CS 4013 Compiler Construction SYNTAX ANALYSIS

Michael Collett

September 2018

INTRODUCTION

------------

This Syntax analyzer is written in C for the language Pascal. It is a separate machine that invokes the lexical analyzer discussed previously. The make file is included and can be compiled by simply typing make. The code is then run by typing ./Lex. The grammar given to us was modified and massaged into a LL(1) grammar. I used this grammar to construct a recursive descent parser. The, future, compiler catches all syntactic and lexical errors and reports them via the listing file.

METHODOLOGY

-----------

The recursive descent parser was built to match the final grammar. I utilized the first and follows as well as the parse table to assist me in its creation. The transformations made to the grammar include:

1. Initial specified modifications (part of project)
2. Removal of nullable productions
3. Left recursion elimination
4. Left factored grammar
5. First and Follow sets creation
6. Parse Table creation

0.

1. *program* --> **program id (**idlst)**;** *declarations subdeclarations cmpdstmnt* .

2.1 *idlst* --> **id**

2.2 *idlst* --> *idlst***, id**

3.1 *declarations* --> *declarations* **var** *idlst*: *type* **;**

3.2 declarations --> **ϵ**

4.1 *type* --> *standard\_type*

4.2 *type* --> **array** [**num** **..** **num**] **of** *standard\_type*

5.1 *standard*\_*type* --> **integer**

5.2 *standard\_type*  --> **real**

6.1 *subdeclarations* --> *subdeclarations subdeclaration* ;

6.2 *subdeclarations* --> **ϵ**

7 *subdeclaration* --> subprog\_head *declarations* *cmpdstmnt*

8.1 *subprogram\_head* --> **function id** *arguments* : *standard\_type* **;**

8.2 *subprogram\_head* --> **procedure** **id** *arguments* **;**

9.1 *arguments* --> (*parameter\_list* )

9.2 *arguments* --> **ϵ**

10.1 *parameter\_list* --> *identifier\_list* : *type*

10.2 *parameter\_list*  --> *parameter\_list* ; *identifier\_list* : *type*

11 *compound statement* --> **begin** *optional\_statements* **end**

12.1 *optional\_statements* --> *statement\_list*

12.1 *optional\_statements* --> *statement\_list ; statement*

13.1 *statement* --> *variable* **assignop** *expression*

13.2 statement --> *procedure\_statement*

13.3 *statement* --> *compound\_statement*

13.4 *statement* --> **if** *expression* **then** *statement* **else** *statement*

13.5 *statement* --> **while** *expression* **do** *statement*

14.1 *variable* --> **id**

14.2 *variable* --> **id** [*expression*]

15.1 *procedure\_statement* --> **id**

15.2 *procedure\_statement* --> **id** (*expression\_list* )

16.1 *expression\_list*  --> *expression*

16.2 *expression\_list* --> *expression\_list*, *expression*

17.1 *expression* --> *simple\_expression*

17.2 *expression* --> *simple\_expression* **relop** *simple\_expression*

18.1 *simple\_expression* --> *term*

18.2 *simple\_expression* --> *sign term*

18.1 *simple\_expression* --> *simple\_expression* **addop** *term*

19.1 *term* --> *factor*

19.2 *term* --> *term* **mulop** *factor*

20.1 *factor* --> **id**

20.2 *factor* --> **id** ( *expression\_list* )

20.3 *factor* --> **num**

20.4 *factor* --> ( *expression* )

20.5 *factor* --> **not** *factor*

21.1 *sign* --> +

21.2 *sign* --> -

1.

1. *program* --> **program id (**idlst)**;** *declarations subdeclarations cmpdstmnt* .

