size -
                                    55
       1], item) == 0)
                                       for (size_t i = 0; i <
  return arr_size - 1;
                                           numReserved; i++) {
  arr_size--;
                                       reservedWords[i] = (char *)
```

```
node -> data;
                                        return 1;
   node = node -> next;
                                        }
   }
59
                                        static int isReserved(char*
   // Initialize the attribute
61
       table
   categories = malloc(
                                       // Check the reserved words
       numReserved*sizeof(enum
                                            table for a match first
                                       for (size_t i = 0; i <
       TokenType));
   node = cats -> head;
                                            numReserved; i++) {
                                        if (!reservedWords[i] ||
64
   for (size_t i = 0; i <
                                            strcmp(reservedWords[i],
                                            word) == 0) // Match
       numReserved; i++) {
   categories[i] = (enum
                                       return i;
       TokenType) getIndex(
                                        }
                                    99
    sizeof(catNames)/sizeof(char*)101
                                       return -1;
67
                                    102
    (char *) node -> data);
                                    103
   node = node -> next;
                                        int idres(Token* storage, char
                                   104
                                            * str, int start)
71
                                    105
   // Initialize the attribute
                                        int initial = start;
       table
                                        LinkedList* id = malloc(sizeof
   attributes = malloc(
                                            (*id));
                                       storage -> attribute = ID;
       numReserved*sizeof(int));
                                   108
   node = attrs -> head;
                                        storage -> aspect = 0;
                                        char next = str[start];
75
   for (size_t i = 0; i <
                                        if (isalpha(next)) // Can
       numReserved; i++) {
                                            actually be an id/reserved
   attributes[i] = *(int *) node
                                   112
       -> data:
                                        size_t wordSize = 0;
                                    113
   node = node -> next;
                                    114
79
                                    115
                                        wordSize = add(id, &next,
                                    116
80
                                            sizeof(char*));
                                        start++;
   return 0;
82
                                    117
                                        next = str[start];
   }
                                        } while(isalpha(next) ||
84
                                    119
   int initIDResMachine(FILE*
                                            isdigit(next)); // Match ID
       resFile)
                                    120
                                        // The string of the id name
   if (initSymbolTable() == 0 &&
                                        char* name = malloc((wordSize
                                    122
       initResWords(resFile) == 0)
                                            + 1)*sizeof(char));
                                       name[wordSize] = '\0';
   return 0:
   else
                                        struct node* node = id -> head
```

```
storage -> attribute = MULOP;
   for (size_t i = 0; i <
                                        if (str[start] == '*')
       wordSize; i++) {
   name[wordSize - i - 1] = *(
                                     9 storage -> aspect = 0;
       char *)(node -> data);
                                    10 start++;
   node = node -> next;
                                    11 } else if (str[start] == '/')
128
                                        storage -> aspect = 1;
129
   int index = -1;
                                        start++;
130
   char* address = 0;
   if ((index = isReserved(name)) 16
132
         >= ()
                                        return start;
   { // It's a reserved word!
   storage -> attribute =
134
       categories[index];
                                               Listing 70: numbers.c
   storage -> aspect = attributes
135
                                        #include<stdbool.h>
        [index];
                                        #include<stdlib.h>
136
                                        #include<ctype.h>
   else if ((address =
137
       checkSymbolTable(name)))
                                        #include "../tokens.h"
   storage -> id = address;
                                        #include "machines.h"
   else
139
   storage -> id =
                                        #include "../../errorHandler/
                                            errorHandler.h"
       pushToSymbolTable(name,
       wordSize);
                                        // Assumes that "str" is valid
141
                                             as an integer.
