

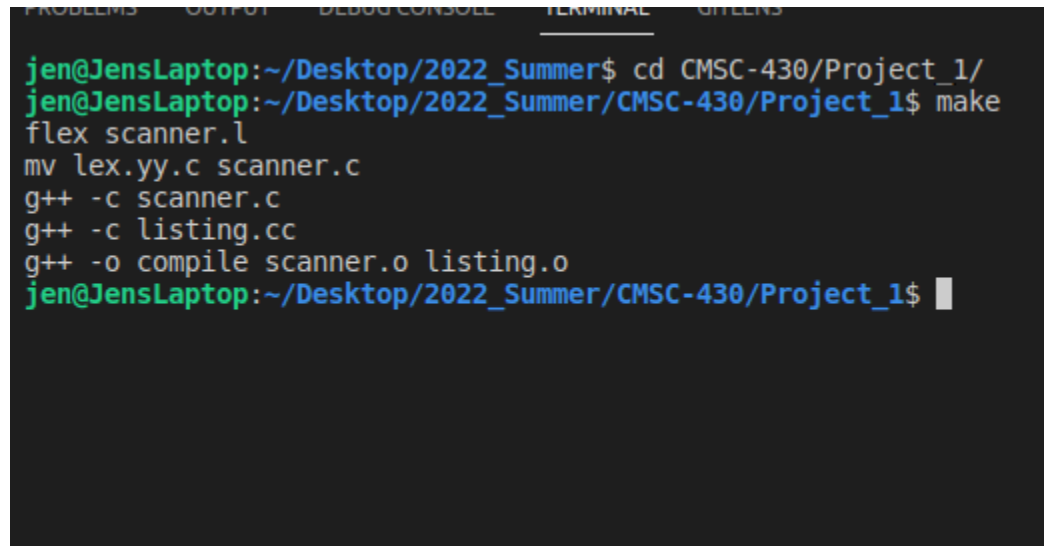
Project Approach

Before I started building out the lexical analyzer portion of my compiler, I first read the required course material, and watched the slides. In order to gain a better understanding, I decided to search YouTube in search of any instructional or tutorial videos that might put the course reading into a more visual perspective. However, I did find the course provided textbooks a bit easier to read oppose to other programming textbooks offered by other courses.

After I had a better understanding of how the lexical analyzer works, how it is implemented, and its basic job functions in the compiler, I started going over the skeleton code of project 1, and building onto what was already there. The instructions provided for project 1 give a pretty clear road map of how to build out the lexical analyzer. Therefore, I mainly just followed project 1's instructions step-by-step. Once everything seemed to be running correctly, and I was getting expected outputs, I decided to re-read some parts of the course readings and see if there was anything that I missed. This actually pushed me to review my code and make it bit more organized, and simplified in some areas.

Test Plan/Cases

To build/compile the project I ran 'make' in the console. Which then built out the required files to be able to run the compile output file. I created four test cases that should make sure the lexical analyzer is working properly.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GREENS
jen@JensLaptop:~/Desktop/2022_Summer$ cd CMSC-430/Project_1/
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ make
flex scanner.l
mv lex.yy.c scanner.c
g++ -c scanner.c
g++ -c listing.cc
g++ -o compile scanner.o listing.o
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$
```

- **Test 1: Testing Real literals and Euler's constant (eE)**

Test1 has two parts, test1.txt and test1_2.txt.

- **Test1.txt** tests the use of real literals, and throws an error when there is no digit after the period. The lexical analyzer will see this as an Invalid Character.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test1.txt
1  -- Function returns 1 Lexical error: a real_lit can not have a trailing '.'
2  -- The value has to have a digit after the period.
3
4  function test1 returns boolean;
5
6  begin
7
8      if 1 + 1.e-9 = 8:
Lexical Error, Invalid Character .
9          returns true;
10         else:
11             returns false;
12         endif;
13     end;
Lexical Errors:    1
Syntax Errors:    0
Semantic Errors:   0
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test1_2.txt
```

- **Test1_2.txt** is the same function but a 2 is inserted after the period. Which will now Compile Successfully!

