

# **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.

# Grammar with Decorations

Return type (s)

Global

Synthesized  
Inherited (or other  
semantic action)

local

→ true on success  
→ false otherwise

checkAddGreenNode (id: lex, PGNAME)  
offset = 0

1.	void	program	→	<b>program</b> id ( identifier_list ) ; declarations subprogram_declarations compound_statement .	
2.1.	void	identifier_list	→	<b>id</b> identifier_list'	checkAddBlueNode (id: lex, PGNAME)
2.2.1.	void	identifier_list'	→	, id identifier_list'	
2.2.2.		identifier_list'	→	ε	
3.1.	void	declarations	→	<b>var</b> id : type ; declarations	checkAddBlueNode (id: lex, type, type)
3.2.		declarations	→	ε	
4.1.	TYPE	type	→	<b>standard_type</b>	type ← st.type.type width ← st.width
4.2.		type	→	<b>array</b> [ num .. num ] <b>of</b> standard_type	type ← AINT :S INT AREAL :F REAL width ← (num - num + 1) * st.width E1R :S E2R E1R* : otherwise offset += width
5.1.	TYPE	standard_type	→	<b>integer</b>	type ← INT width ← 4
5.2.		standard_type	→	<b>real</b>	type ← REAL width ← 8
6.1.	void	subprogram_declarations	→	subprogram_declaration ; subprogram_declarations	
6.2.		subprogram_declarations	→	ε	
7.	void	subprogram_declaration	→	subprogram_head declarations subprogram_declarations compound_statement	pop From Green Stack
8.	void	subprogram_head	→	<b>procedure</b> id arguments ;	checkAddGreenNode (id: lex, PROC) offset = 0
9.1.	void	arguments	→	( parameter_list )	
9.2.		arguments	→	ε	
10.1.	void	parameter_list	→	<b>id</b> : type parameter_list'	checkAddBlueNode (id: lex, 'pp' ++ type)
10.2.1.	void	parameter_list'	→	; id : type parameter_list'	
10.2.2.		parameter_list'	→	ε	
11.	void	compound_statement	→	<b>begin</b> optional_statements <b>end</b>	
12.1.	void	optional_statements	→	statement_list	
12.2.		optional_statements	→	ε	
13.1.	void	statement_list	→	statement statement_list'	
13.2.1.	void	statement_list'	→	; statement statement_list'	
13.2.2.		statement_list'	→	ε	
14.1.	void	statement	→	<b>variable assignop</b> expression	
14.2.		statement	→	<b>procedure_statement</b>	
14.3.		statement	→	<b>compound_statement</b>	
14.4.		statement	→	<b>while</b> expression <b>do</b> statement	
14.5.		statement	→	<b>if</b> expression <b>then</b> statement <b>else</b> statement	Check type BOGL
15.1.	void	else'	→	<b>else</b> statement	
15.2.		else'	→	ε	
16.	TYPE	variable	→	<b>id</b> array_access	
17.1.	TYPE	array_access	→	[ expression ]	Same as factor → id factor'
17.2.		array_access	→	ε	
18.	void	procedure_statement	→	<b>call</b> id optional_expressions	Must exist & be a procedure
19.1.	void	optional_expressions	→	( expression_list )	i ← pointer to the first item
19.2.		optional_expressions	→	ε	
20.1.	void	expression_list	→	expression expression_list'	i ← pointer to the right type - if match, continue; else, fail
20.2.1.	void	expression_list'	→	, expression expression_list'	
20.2.2.		expression_list'	→	ε	
21.	TYPE	expression	→	<b>simple_expression</b> related_expression	
22.1.	TYPE	related_expression	→	<b>relop</b> simple_expression	Looks exactly like the table for * and / for factor; so do
22.2.		related_expression	→	ε	
23.1.1.	TYPE	simple_expression	→	<b>term</b> simple_expression'	these
23.1.2.		simple_expression	→	<b>sign</b> term simple_expression'	
23.2.1.	TYPE	simple_expression'	→	<b>addop</b> term simple_expression'	
23.2.2.		simple_expression'	→	ε	

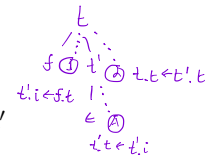
±

or

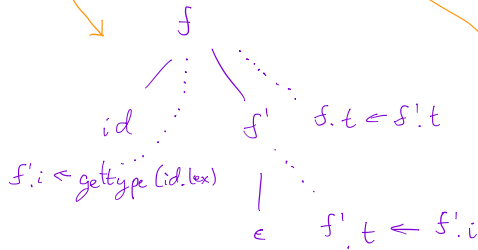
s. t	v. t	a. t
ERR*	UNDEF	INT
—	INT	REAL
—	REAL	ERR
ERR	Any thing	Anything



24.1.	TYPE	term	→	factor term'
24.2.1.	TYPE	term'	→	mulop factor term'
24.2.2.		term'	→	ε
25.1.1.	TYPE	factor	→	id factor' {type ← factor'.type}
25.1.2.		factor	→	num {type ← num.type}
25.1.3.		factor	→	( expression ) {type ← exp.type}
25.1.4.		factor	→	not factor
25.2.1.		factor'	→	[ expression ] See table down below
25.2.1.		factor'	→	ε {type ← gettype(id.lex)} (undeclared error, procedure, etc. - use a table)
26.1.	void	sign	→	+
26.2.		sign	→	-



f.type	f'.type
Bool	Bool
ERR*	rBool ^ rERR
ERR?	ERR



Array:

f'.t	e.t	f'.i
INT	INT	AINT
REAL	INT	AREAL
ERR*	ERR	ERR
ERR*	ERR	ERR
ERR	ERR	ERR
ERR	ERR	ERR

AINT :: Array of ints  
AREAL :: Array of reals

Mulops:

t.i	f.t	Mulop.op	t'.i
INT	INT	*	INT
REAL	REAL	*	REAL
ERR	ERR	*	ERR
ERR*	INT	*	REAL

But there is no type coercion

Watch out for the possibility of array types !!

## 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

---

```
1 program fib(input; output):
2   var a: int; var p: integer;
3   var numsArray : array [6..12] on integer;
4   var q: real;
5
6   procedure fib1(aReallyLongInt : integer; b : real, c
7     : real);
8     begin
9       if a <= 1.20 then fib := c
10        else call fib (a - 01, c, b + c)
11    end;
12
13  procedure fib2(a : integer);
14    var b : real; var c : real; var sum : ;
15    var b : real;
16    procedure rawr3(b : real);
17      var q : real;
18      begin
19        q := b + 2.0;
20        call fib2(q).
21      end;
22
23  begin
24    a := a - 1;
25    fib1(3.00);
26    sum := 1;
27    c := b;
28    while a > 0) do
29      call 3;
30      begin
31        a := a - 1;
32        b := sum;
33        sum := c + sum;
34        c := b
35      end;
36      fib2 := sum
37    end;
38
39  procedure init;
40    begin
```

```

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

```

---

Listing 2: Error-Full Listing File

---

```

1      1 program fib(input; output):
2  SYNERR: Found ','; expected ')', ',' instead.
3  SYNERR: Found ':'; expected ';' instead.
4      2 var a: int; var p: integer;
5  SYNERR: Found 'ID'; expected 'array', 'real', 'integer
   ' instead.
6  SYNERR: Found 'integer'; expected ';' instead.
7  SYNERR: Found ','; expected 'begin', 'procedure', 'var
   ' instead.
8      3 var numsArray : array [6..12] on integer;
9      4 var q: real;
10     5
11     6 procedure fib1(aReallyLongInt : integer; b :
       real, c : real);
12  LEXERR: ID length exceeded 10
       characters: aReallyLongInt
13  SYNERR: Found ','; expected ')', ';' instead.
14     7 begin
15     8 if a <= 1.20 then fib := c
16  SEMERR: No variable 'a' is defined in the local scope!
17  LEXERR: Trailing 0 in
       real: 1.20
18  SEMERR: Cannot assign to ID 'fib' of type 'PROGRAM'!
19     9 else call fib (a - 01, c, b + c)
20  SEMERR: No variable ' ' is defined in the local scope!
21  SEMERR: Procedure 'fib' not in scope!
22  SEMERR: No variable 'a' is defined in the local scope!
23  LEXERR: Leading 0 in

```

```

int:                                01
24 SEMERR: No variable 'c' is defined in the local scope!
25 SEMERR: No variable 'c' is defined in the local scope!
26     10     end;
27     11
28     12     procedure fib2(a : integer);
29     13         var b : real; var c : real; var sum : ;
30 SYNERR: Found ';;'; expected 'array', 'real', 'integer'
    instead.
31     14         var b : real;
32 SYNERR: Found 'real'; expected ';' instead.
33 SYNERR: Found ';;'; expected 'begin', 'procedure', 'var'
    ' instead.
34     15     procedure rawr3(b : real);
35     16         var q : real;
36     17         begin
37     18             q := b + 2.0;
38     19             call fib2(q).
39 SEMERR: Expected type INT, not REAL!
40 SYNERR: Found '.'; expected 'end', ';' instead.
41     20     end;
42     21
43     22     begin
44     23         a := a - 1;
45     24         fib1(3.00);
46 SYNERR: Found '('; expected '[', ':= ' instead.
47 LEXERR:                                Trailing 0 in
    real:                                3.00
48     25         sum := 1;
49     26         c := b;
50     27         while a > 0) do
51 SYNERR: Found ')'; expected 'do' instead.
52 SYNERR: Found 'do'; expected 'if', 'while', 'begin', '
    call', 'ID' instead.
53     28         call 3;
54     29         begin
55     30             a := a - 1;
56     31             b := sum;
57 SEMERR: No variable 'sum' is defined in the local
    scope!
58     32             sum := c + sum;
59 SEMERR: ID 'sum' not in scope!
60 SEMERR: No variable 'sum' is defined in the local
    scope!
61     33         c := b
62     34     end;

```



```

63         35         fib2 := sum
64 SEMERR: Cannot assign to ID 'fib2' of type 'PROCEDURE
        '!
65         36         end;
66 SEMERR: No variable 'sum' is defined in the local
        scope!
67         37
68         38         procedure init;
69         39         begin
70         40             n := 12;
71 SEMERR: ID 'n' not in scope!
72         41             if (1 and 2) or 3 then p := 12
73 SEMERR: Expected BOOL and BOOL for use with 'and',
        received INT and INT!
74 SEMERR: ID 'p' not in scope!
75         42             else p := 14;
76 SEMERR: ID 'p' not in scope!
77         43             numsArray[3] := 15.560;
78 LEXERR:                                     Trailing 0 in
        real:                                15.560
79 SEMERR: ID 'numsArray' not in scope!
80         44             q := q[4];
81 SEMERR: ID 'q' not in scope!
82 SEMERR: No variable 'q' is defined in the local scope!
83         45             q[4] := 12
84 SEMERR: ID 'q' not in scope!
85         46         end;
86         47
87         48         begin*
88 SYNERR: Found '*'; expected 'array', 'end', 'if', '
        while', 'begin', 'call', 'ID' instead.
89         49             call init;
90         50             call fib2;
91         51             call rawr3(34, 56);
92         52         end.
93         53
94         54 a
95 SYNERR: Found 'ID'; expected 'EOF' instead.

```

---

Listing 3: Error-Full Semantic Mem File

---

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

Listing 4: Error-Full Token File

1	1	FILEEND				
2	2	ASSIGNOP				
3	3	RELOP				
4	4	ID				
5	5	CONTROL				
6	6	ADDOP				
7	7	MULOP				
8	8	WS				
9	9	ARRAY				
10	10	TYPE				
11	11	VAR				
12	12	NUM				
13	13	PUNC				
14	14	GROUP				
15	15	INVERSE				
16	16	LEXERR				
17	17	SYNERR				
18	18	SEMERR				
19		LineLexeme	Token	Attribute	Token	Type
20	1	program	5	7		
21	1	fib	4	0x7ff6d6600fd0		
22	1	(	14	0		
23	1	input	4	0x7ff6d66012e0		
24	1	;	13	1		
25	1	;	17	0		
26	1	output	4	0x7ff6d6601760		
27	1	)	14	1		
28	1	:	10	0		
29	1	:	17	0		
30	2	var	11	0		
31	2	a	4	0x7ff6d6601f00		
32	2	:	10	0		
33	2	int	4	0x7ff6d66021f0		
34	2	int	17	0		
35	2	;	13	1		
36	2	var	11	0		
37	2	p	4	0x7ff6d6602780		
38	2	:	10	0		
39	2	integer	10	1		
40	2	integer	17	0		
41	2	;	13	1		
42	2	;	17	0		

```

43 3   var      11      0
44 3   numsArray 4      0x7ff6d6700a60
45 3   :        10      0
46 3 array      9      0
47 3   [        14      2
48 3   6        12      0
49 3   ..       9      1
50 3   12       12      0
51 3   ]        14      3
52 3   on       4      0x7ff6d67016c0
53 3   integer  10      1
54 3   ;        13      1
55 4   var      11      0
56 4   q        4      0x7ff6d6701eb0
57 4   :        10      0
58 4   real     10      2
59 4   ;        13      1
60 6   procedure 5      6
61 6   fib1     4      0x7ff6d67028f0
62 6   (        14      0
63 6   aReallyLongInt 4      0x7ff6d6702d00
64 6   aReallyLongInt 16      1
65 6   :        10      0
66 6   integer  10      1
67 6   ;        13      1
68 6   b        4      0x7ff6d6703490
69 6   :        10      0
70 6   real     10      2
71 6   ,        13      0
72 6   ,        17      0
73 6   c        4      0x7ff6d6703bf0
74 6   :        10      0
75 6   real     10      2
76 6   )        14      1
77 6   ;        13      1
78 7 begin     5      0
79 8   if       5      5
80 8   a        4      0x7ff6d6601f00
81 8   <=       3      1
82 8   18       0
83 8   1.20     12      0
84 8   1.20     16      9
85 8   then     5      8
86 8   fib      4      0x7ff6d6600fd0
87 8   :=       2      0
88 8   c        4      0x7ff6d6703bf0

```

```

89      8      18      0
90      9  else      5      2
91      9      18      0
92      9  call      5     10
93      9  fib       4     0x7ff6d6600fd0
94      9  (         14      0
95      9      18      0
96      9    a       4     0x7ff6d6601f00
97      9    -       6      1
98      9      18      0
99      9      01     12      0
100     9      01     16      7
101     9      ,      13      0
102     9      c       4     0x7ff6d6703bf0
103     9      ,      13      0
104     9      18      0
105     9      b       4     0x7ff6d6703490
106     9      +       6      0
107     9      c       4     0x7ff6d6703bf0
108     9      )      14      1
109     9      18      0
110    10  end        5      3
111    10      ;      13      1
112    12  procedure   5      6
113    12  fib2        4     0x7ff6d67088f0
114    12  (          14      0
115    12      a       4     0x7ff6d6601f00
116    12      :      10      0
117    12      integer 10      1
118    12      )      14      1
119    12      ;      13      1
120    13  var        11      0
121    13      b       4     0x7ff6d6703490
122    13      :      10      0
123    13  real       10      2
124    13      ;      13      1
125    13  var        11      0
126    13      c       4     0x7ff6d6703bf0
127    13      :      10      0
128    13  real       10      2
129    13      ;      13      1
130    13  var        11      0
131    13  sum         4     0x7ff6d670a770
132    13      :      10      0
133    13      ;      13      1
134    13      ;      17      0

```

```

135 14 var 11 0
136 14 b 4 0x7ff6d6703490
137 14 : 10 0
138 14 real 10 2
139 14 real 17 0
140 14 ; 13 1
141 14 ; 17 0
142 15 procedure 5 6
143 15 rawr3 4 0x7ff6d670bf00
144 15 ( 14 0
145 15 b 4 0x7ff6d6703490
146 15 : 10 0
147 15 real 10 2
148 15 ) 14 1
149 15 ; 13 1
150 16 var 11 0
151 16 q 4 0x7ff6d6701eb0
152 16 : 10 0
153 16 real 10 2
154 16 ; 13 1
155 17 begin 5 0
156 18 q 4 0x7ff6d6701eb0
157 18 := 2 0
158 18 b 4 0x7ff6d6703490
159 18 + 6 0
160 18 2.0 12 1
161 18 ; 13 1
162 19 call 5 10
163 19 fib2 4 0x7ff6d67088f0
164 19 ( 14 0
165 19 q 4 0x7ff6d6701eb0
166 19 ) 14 1
167 19 18 0
168 19 . 13 2
169 19 . 17 0
170 20 end 5 3
171 20 ; 13 1
172 22 begin 5 0
173 23 a 4 0x7ff6d6601f00
174 23 := 2 0
175 23 a 4 0x7ff6d6601f00
176 23 - 6 1
177 23 1 12 0
178 23 ; 13 1
179 24 fib1 4 0x7ff6d67028f0
180 24 ( 14 0

```

181	24	(	17	0
182	24	3.00	12	0
183	24	3.00	16	9
184	24	)	14	1
185	24	;	13	1
186	25	sum	4	0x7ff6d670a770
187	25	:=	2	0
188	25	1	12	0
189	25	;	13	1
190	26	c	4	0x7ff6d6703bf0
191	26	:=	2	0
192	26	b	4	0x7ff6d6703490
193	26	;	13	1
194	27	while	5	9
195	27	a	4	0x7ff6d6601f00
196	27	>	3	3
197	27	0	12	0
198	27	)	14	1
199	27	)	17	0
200	27	do	5	1
201	27	do	17	0
202	28	call	5	10
203	28	3	12	0
204	28	;	13	1
205	29	begin	5	0
206	30	a	4	0x7ff6d6601f00
207	30	:=	2	0
208	30	a	4	0x7ff6d6601f00
209	30	-	6	1
210	30	1	12	0
211	30	;	13	1
212	31	b	4	0x7ff6d6703490
213	31	:=	2	0
214	31	sum	4	0x7ff6d670a770
215	31	;	13	1
216	31	18	0	
217	32	sum	4	0x7ff6d670a770
218	32	:=	2	0
219	32	c	4	0x7ff6d6703bf0
220	32	18	0	
221	32	+	6	0
222	32	sum	4	0x7ff6d670a770
223	32	;	13	1
224	32	18	0	
225	33	c	4	0x7ff6d6703bf0
226	33	:=	2	0

```

227 33      b      4      0x7ff6d6703490
228 34      end    5      3
229 34      ;      13     1
230 35      fib2   4      0x7ff6d67088f0
231 35      :=     2      0
232 35      sum    4      0x7ff6d670a770
233 35      18     0
234 36      end    5      3
235 36      18     0
236 36      ;      13     1
237 38      procedure 5      6
238 38      init   4      0x7ff6d671a030
239 38      ;      13     1
240 39      begin  5      0
241 40      n      4      0x7ff6d671aae0
242 40      :=     2      0
243 40      12     12     0
244 40      18     0
245 40      ;      13     1
246 41      if     5      5
247 41      (      14     0
248 41      1      12     0
249 41      and    7      2
250 41      2      12     0
251 41      )      14     1
252 41      18     0
253 41      or     6      2
254 41      3      12     0
255 41      then   5      8
256 41      p      4      0x7ff6d6602780
257 41      :=     2      0
258 41      12     12     0
259 41      18     0
260 42      else   5      2
261 42      p      4      0x7ff6d6602780
262 42      :=     2      0
263 42      14     12     0
264 42      18     0
265 42      ;      13     1
266 43      numsArray 4      0x7ff6d6700a60
267 43      [      14     2
268 43      3      12     0
269 43      ]      14     3
270 43      :=     2      0
271 43 15.560     12     0
272 43 15.560     16     9

```

273	43	18	0	
274	43	;	13	1
275	44	q	4	0x7ff6d6701eb0
276	44	:=	2	0
277	44	q	4	0x7ff6d6701eb0
278	44	18	0	
279	44	[	14	2
280	44	18	0	
281	44	4	12	0
282	44	]	14	3
283	44	;	13	1
284	45	q	4	0x7ff6d6701eb0
285	45	[	14	2
286	45	4	12	0
287	45	]	14	3
288	45	:=	2	0
289	45	12	12	0
290	45	18	0	
291	46	end	5	3
292	46	;	13	1
293	48	begin	5	0
294	48	*	7	0
295	48	*	17	0
296	49	call	5	10
297	49	init	4	0x7ff6d671a030
298	49	;	13	1
299	50	call	5	10
300	50	fib2	4	0x7ff6d67088f0
301	50	;	13	1
302	51	call	5	10
303	51	rawr3	4	0x7ff6d670bf00
304	51	(	14	0
305	51	34	12	0
306	51	,	13	0
307	51	56	12	0
308	51	)	14	1
309	51	;	13	1
310	52	end	5	3
311	52	.	13	2
312	54	a	4	0x7ff6d6601f00
313	54	a	17	0
314	55	EOF	1	0

---



## 3.2 Just Semantic Errors

Listing 5: Just Semantic Source Code

---

```
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
9     if a <= 1 then fib := c
10    else call fib (a - 1, c, b + c)
11  end;
12
13  procedure fib2(a : integer);
14    var b : real; var c : real; var sum : integer;
15    var b : real;
16    procedure rawr3(b : real);
17      var q : real;
18      begin
19        q := b + sum;
20        call fib2(q)
21      end;
22
23    begin
24      q := not 3;
25      q := (3 < 4) and (3.6 < p);
26      a := a - 1;
27      call fib1(3.2, 1);
28      sum := 1;
29      c := b;
30      while not (a > 0) do
31        call rawr3(b * 4);
32      begin
33        a := a - 1;
34        b := sum;
35        sum := c + sum;
36        c := b
37      end;
38      fib2 := sum
39    end;
40
41  procedure init;
42    begin
```

```

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

```

---

Listing 6: Just Semantic Listing File

---

```

1      1 program fib(input, output);
2      2   var a: integer; var p: integer;
3      3   var numsArray : array [6..12] of integer;
4      4   var q: real;
5      5
6      6   procedure fib1(aLongInt : integer; b : real;
7      7                       c : integer; d: integer);
8      8       begin
9      9           if a <= 1 then fib := c
10 SEMERR: Cannot assign to ID 'fib' of type 'PROGRAM'!
11      10           else call fib (a - 1, c, b + c)
12 SEMERR: Procedure 'fib' not in scope!
13 SEMERR: Attempt to add incompatible types REAL and INT
14      11       end;
15      12
16      13   procedure fib2(a : integer);
17      14       var b : real; var c : real; var sum :
18      15       integer;
19 SEMERR: A variable named 'b' is already defined in the
20      16       procedure rawr3(b : real);
21      17       var q : real;
22      18       begin
23      19           q := b + sum;
24 SEMERR: Attempt to add incompatible types REAL and INT
25      20           call fib2(q)
26 SEMERR: Expected type INT, not REAL!