                                        char* parseNum(LinkedList*
   if (start - initial > 10) //
143
                                            chars, bool real)
        ID Too long err
144
   //storage -> attribute = NOOP; 12 char* num = malloc((chars ->
                                            size + 1) * sizeof(char));
         TODO investigate
                                        size_t count = chars -> size;
   throw_lex_error(LEXERR, 1,
                                    13
146
                                        num[count--] = 0;
       initial, start - initial);
                                    14
                                        struct node* node = chars ->
147
                                            head;
   return start;
148
                                       while (node)
149
                                    17
                                        num[count--] = *(char *)node
           Listing 69: mulop.c
                                            -> data;
   #include "../tokens.h"
                                        node = node -> next;
   #include "machines.h"
                                    20
   int mulop(Token* storage, char 22 return num;
       * str, int start)
 5 {
```

```
double parseReal(LinkedList*
                                        throw_lex_error(LEXERR, 7,
       digits)
                                            initial, start - initial);
26
                                    61
   char* array = parseNum(digits,
                                        // TODO investigate (all of
                                    62
        true);
                                            these machines)
   double val = strtod(array,
                                        /*if (errored)
                                               storage -> attribute =
       NULL);
                                    64
                                                    NOOP;
   free(array);
                                            else*/ if (start > initial)
   return val;
                                    65
   }
                                                 // It's a proper
                                                integer!
32
   int parseInt(LinkedList*
                                    66
33
                                        storage -> aspect = 0;
       digits)
                                        storage -> int_val = parseInt(
34
   char* array = parseNum(digits,
                                            digits);
        false);
                                    69
   int val = (int) strtol(array,
                                    70
       NULL, 10);
                                        return start;
                                    71
   free(array);
                                    72
   return val;
38
                                        // NOTE: Pay attention to
                                            memory stuff here (the
40
   int intMachine(Token* storage,
                                            linked list takes up space)
        char* str, int start)
                                        int realMachine(Token* storage
42
   storage -> attribute = NUM;
                                            , char* str, int start)
43
   bool errored = false;
                                        storage -> attribute = NUM;
45
                                    77
   int initial = start;
46
                                    78
                                        int initial = start;
47
                                    79
   LinkedList* digits = malloc(
                                        bool errored = false;
48
       sizeof(*digits));
                                    81
   while (isdigit(str[start]))
                                        int intPart = 0;
                                    82
49
   add(digits, &str[start++],
                                        int fracPart = 0;
                                    83
       sizeof(char*));
                                        LinkedList* digits = malloc(
   if (start - initial > 10)
                                            sizeof(*digits));
52
   {
                                        while (isdigit(str[start]))
                                        add(digits, &str[start++],
   errored = true;
   throw_lex_error(LEXERR, 2,
                                            sizeof(char*));
       initial, start - initial);
                                        intPart = start - initial;
   if (start > initial + 1 && str 90
                                        if (intPart == 0) // Not a
       [initial] == '0')
                                            real. Must start with a
   {
                                            digit.
58
                                       return initial;
   errored = true;
```

```
errored = true;
                                      126
92
    if (str[start] == '.')
                                         }
                                      127
    add(digits, &str[start++],
                                      128
                                              if (errored)
        sizeof(char*));
                                      129
    else // Not a real
                                                  storage -> attribute =
95
                                      130
                                                      NOOP; */
    return initial;
                                         else
97
                                      131
    while (isdigit(str[start]))
                                          storage -> aspect = 1;
99
                                      133
    add(digits, &str[start++],
                                          storage -> real_val =
        sizeof(char*));
                                              parseReal(digits);
                                      135
101
    fracPart = start - (initial +
102
                                     136
        intPart + 1);
                                      137
                                         return start;
                                         }
103
                                      138
    if (fracPart == 0) // Not a
104
                                     139
        real.
                                         int longRealMachine(Token*
    return initial;
                                              storage, char* str, int
105
                                              start)
106
    // Now, we check for errors.
107
                                     141
    if (intPart > 5)
                                          storage -> attribute = NUM;
    {
                                     143
109
    throw_lex_error(LEXERR, 3,
                                          int initial = start;
        initial, start - initial); 145
                                          bool errored = false;
    errored = true;
111
                                         int intPart = 0;
112
                                      147
   if (fracPart > 5)
                                         int fracPart = 0;
113
                                          int expPart = 0;
114
    throw_lex_error(LEXERR, 4,
                                      150
        initial, start - initial); 151
                                         LinkedList* digits = malloc(
    errored = true;
                                              sizeof(*digits));
116
                                         while (isdigit(str[start]))
117
    if (str[initial] == '0' &&
                                          add(digits, &str[start++],
118
                                     153
        intPart > 1) // Leading
                                              sizeof(char*));
        zero!
