

# **CS 4013: Compiler Construction: Project 2**

Nate Beckemeyer

December 2016

## Introduction

For Project 2, I massaged the modified Pascal grammar into an  $LL(1)$  grammar. According to that grammar, I implemented a recursive descent parser to construct the parse tree.

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

## 1 Methodology

The recursive descent parser matches productions, starting with the “program” production. The productions are outlined in the grammars included below. The documents are in the following order:

1. The initial grammar without epsilon productions.
2. The new grammar having eliminated left recursion.
3. That grammar now left-factored to become an  $LL(1)$  grammar.
4. The first and follows sets for each production.
5. The parse table.

1.1.	<i>program</i>	→	<b>program id ( identifier_list ) ;</b> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
1.2.	<i>program</i>	→	<b>program id ( identifier_list ) ;</b> <i>declarations</i> <i>compound_statement</i>
1.3.	<i>program</i>	→	<b>program id ( identifier_list ) ;</b> <i>subprogram_declarations</i> <i>compound_statement</i>
1.4.	<i>program</i>	→	<b>program id ( identifier_list ) ;</b> <i>compound_statement</i>
2.1.	<i>identifier_list</i>	→	<b>id</b>
2.2.	<i>identifier_list</i>	→	<i>identifier_list</i> , <b>id</b>
3.1.	<i>declarations</i>	→	<b>var id : type ;</b>
3.2.	<i>declarations</i>	→	<i>declarations</i> <b>var id : type ;</b>
4.1.	<i>type</i>	→	<i>standard_type</i>
4.2.	<i>type</i>	→	<b>array [ num .. num ] of</b> <i>standard_type</i>
5.1.	<i>standard_type</i>	→	<b>integer</b>
5.2.	<i>standard_type</i>	→	<b>real</b>
6.1.	<i>subprogram_declarations</i>	→	<i>subprogram_declaration</i> ;
6.2.	<i>subprogram_declarations</i>	→	<i>subprogram_declarations</i> <i>subprogram_declaration</i> ;
7.1.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
7.2.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>subprogram_declarations</i> <i>compound_statement</i>
7.3.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>declarations</i> <i>compound_statement</i>
7.4.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>compound_statement</i>
8.	<i>subprogram_head</i>	→	<b>procedure id arguments ;</b>
9.1.	<i>arguments</i>	→	( <i>parameter_list</i> )
9.2.	<i>arguments</i>	→	ϵ
10.1.	<i>parameter_list</i>	→	<b>id : type</b>
10.2.	<i>parameter_list</i>	→	<i>parameter_list</i> ; <b>id : type</b>
11.	<i>compound_statement</i>	→	<b>begin</b> <i>optional_statements</i> <b>end</b>
12.1.	<i>optional_statements</i>	→	<i>statement_list</i>
12.2.	<i>optional_statements</i>	→	ϵ
13.1.	<i>statement_list</i>	→	<i>statement</i>
13.2.	<i>statement_list</i>	→	<i>statement_list</i> ; <i>statement</i>
14.1.	<i>statement</i>	→	<i>variable</i> <b>assignop</b> <i>expression</i>
14.2.	<i>statement</i>	→	<i>procedure_statement</i>
14.3.	<i>statement</i>	→	<i>compound_statement</i>
14.4.	<i>statement</i>	→	<b>if expression then</b> <i>statement</i>
14.5.	<i>statement</i>	→	<b>if expression then</b> <i>statement</i> <b>else</b> <i>statement</i>
14.6.	<i>statement</i>	→	<b>while expression do</b> <i>statement</i>
15.1.	<i>variable</i>	→	<b>id</b>
15.2.	<i>variable</i>	→	<b>id [ expression ]</b>
16.1.	<i>procedure_statement</i>	→	<b>call id</b>
16.2.	<i>procedure_statement</i>	→	<b>call id ( expression_list )</b>

17.1.	<i>expression_list</i>	→	<i>expression</i>
17.2.	<i>expression_list</i>	→	<i>expression_list</i> , <i>expression</i>
18.1.	<i>expression</i>	→	<i>simple_expression</i>
18.2.	<i>expression</i>	→	<i>simple_expression</i> <b>relop</b> <i>simple_expression</i>
19.1.	<i>simple_expression</i>	→	<i>term</i>
19.2.	<i>simple_expression</i>	→	<i>sign</i> <i>term</i>
19.3.	<i>simple_expression</i>	→	<i>simple_expression</i> <b>addop</b> <i>term</i>
20.1.	<i>term</i>	→	<i>factor</i>
20.2.	<i>term</i>	→	<i>term</i> <b>mulop</b> <i>factor</i>
21.1.	<i>factor</i>	→	<b>id</b>
21.2.	<i>factor</i>	→	<b>id</b> [ <i>expression</i> ]
21.3.	<i>factor</i>	→	<b>num</b>
21.4.	<i>factor</i>	→	( <i>expression</i> )
21.5.	<i>factor</i>	→	<b>not</b> <i>factor</i>
22.1.	<i>sign</i>	→	+
22.2.	<i>sign</i>	→	−

1.1.	<i>program</i>	→	<b>program</b> <i>id</i> ( <i>identifier_list</i> ) ; <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i> .
1.2.	<i>program</i>	→	<b>program</b> <i>id</i> ( <i>identifier_list</i> ) ; <i>declarations</i> <i>compound_statement</i> .
1.3.	<i>program</i>	→	<b>program</b> <i>id</i> ( <i>identifier_list</i> ) ; <i>subprogram_declarations</i> <i>compound_statement</i> .
1.4.	<i>program</i>	→	<b>program</b> <i>id</i> ( <i>identifier_list</i> ) ; <i>compound_statement</i> .
2.1.	<i>identifier_list</i>	→	<b>id</b> <i>identifier_list'</i>
2.2.1.	<i>identifier_list'</i>	→	, <b>id</b> <i>identifier_list'</i>
2.2.2.	<i>identifier_list'</i>	→	ε
3.1.	<i>declarations</i>	→	<b>var</b> <i>id</i> : <i>type</i> ; <i>declarations'</i>
3.2.1.	<i>declarations'</i>	→	<b>var</b> <i>id</i> : <i>type</i> ; <i>declarations'</i>
3.2.2.	<i>declarations'</i>	→	ε
4.1.	<i>type</i>	→	<i>standard_type</i>
4.2.	<i>type</i>	→	<b>array</b> [ <b>num</b> .. <b>num</b> ] <b>of</b> <i>standard_type</i>
5.1.	<i>standard_type</i>	→	<b>integer</b>
5.2.	<i>standard_type</i>	→	<b>real</b>
6.1.	<i>subprogram_declarations</i>	→	<i>subprogram_declaration</i> ; <i>subprogram_declarations'</i>
6.2.1.	<i>subprogram_declarations'</i>	→	<i>subprogram_declaration</i> ; <i>subprogram_declarations'</i>
6.2.2.	<i>subprogram_declarations'</i>	→	ε
7.1.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
7.2.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>declarations</i> <i>compound_statement</i>
7.3.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>subprogram_declarations</i> <i>compound_statement</i>
7.4.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>compound_statement</i>
8.	<i>subprogram_head</i>	→	<b>procedure</b> <i>id</i> <i>arguments</i> ;
9.1.	<i>arguments</i>	→	( <i>parameter_list</i> )
9.2.	<i>arguments</i>	→	ε
10.1.	<i>parameter_list</i>	→	<b>id</b> : <i>type</i> <i>parameter_list'</i>
10.2.1.	<i>parameter_list'</i>	→	; <b>id</b> : <i>type</i> <i>parameter_list'</i>
10.2.2.	<i>parameter_list'</i>	→	ε
11.	<i>compound_statement</i>	→	<b>begin</b> <i>optional_statements</i> <b>end</b>
12.1.	<i>optional_statements</i>	→	<i>statement_list</i>
12.2.	<i>optional_statements</i>	→	ε
13.1.	<i>statement_list</i>	→	<i>statement</i> <i>statement_list'</i>
13.2.1.	<i>statement_list'</i>	→	; <i>statement</i> <i>statement_list'</i>
13.2.2.	<i>statement_list'</i>	→	ε
14.1.	<i>statement</i>	→	<i>variable</i> <b>assignop</b> <i>expression</i>
14.2.	<i>statement</i>	→	<i>procedure_statement</i>
14.3.	<i>statement</i>	→	<i>compound_statement</i>
14.4.	<i>statement</i>	→	<b>if</b> <i>expression</i> <b>then</b> <i>statement</i>
14.5.	<i>statement</i>	→	<b>if</b> <i>expression</i> <b>then</b> <i>statement</i> <b>else</b> <i>statement</i>

14.6.	<i>statement</i>	→	<b>while</b> <i>expression</i> <b>do</b> <i>statement</i>
15.1.	<i>variable</i>	→	<b>id</b>
15.2.	<i>variable</i>	→	<b>id</b> [ <i>expression</i> ]
16.1.	<i>procedure_statement</i>	→	<b>call id</b>
16.2.	<i>procedure_statement</i>	→	<b>call id</b> ( <i>expression_list</i> )
17.1.	<i>expression_list</i>	→	<i>expression</i> <i>expression_list'</i>
17.2.1.	<i>expression_list'</i>	→	, <i>expression</i> <i>expression_list'</i>
17.2.2.	<i>expression_list'</i>	→	ε
18.1.	<i>expression</i>	→	<i>simple_expression</i>
18.2.	<i>expression</i>	→	<i>simple_expression</i> <b>relop</b> <i>simple_expression</i>
19.1.	<i>simple_expression</i>	→	<i>term</i> <i>simple_expression'</i>
19.2.	<i>simple_expression</i>	→	<i>sign</i> <i>term</i> <i>simple_expression'</i>
19.3.1.	<i>simple_expression'</i>	→	<b>addop</b> <i>term</i> <i>simple_expression'</i>
19.3.2.	<i>simple_expression'</i>	→	ε
20.1.	<i>term</i>	→	<i>factor</i> <i>term'</i>
20.2.1.	<i>term'</i>	→	<b>mulop</b> <i>factor</i> <i>term'</i>
20.2.2.	<i>term'</i>	→	ε
21.1.	<i>factor</i>	→	<b>id</b>
21.2.	<i>factor</i>	→	<b>id</b> [ <i>expression</i> ]
21.3.	<i>factor</i>	→	<b>num</b>
21.4.	<i>factor</i>	→	( <i>expression</i> )
21.5.	<i>factor</i>	→	<b>not</b> <i>factor</i>
22.1.	<i>sign</i>	→	+
22.2.	<i>sign</i>	→	−

1.	<i>program</i>	→	<b>program</b> <i>id</i> ( <i>identifier_list</i> ) ; <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i> .
2.1.	<i>identifier_list</i>	→	<b>id</b> <i>identifier_list'</i>
2.2.1.	<i>identifier_list'</i>	→	, <b>id</b> <i>identifier_list'</i>
2.2.2.	<i>identifier_list'</i>	→	ϵ
3.1.	<i>declarations</i>	→	<b>var</b> <i>id</i> : <i>type</i> ; <i>declarations</i>
3.2.	<i>declarations</i>	→	ϵ
4.1.	<i>type</i>	→	<i>standard_type</i>
4.2.	<i>type</i>	→	<b>array</b> [ <i>num</i> .. <i>num</i> ] <b>of</b> <i>standard_type</i>
5.1.	<i>standard_type</i>	→	<b>integer</b>
5.2.	<i>standard_type</i>	→	<b>real</b>
6.1.	<i>subprogram_declarations</i>	→	<i>subprogram_declaration</i> ; <i>subprogram_declarations</i>
6.2.	<i>subprogram_declarations</i>	→	ϵ
7.	<i>subprogram_declaration</i>	→	<i>subprogram_head</i> <i>declarations</i> <i>subprogram_declarations</i> <i>compound_statement</i>
8.	<i>subprogram_head</i>	→	<b>procedure</b> <i>id</i> <i>arguments</i> ;
9.1.	<i>arguments</i>	→	( <i>parameter_list</i> )
9.2.	<i>arguments</i>	→	ϵ
10.1.	<i>parameter_list</i>	→	<b>id</b> : <i>type</i> <i>parameter_list'</i>
10.2.1.	<i>parameter_list'</i>	→	; <b>id</b> : <i>type</i> <i>parameter_list'</i>
10.2.2.	<i>parameter_list'</i>	→	ϵ
11.	<i>compound_statement</i>	→	<b>begin</b> <i>optional_statements</i> <b>end</b>
12.1.	<i>optional_statements</i>	→	<i>statement_list</i>
12.2.	<i>optional_statements</i>	→	ϵ
13.1.	<i>statement_list</i>	→	<i>statement</i> <i>statement_list'</i>
13.2.1.	<i>statement_list'</i>	→	; <i>statement</i> <i>statement_list'</i>
13.2.2.	<i>statement_list'</i>	→	ϵ
14.1.	<i>statement</i>	→	<i>variable</i> <b>assignop</b> <i>expression</i>
14.2.	<i>statement</i>	→	<i>procedure_statement</i>
14.3.	<i>statement</i>	→	<i>compound_statement</i>
14.4.	<i>statement</i>	→	<b>while</b> <i>expression</i> <b>do</b> <i>statement</i>
14.5.	<i>statement</i>	→	<b>if</b> <i>expression</i> <b>then</b> <i>statement</i> <b>else'</b>
15.1.	<i>else'</i>	→	<i>else</i> <i>statement</i>
15.2.	<i>else'</i>	→	ϵ
16.	<i>variable</i>	→	<b>id</b> <i>array_access</i>
17.1.	<i>array_access</i>	→	[ <i>expression</i> ]
17.2.	<i>array_access</i>	→	ϵ
18.	<i>procedure_statement</i>	→	<b>call</b> <i>id</i> <i>optional_expressions</i>
19.1.	<i>optional_expressions</i>	→	( <i>expression_list</i> )
19.2.	<i>optional_expressions</i>	→	ϵ
20.1.	<i>expression_list</i>	→	<i>expression</i> <i>expression_list'</i>
20.2.1.	<i>expression_list'</i>	→	, <i>expression</i> <i>expression_list'</i>
20.2.2.	<i>expression_list'</i>	→	ϵ
21.	<i>expression</i>	→	<i>simple_expression</i> <i>related_expression</i>
22.1.	<i>related_expression</i>	→	<b>relop</b> <i>simple_expression</i>
22.2.	<i>related_expression</i>	→	ϵ
23.1.1.	<i>simple_expression</i>	→	<i>term</i> <i>simple_expression'</i>
23.1.2.	<i>simple_expression</i>	→	<i>sign</i> <i>term</i> <i>simple_expression'</i>
23.2.1.	<i>simple_expression'</i>	→	<b>addop</b> <i>term</i> <i>simple_expression'</i>
23.2.2.	<i>simple_expression'</i>	→	ϵ

24.1.	$term$	$\rightarrow$	$factor\ term'$
24.2.1.	$term'$	$\rightarrow$	<b>mulop</b> $factor\ term'$
24.2.2.	$term'$	$\rightarrow$	$\epsilon$
25.1.1.	$factor$	$\rightarrow$	<b>id</b> $factor'$
25.1.2.	$factor$	$\rightarrow$	<b>num</b>
25.1.3.	$factor$	$\rightarrow$	( $expression$ )
25.1.4.	$factor$	$\rightarrow$	<b>not</b> $factor$
25.2.1.	$factor'$	$\rightarrow$	[ $expression$ ]
25.2.1.	$factor'$	$\rightarrow$	$\epsilon$
26.1.	$sign$	$\rightarrow$	+
26.2.	$sign$	$\rightarrow$	-



