

Project 1

Approach to the project:

Firstly, a Debian Linux distribution was chosen as the main OS, because of the easy installation of FLEX on the system. As a help with FLEX, I used flex & bison book by John R. Levine which has very nice, and easy to follow along, practical examples on the topic to help me get a grasp on how to write the FLEX rules and regexes. Also, some programming research was done in order to find the queue data structure which is very convenient in our case since we want to print multiple errors for each line (if there are) and in the correct order which is FIFO.

Writing the rules:

Writing the rules, was an easy task, just had to correct an error I firstly didn't notice at the regex for the Real literal:

`{digit}+(.){digit}*([eE][+-]?{digit}+)*`

The ***** was a mistake because in this case the real literal could have 0 OR MORE exponentials which in the case of more than one would be a mistake. So I changed the ***** with **?**, which solved my problem because now we can only have 0 OR 1 exponentials in the real literal. Now I have the correct regex which is:

`{digit}+(.){digit}*([eE][+-]?{digit}+)?`

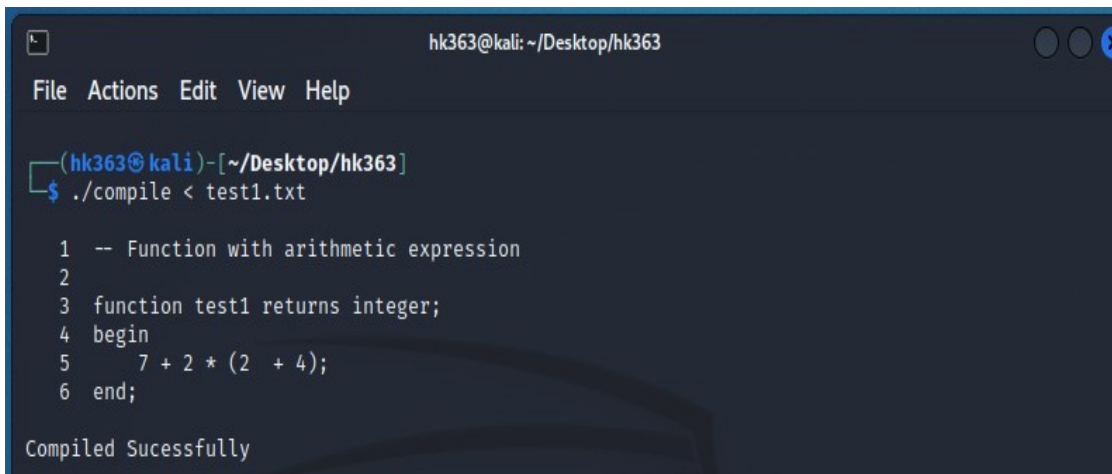
Running the tests:

Test 1:

```
-- Function with arithmetic expression

function test1 returns integer;
begin
    7 + 2 * (2 + 4);
end;
```

Results of test 1:

A terminal window titled 'hk363@kali: ~/Desktop/hk363' with a menu bar (File, Actions, Edit, View, Help). The prompt is '(hk363@kali)-[~/Desktop/hk363]'. The command './compile < test1.txt' has been executed. The output shows the source code of test1.txt with line numbers 1 through 6. At the bottom, it says 'Compiled Sucessfully' (note the typo).