                                      154
    {
                                          intPart = start - initial;
119
    throw_lex_error(LEXERR, 8,
                                          if (intPart == 0) // Not a
                                     156
120
                                              real. Must start with a
        initial, start - initial);
    errored = true;
                                              digit.
121
    }
                                          return initial;
122
    if (str[start - 1] == '0' &&
123
                                      158
        fracPart > 1) // Trailing
                                         // REAL part
                                         if (str[start] == '.')
        zero!
                                         add(digits, &str[start++],
124
                                              sizeof(char*));
   throw_lex_error(LEXERR, 9,
        initial, start - initial); 162 else // Not a real
```

```
return initial;
                                          if (intPart > 5)
163
164
                                          throw_lex_error(LEXERR, 3,
165
                                      201
    while (isdigit(str[start]))
                                              initial, start - initial);
    add(digits, &str[start++],
                                          errored = true;
167
                                      202
        sizeof(char*));
                                          }
                                          if (fracPart > 5)
168
                                      204
    fracPart = start - (initial +
169
        intPart + 1);
                                          throw_lex_error(LEXERR, 4,
                                      206
                                              initial, start - initial);
170
    if (fracPart == 0) // Not a
                                          errored = true;
171
                                      207
        real.
                                      208
                                          if (str[initial] == '0' &&
    return initial;
172
                                      209
                                              intPart > 1) // Leading
173
                                              zero!
174
    // LONG REAL part
175
                                      210
    int signum = 0;
                                          throw_lex_error(LEXERR, 8,
176
                                      211
                                              initial, start - initial);
177
    if (str[start] == 'E')
                                          errored = true;
                                      212
178
    add(digits, &str[start++],
                                      213
179
        sizeof(char*));
                                          if (str[start - expPart - 2]
    else // Not a long real
                                              == '0' && fracPart > 1) //
180
                                              Trailing zero in real!
    return initial;
181
182
                                      215
    if (str[start] == '+' || str[
                                          throw_lex_error(LEXERR, 9,
                                      216
183
        start] == '-')
                                              initial, start - initial);
    {
                                          errored = true;
184
    signum++;
                                          }
                                      218
185
                                          if (expPart > 2) // Exponent
    add(digits, &str[start++],
        sizeof(char*));
                                              too long!
    }
                                      220
187
                                          throw_lex_error(LEXERR, 5,
188
                                      221
    while (isdigit(str[start]))
                                              initial, start - initial);
189
    add(digits, &str[start++],
                                          errored = true;
190
                                      222
        sizeof(char*));
                                      223
                                          if (str[start - expPart] == '0
191
    expPart = start - (initial +
                                              ') // Leading zero in
192
        fracPart + intPart + signum
                                              exponent!
         + 2);
                                      225
                                          throw_lex_error(LEXERR, 10,
193
    if (expPart == 0) // Not a
                                              initial, start - initial);
194
        long real
                                          errored = true;
                                      227
    return initial;
                                      228
195
                                      229
                                              if (errored)
197
                                      230
                                                  storage -> attribute =
    // Now, we check for errors.
                                      231
```

```
NOOP;
                                         storage -> aspect = 4;
        else*/
                                         start++;
232
    {
                                         } else {
233
                                         storage -> aspect = 3;
    storage -> aspect = 1;
234
    storage -> real_val =
                                     37
235
        parseReal(digits);
                                         break;
236
                                         default: break; // Do not
237
                                             increment; continue on to
   return start;
238
                                             the next machine.