```

```

27         21         end;
28         22
29         23         begin
30         24             q := not 3;
31 SEMERR: Expected BOOL use with 'not', received INT!
32         25             q := (3 < 4) and (3.6 < p);
33 SEMERR: Attempt to compare incompatible types REAL and
        INT!
34         26             a := a - 1;
35         27             call fib1(3.2, 1);
36 SEMERR: Expected type INT, not REAL!
37 SEMERR: Expected type REAL, not INT!
38 SEMERR: Expected INT, not the end of the parameters!
39         28             sum := 1;
40         29             c := b;
41         30             while not (a > 0) do
42         31                 call rawr3(b * 4);
43 SEMERR: Attempt to multiply or divide incompatible
        types REAL and INT!
44         32             begin
45         33                 a := a - 1;
46         34                 b := sum;
47 SEMERR: Attempt to convert type REAL into type INT in
        assignment!
48         35                 sum := c + sum;
49 SEMERR: Attempt to add incompatible types REAL and INT
        !
50         36                 c := b
51         37             end;
52         38             fib2 := sum
53 SEMERR: Cannot assign to ID 'fib2' of type 'PROCEDURE
        '!
54         39         end;
55         40
56         41         procedure init;
57         42             begin
58         43                 n := 12;
59 SEMERR: ID 'n' not in scope!
60         44                 if not (1 < 74) or 3 then p:=(1 > 2) and
        (3 < n)
61 SEMERR: Expected BOOL and BOOL for use with 'or',
        received BOOL and INT!
62 SEMERR: No variable 'n' is defined in the local scope!
63         45                 else p:=not 2;
64 SEMERR: Expected BOOL use with 'not', received INT!
65         46                 numsArray[3] := 15.56;

```

```

66 SEMERR: Attempt to convert type INT into type REAL in
    assignment!
67     47     q := q[4];
68 SEMERR: Attempt to index variable of type REAL!
69     48     q[4] := 12
70 SEMERR: Attempt to index variable of type REAL!
71     49     end;
72     50
73     51     begin
74     52         call init;
75     53         call fib2;
76 SEMERR: Expected an argument of type INT!
77     54         call rawr3(34, 56)
78 SEMERR: Procedure 'rawr3' not in scope!
79     55     end.

```

---

Listing 7: Just Semantic Mem File

---

	ID	Memory Offset
1	a	0
2	p	4
3	numsArray	8
4	q	36
5	b	0
6	c	8
7	sum	16
8	q	0
9		

---

Listing 8: Just Semantic Token File

---

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

```

17 17 SYNERR
18 18 SEMERR
19      LineLexeme      Token Attribute      Token Type
20      1      program      5      7
21      1      fib      4      0x7fba52500fd0
22      1      (      14      0
23      1      input      4      0x7fba525012e0
24      1      ,      13      0
25      1      output      4      0x7fba52501660
26      1      )      14      1
27      1      ;      13      1
28      2      var      11      0
29      2      a      4      0x7fba52501d30
30      2      :      10      0
31      2      integer      10      1
32      2      ;      13      1
33      2      var      11      0
34      2      p      4      0x7fba52502530
35      2      :      10      0
36      2      integer      10      1
37      2      ;      13      1
38      3      var      11      0
39      3      numsArray      4      0x7fba52502f50
40      3      :      10      0
41      3      array      9      0
42      3      [      14      2
43      3      6      12      0
44      3      ..      9      1
45      3      12      12      0
46      3      ]      14      3
47      3      of      9      2
48      3      integer      10      1
49      3      ;      13      1
50      4      var      11      0
51      4      q      4      0x7fba525043b0
52      4      :      10      0
53      4      real      10      2
54      4      ;      13      1
55      6      procedure      5      6
56      6      fib1      4      0x7fba52504e20
57      6      (      14      0
58      6      aLongInt      4      0x7fba52505170
59      6      :      10      0
60      6      integer      10      1
61      6      ;      13      1
62      6      b      4      0x7fba52505830

```

```

63      6      :      10      0
64      6  real      10      2
65      6      ;      13      1
66      7      c      4      0x7fba52506790
67      7      :      10      0
68      7      integer 10      1
69      7      ;      13      1
70      7      d      4      0x7fba52506e50
71      7      :      10      0
72      7      integer 10      1
73      7      )      14      1
74      7      ;      13      1
75      8  begin      5      0
76      9      if      5      5
77      9      a      4      0x7fba52501d30
78      9      <=      3      1
79      9      1      12      0
80      9      then     5      8
81      9      fib      4      0x7fba52500fd0
82      9      :=      2      0
83      9      c      4      0x7fba52506790
84      9      18      0
85      10  else      5      2
86      10  call      5      10
87      10  fib      4      0x7fba52500fd0
88      10  (      14      0
89      10  18      0
90      10  a      4      0x7fba52501d30
91      10  -      6      1
92      10  1      12      0
93      10  ,      13      0
94      10  c      4      0x7fba52506790
95      10  ,      13      0
96      10  b      4      0x7fba52505830
97      10  +      6      0
98      10  c      4      0x7fba52506790
99      10  )      14      1
100     10  18      0
101     11  end      5      3
102     11  ;      13      1
103     13  procedure  5      6
104     13  fib2      4      0x7fba5250b340
105     13  (      14      0
106     13  a      4      0x7fba52501d30
107     13  :      10      0
108     13  integer  10      1

```

109	13	)	14	1
110	13	;	13	1
111	14	var	11	0
112	14	b	4	0x7fba52505830
113	14	:	10	0
114	14	real	10	2
115	14	;	13	1
116	14	var	11	0
117	14	c	4	0x7fba52506790
118	14	:	10	0
119	14	real	10	2
120	14	;	13	1
121	14	var	11	0
122	14	sum	4	0x7fba5250d1c0
123	14	:	10	0
124	14	integer	10	1
125	14	;	13	1
126	15	var	11	0
127	15	b	4	0x7fba52505830
128	15	:	10	0
129	15	real	10	2
130	15	;	13	1
131	15	18	0	
132	16	procedure	5	6
133	16	rawr3	4	0x7fba5250e950
134	16	(	14	0
135	16	b	4	0x7fba52505830
136	16	:	10	0
137	16	real	10	2
138	16	)	14	1
139	16	;	13	1
140	17	var	11	0
141	17	q	4	0x7fba525043b0
142	17	:	10	0
143	17	real	10	2
144	17	;	13	1
145	18	begin	5	0
146	19	q	4	0x7fba525043b0
147	19	:=	2	0
148	19	b	4	0x7fba52505830
149	19	+	6	0
150	19	sum	4	0x7fba5250d1c0
151	19	;	13	1
152	19	18	0	
153	20	call	5	10
154	20	fib2	4	0x7fba5250b340

155	20	(	14	0
156	20	q	4	0x7fba525043b0
157	20	)	14	1
158	20	18	0	
159	21	end	5	3
160	21	;	13	1
161	23	begin	5	0
162	24	q	4	0x7fba525043b0
163	24	:=	2	0
164	24	not	15	0
165	24	3	12	0
166	24	;	13	1
167	24	18	0	
168	25	q	4	0x7fba525043b0
169	25	:=	2	0
170	25	(	14	0
171	25	3	12	0
172	25	<	3	0
173	25	4	12	0
174	25	)	14	1
175	25	and	7	2
176	25	(	14	0
177	25	3.6	12	1
178	25	<	3	0
179	25	p	4	0x7fba52502530
180	25	)	14	1
181	25	18	0	
182	25	;	13	1
183	26	a	4	0x7fba52501d30
184	26	:=	2	0
185	26	a	4	0x7fba52501d30
186	26	-	6	1
187	26	1	12	0
188	26	;	13	1
189	27	call	5	10
190	27	fib1	4	0x7fba52504e20
191	27	(	14	0
192	27	3.2	12	1
193	27	,	13	0
194	27	18	0	
195	27	1	12	0
196	27	)	14	1
197	27	18	0	
198	27	18	0	
199	27	;	13	1
200	28	sum	4	0x7fba5250d1c0



201	28	:=	2	0
202	28	1	12	0
203	28	;	13	1
204	29	c	4	0x7fba52506790
205	29	:=	2	0
206	29	b	4	0x7fba52505830
207	29	;	13	1
208	30	while	5	9
209	30	not	15	0
210	30	(	14	0
211	30	a	4	0x7fba52501d30
212	30	>	3	3
213	30	0	12	0
214	30	)	14	1
215	30	do	5	1
216	31	call	5	10
217	31	rawr3	4	0x7fba5250e950
218	31	(	14	0
219	31	b	4	0x7fba52505830
220	31	*	7	0
221	31	4	12	0
222	31	)	14	1
223	31	18	0	
224	31	;	13	1
225	32	begin	5	0
226	33	a	4	0x7fba52501d30
227	33	:=	2	0
228	33	a	4	0x7fba52501d30
229	33	-	6	1
230	33	1	12	0
231	33	;	13	1
232	34	b	4	0x7fba52505830
233	34	:=	2	0
234	34	sum	4	0x7fba5250d1c0
235	34	;	13	1
236	34	18	0	
237	35	sum	4	0x7fba5250d1c0
238	35	:=	2	0
239	35	c	4	0x7fba52506790
240	35	+	6	0
241	35	sum	4	0x7fba5250d1c0
242	35	;	13	1
243	35	18	0	
244	36	c	4	0x7fba52506790
245	36	:=	2	0
246	36	b	4	0x7fba52505830

```

247 37     end         5      3
248 37     ;          13     1
249 38 fib2          4      0x7fba5250b340
250 38     :=         2      0
251 38     sum        4      0x7fba5250d1c0
252 38     18         0
253 39     end         5      3
254 39     ;          13     1
255 41     procedure   5      6
256 41     init        4      0x7fba5260f370
257 41     ;          13     1
258 42 begin          5      0
259 43     n           4      0x7fba5260fe20
260 43     :=         2      0
261 43     12         12     0
262 43     18         0
263 43     ;          13     1
264 44     if          5      5
265 44     not        15     0
266 44     (          14     0
267 44     1          12     0
268 44     <          3      0
269 44     74         12     0
270 44     )          14     1
271 44     or          6      2
272 44     3          12     0
273 44     then        5      8
274 44     18         0
275 44     p           4      0x7fba52502530
276 44     :=         2      0
277 44     (          14     0
278 44     1          12     0
279 44     >          3      3
280 44     2          12     0
281 44     )          14     1
282 44     and         7      2
283 44     (          14     0
284 44     3          12     0
285 44     <          3      0
286 44     n           4      0x7fba5260fe20
287 44     )          14     1
288 44     18         0
289 45 else          5      2
290 45     p           4      0x7fba52502530
291 45     :=         2      0
292 45     not        15     0

```

293	45	2	12	0	
294	45	;	13	1	
295	45	18	0		
296	46	numsArray	4		0x7fba52502f50
297	46	[	14	2	
298	46	3	12	0	
299	46	]	14	3	
300	46	:=	2	0	
301	46	15.56	12	1	
302	46	;	13	1	
303	46	18	0		
304	47	q	4		0x7fba525043b0
305	47	:=	2	0	
306	47	q	4		0x7fba525043b0
307	47	[	14	2	
308	47	4	12	0	
309	47	]	14	3	
310	47	;	13	1	
311	47	18	0		
312	48	q	4		0x7fba525043b0
313	48	[	14	2	
314	48	4	12	0	
315	48	]	14	3	
316	48	:=	2	0	
317	48	18	0		
318	48	12	12	0	
319	49	end	5	3	
320	49	;	13	1	
321	51	begin	5	0	
322	52	call	5	10	
323	52	init	4		0x7fba5260f370
324	52	;	13	1	
325	53	call	5	10	
326	53	fib2	4		0x7fba5250b340
327	53	;	13	1	
328	53	18	0		
329	54	call	5	10	
330	54	rawr3	4		0x7fba5250e950
331	54	(	14	0	
332	54	18	0		
333	54	34	12	0	
334	54	,	13	0	
335	54	56	12	0	
336	54	)	14	1	
337	55	end	5	3	
338	55	.	13	2	

339	56	EOF	1	0
-----	----	-----	---	---

---

### 3.3 Error-Free

Listing 9: Error-Free Source Code

```
1
2 program test (input, output);
3   var a : integer;
4   var b : real;
5   var c : array [1..2] of integer;
6
7   procedure proc1(x:integer; y:real;
8                 z:array [1..2] of integer; q: real);
9     var d: integer;
10    begin
11      a:= 2;
12      z[a] := 4;
13      c[3] := 3
14    end;
15
16    procedure proc2(x: integer; y: integer);
17      var e: real;
18
19      procedure proc3(n: integer; z: real);
20        var e: integer;
21
22        procedure proc4(a: integer; z: array [1..3] of
23          real);
24          var x: integer;
25          begin
26            a:= e
27          end;
28
29          begin
30            a:= e;
31            e:= c[e]
32          end;
33
34          begin
35            call proc1(x, e, c, b);
36            call proc3(c[1], e);
37            e := e + 4.44;
38            a:= (a mod y) div x;
39            while ((a >= 4) and ((b <= e)
40              or (not (a = c[a])))) do
41              begin
42                a:= c[a] + 1
```

```

42         end
43     end;
44
45 begin
46     call proc2(c[4], c[5]);
47     call proc2(c[4], 2);
48     if (a < 2) then a:= 1 else a := a + 2;
49     if (b > 4.2) then a := c[a]
50 end.

```

---

Listing 10: Error-Free Listing File

---

```

1      1
2      2 program test (input, output);
3      3   var a : integer;
4      4   var b : real;
5      5   var c : array [1..2] of integer;
6      6
7      7   procedure proc1(x:integer; y:real;
8      8                       z:array [1..2] of integer; q
9      9       : real);
10     10   var d: integer;
11     11   begin
12     12       a:= 2;
13     13       z[a] := 4;
14     14       c[3] := 3
15     15   end;
16     16   procedure proc2(x: integer; y: integer);
17     17       var e: real;
18     18
19     19   procedure proc3(n: integer; z: real);
20     20       var e: integer;
21     21
22     22   procedure proc4(a: integer; z: array
23     23       [1..3] of real);
24     24       var x: integer;
25     25       begin
26     26           a:= e
27     27       end;
28     28   begin
29     29       a:= e;
30     30       e:= c[e]
31     31   end;
32     32

```

```

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

```

---

Listing 11: Error-Free Mem File

---

	ID	Memory	Offset
1	a		0
2	b		4
3	c		12
4	d		0
5	e		0
6	e		0
7	e		0
8	x		0

---

Listing 12: Error-Free Token File

---

```

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

```

14	14	GROUP			
15	15	INVERSE			
16	16	LEXERR			
17	17	SYNERR			
18	18	SEMERR			
19		LineLexeme	Token	Attribute	Token Type
20	2	program	5	7	
21	2	test	4	0x7f8012600dc0	
22	2	(	14	0	
23	2	input	4	0x7f8012601160	
24	2	,	13	0	
25	2	output	4	0x7f80126014e0	
26	2	)	14	1	
27	2	;	13	1	
28	3	var	11	0	
29	3	a	4	0x7f8012601bb0	
30	3	:	10	0	
31	3	integer	10	1	
32	3	;	13	1	
33	4	var	11	0	
34	4	b	4	0x7f8012602560	
35	4	:	10	0	
36	4	real	10	2	
37	4	;	13	1	
38	5	var	11	0	
39	5	c	4	0x7f8012602eb0	
40	5	:	10	0	
41	5	array	9	0	
42	5	[	14	2	
43	5	1	12	0	
44	5	..	9	1	
45	5	2	12	0	
46	5	]	14	3	
47	5	of	9	2	
48	5	integer	10	1	
49	5	;	13	1	
50	7	procedure	5	6	
51	7	proc1	4	0x7f8012604480	
52	7	(	14	0	
53	7	x	4	0x7f80126046f0	
54	7	:	10	0	
55	7	integer	10	1	
56	7	;	13	1	
57	7	y	4	0x7f8012604c90	
58	7	:	10	0	
59	7	real	10	2	



```

60      7      ;      13      1
61      8      z      4      0x7f8012605bf0
62      8      :      10      0
63      8 array      9      0
64      8      [      14      2
65      8      1      12      0
66      8      ..      9      1
67      8      2      12      0
68      8      ]      14      3
69      8      of      9      2
70      8      integer 10      1
71      8      ;      13      1
72      8      q      4      0x7f8012606be0
73      8      :      10      0
74      8 real      10      2
75      8      )      14      1
76      8      ;      13      1
77      9      var      11      0
78      9      d      4      0x7f80126076c0
79      9      :      10      0
80      9      integer 10      1
81      9      ;      13      1
82 10 begin      5      0
83 11      a      4      0x7f8012601bb0
84 11      :=      2      0
85 11      2      12      0
86 11      ;      13      1
87 12      z      4      0x7f8012605bf0
88 12      [      14      2
89 12      a      4      0x7f8012601bb0
90 12      ]      14      3
91 12      :=      2      0
92 12      4      12      0
93 12      ;      13      1
94 13      c      4      0x7f8012602eb0
95 13      [      14      2
96 13      3      12      0
97 13      ]      14      3
98 13      :=      2      0
99 13      3      12      0
100 14 end      5      3
101 14      ;      13      1
102 16      procedure 5      6
103 16 proc2      4      0x7f801260ad50
104 16      (      14      0
105 16      x      4      0x7f80126046f0

