Project Approach

Project two really threw me for a loop. Although, I blame myself for not taking the time to go through the course material and do a little more research, as I always do. After project one, I thought maybe I could figure out how to write the parser without having to do addition learning, man was I wrong. I was having so much trouble with different errors popping up, and they were really inconsistent. Ultimately, I took some time and watched the course videos and read the yacc portions of our textbook. Also, I was doing some addition web browsing trying to see if there was anything else that would help put things into perspective. I did come across a text book that that helped me to better understand compiler basics and theory, called 'Introduction to Compilers and Language Design' written by Prof. Douglas Thain University of Notre Dame.

After reading, watching and finally understanding what the parser is doing, I made the rash decision to scrap my parser.y and start over. Which, usually is a terrible decision, and one that I tend to do a lot when I get stressed and frustrated. However, going back with a much clearer vision of how this should work was probably the best thing I could have done.

After re-writing the parser.y file, I did run into a few errors (which I will discuss in the 'Lessons Learned' column). Although, sometimes that can be my favorite part of writing code. Once I got everything working, I was really happy I started over, I was able to take a step back and understand a lot more.

Test Cases

Case 1:

- Case 1 is testing the syntax for:
- optional variables
- if statement
- arithmetic operations
- nested reduce statement

This case should print string 'Compiled Successfully!' in the terminal. There should not be any lexical, or syntax errors in this case.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ./compile < test_cases/test_1.txt

   1  -- Testing the use of:
   2  --       - optional variables
   3  --       - if/else statement
   4  --       - arithmetic operations
   5  --       - nested reduce statement
   6
   7  function test_1 returns integer;
   8      a: integer is 10;
   9      b: integer is 0;
  10  begin
  11      if (a > 5) and (b < 5) then
  12          reduce -
  13              a + b;
  14              reduce /
  15                  a;
  16              endreduce;
  17          endreduce;
  18      else
  19          a + b;
  20      endif;
  21  end;

Compiled Sucessfully!
```

Case 2:

Case 2 is testing syntax for:

- user passed input (in the command line)
- if statement
- case statement
- the use of boolean literals/booleans

This case should print string 'Compiled Successfully!' in the terminal. There should not be any lexical, or syntax errors in this case.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ./compile < test_cases/test_2.txt

   1  -- A guessing game that tests the use of:
   2  --       - user passed input
   3  --       - if statments
   4  --       - switch case
   5
   6  function test_2 a: integer returns boolean;
   7  begin
   8      if (a < 5) and (a > 0) then
   9          case a is
  10              when 0 =>  false;
  11              when 1 =>  false;
  12              when 2 =>  true;
  13              when 3 =>  false;
  14              when 4 =>  false;
  15              others =>  fasle;
  16          endcase;
  17      else
  18          false;
  19      endif;
  20  end;

Compiled Sucessfully!
```

Case 3:

Case 3 is testing syntax for:

- all arithmetic operations
- case statements
- the use of real literals/real

This case should print string ‘Compiled Successfully!’ in the terminal. There should not be any lexical, or syntax errors in this case.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ./compile < test_cases/test_3.txt

   1  // Testing the use of:
   2  //      - all arithmetic operations
   3  //      - case statements
   4  //      - the use of real literals/real
   5
   6  function test_3 z: integer returns real;
   7      x: real is 9e+3;
   8      y: real is 0.54;
   9  begin
  10      if (z >= 0) and (z <= 10) then
  11          case z is
  12              when 0 =>  x + z;
  13              when 1 =>  y + z;
  14              when 2 =>  X * y;
  15              when 3 =>  z - y;
  16              when 4 =>  z / x;
  17              when 5 =>  y + x;
  18              when 6 =>  y - x;
  19              when 7 =>  x / y;
  20              when 8 =>  z ** y;
  21              when 9 =>  z rem y;
  22              others =>  y rem x;
  23          endcase;
  24      else
  25          x /= y;
  26      endif;
      end;
Compiled Sucessfully!
```

Case 4:

Case 4 is testing syntax for:

- all relational operators
- arithmetic operations
- all logical operators

This case should print string 'Compiled Successfully!' in the terminal. There should not be any lexical, or syntax errors in this case.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ./compile < test_cases/test_4.txt

   1  -- Testing the use of:
   2  //      - all relational operators
   3  --      - arithmetic operations
   4  //      - all logical operators
   5
   6  function test_4 returns boolean;
   7      a: integer is 2;
   8      b: integer is 4;
   9      c: integer is a * b;
  10      d: integer is c - a;
  11  begin
  12      if ((a < b) or (c < a)) and (not ((b <= d) or (a >= b))) then
  13          true;
  14      else
  15          false;
  16      endif;
  17  end;

Compiled Sucessfully!
```