ID	Name	First	Leads	Follows
1.	<i>program</i>	<b>program</b>		\$
2.1.	<i>identifier_list</i>	<b>id</b>		)
2.2.1.	<i>identifier_list'</i>	<b>,</b>		)
2.2.2.	<i>identifier_list'</i>	<b>ε</b>	→	)
3.1.	<i>declarations</i>	<b>var</b>		<b>procedure begin</b>
3.2.	<i>declarations</i>	<b>ε</b>	→	<b>procedure begin</b>
4.1.	<i>type</i>	<b>integer real</b>		; )
4.2.	<i>type</i>	<b>array</b>		; )
5.1.	<i>standard_type</i>	<b>integer</b>		; )
5.2.	<i>standard_type</i>	<b>real</b>		; )
6.1.	<i>subprogram_declarations</i>	<b>procedure</b>		<b>begin</b>
6.2.	<i>subprogram_declarations</i>	<b>ε</b>	→	<b>begin</b>
7.	<i>subprogram_declaration</i>	<b>procedure</b>		<b>;</b>
8.	<i>subprogram_head</i>	<b>procedure</b>		<b>var procedure begin</b>
9.1.	<i>arguments</i>	<b>(</b>		<b>;</b>
9.2.	<i>arguments</i>	<b>ε</b>	→	<b>;</b>
10.1.	<i>parameter_list</i>	<b>id</b>		)
10.2.1.	<i>parameter_list'</i>	<b>;</b>		)
10.2.2.	<i>parameter_list'</i>	<b>ε</b>	→	)
11.	<i>compound_statement</i>	<b>begin</b>		<b>; . end else</b>
12.1.	<i>optional_statements</i>	<b>id call begin while if</b>		<b>end</b>
12.2.	<i>optional_statements</i>	<b>ε</b>	→	<b>end</b>
13.1.	<i>statement_list</i>	<b>id call begin while if</b>		<b>end</b>
13.2.1.	<i>statement_list'</i>	<b>;</b>		<b>end</b>
13.2.2.	<i>statement_list'</i>	<b>ε</b>	→	<b>end</b>
14.1.	<i>statement</i>	<b>id</b>		<b>; end else</b>
14.2.	<i>statement</i>	<b>call</b>		<b>; end else</b>
14.3.	<i>statement</i>	<b>begin</b>		<b>; end else</b>
14.4.	<i>statement</i>	<b>while</b>		<b>; end else</b>
14.5.	<i>statement</i>	<b>if</b>		<b>; end else</b>
15.1.	<i>else'</i>	<b>else</b>		<b>; end else</b>
15.2.	<i>else'</i>	<b>ε</b>	→	<b>; end else</b>
16.	<i>variable</i>	<b>id</b>		<b>assignop</b>
17.1.	<i>array_access</i>	<b>[</b>		<b>assignop</b>
17.2.	<i>array_access</i>	<b>ε</b>	→	<b>assignop</b>
18.	<i>procedure_statement</i>	<b>call</b>		<b>; end else</b>
19.1.	<i>optional_expressions</i>	<b>(</b>		<b>; end else</b>
19.2.	<i>optional_expressions</i>	<b>ε</b>	→	<b>; end else</b>
20.1.	<i>expression_list</i>	<b>id num ( not + -</b>		<b>)</b>
20.2.1.	<i>expression_list'</i>	<b>,</b>		<b>)</b>
20.2.2.	<i>expression_list'</i>	<b>ε</b>	→	<b>)</b>
21.	<i>expression</i>	<b>id num ( not + -</b>		<b>; end else do then ] ) ,</b>
22.1.	<i>related_expression</i>	<b>relop</b>		<b>; end else do then ] ) ,</b>
22.2.	<i>related_expression</i>	<b>ε</b>	→	<b>; end else do then ] ) ,</b>
23.1.1.	<i>simple_expression</i>	<b>id num ( not</b>		<b>relop ; end else do then ] ) ,</b>
23.1.2.	<i>simple_expression</i>	<b>+ -</b>		<b>relop ; end else do then ] ) ,</b>
23.2.1.	<i>simple_expression'</i>	<b>addop</b>		<b>relop ; end else do then ] ) ,</b>
23.2.2.	<i>simple_expression'</i>	<b>ε</b>	→	<b>relop ; end else do then ] ) ,</b>
24.1.	<i>term</i>	<b>id num ( not</b>		<b>addop relop ; end else do then ] ) ,</b>
24.2.1.	<i>term'</i>	<b>mulop</b>		<b>addop relop ; end else do then ] ) ,</b>
24.2.2.	<i>term'</i>	<b>ε</b>	→	<b>addop relop ; end else do then ] ) ,</b>
25.1.1.	<i>factor</i>	<b>id</b>		<b>mulop addop relop ; end else do then ] ) ,</b>
25.1.2.	<i>factor</i>	<b>num</b>		<b>mulop addop relop ; end else do then ] ) ,</b>
25.1.3.	<i>factor</i>	<b>(</b>		<b>mulop addop relop ; end else do then ] ) ,</b>
25.1.4.	<i>factor</i>	<b>not</b>		<b>mulop addop relop ; end else do then ] ) ,</b>
25.2.1.	<i>factor'</i>	<b>[</b>		<b>mulop addop relop ; end else do then ] ) ,</b>
25.2.1.	<i>factor'</i>	<b>ε</b>	→	<b>mulop addop relop ; end else do then ] ) ,</b>
26.1.	<i>sign</i>	<b>+</b>		<b>id num not (</b>
26.2.	<i>sign</i>	<b>-</b>		<b>id num not (</b>

[illegible]

## 2 Implementation

Each production resides in its own file, where its first and synch sets are specified. The “program” production is called when the compiler begins, and it requests the next tokens. From there, productions are called in accordance to the grammar rules and the tokens received from the lexical analyzer.

The “match” function matched the token and set a global variable named “current\_tok” to the next token. If the match failed, a syntax error was printed to the listing file. Another operation called “require\_synch” was included, in the event that no token could identify the current production in a grammar variable. *require\_synch* would take in the firsts and synch sets of the grammar variables, print the appropriate error message (“received X; expected Y, Z”), then discard tokens until an item from the synch set were matched.

The productions mapped 1 : 1 from terminal symbols to match and from grammar variables to productions. Which production to use in which grammar variable was determined by the value of *current\_tok* (hence the *LL*(1) property of our grammar).

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.

## 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. I also noticed that the *require\_synch*() function would have been wonderful as a dynamically-scoped function.

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-Full

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  LEXERR: Trailing 0 in
       real: 1.20
17     9 else call fib (a - 01, c, b + c)
18  LEXERR: Leading 0 in
       int: 01
19     10 end;
20     11
21     12 procedure fib2(a : integer);
22     13 var b : real; var c : real; var sum : ;

```

```

23 SYNERR: Found ';;'; expected 'array', 'real', 'integer'
    instead.
24     14      var b : real;
25 SYNERR: Found 'real'; expected ';' instead.
26 SYNERR: Found ';;'; expected 'begin', 'procedure', 'var'
    ' instead.
27     15      procedure rawr3(b : real);
28     16          var q : real;
29     17          begin
30     18              q := b + 2.0;
31     19              call fib2(q).
32 SYNERR: Found '.'; expected 'end', ';' instead.
33     20          end;
34     21
35     22      begin
36     23          a := a - 1;
37     24          fib1(3.00);
38 SYNERR: Found '('; expected '[', ':=' instead.
39 LEXERR:                                     Trailing 0 in
    real:                                     3.00
40     25          sum := 1;
41     26          c := b;
42     27          while a > 0) do
43 SYNERR: Found ')'; expected 'do' instead.
44 SYNERR: Found 'do'; expected 'if', 'while', 'begin', '
    call', 'ID' instead.
45     28          call 3;
46     29          begin
47     30              a := a - 1;
48     31              b := sum;
49     32              sum := c + sum;
50     33              c := b
51     34          end;
52     35          fib2 := sum
53     36      end;
54     37
55     38      procedure init;
56     39          begin
57     40              n := 12;
58     41              if (1 and 2) or 3 then p := 12
59     42              else p := 14;
60     43              numsArray[3] := 15.560;
61 LEXERR:                                     Trailing 0 in
    real:                                     15.560
62     44          q := q[4];
63     45          q[4] := 12

```

```

64         46         end;
65         47
66         48         begin*
67 SYNERR: Found '*'; expected 'array', 'end', 'if', '
        while', 'begin', 'call', 'ID' instead.
68         49         call init;
69         50         call fib2;
70         51         call rawr3(34, 56);
71         52         end.
72         53
73         54 a
74 SYNERR: Found 'ID'; expected 'EOF' instead.

```

---

Listing 3: 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	0x7ffcd8e003d0	
22	1	(	14	0	
23	1	input	4	0x7ffcd8e006e0	
24	1	;	13	1	
25	1	;	17	0	
26	1	output	4	0x7ffcd8e00b60	
27	1	)	14	1	
28	1	:	10	0	
29	1	:	17	0	
30	2	var	11	0	
31	2	a	4	0x7ffcd8e01300	

```

32 2      :      10      0
33 2  int      4      0x7ffcd8e015f0
34 2  int      17      0
35 2      ;      13      1
36 2  var      11      0
37 2      p      4      0x7ffcd8e01b80
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      0x7ffcd8e027b0
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      0x7ffcd8e03410
53 3      integer  10      1
54 3      ;      13      1
55 4  var      11      0
56 4      q      4      0x7ffcd8e03c00
57 4      :      10      0
58 4  real      10      2
59 4      ;      13      1
60 6  procedure  5      6
61 6  fib1      4      0x7ffcd8e04640
62 6      (      14      0
63 6      aReallyLongInt  4      0x7ffcd8e04a50
64 6      aReallyLongInt  16      1
65 6      :      10      0
66 6      integer  10      1
67 6      ;      13      1
68 6      b      4      0x7ffcd8e051e0
69 6      :      10      0
70 6  real      10      2
71 6      ,      13      0
72 6      ,      17      0
73 6      c      4      0x7ffcd8e05940
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      0x7ffcd8e01300
81 8   <=      3      1
82 8   1.20    12     0
83 8   1.20    16     9
84 8   then    5      8
85 8   fib     4      0x7ffcd8e003d0
86 8   :=      2      0
87 8   c       4      0x7ffcd8e05940
88 9   else    5      2
89 9   call    5     10
90 9   fib     4      0x7ffcd8e003d0
91 9   (       14     0
92 9   a       4      0x7ffcd8e01300
93 9   -       6      1
94 9   01      12     0
95 9   01      16     7
96 9   ,       13     0
97 9   c       4      0x7ffcd8e05940
98 9   ,       13     0
99 9   b       4      0x7ffcd8e051e0
100 9   +       6      0
101 9   c       4      0x7ffcd8e05940
102 9   )      14     1
103 10 end      5      3
104 10 ;       13     1
105 12 procedure 5      6
106 12 fib2     4      0x7ffcd8c06d80
107 12 (       14     0
108 12 a       4      0x7ffcd8e01300
109 12 :      10     0
110 12 integer 10     1
111 12 )      14     1
112 12 ;      13     1
113 13 var     11     0
114 13 b       4      0x7ffcd8e051e0
115 13 :      10     0
116 13 real    10     2
117 13 ;      13     1
118 13 var     11     0
119 13 c       4      0x7ffcd8e05940
120 13 :      10     0
121 13 real    10     2
122 13 ;      13     1
123 13 var     11     0

```

124	13	sum	4	0x7ffcd8c08c00
125	13	:	10	0
126	13	;	13	1
127	13	;	17	0
128	14	var	11	0
129	14	b	4	0x7ffcd8e051e0
130	14	:	10	0
131	14	real	10	2
132	14	real	17	0
133	14	;	13	1
134	14	;	17	0
135	15	procedure	5	6
136	15	rawr3	4	0x7ffcd8e06f40
137	15	(	14	0
138	15	b	4	0x7ffcd8e051e0
139	15	:	10	0
140	15	real	10	2
141	15	)	14	1
142	15	;	13	1
143	16	var	11	0
144	16	q	4	0x7ffcd8e03c00
145	16	:	10	0
146	16	real	10	2
147	16	;	13	1
148	17	begin	5	0
149	18	q	4	0x7ffcd8e03c00
150	18	:=	2	0
151	18	b	4	0x7ffcd8e051e0
152	18	+	6	0
153	18	2.0	12	1
154	18	;	13	1
155	19	call	5	10
156	19	fib2	4	0x7ffcd8c06d80
157	19	(	14	0
158	19	q	4	0x7ffcd8e03c00
159	19	)	14	1
160	19	.	13	2
161	19	.	17	0
162	20	end	5	3
163	20	;	13	1
164	22	begin	5	0
165	23	a	4	0x7ffcd8e01300
166	23	:=	2	0
167	23	a	4	0x7ffcd8e01300
168	23	-	6	1
169	23	1	12	0

170	23	;	13	1
171	24	fib1	4	0x7ffcd8e04640
172	24	(	14	0
173	24	(	17	0
174	24	3.00	12	0
175	24	3.00	16	9
176	24	)	14	1
177	24	;	13	1
178	25	sum	4	0x7ffcd8c08c00
179	25	:=	2	0
180	25	1	12	0
181	25	;	13	1
182	26	c	4	0x7ffcd8e05940
183	26	:=	2	0
184	26	b	4	0x7ffcd8e051e0
185	26	;	13	1
186	27	while	5	9
187	27	a	4	0x7ffcd8e01300
188	27	>	3	3
189	27	0	12	0
190	27	)	14	1
191	27	)	17	0
192	27	do	5	1
193	27	do	17	0
194	28	call	5	10
195	28	3	12	0
196	28	;	13	1
197	29	begin	5	0
198	30	a	4	0x7ffcd8e01300
199	30	:=	2	0
200	30	a	4	0x7ffcd8e01300
201	30	-	6	1
202	30	1	12	0
203	30	;	13	1
204	31	b	4	0x7ffcd8e051e0
205	31	:=	2	0
206	31	sum	4	0x7ffcd8c08c00
207	31	;	13	1
208	32	sum	4	0x7ffcd8c08c00
209	32	:=	2	0
210	32	c	4	0x7ffcd8e05940
211	32	+	6	0
212	32	sum	4	0x7ffcd8c08c00
213	32	;	13	1
214	33	c	4	0x7ffcd8e05940
215	33	:=	2	0

