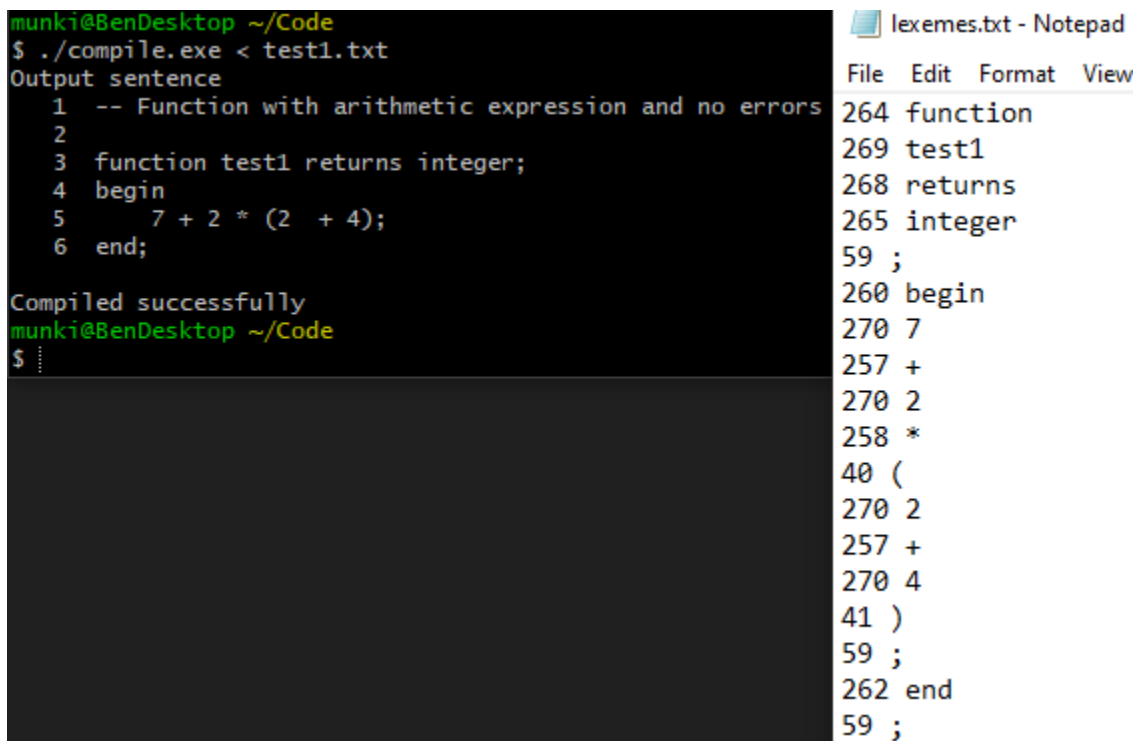


My approach to this project was first figuring out how scanner.I worked. However, this was my first time working with Linux, g++, make, flex, and bison so I had quite a bit of trouble simply getting the skeleton code to run. Once I finally got the skeleton code built and running, I started slowly working away adding the lexemes and their tokens. For the most part, it was very easy to figure out by referencing the skeleton code. Th second comment was a little more difficult, but after figuring out the regex it wasn't too hard. The regex for the real_literal required the most work by far. Changing the code to allow multiple errors from one line and total number of errors was pretty straightforward.

Some improvements to the program would be adding a way to go into more detail about what error occurred. For example, "failed to parse real_literal or identifier with underscores" rather than simply ignoring the problematic character and parsing the rest of it as an identifier. I'm assuming smarter errors will be included in future projects, but I'll find out I suppose. Overall, I learned much about Linux, Flux, and how Compilers "think"/work, but there's much more to look forward to throughout the rest of the class. Test cases in the form of figures of the program running and proof of parsed lexemes starts below:



```
munki@BenDesktop ~/Code
$ ./compile.exe < test1.txt
Output sentence
1  -- Function with arithmetic expression and no errors
2
3  function test1 returns integer;
4  begin
5      7 + 2 * (2  + 4);
6  end;

Compiled successfully
munki@BenDesktop ~/Code
$ .....
```

```
lexemes.txt - Notepad
File Edit Format View
264 function
269 test1
268 returns
265 integer
59 ;
260 begin
270 7
257 +
270 2
258 *
40 (
270 2
257 +
270 4
41 )
59 ;
262 end
59 ;
```