Case 5:

Case 5 is testing for syntax errors, and lexical errors found in the program. It should return multiple syntax and lexical errors that might occur.

Syntax errors/Lexical errors:

line 5: ‘ Lexical Error, Invalid Character _ ’ and ‘ Syntax Error, syntax error, unexpected INTEGER, expecting ':' ’

line 6: ‘ Syntax Error, syntax error, unexpected ADDOP ’

line 10: ‘ Syntax Error, syntax error, unexpected INT_LITERAL, expecting ';' ’ and ‘ Lexical Error, Invalid Character . ’

line 13: ‘ Syntax Error, syntax error, unexpected ADDOP ’

line 15: ‘ Lexical Error, Invalid Character $ ‘

line 17: ‘ Syntax Error, syntax error, unexpected ';', expecting ENDIF ’

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ./compile < test_cases/test_5.txt

   1  // Function that throws syntax errors and lexical errors
   2  --      lexical errors: 3
   3  //      syntax errors: 5
   4
   5  function test5_ a integer returns real;
Lexical Error, Invalid Character _
Syntax Error, syntax error, unexpected INTEGER, expecting ':'
   6      b: integer is + 2;
Syntax Error, syntax error, unexpected ADDOP
   7      c: integer is (a + b);
   8  begin
   9      if (c > 0) then
  10          c 2.e-2;
Syntax Error, syntax error, unexpected INT_LITERAL, expecting ';'
Lexical Error, Invalid Character .
  11      else
  12          case c is
  13              when 0 =>  + a;
Syntax Error, syntax error, unexpected ADDOP
  14              when 1 => c ** b;
  15              others => $0.50;
Lexical Error, Invalid Character $
  16          endcase;
  17      ;
Syntax Error, syntax error, unexpected ';', expecting ENDIF
  18  end;

