The testing for the recognizer consisted of looping through an array of filenames and creating a new recognizer for each one, testing the contents of the file to see if it is syntaxically correct. The testing of the scanner is embedded in to this testing, as it must properly recognize and the tokens and send them to the recognizer. The files used are as follows:

An empty Pascal file:

```
program testProg;
begin
end
.
```

Testing declarations:

```
program testDeclarations;

var foo, bar : integer;
var foobar : real;
begin
end
.
```

Testing array declarations:

```
program testArray;
var listOfInts : array [ 47 : 22 ] of integer
;   begin
end
.
```

Testing a function declaration with a compound declaration inside that has one ID in it, should be a procedure call.

```
program testArray;
var listOfInts : array [ 47 : 22 ] of integer;
function testFunc ( testParam : integer ) : integer ;
begin
random
end ;
begin
end
.
```

Testing the same function declaration with two IDs in it, which should not compile (and it does not)

```
program testArray;
var listOfInts : array [ 47 : 22 ] of integer;
function testFunc ( testParam : integer ) : integer ;
begin
isnt correct
end ;
begin
end
.
```

0.1 Second Phase, Parsing and Code Generation

To ensure the compiler worked properly, a happy path" Pascal file was created. This tested edge cases for all different kinds of conditions. In addition to this program that compiles properly, there are different programs to test that each of the errors will be handled properly.

0.1.1 Testing Good Code

This first first program is a follows:

```
  program happyPath ;
var foo, bar : integer ;
var foobar, pi : real ;
begin
    foo := 2;
    bar := 10;
    pi := 3.1415 ;
    foobar := pi / 2 ;
    if foobar < pi then
        write(foobar)
    else
        begin
        end
    if foobar > pi then
        begin
        end
    else
```

```
        begin
            write(pi)
        end
    if 2 <> 3 then
        write(5)
    else
        begin
        end
    if 2 + 2 = 4 then
        write(4)
    else
        begin
        end
    if not (2 + 2 = 4) then
        write(4)
    else
        begin
        write(5)
        end
    foobar := 2*pi + 2 / 17 * 2.7183;
    write(foobar);
    while bar > 0 do
    begin
        bar := bar - 1;
        write(bar)
    end


end .
```

This starts by testing the declarations of integers and reals. Next basic assignment is tested, followed by a series of if statements. Each of these if statements check whether the boolean evaluations of expressions work properly both for reals and integers, while simulaneously checking the write() method. A more complicated assignment is created next, to ensure the proper evaluation of a complex operation tree. Next a while loop is tested, to ensure it terminates at the proper step, and there are no off by one errors in the code generation.

0.1.2 Testing Bad Pascal Code

The next series of tests consist of different cases that will cause compilation

TOKEN_MISMATCH will occurr when a certain token is expected, but there was another token in its place.

```
program tmm ;
var foo : integer ;
```

```
begin
    foo := ¡ 23
end .
```

AFTER_PROGRAM handles the error in which the semicolon after program is not present.

```
  program tmm
var foo : integer ;
begin
    foo := ¡ 23
end .
```

PROGRAM_NOT_FOUND is the error thrown when the first line does not contain program <programname ;>.

```
  var foo : integer ;
begin
    foo := ¡ 23
end .
```

EXPECTED_EOF error occurrs when there is extra text after the end. for the main of the program.

```
  program tmm ;
var foo : integer ;
begin
    foo := 23
end .
var foobar : integer ;
```

COMPOUND_STMT_SEMICOLON is thrown when there is a semicolon after the last statement in a compound statement.

```
  program tmm ;
var foo : integer ;
begin
    foo := 23 ;
end .
```

VARIBLE_NOT_DEC occurrs when an attempt to use a variable that has not been declared.

```
  program tmm ;
var foo : integer ;
begin
    bar := 23
end .
```

ASSIGN_REAL_TO_INT error is thrown when a real number is attempted to be assigned into an integer variable.

```
program tmm ;
var foo : integer ;
begin
    foo := 23.22
end .
```

REAL_INT_COMPARISON is thrown when a real number and an integer are compared.

```
program tmm ;
var foo : integer ;
begin
if 1 > 55.5 then
    begin
    end
else
    begin
    end
end .
```