```

216 33      b      4      0x7ffcd8e051e0
217 34      end    5      3
218 34      ;      13     1
219 35      fib2    4      0x7ffcd8c06d80
220 35      :=      2      0
221 35      sum     4      0x7ffcd8c08c00
222 36      end     5      3
223 36      ;      13     1
224 38      procedure 5      6
225 38      init    4      0x7ffcd8e14fb0
226 38      ;      13     1
227 39      begin   5      0
228 40      n       4      0x7ffcd8e15a60
229 40      :=      2      0
230 40      12      12     0
231 40      ;      13     1
232 41      if      5      5
233 41      (       14     0
234 41      1       12     0
235 41      and      7      2
236 41      2       12     0
237 41      )       14     1
238 41      or       6      2
239 41      3       12     0
240 41      then     5      8
241 41      p       4      0x7ffcd8e01b80
242 41      :=      2      0
243 41      12      12     0
244 42      else     5      2
245 42      p       4      0x7ffcd8e01b80
246 42      :=      2      0
247 42      14      12     0
248 42      ;      13     1
249 43      numsArray 4      0x7ffcd8e027b0
250 43      [       14     2
251 43      3       12     0
252 43      ]       14     3
253 43      :=      2      0
254 43 15.560      12     0
255 43 15.560      16     9
256 43      ;      13     1
257 44      q       4      0x7ffcd8e03c00
258 44      :=      2      0
259 44      q       4      0x7ffcd8e03c00
260 44      [       14     2
261 44      4       12     0

```

262	44	]	14	3
263	44	;	13	1
264	45	q	4	0x7ffcd8e03c00
265	45	[	14	2
266	45	4	12	0
267	45	]	14	3
268	45	:=	2	0
269	45	12	12	0
270	46	end	5	3
271	46	;	13	1
272	48	begin	5	0
273	48	*	7	0
274	48	*	17	0
275	49	call	5	10
276	49	init	4	0x7ffcd8e14fb0
277	49	;	13	1
278	50	call	5	10
279	50	fib2	4	0x7ffcd8c06d80
280	50	;	13	1
281	51	call	5	10
282	51	rawr3	4	0x7ffcd8e06f40
283	51	(	14	0
284	51	34	12	0
285	51	,	13	0
286	51	56	12	0
287	51	)	14	1
288	51	;	13	1
289	52	end	5	3
290	52	.	13	2
291	54	a	4	0x7ffcd8e01300
292	54	a	17	0
293	55	EOF	1	0

---

## 3.2 Just Syntax Errors

Listing 4: Just Syntax 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(anInt : integer; b : real, c : real);
7     begin
8       if a <= 1 then fib := c

```

```

9         else call fib (a - 1, c, b + c)
10     end;
11
12 procedure fib2(a : integer);
13     var b : real; var c : real; var sum : ;
14     var b : real;
15     procedure rawr3(b : real);
16         var q : real;
17         begin
18             q := b + 2.0;
19             call fib2(q).
20         end;
21
22     begin
23         a := a - 1;
24         fib1(3);
25         sum := 1;
26         c := b;
27         while a > 0) do
28             call 3;
29             begin
30                 a := a - 1;
31                 b := sum;
32                 sum := c + sum;
33                 c := b
34             end
35             fib2 := sum
36         end;
37
38 procedure init;
39     begin
40         n := 12;
41         if (1 and 2) or 3 then p := 12
42         else p := 14;
43         numsArray[3] := 15.56;
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.

```

---

Listing 5: Just Syntax 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(anInt : integer; b : real, c
      : real);
12 SYNERR: Found ',,'; expected ')', ',' instead.
13     7   begin
14     8       if a <= 1 then fib := c
15     9       else call fib (a - 1, c, b + c)
16    10   end;
17    11
18    12   procedure fib2(a : integer);
19    13       var b : real; var c : real; var sum : ;
20 SYNERR: Found ';;' expected 'array', 'real', 'integer'
      instead.
21    14       var b : real;
22 SYNERR: Found 'real'; expected ';' instead.
23 SYNERR: Found ';;' expected 'begin', 'procedure', 'var'
      ' instead.
24    15   procedure rawr3(b : real);
25    16       var q : real;
26    17   begin
27    18       q := b + 2.0;
28    19       call fib2(q).
29 SYNERR: Found '.;' expected 'end', ',' instead.
30    20   end;
31    21
32    22   begin
33    23       a := a - 1;
34    24       fib1(3);
35 SYNERR: Found '('; expected '[', ':=' instead.
36    25       sum := 1;
37    26       c := b;
38    27       while a > 0) do
39 SYNERR: Found ')'; expected 'do' instead.

```

```

40 SYNERR: Found 'do'; expected 'if', 'while', 'begin', '
    call', 'ID' instead.
41     28         call 3;
42     29         begin
43     30             a := a - 1;
44     31             b := sum;
45     32             sum := c + sum;
46     33             c := b
47     34         end
48     35         fib2 := sum
49 SYNERR: Found 'ID'; expected 'end', ';' instead.
50     36         end;
51     37
52     38     procedure init;
53     39         begin
54     40             n := 12;
55     41             if (1 and 2) or 3 then p := 12
56     42             else p := 14;
57     43             numsArray[3] := 15.56;
58     44             q := q[4];
59     45             q[4] := 12;
60     46         end;
61 SYNERR: Found 'end'; expected 'if', 'while', 'begin',
    'call', 'ID' instead.
62     47
63     48         begin
64     49             call init;
65     50             call fib2;
66     51             call rawr3(34, 56)
67     52         end.

```

---

Listing 6: Just Syntax 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      0x7ff69f5003d0
22      1      (      14      0
23      1 input      4      0x7ff69f5006e0
24      1      ;      13      1
25      1      ;      17      0
26      1 output      4      0x7ff69f500b60
27      1      )      14      1
28      1      :      10      0
29      1      :      17      0
30      2      var      11      0
31      2      a      4      0x7ff69f501300
32      2      :      10      0
33      2      int      4      0x7ff69f5015f0
34      2      int      17      0
35      2      ;      13      1
36      2      var      11      0
37      2      p      4      0x7ff69f501b80
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      0x7ff69f5027b0
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      0x7ff69f503410
53      3      integer      10      1
54      3      ;      13      1
55      4      var      11      0
56      4      q      4      0x7ff69f503c00
57      4      :      10      0
58      4      real      10      2
59      4      ;      13      1

```

60	6	procedure	5	6
61	6	fib1	4	0x7ff69f504640
62	6	(	14	0
63	6	anInt	4	0x7ff69f504930
64	6	:	10	0
65	6	integer	10	1
66	6	;	13	1
67	6	b	4	0x7ff69f504ff0
68	6	:	10	0
69	6	real	10	2
70	6	,	13	0
71	6	,	17	0
72	6	c	4	0x7ff69f505750
73	6	:	10	0
74	6	real	10	2
75	6	)	14	1
76	6	;	13	1
77	7	begin	5	0
78	8	if	5	5
79	8	a	4	0x7ff69f501300
80	8	<=	3	1
81	8	1	12	0
82	8	then	5	8
83	8	fib	4	0x7ff69f5003d0
84	8	:=	2	0
85	8	c	4	0x7ff69f505750
86	9	else	5	2
87	9	call	5	10
88	9	fib	4	0x7ff69f5003d0
89	9	(	14	0
90	9	a	4	0x7ff69f501300
91	9	-	6	1
92	9	1	12	0
93	9	,	13	0
94	9	c	4	0x7ff69f505750
95	9	,	13	0
96	9	b	4	0x7ff69f504ff0
97	9	+	6	0
98	9	c	4	0x7ff69f505750
99	9	)	14	1
100	10	end	5	3
101	10	;	13	1
102	12	procedure	5	6
103	12	fib2	4	0x7ff69f6037f0
104	12	(	14	0
105	12	a	4	0x7ff69f501300

106	12	:	10	0	
107	12	integer	10	1	
108	12	)	14	1	
109	12	;	13	1	
110	13	var	11	0	
111	13	b	4	0x7ff69f504ff0	
112	13	:	10	0	
113	13	real	10	2	
114	13	;	13	1	
115	13	var	11	0	
116	13	c	4	0x7ff69f505750	
117	13	:	10	0	
118	13	real	10	2	
119	13	;	13	1	
120	13	var	11	0	
121	13	sum	4	0x7ff69f605670	
122	13	:	10	0	
123	13	;	13	1	
124	13	;	17	0	
125	14	var	11	0	
126	14	b	4	0x7ff69f504ff0	
127	14	:	10	0	
128	14	real	10	2	
129	14	real	17	0	
130	14	;	13	1	
131	14	;	17	0	
132	15	procedure	5	6	
133	15	rawr3	4	0x7ff69f5076b0	
134	15	(	14	0	
135	15	b	4	0x7ff69f504ff0	
136	15	:	10	0	
137	15	real	10	2	
138	15	)	14	1	
139	15	;	13	1	
140	16	var	11	0	
141	16	q	4	0x7ff69f503c00	
142	16	:	10	0	
143	16	real	10	2	
144	16	;	13	1	
145	17	begin	5	0	
146	18	q	4	0x7ff69f503c00	
147	18	:=	2	0	
148	18	b	4	0x7ff69f504ff0	
149	18	+	6	0	
150	18	2.0	12	1	
151	18	;	13	1	

152	19	call	5	10	
153	19	fib2	4		0x7ff69f6037f0
154	19	(	14	0	
155	19	q	4		0x7ff69f503c00
156	19	)	14	1	
157	19	.	13	2	
158	19	.	17	0	
159	20	end	5	3	
160	20	;	13	1	
161	22	begin	5	0	
162	23	a	4		0x7ff69f501300
163	23	:=	2	0	
164	23	a	4		0x7ff69f501300
165	23	-	6	1	
166	23	1	12	0	
167	23	;	13	1	
168	24	fib1	4		0x7ff69f504640
169	24	(	14	0	
170	24	(	17	0	
171	24	3	12	0	
172	24	)	14	1	
173	24	;	13	1	
174	25	sum	4		0x7ff69f605670
175	25	:=	2	0	
176	25	1	12	0	
177	25	;	13	1	
178	26	c	4		0x7ff69f505750
179	26	:=	2	0	
180	26	b	4		0x7ff69f504ff0
181	26	;	13	1	
182	27	while	5	9	
183	27	a	4		0x7ff69f501300
184	27	>	3	3	
185	27	0	12	0	
186	27	)	14	1	
187	27	)	17	0	
188	27	do	5	1	
189	27	do	17	0	
190	28	call	5	10	
191	28	3	12	0	
192	28	;	13	1	
193	29	begin	5	0	
194	30	a	4		0x7ff69f501300
195	30	:=	2	0	
196	30	a	4		0x7ff69f501300
197	30	-	6	1	

```

198 30      1      12      0
199 30      ;      13      1
200 31      b      4      0x7ff69f504ff0
201 31      :=     2      0
202 31      sum     4      0x7ff69f605670
203 31      ;      13     1
204 32      sum     4      0x7ff69f605670
205 32      :=     2      0
206 32      c      4      0x7ff69f505750
207 32      +      6      0
208 32      sum     4      0x7ff69f605670
209 32      ;      13     1
210 33      c      4      0x7ff69f505750
211 33      :=     2      0
212 33      b      4      0x7ff69f504ff0
213 34      end     5      3
214 35      fib2    4      0x7ff69f6037f0
215 35      fib2    17     0
216 35      :=     2      0
217 35      sum     4      0x7ff69f605670
218 36      end     5      3
219 36      ;      13     1
220 38      procedure 5      6
221 38      init    4      0x7ff69f515340
222 38      ;      13     1
223 39      begin   5      0
224 40      n      4      0x7ff69f515df0
225 40      :=     2      0
226 40      12     12     0
227 40      ;      13     1
228 41      if     5      5
229 41      (      14     0
230 41      1      12     0
231 41      and     7      2
232 41      2      12     0
233 41      )      14     1
234 41      or     6      2
235 41      3      12     0
236 41      then    5      8
237 41      p      4      0x7ff69f501b80
238 41      :=     2      0
239 41      12     12     0
240 42      else    5      2
241 42      p      4      0x7ff69f501b80
242 42      :=     2      0
243 42      14     12     0

```

244	42	;	13	1	
245	43	numsArray	4	4	0x7ff69f5027b0
246	43	[	14	2	
247	43	3	12	0	
248	43	]	14	3	
249	43	:=	2	0	
250	43	15.56	12	1	
251	43	;	13	1	
252	44	q	4	4	0x7ff69f503c00
253	44	:=	2	0	
254	44	q	4	4	0x7ff69f503c00
255	44	[	14	2	
256	44	4	12	0	
257	44	]	14	3	
258	44	;	13	1	
259	45	q	4	4	0x7ff69f503c00
260	45	[	14	2	
261	45	4	12	0	
262	45	]	14	3	
263	45	:=	2	0	
264	45	12	12	0	
265	45	;	13	1	
266	46	end	5	3	
267	46	end	17	0	
268	46	;	13	1	
269	48	begin	5	0	
270	49	call	5	10	
271	49	init	4	4	0x7ff69f515340
272	49	;	13	1	
273	50	call	5	10	
274	50	fib2	4	4	0x7ff69f6037f0
275	50	;	13	1	
276	51	call	5	10	
277	51	rawr3	4	4	0x7ff69f5076b0
278	51	(	14	0	
279	51	34	12	0	
280	51	,	13	0	
281	51	56	12	0	
282	51	)	14	1	
283	52	end	5	3	
284	52	.	13	2	
285	53	EOF	1	0	

---

### 3.3 Error-Free

Listing 7: Error-Free 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(a : integer; b : real; c : real);
7     begin
8       if a <= 1 then fib := c
9       else call fib (a - 1, c, b + c)
10    end;
11
12  procedure fib2(a : integer);
13    var b : real; var c : real; var sum : integer;
14    var b : real;
15    procedure rawr3(b : real);
16      var q : real;
17      begin
18        q := b + 2.0;
19        call fib2(q)
20      end;
21
22    begin
23      a := a - 1;
24      b := 0;
25      sum := 1;
26      c := b;
27      while (a > 0) do
28        begin
29          a := a - 1;
30          b := sum;
31          sum := c + sum;
32          c := b
33        end;
34      fib2 := sum
35    end;
36
37  procedure init;
38    begin
39      n := 12;
40      if (1 and 2) or 3 then p := 12
41      else p := 14;
42      numsArray[3] := 15.56;
43      q := q[4];
44      q[4] := 12

```

```

45     end;
46
47     begin
48         call init;
49         call fib2;
50         call rawr3(34, 56)
51     end.

```

---

#### Listing 8: Error-Free 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(a : integer; b : real; c :
7      7       real);
8      8       begin
9      9         if a <= 1 then fib := c
10     10         else call fib (a - 1, c, b + c)
11     11     end;
12     12   procedure fib2(a : integer);
13     13       var b : real; var c : real; var sum :
14     14       integer;
15     15       var b : real;
16     16       procedure rawr3(b : real);
17     17         var q : real;
18     18         begin
19     19             q := b + 2.0;
20     20             call fib2(q)
21     21         end;
22     22   begin
23     23       a := a - 1;
24     24       b := 0;
25     25       sum := 1;
26     26       c := b;
27     27       while (a > 0) do
28     28         begin
29     29             a := a - 1;
30     30             b := sum;
31     31             sum := c + sum;
32     32             c := b
33     33         end;
34     34       fib2 := sum

```



```

35      35      end;
36      36
37      37      procedure init;
38      38      begin
39      39          n := 12;
40      40          if (1 and 2) or 3 then p := 12
41      41          else p := 14;
42      42          numsArray[3] := 15.56;
43      43          q := q[4];
44      44          q[4] := 12
45      45      end;
46      46
47      47      begin
48      48          call init;
49      49          call fib2;
50      50          call rawr3(34, 56)
51      51      end.