 Lexical Errors:    3
 Sytax Errors:    5
 Semantic Errors:    0
 Total num. of errors:    8
```

Case 6:

Case 6 is testing the program's syntax and catching only syntax errors.

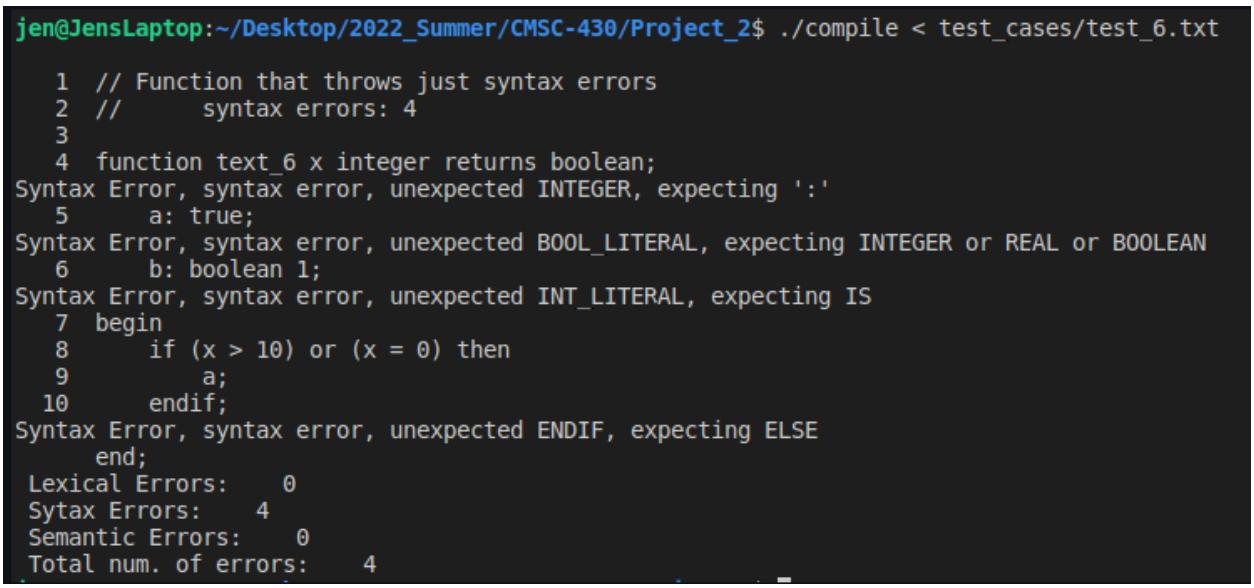
Syntax errors:

line 4: ' Syntax Error, syntax error, unexpected INTEGER, expecting ':' '

line 5: ' Syntax Error, syntax error, unexpected BOOL_LITERAL, expecting INTEGER or REAL or BOOLEAN '

line 6: ' Syntax Error, syntax error, unexpected INT_LITERAL, expecting IS '

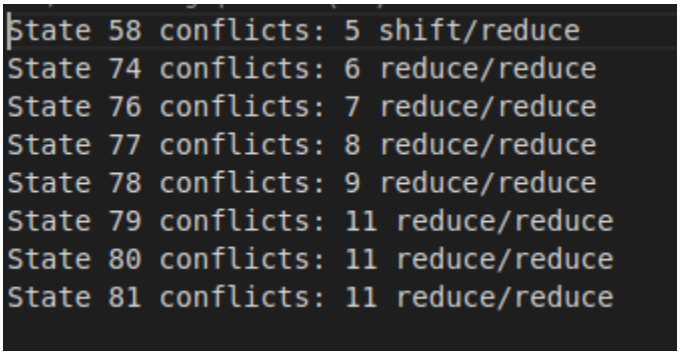
line 10: ' Syntax Error, syntax error, unexpected ENDIF, expecting ELSE '



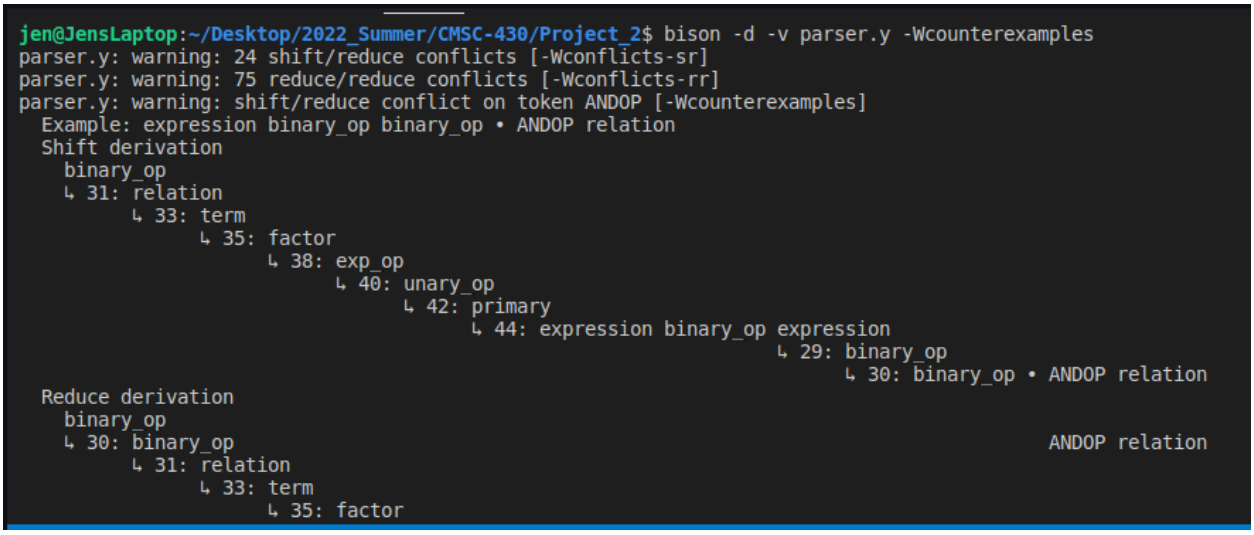
Lessons Learned

Project two taught me a lot of valuable lessons. The first lesson, was taking the time to learn and understand concepts before starting the work. Last week I thought maybe I had a good understanding of what the parser.y file was supposed to look like and how it should function. While writing/adding-to the parser.y file I quickly learned that I should not assume I know what I am doing without doing an adequate amount of research, and trying to understand the materials and outline of the project. After taking advantage of the course materials and doing outside learning, I decided to re-start and go at it again, this time with a much better idea of how the parser.y file should look and function.

The second lesson was learning to read and correct errors that were being returned. At first I ended up getting multiple shift/reduce and reduce/reduce conflicts.



Which I learned running '$ bison –d –v parser.y –Wcounterexamples', allowed me to see the breakdown of the parser with examples. This really helped me see where the file might have been conflicted.



After doing a little digging, I found that there was a rule under primary that causing the shift/reduce and reduce/reduce conflicts. Under the primary terminal the second rule stating 'expression binary_op expression' was causing my parser to run into conflicts.