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test1_2.txt
1  -- Function Compiles Successfully!
2
3  function test1 returns boolean;
4
5  begin
6
7      if 1 + 1.2e-9 = 8:
8          returns true;
9      else:
10         returns false;
11     endif;
12 end;
Compiled Successfully!
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$
```

- **Test 2: Testing Boolean, Logic Operators, and More Math Operators**

For test2.txt, I tested some logic operators (RELOP) and some math operators (ADDOP/MULOP). The function that was written for this test returns a Boolean.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test2.txt
1  -- Function comiles sucessfully; testing boolean, logic ops, and math ops
2
3  function test2 returns boolean;
4    10 / 10 >= 0 and 9 + (-9.0) <= 18 * 0;
5  end;

Compiled Sucessfully!
```

- **Test 3: Testing Mostly All Lexemes**

For this test3.txt, I wanted to try and test all lexemes in the scanner.I file, and make sure they are being read correctly. Also, I tested the second comment type and the ability to correctly use the underscore when naming/identifying a variable. Everything seemed to be working correctly, as I received “Compiled Successfully!” as the output.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test3.txt
1  // Function testing case testing all lexemes.
2
3  function test_3 returns real;
4    x_1: integer is (2 * 3.0) rem 2;
5
6  begin
7    if x_1 then
8
9      case x_1 is
10       when 0 => x_1 - 0;
11       when 1 => x_1 + 1;
12       others => x_1 / 3;
13     endcase;
14
15   else
16     if x_1 >= 3 and not x < 0 then
17       x ** 3;
18     else
19       returns x_1;
20     endif;
21   endif;
22 end;

Compiled Sucessfully!
```

- **Test 4: Testing for Lexical Errors**

Test4.txt is testing for lexical error, and making sure unidentified characters are being caught. In test4.txt, there are a total of 7 lexical errors (trailing '_', '#', '@', '[', ']', '\$', '^'). These characters have not either been defined in the scanner.l file or symbol is in an incorrect location (the trailing underscore), and they should be identified and returned as 'Invalid Character' in the listing.cc file.

```
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$ ./compile < test_cases/test4.txt

1 // Fuction with 7 lexical errors
2
3 function main test4_: integer returns integer;
Lexical Error, Invalid Character _

4     b# integer is a * 2;
Lexical Error, Invalid Character #

5
6     begin@
Lexical Error, Invalid Character @

7
8         if a [=] 0 then
Lexical Error, Invalid Character [
Lexical Error, Invalid Character ]

9             b $ b;
Lexical Error, Invalid Character $

10        else
11            b * b^3;
Lexical Error, Invalid Character ^

12        endif;
13
14    end;

Lexical Errors:    7
Syntax Errors:    0
Semantic Errors:    0
jen@JensLaptop:~/Desktop/2022_Summer/CMSC-430/Project_1$
```

Lessons Learned

Before starting this project, I was very hesitant due to me being unfamiliar with cpp. Before starting this class, I even took some time to go through Codecademy's cpp course, in order to gain a bit more experience and become more comfortable with writing in cpp. Which I think really helped me.

Not only did going through Codecademy's course help, but the textbook provided in this course really broke down how a compiler/interpreter works and what it does for most of our typically used higher level languages. In turn, understanding exactly what the lexical analyzer is doing became clearer. Stating off, I knew the tokens needed to be added to the header and to the lex file, mainly because this was what the instructions said to do. However, when I first read the instructions, I didn't fully understand why they needed to be added or how the files worked together. After reading through most of chapter 2 of our book, I got the main idea of what exactly is going on and how these files work together. For me, programming becomes easier and more natural the more I understand what exactly I am supposed to be building, and how the different parts fit together.

Once I built out the lex file and added the additional tokens to my header file, I started work on the listings.cc file in order to count the number of lexical errors that occur in the test files. I think the biggest confusion of the project happened during this part. Mapping each error value had me a little confused at first, and it took me many read throughs to finally understand how these values are supposed to be populated/incremented. At first, I had thought of just using variables for the lex_err, syntax_err, and sem_err. Then later settled on using an array, as it looked a little cleaner and more simplified. The instructions mentioned using queue, something I have not actually used before in cpp. Therefore, I had done some quick research and find out how to use the queue class in cpp. A good resource I used for this was an article from GeeksForGeeks.com. They discussed a lot about the queue class and how to utilize the front() and back() methods. Then I just had to print out the first element in the queue while the queue was not empty.

In all, it was really interesting to work on, and I really enjoyed getting more familiar with cpp. It has been something I have been really nervous about, but the career path I would like to get into can deal a lot with lower-level languages and some machine level coding. So, this gives me some good experience. Also, being able to understand how a compiler works and write one for a language is also pretty neat.