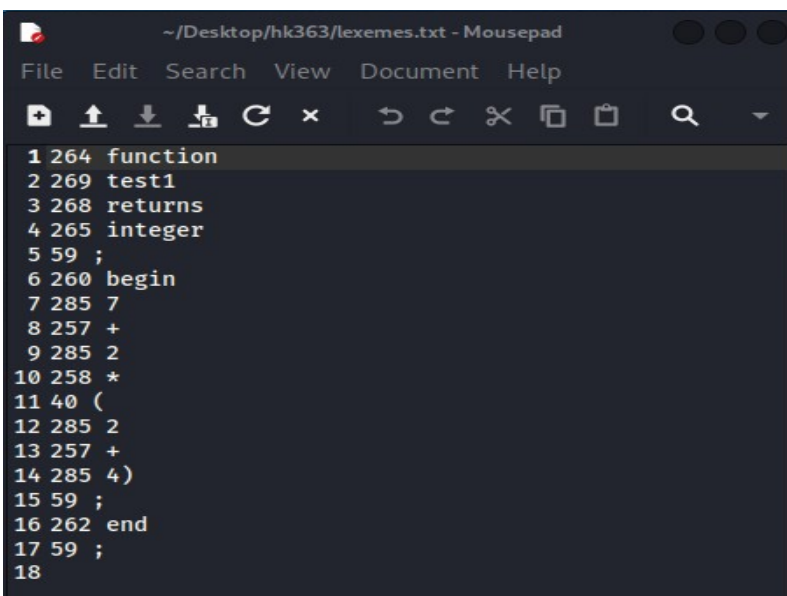
```
hk363@kali: ~/Desktop/hk363
File Actions Edit View Help

(hk363@kali)-[~/Desktop/hk363]
$ ./compile < test1.txt

1 -- Function with arithmetic expression
2
3 function test1 returns integer;
4 begin
5     7 + 2 * (2 + 4);
6 end;

Compiled Sucessfully
```

Lexemes of Test 1:

A Mousepad window titled '~/Desktop/hk363/lexemes.txt - Mousepad' with a menu bar (File, Edit, Search, View, Document, Help) and a toolbar. The text shows the lexemes for each token in the test1.txt file, with line numbers and column numbers. The lexemes are: function, test1, returns, integer, ;, begin, 7, +, 2, *, (, 2, +, 4,), ;, end, ;.

```
~/Desktop/hk363/lexemes.txt - Mousepad
File Edit Search View Document Help

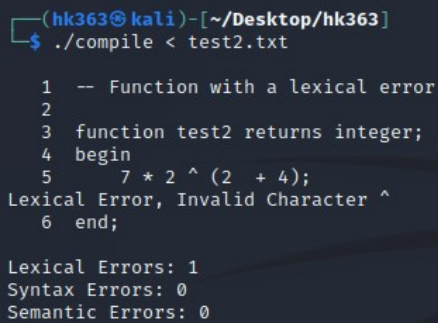
1 264 function
2 269 test1
3 268 returns
4 265 integer
5 59 ;
6 260 begin
7 285 7
8 257 +
9 285 2
10 258 *
11 40 (
12 285 2
13 257 +
14 285 4)
15 59 ;
16 262 end
17 59 ;
18
```

Test 2:

```
-- Function with a lexical error

function test2 returns integer;
begin
    7 * 2 ^ (2 + 4);
end;
```

Results of Test 2:

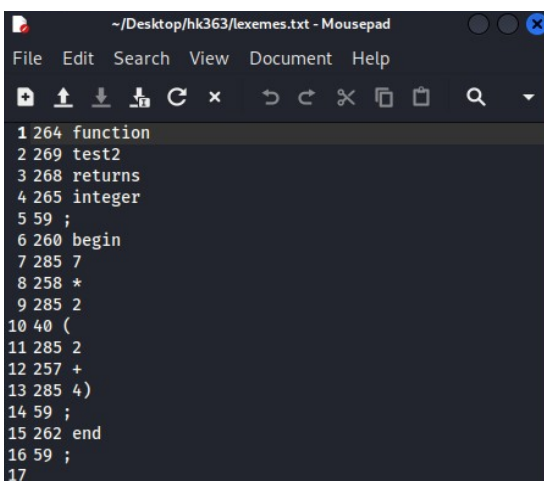
A terminal window with a dark background. The prompt is (hk363@kali)~[~/Desktop/hk363]. The command ./compile < test2.txt has been executed. The output shows the source code lines 1 through 6, with a message 'Lexical Error, Invalid Character ^' pointing to the space before the closing parenthesis in the exponentiation operation. At the bottom, it reports 'Lexical Errors: 1', 'Syntax Errors: 0', and 'Semantic Errors: 0'.

```
(hk363@kali)~[~/Desktop/hk363]
$ ./compile < test2.txt

1 -- Function with a lexical error
2
3 function test2 returns integer;
4 begin
5     7 * 2 ^ (2 + 4);
Lexical Error, Invalid Character ^
6 end;

Lexical Errors: 1
Syntax Errors: 0
Semantic Errors: 0
```