2.1 *idlst* --> **id**

2.2 *idlst* --> *idlst***, id**

3.1 *declarations* --> *declarations* **var** **id**: *type* **;**

3.2 *declarations* --> **ϵ**

4.1 *type* --> *standard\_type*

4.2 *type* --> **array** [**num** **..** **num**] **of** *standard\_type*

5.1 *standard*\_*type* --> **integer**

5.2 *standard\_type*  --> **real**

6.1 *subdeclarations* --> *subdeclarations subdeclaration* ;

6.2 *subdeclarations* --> **ϵ**

7 *subdeclaration* --> subprog\_head *declarations subdeclarations* *cmpdstmnt*

8.1 *subprogram\_head* --> **procedure** **id** *arguments* **;**

9.1 *arguments* --> (*parameter\_list* )

9.2 *arguments* --> **ϵ**

10.1 *parameter\_list* --> **id** : *type*

10.2 *parameter\_list*  --> *parameter\_list* ; ***id*** **:** *type*

11 *compound statement* --> **begin** *optional\_statements* **end**

12.1 *optional\_statements* --> *statement\_list*

12.2 *optional\_statements* --> *statement\_list ; statement*

12.3 *optional\_statements* --> **ϵ**

13.1 *statement\_list --> statement*

13.2 *statement\_list --> statement\_list* **;** *statement*

14.1 *statement* --> *variable* **assignop** *expression*

14.2 statement --> *procedure\_statement*

14.3 *statement* --> *compound\_statement*

14.4 *statement* --> **if** *expression* **then** *statement* **else** *statement*

14.5 *statement* --> **if** *expression* **then** *statement*

14.6 *statement* --> **while** *expression* **do** *statement*

15.1 *variable* --> **id**

15.2 *variable* --> **id** [*expression*]

16.1 *procedure\_statement* --> **call** **id**

16.2 *procedure\_statement* --> **call** **id** (*expression\_list* )

17.1 *expression\_list*  --> *expression*

17.2 *expression\_list* --> *expression\_list*, *expression*

18.1 *expression* --> *simple\_expression*

18.2 *expression* --> *simple\_expression* **relop** *simple\_expression*

19.1 *simple\_expression* --> *term*

19.2 *simple\_expression* --> *sign term*

19.3 *simple\_expression* --> *simple\_expression* **addop** *term*

20.1 *term* --> *factor*

20.2 *term* --> *term* **mulop** *factor*

21.1 *factor* --> **id**

21.2 *factor* --> **num**

21.3 *factor* --> ( *expression* )

21.4 *factor* --> **not** *factor*

21.5 *factor* --> **id** *[expression]*

22.1 *sign* --> +

22.2 *sign* --> -

2.

1.1.1 *program* --> **program id (**idlst)**;**

*declarations*

*subdeclarations*

*compound\_statement*

**.**

1.1.2 *program* --> **program id (**idlst)**;**

*subdeclarations*

*compound\_statement*

**.**

1.1.3 *program* --> **program id (**idlst)**;**

*declarations*

*compound\_statement*

**.**

1.1.4 *program* --> **program id (**idlst)**;**

*compound\_statement*

**.**

2.1 *idlst* --> **id**

2.2 *idlst* --> *idlst***, id**

3.1 *declarations* --> *declarations* **var** **id**: *type* **;**

3.2 *declarations* --> **var** **id**: *type* **;**

4.1 *type* --> *standard\_type*

4.2 *type* --> **array** [**num** **..** **num**] **of** *standard\_type*

5.1 *standard*\_*type* --> **integer**

5.2 *standard\_type*  --> **real**

6.1 *subdeclarations* --> *subdeclarations subdeclaration* ;

6.2 *subdeclarations* --> *subdeclaration* ;

7.1 *subdeclaration* --> subprog\_head *declarations subdeclarations* *compound\_statement*

7.2 *subdeclaration* --> subprog\_head *subdeclarations* *compound\_statement*

7.3 *subdeclaration* --> subprog\_head *declarations compound\_statement*

7.4 *subdeclaration* --> subprog\_head *compound\_statement*

8.1 *subprogram\_head* --> **procedure** **id** *arguments* **;**

8.2 *subprogram\_head* --> **procedure** **id;**

9.1 *arguments* --> (*parameter\_list* )

