```
import Test.HUnit
import qualified Straight as S
```

Test basic assignments

Tests to make sure we can assign values to variables

```
a1 = S.Prog [('a', S.Integer)] [] [S.Asgn 'a' (S.Const (S.I 2))]
res1 = S.run a1

basicAssignmentTest1 = TestCase (assertEqual "Resulting store should be [[('a', 2)]]"
                                 res1 [[('a', (S.I 2))]])
```

Same as previous test but checking it for boolean values

```
a2 = S.Prog [('a', S.Boolean)] [] [S.Asgn 'a' (S.Const (S.B True))]
res8 = S.run a2

basicAssignmentTest2 = TestCase (assertEqual "Resulting store should be [[('a', 2)]]"
                                 res8 [[('a', (S.B True))]])
```

Test If Statmemnts

Tests if when the condition for an if statment is true then the true block is executed and the else block is skipped.

```
b4 = S.If (S.Bin S.Eq (S.Var 'a') (S.Var 'b')) [S.Asgn 'a' (S.Const (S.I 2))] [S.Asgn 'b' (S
b3 = S.Asgn 'b' (S.Const (S.I 1))
b2 = S.Asgn 'a' (S.Const (S.I 1))
b1 = S.Prog [('a', S.Integer), ('b', S.Integer)] [] [b2, b3, b4]
res2 = S.run b1

ifTest1 = TestCase (assertEqual "a and b are equal so a should be set to 2"
                    res2 [[('a', (S.I 2)), ('b', (S.I 1))]])
```

Tests if when the condition of an if statment is false then the true branch is skipped and the else block is excuted.

```
c4 = S.If (S.Bin S.Eq (S.Var 'a') (S.Var 'b')) [S.Asgn 'a' (S.Const (S.I 2))] [S.Asgn 'b' (S
c3 = S.Asgn 'b' (S.Const (S.I 1))
c2 = S.Asgn 'a' (S.Const (S.I 0))
c1 = S.Prog [('a', S.Integer), ('b', S.Integer)] [] [c2, c3, c4]
res3 = S.run c1
```

```
ifTest2 = TestCase (assertEqual "a and b are not equal so b should be set to 2"
                    res3 [[('a', (S.I 0)), ('b', (S.I 2))]])
```

While loop tests

Tests to make sure that while the condition is true the statments are executed and once its false then it stops.

```
d4 = S.Bin S.Plus (S.Var 'a') (S.Const (S.I 1))
d3 = S.Asgn 'a' d4
d2 = S.While (S.Bin S.Ne (S.Var 'a') (S.Const (S.I 9))) [d3]
d1 = S.Prog [('a', S.Integer)] [] [d2]
res4 = S.run d1

whileTest1 = TestCase (assertEqual "Loop should keep adding to a untill a is 9"
                       res4 [[('a', (S.I 9))]])
```

Procedure tests

Tests a simple procedure, no arguments or local variables just modifys a global variable.

```
e3 = S.Bin S.Plus (S.Var 'a') (S.Const (S.I 1))
e2 = S.Procedure "increment" [] [] [S.Asgn 'a' e3]
e1 = S.Prog [('a', S.Integer)] [e2] [S.Call "increment" []]
res5 = S.run e1

procedureTest1 = TestCase (assertEqual "Procedure should be called and increment a by 1"
                           res5 [[('a', (S.I 1))]])
```

Demonstrates procedures with paramaters, tests whether the values are passed correctly and wether they stay in the store after execution (which they shouldent).

```
f3 = S.Bin S.Plus (S.Var 'h') (S.Const (S.I 1))
f2 = S.Procedure "increment" [('h', S.Integer)] [] [S.Asgn 'a' f3]
f1 = S.Prog [('a', S.Integer), ('b', S.Integer)] [f2] [S.Asgn 'b' (S.Const (S.I 3)),
                                                       S.Call "increment" ['b']]
res6 = S.run f1

procedureTest2 = TestCase (assertEqual "Procedure should be called and assign a to b + 1"
                           res6 [[('a', (S.I 4)), ('b', (S.I 3))]])
```

Last test test procedures with paramaters and local variables, tests whether look up works correctly and wehter they are on the stack after procedure execution.

```
g4 = S.If (S.Bin S.Lt (S.Var 'h') (S.Var 'l')) [S.Asgn 'j' g3, S.Call "increment" ['j', 'l']
g3 = S.Bin S.Plus (S.Var 'h') (S.Const (S.I 1))
g2 = S.Procedure "increment" [('h', S.Integer), ('l', S.Integer)] [('j', S.Integer)] [g4]
g1 = S.Prog [('a', S.Integer), ('b', S.Integer)] [g2] [S.Asgn 'b' (S.Const (S.I 3)),
                                                      S.Call "increment" ['a', 'b']]
res7 = S.run g1

procedureTest3 = TestCase (assertEqual "Procedure should be called and increment a by 1"
                           res7 [[('a', (S.I 3)), ('b', (S.I 3))]])

tests = TestList [basicAssignmentTest1,
                  basicAssignmentTest2,
                  ifTest1,
                  ifTest2,
                  whileTest1,
                  procedureTest1,
                  procedureTest2,
                  procedureTest3]

run = runTestTT tests
main = run
```

I also did tests to check whether errors where thrown when there was a type error, multiple variable declrations with same name or multiple procedure declrations with the same name. But wasnt sure how to use HUnit to test for an exception so dident include them here.