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:

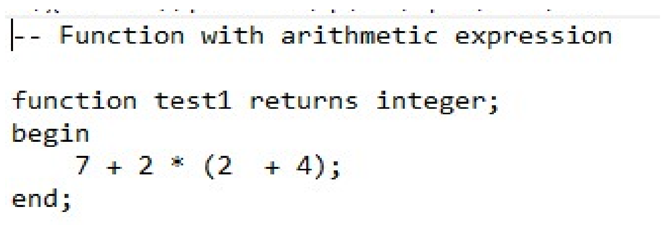
{digit}+(.){digit}*([eE}[+-]?{digit}+)*

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}+)?

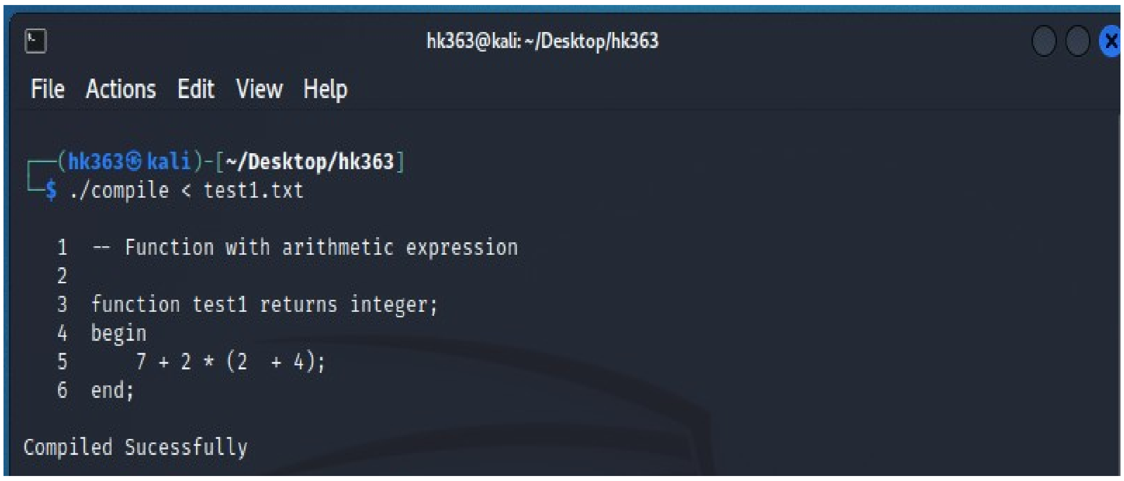
Running the tests:

Test 1:



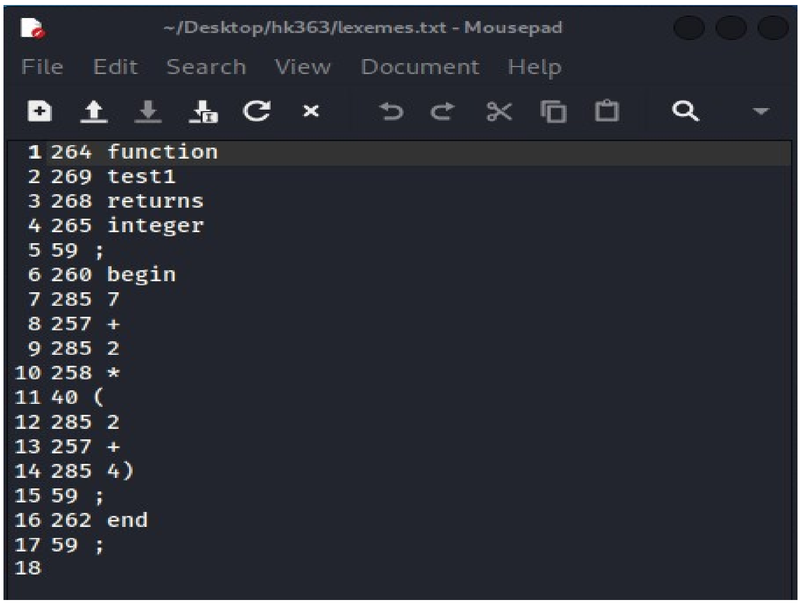
```
-- Function with arithmetic expression

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

Results of test 1:



```
┌──(hk363㉿kali)-[~/Desktop/hk363]
└─$ ./compile < test1.txt

-- Function with arithmetic expression

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

Compiled Sucessfully
```

Lexemes of Test 1:



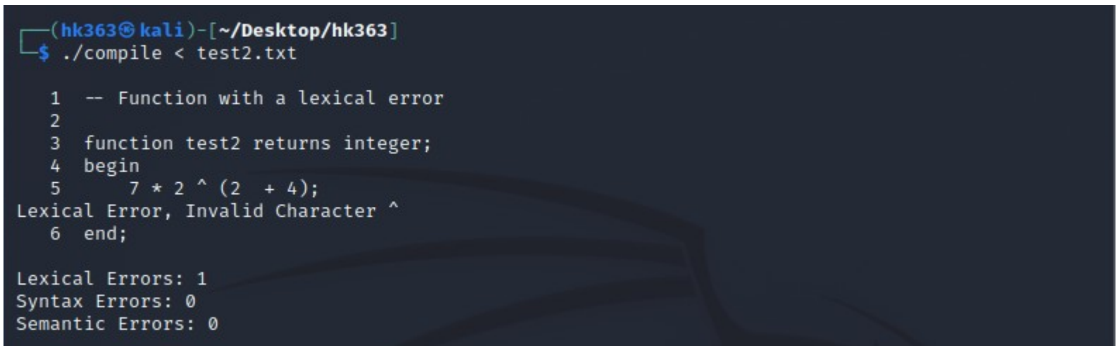
~/Desktop/hk363/lexemes.txt - Mousepad
File Edit Search View Document Help

```
264 function
269 test1
268 returns
265 integer
59 ;
260 begin
285 7
257 +
285 2
258 *
40 (
285 2
257 +
285 4)
59 ;
262 end
59 ;
```

Test 2:

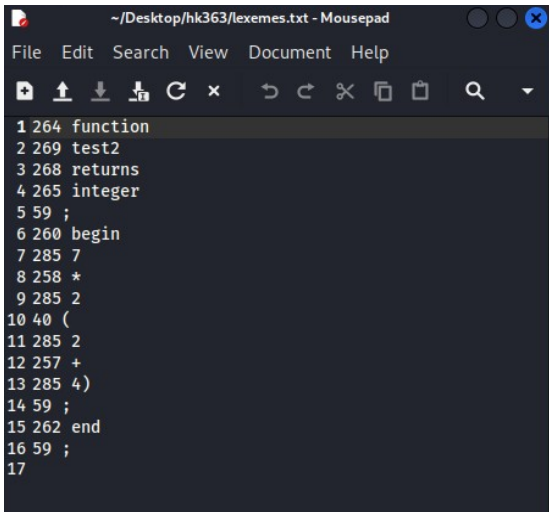
```
-- Function with a lexical error

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

Results of Test 2:



```
┌──(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
```

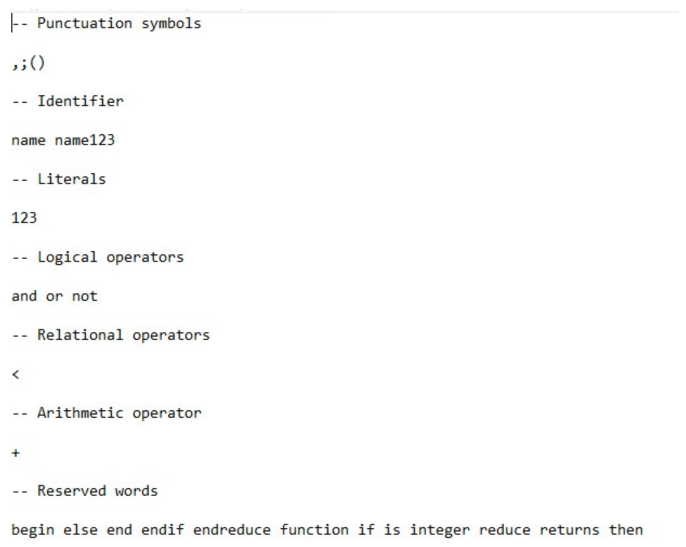
Lexemes of Test 2:



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

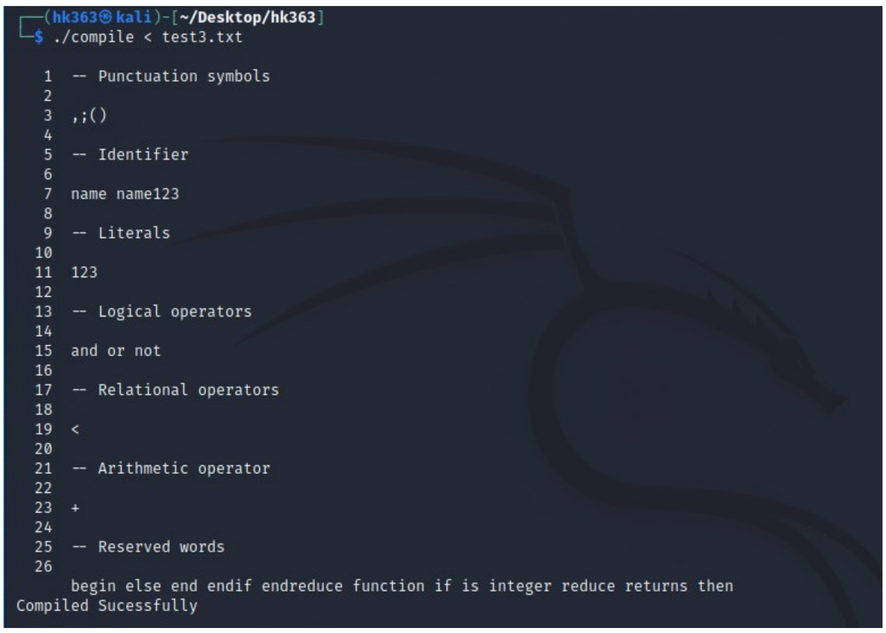
```
264 function
269 test2
268 returns
265 integer
59 ;
260 begin
285 7
258 *
285 2
40 (
285 2
257 +
285 4)
59 ;
262 end
59 ;
```

Test 3:



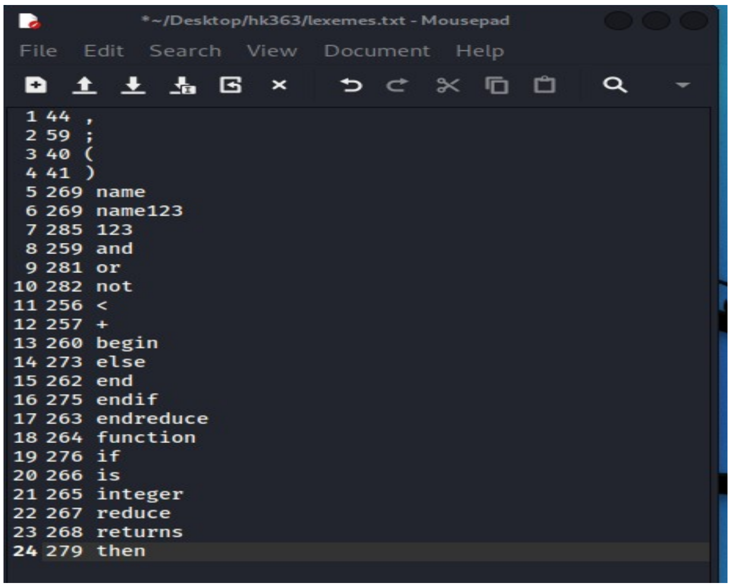
```
-- Punctuation symbols

,;()

-- 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 Sucessfully
```

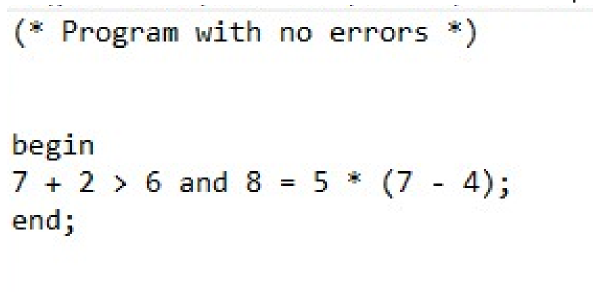
Lexemes of Test 3:



```
44 ,
59 ;
40 (
41 )
269 name
269 name123
285 123
259 and
281 or
282 not
256 <
257 +
260 begin
273 else
262 end
275 endif
263 endreduce
264 function
276 if
266 is
265 integer
267 reduce
268 returns
279 then
```