Lexemes of Test 2:

A screenshot of a text editor window titled '~/Desktop/hk363/lexemes.txt - Mousepad'. The window shows a list of lexemes with their corresponding line numbers from the source code. The lexemes are: function (line 1), test2 (line 2), returns (line 3), integer (line 4), ; (line 5), begin (line 6), 7 (line 7), * (line 8), 2 (line 9), ((line 10), 40 (line 11), 2 (line 12), + (line 13), 285 (line 13), 4) (line 13), ; (line 14), end (line 15), ; (line 16), and a blank line (line 17).

```
1 264 function
2 269 test2
3 268 returns
4 265 integer
5 59 ;
6 260 begin
7 285 7
8 258 *
9 285 2
10 40 (
11 285 2
12 257 +
13 285 4)
14 59 ;
15 262 end
16 59 ;
17
```

Test 3:

```
-- Punctuation symbols
,;()

-- Identifier
name name123

-- Literals
123

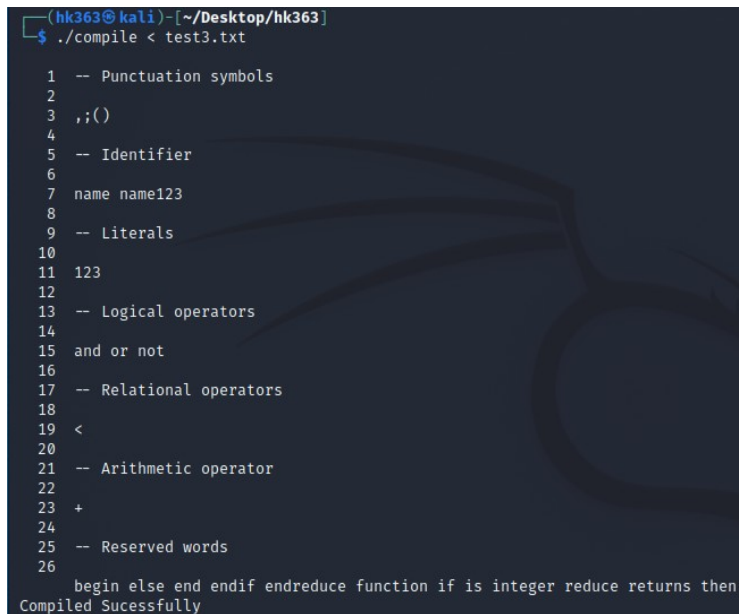
-- Logical operators
and or not

-- Relational operators
<

-- Arithmetic operator
+

-- Reserved words
begin else end endif endreduce function if is integer reduce returns then
```

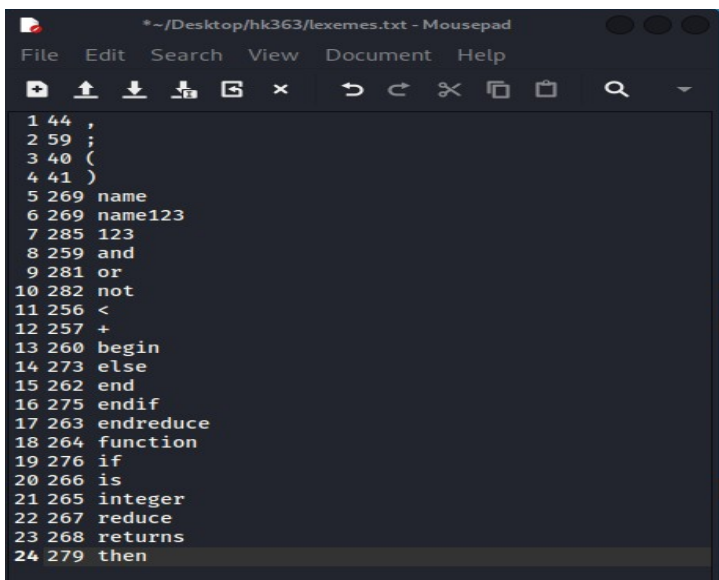
Results of Test 3:



```
(hk363@kali)~[~/Desktop/hk363]
$ ./compile < test3.txt

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
    begin else end endif endreduce function if is integer reduce returns then
Compiled Successfully
```