   }
                                     41
            Listing 71: relop.c
                                        return start;
                                     - 43
    #include "../tokens.h"
                                     44
                                         }
    #include "machines.h"
                                               Listing 72: whitespace.c
    int relop(Token* storage, char
                                         #include<stdlib.h>
        * str, int start)
                                         #include<ctype.h>
 5
   storage -> attribute = RELOP;
                                         #include "../tokens.h"
    char next = str[start];
                                         #include "machines.h"
    switch (next) {
    case '<':
                                         int whitespace(Token* storage,
    start++;
                                              char* str, int start)
    if (str[start] == '=')
                                         storage -> attribute = WS;
    storage -> aspect = 1;
                                         if (isspace(str[start]))
    start++;
    } else if (str[start] == '>')
                                         storage -> aspect = 0;
                                     12
                                         if (str[start] == '\n')
    storage -> aspect = 5;
                                         storage -> aspect = 1;
    start++;
                                      15 start++;
    } else {
                                     16
                                         }
    storage -> aspect = 0;
                                         return start;
21
                                         }
                                     18
    break;
22
23
                                               Listing 73: tokenizer.c
   case '=':
^{24}
    start++;
                                         #include<stdio.h>
    storage -> aspect = 2;
                                         #include<stdlib.h>
    break;
                                         #include<stdbool.h>
28
                                         #include<string.h>
   case '>':
    start++;
                                         #include "tokenizer.h"
   if (str[start] == '=')
                                         #include "../dataStructures/
   {
32
                                             linkedList/linkedlist.h"
```

```
#include "machines/machines.h"
                                       int end;
   #include "../errorHandler/
                                       current -> start = START;
       errorHandler.h"
                                       for (int i = 0; i < sizeof(</pre>
   #include "../globals/globals.h
                                           machines)/sizeof(machine);
                                       {
11
   const machine machines[] = {
                                       current -> aspect = 0;
                                    42
       whitespace, idres,
                                       end = (*machines[i])(current,
       longRealMachine,
                                           BUFFER, START);
  realMachine, intMachine,
                                      if (end > START) {
                                       current -> length = end -
       grouping, catchall, relop,
                                    45
       addop, mulop};
                                           START;
                                       START = end;
14
   // Initialization stuff
                                       return current;
   static bool initialized =
                                       }
       false;
                                       }
17
   int initializeTokens(FILE*
                                       // Unrecognized symbol error.
18
       resFile)
                                           This error is manual
                                           because it takes
19
  if (resFile) {
                                       // the place of a lexeme,
   initIDResMachine(resFile);
                                           rather than being processed
   initialized = true;
                                            during one.
   } else {
                                       throw_lex_error(LEXERR, 0,
   fprintf(stderr, "%s\n", "
                                           START, 1);
       Reserved words file for
                                       //current -> attribute = NOOP;
       tokenizer null!");
                                      START++;
   }
                                    56 return current;
25
                                      } else {
   return 1;
                                       fprintf(stderr, "%s\n", "
   }
27
                                           Tokenizer not initialized.
28
   static Token*
                                           Aborting.");
       generateNextToken()
                                      return NULL;
                                       }
30
   if (initialized) {
                                       }
   Token* current = malloc(sizeof
       (*current)); // TODO
                                    63
       necessary allocation?
                                       Token* getNextToken()
                                    64
   if ((current =
                                    65
       getNextErrorToken()) !=
                                       Token* next = malloc(sizeof(*
       NULL)
                                           next));
   return current;
                                       do {
                                       next = generateNextToken();
    current = malloc(sizeof(*
                                       } while (next -> attribute ==
        current));
                                           NOOP):
37
```

```
return next;
                                           ARRAY, 1, true, 0, 0};
   }
                                        const Token period_tok = {PUNC
72
                                            , 2, true, 0, 0};
                                        const Token lbrac_tok = {GROUP
          Listing 74: tokens.c