```

```

106 16      :      10      0
107 16      integer    10      1
108 16      ;      13      1
109 16      y      4      0x7f8012604c90
110 16      :      10      0
111 16      integer    10      1
112 16      )      14      1
113 16      ;      13      1
114 17      var      11      0
115 17      e      4      0x7f801260c180
116 17      :      10      0
117 17      real      10      2
118 17      ;      13      1
119 19      procedure   5      6
120 19      proc3      4      0x7f801260cdc0
121 19      (      14      0
122 19      n      4      0x7f801260d030
123 19      :      10      0
124 19      integer    10      1
125 19      ;      13      1
126 19      z      4      0x7f8012605bf0
127 19      :      10      0
128 19      real      10      2
129 19      )      14      1
130 19      ;      13      1
131 20      var      11      0
132 20      e      4      0x7f801260c180
133 20      :      10      0
134 20      integer    10      1
135 20      ;      13      1
136 22      procedure   5      6
137 22      proc4      4      0x7f801260f070
138 22      (      14      0
139 22      a      4      0x7f8012601bb0
140 22      :      10      0
141 22      integer    10      1
142 22      ;      13      1
143 22      z      4      0x7f8012605bf0
144 22      :      10      0
145 22      array      9      0
146 22      [      14      2
147 22      1      12      0
148 22      ..      9      1
149 22      3      12      0
150 22      ]      14      3
151 22      of      9      2

```

152	22	real	10	2	
153	22	)	14	1	
154	22	;	13	1	
155	23	var	11	0	
156	23	x	4		0x7f80126046f0
157	23	:	10	0	
158	23	integer	10	1	
159	23	;	13	1	
160	24	begin	5	0	
161	25	a	4		0x7f8012601bb0
162	25	:=	2	0	
163	25	e	4		0x7f801260c180
164	26	end	5	3	
165	26	;	13	1	
166	28	begin	5	0	
167	29	a	4		0x7f8012601bb0
168	29	:=	2	0	
169	29	e	4		0x7f801260c180
170	29	;	13	1	
171	30	e	4		0x7f801260c180
172	30	:=	2	0	
173	30	c	4		0x7f8012602eb0
174	30	[	14	2	
175	30	e	4		0x7f801260c180
176	30	]	14	3	
177	31	end	5	3	
178	31	;	13	1	
179	33	begin	5	0	
180	34	call	5	10	
181	34	proc1	4		0x7f8012604480
182	34	(	14	0	
183	34	x	4		0x7f80126046f0
184	34	,	13	0	
185	34	e	4		0x7f801260c180
186	34	,	13	0	
187	34	c	4		0x7f8012602eb0
188	34	,	13	0	
189	34	b	4		0x7f8012602560
190	34	)	14	1	
191	34	;	13	1	
192	35	call	5	10	
193	35	proc3	4		0x7f801260cdc0
194	35	(	14	0	
195	35	c	4		0x7f8012602eb0
196	35	[	14	2	
197	35	1	12	0	

198	35	]	14	3
199	35	,	13	0
200	35	e	4	0x7f801260c180
201	35	)	14	1
202	35	;	13	1
203	36	e	4	0x7f801260c180
204	36	:=	2	0
205	36	e	4	0x7f801260c180
206	36	+	6	0
207	36	4.44	12	1
208	36	;	13	1
209	37	a	4	0x7f8012601bb0
210	37	:=	2	0
211	37	(	14	0
212	37	a	4	0x7f8012601bb0
213	37	mod	7	4
214	37	y	4	0x7f8012604c90
215	37	)	14	1
216	37	div	7	3
217	37	x	4	0x7f80126046f0
218	37	;	13	1
219	38	while	5	9
220	38	(	14	0
221	38	(	14	0
222	38	a	4	0x7f8012601bb0
223	38	>=	3	4
224	38	4	12	0
225	38	)	14	1
226	38	and	7	2
227	38	(	14	0
228	38	(	14	0
229	38	b	4	0x7f8012602560
230	38	<=	3	1
231	38	e	4	0x7f801260c180
232	38	)	14	1
233	39	or	6	2
234	39	(	14	0
235	39	not	15	0
236	39	(	14	0
237	39	a	4	0x7f8012601bb0
238	39	=	3	2
239	39	c	4	0x7f8012602eb0
240	39	[	14	2
241	39	a	4	0x7f8012601bb0
242	39	]	14	3
243	39	)	14	1

244	39	)	14	1
245	39	)	14	1
246	39	)	14	1
247	39	do	5	1
248	40	begin	5	0
249	41	a	4	0x7f8012601bb0
250	41	:=	2	0
251	41	c	4	0x7f8012602eb0
252	41	[	14	2
253	41	a	4	0x7f8012601bb0
254	41	]	14	3
255	41	+	6	0
256	41	1	12	0
257	42	end	5	3
258	43	end	5	3
259	43	;	13	1
260	45	begin	5	0
261	46	call	5	10
262	46	proc2	4	0x7f801260ad50
263	46	(	14	0
264	46	c	4	0x7f8012602eb0
265	46	[	14	2
266	46	4	12	0
267	46	]	14	3
268	46	,	13	0
269	46	c	4	0x7f8012602eb0
270	46	[	14	2
271	46	5	12	0
272	46	]	14	3
273	46	)	14	1
274	46	;	13	1
275	47	call	5	10
276	47	proc2	4	0x7f801260ad50
277	47	(	14	0
278	47	c	4	0x7f8012602eb0
279	47	[	14	2
280	47	4	12	0
281	47	]	14	3
282	47	,	13	0
283	47	2	12	0
284	47	)	14	1
285	47	;	13	1
286	48	if	5	5
287	48	(	14	0
288	48	a	4	0x7f8012601bb0
289	48	<	3	0

290	48	2	12	0
291	48	)	14	1
292	48	then	5	8
293	48	a	4	0x7f8012601bb0
294	48	:=	2	0
295	48	1	12	0
296	48	else	5	2
297	48	a	4	0x7f8012601bb0
298	48	:=	2	0
299	48	a	4	0x7f8012601bb0
300	48	+	6	0
301	48	2	12	0
302	48	;	13	1
303	49	if	5	5
304	49	(	14	0
305	49	b	4	0x7f8012602560
306	49	>	3	3
307	49	4.2	12	1
308	49	)	14	1
309	49	then	5	8
310	49	a	4	0x7f8012601bb0
311	49	:=	2	0
312	49	c	4	0x7f8012602eb0
313	49	[	14	2
314	49	a	4	0x7f8012601bb0
315	49	]	14	3
316	50	end	5	3
317	50	.	13	2
318	51	EOF	1	0

---

## Appendix 2: Program Listings

Listing 13: compiler.c

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<stdbool.h>
4
5 #include "dataStructures/
  linkedList/linkedList.h"
6 #include "errorHandler/
  errorHandler.h"
7 #include "globals/globals.h"
8 #include "handler/handler.h"
9 #include "parser/parser.h"
10
11 // Global file constants
12 static const char TOKEN_PATH[]
  = "out/tokens.dat";
13 static const char LISTING_PATH
  [] = "out/listing.txt";
14 static const char MEM_PATH[] =
  "out/mem.txt";
15 static const char RESWORD_PATH
  [] = "compiler/data/
  reswords.dat";
16
17 // Returns 1 on failure, 0 on
  success.
18 int init(char* sourcePath) {
19 return initializeGlobals() &&
  initializeErrorHandler() &&
20 initializeHandler(sourcePath,
  RESWORD_PATH, LISTING_PATH,
  TOKEN_PATH, MEM_PATH)
21 ? 0 : 1;
22 }
23
24 int run()
25 {
26 generateParseTree();
27
28 return 0;
29 }

```

```

30
31 int main(int argc, char *argv
  []) {
32 if (argc != 2) {
33 fprintf(stderr, "%s\n", "
  Expected exactly one file
  to compile!");
34 } else {
35 if (init(argv[1]) == 0) {
36 if (run() != 0)
37 fprintf(stderr, "%s\n", "Run
  failed. Could not terminate
  properly.");
38 } else {
39 fprintf(stderr, "%s\n", "
  Initialization process
  failed in tokenizer.");
40 }
41 }
42 return 0;
43 }

```

Listing 14: declarationsTree.h

```

1 #ifndef DECLARATIONS_TREE_H
2 #define DECLARATIONS_TREE_H
3
4 #include <stdbool.h>
5
6 #include "../tokenizer/
  tokens.h"
7
8 typedef struct tree_node {
9 char* lex; // The lexeme
10 LangType type; // The type
11 union {
12 bool param; // True if param
13 bool add_right; // True if add
  right to green node
14 };
15
16 struct tree_node* left;
17 struct tree_node* right;
18 struct tree_node* parent;
19 } tree_node;
20

```

<pre> 21 typedef struct LinkedTree { 22     struct node* head; 23 } DeclarationsTree; 24 25 // Green nodes designate       scopes, and blue nodes       designate variables 26 bool check_add_node(Token*       decl); 27 tree_node* get_last_green_node       (); 28 tree_node*       start_param_matching(Token*       id); 29 void reached_end_of_scope(); 30 LangType get_type(Token* id); 31 32 #endif // DECLARATIONS_TREE_H </pre>	<pre>       of the linked list 22 void* pop(LinkedList* list); 23 24 #endif // LINKED_H_ </pre> <hr/> <p style="text-align: center;">Listing 16: errorHandler.h</p> <hr/> <pre> 1 #ifndef ERROR_HANDLER_H 2 #define ERROR_HANDLER_H 3 #include "../tokenizer/tokens.       h" 4 5 extern const char* lexErrs[]; 6 char* synErr; 7 LinkedList* semErrs; 8 9 void throw_sem_error(char* msg       ); 10 void throw_syn_error(Token*       received, const Token**       expected, int exp_size); 11 void throw_lex_error(enum       TokenType attribute, int       aspect, int start, int       length); 12 int initializeErrorHandler(); 13 14 Token* getNextErrorToken(); 15 16 #endif // ERROR_HANDLER_H </pre> <hr/> <p style="text-align: center;">Listing 17: globals.h</p> <hr/> <pre> 1 #ifndef GLOBALS_H 2 #define GLOBALS_H 3 4 extern int START; 5 extern int LINE; 6 extern char* BUFFER; 7 8 void updateLine(char* line); 9 int initializeGlobals(); 10 11 #endif // GLOBALS_H </pre>
---	--

---

Listing 15: linkedlist.h

---

```

1 #ifndef LINKED_H_
2 #define LINKED_H_
3
4 // Behaves like a stack
5 struct node {
6     void* data;
7     struct node* next;
8 };
9
10 typedef struct LinkedNodes {
11     struct node* head;
12     int size;
13 } LinkedList;
14
15 // Add an item to the front of
      the linked list
16 int add(LinkedList* list, void
      * data, int size);
17 // For use as a queue; slow,
      do not use
18 int addLast(LinkedList* list,
      void *data, int size);
19
20
21 // Pop an item from the front

```



Listing 18: handler.h

---

```

1 #ifndef HANDLER_H
2 #define HANDLER_H
3
4 #include<stdbool.h>
5 #include "../tokenizer/tokens.
  h"
6
7 int initializeHandler(const
  char* sourcePath, const
  char* resPath,
8 const char* listingPath, const
  char* tokenPath,
9 const char* memPath);
10 bool handleToken(Token* token)
  ;
11 void outputWidth(char* lex,
  int width);
12
13 #endif // HANDLER_H

```

---

Listing 19: parser.h

---

```

1 #ifndef PARSER_H
2 #define PARSER_H
3 #include<stdbool.h>
4
5 int generateParseTree();
6 Token* match(const Token*
  source, bool strict);
7 void require_sync(const Token*
  sync_set[], int size,
8 const Token* first_set[], int
  first_size);
9
10 #endif // PARSER_H

```

---

Listing 20: productions.h

---

```

1 #ifndef voidS_H
2 #define voidS_H
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include "../globals/
  globals.h"

```

---

```

7 #include "../tokenizer/
  tokens.h"
8 #include "../dataStructures
  /declarationsTree/
  declarationsTree.h"
9 #include "../errorHandler/
  errorHandler.h"
10
11 extern Token* current_tok;
12
13 // All of these must have
  their follows added to the
  sync set
14 void program();
15 void id_list();
16 void id_list_tail();
17 void declarations();
18 LangType type();
19 LangType standard_type();
20 void subprogram_declarations()
  ;
21 void subprogram_declaration();
22 bool subprogram_head();
23 void arguments();
24 void parameter_list();
25 void parameter_list_tail();
26 void compound_statement();
27 void optional_statements();
28 void statement_list();
29 void statement_list_tail();
30 void statement();
31 void else_tail();
32 LangType variable();
33 LangType array_access(LangType
  id_type);
34 void procedure_statement();
35 void optional_expressions(
  tree_node* to_match, bool
  should_error);
36 void expression_list(tree_node
  * to_match, bool
  should_error);
37 void expression_list_tail(
  tree_node* to_match, bool
  should_error);
38 LangType expression();

```

```

39 LangType related_expression();
40 LangType simple_expression();
41 LangType
    simple_expression_tail();
42 LangType term();
43 LangType term_tail();
44 LangType factor();
45 LangType factor_tail();
46 void sign();
47
48 #endif // voidS_H

```

---

Listing 21: symbolTable.h

---

```

1 #ifndef SYMBOL_TABLE_H
2 #define SYMBOL_TABLE_H
3
4 int initSymbolTable();
5 char* checkSymbolTable(char*
    name);
6 char* pushToSymbolTable(char*
    name, size_t length);
7
8 #endif // SYMBOL_TABLE_H

```

---

Listing 22: machines.h

---

```

1 #ifndef MACHINES_H
2 #define MACHINES_H
3 #include <stdio.h>
4 #include "../tokens.h"
5
6 typedef int (*machine)(Token*,
    char*, int);
7
8 int intMachine(Token* storage,
    char* str, int start);
9 int realMachine(Token* storage
    , char* str, int start);
10 int longRealMachine(Token*
    storage, char* str, int
    start);
11 int grouping(Token* storage,
    char* str, int start);
12 int catchall(Token* storage,
    char* str, int start);

```

```

13 int mulop(Token* storage, char
    * str, int start);
14 int addop(Token* storage, char
    * str, int start);
15 int whitespace(Token* storage,
    char* str, int start);
16 int relop(Token* storage, char
    * str, int start);
17
18 int idres(Token* storage, char
    * str, int start);
19 int initIDResMachine(FILE*
    resFile);
20
21 extern const machine machines
    [];
22 #endif // MACHINES_H

```

---

Listing 23: tokenizer.h

---

```

1 #ifndef PROCESSOR_H_
2 #define PROCESSOR_H_
3 #include<stdio.h>
4 #include "tokens.h"
5
6 Token* getNextToken();
7 int initializeTokens(FILE*
    resFile);
8
9 #endif // PROCESSOR_H_

```

---

Listing 24: tokens.h

---

```

1 #ifndef TOKENS_H
2 #define TOKENS_H
3
4 #include<stdbool.h>
5
6 #include "../dataStructures/
    linkedList/linkedList.h"
7
8 // Must have a boolean
    indicating whether it is a
    parameter or not
9 typedef enum LangType {ERR,
    REAL, INT, BOOL, PNAME,
    PPNAME,

```

```

10  PROC, AINT, AREAL} LangType;
11
12  enum TokenType {NOOP, FILEEND,
13                  ASSIGNOP, RELOP, ID,
14  CONTROL, ADDOP, MULOP, WS,
15                  ARRAY, TYPE,
16  VAR, NUM, PUNC, GROUP, INVERSE
17  ,
18  LEXERR, SYNERR, SEMERR};
19
20  // The token data type
21  typedef struct T_Type {
22  enum TokenType attribute; //
23      Attribute
24
25  union { // Aspect or character
26      pointer
27      int aspect;
28      char* id;
29  };
30
31  int start; // Start in the
32      line
33  int length; // Length of the
34      lexeme
35
36  union { // Value of the number
37      , or length of the array
38      int int_val;
39      double real_val;
40      int array_length;
41  };
42
43  LangType type; // The type of
44      the token
45  bool param; // Whether the
46      token is a parameter or not
47
48  } Token;
49
50  extern const Token eof_tok;
51  extern const Token lparen_tok;
52  extern const Token rparen_tok;
53  extern const Token plus_tok;
54  extern const Token comma_tok;
55  extern const Token minus_tok;
56
57  extern const Token semic_tok;
58  extern const Token colon_tok;
59  extern const Token period_tok;
60  extern const Token dotdot_tok;
61  extern const Token lbrac_tok;
62  extern const Token rbrac_tok;
63  extern const Token addop_tok;
64  extern const Token array_tok;
65  extern const Token
66      assignop_tok;
67  extern const Token begin_tok;
68  extern const Token call_tok;
69  extern const Token do_tok;
70  extern const Token else_tok;
71  extern const Token end_tok;
72  extern const Token id_tok;
73  extern const Token if_tok;
74  extern const Token integer_tok
75      ;
76  extern const Token
77      integer_val_tok;
78  extern const Token of_tok;
79  extern const Token
80      real_val_tok;
81  extern const Token mulop_tok;
82  extern const Token not_tok;
83  extern const Token num_tok;
84  extern const Token
85      procedure_tok;
86  extern const Token program_tok
87      ;
88  extern const Token real_tok;
89  extern const Token relop_tok;
90  extern const Token then_tok;
91  extern const Token var_tok;
92  extern const Token while_tok;
93
94  extern const char* catNames
95      [19];
96  extern const char* typeNames
97      [9];
98  const Token* getTokenFromLex(
99      char* lex);
100 const char* getLexFromToken(
101     Token* token, bool strict);

```

```

82 // The type; else null if
    impossible
83 LangType convert_to_array(
    LangType type);
84 LangType convert_from_array(
    LangType type);
85
86 // Returns the type produced
    by the operation
87 LangType type_lookup(LangType
    first, LangType second,
    Token* op);
88
89 // Returns true if the tokens
    are equivalent, false
    otherwise
90 bool tokens_equal(const Token*
    p1, Token* p2, bool strict
    );
91
92
93 #endif // TOKENS_H

```

Listing 25: declarationsTree.c

```

1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include "../handler/
    handler.h"
5 #include "../errorHandler/
    errorHandler.h"
6 #include "../globals/
    globals.h"
7 #include "declarationsTree.h"
8
9 static int offset = 0;
10
11 static DeclarationsTree*
    d_tree = NULL;
12 static tree_node* bottom_node
    = NULL;
13
14 static LinkedList*
    green_node_stack = NULL;
15
16 static void initialize_d_tree
    () {
17     d_tree = malloc(sizeof(*d_tree
    ));
18     green_node_stack = malloc(
    sizeof(*green_node_stack));
19 }
20
21 static int get_width(Token*
    val) {
22     switch (val -> type) {
23     case INT: return 4;
24     case REAL: return 8;
25
26     case AINT: return 4*(val ->
    array_length);
27     case AREAL: return 8*(val ->
    array_length);
28
29     default: return 1000000;
30     }
31 }
32
33 static bool check_node(char*
    id, bool green) {
34     tree_node* current_node =
    bottom_node;
35     while (current_node != NULL) {
36         // Already exists
37         if (id == current_node -> lex)
38             return true;
39
40         if (!green && (current_node ->
    type == PROC ||
    current_node -> type ==
    PGNAME))
41             return false;
42
43         // We've passed the most
    recent green node, and this
    is a blue one
44         if (!green && (current_node ->
    type == PROC ||
    current_node -> type ==
    PGNAME))
45             break;

```

```

46     current_node = current_node ->
47         parent;
48 }
49
50 return false;
51 }
52
53 static bool
54     check_add_green_node(Token*
55         decl) {
56     if (d_tree == NULL)
57         initialize_d_tree();
58     if (bottom_node == NULL)
59     {
60         tree_node* addition = malloc(
61             sizeof(*addition));
62         addition -> lex = decl -> id;
63         addition -> type = decl ->
64             type;
65         addition -> add_right = false;
66         // Add it to the top of the
67             stack
68         add(green_node_stack, &
69             addition, sizeof(&addition)
70             );
71         addition -> left = NULL;
72         addition -> right = NULL;
73         addition -> parent = NULL;
74         bottom_node = addition;
75         return true;
76     }
77     // Check if it's been declared
78         at all
79     if (check_node(decl -> id,
80         true))
81         return false;
82     offset = 0;
83     // It hasn't been declared;
84         create it
85
86     tree_node* addition = malloc(
87         sizeof(*addition));
88     addition -> lex = decl -> id;
89     addition -> type = decl ->
90         type;
91     addition -> add_right = false;
92     // Add it to the top of the
93         stack
94     add(green_node_stack, &
95         addition, sizeof(&addition)
96         );
97     addition -> left = NULL;
98     addition -> right = NULL;
99     addition -> parent =
100         bottom_node;
101     // Make it the new bottom node
102     if (bottom_node -> add_right
103         == true)
104         bottom_node -> right =
105             addition;
106     else
107         bottom_node -> left = addition
108             ;
109     bottom_node = addition;
110     return true;
111 }
112
113 static bool
114     check_add_blue_node(Token*
115         decl) {
116     // If there's no scope, that's
117         an error!
118     if (d_tree == NULL)
119         return false;
120     // It's been declared in the
121         scope already
122     if (check_node(decl -> id,
123         false))
124         return false;

```