Figure 1. Running a file with no errors.

```

$ ./compile.exe < test2.txt
Output sentence
1  -- Function with two lexical errors
2
3  function test2 returns integer;
4  begin
5      7 $ 2 ^ (2 + 4);
Lexical Error, Invalid Character $
Lexical Error, Invalid Character ^
6  end;

Lexical errors: 2
Syntax errors: 0
Semantic errors: 0
munki@BenDesktop ~/Code
$

```

lexemes.txt - Notepad

File	Edit	Format	View
264	function		
269	test2		
268	returns		
265	integer		
59	;		
260	begin		
270	7		
270	2		
40	(
270	2		
257	+		
270	4		
41)		
59	;		
262	end		
59	;		

Figure 2. Running a file with multiple errors on the same line.

```

munki@BenDesktop ~/Code
$ ./compile.exe < test3.txt
Output sentence
1  // Second comment type and underscore identifier
2
3  under_score
4  _underscore //This will make an error
Lexical Error, Invalid Character _
5  underscore_ //This will make an error
Lexical Error, Invalid Character _
6  under__score //This will make two errors
Lexical Error, Invalid Character _
Lexical Error, Invalid Character _
7
8  // Real literal (first line will run fine)
9
10 12.3e+456 9.7E-865 12.3e456 9.e+876 12. 98.7
11 .e123 //This will not parse as a real literal
Lexical Error, Invalid Character .
12 9.7e //This will not parse as a real literal

Lexical errors: 5
Syntax errors: 0
Semantic errors: 0

```

lexemes.txt - Notepad

File	Edit	Format	View
269	under_score		
269	underscore		
269	underscore		
269	under		
269	score		
285	12.3e+456		
285	9.7E-865		
285	12.3e456		
285	9.e+876		
285	12.		
285	98.7		
269	e123		
285	9.7		
269	e		

Figure 3. Showing second comment, underscores in identifier, and real literals.

The image shows a terminal window on the left and a Notepad window on the right. The terminal window displays the output of a compilation process, listing various lexemes and their corresponding line numbers in a source file. The Notepad window shows the same list of lexemes, with each line prefixed by a line number from 44 to 286.

```

munki@BenDesktop ~/Code
$ ./compile.exe < test4.txt
Output sentence
1  -- Punctuation symbols
2
3  ,;()
4
5  -- Identifier
6
7  name name123
8
9  -- Literals
10
11 123
12
13 -- Logical operators
14
15 and or not
16
17 -- Relational operators
18
19 < = /= > >= <= =>
20
21 -- Arithmetic operator
22
23 +*-* */
24
25 -- Reserved words
26
27 begin boolean end endreduce function integer is reduce return and case
   else endcase endif if others real then when or not rem true false
Compiled successfully
munki@BenDesktop ~/Code
$

```

lexemes.txt - Notepad

File	Edit	Format	View
44	,		
59	;		
40	(
41)		
269	name		
269	name123		
270	123		
259	and		
281	or		
282	not		
256	<		
256	=		
256	/=		
256	>		
256	>=		
256	<=		
271	=>		
257	+		
258	*		
257	-		
284	**		
258	/		
260	begin		
261	boolean		
262	end		
263	endreduce		
264	function		
265	integer		
266	is		
267	reduce		
269	return		
259	and		
272	case		
273	else		
274	endcase		
275	endif		
276	if		
277	others		
278	real		
279	then		
280	when		
281	or		
282	not		
283	rem		
286	true		
286	false		

Figure 4. Showcasing the rest of the language and all lexemes that were added.