                                            , 2, true, 0, 0};
   #include<string.h>
                                        const Token rbrac_tok = {GROUP
   #include<stdlib.h>
                                            , 3, true, 0, 0};
   #include<stdio.h>
                                        const Token addop_tok = {ADDOP
                                            , 0, false, 0, 0};
   #include "../errorHandler/
                                        const Token array_tok = {ARRAY
       errorHandler.h"
                                           , 0, true, 0, 0};
   #include "tokens.h"
6
                                        const Token assignop_tok = {
                                           ASSIGNOP, 0, true, 0, 0};
   const char* catNames[] = {"
                                        const Token begin_tok = {
       NOOP", "FILEEND", "ASSIGNOP
                                           CONTROL, 0, true, 0, 0);
       ", "RELOP", "ID",
                                        const Token call_tok = {
    "CONTROL", "ADDOP", "MULOP",
                                           CONTROL, 10, true, 0, 0};
        WS", "ARRAY", "TYPE",
                                        const Token do_tok = {CONTROL,
    "VAR", "NUM", "PUNC", "GROUP"
                                             1, true, 0, 0};
10
         "INVERSE",
                                        const Token else_tok = {
    "LEXERR", "SYNERR", "SEMERR"};
                                           CONTROL, 2, true, 0, 0};
11
                                        const Token end_tok = {CONTROL
12
   const char* typeNames[] = {"
                                            , 3, true, 0, 0};
13
       ERR", "REAL", "INT", "BOOL"
                                        const Token id_tok = {ID, 0,
       , "PROGRAM",
                                           false, 0, 0};
    "PROGRAM_PARAMETER", "
                                        const Token if_tok = {CONTROL,
        PROCEDURE",
                                             5, true, 0, 0};
    "INT ARRAY", "REAL ARRAY"};
                                        const Token integer_tok = {
16
                                           TYPE, 1, true, 0, 0};
   const Token eof_tok = {FILEEND
                                       const Token integer_val_tok =
17
                                    40
       , 0, false, 0, 0};
                                           {NUM, 0, true, 0, 0};
   const Token lparen_tok = {
                                       const Token of_tok = {ARRAY,
18
       GROUP, 0, true, 0, 0};
                                           2, true, 0, 0};
   const Token rparen_tok = {
                                        const Token real_val_tok = {
19
       GROUP, 1, true, 0, 0};
                                           NUM, 1, true, 0, 0};
   const Token plus_tok = {ADDOP,
                                        const Token mulop_tok = {MULOP
20
        0, true, 0, 0};
                                            , 0, false, 0, 0};
   const Token comma_tok = {PUNC,
                                       const Token not_tok = {INVERSE
        0, true, 0, 0};
                                            , 0, true, 0, 0};
   const Token minus_tok = {ADDOP
                                        const Token num_tok = {NUM, 0,
22
       , 1, true, 0, 0};
                                            false, 0, 0};
   const Token semic_tok = {PUNC,
                                       const Token procedure_tok = {
23
        1, true, 0, 0};
                                           CONTROL, 6, true, 0, 0);
   const Token colon_tok = {TYPE,
                                       const Token program_tok = {
        0, true, 0, 0};
                                           CONTROL, 7, true, 0, 0};
   const Token dotdot_tok = {
                                       const Token real_tok = {TYPE,
```

```
2, true, 0, 0};
                                        const Token* getTokenFromLex(
   const Token relop_tok = {RELOP
                                            char* lex) {
       , 0, false, 0, 0};
                                        for (int i = 0;i < sizeof(</pre>
                                            lexes); i++) {
   const Token then_tok = {
       CONTROL, 8, true, 0, 0};
                                        if (strcmp(lexes[i], lex) ==
   const Token var_tok = {VAR, 0,
                                            0)
        true, 0, 0};
                                       return tokens[i];
                                    73
   const Token while_tok = {
                                        }
52
       CONTROL, 9, true, 0, 0);
                                       return NULL;
53
   static const char* lexes[] = { 77
54
       "(", ")", "+", ",", "-", "; 78
       ", ":", "[", "]", "addop", 79
                                        const char* getLexFromToken(
   "array", "assignop", "begin",
                                            Token* token, bool strict)
55
       "call", "do", "else",
                                            {
   "end", "ID", "if", "integer",
                                        switch (token -> attribute) {
56
       "mulop", "not",
                                        case FILEEND: return "EOF";
   "num", "procedure", "program",
                                        case ASSIGNOP: return ":=";
                                    82
57
        "real", "relop",
   "then", "var", "while", "EOF",
                                        case RELOP: if (strict)
58
        "..", ":", ".",
                                        switch (token -> aspect) {
    "int value", "of", "real value 86
                                        case 0: return "<";
59
        "};
                                        case 1: return "<=";
                                        case 2: return "=";
60
                                        case 3: return ">";
   static const Token* tokens[] =
        {&lparen_tok, &rparen_tok, 90
                                        case 4: return ">=";
        &plus_tok, &comma_tok, &
                                        case 5: return "<>";
                                    91
