Week 4: Project 2

John Kucera

Prof. Renee McDonald

CMSC 430 Compiler Theory and Design

7 April 2021

Week 4: Project 2 Documentation

Modifying the Syntactical Analyzer Generator (C++ with Flex and Bison)

Approaching the Project

Just like the last project, when given skeleton code, my first step is always to understand everything about that given code. That way, I can modify it comfortably as if it were my own code. The videos from this week's course material were again extremely helpful for me when trying to understand the code. Dr. Duane J. Jarc explained the code very well, going over the syntax and purpose of every section. I was not only able to understand how the code relates to the course readings, but also how this project ties directly to the lexical analyzer from two weeks ago.

In modifying the code, I knew the first thing I had to do was replace some of the skeleton code with my own from the lexical analyzer, then alter scanner.l so that it would not interfere with parser.y. For example, parser.y (new skeleton code) already holds the main method, so I had that part out of scanner.l so there would not be two main methods causing an error.

After adding to the token declarations in parser.y, the next big step was modifying the grammar productions. The first I made sure to include were syntax error detections, which I realized was only necessary in the productions near the root. Including error productions with the

nonterminals function_header, variable, and statement were all that I needed to cover all possible production errors.

Another thing that took a lot of thinking and planning was maintaining proper precedence among the operators. I was confused at first on how to manage precedence with the bison file syntax but realized how simple it was when I took a step back and saw how the parser works overall. Adding precedence is simply like adding more branches. For example, an input term could be an OROP or something of higher precedence. It could then be an ANDOP or something of even higher precedence. Then a RELOP or something of even higher precedence, and so on.

Week 4: Project 2

Test Cases

(Test Case 1)

Aspect tested: An input file containing most of the grammar productions

Input file: testproductions.txt. Contains most productions listed in parser.y, no errors

Open testproductions.txt ~/Documents/CMSC 430/JohnK... Save

```
-- This tests multiple grammar productions in parser.y.
// Second type of comment

function testproductions a: integer returns boolean;
        my_var: real is 7.8E-3;
        my_var2: real is 5.1E11;
        b: integer is 10 * a / 2 + 4 - a ** 2;
        c: integer is 7 rem 9;
begin
        case a is
                when 1 =>
                        if a < b and a > c then
                                true;
                        else
                                false;
                        endif;
                when 2 =>
                        if b >= a or b <= c then
                                true;
                        else
                                false;
                        endif;
                when 3 =>
                        if not a /= b then
                                true;
                        else
                                false;
                        endif;
                when 4 =>
                        if a = b then
                                true;
                        else
                                false;
                        endif;
                when 5 =>
                        reduce +
                                true;
                        endreduce;
                others =>
                        false;
        endcase;
end;
```

Week 4: Project 2

Compilation output: SUCCESS. All grammar productions are valid, last line prints “Compiled successfully.”

```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$ ./compile < /home/
john/Documents/'CMSC 430'/'JohnKucera-Project2-Tests'/testproductions.txt

   1  -- This tests multiple grammar productions in parser.y.
   2  // Second type of comment
   3
   4  function testproductions a: integer returns boolean;
   5    my_var: real is 7.8E-3;
   6    my_var2: real is 5.1E11;
   7    b: integer is 10 * a / 2 + 4 - a ** 2;
   8    c: integer is 7 rem 9;
   9  begin
  10    case a is
  11            when 1 =>
  12                    if a < b and a > c then
  13                            true;
  14                    else
  15                            false;
  16                    endif;
  17            when 2 =>
  18                    if b >= a or b <= c then
  19                            true;
  20                    else
  21                            false;
  22                    endif;
  23            when 3 =>
  24                    if not a /= b then
  25                            true;
  26                    else
  27                            false;
  28                    endif;
  29            when 4 =>
  30                    if a = b then
  31                            true;
  32                    else
  33                            false;
  34                    endif;
  35            when 5 =>
  36                    reduce +
  37                            true;
  38                    endreduce;
  39            others =>
  40                    false;
  41    endcase;
  42  end;

Compiled Successfully.

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$
```

Week 4: Project 2

(Test Case 2)

