Project 2

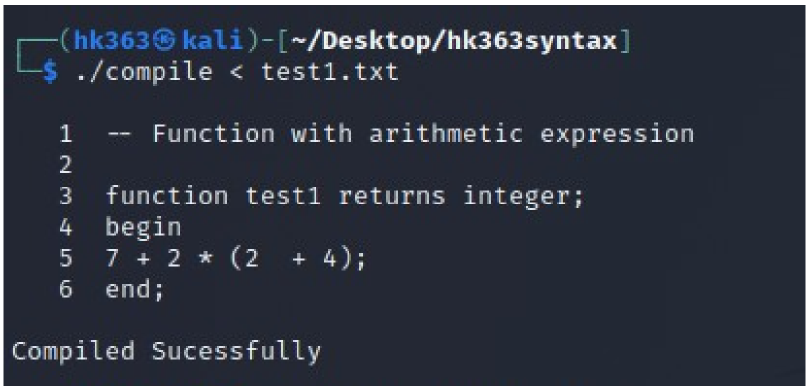
Approach to the project:

Some important things I noticed about how I need to implement the parser rules were:

- All operators are left associative except the exponent operator (EXPOP)
- The parsing rules were implemented based on the order of precedence of the operators which is ***NOTOP, EXPOP, MULOP, REMOP. ADDOP, RELOP, ANDOP, OROP***.
- The Bison shouldn't produce warnings about shift/reduce and reduce/reduce errors which is case in this project and it was achieved after using the parameter '-Wcounterexamples' which was very helpful in finding and correcting the warnings.
- Also, I added \\. to the real literal token from the lexical analyzer changing it from
  {digit}+.{digit}*([eE}[+-]?{digit}+)*
  to
  {digit}+\\.{digit}*([eE}[+-]?{digit}+)*

Running the tests:

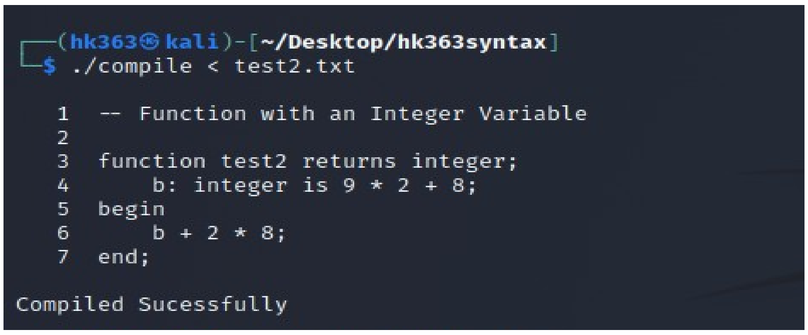
Test 1:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./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
```

Test 1 compiled successfully, as it should, without any syntax or lexical errors → correct result

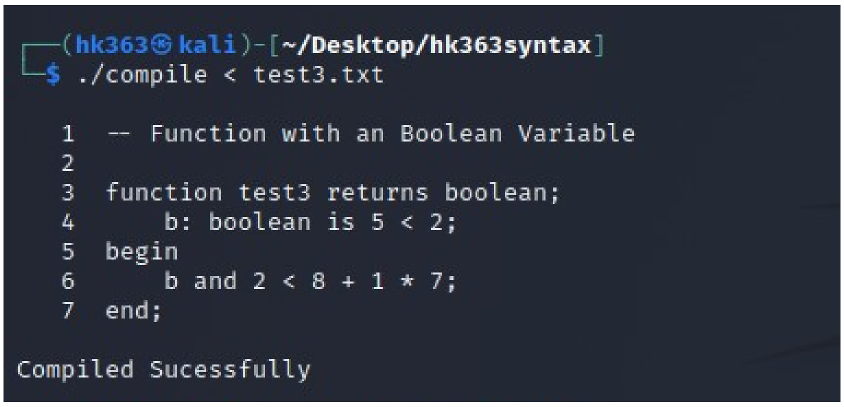
Test 2:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test2.txt

   1  -- Function with an Integer Variable
   2
   3  function test2 returns integer;
   4      b: integer is 9 * 2 + 8;
   5  begin
   6      b + 2 * 8;
   7  end;

Compiled Sucessfully
```