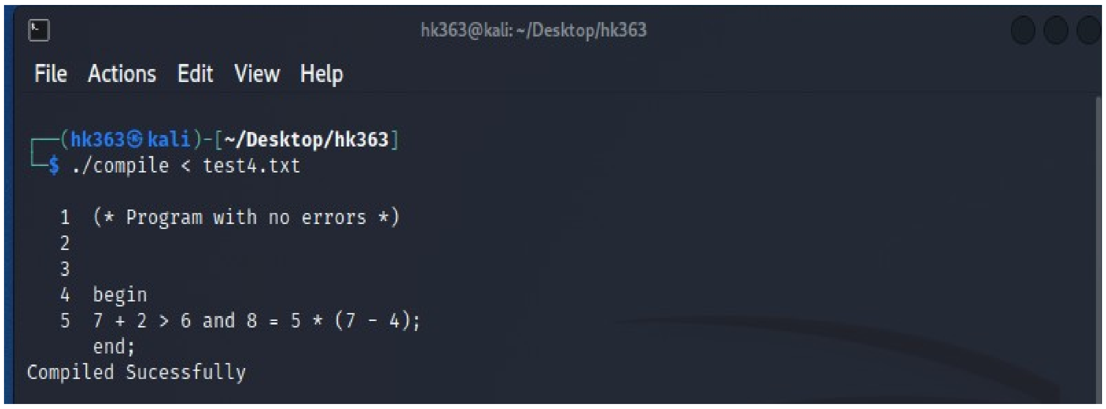
Test 4:

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



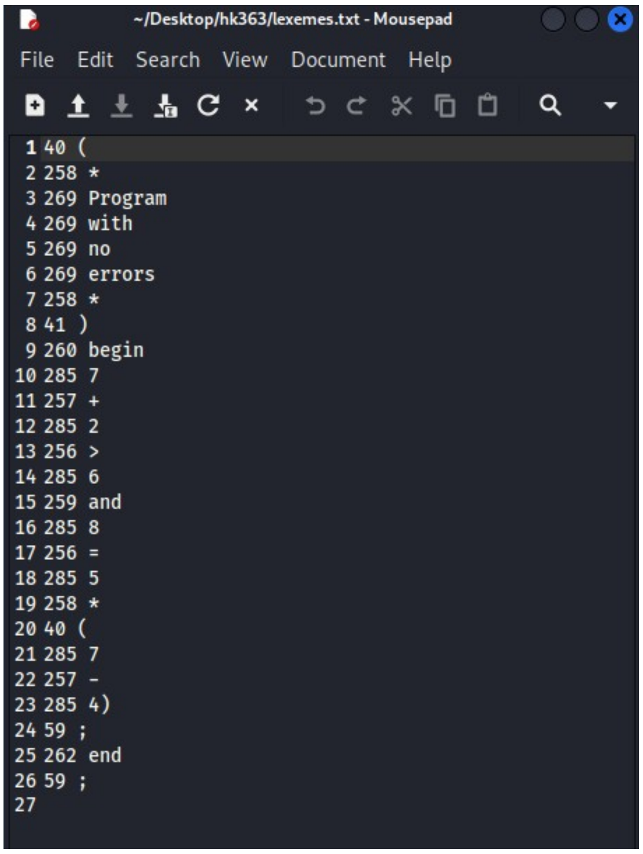
```
(* Program with no errors *)


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

Results of Test 4:



```
┌──(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
```

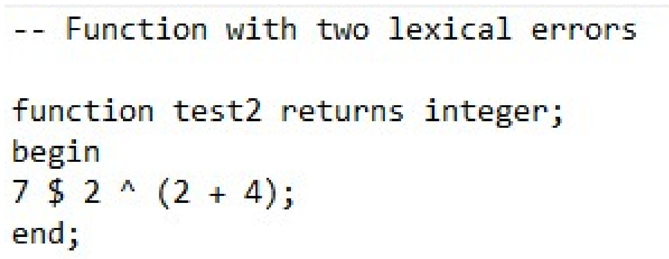
Lexemes of Test 4:


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