```

113 // It hasn't been declared;
114 // create it
115 tree_node* addition = malloc(
116     sizeof(*addition));
117 addition -> lex = decl -> id;
118 //printf("%s\n", addition ->
119     lex);
120 addition -> type = decl ->
121     type;
122 addition -> param = decl ->
123     param;
124 if (!addition -> param)
125 {
126     outputWidth(addition -> lex,
127         offset);
128     offset += get_width(decl);
129 }
130 addition -> left = NULL;
131 addition -> right = NULL;
132 addition -> parent =
133     bottom_node;
134 bottom_node -> left = addition
135     ;
136 bottom_node = addition;
137 //printf("(%s, %s)\n",
138     bottom_node -> lex,
139     bottom_node -> parent ->
140     lex);
141 return true;
142 }
143 tree_node*
144     start_param_matching(Token*
145     id) {
146 tree_node* current_node =
147     bottom_node;
148 while (current_node != NULL)
149 {
150     if (current_node -> type ==
151         PROC && current_node -> lex
152         == id -> id)
153     {
154         return current_node;
155         current_node = current_node ->
156             parent;
157     }
158     return NULL;
159 }
160 bool check_add_node(Token*
161     decl) {
162     char* errorMessage ;
163     switch (decl -> type) {
164     case PGNAME:
165     case PROC: if (!
166         check_add_green_node(decl))
167     {
168         errorMessage = calloc(100,
169             sizeof(*errorMessage));
170         sprintf(errorMessage,
171             "A program or procedure named
172             '%.*s' is already defined
173             in this scope!",
174             decl -> length, &BUFFER[decl
175                 -> start]);
176         throw_sem_error(errorMessage);
177         return false;
178     }
179     return true;
180     default: if (!
181         check_add_blue_node(decl))
182     {
183         errorMessage = calloc(100,
184             sizeof(*errorMessage));
185         sprintf(errorMessage,
186             "A variable named '%.*s' is
187             already defined in the
188             local scope!",
189             decl -> length, &BUFFER[decl
190                 -> start]);
191         throw_sem_error(errorMessage);
192         return false;
193     }
194     return true;
195 }

```

<pre> 175 176 void reached_end_of_scope() { 177     bottom_node = (*(tree_node**) 178         pop(green_node_stack)); 179 } 180 181 182 LangType get_type(Token* id) { 183     char* errorMessage; 184     if (id == NULL) 185         return ERR; 186 187     tree_node* current_node = 188         bottom_node; 189     while (current_node != NULL) 190     { 191         if (current_node -&gt; lex == id 192             -&gt; id) 193             return current_node -&gt; type; 194 195         current_node = current_node -&gt; 196             parent; 197     } 198     return NULL; 199 } </pre>	<pre> 12 // Do a byte-by-byte copy of 13     the data 14 for (int i = 0; i &lt; size; i++) 15     *(char *) (addition -&gt; data + 16         i) = *(char *) (data + i); 17 list -&gt; size++; 18 19 list -&gt; head = addition; 20 21 return list -&gt; size; 22 } 23 24 // For use as a queue; slow, 25     do not use 26 int addLast(LinkedList* list, 27     void *data, int size) 28 { 29     struct node* addition = malloc 30         (sizeof(*addition)); 31     addition -&gt; data = malloc(size 32         ); 33     addition -&gt; next = NULL; 34 35     for (int i = 0; i &lt; size; i++) 36     *(char *) (addition -&gt; data + 37         i) = *(char *) (data + i); 38 39     struct node* current = list -&gt; 40         head; 41 42     if (list -&gt; size == 0) 43         list -&gt; head = addition; 44     else { 45         while (current -&gt; next != NULL 46             ) 47             current = current -&gt; next; 48 49         current -&gt; next = addition; 50     } 51     list -&gt; size++; 52 } 53 54 void* pop(LinkedList* list) 55 { 56     struct node* head = list -&gt; </pre>
---	---

---

Listing 26: linkedlist.c

<pre> 1 #include&lt;stdlib.h&gt; 2 #include&lt;stdio.h&gt; 3 4 #include "linkedlist.h" 5 6 7 int add(LinkedList* list, void 8     *data, int size) 9 { 10     struct node* addition = malloc 11         (sizeof(*addition)); 12     addition -&gt; data = malloc(size 13         ); 14     addition -&gt; next = (list -&gt; 15         head); </pre>	<pre> 57     head = list -&gt; </pre>
---	---------------------------------------

<pre>         head; 49 struct node* next = head -&gt;         next; 50 51 void* data = head -&gt; data; 52 list -&gt; head = next; 53 list -&gt; size--; 54 55 //free(head); // TODO this is     necessary; should fix 56 return data; 57 } </pre>	<pre> 23 24 25 int initializeErrorHandler() 26 { 27     errorList = malloc(sizeof(*         errorList)); 28     semErrs = malloc(sizeof(*         semErrs)); 29     return errorList != NULL &amp;&amp;         semErrs != NULL; 30 } 31 32 void throw_syn_error(Token*     received, const Token**         expected, int exp_size) 33 { 34     // Generate token 35     Token* errToken = malloc(         sizeof(*errToken)); 36     errToken -&gt; attribute = SYNERR         ; 37     errToken -&gt; aspect = 0; 38     errToken -&gt; start = received         -&gt; start; 39     errToken -&gt; length = received         -&gt; length; 40 41     add(errorList, errToken,         sizeof(*errToken)); 42 43     // Generate error message 44     // Calculate space needed 45     int size = strlen("Found '';         expected "); 46     size += strlen(getLexFromToken         (received, true)); 47     for (int i = exp_size - 1; i         &gt;= 0; i--) { 48         size += strlen("''"); 49         size += strlen(getLexFromToken         (expected[i], expected[i]             -&gt; start)); 50     if (i &gt; 0) 51         size += strlen(", "); 52     } 53     size += strlen(" instead."); </pre>
--	---

---

Listing 27: errorHandler.c

<pre> 1 #include&lt;string.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 5 #include "errorHandler.h" 6 7 static LinkedList* errorList; 8 9 const char* lexErrs[] = {"     Unrecognized symbol:", 10 "ID length exceeded 10     characters:", 11 "Int length exceeded 10     characters:", 12 "Integer part of real exceeded     5 characters:", 13 "Fractional part of real     exceeded 5 characters:", 14 "Exponent part of long real     exceeded 2 characters:", 15 "Missing exponent part of long     real:", 16 "Leading 0 in int:", 17 "Excessive leading 0 in real:"     , 18 "Trailing 0 in real:", 19 "Leading 0 in exponent:", 20 "Attempt to use real exponent:     "}; 21 char* synErr; 22 LinkedList* semErrs; </pre>	<pre> 33 { 34     // Generate token 35     Token* errToken = malloc(         sizeof(*errToken)); 36     errToken -&gt; attribute = SYNERR         ; 37     errToken -&gt; aspect = 0; 38     errToken -&gt; start = received         -&gt; start; 39     errToken -&gt; length = received         -&gt; length; 40 41     add(errorList, errToken,         sizeof(*errToken)); 42 43     // Generate error message 44     // Calculate space needed 45     int size = strlen("Found '';         expected "); 46     size += strlen(getLexFromToken         (received, true)); 47     for (int i = exp_size - 1; i         &gt;= 0; i--) { 48         size += strlen("''"); 49         size += strlen(getLexFromToken         (expected[i], expected[i]             -&gt; start)); 50     if (i &gt; 0) 51         size += strlen(", "); 52     } 53     size += strlen(" instead."); </pre>
--	---



```

54 size += 1; // Null terminator ;
55
56 synErr = malloc(sizeof(*synErr
57 ) * size);
58 synErr[size - 1] = '\0';
59 strcpy(synErr, "Found ");
60 int current = 7;
61 int len = strlen(
62     getLexFromToken(received,
63 true));
64 strcpy(&synErr[current],
65     getLexFromToken(received,
66 true));
67 current += len;
68 strcpy(&synErr[current], "");
69 expected ";
70 current += 12;
71 for (int i = exp_size - 1; i
72     >= 0; i--) {
73     strcpy(&synErr[current], "");
74     current += 1;
75     len = strlen(getLexFromToken(
76     expected[i], expected[i] ->
77 start));
78     strcpy(&synErr[current],
79     getLexFromToken(expected[i
80 ], expected[i] -> start));
81     current += len;
82     strcpy(&synErr[current], "");
83     current += 1;
84     if (i > 0) {
85         strcpy(&synErr[current], ", ")
86         ;
87         current += 2;
88     }
89     strcpy(&synErr[current], "
90     instead.");
91 }
92
93 void throw_lex_error(enum
94 TokenType attribute, int
95 aspect, int start, int
96 length)
97 {
98     Token* errToken = malloc(
99     sizeof(*errToken));
100     errToken -> attribute =
101     attribute;
102     errToken -> aspect = aspect;
103     errToken -> start = start;
104     errToken -> length = length;
105     add(errorList, errToken,
106     sizeof(*errToken));
107 }
108
109 Token* getNextErrorToken()
110 {
111     if (errorList -> size > 0)
112     return (Token *) pop(errorList
113 );
114
115     return NULL;
116 }
117
118 void throw_sem_error(char* msg
119 ) {
120     // Generate error token
121     Token* errToken = malloc(
122     sizeof(*errToken));
123     errToken -> attribute = SEMERR
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Listing 28: globals.c

<pre> 6 7 char* BUFFER; 8 int LINE = 0; 9 int START = 0; 10 11 int initializeGlobals() 12 { 13     BUFFER = malloc(sizeof(char*) 14         *73); 15     return (BUFFER != NULL); 16 } 17 void updateLine(char* line) 18 { 19     START = 0; 20     LINE++; 21     strcpy(BUFFER, line); 22 } </pre>	<pre>         ListingLineSpace = 7; 19 static const int         ListingErrSpace = 50; 20 static const int         ListingLexSpace = 20; 21 22 static const int MemNameSpace         = 10; 23 static const int MemValSpace =         20; 24 25 void writeEOFToken() 26 { 27     fprintf(tokenFile, "%*d%*.s%*         d%*d\n", TokenLineSpace,         LINE, TokenLexSpace, 28     3, "EOF", TokenTypeSpace,         FILEEND, TokenAttrSpace, 0)         ; 29 } 30 31 int initializeHandler(const         char* sourcePath, const         char* resPath, 32 const char* listingPath, const         char* tokenPath, 33 const char* memPath) 34 { 35     if ((sourceFile = fopen(         sourcePath, "r")) == NULL) 36     { 37         fprintf(stderr, "%s\n", "         Source was null?"); 38         return 0; 39     } 40 41     FILE* resFile = fopen(resPath,         "r"); 42     initializeTokens(resFile); 43     fclose(resFile); 44 45     if ((listingFile = fopen(         listingPath, "w+")) == NULL            46     (tokenFile = fopen(tokenPath,         "w+")) == NULL    </pre>
--	--

---

Listing 29: handler.c

29 }  
30

<pre> 1 #include&lt;stdio.h&gt; 2 3 #include "handler.h" 4 #include "../globals/globals.h 5     " 6 #include "../tokenizer/ 7     tokenizer.h" 8 #include "../errorHandler/ 9     errorHandler.h" 10 11 static FILE* listingFile; 12 static FILE* tokenFile; 13 static FILE* sourceFile; 14 static FILE* memFile; 15 16 static const int 17     TokenLineSpace = 10; 18 static const int 19     TokenTypeSpace = 20; 20 static const int 21     TokenAttrSpace = 20; 22 static const intTokenLexSpace 23     = 20; 24 25 static const int </pre>	<pre> 31 int initializeHandler(const         char* sourcePath, const         char* resPath, 32 const char* listingPath, const         char* tokenPath, 33 const char* memPath) 34 { 35     if ((sourceFile = fopen(         sourcePath, "r")) == NULL) 36     { 37         fprintf(stderr, "%s\n", "         Source was null?"); 38         return 0; 39     } 40 41     FILE* resFile = fopen(resPath,         "r"); 42     initializeTokens(resFile); 43     fclose(resFile); 44 45     if ((listingFile = fopen(         listingPath, "w+")) == NULL            46     (tokenFile = fopen(tokenPath,         "w+")) == NULL    </pre>
--	--

```

47 (memFile = fopen(memPath, "w+"
48 )) == NULL)
49 return 0;
50 for (size_t i = FILEEND; i <=
51 SEMERR; i++) {
52     fprintf(tokenFile, "%-5zu%s\n"
53         , i, catNames[i]);
54 }
55 char line[72];
56 if (fgets(line, sizeof(line),
57     sourceFile) != NULL)
58 {
59     updateLine(line);
60     fprintf(listingFile, "%d\t%s"
61         , ListingLineSpace, LINE,
62         line);
63 } else {
64     writeEOFToken();
65 }
66 fprintf(tokenFile, "%s%s%s*s
67 %s\n", TokenLineSpace, "
68 Line",
69 TokenLexSpace, "Lexeme",
70 TokenAttrSpace, "Token
71 Attribute",
72 TokenTypeSpace, "Token Type");
73
74 fprintf(memFile, "%s%s*s\n",
75     MemNameSpace, "ID",
76     MemValSpace, "Memory Offset");
77 return 1;
78 }
79
80 void outputWidth(char* lex,
81     int width) {
82     fprintf(memFile, "%s*d\n",
83         MemNameSpace, lex,
84         MemValSpace, width);
85 }
86
87 void writeError(Token*
88     description)
89 {
90     fprintf(tokenFile, "%d*.*s*
91     d*d\n", TokenLineSpace,
92     LINE,
93     TokenLexSpace, description ->
94     length, &BUFFER[description
95     -> start],
96     TokenTypeSpace, description ->
97     attribute, TokenAttrSpace,
98     description -> aspect);
99     if (description -> attribute
100     == LEXERR)
101     fprintf(listingFile, "%s:%s
102     %*.s*\n", ListingLineSpace
103     - 1,
104     catNames[description ->
105     attribute], ListingErrSpace
106     ,
107     lexErrs[description -> aspect
108     ], ListingLexSpace,
109     description -> length, &BUFFER
110     [description -> start]);
111     else if (description ->
112     attribute == SYNERR)
113     fprintf(listingFile, "%s: %s\
114     n", ListingLineSpace - 1,
115     catNames[description ->
116     attribute], synErr);
117     else if (description ->
118     attribute == SEMERR)
119     fprintf(listingFile, "%s: %s\
120     n", ListingLineSpace - 1,
121     catNames[description ->
122     attribute], *(char**) pop(
123     semErrs));
124 }
125
126 void writeToken(Token* token)
127 {
128     // Don't bother including in
129     the output file.
130     if (token -> attribute == WS
131     || token -> attribute ==
132     NOOP)
133     return;
134 }

```

103	if (token -> attribute >=		<i>file (assume the latter)</i>
	LEXERR)	136	LINE++;
104	{	137	writeEOFToken();
105	writeError(token);	138	return false; // <i>Terminate</i>
106	return;	139	}
107	}	140	}
108		141	return true; // <i>Continue</i>
109		142	}
110	fprintf(tokenFile, "%*d%*.s%* d", TokenLineSpace, LINE, TokenLexSpace,	<hr/>	
111	token -> length, &BUFFER[token -> start], TokenTypeSpace,	Listing 30: parser.c	
112	token -> attribute);	1	#include<stdlib.h>
113	switch (token -> attribute) {	2	#include<stdbool.h>
114	case ID:	3	
115	fprintf(tokenFile, "%*p",	4	#include "../tokenizer/tokens. h"
	TokenAttrSpace, token -> id	5	#include "productions/ productions.h"
	);	6	#include "../tokenizer/ tokenizer.h"
116	break;	7	#include "../handler/handler.h "
117		8	#include "../errorHandler/ errorHandler.h"
118	default:	9	
119	fprintf(tokenFile, "%*d",	10	Token* current_tok = NULL;
	TokenAttrSpace, token ->	11	
	aspect);	12	static bool sequence_running = true;
120	break;	13	
121	}	14	Token* get_next_relevant_token ( )
122	fprintf(tokenFile, "\n");	15	{
123	}	16	const Token* next = malloc( sizeof(*next));
124		17	if (sequence_running)
125	bool handleToken(Token* token)	18	{
126	{	19	do {
127	writeToken(token);	20	next = getNextToken();
128	if (token -> attribute == WS	21	if (!handleToken(next))
	&& token -> aspect == 1) //	22	{
	<i>A newline</i>	23	sequence_running = false;
129	{	24	next = &eof_tok;
130	char line[72];	25	break;
131	if (fgets(line, sizeof(line),	26	}
	sourceFile) != NULL)	27	} while (next -> attribute ==
132	{		
133	updateLine(line);		
134	fprintf(listingFile, "%*d\t%s" , ListingLineSpace, LINE, line);		
135	} else { // <i>Error or end of</i>		

<pre> WS    next -&gt; attribute ==     NOOP 28    next -&gt; attribute &gt;= LEXERR     ); 29 } else { 30 next = &amp;eof_tok; 31 } 32 33 return next; 34 } 35 36 void require_sync(const Token*     sync_set[], int size, 37 const Token* first_set[], int     first_size) 38 { 39 throw_syn_error(current_tok,     first_set, first_size); 40 41 while (true) { 42 for (int i = 0; i &lt; size; i++) 43 if (tokens_equal(sync_set[i],     current_tok, sync_set[i] -&gt;     start)) 44 return; 45 46 current_tok =     get_next_relevant_token(); 47 } 48 } 49 50 <i>// Attempts to match the     source token with the     current token;</i> 51 <i>// if it is found, it returns     the matched token (for use     in the RDP).</i> 52 <i>// If it is not found, then     match returns null.</i> 53 Token* match(const Token*     source, bool strict) 54 { 55 if (tokens_equal(source,     current_tok, strict)) 56 { 57 Token* prev_tok = current_tok; </pre>	<pre> 58 current_tok =     get_next_relevant_token(); 59 return prev_tok; 60 } 61 else 62 { 63 throw_syn_error(current_tok, &amp;     source, 1); 64 current_tok =     get_next_relevant_token(); 65 return NULL; 66 } 67 } 68 69 bool generateParseTree() 70 { 71 current_tok = malloc(sizeof(*     current_tok)); 72 current_tok =     get_next_relevant_token(); 73 program(); 74 return match(&amp;eof_tok, false); 75 } </pre>
---	---

---

Listing 31: arguments.c

<pre> 54 55 56 57 </pre>	<pre> 1 #include&lt;stdbool.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 #include "productions.h" 5 #include "../parser.h" 6 #include "../tokenizer/     tokens.h" 7 8 static const Token* first_set     [] = {&amp;lparen_tok, &amp;     semic_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 10 11 static const Token* sync_set[]     = {&amp;eof_tok, &amp;semic_tok}; 12 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); </pre>
--------------------------	---

```

13 static void synch()
14 {
15     require_sync(sync_set,
16                 sync_size, first_set,
17                 first_size);
18 }
19 // Needs implementing: None
20 void arguments()
21 {
22     // Production 9.1
23     if (tokens_equal(&lparen_tok,
24                     current_tok, true))
25     {
26         match(&lparen_tok, true);
27         parameter_list();
28         match(&rparen_tok, true);
29         return;
30     }
31     // Production 9.2
32     } else if (tokens_equal(&
33                         semic_tok, current_tok,
34                         true))
35     return; // Epsilon
36 synch();
37 }

```

---

Listing 32: array<sub>a</sub>ccess.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
7     tokens.h"
8 static const Token* first_set
9     [] = {&assignop_tok, &
10         lbrac_tok};
11 static const int first_size =
12     sizeof(first_set)/sizeof(
13         first_set[0]);

```