Test 2 compiled successfully, as it should, without any syntax or lexical errors → correct result

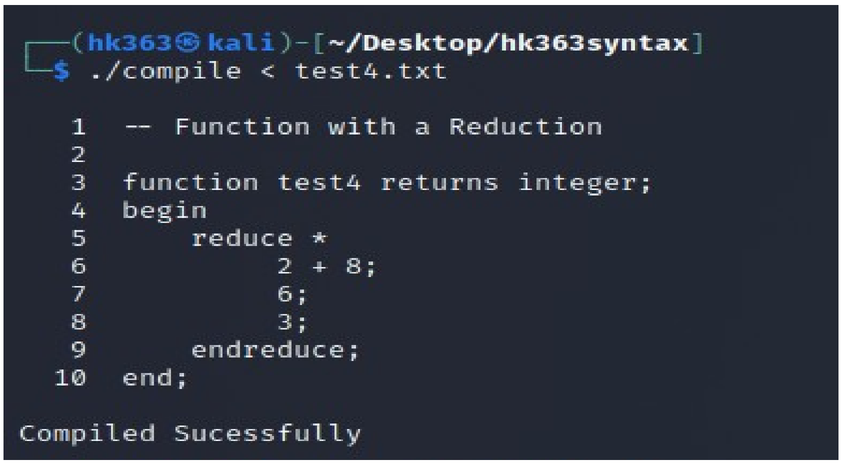
Test 3:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test3.txt

   1  -- Function with an Boolean Variable
   2
   3  function test3 returns boolean;
   4      b: boolean is 5 < 2;
   5  begin
   6      b and 2 < 8 + 1 * 7;
   7  end;

Compiled Sucessfully
```

Test 3 compiled successfully, as it should, without any syntax or lexical errors → correct result

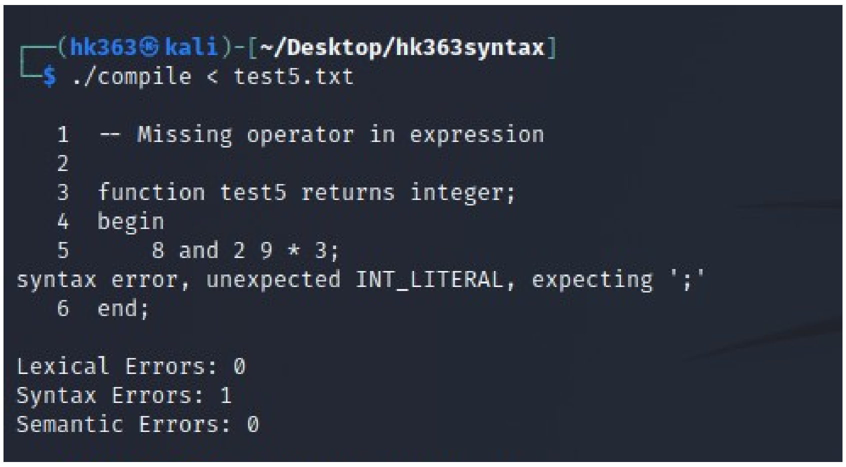
Test 4:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test4.txt

   1  -- Function with a Reduction
   2
   3  function test4 returns integer;
   4  begin
   5      reduce *
   6          2 + 8;
   7          6;
   8          3;
   9      endreduce;
  10  end;

Compiled Sucessfully
```

Test 4 compiled successfully, as it should, without any syntax or lexical errors → correct result