       minus_tok, &semic_tok,
                                    92
   &colon_tok, &lbrac_tok, &
                                        else return "RELOP";
       rbrac_tok, &addop_tok, &
                                    94
       array_tok, &assignop_tok,
                                        case ID: return "ID";
   &begin_tok, &call_tok, &do_tok 96
       , &else_tok, &end_tok, &
                                        case CONTROL: if (!strict)
       id_tok,
                                            return "CONTROL"; else
   &if_tok, &integer_tok, &
                                        switch (token -> aspect) {
                                        case 0: return "begin";
       mulop_tok, &not_tok, &
                                        case 1: return "do";
       num_tok,
                                    100
                                        case 2: return "else";
   &procedure_tok, &program_tok,
                                    101
       &real_tok, &relop_tok, &
                                        case 3: return "end";
                                    102
                                        case 4: return "function";
       then_tok,
                                        case 5: return "if";
   &var_tok, &while_tok, &eof_tok 104
       , &dotdot_tok,
                                        case 6: return "procedure";
                                    105
                                        case 7: return "program";
   &colon_tok, &period_tok, &
                                    106
67
       integer_val_tok,
                                        case 8: return "then";
   &of_tok, &real_val_tok};
                                       case 9: return "while";
                                    108
                                       case 10: return "call";
69
```

```
110 }
                                          case 0: return ",";
                                          case 1: return ";";
111
    case ADDOP: if (!strict)
                                          case 2: return ".";
                                     150
112
        return "ADDOP"; else
                                     151
    switch (token -> aspect) {
113
                                     152
    case 0: return "+";
                                         case GROUP: switch (token ->
    case 1: return "-";
                                              aspect) {
115
                                          case 0: return "(";
    }
116
                                     154
                                          case 1: return ")";
117
                                     155
    case MULOP: if (!strict)
                                          case 2: return "[";
        return "MULOP"; else
                                          case 3: return "]";
                                     157
    switch (token -> aspect) {
119
                                     158
    case 0: return "*";
                                     159
    case 1: return "/";
                                     160
                                         case INVERSE: switch (token ->
    }
                                               aspect) {
122
                                          case 0: return "not";
123
                                     161
   case ARRAY: if (!strict)
                                         }
124
                                     162
        return "ARRAY"; else
                                     163
    switch (token -> aspect) {
                                         case NOOP:
                                     164
    case 0: return "array";
                                         case WS:
                                     165
126
    case 1: return "..";
                                         case LEXERR:
    case 2: return "of";
                                         case SYNERR:
                                     167
128
                                         case SEMERR: return "An error
                                              in the compiler has
130
    case TYPE: switch (token ->
                                              occurred.";
131
        aspect) {
                                     169
    case 0: return ":";
                                         }
                                     170
    case 1: return "integer";
                                     171
133
    case 2: return "real";
                                         // Returns true if the tokens
                                              are equivalent, false
135
                                              otherwise
136
    case VAR: switch (token ->
                                         bool tokens_equal(const Token*
137
        aspect) {
                                              p1, Token* p2, bool strict
    case 0: return "var";
                                              ) {
138
                                         return p1 -> attribute == p2
139
                                              -> attribute &&
140
    case NUM: if (!strict) return
                                          (!strict || p1 -> aspect == p2
                                     175
141
        "a number"; else
                                               -> aspect);
    switch (token -> aspect) {
                                     176
142
    case 0: return "integer value"
                                         LangType convert_to_array(
                                     178
    case 1: return "real value";
                                              LangType type) {
                                          char* errorMessage;
145
                                     179
                                          switch (type) {
147 case PUNC: switch (token ->
                                          case INT: return AINT;
                                     181
        aspect) {
                                         case REAL: return AREAL;
```

```
sprintf(errorMessage, "Cannot
183
                                             assign values to variables
   // Type mismatch!!