Lexemes of Test 3:



```
*~/Desktop/hk363/lexemes.txt - Mousepad
File Edit Search View Document Help

1 44 ,
2 59 ;
3 40 (
4 41 )
5 269 name
6 269 name123
7 285 123
8 259 and
9 281 or
10 282 not
11 256 <
12 257 +
13 260 begin
14 273 else
15 262 end
16 275 endif
17 263 endreduce
18 264 function
19 276 if
20 266 is
21 265 integer
22 267 reduce
23 268 returns
24 279 then
```

Test 4:

This is a custom made test taken from the description of project1

```
(* Program with no errors *)

begin
7 + 2 > 6 and 8 = 5 * (7 - 4);
end;
```

Results of Test 4:



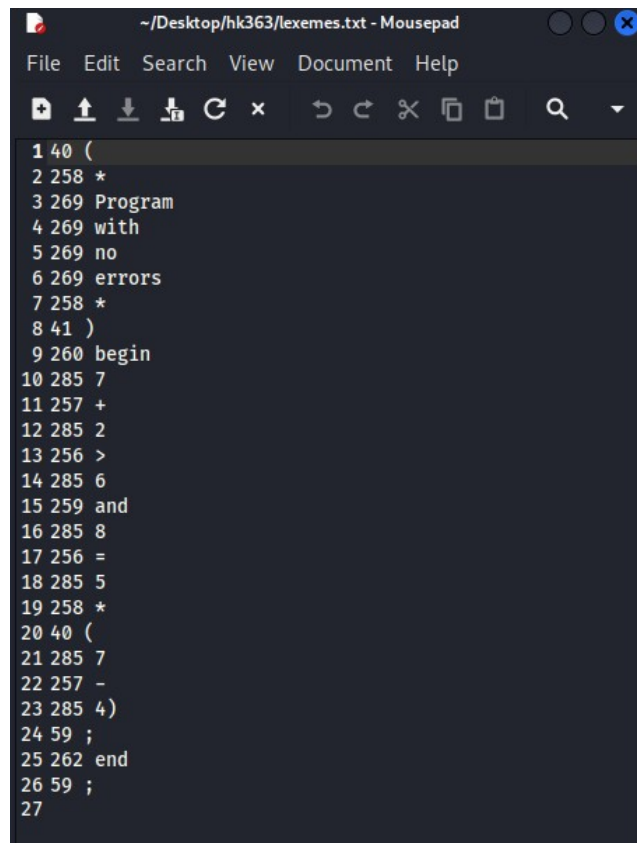
A screenshot of a terminal window titled 'hk363@kali: ~/Desktop/hk363'. The window shows the execution of a command to compile a file named 'test4.txt'. The output displays the source code of the file, which is a PL/SQL block with a comment, a 'begin' statement, a logical expression, and an 'end;' statement. The final output is 'Compiled Sucessfully'.

```
hk363@kali: ~/Desktop/hk363
File Actions Edit View Help

(hk363@kali)~[~/Desktop/hk363]
$ ./compile < test4.txt

1  (* Program with no errors *)
2
3
4  begin
5  7 + 2 > 6 and 8 = 5 * (7 - 4);
   end;
Compiled Sucessfully
```

Lexemes of Test 4:



A screenshot of a text editor window titled '~/Desktop/hk363/lexemes.txt - Mousepad'. The window displays the lexemes of the code from the previous block, with each token on a new line and its corresponding line and column numbers. The tokens include parentheses, asterisks, the word 'Program', 'with', 'no', 'errors', 'begin', the logical expression, and 'end;'.

```
1 40 (
2 258 *
3 269 Program
4 269 with
5 269 no
6 269 errors
7 258 *
8 41 )
9 260 begin
10 285 7
11 257 +
12 285 2
13 256 >
14 285 6
15 259 and
16 285 8
17 256 =
18 285 5
19 258 *
20 40 (
21 285 7
22 257 -
23 285 4)
24 59 ;
25 262 end
26 59 ;
27
```