```

---

Listing 9: 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	1	program	5	7	
21	1	fib	4	0x7fab12d03700	
22	1	(	14	0	
23	1	input	4	0x7fab12d03a10	
24	1	,	13	0	
25	1	output	4	0x7fab12d03d90	
26	1	)	14	1	

27	1	;	13	1	
28	2	var	11	0	
29	2	a	4		0x7fab12d04460
30	2	:	10	0	
31	2	integer		10	1
32	2	;	13	1	
33	2	var	11	0	
34	2	p	4		0x7fab12d04c60
35	2	:	10	0	
36	2	integer		10	1
37	2	;	13	1	
38	3	var	11	0	
39	3	numsArray		4	0x7fab12d05680
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		0x7fab12d06ae0
52	4	:	10	0	
53	4	real	10	2	
54	4	;	13	1	
55	6	procedure		5	6
56	6	fib1	4		0x7fab12d07550
57	6	(	14	0	
58	6	a	4		0x7fab12d04460
59	6	:	10	0	
60	6	integer		10	1
61	6	;	13	1	
62	6	b	4		0x7fab12d07e60
63	6	:	10	0	
64	6	real	10	2	
65	6	;	13	1	
66	6	c	4		0x7fab12d084c0
67	6	:	10	0	
68	6	real	10	2	
69	6	)	14	1	
70	6	;	13	1	
71	7	begin	5	0	
72	8	if	5	5	

```

73      8      a      4      0x7fab12d04460
74      8      <=     3      1
75      8      1      12     0
76      8      then   5      8
77      8      fib    4      0x7fab12d03700
78      8      :=     2      0
79      8      c      4      0x7fab12d084c0
80      9      else   5      2
81      9      call   5      10
82      9      fib    4      0x7fab12d03700
83      9      (      14     0
84      9      a      4      0x7fab12d04460
85      9      -      6      1
86      9      1      12     0
87      9      ,      13     0
88      9      c      4      0x7fab12d084c0
89      9      ,      13     0
90      9      b      4      0x7fab12d07e60
91      9      +      6      0
92      9      c      4      0x7fab12d084c0
93      9      )      14     1
94     10      end     5      3
95     10      ;      13     1
96     12      procedure 5      6
97     12      fib2    4      0x7fab12d0c840
98     12      (      14     0
99     12      a      4      0x7fab12d04460
100    12      :      10     0
101    12      integer 10      1
102    12      )      14     1
103    12      ;      13     1
104    13      var     11     0
105    13      b      4      0x7fab12d07e60
106    13      :      10     0
107    13      real    10     2
108    13      ;      13     1
109    13      var     11     0
110    13      c      4      0x7fab12d084c0
111    13      :      10     0
112    13      real    10     2
113    13      ;      13     1
114    13      var     11     0
115    13      sum      4      0x7fab12d0e6c0
116    13      :      10     0
117    13      integer 10      1
118    13      ;      13     1

```

```

119 14 var 11 0
120 14 b 4 0x7fab12d07e60
121 14 : 10 0
122 14 real 10 2
123 14 ; 13 1
124 15 procedure 5 6
125 15 rawr3 4 0x7fab12d0fe30
126 15 ( 14 0
127 15 b 4 0x7fab12d07e60
128 15 : 10 0
129 15 real 10 2
130 15 ) 14 1
131 15 ; 13 1
132 16 var 11 0
133 16 q 4 0x7fab12d06ae0
134 16 : 10 0
135 16 real 10 2
136 16 ; 13 1
137 17 begin 5 0
138 18 q 4 0x7fab12d06ae0
139 18 := 2 0
140 18 b 4 0x7fab12d07e60
141 18 + 6 0
142 18 2.0 12 1
143 18 ; 13 1
144 19 call 5 10
145 19 fib2 4 0x7fab12d0c840
146 19 ( 14 0
147 19 q 4 0x7fab12d06ae0
148 19 ) 14 1
149 20 end 5 3
150 20 ; 13 1
151 22 begin 5 0
152 23 a 4 0x7fab12d04460
153 23 := 2 0
154 23 a 4 0x7fab12d04460
155 23 - 6 1
156 23 1 12 0
157 23 ; 13 1
158 24 b 4 0x7fab12d07e60
159 24 := 2 0
160 24 0 12 0
161 24 ; 13 1
162 25 sum 4 0x7fab12d0e6c0
163 25 := 2 0
164 25 1 12 0

```

```

165 25      ;      13      1
166 26      c      4      0x7fab12d084c0
167 26      :=     2      0
168 26      b      4      0x7fab12d07e60
169 26      ;      13      1
170 27 while      5      9
171 27      (      14      0
172 27      a      4      0x7fab12d04460
173 27      >      3      3
174 27      0      12      0
175 27      )      14      1
176 27      do      5      1
177 28 begin      5      0
178 29      a      4      0x7fab12d04460
179 29      :=     2      0
180 29      a      4      0x7fab12d04460
181 29      -      6      1
182 29      1      12      0
183 29      ;      13      1
184 30      b      4      0x7fab12d07e60
185 30      :=     2      0
186 30      sum     4      0x7fab12d0e6c0
187 30      ;      13      1
188 31      sum     4      0x7fab12d0e6c0
189 31      :=     2      0
190 31      c      4      0x7fab12d084c0
191 31      +      6      0
192 31      sum     4      0x7fab12d0e6c0
193 31      ;      13      1
194 32      c      4      0x7fab12d084c0
195 32      :=     2      0
196 32      b      4      0x7fab12d07e60
197 33 end        5      3
198 33      ;      13      1
199 34 fib2       4      0x7fab12d0c840
200 34      :=     2      0
201 34      sum     4      0x7fab12d0e6c0
202 35 end        5      3
203 35      ;      13      1
204 37      procedure 5      6
205 37      init      4      0x7fab12d1ce40
206 37      ;      13      1
207 38 begin      5      0
208 39      n      4      0x7fab12d1d8f0
209 39      :=     2      0
210 39      12      12      0

```

211	39	;	13	1	
212	40	if	5	5	
213	40	(	14	0	
214	40	1	12	0	
215	40	and	7	2	
216	40	2	12	0	
217	40	)	14	1	
218	40	or	6	2	
219	40	3	12	0	
220	40	then	5	8	
221	40	p	4		0x7fab12d04c60
222	40	:=	2	0	
223	40	12	12	0	
224	41	else	5	2	
225	41	p	4		0x7fab12d04c60
226	41	:=	2	0	
227	41	14	12	0	
228	41	;	13	1	
229	42	numsArray	4		0x7fab12d05680
230	42	[	14	2	
231	42	3	12	0	
232	42	]	14	3	
233	42	:=	2	0	
234	42	15.56	12	1	
235	42	;	13	1	
236	43	q	4		0x7fab12d06ae0
237	43	:=	2	0	
238	43	q	4		0x7fab12d06ae0
239	43	[	14	2	
240	43	4	12	0	
241	43	]	14	3	
242	43	;	13	1	
243	44	q	4		0x7fab12d06ae0
244	44	[	14	2	
245	44	4	12	0	
246	44	]	14	3	
247	44	:=	2	0	
248	44	12	12	0	
249	45	end	5	3	
250	45	;	13	1	
251	47	begin	5	0	
252	48	call	5	10	
253	48	init	4		0x7fab12d1ce40
254	48	;	13	1	
255	49	call	5	10	
256	49	fib2	4		0x7fab12d0c840

257	49	;	13	1
258	50	call	5	10
259	50	rawr3	4	0x7fab12d0fe30
260	50	(	14	0
261	50	34	12	0
262	50	,	13	0
263	50	56	12	0
264	50	)	14	1
265	51	end	5	3
266	51	.	13	2
267	52	EOF	1	0

---

## Appendix 2: Program Listings

Listing 10: 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 11: 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
14                        right to green node
15    };
16
17    struct tree_node* left;
18    struct tree_node* right;
19    struct tree_node* parent;
20 } tree_node;

```



```

21 typedef struct LinkedTree {
22     struct node* head;
23 } DeclarationsTree;
24
25 // Green nodes designate
26     scopes, and blue nodes
27     designate variables
28 bool check_add_node(Token*
29     decl);
30 tree_node* get_last_green_node
31     ();
32 tree_node*
33     start_param_matching(Token*
34     id);
35 void reached_end_of_scope();
36 LangType get_type(Token* id);
37
38 #endif // DECLARATIONS_TREE_H

```

Listing 12: 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
16     the linked list
17 int add(LinkedList* list, void
18     * data, int size);
19
20 // Pop an item from the front
21     of the linked list
22 void* pop(LinkedList* list);
23
24 #endif // LINKED_H_

```

Listing 13: errorHandler.h

```

1 #ifndef ERROR_HANDLER_H
2 #define ERROR_HANDLER_H
3 #include "../tokenizer/tokens.
4     h"
5 extern const char* lexErrs[];
6 char* synErr;
7 char* semErr;
8
9 void throw_sem_error(char* msg
10     );
11 void throw_syn_error(Token*
12     received, const Token**
13     expected, int exp_size);
14 void throw_lex_error(enum
15     TokenType attribute, int
16     aspect, int start, int
17     length);
18 int initializeErrorHandler();
19
20 Token* getNextErrorToken();
21
22 #endif // ERROR_HANDLER_H

```

Listing 14: globals.h

```

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

```

Listing 15: handler.h

```

1 #ifndef HANDLER_H
2 #define HANDLER_H
3
4 #include<stdbool.h>

```

```

5 #include "../tokenizer/tokens. 9 #include "../errorHandler/
   h"                               errorHandler.h"
6
7 int initializeHandler(const    10
   char* sourcePath, const      11 extern Token* current_tok;
   char* resPath,               12
8 const char* listingPath, const 13 // All of these must have
   char* tokenPath,             their follows added to the
9 const char* memPath);         sync set
10 bool handleToken(Token* token) 14 void program();
   ;                             15 void id_list();
11 void outputWidth(char* lex,    16 void id_list_tail();
   int width);                  17 void declarations();
12                                18 LangType type();
13 #endif // HANDLER_H           19 LangType standard_type();
                                20 void subprogram_declarations()

```

---

Listing 16: 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 17: 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
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

```

<pre> 43 LangType term_tail(); 44 LangType factor(); 45 LangType factor_tail(); 46 void sign(); 47 48 #endif // voidS_H </pre>	<pre> 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 </pre>
<hr/> <div style="display: flex; justify-content: space-around;"> <div style="text-align: left; width: 45%;"> <p>Listing 18: symbolTable.h</p> <pre> 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 </pre> </div> <div style="text-align: left; width: 45%;"> <p>Listing 20: tokenizer.h</p> <pre> 1 #ifndef PROCESSOR_H_ 2 #define PROCESSOR_H_ 3 #include&lt;stdio.h&gt; 4 #include "tokens.h" 5 6 Token* getNextToken(); 7 int initializeTokens(FILE*     resFile); 8 9 #endif // PROCESSOR_H_ </pre> </div> </div> <hr/>	
<pre> 1 #ifndef MACHINES_H 2 #define MACHINES_H 3 #include &lt;stdio.h&gt; 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); </pre>	<pre> 1 #ifndef TOKENS_H 2 #define TOKENS_H 3 4 #include&lt;stdbool.h&gt; 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, PGNAME,     PPNAME, 10 PROC, AINT, AREAL} LangType; 11 12 enum TokenType {NOOP, FILEEND,     ASSIGNOP, RELOP, ID, 13 CONTROL, ADDOP, MULOP, WS,     ARRAY, TYPE, </pre>
<hr/> <div style="display: flex; justify-content: space-around;"> <div style="text-align: left; width: 45%;"> <p>Listing 19: machines.h</p> </div> <div style="text-align: left; width: 45%;"> <p>Listing 21: tokens.h</p> </div> </div> <hr/>	

```

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

```

<pre> 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 </pre>	<pre> 19 } 20 21 static int get_width(Token*     val) { 22     switch (val -&gt; type) { 23     case INT: return 4; 24     case REAL: return 8; 25 26     case AINT: return 4*(val -&gt;         array_length); 27     case AREAL: return 8*(val -&gt;         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 -&gt; lex) 38             return true; 39 40         if (!green &amp;&amp; (current_node -&gt;             type == PROC                current_node -&gt; type ==             PGNAME)) 41             return false; 42 43         // We've passed the most             recent green node, and this             is a blue one 44         if (!green &amp;&amp; (current_node -&gt;             type == PROC                current_node -&gt; type ==             PGNAME)) 45             break; 46 47         current_node = current_node -&gt;             parent; 48     } 49 50     return false; </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 22: declarationsTree.c

<pre> 1 #include &lt;stdlib.h&gt; 2 #include &lt;stdio.h&gt; 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)); </pre>	<pre> 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 -&gt; lex) 38             return true; 39 40         if (!green &amp;&amp; (current_node -&gt;             type == PROC                current_node -&gt; type ==             PGNAME)) 41             return false; 42 43         // We've passed the most             recent green node, and this             is a blue one 44         if (!green &amp;&amp; (current_node -&gt;             type == PROC                current_node -&gt; type ==             PGNAME)) 45             break; 46 47         current_node = current_node -&gt;             parent; 48     } 49 50     return false; </pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

51 }
52
53 static bool
54     check_add_green_node(Token* decl) {
55     if (d_tree == NULL)
56         initialize_d_tree();
57     if (bottom_node == NULL)
58     {
59         tree_node* addition = malloc(
60             sizeof(*addition));
61         addition -> lex = decl -> id;
62         addition -> type = decl ->
63             type;
64         addition -> add_right = false;
65         // Add it to the top of the
66             stack
67         add(green_node_stack, &
68             addition, sizeof(&addition)
69             );
70         addition -> left = NULL;
71         addition -> right = NULL;
72         addition -> parent = NULL;
73         bottom_node = addition;
74         return true;
75     }
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
84     // It hasn't been declared;
85         create it
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
93     // Add it to the top of the
94         stack
95     add(green_node_stack, &
96         addition, sizeof(&addition)
97         );
98     addition -> left = NULL;
99     addition -> right = NULL;
100     addition -> parent = NULL;
101     bottom_node = addition;
102     return true;
103 }
104
105 static bool
106     check_add_blue_node(Token* decl) {
107     // If there's no scope, that's
108         an error!
109     if (d_tree == NULL)
110         return false;
111
112     // It's been declared in the
113         scope already
114     if (check_node(decl -> id,
115         false))
116         return false;
117
118     // It hasn't been declared;
119         create it
120     tree_node* addition = malloc(
121         sizeof(*addition));
122     addition -> lex = decl -> id;

```

```

117 //printf("%s\n", addition -> lex);
118 addition -> type = decl -> type;
119 addition -> param = decl -> param;
120
121 if (!addition -> param)
122 {
123     outputWidth(addition -> lex,
124                 offset);
125     offset += get_width(decl);
126 }
127 addition -> left = NULL;
128 addition -> right = NULL;
129 addition -> parent =
130     bottom_node;
131 bottom_node -> left = addition
132     ;
133 bottom_node = addition;
134 //printf("(%s, %s)\n",
135     bottom_node -> lex,
136     bottom_node -> parent ->
137     lex);
138 return true;
139 }
140
141 tree_node*
142     start_param_matching(Token*
143     id) {
144     tree_node* current_node =
145         bottom_node;
146     while (current_node != NULL)
147     {
148         if (current_node -> type ==
149             PROC && current_node -> lex
150             == id -> id)
151         {
152             return current_node;
153         }
154         current_node = current_node ->
155             parent;
156     }
157     return NULL;
158 }
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
181     default: if (!
182         check_add_blue_node(decl))
183     {
184         errorMessage = calloc(100,
185             sizeof(*errorMessage));
186         sprintf(errorMessage,
187             "A variable named '%.*s' is
188             already defined in the
189             local scope!",
190             decl -> length, &BUFFER[decl
191             -> start]);
192         throw_sem_error(errorMessage);
193         return false;
194     }
195     return true;
196 }
197
198 void reached_end_of_scope() {
199     bottom_node = (*(tree_node**)
200         pop(green_node_stack));
201     bottom_node -> add_right =
202         true;
203 }