   default:errorMessage= calloc
                                             of type %s!", typeNames[
        (150, sizeof(*errorMessage)
                                             first]);
                                         throw_sem_error(errorMessage);
   {\tt sprintf(errorMessage,~"Attempt~$_{215}$}
                                         return ERR;
         to create array using type216
                                         }
                                         else if (second != INT &&
         %s; must use integer or
        real instead!", typeNames[
                                             second != REAL) {
        type]);
                                         errorMessage= calloc(100,
   throw_sem_error(errorMessage);
                                             sizeof(*errorMessage));
187
                                         sprintf(errorMessage, "Attempt
188
   case ERR: return ERR;
                                              to assign %s value; only
189
                                             reals and integers can be
   }
190
   }
                                             assigned!", typeNames[
191
                                             second]);
192
   LangType convert_from_array(
                                         throw_sem_error(errorMessage);
193
        LangType type) {
                                         return ERR;
                                     221
    char* errorMessage;
                                     222
194
    switch (type) {
                                         else if (first != second) {
195
                                         errorMessage= calloc(100,
    case AINT: return INT;
    case AREAL: return REAL;
                                             sizeof(*errorMessage));
197
                                         sprintf(errorMessage, "Attempt
                                              to convert type %s into
199
   default: errorMessage = calloc
                                             type %s in assignment!",
200
        (100, sizeof(*errorMessage)
                                             typeNames[first], typeNames
                                             [second]);
     sprintf(errorMessage, "Attempt226
                                         throw_sem_error(errorMessage);
201
          to index variable of type 227
                                         return ERR;
          %s!", typeNames[type]);
     throw_sem_error(errorMessage); 229
202
    case ERR: return ERR:
                                         return NULL;
203
                                     230
   }
204
                                     231
   }
205
                                     232
                                         static LangType relop_lookup(
                                     233
206
   static LangType
                                             LangType first, LangType
        assignop_lookup(LangType
                                             second) {
                                         char* errorMessage;
        first, LangType second) {
    char* errorMessage;
                                         if (first == second && (first
208
                                     235
    if (first == ERR || second ==
                                             == INT || first == REAL))
        ERR) // just an err
                                         return BOOL;
                                     236
   return ERR;
                                         else if (first != ERR &&
   else if (first != INT && first
                                             second != ERR) {
         != REAL) {
                                         errorMessage= calloc(100,
                                             sizeof(*errorMessage));
   errorMessage= calloc(100,
        sizeof(*errorMessage));
                                         sprintf(errorMessage, "Attempt
```

```
to compare incompatible
                                              typeNames[second]);
        types %s and %s!",
                                         throw_sem_error(errorMessage);
        typeNames[first], typeNames268
        [second]);
    throw_sem_error(errorMessage); 270
                                         return ERR;
240
    }
241
                                         default: return NULL;
242
                                     272
                                         }
    return ERR;
                                     273
243
244
                                     274
245
    static LangType addop_lookup(
                                         static LangType mulop_lookup(
246
                                     276
        LangType first, LangType
                                             LangType first, LangType
        second, int opcode) {
                                             second, int opcode) {
    char* errorMessage;
                                         char* errorMessage;
247
                                     277
    switch (opcode) {
248
                                     278
    case 0:
                                         switch (opcode) {
249
    case 1: if (first == second && 280
                                         case 0:
         (first == INT || first == 281
                                         case 1: if (first == second &&
        REAL))
                                              (first == INT || first ==
    return first;
                                             REAL))
251
    else if (first != ERR &&
                                         return first;
        second != ERR) {
                                         else if ((first == REAL &&
                                     283
    errorMessage= calloc(100,
                                             second == INT)
                                          || (first == INT && second ==
        sizeof(*errorMessage));
    sprintf(errorMessage, "Attempt
                                              REAL)) {
         to add incompatible types 285
                                          errorMessage= calloc(100,
        %s and %s!", typeNames[
                                              sizeof(*errorMessage));
        first], typeNames[second]);286
                                          sprintf(errorMessage, "Attempt
    throw_sem_error(errorMessage);
                                               to multiply or divide
    return ERR;
                                              incompatible types %s and