10.1 *parameter\_list* --> **id** : *type*

10.2 *parameter\_list*  --> *parameter\_list* ; ***id*** **:** *type*

11.1 *compound\_statement* --> **begin**

*optional\_statements*

**end**

11.2 *compound\_statement* --> **begin**

**end**

12.1 *optional\_statements* --> *statement\_list*

13.1 *statement\_list --> statement*

13.2 *statement\_list --> statement\_list* **;** *statement*

14.1 *statement* --> *variable* **assignop** *expression*

14.2 statement --> *procedure\_statement*

14.3 *statement* --> *compound\_statement*

14.4 *statement* --> **if** *expression* **then** *statement* **else** *statement*

14.5 *statement* --> **if** *expression* **then** *statement*

14.6 *statement* --> **while** *expression* **do** *statement*

15.1 *variable* --> **id**

15.2 *variable* --> **id** [*expression*]

16.1 *procedure\_statement* --> **call** **id**

16.2 *procedure\_statement* --> **call** **id** (*expression\_list* )

17.1 *expression\_list*  --> *expression*

17.2 *expression\_list* --> *expression\_list*, *expression*

18.1 *expression* --> *simple\_expression*

18.2 *expression* --> *simple\_expression* **relop** *simple\_expression*

19.1 *simple\_expression* --> *term*

19.2 *simple\_expression* --> *sign term*

19.3 *simple\_expression* --> *simple\_expression* **addop** *term*

20.1 *term* --> *factor*

20.2 *term* --> *term* **mulop** *factor*

21.1 *factor* --> **id**

21.2 *factor* --> **num**

21.3 *factor* --> ( *expression* )

21.4 *factor* --> **not** *factor*

21.5 *factor* --> **id** *[expression]*

22.1 *sign* --> +

22.2 *sign* --> -

3.

1.1.1 *program* --> **program id (**idlst**);**

*declarations*

*subdeclarations*

*compound\_statement*

**.**

1.1.2 *program* --> **program id (**idlst**);**

*subdeclarations*

*compound\_statement*

**.**

1.1.3 *program* --> **program id (**idlst**);**

*declarations*

*compound\_statement*

**.**

1.1.4 *program* --> **program id (**idlst**);**

*compound\_statement*

**.**

2.1 *idlst* --> **id** *idlst’*

2.2 *idlst’* --> **, id** *idlst’*

2.3 idlst’ --> ϵ

3.1 *declarations* --> **var** **id:** *type* **;** *declarations’*

3.2 *declarations’* --> **var** **id:** *type* **;** *declarations’*

3.3 *declarations’* --> ϵ

4.1 *type* --> *standard\_type*

4.2 *type* --> **array** [**num** **..** **num**] **of** *standard\_type*

5.1 *standard*\_*type* --> **integer**

5.2 *standard\_type*  --> **real**

6.1 *subdeclarations* --> *subdeclaration* **;** *subdeclarations’*

6.2 *subdeclarations’* --> *subdeclaration* **;** *subdeclarations’*

6.3 *subdeclarations’* --> ϵ

7.1 *subdeclaration* --> subprog\_head *declarations subdeclarations* *compound\_statement*

7.2 *subdeclaration* --> subprog\_head *subdeclarations* *compound\_statement*

7.3 *subdeclaration* --> subprog\_head *declarations compound\_statement*

7.4 *subdeclaration* --> subprog\_head *compound\_statement*

8.1 *subprogram\_head* --> **procedure** **id** *arguments* **;**

8.2 *subprogram\_head* --> **procedure** **id;**

9.1 *arguments* --> **(***parameter\_list***)**

10.1 *parameter\_list* --> **id** **:** *type parameter\_list’*

10.2 *parameter\_list’*  --> **;** ***id*** **:** *type parameter\_list’*

10.3 *parameter\_list’*  --> ϵ

11.1 *compound\_statement* --> **begin**

*optional\_statements*

**end**

11.2 *compound\_statement* --> **begin**

**end**

12.1 *optional\_statements* --> *statement\_list*

13.1 *statement\_list --> statement statement\_list’*

13.2 *statement\_list’ -->* **;** *statement statement\_list’*

13.3 *statement\_list’ -->* ϵ

14.1 *statement* --> *variable* **assignop** *expression*

14.2 *statement* --> *procedure\_statement*

14.3 *statement* --> *compound\_statement*

14.4 *statement* --> **if** *expression* **then** *statement* **else** *statement*