Aspect tested: An input file containing more grammar productions

Input file: testproductions2.txt. Contains more productions listed in parser.y, such as having no parameters, no variables, and using parentheses. No errors.

```
-- This tests multiple grammar productions in parser.y.
// Second type of comment

function testproductions2 returns real;
begin
        reduce *
                6.8E5 + 8.8E9;
                4.1E3 ** (4.1E6);
                (8.7E1 / 23.1E-90);
        endreduce;
end;
```

Compilation output: All grammar productions are valid, last line prints “Compiled successfully.”

```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$ ./compile < /home/
john/Documents/'CMSC 430'/'JohnKucera-Project2-Tests'/testproductions2.txt

   1  -- This tests multiple grammar productions in parser.y.
   2  // Second type of comment
   3
   4  function testproductions2 returns real;
   5  begin
   6    reduce *
   7            6.8E5 + 8.8E9;
   8            4.1E3 ** (4.1E6);
   9            (8.7E1 / 23.1E-90);
  10    endreduce;
  11  end;

Compiled Successfully.

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$
```

Week 4: Project 2

(Test Case 3)

Aspect tested: An input file containing more grammar productions

Input file: testproductions3.txt. Contains more productions listed in parser.y, such as having multiple parameters and having a single variable. No errors.

testproductions3.txt
~/Documents/CMSC 430/JohnK...

```
-- This tests multiple grammar productions in parser.y.
// Second type of comment

function testproductions2 a: boolean, b: real returns integer;
        the_var: boolean is false;
begin
        if the_var = a then
                3;
        else
                4;
        endif;
end;
```

Compilation output: All grammar productions are valid, last line prints “Compiled successfully.”

```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$ ./compile < /home/john/Documents/'CMSC 430'/'
JohnKucera-Project2-Tests'/testproductions3.txt

  1  -- This tests multiple grammar productions in parser.y.
  2  // Second type of comment
  3
  4  function testproductions2 a: boolean, b: real returns integer;
  5    the_var: boolean is false;
  6  begin
  7    if the_var = a then
  8            3;
  9    else
 10            4;
 11    endif;
 12  end;

Compiled Successfully.

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$
```

Week 4: Project 2

(Test Case 4)

Aspect tested: An input file containing multiple errors

Input file: testmulterrors.txt. Contains multiple lines where there are syntax errors, one being in the function_header.

```
-- This tests multiple syntactical errors.
// Second type of comment

function main my_var: real retrn boolean;
        my_var2: real is ** 2.11E5;
begin
        if my_var /= my_var2 then
                my_var > ;
        else
                my_var 90.1;
        endif;
;
```

Compilation output: SUCCESS. Error messages are displayed after each line that has an error. At the end, a count of errors of each type in addition to total number of errors is displayed.

```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$ ./compile < /home/
john/Documents/'CMSC 430'/'JohnKucera-Project2-Tests'/testmulterrors.txt

   1  -- This tests multiple syntactical errors.
   2  // Second type of comment
   3
   4  function main my_var: real retrn boolean;

syntax error, unexpected IDENTIFIER, expecting RETURNS

   5    my_var2: real is ** 2.11E5;

syntax error, unexpected EXPOP

   6  begin
   7    if my_var /= my_var2 then
   8            my_var > ;

syntax error, unexpected ';'

   9    else
  10            my_var 90.1;

syntax error, unexpected REAL_LITERAL, expecting ';'

  11    endif;
  12  ;

syntax error, unexpected ';', expecting END


Lexical Errors: 0
Syntactic Errors: 5
Semantic Errors: 0
Total Number of Errors: 5

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$
```

Week 4: Project 2

(Test Case 5)

Aspect tested: An input file containing a single error

Input file: testsingleerror.txt. Contains one line with a syntax error.

testsingleerror.txt
~/Documents/CMSC 430/...

```
-- This tests a single syntactical error.
// Second type of comment

function main my_var: real returns boolean;
        my_var2: real is 2.11E5;
begin
        if my_var /= my_var2 then
                my_var > ;
        else
                90.1;
        endif;
end;
```