```

11 static const Token* sync_set[]
12     = {&eof_tok, &assignop_tok
13         };
14 static const int sync_size =
15     sizeof(sync_set)/sizeof(
16         sync_set[0]);
17
18 static void synch()
19 {
20     require_sync(sync_set,
21                 sync_size, first_set,
22                 first_size);
23 }
24
25 static LangType array_compare(
26     LangType a_vals, LangType
27     e_type) {
28     if ((a_vals == INT || a_vals
29         == REAL) && e_type == INT)
30     return a_vals;
31     if (a_vals != ERR)
32     {
33         char* errorMessage = calloc
34             (100, sizeof(*errorMessage)
35             );
36         sprintf(errorMessage, "Attempt
37             to index variable of type
38             %s!", typeNames[a_vals]);
39         throw_sem_error(errorMessage);
40     }
41
42     return ERR;
43 }
44
45 // Needs implementing: None
46 LangType array_access(LangType
47     id_type)
48 {
49     // Production 17.1
50     if (tokens_equal(&lbrac_tok,
51                     current_tok, true))
52     {
53         match(&lbrac_tok, true);
54         LangType e_type = expression()
55             ;
56         match(&rbrac_tok, true);

```

```

41 LangType n_type =
    convert_from_array(id_type)
    ;
42 return array_compare(n_type,
    e_type);
43
44 // Production 17.2
45 } else if (tokens_equal(&
    assignop_tok, current_tok,
    true))
46 return id_type; // epsilon
47
48 synch();
49 return ERR;
50 }

```

---

Listing 33: compound<sub>s</sub>tatement.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&begin_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &semic_tok, &
    period_tok,
12 &end_tok, &else_tok};
13 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
14
15 static void synch()
16 {
17 require_sync(sync_set,
    sync_size, first_set,
    first_size);
18 }

```

```

19
20 // Needs implementing: None
21 void compound_statement()
22 {
23 // Production 11
24 if (tokens_equal(&begin_tok,
    current_tok, true))
25 {
26 match(&begin_tok, true);
27 optional_statements();
28 match(&end_tok, true);
29 return;
30 }
31 }
32
33 synch();
34 }

```

---

Listing 34: declarations.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&var_tok, &
    procedure_tok, &begin_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &
    procedure_tok, &begin_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16 require_sync(sync_set,
    sync_size, first_set,

```

```

        first_size);
17 }
18
19 // Needs implementing: None
20 void declarations()
21 {
22     // Production 3.1
23     if (tokens_equal(&var_tok,
24                     current_tok, true))
25     {
26         match(&var_tok, true);
27         Token* id_ref = match(&id_tok,
28                             false);
29         match(&colon_tok, true);
30         if (id_ref != NULL) {
31             id_ref -> type = type(id_ref);
32             id_ref -> param = false;
33             check_add_node(id_ref);
34         } else {
35             type(NULL);
36         }
37         match(&semicolon_tok, true);
38         declarations();
39         return;
40     }
41     // Production 3.2
42     if (tokens_equal(&begin_tok, current_tok,
43                     true) || tokens_equal(&procedure_tok,
44                                         current_tok, true))
45         return; // epsilon
46 }

```

---

Listing 35: `else_tail.c`

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7

```

```

8 static const Token* first_set
    [] = {&else_tok, &semicolon_tok,
    &end_tok, &else_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &semicolon_tok, &
    end_tok, &else_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
17                 sync_size, first_set,
18                 first_size);
19 }
20
21 // Needs implementing: None
22 void else_tail()
23 {
24     // Production 15.1
25     if (tokens_equal(&else_tok,
26                     current_tok, true)) // else
27     {
28         match(&else_tok, true);
29         statement();
30         return;
31     }
32     // Production 15.2
33     if (tokens_equal(&end_tok, current_tok, true)
34         // end
35         || tokens_equal(&semicolon_tok,
36                         current_tok, true)) // ;
37         return; // epsilon
38
39     synch();
40 }

```

---

Listing 36: `expression.c`

---

```

1 #include<stdbool.h>

```



```

2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../tokenizer/
    tokens.h"
7
8  static const Token* first_set
    [] = {&id_tok, &num_tok, &
    lparen_tok, &not_tok,
9    &plus_tok, &minus_tok};
10 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
11
12 static const Token* sync_set[]
    = {&eof_tok, &semic_tok, &
    end_tok, &else_tok,
13 &do_tok, &then_tok, &rbrac_tok
    , &rparen_tok,
14 &comma_tok};
15 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
16
17 static void synch()
18 {
19     require_sync(sync_set,
        sync_size, first_set,
        first_size);
20 }
21
22 // Needs implementing: None
23 LangType expression()
24 {
25     // Production 21
26     if (tokens_equal(&lparen_tok,
        current_tok, true)
27     || tokens_equal(&addop_tok,
        current_tok, false)
28     || tokens_equal(&id_tok,
        current_tok, false)
29     || tokens_equal(&not_tok,
        current_tok, true)
30     || tokens_equal(&num_tok,
        current_tok, false))

```

```

31 {
32     LangType s_type =
        simple_expression();
33     return related_expression(
        s_type);
34 }
35
36 synch();
37 return ERR;
38 }

```

---

Listing 37: *expression\_list.c*

---

```

1  #include<stdbool.h>
2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../tokenizer/
    tokens.h"
7
8  static const Token* first_set
    [] = {&id_tok, &num_tok, &
    lparen_tok, &not_tok,
9    &plus_tok, &minus_tok};
10 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
11
12 static const Token* sync_set[]
    = {&eof_tok, &rparen_tok};
13 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
14
15 static void synch()
16 {
17     require_sync(sync_set,
        sync_size, first_set,
        first_size);
18 }
19
20 // Needs implementing: None
21 void expression_list(tree_node
    * to_match, bool
    should_error)

```

22	{	47	}
23	// Production 20.1	48	
24	if (tokens_equal(&lparen_tok,	49	synch();
	current_tok, true)	50	}
25	tokens_equal(&addop_tok,		
	current_tok, false) // + OR		
	-		
26	tokens_equal(&id_tok,	1	#include<stdbool.h>
	current_tok, false) // ID	2	#include<stdlib.h>
27	tokens_equal(&not_tok,	3	
	current_tok, true)	4	#include "productions.h"
28	tokens_equal(&num_tok,	5	#include "../parser.h"
	current_tok, false)) // num	6	#include "../tokenizer/
29	{		tokens.h"
30	char* errorMessage;	7	
31	if (to_match == NULL &&	8	static const Token* first_set
	should_error)		[] = {&comma_tok, &
32	{		rparen_tok};
33	errorMessage= calloc(100,	9	static const int first_size =
	sizeof(*errorMessage));		sizeof(first_set)/sizeof(
34	sprintf(errorMessage, "Attempt		first_set[0]);
	to pass extraneous	10	
	parameter!");	11	static const Token* sync_set[]
35	throw_sem_error(errorMessage);		= {&eof_tok, &rparen_tok};
36	}	12	static const int sync_size =
37	LangType e_type = expression()		sizeof(sync_set)/sizeof(
	;		sync_set[0]);
38	if (should_error && to_match	13	
	!= NULL && to_match ->	14	static void synch()
	param && e_type != ERR &&	15	{
	e_type != to_match -> type)	16	require_sync(sync_set,
	{		sync_size, first_set,
39	errorMessage= calloc(100,		first_size);
	sizeof(*errorMessage));	17	}
40	sprintf(errorMessage, "	18	
	Expected type %s, not %s!",	19	// Needs implementing: None
41	typeNameNames[to_match -> type],	20	void expression_list_tail(
	typeNameNames[e_type]);		tree_node* to_match, bool
42	throw_sem_error(errorMessage);		should_error)
43	}	21	{
44	expression_list_tail(to_match	22	char* errorMessage;
	== NULL    !to_match ->	23	// Production 20.2.1
	param ? NULL :	24	if (tokens_equal(&comma_tok,
45	to_match -> left, e_type !=		current_tok, true))
	ERR && should_error);	25	{
46	return;	26	match(&comma_tok, true);

Listing 38: *expression\_list\_tail.c*

<pre> 27 if (to_match == NULL &amp;&amp;     should_error) 28 { 29     errorMessage = calloc(100,         sizeof(*errorMessage)); 30     sprintf(errorMessage, "Attempt         to pass extraneous         parameters!"); 31     throw_sem_error(errorMessage); 32 } 33 LangType e_type = expression()     ; 34 if (should_error &amp;&amp; to_match     != NULL &amp;&amp; e_type != ERR &amp;&amp;     e_type != to_match -&gt; type     ) { 35     errorMessage = calloc(100,         sizeof(*errorMessage)); 36     sprintf(errorMessage, "         Expected type %s, not %s!", 37         typeName[to_match -&gt; type],         typeName[e_type]); 38     throw_sem_error(errorMessage); 39 } 40 expression_list_tail(to_match     == NULL    !to_match -&gt;     param ? NULL : 41 to_match -&gt; left, e_type !=     ERR &amp;&amp; should_error); 42 return; 43 44 // Production 20.2.2 45 } else if (tokens_equal(&amp;     rparen_tok, current_tok,     true)) 46 { 47     if (to_match != NULL &amp;&amp;         to_match -&gt; param &amp;&amp;         should_error) { 48         errorMessage = calloc(100,             sizeof(*errorMessage)); 49         sprintf(errorMessage, "             Expected %s, not the end of             the parameters!", 50             typeName[to_match -&gt; type]); 51         throw_sem_error(errorMessage); </pre>	<pre> 52 } 53 return; // epsilon 54 } 55 56 synch(); 57 } </pre> <hr/> <p style="text-align: center;">Listing 39: factor.c</p> <hr/> <pre> 1 #include&lt;stdbool.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 #include "productions.h" 5 #include "../parser.h" 6 #include "../tokenizer/     tokens.h" 7 8 static const Token* first_set     [] = {&amp;id_tok, &amp;num_tok, &amp;     lparen_tok, &amp;not_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 10 11 static const Token* sync_set[]     = {&amp;eof_tok, &amp;mulop_tok, &amp;     addop_tok, &amp;relop_tok, 12 &amp;semic_tok, &amp;end_tok, &amp;     else_tok, &amp;do_tok, 13 &amp;then_tok, &amp;rbrac_tok, &amp;     rparen_tok, 14 &amp;comma_tok}; 15 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); 16 17 static void synch() 18 { 19     require_sync(sync_set,         sync_size, first_set,         first_size); 20 } 21 22 // Needs implementing: 25.1.2 23 LangType factor() 24 { </pre>
--	--

<pre> 25 // Production 25.1.1 26 if (tokens_equal(&amp;id_tok,     current_tok, false)) { //     id 27 char* errorMessage; 28 Token* id_ref = match(&amp;id_tok,     false); // id 29 LangType id_type = get_type(     id_ref); 30 if (id_type == NULL) { 31 errorMessage = calloc(100,     sizeof(*errorMessage)); 32 sprintf(errorMessage, 33 "No variable '%.*s' is defined     in the local scope!", 34 id_ref -&gt; length, &amp;BUFFER[     id_ref -&gt; start]); 35 throw_sem_error(errorMessage); 36 id_type = ERR; 37 } 38 return factor_tail(id_type); 39 40 // Production 25.1.2 41 } else if (tokens_equal(&amp;     num_tok, current_tok, false     )) { // num 42 Token* num_type; 43 num_type = match(&amp;num_tok,     false); 44 return num_type -&gt; aspect == 0     ? INT : REAL; 45 46 // Production 25.1.3 47 } else if (tokens_equal(&amp;     lparen_tok, current_tok,     true)) { // ( 48 match(&amp;lparen_tok, true); 49 LangType e_type = expression()     ; 50 match(&amp;rparen_tok, true); // ) 51 return e_type; 52 53 // Production 25.1.4 54 } else if (tokens_equal(&amp;     not_tok, current_tok, true)     ) { // not </pre>	<pre> 55 match(&amp;not_tok, true); 56 LangType f_type = factor(); 57 return type_lookup(f_type, INT     , &amp;not_tok); 58 } 59 60 61 synch(); 62 return ERR; 63 } </pre> <hr/> <p style="text-align: center;">Listing 40: factor<sub>tail</sub>.c</p> <hr/> <pre> 1 #include&lt;stdbool.h&gt; 2 3 #include "productions.h" 4 #include "../parser.h" 5 #include "../../tokenizer/     tokens.h" 6 7 static const Token* first_set     [] = {&amp;lbrac_tok, &amp;     mulop_tok, &amp;addop_tok, &amp;     relop_tok, 8 &amp;semic_tok, &amp;end_tok, &amp;     else_tok, &amp;do_tok, 9 &amp;then_tok, &amp;rbrac_tok, &amp;     rparen_tok, 10 &amp;comma_tok}; 11 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 12 13 static const Token* sync_set[]     = {&amp;eof_tok, &amp;mulop_tok, &amp;     addop_tok, &amp;relop_tok, 14 &amp;semic_tok, &amp;end_tok, &amp;     else_tok, &amp;do_tok, 15 &amp;then_tok, &amp;rbrac_tok, &amp;     rparen_tok, 16 &amp;comma_tok}; 17 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); 18 19 static void synch() </pre>
---	--

<pre> 20 { 21     require_sync(sync_set, 22                 sync_size, first_set, 23                 first_size); 24 } 25 26 static LangType array_compare( 27     LangType a_vals, LangType 28     e_type) { 29     if ((a_vals == INT    a_vals 30         == REAL) &amp;&amp; e_type == INT) 31         return a_vals; 32     if (a_vals != ERR) 33     { 34         char* errorMessage = calloc 35             (100, sizeof(*errorMessage) 36             ); 37         sprintf(errorMessage, "Attempt 38             to index variable of type 39             %s!", typeNames[a_vals]); 40         throw_sem_error(errorMessage); 41     } 42     return ERR; 43 } 44 45 // Needs implementing: None 46 LangType factor_tail(id_type) 47 { 48     // Production 25.2.1 49     if (tokens_equal(&amp;lbrac_tok, 50                     current_tok, true)) { 51         match(&amp;lbrac_tok, true); 52         LangType e_type = expression() 53             ; 54         match(&amp;rbrac_tok, true); 55         LangType n_type = 56             convert_from_array(id_type) 57             ; 58         return array_compare(n_type, 59                             e_type); 60     } 61     // Production 25.2.2 62     } else if (tokens_equal(&amp; 63                             rparen_tok, current_tok, 64                             true) </pre>	<pre> 50        tokens_equal(&amp;comma_tok, 51                     current_tok, true) 52        tokens_equal(&amp;semic_tok, 53                     current_tok, true) 54        tokens_equal(&amp;rbrac_tok, 55                     current_tok, true) 56        tokens_equal(&amp;addop_tok, 57                     current_tok, false) 58        tokens_equal(&amp;do_tok, 59                     current_tok, true) 60        tokens_equal(&amp;else_tok, 61                     current_tok, true) 62        tokens_equal(&amp;end_tok, 63                     current_tok, true) 64        tokens_equal(&amp;mulop_tok, 65                     current_tok, false) 66        tokens_equal(&amp;relop_tok, 67                     current_tok, false) 68        tokens_equal(&amp;then_tok, 69                     current_tok, true)) 70     return id_type; // epsilon 71 72     synch(); 73     return ERR; 74 } </pre>
--	--

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
7     tokens.h"
8
9 static const Token* first_set
10     [] = {&id_tok};
11 static const int first_size =
12     sizeof(first_set)/sizeof(
13     first_set[0]);
14
15 static const Token* sync_set[]
16     = {&eof_tok, &rparen_tok};
17 static const int sync_size =
18     sizeof(sync_set)/sizeof(

```

Listing 41: *idlist.c*

```

        sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 void id_list()
21 {
22     Token* id_ref;
23     // Production 2.1
24     if (tokens_equal(&id_tok,
        current_tok, false)) {
25         id_ref = match(&id_tok, false)
        ;
26         if (id_ref != NULL) {
27             id_ref -> type = PPNAME;
28             id_ref -> param = true;
29             check_add_node(id_ref);
30         }
31         id_list_tail();
32         return;
33     }
34
35     synch();
36 }

```

---

Listing 42: *id\_list\_tail.c*

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&comma_tok, &
    rparen_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);

```

```

10
11 static const Token* sync_set[]
    = {&eof_tok, &rparen_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 void id_list_tail()
21 {
22     // Production 2.2.1
23     if (tokens_equal(&comma_tok,
        current_tok, true))
24     {
25         match(&comma_tok, true);
26         Token* id_ref;
27
28         id_ref = match(&id_tok, false)
        ;
29         if (id_ref != NULL) {
30             id_ref -> type = PPNAME;
31             id_ref -> param = true;
32             check_add_node(id_ref);
33         }
34
35         id_list_tail();
36         return;
37
38         // Production 2.2.2
39     } else if (tokens_equal(&
        rparen_tok, current_tok,
        true))
40         return; // Epsilon
41
42     synch();
43 }

```

---

Listing 43: *optional\_expressions.c*

---

```

1  #include<stdbool.h>
2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../../tokenizer/
    tokens.h"
7
8  static const Token* first_set
    [] = {&lparen_tok, &
    semic_tok, &end_tok,
9    &else_tok};
10 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
11
12 static const Token* sync_set[]
    = {&eof_tok, &semic_tok, &
    end_tok, &else_tok};
13 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
14
15 static void synch()
16 {
17     require_sync(sync_set,
        sync_size, first_set,
        first_size);
18 }
19
20 // Needs implementing: None
21 void optional_expressions(
    tree_node* to_match, bool
    should_error)
22 {
23     char* errorMessage;
24     // Production 19.1
25     if (tokens_equal(&lparen_tok,
        current_tok, true))
26     {
27         match(&lparen_tok, true);
28         expression_list(to_match,
            should_error);
29         match(&rparen_tok, true);
30         return;
31
32     // Production 19.2
33     } else if (tokens_equal(&
        semic_tok, current_tok,
        true)
34     || tokens_equal(&else_tok,
        current_tok, true)
35     || tokens_equal(&end_tok,
        current_tok, true))
36     {
37         if (to_match != NULL &&
            should_error) {
38             errorMessage= calloc(100,
                sizeof(*errorMessage));
39             sprintf(errorMessage, "
                Expected an argument of
                type %s!",
40                 typeNames[to_match -> type]);
41             throw_sem_error(errorMessage);
42         }
43
44         return; // epsilon
45     }
46
47     synch();
48 }

```

---

Listing 44: optional<sub>s</sub>tatements.c

```

1  #include<stdbool.h>
2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../../tokenizer/
    tokens.h"
7
8  static const Token* first_set
    [] = {&id_tok, &call_tok, &
    begin_tok, &while_tok,
9    &if_tok, &end_tok, &array_tok
    };
10 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
11
12 static const Token* sync_set[]

```

```

        = {&eof_tok, &end_tok};
13 static const int sync_size =
    sizeof(sync_set)/sizeof(
        sync_set[0]);
14
15 static void synch()
16 {
17     require_sync(sync_set,
        sync_size, first_set,
        first_size);
18 }
19
20 // Needs implementing: None
21 void optional_statements()
22 {
23     // Production 12.1
24     if (tokens_equal(&begin_tok,
        current_tok, true) // begin
25     || tokens_equal(&call_tok,
        current_tok, true) // call
26     || tokens_equal(&id_tok,
        current_tok, false) // ID
27     || tokens_equal(&if_tok,
        current_tok, true) // if
28     || tokens_equal(&while_tok,
        current_tok, true)) //
        while
29 {
30     statement_list();
31     return;
32
33     // Production 12.2
34 } else if (tokens_equal(&
        end_tok, current_tok, true)
        ) // end
35 return; // epsilon
36
37 synch();
38 }

```

---

Listing 45: parameter\_list.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&id_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &rparen_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
        sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 void parameter_list()
21 {
22     // Production 10.1
23     if (tokens_equal(&id_tok,
        current_tok, false)) {
24         Token* id_ref;
25         id_ref = match(&id_tok, false)
            ; // ID
26         match(&colon_tok, true);
27         if (id_ref != NULL) {
28             id_ref -> param = true;
29             id_ref -> type = type(id_ref);
30             check_add_node(id_ref);
31         } else {
32             type(NULL);
33         }
34         parameter_list_tail();
35         return;
36     }
37
38     synch();
39 }

```



Listing 46: *parameter\_list\_tail.c*


---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&semic_tok, &
    rparen_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &rparen_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 void parameter_list_tail()
21 {
22     // Production 10.2.1
23     if (tokens_equal(&semic_tok,
        current_tok, true)) // ;
24     {
25         match(&semic_tok, true); // ;
26         Token* id_ref = match(&id_tok,
            false); // ID
27         match(&colon_tok, true); // :
28         if (id_ref != NULL) {
29             id_ref -> param = true;
30             id_ref -> type = type(id_ref);
31             check_add_node(id_ref);
32         } else {
33             type(NULL);
34         }
35         parameter_list_tail();
36         return;
37
38         // Production 10.2.2
39     } else if (tokens_equal(&
        rparen_tok, current_tok,
        true)) // )
40         return; // epsilon
41
42     synch();
43 }