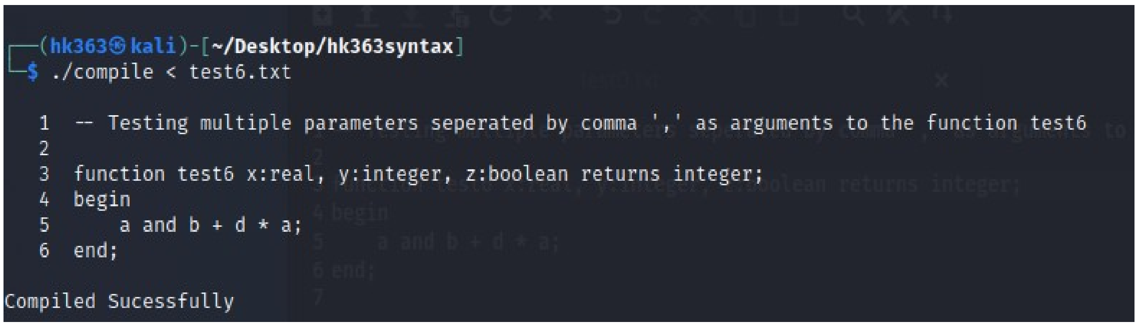
Test 5:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test5.txt

   1  -- Missing operator in expression
   2
   3  function test5 returns integer;
   4  begin
   5      8 and 2 9 * 3;
syntax error, unexpected INT_LITERAL, expecting ';'
   6  end;

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

Test 5 found the syntax error, as it should → correct result

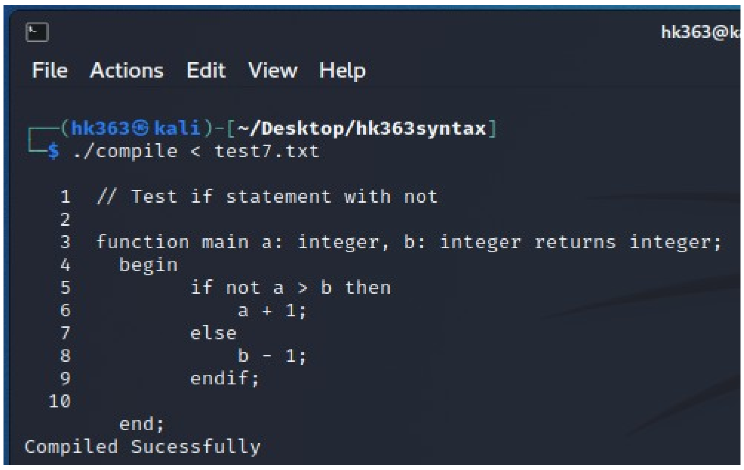
Test 6:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test6.txt

-- Testing multiple parameters seperated by comma ',' as arguments to the function test6

function test6 x:real, y:integer, z:boolean returns integer;
begin
    a and b + d * a;
end;

Compiled Sucessfully
```

Test 6 compiled successfully, as it should, without any syntax or lexical errors → correct result

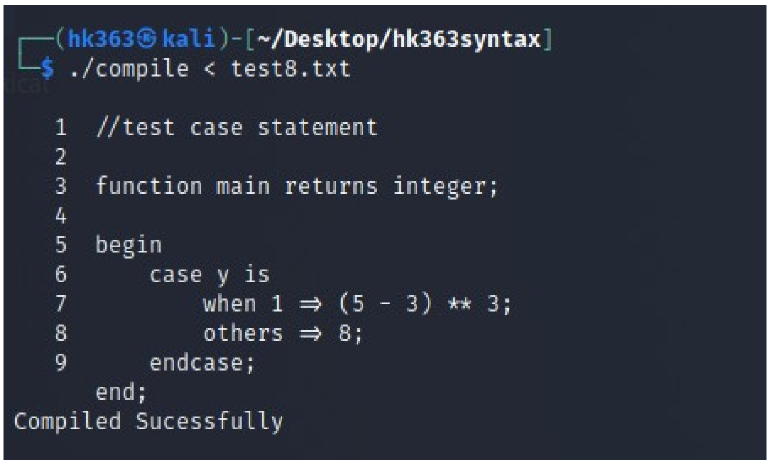
Test 7:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test7.txt

// Test if statement with not

function main a: integer, b: integer returns integer;
  begin
        if not a > b then
            a + 1;
        else
            b - 1;
        endif;

  end;
Compiled Sucessfully
```