Compilation output: SUCCESS. Error message is displayed after the line that has an error. At the end, a count of errors of each type in addition to total number of errors is displayed.

```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$ ./compile < /home/
john/Documents/'CMSC 430'/'JohnKucera-Project2-Tests'/testsingleerror.txt

   1  -- This tests a single syntactical error.
   2  // Second type of comment
   3
   4  function main my_var: real returns boolean;
   5    my_var2: real is 2.11E5;
   6  begin
   7    if my_var /= my_var2 then
   8            my_var > ;

syntax error, unexpected ';'

   9    else
  10            90.1;
  11    endif;
  12  end;

Lexical Errors: 0
Syntactic Errors: 1
Semantic Errors: 0
Total Number of Errors: 1

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project2$
```

Week 4: Project 2

(Test Cases: ERRORS)

Aspect(s) tested: A variety of specific syntax errors in the grammar productions.

Input file(s): testproductions.txt, testproductions2.txt. I modified parts of the files that would raise errors, bit by bit, while inputting them into the parser each time (and then reverted the files back to normal. The attached test files do NOT have these errors). This table shows the error messages raised for syntax errors in each grammar production.

Production tested	**Screenshot of error and error message**
function_header, colon missing	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 function testproductions a integer returns boolean; syntax error, unexpected INTEGER, expecting ':' 5 my_var: real is 7.8E-3;
function_header, mispelled	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 func testproductions a: integer returns boolean; syntax error, unexpected IDENTIFIER, expecting FUNCTION 5 my_var: real is 7.8E-3;
function, text between function_header and variable	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 function testproductions a: integer returns boolean; 5 9999: syntax error, unexpected INT_LITERAL, expecting IDENTIFIER or BEGIN_ 6 my_var: real is 7.8E-3;

Week 4: Project 2

variable, colon missing	`1 -- This tests multiple grammar productions in parser.y.`
	`2 // Second type of comment`
	`3`
	`4 function testproductions a: integer returns boolean;`
	`5 my_var real is 7.8E-3;`
	`syntax error, unexpected REAL, expecting ':'`
	`6 my_var2: real is 5.1E11;`

variable, misspelled IDENTIFIER	`1 -- This tests multiple grammar productions in parser.y.`
	`2 // Second type of comment`
	`3`
	`4 function testproductions a: integer returns boolean;`
	`5 my_var: real is 7.8E-3;`
	`6 my_var2: ral is 5.1E11;`
	`syntax error, unexpected IDENTIFIER, expecting BOOLEAN or INTEGER or REAL`
	`7 b: integer is 10 * a / 2 + 4 - a ** 2;`

variable, improper literal	`1 -- This tests multiple grammar productions in parser.y.`
	`2 // Second type of comment`
	`3`
	`4 function testproductions a: integer returns boolean;`
	`5 my_var: real is 8/;`
	`syntax error, unexpected ';'`
	`6 my_var2: real is 5.1E11;`

body, missing begin	`4 function testproductions a: integer returns boolean;`
	`5 my_var: real is 7.8E-3;`
	`6 my_var2: real is 5.1E11;`
	`7 b: integer is 10 * a / 2 + 4 - a ** 2;`
	`8 c: integer is 7 rem 9;`
	`9`
	`10 case a is`
	`syntax error, unexpected CASE, expecting IDENTIFIER or BEGIN_`
	`11 when 1 =>`

MULOP, no int or real on one side	`1 -- This tests multiple grammar productions in parser.y.`
	`2 // Second type of comment`
	`3`
	`4 function testproductions a: integer returns boolean;`
	`5 my_var: real is 7.8E-3;`
	`6 my_var2: real is 5.1E11;`
	`7 b: integer is 10 * / 2 + 4 - a ** 2;`
	`syntax error, unexpected MULOP`
	`8 c: integer is 7 rem 9;`