```

---

Listing 47: *procedure\_statement.c*


---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&call_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &semic_tok, &
    end_tok, &else_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18

```

---

```

19 // Needs implementing: None
20 void procedure_statement()
21 {
22     char* errorMessage;
23     // Production 18
24     if (tokens_equal(&call_tok,
25                     current_tok, true)) // call
26     {
27         Token* id_ref;
28         match(&call_tok, true); // call
29         id_ref = match(&id_tok, false)
30         ;
31         if (id_ref != NULL) {
32             tree_node* addition =
33                 start_param_matching(id_ref
34                                     );
35             if (addition == NULL) {
36                 errorMessage= calloc(100,
37                                     sizeof(*errorMessage));
38                 sprintf(errorMessage, "
39                     Procedure '%s' not in scope
40                     !", id_ref -> id);
41                 throw_sem_error(errorMessage);
42             }
43             optional_expressions(NULL,
44                                 false);
45         } else
46             optional_expressions(addition
47                                 -> left == NULL ? NULL :
48                                 addition -> left -> param
49                                 ? addition -> left : NULL,
50                                 true);
51         } else {
52             optional_expressions(NULL,
53                                 false);
54         }
55     }
56     return;
57 }
58 synch();
59 }

```

---

Listing 48: program.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3 #include<stdio.h>
4
5 #include "productions.h"
6 #include "../parser.h"
7 #include "../tokenizer/
8     tokens.h"
9
10 static const Token* first_set
11     [] = {&program_tok};
12 static const int first_size =
13     sizeof(first_set)/sizeof(
14         first_set[0]);
15
16 static const Token* sync_set[]
17     = {&eof_tok};
18 static const int sync_size =
19     sizeof(sync_set)/sizeof(
20         sync_set[0]);
21
22 static void synch()
23 {
24     require_sync(sync_set,
25                 sync_size, first_set,
26                 first_size);
27 }
28
29 // Needs implementing: None
30 void program()
31 {
32     Token* id_ref;
33     // Production 1
34     if (tokens_equal(&program_tok,
35                     current_tok, true)) {
36         match(&program_tok, true); //
37             program
38         id_ref = match(&id_tok, false)
39         ; // id
40         if (id_ref != NULL) {
41             id_ref -> type = PGNAME;
42             id_ref -> param = false;
43             check_add_node(id_ref);
44         }
45         match(&lparen_tok, true); // (
46         id_list();
47         match(&rparen_tok, true); // )

```

```

36 match(&semic_tok, true); // ;
37 declarations();
38 subprogram_declarations();
39 compound_statement();
40 match(&period_tok, true); // .
41 return;
42 }
43
44 synch();
45 }

```

---

Listing 49: *related\_expression.c*

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
   tokens.h"
7
8 static const Token* first_set
   [] = {&relop_tok,
9   &semic_tok, &end_tok, &
   else_tok, &do_tok,
10  &then_tok, &rbrac_tok, &
   rparen_tok,
11  &comma_tok};
12 static const int first_size =
   sizeof(first_set)/sizeof(
   first_set[0]);
13
14 static const Token* sync_set[]
   = {&eof_tok, &semic_tok, &
   end_tok,
15  &else_tok, &do_tok, &then_tok,
   &rbrac_tok,
16  &rparen_tok, &comma_tok};
17 static const int sync_size =
   sizeof(sync_set)/sizeof(
   sync_set[0]);
18
19 static void synch()
20 {
21   require_sync(sync_set,
   sync_size, first_set,

```

```

   first_size);
22 }
23
24 // Needs implementing: None
25 LangType related_expression(
   LangType s_type)
26 {
27   // Production 22.1
28   if (tokens_equal(&relop_tok,
   current_tok, false)) {
29     Token* relop_op;
30     relop_op = match(&relop_tok,
   false);
31     LangType s1_type =
   simple_expression();
32     return type_lookup(s_type,
   s1_type, relop_op);
33
34   // Production 22.2
35   } else if (tokens_equal(&
   rparen_tok, current_tok,
   true)
36   || tokens_equal(&comma_tok,
   current_tok, true)
37   || tokens_equal(&semic_tok,
   current_tok, true)
38   || tokens_equal(&rbrac_tok,
   current_tok, true)
39   || tokens_equal(&do_tok,
   current_tok, true)
40   || tokens_equal(&else_tok,
   current_tok, true)
41   || tokens_equal(&end_tok,
   current_tok, true)
42   || tokens_equal(&then_tok,
   current_tok, true))
43     return s_type; // epsilon
44
45   synch();
46   return ERR;
47 }

```

---

Listing 50: *sign.c*

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>

```

```

3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
  tokens.h"
7
8 static const Token* first_set
  [] = {&plus_tok, &minus_tok
  };
9 static const int first_size =
  sizeof(first_set)/sizeof(
  first_set[0]);
10
11 static const Token* sync_set[]
  = {&eof_tok, &id_tok, &
  num_tok,
12 &not_tok, &rparen_tok};
13 static const int sync_size =
  sizeof(sync_set)/sizeof(
  sync_set[0]);
14
15 static void synch()
16 {
17   require_sync(sync_set,
  sync_size, first_set,
  first_size);
18 }
19
20 // Needs implementing: None
21 void sign()
22 {
23   // Production 24.2.1
24   if (tokens_equal(&plus_tok,
  current_tok, true)) {
25     match(&plus_tok, true);
26     return;
27
28   // Production 24.2.2
29   } else if (tokens_equal(&
  minus_tok, current_tok,
  true)) {
30     match(&minus_tok, true);
31     return; // epsilon
32   }
33   synch();
34 }

```

---

Listing 51: simple<sub>e</sub>expression.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
  tokens.h"
7
8 static const Token* first_set
  [] = {&id_tok, &num_tok, &
  lparen_tok, &not_tok, &
  plus_tok, &minus_tok};
9
10 static const int first_size =
  sizeof(first_set)/sizeof(
  first_set[0]);
11
12 static const Token* sync_set[]
  = {&eof_tok, &relop_tok, &
  semic_tok,
13 &end_tok, &else_tok, &do_tok,
  &then_tok,
14 &rbrac_tok, &rparen_tok, &
  comma_tok};
15 static const int sync_size =
  sizeof(sync_set)/sizeof(
  sync_set[0]);
16
17 static void synch()
18 {
19   require_sync(sync_set,
  sync_size, first_set,
  first_size);
20 }
21
22 // Needs implementing: None
23 LangType simple_expression()
24 {
25   char* errorMessage;
26   // Production 23.1.1
27   if (tokens_equal(&lparen_tok,
  current_tok, true)
28   || tokens_equal(&id_tok,

```

<pre> current_tok, false) 29    tokens_equal(&amp;not_tok,     current_tok, true) 30    tokens_equal(&amp;num_tok,     current_tok, false)) 31 { 32   LangType t_type = term(); 33   return simple_expression_tail(     t_type); 34 35   // Production 23.1.2 36 } else if (tokens_equal(&amp;     plus_tok, current_tok, true     ) 37    tokens_equal(&amp;minus_tok,     current_tok, true)) { 38   sign(); 39   LangType t_type = term(); 40   if (t_type != INT &amp;&amp; t_type !=     REAL &amp;&amp; t_type != ERR) 41   { 42     errorMessage= calloc(100,     sizeof(*errorMessage)); 43     sprintf(errorMessage, "     Expected number for use     with sign, not %s!", 44     typeNames[t_type]); 45     throw_sem_error(errorMessage); 46   } 47   return simple_expression_tail(     t_type); 48 } 49 50 synch(); 51 return ERR; 52 } </pre>	<pre> 7 8 static const Token* first_set     [] = {&amp;addop_tok, &amp;     relop_tok, 9    &amp;semic_tok, &amp;end_tok, &amp;     else_tok, &amp;do_tok, 10   &amp;then_tok, &amp;rbrac_tok, &amp;     rparen_tok, 11   &amp;comma_tok}; 12 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 13 14 static const Token* sync_set[]     = {&amp;eof_tok, &amp;relop_tok, &amp;     semic_tok, 15   &amp;end_tok, &amp;else_tok, &amp;do_tok,     &amp;then_tok, 16   &amp;rbrac_tok, &amp;rparen_tok, &amp;     comma_tok}; 17 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); 18 19 static void synch() 20 { 21   require_sync(sync_set,     sync_size, first_set,     first_size); 22 } 23 24 // Needs implementing: None 25 LangType     simple_expression_tail(     LangType t_type) 26 { 27   // Production 23.2.1 28   if (tokens_equal(&amp;addop_tok,     current_tok, false)) { 29     Token* addop_op; 30     addop_op = match(&amp;addop_tok,     false); 31     LangType t_type2 = term(); 32     return simple_expression_tail(     type_lookup(t_type, t_type2     , addop_op)); </pre>
--	---

---

Listing 52: `simple_expression_tail.c`

<pre> 1 #include&lt;stdbool.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 #include "productions.h" 5 #include "../parser.h" 6 #include "../tokenizer/     tokens.h" </pre>	<pre> 29 Token* addop_op; 30 addop_op = match(&amp;addop_tok,     false); 31 LangType t_type2 = term(); 32 return simple_expression_tail(     type_lookup(t_type, t_type2     , addop_op)); </pre>
---	--

```

33                                     = {&eof_tok, &semic_tok, &
34                                     rparen_tok};
35 // Production 23.2.2
36 } else if (tokens_equal(&
    rparen_tok, current_tok,
    true)
37 || tokens_equal(&comma_tok,
    current_tok, true)
38 || tokens_equal(&semic_tok,
    current_tok, true)
39 || tokens_equal(&rbrac_tok,
    current_tok, true)
40 || tokens_equal(&do_tok,
    current_tok, true)
41 || tokens_equal(&else_tok,
    current_tok, true)
42 || tokens_equal(&end_tok,
    current_tok, true)
43 || tokens_equal(&relop_tok,
    current_tok, false)
44 || tokens_equal(&then_tok,
    current_tok, true))
45 return t_type; // epsilon
46
47 synch();
48 return ERR;
49 }

```

---

Listing 53: standard\_type.c

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7
8 static const Token* first_set
    [] = {&integer_tok, &
    real_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]

```

```

12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 LangType standard_type()
21 {
22     // Production 5.1
23     if (tokens_equal(&integer_tok,
        current_tok, true)) //
        integer
24     {
25         match(&integer_tok, true);
26         return INT;
27
28     // Production 5.2
29     } else if (tokens_equal(&
        real_tok, current_tok, true
        )) { // real
30         match(&real_tok, true);
31         return REAL;
32     }
33
34     synch();
35     return ERR;
36 }

```

---

Listing 54: statement.c

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7

```

```

8 static const Token* first_set
   [] = {&id_tok, &call_tok, &
9   &if_tok};
10 static const int first_size =
   sizeof(first_set)/sizeof(
   first_set[0]);
11
12 static const Token* sync_set[]
   = {&eof_tok, &semic_tok, &
   end_tok, &else_tok};
13 static const int sync_size =
   sizeof(sync_set)/sizeof(
   sync_set[0]);
14
15 static void synch()
16 {
17   require_sync(sync_set,
   sync_size, first_set,
   first_size);
18 }
19
20 // Needs implementing: None
21 void statement()
22 {
23   char* errorMessage;
24   // Production 14.1
25   if (tokens_equal(&id_tok,
   current_tok, false)) { //
   id
26     Token* id_ref = current_tok;
27     LangType v_type = variable();
28     match(&assignop_tok, true);
29
30     if (get_type(id_ref) == ERR)
31       // The only way for this to
   error is an undeclared
   variable
32     {
33       errorMessage = calloc(100,
   sizeof(*errorMessage));
34       sprintf(errorMessage, "ID '%s'
   not in scope!",
35       id_ref -> id);
36       throw_sem_error(errorMessage);
37       expression();
38     } else if (v_type != ERR &&
   v_type != INT && v_type !=
   REAL)
39     {
40       errorMessage = calloc(100,
   sizeof(*errorMessage));
41       sprintf(errorMessage, "Cannot
   assign to ID '%s' of type
   '%s'!",
42       id_ref -> id, typeNames[v_type
   ]);
43       throw_sem_error(errorMessage);
44       expression();
45     } else {
46       LangType e_type = expression()
   ;
47       type_lookup(v_type, e_type, &
   assignop_tok);
48     }
49     return;
50
51     // Production 14.2
52     } else if (tokens_equal(&
   call_tok, current_tok, true
   )) { // call
53       procedure_statement();
54       return;
55
56       // Production 14.3
57       } else if (tokens_equal(&
   begin_tok, current_tok,
   true)) { // begin
58       compound_statement();
59       return;
60
61       // Production 14.4
62       } else if (tokens_equal(&
   while_tok, current_tok,
   true)) { // while
63       match(&while_tok, true); //
   while
64       LangType e_type = expression()
   ;
65       if (e_type != BOOL && e_type
   != ERR)
66       {

```

```

67 errorMessage= calloc(100,
    sizeof(*errorMessage));
68 sprintf(errorMessage, "
    Expression in while must be
    boolean, not %s!",
69 typeNames[e_type]);
70 throw_sem_error(errorMessage);
71 }
72 match(&do_tok, true);
73 statement();
74 return;
75
76 // Production 14.5
77 } else if (tokens_equal(&
    if_tok, current_tok, true))
    { // if
78 match(&if_tok, true); // if
79 LangType e_type = expression()
    ;
80 if (e_type != BOOL && e_type
    != ERR)
81 {
82 errorMessage= calloc(100,
    sizeof(*errorMessage));
83 sprintf(errorMessage, "If
    clause must be a boolean
    expression, not %s!",
84 typeNames[e_type]);
85 throw_sem_error(errorMessage);
86 }
87 match(&then_tok, true); //
    then
88 statement();
89 else_tail();
90 return;
91 }
92
93
94 synch();
95 }

```

---

Listing 55: *statement\_list.c*

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
    tokens.h"
7 static const Token* first_set
    [] = {&id_tok, &call_tok, &
    begin_tok, &while_tok, &
8     &if_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &end_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
    sync_set[0]);
13
14 static void synch()
15 {
16 require_sync(sync_set,
    sync_size, first_set,
    first_size);
17 }
18
19 // Needs implementing: None
20 void statement_list()
21 {
22 // Production 13.1
23 if (tokens_equal(&begin_tok,
    current_tok, true)
24 || tokens_equal(&call_tok,
    current_tok, true)
25 || tokens_equal(&id_tok,
    current_tok, false)
26 || tokens_equal(&if_tok,
    current_tok, true)
27 || tokens_equal(&while_tok,
    current_tok, true))
28 {
29 statement();
30 statement_list_tail();
31 return;
32 }
33
34 synch();

```



```

35 }

```

---

Listing 56: `statement_list_tail.c`

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
   tokens.h"
7
8 static const Token* first_set
   [] = {&semic_tok, &end_tok
   };
9 static const int first_size =
   sizeof(first_set)/sizeof(
   first_set[0]);
10
11 static const Token* sync_set[]
   = {&eof_tok, &end_tok};
12 static const int sync_size =
   sizeof(sync_set)/sizeof(
   sync_set[0]);
13
14 static void synch()
15 {
16 require_sync(sync_set,
   sync_size, first_set,
   first_size);
17 }
18
19 // Needs implementing: None
20 void statement_list_tail()
21 {
22 // Production 13.2.1
23 if (tokens_equal(&semic_tok,
   current_tok, true))
24 {
25 match(&semic_tok, true);
26 statement();
27 statement_list_tail();
28 return;
29
30
31 // Production 13.2.2

```

```

32 } else if (tokens_equal(&
   end_tok, current_tok, true)
   ) // end
33 return; // epsilon
34
35 synch();
36 }

```

---

Listing 57: `subprogram_declaration.c`

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
   tokens.h"
7
8 static const Token* first_set
   [] = {&procedure_tok};
9 static const int first_size =
   sizeof(first_set)/sizeof(
   first_set[0]);
10
11 static const Token* sync_set[]
   = {&procedure_tok};
12 static const int sync_size =
   sizeof(sync_set)/sizeof(
   sync_set[0]);
13
14 static void synch()
15 {
16 require_sync(sync_set,
   sync_size, first_set,
   first_size);
17 }
18
19 // Needs implementing: None
20 void subprogram_declaration()
21 {
22 // Production 7
23 if (tokens_equal(&
   procedure_tok, current_tok,
   true)) // procedure
24 {
25 bool declared =

```

<pre> subprogram_head(); 26 declarations(); 27 subprogram_declarations(); 28 compound_statement(); 29 30 if (declared) 31 reached_end_of_scope(); // pop     from stack 32 return; 33 } 34 35 synch(); 36 } </pre> <hr/> <p>Listing 58: subprogram<sub>d</sub>clarations.c</p> <pre> 1 #include&lt;stdbool.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 #include "productions.h" 5 #include "../parser.h" 6 #include "../tokenizer/     tokens.h" 7 8 static const Token* first_set     [] = {&amp;procedure_tok, &amp;     begin_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 10 11 static const Token* sync_set[]     = {&amp;eof_tok, &amp;begin_tok}; 12 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); 13 14 static void synch() 15 { 16 require_sync(sync_set,     sync_size, first_set,     first_size); 17 } 18 19 // Needs implementing: None </pre>	<pre> 20 void subprogram_declarations() 21 { 22 // Production 6.1 23 if (tokens_equal(&amp;     procedure_tok, current_tok,     true)) // procedure 24 { 25 subprogram_declaration(); 26 match(&amp;semic_tok, true); // ; 27 subprogram_declarations(); 28 return; 29 30 // Production 6.2 31 } else if (tokens_equal(&amp;     begin_tok, current_tok,     true)) // begin 32 return; // Epsilon 33 34 synch(); 35 } </pre> <hr/> <p>Listing 59: subprogram<sub>h</sub>ead.c</p> <pre> 1 #include&lt;stdbool.h&gt; 2 #include&lt;stdlib.h&gt; 3 4 #include "productions.h" 5 #include "../parser.h" 6 #include "../tokenizer/     tokens.h" 7 8 static const Token* first_set     [] = {&amp;procedure_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof(     first_set[0]); 10 11 static const Token* sync_set[]     = {&amp;eof_tok, &amp;var_tok, &amp;     procedure_tok, 12 &amp;begin_tok}; 13 static const int sync_size =     sizeof(sync_set)/sizeof(     sync_set[0]); 14 15 static void synch() </pre>
--	--

```

16 {
17     require_sync(sync_set,
18         sync_size, first_set,
19         first_size);
20 }
21 // Needs implementing: None
22 bool subprogram_head()
23 {
24     bool result = false;
25     // Production 8
26     if (tokens_equal(&
27         procedure_tok, current_tok,
28         true)) // procedure
29     {
30         Token* id_ref;
31         match(&procedure_tok, true);
32         // procedure
33         id_ref = match(&id_tok, false)
34         ;
35         if (id_ref != NULL) {
36             id_ref -> type = PROC;
37             id_ref -> param = false;
38             result = check_add_node(id_ref
39                 );
40         }
41         arguments();
42         match(&semicolon_tok, true); // ;
43         return result;
44     }
45     synch();
46     return result;
47 }

```

---

Listing 60: term.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../tokenizer/
7     tokens.h"
8
9 static const Token* first_set

```

```

[] = {&id_tok, &num_tok, &
lparen_tok, &not_tok};
9 static const int first_size =
sizeof(first_set)/sizeof(
first_set[0]);
10
11 static const Token* sync_set[]
= {&eof_tok, &addop_tok, &
relop_tok, &semicolon_tok,
12 &end_tok, &else_tok, &do_tok,
&then_tok,
13 &rbrack_tok, &rparen_tok, &
comma_tok};
14 static const int sync_size =
sizeof(sync_set)/sizeof(
sync_set[0]);
15
16 static void synch()
17 {
18     require_sync(sync_set,
19         sync_size, first_set,
20         first_size);
21 }
22 // Needs implementing: None
23 LangType term()
24 {
25     // Production 24.1
26     if (tokens_equal(&lparen_tok,
27         current_tok, true) // (
28     || tokens_equal(&id_tok,
29         current_tok, false) // ID
30     || tokens_equal(&not_tok,
31         current_tok, true) // not
32     || tokens_equal(&num_tok,
33         current_tok, false)) { //
34         num
35         LangType f_type = factor();
36         return term_tail(f_type);
37     }
38     synch();
39     return ERR;
40 }

```

---

Listing 61: term<sub>tail</sub>.c

---