```
40 (
258 *
269 Program
269 with
269 no
269 errors
258 *
41 )
260 begin
285 7
257 +
285 2
256 >
285 6
259 and
285 8
256 =
285 5
258 *
40 (
285 7
257 -
285 4)
59 ;
262 end
59 ;
```

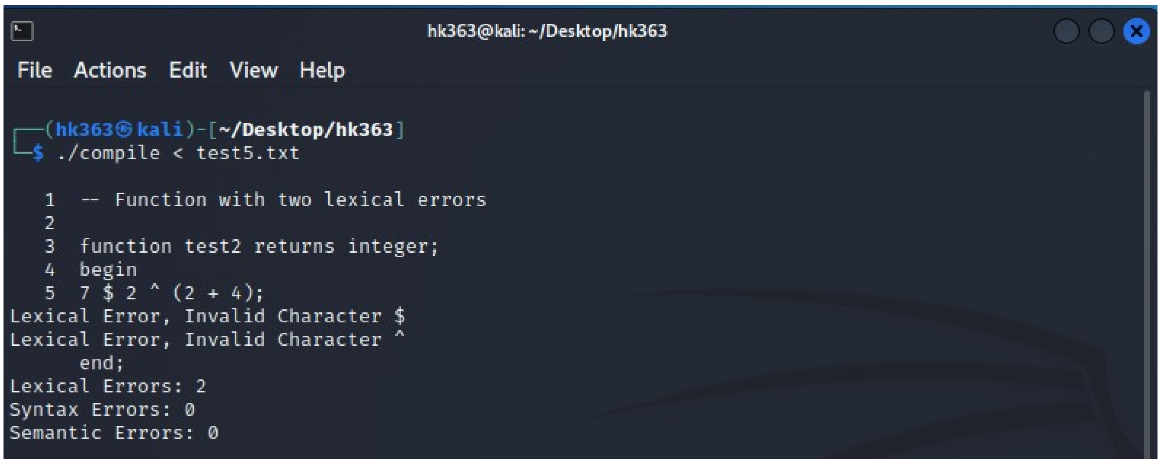
Test 5:

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



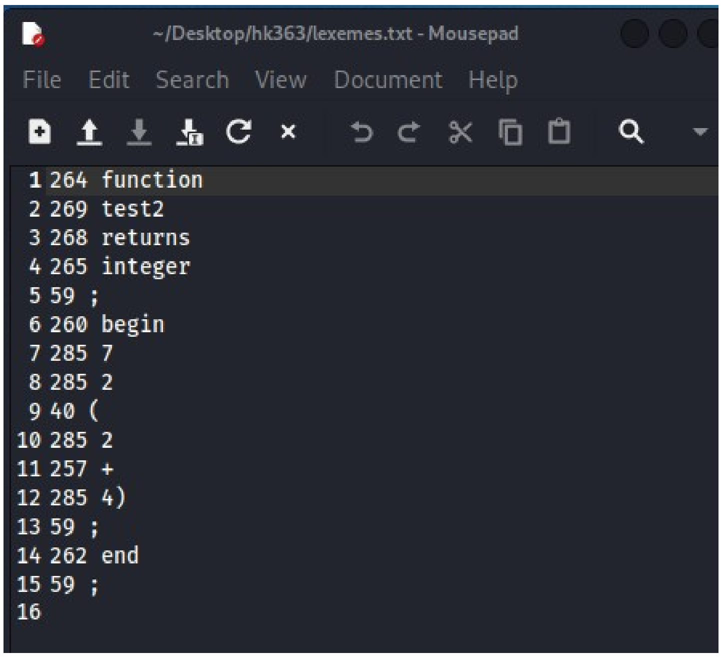
```
-- Function with two lexical errors

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

Results of Test 5:



```
┌──(hk363㉿kali)-[~/Desktop/hk363]
└─$ ./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
```