256
    }
                                              %s!", typeNames[first],
257
                                              typeNames[second]);
258
    return ERR;
                                          throw_sem_error(errorMessage);
259
                                     287
                                         }
260
                                     288
                                         else if (first != ERR &&
                                     289
261
    case 2: if (first == second &&
                                             second != ERR) {
         first == BOOL)
                                         errorMessage= calloc(100,
                                     290
                                             sizeof(*errorMessage));
    return BOOL;
    else if (first != ERR &&
                                         sprintf(errorMessage, "
264
                                     291
        second != ERR) {
                                             Expceted ints or reals for
                                             multiplication, received %s
    errorMessage= calloc(100,
265
        sizeof(*errorMessage));
                                              and %s!", typeNames[first
                                             ], typeNames[second]);
    sprintf(errorMessage, "
        Expected BOOL and BOOL for 292
                                         throw_sem_error(errorMessage);
        use with 'or', received %s 293
        and %s!", typeNames[first], 294
```

```
return ERR;
                                              second) {
295
                                         char* errorMessage;
296
297
                                     329
                                         if (first == BOOL)
    case 2: if (first == second && 330
         first == BOOL) // and
                                         return BOOL;
                                     331
                                         else if (first != ERR)
    return BOOL;
    else if (first != ERR &&
300
                                     333
        second != ERR)
                                         errorMessage= calloc(100,
                                              sizeof(*errorMessage));
301
    errorMessage= calloc(100,
                                         sprintf(errorMessage, "
        sizeof(*errorMessage));
                                             Expected BOOL use with 'not
                                              ', received %s!", typeNames
    sprintf(errorMessage,
303
        Expected BOOL and BOOL for
                                              [first]);
        use with 'and', received %s336
                                         throw_sem_error(errorMessage);
         and %s!", typeNames[first 337
        ], typeNames[second]);
    throw_sem_error(errorMessage); 339
                                         return ERR;
304
                                     340
305
306
                                     341
    return ERR;
                                         static LangType array_lookup(
307
                                     342
                                              LangType first, LangType
    case 3: // div: mod
                                              second) {
309
    case 4: if (first == second && 343
                                         if (first == second && first
310
         first == INT)
                                              == INT)
    return INT;
                                         return INT;
    else if (first != ERR &&
                                         else if (first != ERR)
                                     345
312
        second != ERR) {
    errorMessage= calloc(100,
                                         char* errorMessage = calloc
313
                                     347
        sizeof(*errorMessage));
                                              (100, sizeof(*errorMessage)
    sprintf(errorMessage,
                                             );
314
    "Integers required with %s,
                                         sprintf(errorMessage, "Attempt
315
        received %s and %s!",
                                              to index variable of type
    opcode == 3 ? "div" : "mod",
                                             %s!", typeNames[first]);
316
        typeNames[first],
                                         throw_sem_error(errorMessage);
                                         } else if (second != ERR){
    typeNames[second]);
317
    throw_sem_error(errorMessage); 351
                                         char* errorMessage = calloc
                                              (100, sizeof(*errorMessage)
319
                                             );
                                         sprintf(errorMessage, "Attempt
    return ERR;
321
                                              to use variable of type %s
322
    default: return NULL;
                                              to index array!",
323
                                              typeNames[second]);
                                         throw_sem_error(errorMessage);
325
                                     353
                                         }
                                     354
    static LangType not_lookup(
327
                                     355
        LangType first, LangType
                                     356
                                         return ERR;
```

```
}
357
358
    LangType type_lookup(LangType
359
        first, LangType second,
        Token* op) {
    if (first == ERR || second ==
        ERR || op == NULL)
    return ERR;
361
362
    switch (op -> attribute) {
    // Operations which are
364
        meaninngless
    case NOOP:
365
    case LEXERR:
366
    case SYNERR:
    case SEMERR:
368
    case GROUP:
    case PUNC:
    case FILEEND:
    case ID:
372
    case CONTROL:
    case WS:
374
    case TYPE:
    case VAR:
    case NUM: return NULL;
378
    case ASSIGNOP: return
        assignop_lookup(first,
        second);
    case RELOP: return
380
        relop_lookup(first, second)
    case ADDOP: return
381
        addop_lookup(first, second,
         op -> aspect);
    case ARRAY: return
        array_lookup(first, second)
    case MULOP: return
383
        mulop_lookup(first, second,
         op -> aspect);
    case INVERSE: return
        not_lookup(first, second);
    }
386
    }
387
```