Test 7 compiled successfully, as it should, without any syntax or lexical errors → correct result

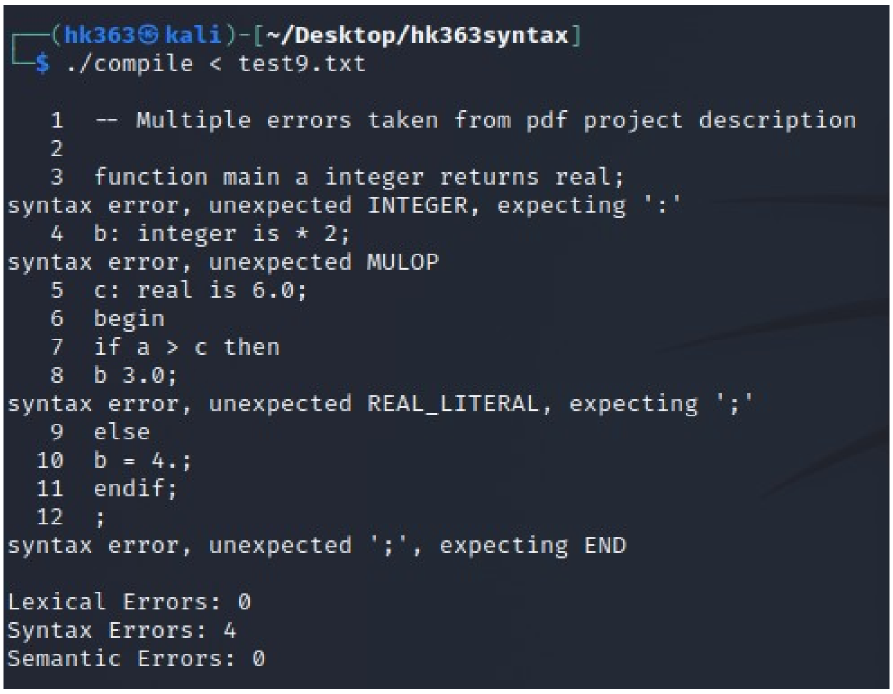
Test 8:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test8.txt

//test case statement

function main returns integer;

begin
    case y is
        when 1 => (5 - 3) ** 3;
        others => 8;
    endcase;
end;
Compiled Sucessfully
```

Test 8 compiled successfully, as it should, without any syntax or lexical errors → correct result

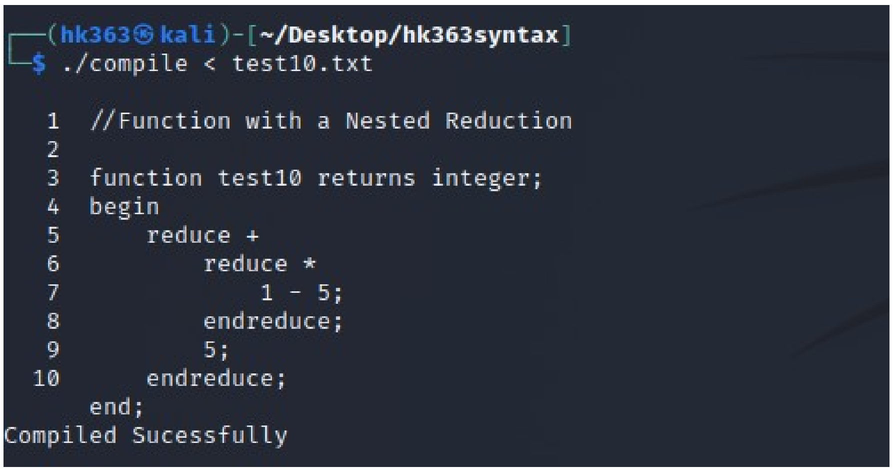
Test 9:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test9.txt

   1  -- Multiple errors taken from pdf project description
   2
   3  function main a integer returns real;
syntax error, unexpected INTEGER, expecting ':'
   4  b: integer is * 2;
syntax error, unexpected MULOP
   5  c: real is 6.0;
   6  begin
   7  if a > c then
   8  b 3.0;
syntax error, unexpected REAL_LITERAL, expecting ';'
   9  else
  10  b = 4.;
  11  endif;
  12  ;
syntax error, unexpected ';', expecting END

Lexical Errors: 0
Syntax Errors: 4
Semantic Errors: 0
```