```

1  #include<stdbool.h>
2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../tokenizer/
    tokens.h"
7
8  static const Token* first_set
    [] = {&mulop_tok, &
        addop_tok, &relop_tok,
9      &semic_tok, &end_tok, &
        else_tok, &do_tok,
10     &then_tok, &rbrac_tok, &
        rparen_tok,
11     &comma_tok};
12 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
13
14 static const Token* sync_set[]
    = {&eof_tok, &addop_tok, &
        relop_tok, &semic_tok,
15     &end_tok, &else_tok, &do_tok,
        &then_tok,
16     &rbrac_tok, &rparen_tok, &
        comma_tok};
17 static const int sync_size =
    sizeof(sync_set)/sizeof(
        sync_set[0]);
18
19 static void synch()
20 {
21     require_sync(sync_set,
        sync_size, first_set,
        first_size);
22 }
23
24 // Needs implementing: None
25 LangType term_tail(LangType
    f_type)
26 {
27     // Production 24.2.1
28     if (tokens_equal(&mulop_tok,
        current_tok, false)) { //
        MULOP
29     Token* mulop_op = match(&
        mulop_tok, false);
30     LangType f2_type = factor();
31     return term_tail(type_lookup(
        f_type, f2_type, mulop_op))
        ;
32
33     // Production 24.2.2
34     } else if (tokens_equal(&
        rparen_tok, current_tok,
        true)
35     || tokens_equal(&comma_tok,
        current_tok, true)
36     || tokens_equal(&semic_tok,
        current_tok, true)
37     || tokens_equal(&rbrac_tok,
        current_tok, true)
38     || tokens_equal(&addop_tok,
        current_tok, false)
39     || tokens_equal(&do_tok,
        current_tok, true)
40     || tokens_equal(&else_tok,
        current_tok, true)
41     || tokens_equal(&end_tok,
        current_tok, true)
42     || tokens_equal(&relop_tok,
        current_tok, false)
43     || tokens_equal(&then_tok,
        current_tok, true))
44     return f_type; // epsilon
45
46     synch();
47     return ERR;
48 }

```

---

Listing 62: type.c

---

```

1  #include<stdbool.h>
2  #include<stdlib.h>
3
4  #include "productions.h"
5  #include "../parser.h"
6  #include "../tokenizer/
    tokens.h"

```

```

7
8 static const Token* first_set
    [] = {&integer_tok, &
        real_tok, &array_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
10
11 static const Token* sync_set[]
    = {&array_tok, &
        integer_tok, &real_tok};
12 static const int sync_size =
    sizeof(sync_set)/sizeof(
        sync_set[0]);
13
14 static void synch()
15 {
16     require_sync(sync_set,
        sync_size, first_set,
        first_size);
17 }
18
19 // Needs implementing: None
20 LangType type(Token* id_ref)
21 {
22     // Production 4.2
23     if (tokens_equal(&array_tok,
        current_tok, true))
24     {
25         char* errorMessage;
26         Token* numI;
27         Token* numF;
28         match(&array_tok, true); //
            array
29         match(&lbrac_tok, true); // [
30         numI = match(&num_tok, false);
            // num
31         match(&dotdot_tok, true); //
            ..
32         numF = match(&num_tok, false);
            // num
33         match(&rbrac_tok, true); // ]
34         match(&of_tok, true); // of
35         if (numI != NULL && numF !=
            NULL && id_ref != NULL)
36         if (type_lookup(numI -> aspect
            == 0 ? INT : REAL, numF ->
            aspect == 0 ? INT : REAL,
            &dotdot_tok) != ERR) {
37         if (numI -> int_val >= numF ->
            int_val) {
38             errorMessage= calloc(100,
                sizeof(*errorMessage));
39             sprintf(errorMessage, "
                Expected array end index %d
                to be strictly greater
                than start %d", numF ->
                int_val, numI -> int_val);
40             throw_sem_error(errorMessage);
41         }
42         id_ref -> array_length = numF
            -> int_val - numI ->
            int_val + 1;
43     }
44     return convert_to_array(
        standard_type());
45
46     // Production 4.1
47 } else if (tokens_equal(&
    integer_tok, current_tok,
    true) // int
48 || tokens_equal(&real_tok,
    current_tok, true)) //
    real
49 {
50     return standard_type();
51 }
52
53 synch();
54 return ERR;
55 }

```

---

Listing 63: variable.c

---

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3
4 #include "productions.h"
5 #include "../parser.h"
6 #include "../..../tokenizer/
    tokens.h"
7

```

<pre> 8 static const Token* first_set    [] = {&amp;id_tok}; 9 static const int first_size =    sizeof(first_set)/sizeof(    first_set[0]); 10 11 static const Token* sync_set[]    = {&amp;eof_tok, &amp;assignop_tok    }; 12 static const int sync_size =    sizeof(sync_set)/sizeof(    sync_set[0]); 13 14 static void synch() 15 { 16     require_sync(sync_set,    sync_size, first_set,    first_size); 17 } 18 19 // Needs implementing: None 20 LangType variable() 21 { 22     // Production 16 23     if (tokens_equal(&amp;id_tok,    current_tok, false)) // id 24     { 25         Token* id_ref; 26         id_ref = match(&amp;id_tok, false)    ; 27         return array_access(get_type(    id_ref)); 28     } 29 30     synch(); 31     return ERR; 32 } </pre>	<pre>    linkedList/linkedList.h" 6 #include "symbolTable.h" 7 8 LinkedList* symbolTable; 9 10 int initSymbolTable() 11 { 12     symbolTable = malloc(sizeof(*    symbolTable)); 13     symbolTable -&gt; head = 0; 14     return 0; 15 } 16 17 char* pushToSymbolTable(char*    name, size_t length) 18 { 19     add(symbolTable, name, sizeof(    char)*length); 20     return (char *) (symbolTable -&gt;    head -&gt; data); 21 } 22 23 char* checkSymbolTable(char*    word) 24 { 25     // Then check the symbol table 26     struct node* node =    symbolTable -&gt; head; 27     while (node) 28     { 29         if (strcmp((char *) node -&gt;    data, word) == 0) // Match 30         return (char *) (node -&gt; data); 31         node = node -&gt; next; 32     } 33 34     return NULL; 35 } </pre>
--	---

---

Listing 64: symbolTable.c

```

1 #include<stdlib.h>
2 #include<string.h>
3 #include<stdio.h> // TODO
   Remove
4
5 #include "../dataStructures/

```

---

Listing 65: addop.c

```

1 #include "../tokens.h"
2 #include "machines.h"
3
4 int addop(Token* storage, char
   * str, int start)

```

<pre> 5  { 6  storage -&gt; attribute = ADDOP; 7  switch (str[start]) 8  { 9  case '+': 10 storage -&gt; aspect = 0; 11 start++; 12 return start; 13 14 case '-': 15 storage -&gt; aspect = 1; 16 start++; 17 return start; 18 19 default: break; 20 } 21 22 return start; 23 } </pre>	<pre> 21 start++; 22 } else if (str[start] == ',') 23 { 24 storage -&gt; attribute = PUNC; 25 storage -&gt; aspect = 0; 26 start++; 27 } else if (str[start] == ';') 28 { 29 storage -&gt; attribute = PUNC; 30 storage -&gt; aspect = 1; 31 start++; 32 } else if (str[start] == '.') 33 { 34 storage -&gt; attribute = PUNC; 35 storage -&gt; aspect = 2; 36 start++; 37 } 38 39 return start; 40 } </pre>
--	--

---

Listing 66: catchall.c

---

```

1 #include<string.h>
2
3 #include "../tokens.h"
4 #include "machines.h"
5
6 int catchall(Token* storage,
7             char* str, int start)
8 {
9     if (strncmp(&str[start], ":",
10                2) == 0)
11     {
12         storage -> attribute =
13             ASSIGNOP;
14         storage -> aspect = 0;
15         start += 2;
16     } else if (strncmp(&str[start],
17                        ".", 2) == 0)
18     {
19         storage -> attribute = ARRAY;
20         storage -> aspect = 1;
21         start += 2;
22     } else if (str[start] == ':'){
23         storage -> attribute = TYPE;
24         storage -> aspect = 0;

```

---

Listing 67: grouping.c

---

```

1 #include "../tokens.h"
2 #include "machines.h"
3
4 int grouping(Token* storage,
5             char* str, int start)
6 {
7     storage -> attribute = GROUP;
8     switch (str[start])
9     {
10     case '(':
11         storage -> aspect = 0;
12         start++;
13         break;
14     case ')':
15         storage -> aspect = 1;
16         start++;
17         break;
18     case '[':
19         storage -> aspect = 2;
20         start++;
21         break;

```

```

23     case '']':
24         storage -> aspect = 3;
25         start++;
26         break;
27     default:
28         break;
29     }
30     return start;
31 }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

```

---

Listing 68: idres.c

---

```

1  #include<string.h>
2  #include<stdlib.h>
3  #include<ctype.h>
4  #include<stdio.h>
5  #include<stdbool.h>
6
7  #include "machines.h"
8  #include "../errorHandler/
   errorHandler.h"
9  #include "../symbolTable/
   symbolTable.h"
10 #include "../dataStructures
   /linkedList/linkedList.h"
11 #include "../tokens.h"
12
13 static char** reservedWords;
14 static int numReserved;
15 static enum TokenType*
   categories;
16 static int* attributes;
17
18 static int getIndex(const char
   ** array, size_t arr_size,
   char* item)
19 {
20     while (arr_size > 0)
21     {
22         if (strcmp(array[arr_size -
   1], item) == 0)
23             return arr_size - 1;
24         arr_size--;
25     }
26     return -1;
27 }
28
29 static int initResWords(FILE*
   resFile)
30 {
31     static const int length = 11;
32     LinkedList* resWords = malloc(
   sizeof(*resWords));
33     LinkedList* cats = malloc(
   sizeof(*cats));
34     LinkedList* attrs = malloc(
   sizeof(*attrs));
35
36     char word[length] = {0};
37     char attribute[length] = {0};
38     int attr = 0;
39     //while (fgets(word, length,
   resFile))
40     while (true)
41     {
42         fscanf(resFile, "%s", word);
43         if (feof(resFile))
44             break;
45         fscanf(resFile, "%s",
   attribute); // The actual
   name.
46         fscanf(resFile, "%d", &attr);
47         numReserved = add(resWords, &
   word, length*sizeof(char));
48         add(cats, &attribute, length*
   sizeof(char));
49         add(attrs, &attr, sizeof(int))
   ;
50     }
51
52     // Initialize the lexeme table
53     reservedWords = malloc(
   numReserved*sizeof(char*));
54     struct node* node = resWords
   -> head;
55
56     for (size_t i = 0; i <
   numReserved; i++) {
57         reservedWords[i] = (char *)

```



```

        node -> data;
58 node = node -> next;
59 }
60
61 // Initialize the attribute
   table
62 categories = malloc(
        numReserved*sizeof(enum
        TokenType));
63 node = cats -> head;
64
65 for (size_t i = 0; i <
        numReserved; i++) {
66 categories[i] = (enum
        TokenType) getIndex(
        catNames,
67 sizeof(catNames)/sizeof(char*)
        ,
68 (char *) node -> data);
69 node = node -> next;
70 }
71
72 // Initialize the attribute
   table
73 attributes = malloc(
        numReserved*sizeof(int));
74 node = attrs -> head;
75
76 for (size_t i = 0; i <
        numReserved; i++) {
77 attributes[i] = *(int *) node
        -> data;
78 node = node -> next;
79 }
80
81
82 return 0;
83 }
84
85 int initIDResMachine(FILE*
        resFile)
86 {
87 if (initSymbolTable() == 0 &&
        initResWords(resFile) == 0)
88 return 0;
89 else
90 return 1;
91 }
92
93 static int isReserved(char*
        word)
94 {
95 // Check the reserved words
        table for a match first
96 for (size_t i = 0; i <
        numReserved; i++) {
97 if (!reservedWords[i] ||
        strcmp(reservedWords[i],
        word) == 0) // Match
98 return i;
99 }
100
101 return -1;
102 }
103
104 int idres(Token* storage, char
        * str, int start)
105 {
106 int initial = start;
107 LinkedList* id = malloc(sizeof
        (*id));
108 storage -> attribute = ID;
109 storage -> aspect = 0;
110 char next = str[start];
111 if (isalpha(next)) // Can
        actually be an id/reserved
112 {
113 size_t wordSize = 0;
114 do
115 {
116 wordSize = add(id, &next,
        sizeof(char*));
117 start++;
118 next = str[start];
119 } while(isalpha(next) ||
        isdigit(next)); // Match ID
120
121 // The string of the id name
122 char* name = malloc((wordSize
        + 1)*sizeof(char));
123 name[wordSize] = '\0';
124 struct node* node = id -> head

```

```

        ;
125 for (size_t i = 0; i <
        wordSize; i++) {
126 name[wordSize - i - 1] = *(
        char *)(node -> data);
127 node = node -> next;
128 }
129
130 int index = -1;
131 char* address = 0;
132 if ((index = isReserved(name))
        >= 0)
133 { // It's a reserved word!
134 storage -> attribute =
        categories[index];
135 storage -> aspect = attributes
        [index];
136 }
137 else if ((address =
        checkSymbolTable(name)))
138 storage -> id = address;
139 else
140 storage -> id =
        pushToSymbolTable(name,
        wordSize);
141
142 }
143 if (start - initial > 10) //
        ID Too long err
144 {
145 //storage -> attribute = NOOP;
        TODO investigate
146 throw_lex_error(LEXERR, 1,
        initial, start - initial);
147 }
148 return start;
149 }

```

---

Listing 69: mulop.c

```

1 #include "../tokens.h"
2 #include "machines.h"
3
4 int mulop(Token* storage, char
        * str, int start)
5 {

```

```

6 storage -> attribute = MULOP;
7 if (str[start] == '*')
8 {
9 storage -> aspect = 0;
10 start++;
11 } else if (str[start] == '/')
12 {
13 storage -> aspect = 1;
14 start++;
15 }
16
17 return start;
18 }

```

---

Listing 70: numbers.c

```

1 #include<stdbool.h>
2 #include<stdlib.h>
3 #include<ctype.h>
4
5 #include "../tokens.h"
6 #include "machines.h"
7 #include "../errorHandler/
        errorHandler.h"
8
9 // Assumes that "str" is valid
        as an integer.
10 char* parseNum(LinkedList*
        chars, bool real)
11 {
12 char* num = malloc((chars ->
        size + 1) * sizeof(char));
13 size_t count = chars -> size;
14 num[count--] = 0;
15 struct node* node = chars ->
        head;
16 while (node)
17 {
18 num[count--] = *(char *)node
        -> data;
19 node = node -> next;
20 }
21
22 return num;
23 }
24

```

```

25 double parseReal(LinkedList*
    digits)
26 {
27     char* array = parseNum(digits,
        true);
28     double val = strtod(array,
        NULL);
29     free(array);
30     return val;
31 }
32
33 int parseInt(LinkedList*
    digits)
34 {
35     char* array = parseNum(digits,
        false);
36     int val = (int) strtol(array,
        NULL, 10);
37     free(array);
38     return val;
39 }
40
41 int intMachine(Token* storage,
    char* str, int start)
42 {
43     storage->attribute = NUM;
44
45     bool errored = false;
46     int initial = start;
47
48     LinkedList* digits = malloc(
        sizeof(*digits));
49     while (isdigit(str[start]))
50         add(digits, &str[start++],
            sizeof(char*));
51
52     if (start - initial > 10)
53     {
54         errored = true;
55         throw_lex_error(LEXERR, 2,
            initial, start - initial);
56     }
57     if (start > initial + 1 && str
        [initial] == '0')
58     {
59         errored = true;
60         throw_lex_error(LEXERR, 7,
            initial, start - initial);
61     }
62     // TODO investigate (all of
        these machines)
63     /*if (errored)
64         storage -> attribute =
        NOOP;
65     else*/ if (start > initial)
        // It's a proper
        integer!
66     {
67         storage->aspect = 0;
68         storage->int_val = parseInt(
            digits);
69     }
70
71     return start;
72 }
73
74 // NOTE: Pay attention to
        memory stuff here (the
        linked list takes up space)
        .
75 int realMachine(Token* storage
    , char* str, int start)
76 {
77     storage->attribute = NUM;
78
79     int initial = start;
80     bool errored = false;
81
82     int intPart = 0;
83     int fracPart = 0;
84
85     LinkedList* digits = malloc(
        sizeof(*digits));
86     while (isdigit(str[start]))
87         add(digits, &str[start++],
            sizeof(char*));
88
89     intPart = start - initial;
90     if (intPart == 0) // Not a
        real. Must start with a
        digit.
91     return initial;

```

```

92     if (str[start] == '.')
93     add(digits, &str[start++],
94         sizeof(char*));
95     else // Not a real
96     return initial;
97
98
99     while (isdigit(str[start]))
100    add(digits, &str[start++],
101        sizeof(char*));
102    fracPart = start - (initial +
103        intPart + 1);
104
105    if (fracPart == 0) // Not a
106        real
107    return initial;
108
109    // Now, we check for errors.
110    if (intPart > 5)
111    {
112        throw_lex_error(LEXERR, 3,
113            initial, start - initial);
114        errored = true;
115    }
116
117    if (fracPart > 5)
118    {
119        throw_lex_error(LEXERR, 4,
120            initial, start - initial);
121        errored = true;
122    }
123
124    if (str[start - 1] == '0' &&
125        fracPart > 1) // Leading
126        zero!
127    {
128        throw_lex_error(LEXERR, 8,
129            initial, start - initial);
130        errored = true;
131    }
132
133    if (str[start - 1] == '0' &&
134        fracPart > 1) // Trailing
135        zero!
136    {
137        throw_lex_error(LEXERR, 9,
138            initial, start - initial);
139
140        errored = true;
141    }
142
143    /*
144     * if (errored)
145     *     storage -> attribute =
146     *         NOOP;*/
147
148    else
149    {
150        storage -> aspect = 1;
151        storage -> real_val =
152            parseReal(digits);
153    }
154
155    return start;
156 }
157
158 int longRealMachine(Token*
159     storage, char* str, int
160     start)
161 {
162     storage -> attribute = NUM;
163
164     int initial = start;
165     bool errored = false;
166
167     int intPart = 0;
168     int fracPart = 0;
169     int expPart = 0;
170
171     LinkedList* digits = malloc(
172         sizeof(*digits));
173     while (isdigit(str[start]))
174     add(digits, &str[start++],
175         sizeof(char*));
176
177     intPart = start - initial;
178     if (intPart == 0) // Not a
179         real. Must start with a
180         digit.
181     return initial;
182
183     // REAL part
184     if (str[start] == '.')
185     add(digits, &str[start++],
186         sizeof(char*));
187     else // Not a real

```

```

163 return initial;
164
165
166 while (isdigit(str[start]))
167 add(digits, &str[start++],
168     sizeof(char*));
169
170 fracPart = start - (initial +
171     intPart + 1);
172
173 if (fracPart == 0) // Not a
174     real
175 return initial;
176
177 // LONG REAL part
178 int signum = 0;
179
180 if (str[start] == 'E')
181 add(digits, &str[start++],
182     sizeof(char*));
183 else // Not a long real
184 return initial;
185
186 if (str[start] == '+' || str[
187     start] == '-')
188 {
189     signum++;
190     add(digits, &str[start++],
191         sizeof(char*));
192 }
193
194 while (isdigit(str[start]))
195 add(digits, &str[start++],
196     sizeof(char*));
197
198 expPart = start - (initial +
199     fracPart + intPart + signum
200     + 2);
201
202 if (expPart == 0) // Not a
203     long real
204 return initial;
205
206 // Now, we check for errors.
207
208 if (intPart > 5)
209 {
210     throw_lex_error(LEXERR, 3,
211         initial, start - initial);
212     errored = true;
213 }
214
215 if (fracPart > 5)
216 {
217     throw_lex_error(LEXERR, 4,
218         initial, start - initial);
219     errored = true;
220 }
221
222 if (str[initial] == '0' &&
223     intPart > 1) // Leading
224     zero!
225 {
226     throw_lex_error(LEXERR, 8,
227         initial, start - initial);
228     errored = true;
229 }
230
231 if (str[start - expPart - 2]
232     == '0' && fracPart > 1) //
233     Trailing zero in real!
234 {
235     throw_lex_error(LEXERR, 9,
236         initial, start - initial);
237     errored = true;
238 }
239
240 if (expPart > 2) // Exponent
241     too long!
242 {
243     throw_lex_error(LEXERR, 5,
244         initial, start - initial);
245     errored = true;
246 }
247
248 if (str[start - expPart] == '0
249     ') // Leading zero in
250     exponent!
251 {
252     throw_lex_error(LEXERR, 10,
253         initial, start - initial);
254     errored = true;
255 }
256
257 /*
258     if (errored)
259         storage -> attribute =