```

179 }	17 return list -> size;
180	18 }
181	19
182 LangType get_type(Token* id) {	20 void* pop(LinkedList* list)
183 if (id == NULL)	21 {
184 return ERR;	22 struct node* head = list ->
185	head;
186 tree_node* current_node =	23 struct node* next = head ->
bottom_node;	next;
187 while (current_node != NULL)	24
188 {	25 void* data = head -> data;
189 if (current_node -> lex == id	26 list -> head = next;
-> id)	27 list -> size--;
190 return current_node -> type;	28
191	29 <i>//free(head); // TODO this is</i>
192 current_node = current_node ->	<i>necessary; should fix</i>
parent;	30 return data;
193 }	31 }
194	
195 return NULL;	
196 }	

---

Listing 23: linkedlist.c

---

1 #include<stdlib.h>	1 #include<string.h>
2 #include "linkedlist.h"	2 #include<stdlib.h>
3	3
4	4
5 int add(LinkedList* list, void	5 #include "errorHandler.h"
*data, int size)	6
6 {	7 static LinkedList* errorList;
7 struct node* addition = malloc	8
(sizeof(*addition));	9 const char* lexErrs[] = {"
8 addition -> data = malloc(size	Unrecognized symbol:",
);	10 "ID length exceeded 10
9 addition -> next = (list ->	characters:",
head);	11 "Int length exceeded 10
10 <i>// Do a byte-by-byte copy of</i>	characters:",
<i>the data</i>	12 "Integer part of real exceeded
11 for (int i = 0; i < size; i++)	5 characters:",
12 *(char *) (addition -> data +	13 "Fractional part of real
i) = *(char *) (data + i);	exceeded 5 characters:",
13 list -> size++;	14 "Exponent part of long real
14	exceeded 2 characters:",
15 list -> head = addition;	15 "Missing exponent part of long
16	real:",
	16 "Leading 0 in int:",
	17 "Excessive leading 0 in real:"
	,

---

Listing 24: errorHandler.c

---



```

18  "Trailing 0 in real:",
19  "Leading 0 in exponent:",
20  "Attempt to use real exponent:
    ";
21
22  char* synErr;
23  char* semErr;
24
25
26
27  int initializeErrorHandler()
28  {
29      errorList = malloc(sizeof(*
          errorList));
30      return errorList != NULL;
31  }
32
33  void throw_syn_error(Token*
          received, const Token**
          expected, int exp_size)
34  {
35      // Generate token
36      Token* errToken = malloc(
          sizeof(*errToken));
37      errToken -> attribute = SYNERR
          ;
38      errToken -> aspect = 0;
39      errToken -> start = received
          -> start;
40      errToken -> length = received
          -> length;
41
42      add(errorList, errToken,
          sizeof(*errToken));
43
44      // Generate error message
45      // Calculate space needed
46      int size = strlen("Found '';
          expected ");
47      size += strlen(getLexFromToken
          (received, true));
48      for (int i = exp_size - 1; i
          >= 0; i--) {
49          size += strlen("''");
50          size += strlen(getLexFromToken
          (expected[i], expected[i]
          -> start));
          -> start));
51      if (i > 0)
52          size += strlen(", ");
53      }
54      size += strlen(" instead.");
55      size += 1; // Null terminator
56
57      synErr = malloc(sizeof(*synErr
          ) * size);
58      synErr[size - 1] = '\0';
59      strcpy(synErr, "Found '");
60      int current = 7;
61      int len = strlen(
          getLexFromToken(received,
          true));
62      strcpy(&synErr[current],
          getLexFromToken(received,
          true));
63      current += len;
64      strcpy(&synErr[current], "'';
          expected ");
65      current += 12;
66      for (int i = exp_size - 1; i
          >= 0; i--) {
67          strcpy(&synErr[current], "'");
68          current += 1;
69          len = strlen(getLexFromToken(
          expected[i], expected[i] ->
          start));
70          strcpy(&synErr[current],
          getLexFromToken(expected[i]
          , expected[i] -> start));
71          current += len;
72          strcpy(&synErr[current], "'");
73          current += 1;
74          if (i > 0) {
75              strcpy(&synErr[current], ", ")
              ;
76              current += 2;
77          }
78      }
79      strcpy(&synErr[current], "
          instead.");
80  }
81
82  void throw_sem_error(char* msg

```

```

    ) {
83 // Generate error token
84 Token* errToken = malloc(
    sizeof(*errToken));
85 errToken -> attribute = SEMERR
    ;
86 errToken -> aspect = 0;
87 errToken -> start = 0;
88 errToken -> length = 0;
89
90 add(errorList, errToken,
    sizeof(*errToken));
91
92 // Set the msg
93 semErr = msg;
94 }
95
96 void throw_lex_error(enum
    TokenType attribute, int
    aspect, int start, int
    length)
97 {
98 Token* errToken = malloc(
    sizeof(*errToken));
99 errToken -> attribute =
    attribute;
100 errToken -> aspect = aspect;
101 errToken -> start = start;
102 errToken -> length = length;
103
104 add(errorList, errToken,
    sizeof(*errToken));
105 }
106
107 Token* getNextErrorToken()
108 {
109 if (errorList -> size > 0)
110 return (Token *) pop(errorList
    );
111
112 return NULL;
113 }

```

---

Listing 25: globals.c

```

1 #include<string.h>

```

```

2 #include<stdlib.h>
3 #include<stdbool.h>
4 #include<stdio.h>
5 #include "globals.h"
6
7 char* BUFFER;
8 int LINE = 0;
9 int START = 0;
10
11 int initializeGlobals()
12 {
13 BUFFER = malloc(sizeof(char*)
    *73);
14 return (BUFFER != NULL);
15 }
16
17 void updateLine(char* line)
18 {
19 START = 0;
20 LINE++;
21 strcpy(BUFFER, line);
22 }

```

---

Listing 26: handler.c

```

1 #include<stdio.h>
2
3 #include "handler.h"
4 #include "../globals/globals.h"
5
6 #include "../tokenizer/
    tokenizer.h"
7
8 #include "../errorHandler/
    errorHandler.h"
9
10 static FILE* listingFile;
11 static FILE* tokenFile;
12 static FILE* sourceFile;
13 static FILE* memFile;
14
15 static const int
    TokenLineSpace = 10;
16 static const int
    TokenTypeSpace = 20;
17 static const int
    TokenAttrSpace = 20;

```

```

16 static const intTokenLexSpace      listingPath, "w+")) == NULL
    = 20;                            ||
17                                     46 (tokenFile = fopen(tokenPath,
18 static const int                    "w+")) == NULL ||
    ListingLineSpace = 7;            47 (memFile = fopen(memPath, "w+"
19 static const int                    )) == NULL)
    ListingErrSpace = 50;            48 return 0;
20 static const int                    49
    ListingLexSpace = 20;            50 for (size_t i = FILEEND; i <=
21                                     SEMERR; i++) {
22 static const int MemNameSpace      51 fprintf(tokenFile, "%-5zu%s\n"
    = 10;                            , i, catNames[i]);
23 static const int MemValSpace =     52 }
    20;                               53
24                                     54
25 void writeEOFToken()               55 char line[72];
26 {                                  56 if (fgets(line, sizeof(line),
27 fprintf(tokenFile, "%*d%*.s%*      sourceFile) != NULL)
    d%*d\n", TokenLineSpace,        57 {
    LINE, TokenLexSpace,            58 updateLine(line);
28 3, "EOF", TokenTypeSpace,         59 fprintf(listingFile, "%*d\t%s"
    FILEEND, TokenAttrSpace, 0)     , ListingLineSpace, LINE,
    ;                                line);
29 }                                  60 } else {
30                                     61 writeEOFToken();
31 int initializeHandler(const         62 }
    char* sourcePath, const         63
    char* resPath,                  64 fprintf(tokenFile, "%*s%*s%*s
32 const char* listingPath, const    %*s\n", TokenLineSpace, "
    char* tokenPath,                Line",
33 const char* memPath)              65 TokenLexSpace, "Lexeme",
34 {                                  66 TokenAttrSpace, "Token
35 if ((sourceFile = fopen(           Attribute",
    sourcePath, "r")) == NULL)      67 TokenTypeSpace, "Token Type");
36 {                                  68
37 fprintf(stderr, "%s\n", "          69 fprintf(memFile, "%*s%*s\n",
    Source was null?");              MemNameSpace, "ID",
38 return 0;                          70 MemValSpace, "Memory Offset");
39 }                                  71 return 1;
40                                     72 }
41 FILE* resFile = fopen(resPath,     73
    "r");                            74 void outputWidth(char* lex,
42 initializeTokens(resFile);          int width) {
43 fclose(resFile);                   75 fprintf(memFile, "%*s%*d\n",
44                                     MemNameSpace, lex,
45 if ((listingFile = fopen(           MemValSpace, width);

```

```

76 } NOOP)
77
78 void writeError(Token* 101 return;
    description) 102
79 { 103 if (token -> attribute >=
    { LEXERR)
80 fprintf(tokenFile, "%d%*.s%* 104 {
    d%d\n", TokenLineSpace, 105 writeError(token);
    LINE, 106 return;
81 TokenLexSpace, description -> 107 }
    length, &BUFFER[description 108
    -> start], 109
82 TokenTypeSpace, description -> 110 fprintf(tokenFile, "%d%*.s%*
    attribute, TokenAttrSpace, d", TokenLineSpace, LINE,
83 description -> aspect); TokenLexSpace,
84 if (description -> attribute 111 token -> length, &BUFFER[token
    == LEXERR) -> start], TokenTypeSpace,
85 fprintf(listingFile, "%s:%s 112 token -> attribute);
    %*.s\n", ListingLineSpace 113 switch (token -> attribute) {
    - 1, 114 case ID:
86 catNames[description -> 115 fprintf(tokenFile, "%*p",
    attribute], ListingErrSpace TokenTypeAttrSpace, token -> id
    , );
87 lexErrs[description -> aspect 116 break;
    ], ListingLexSpace, 117
88 description -> length, &BUFFER 118 default:
    [description -> start]); 119 fprintf(tokenFile, "%d",
89 else if (description -> TokenTypeAttrSpace, token ->
    attribute == SYNERR) aspect);
90 fprintf(listingFile, "%s: %s\ 120 break;
    n", ListingLineSpace - 1, 121 }
91 catNames[description -> 122 fprintf(tokenFile, "\n");
    attribute], synErr); 123 }
92 else if (description -> 124
    attribute == SEMERR) 125 bool handleToken(Token* token)
93 fprintf(listingFile, "%s: %s\ 126 {
    n", ListingLineSpace - 1, 127 writeToken(token);
94 catNames[description -> 128 if (token -> attribute == WS
    attribute], semErr); && token -> aspect == 1) //
95 } A newline
96
97 void writeToken(Token* token) 129 {
98 { 130 char line[72];
99 // Don't bother including in 131 if (fgets(line, sizeof(line),
    the output file. sourceFile) != NULL)
100 if (token -> attribute == WS 132 {
    || token -> attribute == 133 updateLine(line);
    134 fprintf(listingFile, "%d\t%s"

```

```

        , ListingLineSpace, LINE,
        line);
135 } else { // Error or end of
        file (assume the latter)
136 LINE++;
137 writeEOFToken();
138 return false; // Terminate
139 }
140 }
141 return true; // Continue
142 }

```

---

Listing 27: parser.c

---

```

1 #include<stdlib.h>
2 #include<stdbool.h>
3
4 #include "../tokenizer/tokens.
    h"
5 #include "productions/
    productions.h"
6 #include "../tokenizer/
    tokenizer.h"
7 #include "../handler/handler.h
    "
8 #include "../errorHandler/
    errorHandler.h"
9
10 Token* current_tok = NULL;
11
12 static bool sequence_running =
    true;
13
14 Token* get_next_relevant_token
    ()
15 {
16     const Token* next = malloc(
        sizeof(*next));
17     if (sequence_running)
18     {
19         do {
20             next = getNextToken();
21             if (!handleToken(next))
22             {
23                 sequence_running = false;
24                 next = &eof_tok;
25                 break;
26             }
27             } while (next -> attribute ==
                WS || next -> attribute ==
                NOOP
28                 || next -> attribute >= LEXERR
                );
29         } else {
30             next = &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 < size; i++)
43             if (tokens_equal(sync_set[i],
                current_tok, sync_set[i] ->
                start))
44                 return;
45
46         current_tok =
            get_next_relevant_token();
47     }
48 }
49
50 // Attempts to match the
    source token with the
    current token;
51 // if it is found, it returns
    the matched token (for use
    in the RDP).
52 // If it is not found, then
    match returns null.
53 Token* match(const Token*
    source, bool strict)
54 {
55     if (tokens_equal(source,

```

```

        current_tok, strict))
56 {
57 Token* prev_tok = current_tok;
58 current_tok =
    get_next_relevant_token();
59 return prev_tok;
60 }
61 else
62 {
63 throw_syn_error(current_tok, &
    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(&eof_tok, false);
75 }

```

Listing 28: arguments.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};
9 static const int first_size =
    sizeof(first_set)/sizeof(
    first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &semic_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 arguments()
21 {
22 // Production 9.1
23 if (tokens_equal(&lparen_tok,
    current_tok, true))
24 {
25 match(&lparen_tok, true);
26 parameter_list();
27 match(&rparen_tok, true);
28 return;
29
30 // Production 9.2
31 } else if (tokens_equal(&
    semic_tok, current_tok,
    true))
32 return; // Epsilon
33
34 synch();
35 }

```

Listing 29: array\_access.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
    [] = {&assignop_tok, &
    lbrac_tok};
9 static const int first_size =

```

```

        sizeof(first_set)/sizeof(
        first_set[0]));
10
11 static const Token* sync_set[]
    = {&eof_tok, &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 static LangType array_compare(
    LangType a_vals, LangType
    e_type) {
20 if ((a_vals == INT || a_vals
    == REAL) && e_type == INT)
21     return a_vals;
22 if (a_vals != ERR)
23 {
24     char* errorMessage = calloc
        (100, sizeof(*errorMessage)
        );
25     sprintf(errorMessage, "Attempt
        to index variable of type
        %s!", typeNames[a_vals]);
26     throw_sem_error(errorMessage);
27 }
28
29     return ERR;
30 }
31
32 // Needs implementing: None
33 LangType array_access(LangType
    id_type)
34 {
35     // Production 17.1
36     if (tokens_equal(&lbrac_tok,
        current_tok, true))
37     {
38         match(&lbrac_tok, true);
39         LangType e_type = expression()
            ;
40         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 30: 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
21 // Needs implementing: None
22 void compound_statement()
23 {
24 // Production 11
25 if (tokens_equal(&begin_tok,
26                 current_tok, true))
27 {
28 match(&begin_tok, true);
29 optional_statements();
30 match(&end_tok, true);
31 return;
32 }
33 synch();
34 }