14.5 *statement* --> **if** *expression* **then** *statement*

14.6 *statement* --> **while** *expression* **do** *statement*

15.1 *variable* --> **id**

15.2 *variable* --> **id** [*expression*]

16.1 *procedure\_statement* --> **call** **id**

16.2 *procedure\_statement* --> **call** **id** (*expression\_list* )

17.1 *expression\_list*  --> *expression expression\_list’*

17.2 *expression\_list* ‘ --> **,** *expression expression\_list’*

17.3 *expression\_list* ‘ --> ϵ

18.1 *expression* --> *simple\_expression*

18.2 *expression* --> *simple\_expression* **relop** *simple\_expression*

19.1 *simple\_expression* --> *term simple\_expression’*

19.2 *simple\_expression* --> *sign term simple\_expression’*

19.3 *simple\_expression’* --> **addop** *term simple\_expression’*

19.4 *simple\_expression’* --> ϵ

20.1 *term* --> *factor term’*

20.2 *term’* --> **mulop** *factor term’*

20.3 *term’* --> ϵ

21.1 *factor* --> **id**

21.2 *factor* --> **num**

21.3 *factor* --> ( *expression* )

21.4 *factor* --> **not** *factor*

21.5 *factor* --> **id** *[expression]*

22.1 *sign* --> +

22.2 *sign* --> -

4., 5. AND 6.

IMPLEMENTATION

--------------

All of the productions are housed inside of the ./Productions folder. Each productions has its own c file that adheres to its own specified rules. As each line is passed in the file, the line is tokenized and checked for lexical errors via the lexical analyzer. The parser than receives these tokens and checks their syntax according to the grammar.

The parser calls both the match function and the getToken function. Match works by matching the current token with a specified token. If it is incorrect, an error is reported, if it is correct we get the next token via getToken. Once getToken gets to the end of the line, it loads and tokenized the next line of the source code until EOF.

Error recovery works by skipping tokens once an error is reported. It skips tokens until either EOF or the associated follow tokens are found. When errors are detected, they are reported as syntax errors to the listing file.

DISCUSSION AND CONCLUSIONS

--------------------------

The primary lesson from this project was the sheer power of a LL(1) grammar. In hindsight, it would have been much simpler and sexier to have implemented the grammar automatically instead of arbitrarily typing it up myself.

One thing that I need to change is that I have a pretty massive inefficiency when loading my reserved words… I should have loaded them all as global variables. Something I will alter in project 3!

APPENDIX 1

----------

SOURCE:

program fib(input, output);

var n: integer; var p: integer;

var q: real;

var numsArray : array [13..12] of integer;

procedure fib(a : integer; b : real; c : real);

begin

if a <= 1 then fib := c

else call fib(a - 1, c, b + c)

end;

procedure fib2(a : integer);

var b : integer; var c : integer; var sum : integer;

procedure rawr3(b : real);

var q : integer;

begin

q := b + 2.0;

call fib2(q)

end;

begin

a := a - 1;

b := 0;

sum := 1;

c := b;

while (a > 0) do

begin

a := a - 1;

b := sum;

sum := c + sum;

c := b

end;

fib2 := sum

end;

procedure init;

begin

n := 12;

if (1 and 2) or 3 then p := 12

else p := 14;

numsArray[3] := 15.56;

q := 12

end;

begin

call init;

call rawr3(34);

call writeln(+6\*q/p + 4);

call writeln(fib2\*n);

call writeln(numsArray[3] mod 15)

end.

LISTING:

1. program fib(input, output);

2. var n: integer; var p: integer;

3. var q: real;

4. var numsArray : array [13..12] of integer;

5.