```

```

232     NOOP;
233     else*/
234 {
235     storage -> aspect = 1;
236     storage -> real_val =
237         parseReal(digits);
238 }
239 return start;

```

---

Listing 71: relop.c

---

```

1  #include "../tokens.h"
2  #include "machines.h"
3
4  int relop(Token* storage, char
5      * str, int start)
6  {
7      storage -> attribute = RELOP;
8      char next = str[start];
9      switch (next) {
10         case '<':
11             start++;
12             if (str[start] == '=')
13             {
14                 storage -> aspect = 1;
15                 start++;
16             } else if (str[start] == '>')
17             {
18                 storage -> aspect = 5;
19                 start++;
20             } else {
21                 storage -> aspect = 0;
22             }
23             break;
24         case '=':
25             start++;
26             storage -> aspect = 2;
27             break;
28         case '>':
29             start++;
30             if (str[start] == '=')
31             {
32

```

```

33     storage -> aspect = 4;
34     start++;
35     } else {
36         storage -> aspect = 3;
37     }
38     break;
39
40     default: break; // Do not
41         increment; continue on to
42         the next machine.
43 }
44
45 return start;

```

---

Listing 72: whitespace.c

---

```

1  #include<stdlib.h>
2  #include<ctype.h>
3
4  #include "../tokens.h"
5  #include "machines.h"
6
7  int whitespace(Token* storage,
8      char* str, int start)
9  {
10     storage -> attribute = WS;
11     if (isspace(str[start]))
12     {
13         storage -> aspect = 0;
14         if (str[start] == '\n')
15         storage -> aspect = 1;
16         start++;
17     }
18     return start;

```

---

Listing 73: tokenizer.c

---

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<stdbool.h>
4  #include<string.h>
5
6  #include "tokenizer.h"
7  #include "../dataStructures/
8      linkedList/linkedlist.h"

```

```

8 #include "machines/machines.h"
9 #include "../errorHandler/
  errorHandler.h"
10 #include "../globals/globals.h
    "
11
12 const machine machines[] = {
    whitespace, idres,
    longRealMachine,
13 realMachine, intMachine,
    grouping, catchall, relop,
    addop, mulop};
14
15 // Initialization stuff
16 static bool initialized =
    false;
17
18 int initializeTokens(FILE*
    resFile)
19 {
20     if (resFile) {
21         initIDResMachine(resFile);
22         initialized = true;
23     } else {
24         fprintf(stderr, "%s\n", "
            Reserved words file for
            tokenizer null!");
25     }
26     return 1;
27 }
28
29 static Token*
    generateNextToken()
30 {
31     if (initialized) {
32         Token* current = malloc(sizeof
            (*current)); // TODO
            necessary allocation?
33         if ((current =
            getNextErrorToken()) !=
            NULL)
34             return current;
35         else
36             current = malloc(sizeof(*
                current));
37
38         int end;
39         current -> start = START;
40         for (int i = 0; i < sizeof(
            machines)/sizeof(machine);
            i++)
41         {
42             current -> aspect = 0;
43             end = (*machines[i])(current,
                BUFFER, START);
44             if (end > START) {
45                 current -> length = end -
                    START;
46                 START = end;
47                 return current;
48             }
49         }
50
51         // Unrecognized symbol error.
52         // This error is manual
53         // because it takes
54         // the place of a lexeme,
55         // rather than being processed
56         // during one.
57         throw_lex_error(LEXERR, 0,
            START, 1);
58         //current -> attribute = NOOP;
59         START++;
60         return current;
61     } else {
62         fprintf(stderr, "%s\n", "
            Tokenizer not initialized.
            Aborting.");
63     }
64     return NULL;
65 }
66
67 Token* getNextToken()
68 {
69     Token* next = malloc(sizeof(*
        next));
70     do {
71         next = generateNextToken();
72     } while (next -> attribute ==
        NOOP);

```

<pre> 71 return next; 72 } </pre>	<pre> 26 const Token period_tok = {PUNC     , 2, true, 0, 0}; 27 const Token lbrac_tok = {GROUP     , 2, true, 0, 0}; 28 const Token rbrac_tok = {GROUP     , 3, true, 0, 0}; 29 const Token addop_tok = {ADDOP     , 0, false, 0, 0}; 30 const Token array_tok = {ARRAY     , 0, true, 0, 0}; 31 const Token assignop_tok = {     ASSIGNOP, 0, true, 0, 0}; 32 const Token begin_tok = {     CONTROL, 0, true, 0, 0}; 33 const Token call_tok = {     CONTROL, 10, true, 0, 0}; 34 const Token do_tok = {CONTROL,     1, true, 0, 0}; 35 const Token else_tok = {     CONTROL, 2, true, 0, 0}; 36 const Token end_tok = {CONTROL     , 3, true, 0, 0}; 37 const Token id_tok = {ID, 0,     false, 0, 0}; 38 const Token if_tok = {CONTROL,     5, true, 0, 0}; 39 const Token integer_tok = {     TYPE, 1, true, 0, 0}; 40 const Token integer_val_tok =     {NUM, 0, true, 0, 0}; 41 const Token of_tok = {ARRAY,     2, true, 0, 0}; 42 const Token real_val_tok = {     NUM, 1, true, 0, 0}; 43 const Token mulop_tok = {MULOP     , 0, false, 0, 0}; 44 const Token not_tok = {INVERSE     , 0, true, 0, 0}; 45 const Token num_tok = {NUM, 0,     false, 0, 0}; 46 const Token procedure_tok = {     CONTROL, 6, true, 0, 0}; 47 const Token program_tok = {     CONTROL, 7, true, 0, 0}; 48 const Token real_tok = {TYPE, </pre>
-----------------------------------	---

---

Listing 74: tokens.c

<pre> 1 #include&lt;string.h&gt; 2 #include&lt;stdlib.h&gt; 3 #include&lt;stdio.h&gt; 4 5 #include "../errorHandler/     errorHandler.h" 6 #include "tokens.h" 7 8 const char* catNames[] = {"     NOOP", "FILEEND", "ASSIGNOP     ", "RELOP", "ID", 9 "CONTROL", "ADDOP", "MULOP", "     WS", "ARRAY", "TYPE", 10 "VAR", "NUM", "PUNC", "GROUP",     "INVERSE", 11 "LEXERR", "SYNERR", "SEMERR"}; 12 13 const char* typeNames[] = {"     ERR", "REAL", "INT", "BOOL"     , "PROGRAM", 14 "PROGRAM_PARAMETER", "     PROCEDURE", 15 "INT ARRAY", "REAL ARRAY"}; 16 17 const Token eof_tok = {FILEEND     , 0, false, 0, 0}; 18 const Token lparen_tok = {     GROUP, 0, true, 0, 0}; 19 const Token rparen_tok = {     GROUP, 1, true, 0, 0}; 20 const Token plus_tok = {ADDOP,     0, true, 0, 0}; 21 const Token comma_tok = {PUNC,     0, true, 0, 0}; 22 const Token minus_tok = {ADDOP     , 1, true, 0, 0}; 23 const Token semic_tok = {PUNC,     1, true, 0, 0}; 24 const Token colon_tok = {TYPE,     0, true, 0, 0}; 25 const Token dotdot_tok = { </pre>	<pre>     ARRAY, 1, true, 0, 0}; </pre>
--	---



```

        2, true, 0, 0};
49 const Token relop_tok = {RELOP
    , 0, false, 0, 0};
50 const Token then_tok = {
    CONTROL, 8, true, 0, 0};
51 const Token var_tok = {VAR, 0,
    true, 0, 0};
52 const Token while_tok = {
    CONTROL, 9, true, 0, 0};
53
54 static const char* lexes[] = {
    "(", ")", "+", "-", ";",
    ":", "[", "]", "addop",
55 "array", "assignop", "begin",
    "call", "do", "else",
56 "end", "ID", "if", "integer",
    "mulop", "not",
57 "num", "procedure", "program",
    "real", "relop",
58 "then", "var", "while", "EOF",
    "!", ":", ".",
59 "int value", "of", "real value
    "};
60
61 static const Token* tokens[] =
    {&lparen_tok, &rparen_tok,
    &plus_tok, &comma_tok, &
    minus_tok, &semic_tok,
62 &colon_tok, &lbrac_tok, &
    rbrac_tok, &addop_tok, &
    array_tok, &assignop_tok,
63 &begin_tok, &call_tok, &do_tok
    , &else_tok, &end_tok, &
    id_tok,
64 &if_tok, &integer_tok, &
    mulop_tok, &not_tok, &
    num_tok,
65 &procedure_tok, &program_tok,
    &real_tok, &relop_tok, &
    then_tok,
66 &var_tok, &while_tok, &eof_tok
    , &dotdot_tok,
67 &colon_tok, &period_tok, &
    integer_val_tok,
68 &of_tok, &real_val_tok};
69
70 const Token* getTokenFromLex(
    char* lex) {
71 for (int i = 0; i < sizeof(
    lexes); i++) {
72 if (strcmp(lexes[i], lex) ==
    0)
73 return tokens[i];
74 }
75
76 return NULL;
77 }
78
79 const char* getLexFromToken(
    Token* token, bool strict)
    {
80 switch (token->attribute) {
81 case FILEEND: return "EOF";
82 case ASSIGNOP: return "!=";
83
84 case RELOP: if (strict)
85 switch (token->aspect) {
86 case 0: return "<";
87 case 1: return "<=";
88 case 2: return "=";
89 case 3: return ">";
90 case 4: return ">=";
91 case 5: return "<>";
92 }
93 else return "RELOP";
94
95 case ID: return "ID";
96
97 case CONTROL: if (!strict)
98 return "CONTROL"; else
99 switch (token->aspect) {
100 case 0: return "begin";
101 case 1: return "do";
102 case 2: return "else";
103 case 3: return "end";
104 case 4: return "function";
105 case 5: return "if";
106 case 6: return "procedure";
107 case 7: return "program";
108 case 8: return "then";
109 case 9: return "while";
110 case 10: return "call";

```

```

110 }
111
112 case ADDOP: if (!strict)
113     return "ADDOP"; else
114     switch (token -> aspect) {
115     case 0: return "+";
116     case 1: return "-";
117     }
118 case MULOP: if (!strict)
119     return "MULOP"; else
120     switch (token -> aspect) {
121     case 0: return "*";
122     case 1: return "/";
123     }
124 case ARRAY: if (!strict)
125     return "ARRAY"; else
126     switch (token -> aspect) {
127     case 0: return "array";
128     case 1: return "..";
129     case 2: return "of";
130     }
131 case TYPE: switch (token ->
132     aspect) {
133     case 0: return ":";
134     case 1: return "integer";
135     case 2: return "real";
136     }
137 case VAR: switch (token ->
138     aspect) {
139     case 0: return "var";
140     }
141 case NUM: if (!strict) return
142     "a number"; else
143     switch (token -> aspect) {
144     case 0: return "integer value"
145     ;
146     case 1: return "real value";
147     }
148 case PUNC: switch (token ->
149     aspect) {
150     case 0: return ",";
151     case 1: return ";";
152     case 2: return ".";
153     }
154 case GROUP: switch (token ->
155     aspect) {
156     case 0: return "(";
157     case 1: return ")";
158     case 2: return "[";
159     case 3: return "]";
160     }
161 case INVERSE: switch (token ->
162     aspect) {
163     case 0: return "not";
164     }
165 case NOOP:
166 case WS:
167 case LEXERR:
168 case SYNER:
169 case SEMERR: return "An error
170     in the compiler has
171     occurred.";
172 // Returns true if the tokens
173 // are equivalent, false
174 // otherwise
175 bool tokens_equal(const Token*
176     p1, Token* p2, bool strict
177     ) {
178     return p1 -> attribute == p2
179         -> attribute &&
180         (!strict || p1 -> aspect == p2
181             -> aspect);
182 }
183
184 LangType convert_to_array(
185     LangType type) {
186     char* errorMessage;
187     switch (type) {
188     case INT: return AINT;
189     case REAL: return AREAL;

```

```

183 // Type mismatch!!
184 default: errorMessage= calloc
185     (150, sizeof(*errorMessage)
186     );
187 sprintf(errorMessage, "Attempt
188     to create array using type
189     %s; must use integer or
190     real instead!", typeName[
191     type]);
192 throw_sem_error(errorMessage);
193 case ERR: return ERR;
194 }
195 }
196
197 LangType convert_from_array(
198     LangType type) {
199     char* errorMessage;
200     switch (type) {
201     case AINT: return INT;
202     case AREAL: return REAL;
203
204     default: errorMessage = calloc
205         (100, sizeof(*errorMessage)
206         );
207     sprintf(errorMessage, "Attempt
208         to index variable of type
209         %s!", typeName[type]);
210     throw_sem_error(errorMessage);
211     case ERR: return ERR;
212     }
213 }
214
215 static LangType
216     assignop_lookup(LangType
217         first, LangType second) {
218     char* errorMessage;
219     if (first == ERR || second ==
220         ERR) // just an err
221     return ERR;
222     else if (first != INT && first
223         != REAL) {
224     errorMessage= calloc(100,
225         sizeof(*errorMessage));
226     sprintf(errorMessage, "Cannot
227         assign values to variables
228         of type %s!", typeName[
229         first]);
230     throw_sem_error(errorMessage);
231     return ERR;
232     }
233     else if (second != INT &&
234         second != REAL) {
235     errorMessage= calloc(100,
236         sizeof(*errorMessage));
237     sprintf(errorMessage, "Attempt
238         to assign %s value; only
239         reals and integers can be
240         assigned!", typeName[
241         second]);
242     throw_sem_error(errorMessage);
243     return ERR;
244     }
245     else if (first != second) {
246     errorMessage= calloc(100,
247         sizeof(*errorMessage));
248     sprintf(errorMessage, "Attempt
249         to convert type %s into
250         type %s in assignment!",
251         typeName[first], typeName[
252         second]);
253     throw_sem_error(errorMessage);
254     return ERR;
255     }
256     return NULL;
257 }
258
259 static LangType relop_lookup(
260     LangType first, LangType
261     second) {
262     char* errorMessage;
263     if (first == second && (first
264         == INT || first == REAL))
265     return BOOL;
266     else if (first != ERR &&
267         second != ERR) {
268     errorMessage= calloc(100,
269         sizeof(*errorMessage));
270     sprintf(errorMessage, "Attempt

```

```

        to compare incompatible
        types %s and %s!",
        typeNameNames[first], typeNameNames
        [second]);
240 throw_sem_error(errorMessage);
241 }
242
243 return ERR;
244 }
245
246 static LangType addop_lookup(
        LangType first, LangType
        second, int opcode) {
247 char* errorMessage;
248 switch (opcode) {
249 case 0:
250 case 1: if (first == second &&
        (first == INT || first ==
        REAL))
251 return first;
252 else if (first != ERR &&
        second != ERR) {
253 errorMessage= calloc(100,
        sizeof(*errorMessage));
254 sprintf(errorMessage, "Attempt
        to add incompatible types
        %s and %s!", typeNameNames[
        first], typeNameNames[second]);
255 throw_sem_error(errorMessage);
256 return ERR;
257 }
258
259 return ERR;
260
261 case 2: if (first == second &&
        first == BOOL)
262 return BOOL;
263 else if (first != ERR &&
        second != ERR) {
264 errorMessage= calloc(100,
        sizeof(*errorMessage));
265 sprintf(errorMessage, "
        Expected BOOL and BOOL for
        use with 'or', received %s
        and %s!", typeNameNames[first],
        typeNameNames[second]);
        throw_sem_error(errorMessage);
        return ERR;
        }
        default: return NULL;
    }
}

static LangType mulop_lookup(
        LangType first, LangType
        second, int opcode) {
277 char* errorMessage;
278 switch (opcode) {
279 case 0:
280 case 1: if (first == second &&
        (first == INT || first ==
        REAL))
281 return first;
282 else if ((first == REAL &&
        second == INT)
        || (first == INT && second ==
        REAL)) {
283 errorMessage= calloc(100,
        sizeof(*errorMessage));
284 sprintf(errorMessage, "Attempt
        to multiply or divide
        incompatible types %s and
        %s!", typeNameNames[first],
        typeNameNames[second]);
285 throw_sem_error(errorMessage);
286 }
287 else if (first != ERR &&
        second != ERR) {
288 errorMessage= calloc(100,
        sizeof(*errorMessage));
289 sprintf(errorMessage, "
        Expceted ints or reals for
        multiplication, received %s
        and %s!", typeNameNames[first]
        , typeNameNames[second]);
290 throw_sem_error(errorMessage);
291 }
292 }
293 }
294

```

```

295 return ERR;
296
297
298 case 2: if (first == second &&
           first == BOOL) // and
299 return BOOL;
300 else if (first != ERR &&
           second != ERR)
301 {
302     errorMessage= calloc(100,
                           sizeof(*errorMessage));
303     sprintf(errorMessage, "
           Expected BOOL and BOOL for
           use with 'and', received %s
           and %s!", typeNames[first
           ], typeNames[second]);
304     throw_sem_error(errorMessage);
305 }
306
307 return ERR;
308
309 case 3: // div; mod
310 case 4: if (first == second &&
           first == INT)
311 return INT;
312 else if (first != ERR &&
           second != ERR) {
313     errorMessage= calloc(100,
                           sizeof(*errorMessage));
314     sprintf(errorMessage,
315 "Integers required with %s,
           received %s and %s!",
316 opcode == 3 ? "div" : "mod",
           typeNames[first],
           typeNames[second]);
317     throw_sem_error(errorMessage);
318 }
319
320
321 return ERR;
322
323 default: return NULL;
324 }
325 }
326
327 static LangType not_lookup(
           LangType first, LangType
           second) {
328     char* errorMessage;
329
330     if (first == BOOL)
331         return BOOL;
332     else if (first != ERR)
333     {
334         errorMessage= calloc(100,
                               sizeof(*errorMessage));
335         sprintf(errorMessage, "
           Expected BOOL use with 'not
           ', received %s!", typeNames
           [first]);
336         throw_sem_error(errorMessage);
337     }
338     return ERR;
339 }
340
341
342 static LangType array_lookup(
           LangType first, LangType
           second) {
343     if (first == second && first
        == INT)
344         return INT;
345     else if (first != ERR)
346     {
347         char* errorMessage = calloc
           (100, sizeof(*errorMessage)
           );
348         sprintf(errorMessage, "Attempt
           to index variable of type
           %s!", typeNames[first]);
349         throw_sem_error(errorMessage);
350     } else if (second != ERR){
351         char* errorMessage = calloc
           (100, sizeof(*errorMessage)
           );
352         sprintf(errorMessage, "Attempt
           to use variable of type %s
           to index array!",
           typeNames[second]);
353         throw_sem_error(errorMessage);
354     }
355     return ERR;

```

```

357 }
358
359 LangType type_lookup(LangType
    first, LangType second,
    Token* op) {
360 if (first == ERR || second ==
    ERR || op == NULL)
361 return ERR;
362
363 switch (op -> attribute) {
364 // Operations which are
    meaningless
365 case NOOP:
366 case LEXERR:
367 case SYNERR:
368 case SEMERR:
369 case GROUP:
370 case PUNC:
371 case FILEEND:
372 case ID:
373 case CONTROL:
374 case WS:
375 case TYPE:
376 case VAR:
377 case NUM: return NULL;
378
379 case ASSIGNOP: return
    assignop_lookup(first,
    second);
380 case RELOP: return
    relop_lookup(first, second)
    ;
381 case ADDOP: return
    addop_lookup(first, second,
    op -> aspect);
382 case ARRAY: return
    array_lookup(first, second)
    ;
383 case MULOP: return
    mulop_lookup(first, second,
    op -> aspect);
384 case INVERSE: return
    not_lookup(first, second);
385
386 }
387 }

```