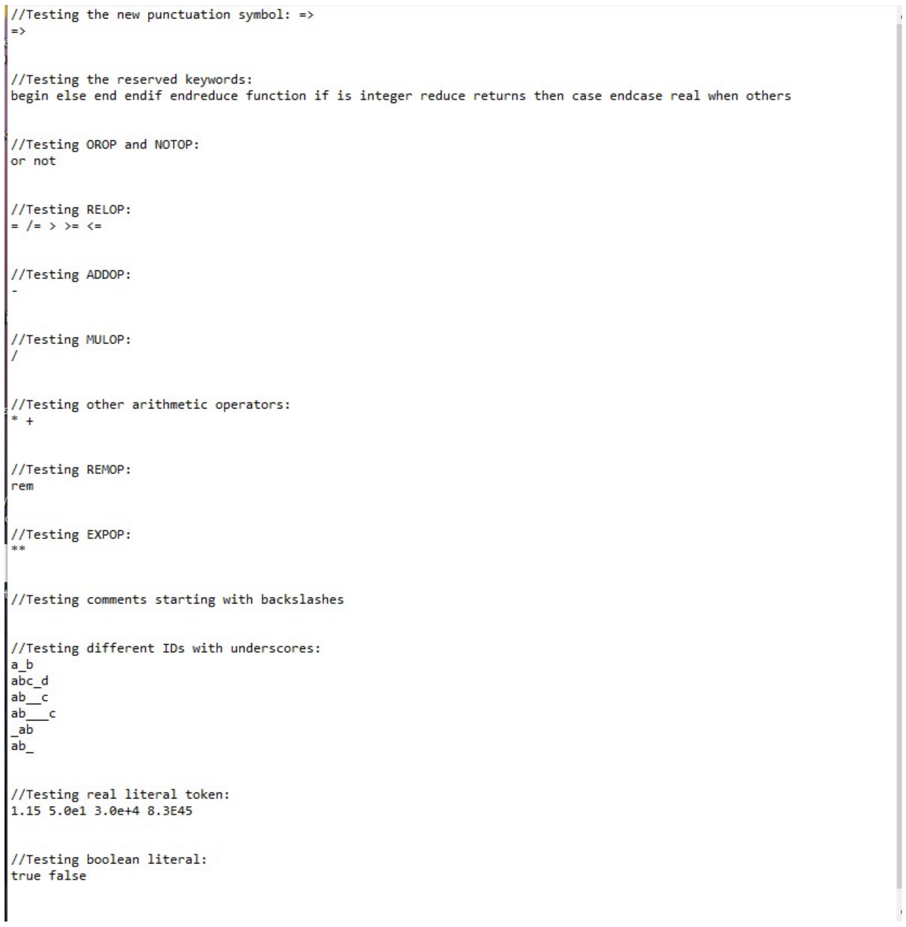
Lexemes of Test 5:


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

```
264 function
269 test2
268 returns
265 integer
59 ;
260 begin
285 7
285 2
40 (
285 2
257 +
285 4)
59 ;
262 end
59 ;
```

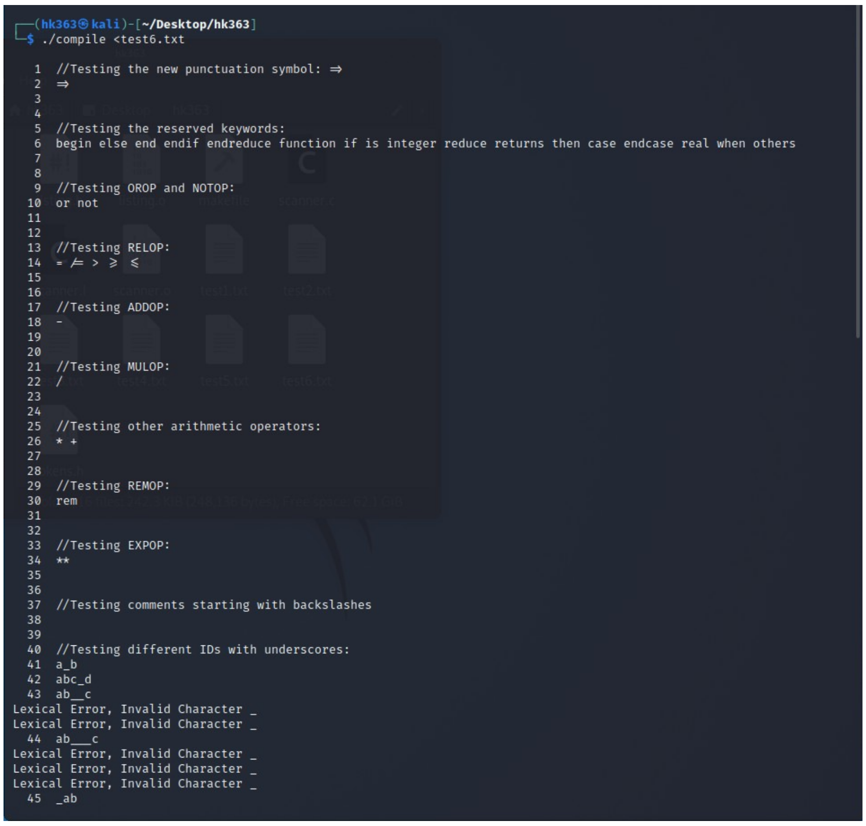
Test 6:

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

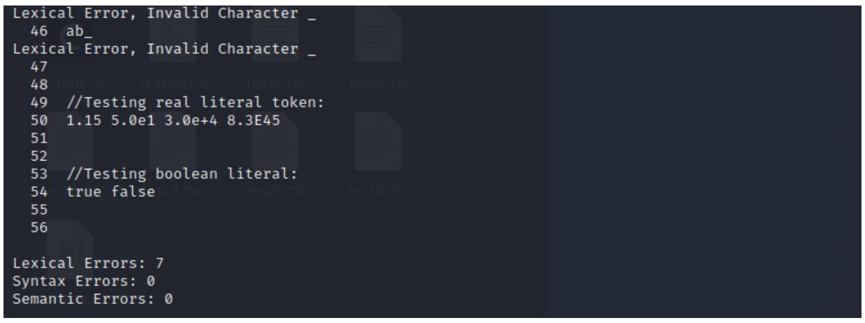


```
//Testing the new punctuation symbol: =>
=>


//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:

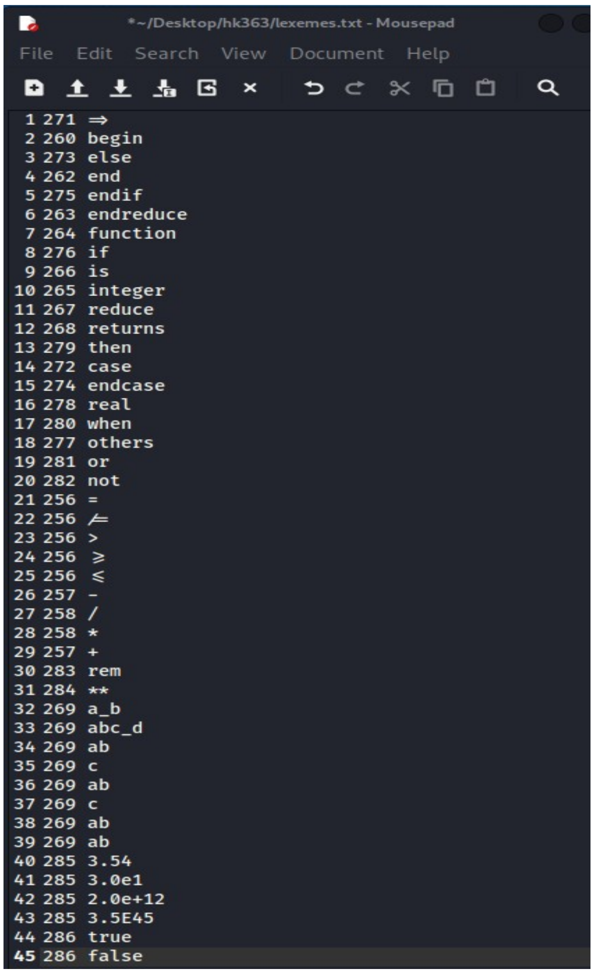


```
┌──(hk363㉿kali)-[~/Desktop/hk363]
└─$ ./compile <test6.txt

   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

```
271 ⇒
260 begin
273 else
262 end
275 endif
263 endreduce
264 function
276 if
266 is
265 integer
267 reduce
268 returns
279 then
272 case
274 endcase
278 real
280 when
277 others
281 or
282 not
256 =
256 /=
256 >
256 ⩾
256 ⩽
257 -
258 /
258 *
257 +
283 rem
284 **
269 a_b
269 abc_d
269 ab
269 c
269 ab
269 c
269 ab
269 ab
285 3.54
285 3.0e1
285 2.0e+12
285 3.5E45
286 true
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.