6. procedure fib(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 : integer; var c : integer; var sum : integer;

14. procedure rawr3(b : real);

15. var q : integer;

16. begin

17. q := b + 2.0;

18. call fib2(q)

19. end;

20. begin

21. a := a - 1;

22. b := 0;

23. sum := 1;

24. c := b;

25. while (a > 0) do

26. begin

27. a := a - 1;

28. b := sum;

29. sum := c + sum;

30. c := b

31. end;

32. fib2 := sum

33. end;

34.

35. procedure init;

36. begin

37. n := 12;

38. if (1 and 2) or 3 then p := 12

39. else p := 14;

40. numsArray[3] := 15.56;

41. q := 12

42. end;

43.

44. begin

45. call init;

46. call rawr3(34);

47. call writeln(+6\*q/p + 4);

48. call writeln(fib2\*n);

49. call writeln(numsArray[3] mod 15)

50. end.

TOKEN:

Line No. Lexeme Token Type Attribute

1 program 30 0

1 fib 1 0x7ff57a402b00

1 ( 2 81

1 input 1 0x7ff57a402b80

1 , 4 85

1 output 1 0x7ff57a402c00

1 ) 2 82

1 ; 4 86

2 var 31 0

2 n 1 0x7ff57a402ce0

2 : 6 0

2 integer 34 0

2 ; 4 86

2 var 31 0

2 p 1 0x7ff57a402df0

2 : 6 0

2 integer 34 0

2 ; 4 86

3 var 31 0

3 q 1 0x7ff57a402f00

3 : 6 0

3 real 35 0

3 ; 4 86

4 var 31 0

4 numsArray 1 0x7ff57a403010

4 : 6 0

4 array 32 0

4 [ 2 83

4 13 10 0

4 .. 5 0

4 12 10 0

4 ] 2 84

4 of 33 0

4 integer 34 0

4 ; 4 86

6 procedure 37 0

6 fib 1 0x7ff57a402b00

6 ( 2 81

6 a 1 0x7ff57a4032d0

6 : 6 0

6 integer 34 0

6 ; 4 86

6 b 1 0x7ff57a4033b0

6 : 6 0

6 real 35 0

6 ; 4 86

6 c 1 0x7ff57a403490

6 : 6 0

6 real 35 0

6 ) 2 82

6 ; 4 86

7 begin 38 70

8 if 39 72

8 a 1 0x7ff57a4032d0

8 <= 7 88

8 1 10 0

8 then 39 73

8 fib 1 0x7ff57a402b00

8 := 3 0

8 c 1 0x7ff57a403490

9 else 39 74

9 call 43 0

9 fib 1 0x7ff57a402b00

9 ( 2 81

9 a 1 0x7ff57a4032d0

9 - 9 97

9 1 10 0

9 , 4 85

9 c 1 0x7ff57a403490

9 , 4 85

9 b 1 0x7ff57a4033b0

9 + 9 96

9 c 1 0x7ff57a403490

9 ) 2 82

10 end 38 71

10 ; 4 86

12 procedure 37 0

12 fib2 1 0x7ff57a403a80

12 ( 2 81

12 a 1 0x7ff57a4032d0

12 : 6 0

12 integer 34 0

12 ) 2 82

12 ; 4 86

13 var 31 0

13 b 1 0x7ff57a4033b0

13 : 6 0

13 integer 34 0

13 ; 4 86

13 var 31 0

13 c 1 0x7ff57a403490

13 : 6 0

13 integer 34 0

13 ; 4 86

13 var 31 0

13 sum 1 0x7ff57a403e00

13 : 6 0

13 integer 34 0

13 ; 4 86

14 procedure 37 0

14 rawr3 1 0x7ff57a403f10

14 ( 2 81

14 b 1 0x7ff57a4033b0

14 : 6 0

14 real 35 0