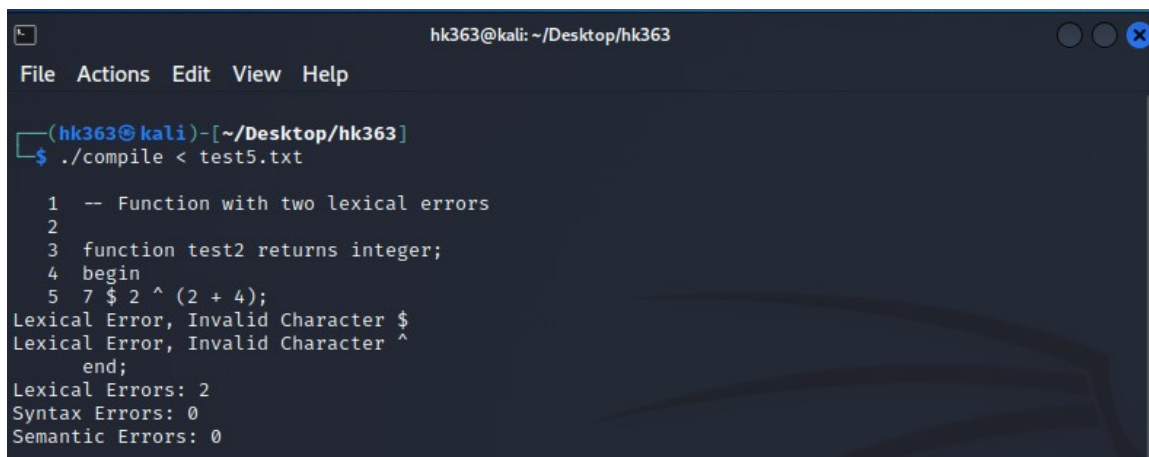
Test 5:

This is a custom made test taken from the description of project 1

```
-- Function with two lexical errors
```

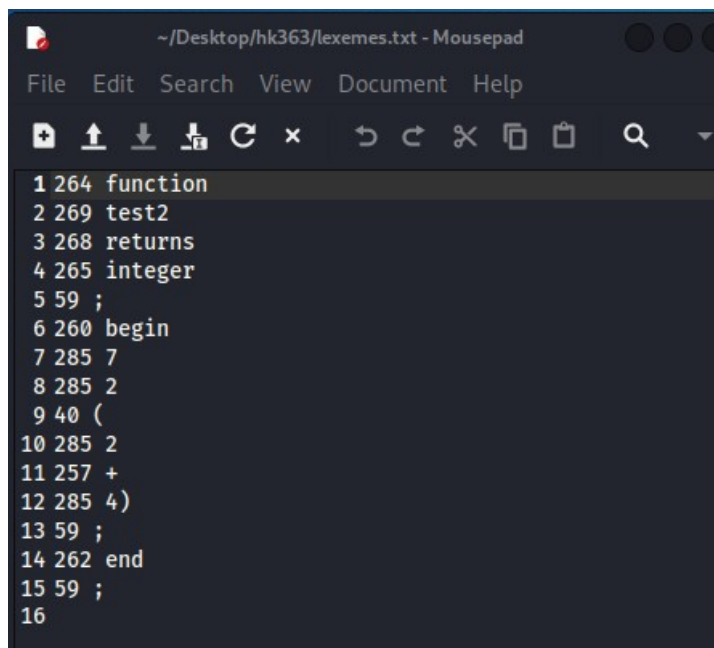
```
function test2 returns integer;  
begin  
7 $ 2 ^ (2 + 4);  
end;
```

Results of Test 5:



```
hk363@kali: ~/Desktop/hk363  
File Actions Edit View Help  
(hk363@kali)-[~/Desktop/hk363]  
$ ./compile < test5.txt  
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 ^  
end;  
Lexical Errors: 2  
Syntax Errors: 0  
Semantic Errors: 0
```

Lexemes of Test 5:



```
~/Desktop/hk363/lexemes.txt - Mousepad  
File Edit Search View Document Help  
1 264 function  
2 269 test2  
3 268 returns  
4 265 integer  
5 59 ;  
6 260 begin  
7 285 7  
8 285 2  
9 40 (  
10 285 2  
11 257 +  
12 285 4)  
13 59 ;  
14 262 end  
15 59 ;  
16
```

Test 6:

This is a custom made test, which includes testing of all the new lexemes the project description required:

```
//Testing the new punctuation symbol: =>
=>

//Testing the reserved keywords:
begin else end endif endreduce function if is integer reduce returns then case endcase real when others

//Testing OROP and NOTOP:
or not

//Testing RELOP:
= /= > >= <=

//Testing ADDOP:
-

//Testing MULOP:
/

//Testing other arithmetic operators:
* +

//Testing REMOP:
rem

//Testing EXPOP:
**

//Testing comments starting with backslashes
//Testing different IDs with underscores:
a_b
abc_d
ab__c
ab__c
_ab
ab_

//Testing real literal token:
1.15 5.0e1 3.0e+4 8.3E45

//Testing boolean literal:
true false
```

Results of Test 6:

```
(hk363@kali) ~/Desktop/hk363
$ ./compile <test6.txt

1 //Testing the new punctuation symbol: =>
2 =>
3
4
5 //Testing the reserved keywords:
6 begin else end endif endreduce function if is integer reduce returns then case endcase real when others
7
8
9 //Testing OROP and NOTOP:
10 or not
11
12
13 //Testing RELOP:
14 = /= > >= <=
15
16
17 //Testing ADDOP:
18 -
19
20
21 //Testing MULOP:
22 /
23
24
25 //Testing other arithmetic operators:
26 * +
27
28
29 //Testing REMOP:
30 rem
31
32
33 //Testing EXPOP:
34 **
35
36
37 //Testing comments starting with backslashes
38
39
40 //Testing different IDs with underscores:
41 a_b
42 abc_d
43 ab__c
Lexical Error, Invalid Character _
Lexical Error, Invalid Character _
44 ab__c
Lexical Error, Invalid Character _
Lexical Error, Invalid Character _
Lexical Error, Invalid Character _
45 _ab
```

```

Lexical Error, Invalid Character _
46  ab_
Lexical Error, Invalid Character _
47
48
49 //Testing real literal token:
50 1.15 5.0e1 3.0e+4 8.3E45
51
52
53 //Testing boolean literal:
54 true false
55
56

Lexical Errors: 7
Syntax Errors: 0
Semantic Errors: 0

```

Lexemes of Test 6:

```

*~/Desktop/hk363/lexemes.txt - Mousepad
File Edit Search View Document Help
1 271 =>
2 260 begin
3 273 else
4 262 end
5 275 endif
6 263 endreduce
7 264 function
8 276 if
9 266 is
10 265 integer
11 267 reduce
12 268 returns
13 279 then
14 272 case
15 274 endcase
16 278 real
17 280 when
18 277 others
19 281 or
20 282 not
21 256 =
22 256 /=
23 256 >
24 256 >=
25 256 <=
26 257 -
27 258 /
28 258 *
29 257 +
30 283 rem
31 284 **
32 269 a_b
33 269 abc_d
34 269 ab
35 269 c
36 269 ab
37 269 c
38 269 ab
39 269 ab
40 285 3.54
41 285 3.0e1
42 285 2.0e+12
43 285 3.5E45
44 286 true
45 286 false

```

We can clearly see from this test that consecutive or leading or trailing underscores are successfully recognized as error.

Also the real literal tokens are correctly recognized as well as the `//` comments.

Finally, all the other lexemes are also correctly recognized.

Conclusion

With those 6 tests we have checked every aspect of our lexical analyzer with great results. All of the tests have returned the desirable outcome and recognized all the lexemes successfully.

Some improvements in the code can be made:

- Add a color only to the errors so they are more easily separable from the original program. This can be done by adding `"\033[0;35m"` inside the printf function in the displayErrors() function. This will result in printing the errors in magenta color on the terminal.
- Misspelled keywords are not recognized as an error but they are recognized as an ID. Firstly, the parser will look if the keyword we typed (for example **caase** instead of **case**) is a keyword. Since we misspelled **caase** the parser will not be able to recognize that we intended to write **case** and return an error, but will continue to check if what we wrote matches any other rule. Finally, it will reach the rule for ID, which is valid since **caase** matches the rule. This will be solved in the next stage of the compiler when will check for syntax errors. So, at the lexical analysis stage this is not an error.