Week 4: Project 2

ADDOP, no int or real on one side	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 function testproductions a: integer returns boolean; 5 my_var: real is 7.8E-3; 6 my_var2: real is 5.1E11; 7 b: integer is 10 * a / 2 + - a ** 2; syntax error, unexpected ADDOP 8 c: integer is 7 rem 9;
EXPOP, no int or real on one side	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 function testproductions a: integer returns boolean; 5 my_var: real is 7.8E-3; 6 my_var2: real is 5.1E11; 7 b: integer is 10 * a / 2 + 4 - a ** ; syntax error, unexpected ';' 8 c: integer is 7 rem 9;
REMOP, no int or real on one side	1 -- This tests multiple grammar productions in parser.y. 2 // Second type of comment 3 4 function testproductions a: integer returns boolean; 5 my_var: real is 7.8E-3; 6 my_var2: real is 5.1E11; 7 b: integer is 10 * a / 2 + 4 - a ** 2; 8 c: integer is 7 rem ; syntax error, unexpected ';' 9 begin
case, int_literal missing	9 begin 10 case is syntax error, unexpected IS 11 when 1 => 12 if a < b and a > c then 13 true;
case, improper arrow	9 begin 10 case a is 11 when 1 = yntax error, unexpected RELOP, expecting ARROW 12 if a < b and a > c then

Week 4: Project 2

or, improper expression on one side	17 when 2 => 18 if b >= a or <= c then syntax error, unexpected RELOP 19 true; 20 else 21 false; 22 endif;
and, improper expression on one side	11 when 1 => 12 if a < b and then syntax error, unexpected THEN 13 true; 14 else 15 false; 16 endif;
if, missing semicolon	17 when 2 => 18 if b >= a or <= c then syntax error, unexpected RELOP 19 true; 20 else 21 false; 22 endif;
/= relop, improper expression on one side	23 when 3 => 24 if not a /= then syntax error, unexpected THEN 25 true; 26 else 27 false; 28 endif;
= relop, improper expression on one side	29 when 4 => 30 if a = then syntax error, unexpected THEN 31 true; 32 else 33 false; 34 endif;

Week 4: Project 2

reduce, missing operator	36 reduce 37 true; syntax error, unexpected BOOL_LITERAL, expecting ADDOP or MULOP 38 endreduce;
others, improper statement	39 others => 40 /; syntax error, unexpected MULOP 41 endcase; 42 end;
case, missing endcase	39 others => 40 false; 41 ; syntax error, unexpected ';', expecting ENDCASE 42 end;
missing end	35 when 5 => 36 reduce + 37 true; 38 endreduce; 39 others => 40 false; 41 endcase; 42 ; syntax error, unexpected ';', expecting END
not, improper expression	23 when 3 => 24 if not /= b then syntax error, unexpected RELOP 25 true; 26 else 27 false; 28 endif;
parentheses, improper expression inside	6 reduce * 7 6.8E5 + 8.8E9; 8 4.1E3 ** (/4.1E6); syntax error, unexpected MULOP 9 (8.7E1 / 23.1E-90);

Week 4: Project 2

reduce, missing endreduce	6 reduce * 7 6.8E5 + 8.8E9; 8 4.1E3 ** (4.1E6); 9 (8.7E1 / 23.1E-90); 10 ; syntax error, unexpected ';' 11 end;
if, missing endif	29 when 4 => 30 if a = b then 31 true; 32 else 33 false; 34 35 when 5 => syntax error, unexpected WHEN, expecting ENDIF

Week 4: Project 2

Lessons Learned

A big lesson I learned from this project was that it is sometimes a good idea to take a step back from the individual code I am working on and see the entire program as a whole and how it flows. This was evident when struggling in the precedence section that I previously mentioned. I was so focused on the fact that I do not know bison syntax well and that there must be some function or logic that I was unaware of. In reality, I forgot that the grammar production section is representative of a bottom-up parser. It is a root with branches and leaves. So, I just added more branches on top of branches to represent the precedence. I would not have come to this conclusion if I did not see the program as a whole and how much it should flow like a parser.

One thing I would like to improve on is also that precedence section. My non-terminals are labeled just as “precedence_1” and “precedence_2”, higher numbers representing higher precedence. For the sake of clarity, I wonder if there is some other naming standard for precedence in grammar productions. On a related note, I also wonder if there is a more convenient way for implementing precedence in these grammar productions. I understand that it is a parser and it is usually best to stick to the root-branches-leaves flow of the program, but it still makes me add seven different grammar productions just for operator precedence. If I knew a way to simplify this section of the code, I definitely would do it.