```

---

Listing 31: 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 } else if (tokens_equal(&
   begin_tok, current_tok,
   true)
43 || tokens_equal(&procedure_tok,
   current_tok, true))
44 return; // epsilon
45
46 synch();
47 }

```

---

Listing 32: else<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
  [] = {&else_tok, &semic_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, &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 else_tail()
21 {
22   // Production 15.1
23   if (tokens_equal(&else_tok,
  current_tok, true)) // else
24   {
25     match(&else_tok, true);
26     statement();
27     return;
28
29     // Production 15.2
30   } else if (tokens_equal(&
  end_tok, current_tok, true)
  // end
31   || tokens_equal(&semic_tok,
  current_tok, true)) // ;
32   return; // epsilon
33
34   synch();
35 }

```

Listing 33: 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, &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 34: *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 {
23     // Production 20.1
24     if (tokens_equal(&lparen_tok,
        current_tok, true)
25         || tokens_equal(&addop_tok,
        current_tok, false) // + OR
        -
26         || tokens_equal(&id_tok,
        current_tok, false) // ID
27         || tokens_equal(&not_tok,
        current_tok, true)
28         || tokens_equal(&num_tok,
        current_tok, false)) // num
29     {
30         char* errorMessage;
31         if (to_match == NULL &&
            should_error)
32         {
33             errorMessage= calloc(100,
                sizeof(*errorMessage));
34             sprintf(errorMessage, "Attempt
                to pass extraneous
                parameter!");
35             throw_sem_error(errorMessage);
36         }
37         LangType e_type = expression()
            ;
38         if (should_error && to_match
            != NULL && e_type != ERR &&
            e_type != to_match -> type
            ) {
39             errorMessage= calloc(100,
                sizeof(*errorMessage));
40             sprintf(errorMessage, "
                Expected type %s, not %s!",
41                 typeNames[to_match -> type],
                    typeNames[e_type]);
42             throw_sem_error(errorMessage);
43         }
44         expression_list_tail(to_match
            == NULL || !to_match ->
            param ? NULL : to_match ->
            left, e_type != ERR &&

```

```

        should_error);
45 return;
46 }
47
48 synch();
49 }

```

---

Listing 35: *expression\_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 expression_list_tail(
   tree_node* to_match, bool
   should_error)
21 {
22     char* errorMessage;
23     // Production 20.2.1
24     if (tokens_equal(&comma_tok,
   current_tok, true))
25     {
26         match(&comma_tok, true);
27         if (to_match == NULL &&
   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 && to_match
   != NULL && e_type !=
   to_match -> type) {
35             errorMessage= calloc(100,
   sizeof(*errorMessage));
36             sprintf(errorMessage, "
   Expected type %s, not %s!",
37             typeNames[to_match -> type],
   typeNames[e_type]);
38             throw_sem_error(errorMessage);
39         }
40         expression_list_tail(to_match
   == NULL ? NULL : to_match
   -> left, should_error);
41         return;
42     }
43     // Production 20.2.2
44     } else if (tokens_equal(&
   rparen_tok, current_tok,
   true))
45     {
46         if (to_match != NULL &&
   to_match -> param &&
   should_error) {
47             errorMessage= calloc(100,
   sizeof(*errorMessage));
48             sprintf(errorMessage, "
   Expected %s, not the end of
   the parameters!",
49             typeNames[to_match -> type]);
50             throw_sem_error(errorMessage);
51         }

```

<pre> 52 return; // epsilon 53 } 54 55 synch(); 56 } </pre>	<pre> 26 if (tokens_equal(&amp;id_tok,                     current_tok, false)) { //                     id 27 Token* id_ref; 28 id_ref = match(&amp;id_tok, false)                 ; // id 29 LangType id_type = get_type(                     id_ref); 30 return factor_tail(id_type); 31 32 // Production 25.1.2 33 } else if (tokens_equal(&amp;                         num_tok, current_tok, false                         )) { // num 34 Token* num_type; 35 num_type = match(&amp;num_tok,                     false); 36 return num_type -&gt; aspect == 0                 ? INT : REAL; 37 38 // Production 25.1.3 39 } else if (tokens_equal(&amp;                         lparen_tok, current_tok,                         true)) { // ( 40 match(&amp;lparen_tok, true); 41 LangType e_type = expression()                 ; 42 match(&amp;rparen_tok, true); // ) 43 return e_type; 44 45 // Production 25.1.4 46 } else if (tokens_equal(&amp;                         not_tok, current_tok, true)                 ) { // not 47 Token* not_op; 48 not_op = match(&amp;not_tok, true)                 ; 49 LangType f_type = factor(); 50 return type_lookup(f_type, ERR                     , not_op); 51 } 52 53 54 synch(); 55 return ERR; 56 } </pre>
-------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 36: factor.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" 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 { 25 // Production 25.1.1 </pre>	<pre> 26 if (tokens_equal(&amp;id_tok,                     current_tok, false)) { //                     id 27 Token* id_ref; 28 id_ref = match(&amp;id_tok, false)                 ; // id 29 LangType id_type = get_type(                     id_ref); 30 return factor_tail(id_type); 31 32 // Production 25.1.2 33 } else if (tokens_equal(&amp;                         num_tok, current_tok, false                         )) { // num 34 Token* num_type; 35 num_type = match(&amp;num_tok,                     false); 36 return num_type -&gt; aspect == 0                 ? INT : REAL; 37 38 // Production 25.1.3 39 } else if (tokens_equal(&amp;                         lparen_tok, current_tok,                         true)) { // ( 40 match(&amp;lparen_tok, true); 41 LangType e_type = expression()                 ; 42 match(&amp;rparen_tok, true); // ) 43 return e_type; 44 45 // Production 25.1.4 46 } else if (tokens_equal(&amp;                         not_tok, current_tok, true)                 ) { // not 47 Token* not_op; 48 not_op = match(&amp;not_tok, true)                 ; 49 LangType f_type = factor(); 50 return type_lookup(f_type, ERR                     , not_op); 51 } 52 53 54 synch(); 55 return ERR; 56 } </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Listing 37: factor<sub>tail</sub>.c

```

1  #include<stdbool.h>
2
3  #include "productions.h"
4  #include "../parser.h"
5  #include "../tokenizer/
    tokens.h"
6
7  static const Token* first_set
    [] = {&lbrac_tok, &
        mulop_tok, &addop_tok, &
        relop_tok,
8  &semic_tok, &end_tok, &
        else_tok, &do_tok,
9  &then_tok, &rbrac_tok, &
        rparen_tok,
10 &comma_tok};
11 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
12
13 static const Token* sync_set[]
    = {&eof_tok, &mulop_tok, &
        addop_tok, &relop_tok,
14 &semic_tok, &end_tok, &
        else_tok, &do_tok,
15 &then_tok, &rbrac_tok, &
        rparen_tok,
16 &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 static LangType array_compare(
    LangType a_vals, LangType
    e_type) {
25     if ((a_vals == INT || a_vals
        == REAL) && e_type == INT)
26         return a_vals;
27     if (a_vals != ERR)
28     {
29         char* errorMessage = calloc
            (100, sizeof(*errorMessage)
            );
30         sprintf(errorMessage, "Attempt
            to index variable of type
            %s!", typeNames[a_vals]);
31         throw_sem_error(errorMessage);
32     }
33
34     return ERR;
35 }
36
37 // Needs implementing: None
38 LangType factor_tail(id_type)
39 {
40     // Production 25.2.1
41     if (tokens_equal(&lbrac_tok,
        current_tok, true)) {
42         match(&lbrac_tok, true);
43         LangType e_type = expression()
            ;
44         match(&rbrac_tok, true);
45         LangType n_type =
            convert_from_array(id_type)
            ;
46         return array_compare(n_type,
            e_type);
47
48     // Production 25.2.2
49     } else if (tokens_equal(&
        rparen_tok, current_tok,
        true)
50 || tokens_equal(&comma_tok,
        current_tok, true)
51 || tokens_equal(&semic_tok,
        current_tok, true)
52 || tokens_equal(&rbrac_tok,
        current_tok, true)
53 || tokens_equal(&addop_tok,
        current_tok, false)
54 || tokens_equal(&do_tok,

```

```

        current_tok, true)
55 || tokens_equal(&else_tok,
        current_tok, true)
56 || tokens_equal(&end_tok,
        current_tok, true)
57 || tokens_equal(&mulop_tok,
        current_tok, false)
58 || tokens_equal(&relop_tok,
        current_tok, false)
59 || tokens_equal(&then_tok,
        current_tok, true))
60 return id_type; // epsilon
61
62 synch();
63 return ERR;
64 }

```

---

Listing 38: *id<sub>list</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
    [] = {&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 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 39: *id<sub>list</sub><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
    [] = {&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 40: 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 another 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 41: 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 42: parameter<sub>list</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
    [] = {&id_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);

```



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

---

Listing 43: *parameter\_list\_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/ </pre>	<pre> // Production 10.2.2     rparen_tok, current_tok, </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------

```

        true)) // )
40 return; // epsilon
41
42 synch();
43 }

```

---

Listing 44: *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,
   current_tok, true)) // call
25 {
26 Token* id_ref;
27 match(&call_tok, true); //

```

```

        call
28 id_ref = match(&id_tok, false)
   ;
29 if (id_ref != NULL) {
30 tree_node* addition =
   start_param_matching(id_ref
   );
31 if (addition == NULL) {
32 errorMessage= calloc(100,
   sizeof(*errorMessage));
33 sprintf(errorMessage, "
   Procedure '%s' not in scope
   !", id_ref -> id);
34 throw_sem_error(errorMessage);
35
36 optional_expressions(NULL,
   false);
37 } else
38 optional_expressions(addition
   -> left == NULL ? NULL :
   addition -> left -> param
   ? addition -> left : NULL,
   true);
39 } else {
40 optional_expressions(NULL,
   false);
41 }
42 return;
43 }
44
45 synch();
46 }

```

---

Listing 45: *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/
   tokens.h"
8
9 static const Token* first_set
   [] = {&program_tok};

```

```

10 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
11
12 static const Token* sync_set[]
    = {&eof_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 program()
22 {
23     Token* id_ref;
24     // Production 1
25     if (tokens_equal(&program_tok,
        current_tok, true)) {
26         match(&program_tok, true); //
            program
27         id_ref = match(&id_tok, false)
            ; // id
28         if (id_ref != NULL) {
29             id_ref -> type = PGNAME;
30             id_ref -> param = false;
31             check_add_node(id_ref);
32         }
33         match(&lparen_tok, true); // (
34         id_list();
35         match(&rparen_tok, true); // )
36         match(&semicolon_tok, true); // ;
37         declarations();
38         subprogram_declarations();
39         compound_statement();
40         match(&period_tok, true); // .
41         return;
42     }
43
44     synch();
45 }

```

---

Listing 46: 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     &semicolon_tok, &end_tok, &
        else_tok, &do_tok,
10     &then_tok, &rbracket_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, &semicolon_tok, &
        end_tok,
15     &else_tok, &do_tok, &then_tok,
        &rbracket_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 47: 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 48: simple\_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, &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,
     current_tok, false)
29   || tokens_equal(&not_tok,
     current_tok, true)
30   || tokens_equal(&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(&
     plus_tok, current_tok, true
37   )
     || tokens_equal(&minus_tok,
     current_tok, true)) {
38     sign();
39     LangType t_type = term();
40     if (t_type != INT && t_type !=
     REAL && t_type != ERR)
41     {
42       errorMessage= calloc(100,
     sizeof(*errorMessage));
43       sprintf(errorMessage, "
     Expected number for use
     with sign, not %s!",
44       typeName[t_type]);
45       throw_sem_error(errorMessage);
46     }
47     return simple_expression_tail(
       t_type);
48   }
49
50   synch();
51   return ERR;
52 }

```

---

Listing 49: *simple\_expression\_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
  [] = {&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 =

```

<pre> 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)); 33 34 35     // Production 23.2.2 36     } else if (tokens_equal(&amp;         rparen_tok, current_tok,         true) 37        tokens_equal(&amp;comma_tok,         current_tok, true) 38        tokens_equal(&amp;semic_tok,         current_tok, true) </pre>	<pre> 39        tokens_equal(&amp;rbrac_tok,         current_tok, true) 40        tokens_equal(&amp;do_tok,         current_tok, true) 41        tokens_equal(&amp;else_tok,         current_tok, true) 42        tokens_equal(&amp;end_tok,         current_tok, true) 43        tokens_equal(&amp;relop_tok,         current_tok, false) 44        tokens_equal(&amp;then_tok,         current_tok, true)) 45     return t_type; // epsilon 46 47     synch(); 48     return ERR; 49 } </pre> <hr/> <p style="text-align: center;">Listing 50: standard<sub>t</sub>type.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;integer_tok, &amp;     real_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, &amp;     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, </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<pre> first_size); 17 } 18 19 // Needs implementing: None 20 LangType standard_type() 21 { 22 // Production 5.1 23 if (tokens_equal(&amp;integer_tok, current_tok, true)) // integer 24 { 25 match(&amp;integer_tok, true); 26 return INT; 27 28 // Production 5.2 29 } else if (tokens_equal(&amp; real_tok, current_tok, true )) { // real 30 match(&amp;real_tok, true); 31 return REAL; 32 } 33 34 synch(); 35 return ERR; 36 } </pre>	<pre> end_tok, &amp;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(&amp;id_tok, current_tok, false)) { // id 26 Token* id_ref = current_tok; 27 LangType v_type = variable(); 28 match(&amp;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 -&gt; id); 36 throw_sem_error(errorMessage); 37 expression(); 38 } else if (v_type != ERR &amp;&amp; v_type != INT &amp;&amp; v_type != REAL) 39 { 40 errorMessage = calloc(100, sizeof(*errorMessage)); 41 sprintf(errorMessage, "Cannot assign to ID '%s' of type '%s'!", 42 id_ref -&gt; id, typeNames[v_type </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 51: statement.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" 7 8 static const Token* first_set [] = {&amp;id_tok, &amp;call_tok, &amp; begin_tok, &amp;while_tok, 9 &amp;if_tok}; 10 static const int first_size = sizeof(first_set)/sizeof( first_set[0]); 11 12 static const Token* sync_set[] = {&amp;eof_tok, &amp;semic_tok, &amp; </pre>	<pre> 32 { 33 errorMessage = calloc(100, sizeof(*errorMessage)); 34 sprintf(errorMessage, "ID '%s' not in scope!", 35 id_ref -&gt; id); 36 throw_sem_error(errorMessage); 37 expression(); 38 } else if (v_type != ERR &amp;&amp; v_type != INT &amp;&amp; v_type != REAL) 39 { 40 errorMessage = calloc(100, sizeof(*errorMessage)); 41 sprintf(errorMessage, "Cannot assign to ID '%s' of type '%s'!", 42 id_ref -&gt; id, typeNames[v_type </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<pre>     }); 43 throw_sem_error(errorMessage); 44 expression(); 45 } else { 46 LangType e_type = expression()     ; 47 type_lookup(v_type, e_type, &amp;     assignop_tok); 48 } 49 return; 50 51 // Production 14.2 52 } else if (tokens_equal(&amp;     call_tok, current_tok, true     )) { // call 53 procedure_statement(); 54 return; 55 56 // Production 14.3 57 } else if (tokens_equal(&amp;     begin_tok, current_tok,     true)) { // begin 58 compound_statement(); 59 return; 60 61 // Production 14.4 62 } else if (tokens_equal(&amp;     while_tok, current_tok,     true)) { // while 63 match(&amp;while_tok, true); //     while 64 LangType e_type = expression()     ; 65 if (e_type != BOOL &amp;&amp; e_type     != ERR) 66 { 67 errorMessage= calloc(100,     sizeof(*errorMessage)); 68 sprintf(errorMessage, "     Expression in while must be     boolean, not %s!", 69 typeName[e_type]); 70 throw_sem_error(errorMessage); 71 } 72 match(&amp;do_tok, true); 73 statement(); </pre>	<pre> 74 return; 75 76 // Production 14.5 77 } else if (tokens_equal(&amp;     if_tok, current_tok, true))     { // if 78 match(&amp;if_tok, true); // if 79 LangType e_type = expression()     ; 80 if (e_type != BOOL &amp;&amp; e_type     != ERR) 81 { 82 errorMessage= calloc(100,     sizeof(*errorMessage)); 83 sprintf(errorMessage, "If     clause must be a boolean     expression, not %s!", 84 typeName[e_type]); 85 throw_sem_error(errorMessage); 86 } 87 match(&amp;then_tok, true); //     then 88 statement(); 89 else_tail(); 90 return; 91 } 92 93 94 synch(); 95 } </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 52: statement\_list.c

<pre> 64 LangType e_type = expression()     ; 65 if (e_type != BOOL &amp;&amp; e_type     != ERR) 66 { 67 errorMessage= calloc(100,     sizeof(*errorMessage)); 68 sprintf(errorMessage, "     Expression in while must be     boolean, not %s!", 69 typeName[e_type]); 70 throw_sem_error(errorMessage); 71 } 72 match(&amp;do_tok, true); 73 statement(); </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 static const Token* first_set     [] = {&amp;id_tok, &amp;call_tok, &amp;     begin_tok, &amp;while_tok, &amp; 8 &amp;if_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof( </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