```
primary:
    '(' expression ')' |
    expression binary_op expression |
    INT_LITERAL |
    REAL_LITERAL |      You, 29 seconds ago • Uncc
    BOOL_LITERAL |
    IDENTIFIER
;
```

After deleting this rule, I did not get anymore shift/reduce and reduce/reduce conflicts.

```
1jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$ ma
bison -d -v parser.y
mv parser.tab.c parser.c
cp parser.tab.h tokens.h
g++ -c scanner.c
g++ -c parser.c
g++ -o compile scanner.o parser.o listing.o
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$
```

The third lesson I learned during project two was to be mindful of simple/common mistakes. I had made a couple of these, and noticed them when running my test files. The biggest one caused me to go back to my scanner.l file and re-write how I described an int. I did have it as '[+-]?{digit}+', but I am pretty sure the [+-]? was causing my case statements to throw an 'Expecting INT_LITERAL, but received REAL_LITERAL' error. After re-writing the description of int to '[0-9]+' it seemed to work fine. Another silly mistake I made was how I set the rule for 'optional_variable'. At first I had the rule either shifting to 'variable' or 'error'.

```
;

optional_variable:
    optional_variable variable |
    error  ';' |
;
```

This was throwing an 'unexpected IDENTIFIER, expecting BEGIN_'. Basically, what I realized was 'optional_variable' was not recursive. So, you were not able to have multiple optional variables without throwing a syntax error. (In the image below, the function was supposed to compile successfully.)

```
   6
   7  function test_1 returns integer;
   8      a: integer is 10;
   9      b: integer is 0;
Syntax Error, syntax error, unexpected IDENTIFIER, expecting BEGIN_
  10  begin
  11      if (a > 5) and (b < 5) then
  12          reduce -
  13              a + b;
  14              reduce /
  15                  a;
  16              endreduce;
  17          endreduce;
  18      else
  19          a + b;
  20      endif;
  21  end;

 Lexical Errors:    0
 Sytax Errors:    1
 Semantic Errors:    0
   1jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_2$
```

When see this error pop up, I realized it had to be something with 'optional_variable'. So, looking at the parser.y file I realized it was not being given the option to re-call itself. Thus, I added 'optional_variable' before calling 'variable'. And that seemed to work!

```
optional_variable:
    variable |
    error  ';' |
;
```

➔

```
;

optional_variable:
    optional_variable variable |
    error  ';' |
;
```

And lead to….

```
  16              endreduce;
  17          endreduce;
  18      else
  19          a + b;
  20      endif;
  21  end;

Compiled Sucessfully!
```

Ultimately, project two was full of lessons for me to learn. Some about not rushing things and others about building a compiler. I am really enjoying this project in its entirety and look forward to moving to the next phase.