Test 9 found the syntax errors, as it should, according to pdf description example→ correct result

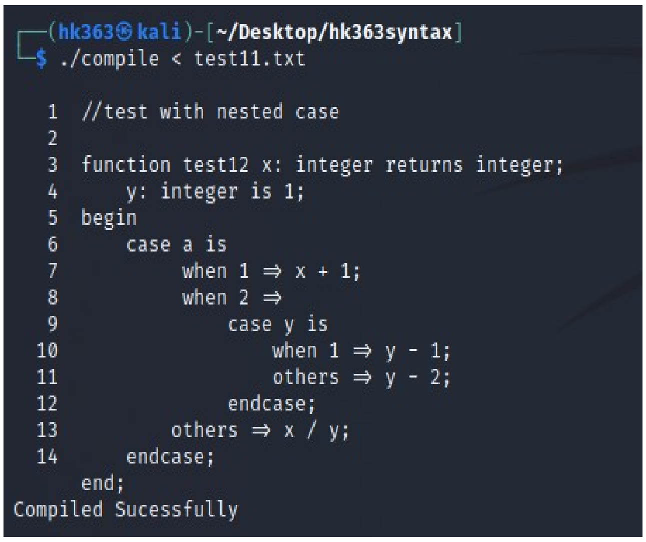
Test 10:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test10.txt

   1  //Function with a Nested Reduction
   2
   3  function test10 returns integer;
   4  begin
   5      reduce +
   6          reduce *
   7              1 - 5;
   8          endreduce;
   9          5;
  10      endreduce;
      end;
Compiled Sucessfully
```

Test 10 compiled successfully, as it should, without any syntax or lexical errors → correct result

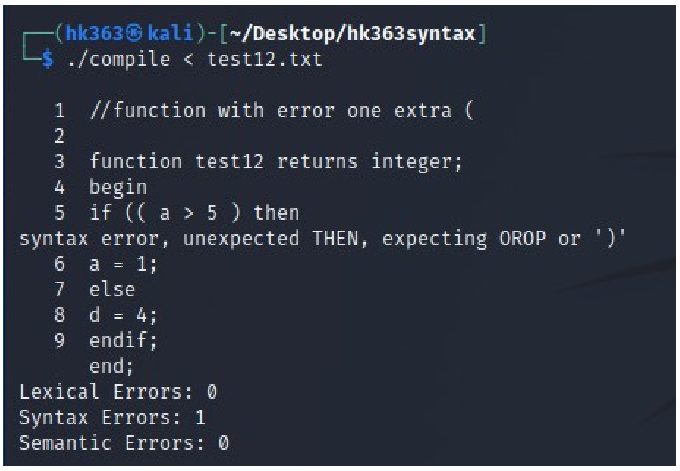
Test 11:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test11.txt

   1  //test with nested case
   2
   3  function test12 x: integer returns integer;
   4      y: integer is 1;
   5  begin
   6      case a is
   7           when 1 ⇒ x + 1;
   8           when 2 ⇒
   9               case y is
  10                   when 1 ⇒ y - 1;
  11                   others ⇒ y - 2;
  12               endcase;
  13          others ⇒ x / y;
  14      endcase;
      end;
Compiled Sucessfully
```