```

        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 53: `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 54: `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 =
            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 }

```

---

Listing 55:  
subprogram\_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
    [] = {&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, &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 subprogram_declarations()
21 {
22     // Production 6.1
23     if (tokens_equal(&
        procedure_tok, current_tok,
        true)) // procedure
24     {
25         subprogram_declaration();
26         match(&semic_tok, true); // ;
27         subprogram_declarations();

```

```

28 return;
29
30 // Production 6.2
31 } else if (tokens_equal(&
    begin_tok, current_tok,
    true)) // begin
32 return; // Epsilon
33
34 synch();
35 }

```

---

Listing 56: subprogram<sub>head</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
    [] = {&procedure_tok};
9 static const int first_size =
    sizeof(first_set)/sizeof(
        first_set[0]);
10
11 static const Token* sync_set[]
    = {&eof_tok, &var_tok, &
        procedure_tok,
12 &begin_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 bool subprogram_head()
22 {
23 bool result = false;

```

```

24 // Production 8
25 if (tokens_equal(&
    procedure_tok, current_tok,
    true)) // procedure
26 {
27 Token* id_ref;
28 match(&procedure_tok, true);
    // procedure
29 id_ref = match(&id_tok, false)
    ;
30 if (id_ref != NULL) {
31 id_ref -> type = PROC;
32 id_ref -> param = false;
33 result = check_add_node(id_ref
    );
34 }
35 arguments();
36 match(&semic_tok, true); // ;
37 return result;
38 }
39
40 synch();
41 return result;
42 }

```

---

Listing 57: term.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 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, &semic_tok,
12 &end_tok, &else_tok, &do_tok,

```

```

        &then_tok,
13  &rbrac_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,
        sync_size, first_set,
        first_size);
19  }
20
21  // Needs implementing: None
22  LangType term()
23  {
24  // Production 24.1
25  if (tokens_equal(&lparen_tok,
        current_tok, true) // (
26  || tokens_equal(&id_tok,
        current_tok, false) // ID
27  || tokens_equal(&not_tok,
        current_tok, true) // not
28  || tokens_equal(&num_tok,
        current_tok, false)) { //
        num
29  LangType f_type = factor();
30  return term_tail(f_type);
31  }
32
33  synch();
34  return ERR;
35  }

```

---

Listing 58: 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(&

```

<pre>         rparen_tok, current_tok,         true) 35    tokens_equal(&amp;comma_tok,         current_tok, true) 36    tokens_equal(&amp;semic_tok,         current_tok, true) 37    tokens_equal(&amp;rbrac_tok,         current_tok, true) 38    tokens_equal(&amp;addop_tok,         current_tok, false) 39    tokens_equal(&amp;do_tok,         current_tok, true) 40    tokens_equal(&amp;else_tok,         current_tok, true) 41    tokens_equal(&amp;end_tok,         current_tok, true) 42    tokens_equal(&amp;relop_tok,         current_tok, false) 43    tokens_equal(&amp;then_tok,         current_tok, true)) 44 return f_type; // epsilon 45 46 synch(); 47 return ERR; 48 } </pre>	<pre> 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(&amp;array_tok,         current_tok, true)) 24     { 25         char* errorMessage; 26         Token* numI; 27         Token* numF; 28         match(&amp;array_tok, true); //             array 29         match(&amp;lbrac_tok, true); // [ 30         numI = match(&amp;num_tok, false);             // num 31         match(&amp;dotdot_tok, true); //             .. 32         numF = match(&amp;num_tok, false);             // num 33         match(&amp;rbrac_tok, true); // ] 34         match(&amp;of_tok, true); // of 35         if (numI != NULL &amp;&amp; numF !=             NULL &amp;&amp; id_ref != NULL) 36         if (type_lookup(numI -&gt; aspect             == 0 ? INT : REAL, numF -&gt;             aspect == 0 ? INT : REAL,             &amp;dotdot_tok) != ERR) { 37         if (numI -&gt; int_val &gt;= numF -&gt;             int_val) { 38             errorMessage= calloc(100,                 sizeof(*errorMessage)); 39             sprintf(errorMessage, "                 Expected array end index %d                 to be strictly greater                 than start %d", numF -&gt; </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 59: type.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" 7 8 static const Token* first_set     [] = {&amp;integer_tok, &amp;         real_tok, &amp;array_tok}; 9 static const int first_size =     sizeof(first_set)/sizeof(         first_set[0]); 10 11 static const Token* sync_set[]     = {&amp;array_tok, &amp;         integer_tok, &amp;real_tok}; </pre>	<pre> 32 numF = match(&amp;num_tok, false);             // num 33 match(&amp;rbrac_tok, true); // ] 34 match(&amp;of_tok, true); // of 35 if (numI != NULL &amp;&amp; numF !=     NULL &amp;&amp; id_ref != NULL) 36 if (type_lookup(numI -&gt; aspect     == 0 ? INT : REAL, numF -&gt;     aspect == 0 ? INT : REAL,     &amp;dotdot_tok) != ERR) { 37 if (numI -&gt; int_val &gt;= numF -&gt;     int_val) { 38     errorMessage= calloc(100,         sizeof(*errorMessage)); 39     sprintf(errorMessage, "         Expected array end index %d         to be strictly greater         than start %d", numF -&gt; </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<pre>         int_val, numI -&gt; int_val); 40 throw_sem_error(errorMessage); 41 } 42 id_ref -&gt; array_length = numF     -&gt; int_val - numI -&gt;         int_val + 1; 43 } 44 return convert_to_array(         standard_type()); 45 46 // Production 4.1 47 } else if (tokens_equal(&amp;         integer_tok, current_tok,         true) // int 48    tokens_equal(&amp;real_tok,         current_tok, true)) //         real 49 { 50 return standard_type(); 51 } 52 53 synch(); 54 return ERR; 55 } </pre>	<pre>         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>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 60: 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
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, &assignop_tok
        };
12 static const int sync_size =
    sizeof(sync_set)/sizeof(

```

---

Listing 61: symbolTable.c

---

```

1 #include<stdlib.h>
2 #include<string.h>
3 #include<stdio.h> // TODO
    Remove
4
5 #include "../dataStructures/
    linkedList/linkedList.h"
6 #include "symbolTable.h"
7
8 LinkedList* symbolTable;
9
10 int initSymbolTable()
11 {
12 symbolTable = malloc(sizeof(*
    symbolTable));
13 symbolTable -> 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 ->
    head -> data);
21 }
22
23 char* checkSymbolTable(char*
    word)
24 {
25 // Then check the symbol table
26 struct node* node =
    symbolTable -> head;
27 while (node)
28 {
29 if (strcmp((char *) node ->
    data, word) == 0) // Match
30 return (char *)(node -> data);
31 node = node -> next;
32 }
33
34 return NULL;
35 }

```

---

Listing 62: addop.c

```

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

```

```

16 start++;
17 return start;
18
19 default: break;
20 }
21
22 return start;
23 }

```

---

Listing 63: catchall.c

```

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

```

```

32 } else if (str[start] == '.')
33 {
34     storage -> attribute = PUNC;
35     storage -> aspect = 2;
36     start++;
37 }
38
39 return start;
40 }

```

---

Listing 64: 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;
22     case ']':
23         storage -> aspect = 3;
24         start++;
25         break;
26     default:
27         break;
28     }
29
30 return start;
31 }
32
33

```

```

34 }

```

---

Listing 65: 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/
9     errorHandler.h"
10 #include "../symbolTable/
11     symbolTable.h"
12 #include "../dataStructures
13     /linkedList/linkedList.h"
14 #include "../tokens.h"
15
16 static char** reservedWords;
17 static int numReserved;
18 static enum TokenType*
19     categories;
20 static int* attributes;
21
22 static int getIndex(const char
23     ** array, size_t arr_size,
24     char* item)
25 {
26     while (arr_size > 0)
27     {
28         if (strcmp(array[arr_size -
29             1], item) == 0)
30             return arr_size - 1;
31         arr_size--;
32     }
33     return -1;
34 }
35
36 static int initResWords(FILE*
37     resFile)
38 {
39     static const int length = 11;
40     LinkedList* resWords = malloc(
41         sizeof(*resWords));
42     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,
        sizeof(catNames)/sizeof(char*)
        ,
        (char *) node -> data);
67  node = node -> next;
68  }
69
70  // Initialize the attribute
        table
71  attributes = malloc(
        numReserved*sizeof(int));
72  node = attrs -> head;
73
74  for (size_t i = 0; i <
        numReserved; i++) {
75  attributes[i] = *(int *) node
        -> data;
76  node = node -> next;
77  }
78
79  return 0;
80
81  }
82
83  int initIDResMachine(FILE*
        resFile)
84  {
85  if (initSymbolTable() == 0 &&
        initResWords(resFile) == 0)
86  return 0;
87  else
88  return 1;
89  }
90
91  static int isReserved(char*
        word)
92  {
93  // Check the reserved words
        table for a match first
94  for (size_t i = 0; i <
        numReserved; i++) {
95  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 66: 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 67: 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     return num;
22 }
23
24 double parseReal(LinkedList*
    digits)
25 {
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
93     if (str[start] == '.')
94         add(digits, &str[start++],
            sizeof(char*));
95     else // Not a real
96     return initial;
97
98
99     while (isdigit(str[start]))
100     add(digits, &str[start++],
        sizeof(char*));
101
102     fracPart = start - (initial +
        intPart + 1);
103
104     if (fracPart == 0) // Not a
        real
105     return initial;
106
107     // Now, we check for errors.
108     if (intPart > 5)
109     {
110         throw_lex_error(LEXERR, 3,
            initial, start - initial);
111         errored = true;
112     }
113     if (fracPart > 5)
114     {
115         throw_lex_error(LEXERR, 4,
            initial, start - initial);
116         errored = true;
117     }
118     if (str[initial] == '0' &&
        intPart > 1) // Leading
        zero!
119     {
120         throw_lex_error(LEXERR, 8,
            initial, start - initial);
121         errored = true;
122     }
123     if (str[start - 1] == '0' &&
        fracPart > 1) // Trailing
        zero!
124     {
125         throw_lex_error(LEXERR, 9,
            initial, start - initial);
126         errored = true;
127     }
128     /*
129         if (errored)
130             storage -> attribute =
                NOOP;*/
131     else
132     {
133         storage -> aspect = 1;
134         storage -> real_val =
            parseReal(digits);

```

```

135 }
136
137 return start;
138 }
139
140 int longRealMachine(Token*
    storage, char* str, int
    start)
141 {
142     storage->attribute = NUM;
143
144     int initial = start;
145     bool errored = false;
146
147     int intPart = 0;
148     int fracPart = 0;
149     int expPart = 0;
150
151     LinkedList* digits = malloc(
        sizeof(*digits));
152     while (isdigit(str[start]))
153         add(digits, &str[start++],
            sizeof(char*));
154
155     intPart = start - initial;
156     if (intPart == 0) // Not a
        real. Must start with a
        digit.
157     return initial;
158
159     // REAL part
160     if (str[start] == '.')
161         add(digits, &str[start++],
            sizeof(char*));
162     else // Not a real
163         return initial;
164
165     while (isdigit(str[start]))
166         add(digits, &str[start++],
            sizeof(char*));
167
168     fracPart = start - (initial +
        intPart + 1);
169
170     if (fracPart == 0) // Not a
171
172         real
173         return initial;
174
175         // LONG REAL part
176         int signum = 0;
177
178         if (str[start] == 'E')
179             add(digits, &str[start++],
                sizeof(char*));
180         else // Not a long real
181             return initial;
182
183         if (str[start] == '+' || str[
            start] == '-')
184             {
185                 signum++;
186                 add(digits, &str[start++],
                    sizeof(char*));
187             }
188
189         while (isdigit(str[start]))
190             add(digits, &str[start++],
                sizeof(char*));
191
192         expPart = start - (initial +
            fracPart + intPart + signum
            + 2);
193
194         if (expPart == 0) // Not a
            long real
195             return initial;
196
197         // Now, we check for errors.
198         if (intPart > 5)
199             {
200                 throw_lex_error(LEXERR, 3,
                    initial, start - initial);
201                 errored = true;
202             }
203         if (fracPart > 5)
204             {
205                 throw_lex_error(LEXERR, 4,
                    initial, start - initial);
206                 errored = true;
207