14 ) 2 82

14 ; 4 86

15 var 31 0

15 q 1 0x7ff57a402f00

15 : 6 0

15 integer 34 0

15 ; 4 86

16 begin 38 70

17 q 1 0x7ff57a402f00

17 := 3 0

17 b 1 0x7ff57a4033b0

17 + 9 96

17 2.0 11 0

17 ; 4 86

18 call 43 0

18 fib2 1 0x7ff57a403a80

18 ( 2 81

18 q 1 0x7ff57a402f00

18 ) 2 82

19 end 38 71

19 ; 4 86

20 begin 38 70

21 a 1 0x7ff57a4032d0

21 := 3 0

21 a 1 0x7ff57a4032d0

21 - 9 97

21 1 10 0

21 ; 4 86

22 b 1 0x7ff57a4033b0

22 := 3 0

22 0 10 0

22 ; 4 86

23 sum 1 0x7ff57a403e00

23 := 3 0

23 1 10 0

23 ; 4 86

24 c 1 0x7ff57a403490

24 := 3 0

24 b 1 0x7ff57a4033b0

24 ; 4 86

25 while 40 75

25 ( 2 81

25 a 1 0x7ff57a4032d0

25 > 7 93

25 0 10 0

25 ) 2 82

25 do 40 76

26 begin 38 70

27 a 1 0x7ff57a4032d0

27 := 3 0

27 a 1 0x7ff57a4032d0

27 - 9 97

27 1 10 0

27 ; 4 86

28 b 1 0x7ff57a4033b0

28 := 3 0

28 sum 1 0x7ff57a403e00

28 ; 4 86

29 sum 1 0x7ff57a403e00

29 := 3 0

29 c 1 0x7ff57a403490

29 + 9 96

29 sum 1 0x7ff57a403e00

29 ; 4 86

30 c 1 0x7ff57a403490

30 := 3 0

30 b 1 0x7ff57a4033b0

31 end 38 71

31 ; 4 86

32 fib2 1 0x7ff57a403a80

32 := 3 0

32 sum 1 0x7ff57a403e00

33 end 38 71

33 ; 4 86

35 procedure 37 0

35 init 1 0x7ff57a404e30

35 ; 4 86

36 begin 38 70

37 n 1 0x7ff57a402ce0

37 := 3 0

37 12 10 0

37 ; 4 86

38 if 39 72

38 ( 2 81

38 1 10 0

38 and 8 80

38 2 10 0

38 ) 2 82

38 or 9 77

38 3 10 0

38 then 39 73

38 p 1 0x7ff57a402df0

38 := 3 0

38 12 10 0

39 else 39 74

39 p 1 0x7ff57a402df0

39 := 3 0

39 14 10 0

39 ; 4 86

40 numsArray 1 0x7ff57a403010

40 [ 2 83

40 3 10 0

40 ] 2 84

40 := 3 0

40 15.56 11 0

40 ; 4 86

41 q 1 0x7ff57a402f00

41 := 3 0

41 12 10 0

42 end 38 71

42 ; 4 86

44 begin 38 70

45 call 43 0

45 init 1 0x7ff57a404e30

45 ; 4 86

46 call 43 0

46 rawr3 1 0x7ff57a403f10

46 ( 2 81

46 34 10 0

46 ) 2 82

46 ; 4 86

47 call 43 0

47 writeln 1 0x7ff57a405720

47 ( 2 81

47 + 9 96

47 6 10 0

47 \* 8 94

47 q 1 0x7ff57a402f00

47 / 8 95

47 p 1 0x7ff57a402df0

47 + 9 96

47 4 10 0

47 ) 2 82

47 ; 4 86

48 call 43 0

48 writeln 1 0x7ff57a405720

48 ( 2 81

48 fib2 1 0x7ff57a403a80

48 \* 8 94

48 n 1 0x7ff57a402ce0

48 ) 2 82

48 ; 4 86

49 call 43 0

49 writeln 1 0x7ff57a405720

49 ( 2 81

49 numsArray 1 0x7ff57a403010

49 [ 2 83

49 3 10 0

49 ] 2 84

49 mod 8 79

49 15 10 0

49 ) 2 82

50 end 38 71

50 . 4 87

-1 EOF 20 0