Test 11 compiled successfully, as it should, without any syntax or lexical errors → correct result

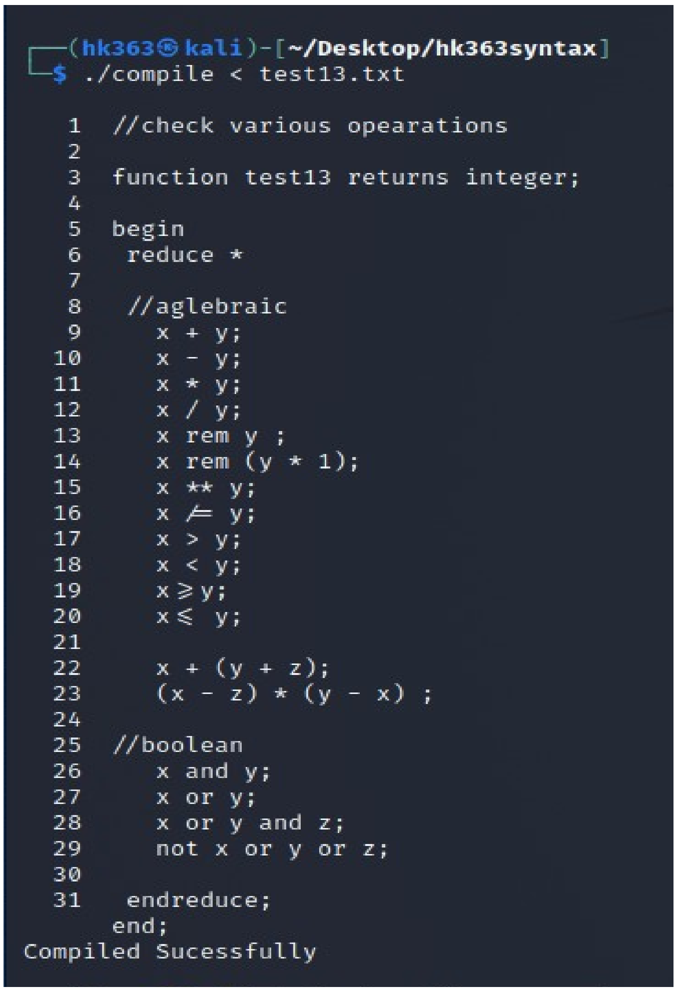
Test 12:



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test12.txt

   1  //function with error one extra (
   2
   3  function test12 returns integer;
   4  begin
   5  if (( a > 5 ) then
syntax error, unexpected THEN, expecting OROP or ')'
   6  a = 1;
   7  else
   8  d = 4;
   9  endif;
      end;
Lexical Errors: 0
Syntax Errors: 1
Semantic Errors: 0
```

Test 12 found the syntax error, as it should → correct result

Test 13



```
┌──(hk363㉿kali)-[~/Desktop/hk363syntax]
└─$ ./compile < test13.txt

   1  //check various opearations
   2
   3  function test13 returns integer;
   4
   5  begin
   6   reduce *
   7
   8   //aglebraic
   9     x + y;
  10     x - y;
  11     x * y;
  12     x / y;
  13     x rem y ;
  14     x rem (y * 1);
  15     x ** y;
  16     x /= y;
  17     x > y;
  18     x < y;
  19     x >=y;
  20     x <= y;
  21
  22     x + (y + z);
  23     (x - z) * (y - x) ;
  24
  25  //boolean
  26     x and y;
  27     x or y;
  28     x or y and z;
  29     not x or y or z;
  30
  31   endreduce;
      end;
Compiled Sucessfully
```

Test 13 compiled successfully, as it should, without any syntax or lexical errors → correct result

Conclusion

The main lessons I have acquired from this project are about grammars, bottom-up parsing, and the importance of recursion in Bison rules. Also, I got a more in depth understanding of how programming languages find syntax errors. Finally after studying the parser.output file I saw how the parser moves between states with shift and reduce from start to finish.