```

<pre> 208 } 209 if (str[initial] == '0' &amp;&amp;     intPart &gt; 1) // <i>Leading                 zero!</i> 210 { 211     throw_lex_error(LEXERR, 8, 212                     initial, start - initial); 213     errored = true; 214 } 215 if (str[start - expPart - 2]     == '0' &amp;&amp; fracPart &gt; 1) //     <i>Trailing zero in real!</i> 216 { 217     throw_lex_error(LEXERR, 9, 218                     initial, start - initial); 219     errored = true; 220 } 221 if (expPart &gt; 2) // <i>Exponent                 too long!</i> 222 { 223     throw_lex_error(LEXERR, 5, 224                     initial, start - initial); 225     errored = true; 226 } 227 if (str[start - expPart] == '0     ') // <i>Leading zero in         exponent!</i> 228 { 229     throw_lex_error(LEXERR, 10, 230                     initial, start - initial); 231     errored = true; 232 } 233 /* 234     if (errored) 235         storage -&gt; attribute = 236             NOOP; 237     else*/ 238 { 239     storage -&gt; aspect = 1; 240     storage -&gt; real_val = 241         parseReal(digits); 242 } 243 return start; 244 } </pre>	<div style="text-align: right; margin-bottom: 10px;">Listing 68: relop.c</div> <hr style="border: 0; border-top: 1px solid black; margin: 0;"/> <pre> 1  #include "../tokens.h" 2  #include "machines.h" 3 4  int relop(Token* storage, char     * str, int start) 5  { 6      storage -&gt; attribute = RELOP; 7      char next = str[start]; 8      switch (next) { 9          case '&lt;': 10             start++; 11             if (str[start] == '=') 12             { 13                 storage -&gt; aspect = 1; 14                 start++; 15             } else if (str[start] == '&gt;') 16             { 17                 storage -&gt; aspect = 5; 18                 start++; 19             } else { 20                 storage -&gt; aspect = 0; 21             } 22             break; 23 24             case '=': 25                 start++; 26                 storage -&gt; aspect = 2; 27                 break; 28 29             case '&gt;': 30                 start++; 31                 if (str[start] == '=') 32                 { 33                     storage -&gt; aspect = 4; 34                     start++; 35                 } else { 36                     storage -&gt; aspect = 3; 37                 } 38                 break; 39 40             default: break; // <i>Do not                             increment; continue on to                             the next machine.</i> 41     } </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<pre> 42 43 return start; 44 } </pre>	<pre> 13 realMachine, intMachine,     grouping, catchall, relop,     addop, mulop}; </pre>
<hr style="width: 50%; margin: 0 auto;"/>	
<div style="display: inline-block; width: 45%; text-align: left;"> Listing 69: whitespace.c </div>	
<pre> 1 #include&lt;stdlib.h&gt; 2 #include&lt;ctype.h&gt; 3 4 #include "../tokens.h" 5 #include "machines.h" 6 7 int whitespace(Token* storage,     char* str, int start) 8 { 9     storage -&gt; attribute = WS; 10    if (isspace(str[start])) 11    { 12        storage -&gt; aspect = 0; 13        if (str[start] == '\n') 14        storage -&gt; aspect = 1; 15        start++; 16    } 17    return start; 18 } </pre>	<pre> 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 -&gt; start = START; 40         for (int i = 0; i &lt; sizeof(             machines)/sizeof(machine);             i++) 41         { 42             current -&gt; aspect = 0; 43             end = (*machines[i])(current,                 BUFFER, START); </pre>
<hr style="width: 50%; margin: 0 auto;"/>	
<div style="display: inline-block; width: 45%; text-align: left;"> Listing 70: tokenizer.c </div>	
<pre> 1 #include&lt;stdio.h&gt; 2 #include&lt;stdlib.h&gt; 3 #include&lt;stdbool.h&gt; 4 #include&lt;string.h&gt; 5 6 #include "tokenizer.h" 7 #include "../dataStructures/     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, </pre>	<pre> 41         { 42             current -&gt; aspect = 0; 43             end = (*machines[i])(current,                 BUFFER, START); </pre>

<pre> 44 if (end &gt; START) { 45     current -&gt; length = end -         START; 46     START = end; 47     return current; 48 } 49 } 50 51 // Unrecognized symbol error.     This error is manual     because it takes 52 // the place of a lexeme,     rather than being processed     during one. 53 throw_lex_error(LEXERR, 0,         START, 1); 54 //current -&gt; attribute = NOOP; 55 START++; 56 return current; 57 } else { 58     fprintf(stderr, "%s\n", "         Tokenizer not initialized.         Aborting."); 59     return NULL; 60 } 61 } 62 63 64 Token* getNextToken() 65 { 66     Token* next = malloc(sizeof(*         next)); 67     do { 68         next = generateNextToken(); 69     } while (next -&gt; attribute ==         NOOP); 70 71     return next; 72 } </pre>	<pre> 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 = {     ARRAY, 1, true, 0, 0}; 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}; </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

---

Listing 71: tokens.c

<pre> 1 #include&lt;string.h&gt; 2 #include&lt;stdlib.h&gt; 3 #include&lt;stdio.h&gt; 4 </pre>	<pre> 5 #include "tokens.h" 6 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 = {     ARRAY, 1, true, 0, 0}; 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}; </pre>
------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



```

30 const Token array_tok = {ARRAY 53
    , 0, true, 0, 0};
31 const Token assignop_tok = { 54 static const char* lexes[] = {
    ASSIGNOP, 0, true, 0, 0};
    "(", ")", "+", " ", "-", ";
32 const Token begin_tok = { 55 "array", "assignop", "begin",
    CONTROL, 0, true, 0, 0};
    "call", "do", "else",
33 const Token call_tok = { 56 "end", "ID", "if", "integer",
    CONTROL, 10, true, 0, 0};
    "mulop", "not",
34 const Token do_tok = {CONTROL, 57 "num", "procedure", "program",
    1, true, 0, 0};
    "real", "relop",
35 const Token else_tok = { 58 "then", "var", "while", "EOF",
    CONTROL, 2, true, 0, 0};
    "..", ":", ".",
36 const Token end_tok = {CONTROL 59 "int value", "of", "real value
    , 3, true, 0, 0};
    "};
37 const Token id_tok = {ID, 0, 60
    false, 0, 0};
38 const Token if_tok = {CONTROL, 61 static const Token* tokens[] =
    5, true, 0, 0};
    {&lparen_tok, &rparen_tok,
39 const Token integer_tok = {
    TYPE, 1, true, 0, 0};
    &plus_tok, &comma_tok, &
40 const Token integer_val_tok = 62 &colon_tok, &lbrac_tok, &
    {NUM, 0, true, 0, 0};
    rbrac_tok, &addop_tok, &
41 const Token of_tok = {ARRAY, 63 &begin_tok, &call_tok, &do_tok
    2, true, 0, 0};
    , &else_tok, &end_tok, &
42 const Token real_val_tok = {
    NUM, 1, true, 0, 0};
    id_tok,
43 const Token mulop_tok = {MULOP 64 &if_tok, &integer_tok, &
    , 0, false, 0, 0};
    mulop_tok, &not_tok, &
44 const Token not_tok = {INVERSE 65 &procedure_tok, &program_tok,
    , 0, true, 0, 0};
    &real_tok, &relop_tok, &
45 const Token num_tok = {NUM, 0,
    false, 0, 0};
    then_tok,
46 const Token procedure_tok = { 66 &var_tok, &while_tok, &eof_tok
    CONTROL, 6, true, 0, 0};
    , &dotdot_tok,
47 const Token program_tok = { 67 &colon_tok, &period_tok, &
    CONTROL, 7, true, 0, 0};
    integer_val_tok,
48 const Token real_tok = {TYPE, 68 &of_tok, &real_val_tok};
    2, true, 0, 0};
69
49 const Token relop_tok = {RELOP 70 const Token* getTokenFromLex(
    , 0, false, 0, 0};
    char* lex) {
71 for (int i = 0; i < sizeof(
50 const Token then_tok = {
    CONTROL, 8, true, 0, 0};
    lexes); i++) {
72 if (strcmp(lexes[i], lex) ==
51 const Token var_tok = {VAR, 0,
    true, 0, 0};
73 return tokens[i];
74 }
52 const Token while_tok = { 75
    CONTROL, 9, true, 0, 0};

```

```

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)
    return "CONTROL"; else
98 switch (token -> aspect) {
99 case 0: return "begin";
100 case 1: return "do";
101 case 2: return "else";
102 case 3: return "end";
103 case 4: return "function";
104 case 5: return "if";
105 case 6: return "procedure";
106 case 7: return "program";
107 case 8: return "then";
108 case 9: return "while";
109 case 10: return "call";
110 }
111
112 case ADDOP: if (!strict)
    return "ADDOP"; else
113 switch (token -> aspect) {
114 case 0: return "+";
115 case 1: return "-";
116 }
117
118 case MULOP: if (!strict)
    return "MULOP"; else
119 switch (token -> aspect) {
120 case 0: return "*";
121 case 1: return "/";
122 }
123
124 case ARRAY: if (!strict)
    return "ARRAY"; else
125 switch (token -> aspect) {
126 case 0: return "array";
127 case 1: return "..";
128 case 2: return "of";
129 }
130
131 case TYPE: switch (token ->
    aspect) {
132 case 0: return ":";
133 case 1: return "integer";
134 case 2: return "real";
135 }
136
137 case VAR: switch (token ->
    aspect) {
138 case 0: return "var";
139 }
140
141 case NUM: if (!strict) return
    "a number"; else
142 switch (token -> aspect) {
143 case 0: return "integer value"
    ;
144 case 1: return "real value";
145 }
146
147 case PUNC: switch (token ->
    aspect) {
148 case 0: return ",";
149 case 1: return ";";
150 case 2: return ".";
151 }
152
153 case GROUP: switch (token ->
    aspect) {
154 case 0: return "(";
155 case 1: return ")";

```

```

156 case 2: return "[";
157 case 3: return "]";
158 }
159
160 case INVERSE: switch (token ->
161     aspect) {
162 case 0: return "not";
163 }
164 case NOOP:
165 case WS:
166 case LEXERR:
167 case SYNERR:
168 case SEMERR: return "An error
169     in the compiler has
170     occurred.";
171
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;
190
191     // Type mismatch!!
192     default: errorMessage = calloc
193         (150, sizeof(*errorMessage))
194         );
195     sprintf(errorMessage, "Attempt
196         to create array using type
197         %s; must use integer or
198         real instead!", typeNames[
199
200     type]);
201     throw_sem_error(errorMessage);
202
203     case ERR: return ERR;
204 }
205
206 LangType convert_from_array(
207     LangType type) {
208     char* errorMessage;
209     switch (type) {
210     case AINT: return INT;
211     case AREAL: return REAL;
212
213     default: errorMessage = calloc
214         (100, sizeof(*errorMessage))
215         );
216     sprintf(errorMessage, "Attempt
217         to index variable of type
218         %s!", typeNames[type]);
219     throw_sem_error(errorMessage);
220
221     case ERR: return ERR;
222 }
223
224 static LangType
225 assignop_lookup(LangType
226     first, LangType second) {
227     char* errorMessage;
228     if (first == ERR || second ==
229         ERR) // just an err
230     return ERR;
231     else if (first != INT && first
232         != REAL) {
233     errorMessage = calloc(100,
234         sizeof(*errorMessage));
235     sprintf(errorMessage, "Cannot
236         assign values to variables
237         of type %s!", typeNames[
238         first]);
239     throw_sem_error(errorMessage);
240     return ERR;
241     }
242     else if (second != INT &&
243         second != REAL) {

```

```

218 errorMessage= calloc(100,      245
        sizeof(*errorMessage)); 246 static LangType addop_lookup(
219 sprintf(errorMessage, "Attempt    LangType first, LangType
        to assign value %s; only    second, int opcode) {
        reals and integers can be 247 char* errorMessage;
        assigned!", typeNames[      248 switch (opcode) {
        second]);                  249 case 0:
220 throw_sem_error(errorMessage); 250 case 1: if (first == second &&
221 return ERR;                      (first == INT || first ==
222 }                                REAL))
223 else if (first != second) {      251 return first;
224 errorMessage= calloc(100,      252 else if (first != ERR &&
        sizeof(*errorMessage));    second != ERR) {
225 sprintf(errorMessage, "Attempt  253 errorMessage= calloc(100,
        to convert type %s into     sizeof(*errorMessage));
        type %s in assignment!",    254 sprintf(errorMessage, "Attempt
        typeNames[first], typeNames  to add incompatible types
        [second]);                 %s and %s!", typeNames[
226 throw_sem_error(errorMessage);    first], typeNames[second]);
227 return ERR;                    255 throw_sem_error(errorMessage);
228 }                               256 return ERR;
229                                257 }
230 return NULL;                    258
231 }                               259 return ERR;
232                                260
233 static LangType relop_lookup(    261
        LangType first, LangType    case 2: if (first == second &&
        second) {                  first == BOOL)
234 char* errorMessage;            263 return BOOL;
235 if (first == second && (first    264 else if (first != ERR &&
        == INT || first == REAL))  second != ERR) {
236 return BOOL;                  265 errorMessage= calloc(100,
237 else if (first != ERR &&        sizeof(*errorMessage));
        second != ERR) {          266 sprintf(errorMessage, "
238 errorMessage= calloc(100,      Expected BOOL and BOOL for
        sizeof(*errorMessage));    use with 'or', received %s
239 sprintf(errorMessage, "Attempt  and %s!", typeNames[first],
        to compare incompatible    typeNames[second]);
        types %s and %s!",         267 throw_sem_error(errorMessage);
        typeNames[first], typeNames  }
        [second]);                268 }
240 throw_sem_error(errorMessage);  269
241 }                               270 return ERR;
242                                271
243 return ERR;                     272 default: return NULL;
244 }                               273 }
                                274 }

```

```

275 static LangType mulop_lookup(
276     LangType first, LangType
        second, int opcode) {
277     char* errorMessage;
278
279     switch (opcode) {
280     case 0:
281     case 1: if (first == second &&
        (first == INT || first ==
            REAL))
282         return first;
283     else if ((first == REAL &&
        second == INT)
284         || (first == INT && second ==
            REAL)) {
285         errorMessage= calloc(100,
            sizeof(*errorMessage));
286         sprintf(errorMessage, "Attempt
            to multiply or divide
            incompatible types %s and
            %s!", typeNames[first],
            typeNames[second]);
287         throw_sem_error(errorMessage);
288     }
289     else if (first != ERR &&
        second != ERR) {
290         errorMessage= calloc(100,
            sizeof(*errorMessage));
291         sprintf(errorMessage, "
            Expceted ints or reals for
            multiplication, received %s
            and %s!", typeNames[first]
            , typeNames[second]);
292         throw_sem_error(errorMessage);
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,
            "Integers required with %s,
            received %s and %s!",
            opcode == 3 ? "div" : "mod",
            typeNames[first],
            typeNames[second]);
315         throw_sem_error(errorMessage);
316     }
317
318     return ERR;
319
320
321     default: return NULL;
322     }
323 }
324
325 static LangType not_lookup(
    LangType first, LangType
        second) {
326     char* errorMessage;
327
328     if (first == BOOL) // and
329         return BOOL;
330     else if (first != ERR)
331     {
332         errorMessage= calloc(100,
            sizeof(*errorMessage));

```

```

335 sprintf(errorMessage, "Expected BOOL use with 'not
      ', received %s!", typeNames
      [first]);
336 throw_sem_error(errorMessage);
337 }
338
339 return ERR;
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
356 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
      365 // meaningless
      366 case NOOP:
      367 case LEXERR:
      368 case SYNERR:
      369 case SEMERR:
      370 case GROUP:
      371 case PUNC:
      372 case FILEEND:
      373 case ID:
      374 case CONTROL:
      375 case WS:
      376 case TYPE:
      377 case VAR:
      378 case NUM: return NULL;
      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 }

```

---

```

359 LangType type_lookup(LangType
      first, LangType second,
      Token* op) {
360 if (first == ERR || second ==
      ERR || op == NULL)
361 return ERR;
362

```