```
import random
import unittest
import Racecar.Compiler as Compiler
import Racecar.SymbolTable as SymbolTable
import Racecar.SemanticAnalyzer as SemanticAnalyzer
import Racecar.Parser as Parser


class TranslatorTests(unittest.TestCase):
    def test_empty_statement(self):
        test_string = \
            """
"""
        correct_translation = \
            """"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_drive_forwards(self):
        test_string1 = \
            """drive forwards 10 steps
"""
        test_string2 = \
            """drive forward 10 steps
"""
        test_string3 = \
            """drive forwards 10 step
"""
        test_string4 = \
            """drive forward 10 step
"""
        correct_translation = \
            """translate_car(10, CarDirection.FORWARDS)
"""

        ast1 = Parser.parseString(test_string1)
        ast2 = Parser.parseString(test_string1)
        ast3 = Parser.parseString(test_string1)
        ast4 = Parser.parseString(test_string1)

        self.assertEqual(len(ast1.errors), 0)
        self.assertEqual(len(ast2.errors), 0)
        self.assertEqual(len(ast3.errors), 0)
        self.assertEqual(len(ast4.errors), 0)

        result1 = Compiler.getPythonCode(test_string1)
        result2 = Compiler.getPythonCode(test_string2)
        result3 = Compiler.getPythonCode(test_string3)
        result4 = Compiler.getPythonCode(test_string4)

        saErrors1 = SemanticAnalyzer.analyzeStart(ast1)
        saErrors2 = SemanticAnalyzer.analyzeStart(ast2)
        saErrors3 = SemanticAnalyzer.analyzeStart(ast3)
        saErrors4 = SemanticAnalyzer.analyzeStart(ast4)

        self.assertEqual(len(saErrors1), 0)
        self.assertEqual(len(saErrors2), 0)
        self.assertEqual(len(saErrors3), 0)
```

```
        self.assertEqual(len(saErrors4), 0)

        self.assertEqual(result1[0], correct_translation)
        self.assertEqual(result2[0], correct_translation)
        self.assertEqual(result3[0], correct_translation)
        self.assertEqual(result4[0], correct_translation)

    def test_drive_backwards(self):
        test_string1 = \
            """drive backwards 10 steps
"""
        test_string2 = \
            """drive backward 10 steps
"""
        test_string3 = \
            """drive backwards 10 step
"""
        test_string4 = \
            """drive backward 10 step
"""
        correct_translation = \
            """translate_car(10, CarDirection.BACKWARDS)
"""

        ast1 = Parser.parseString(test_string1)
        ast2 = Parser.parseString(test_string1)
        ast3 = Parser.parseString(test_string1)
        ast4 = Parser.parseString(test_string1)

        self.assertEqual(len(ast1.errors), 0)
        self.assertEqual(len(ast2.errors), 0)
        self.assertEqual(len(ast3.errors), 0)
        self.assertEqual(len(ast4.errors), 0)

        result1 = Compiler.getPythonCode(test_string1)
        result2 = Compiler.getPythonCode(test_string2)
        result3 = Compiler.getPythonCode(test_string3)
        result4 = Compiler.getPythonCode(test_string4)

        saErrors1 = SemanticAnalyzer.analyzeStart(ast1)
        saErrors2 = SemanticAnalyzer.analyzeStart(ast2)
        saErrors3 = SemanticAnalyzer.analyzeStart(ast3)
        saErrors4 = SemanticAnalyzer.analyzeStart(ast4)

        self.assertEqual(len(saErrors1), 0)
        self.assertEqual(len(saErrors2), 0)
        self.assertEqual(len(saErrors3), 0)
        self.assertEqual(len(saErrors4), 0)

        self.assertEqual(result1[0], correct_translation)
        self.assertEqual(result2[0], correct_translation)
        self.assertEqual(result3[0], correct_translation)
        self.assertEqual(result4[0], correct_translation)

    def test_turn_left(self):
        test_string = \
            """turn left
"""
        correct_translation = \
            """rotate_car(WheelDirection.LEFT)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_turn_right(self):
        test_string = \
            """turn right
"""
        correct_translation = \
            """rotate_car(WheelDirection.RIGHT)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_print(self):
        test_string = \
            """print "hello world"
"""
        correct_translation = \
            """print_to_console("hello world")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_declare(self):
        test_string = \
            """myNum is a number
"""
        correct_translation = \
            """myNum = None
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_assign(self):
        test_string = \
            """set myVar to otherThing
"""
        correct_translation = \
            """myVar = otherThing
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 2)
```

```
195.
196.         self.assertEqual(len(ast.errors), 0)
197.         self.assertEqual(result[0], correct_translation)
198.
199.     def test_define(self):
200.         test_string = \
201.             """define moveForwardFive
202. {
203.    drive forward 5 steps
204. }
205. """
206.         correct_translation = \
207.             """def moveForwardFive():
208.    translate_car(5, CarDirection.FORWARDS)
209. """
210.
211.         result = Compiler.getPythonCode(test_string)
212.         ast = Parser.parseString(test_string)
213.
214.         saErrors = SemanticAnalyzer.analyzeStart(ast)
215.         self.assertEqual(len(saErrors), 0)
216.
217.         self.assertEqual(len(ast.errors), 0)
218.         self.assertEqual(result[0], correct_translation)
219.
220.     def test_function_invocation_no_params(self):
221.         test_string = \
222.             """define moveBackwardFive
223. {
224.    drive backward 5
225. }
226. define moveForwardThenBackward
227. {
228.    drive forward 5
229.    moveBackwardFive
230. }
231. moveForwardThenBackward
232. """
233.         correct_translation = \
234.             """def moveBackwardFive():
235.    translate_car(5, CarDirection.BACKWARDS)
236. def moveForwardThenBackward():
237.    translate_car(5, CarDirection.FORWARDS)
238.    moveBackwardFive()
239. moveForwardThenBackward()
240. """
241.         result = Compiler.getPythonCode(test_string)
242.         ast = Parser.parseString(test_string)
243.
244.         saErrors = SemanticAnalyzer.analyzeStart(ast)
245.         self.assertEqual(len(saErrors), 0)
246.
247.         self.assertEqual(len(ast.errors), 0)
248.         self.assertEqual(result[0], correct_translation)
249.
250.     def test_function_invocation_with_one_parameter(self):
251.         test_string = \
252.             """move5Steps "forwards"
253. """
254.         correct_translation = \
255.             """move5Steps("forwards")
256. """
257.         result = Compiler.getPythonCode(test_string)
258.         ast = Parser.parseString(test_string)
259.
```

```
        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertNotEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_function_invocation_with_two_parameters(self):
        test_string = \
            """define turnLeftThenDriveStraight using numStepsTurn \
           (number) and numStepsDrive (number)
{
turn left
drive forward numStepsTurn steps
turn right
drive forward numStepsDrive steps
}
turnLeftThenDriveStraight 5 10
"""
        correct_translation = \
            """def turnLeftThenDriveStraight(numStepsTurn, numStepsDrive):
   rotate_car(WheelDirection.LEFT)
   translate_car(numStepsTurn, CarDirection.FORWARDS)
   rotate_car(WheelDirection.RIGHT)
   translate_car(numStepsDrive, CarDirection.FORWARDS)
turnLeftThenDriveStraight(5, 10)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_plus_expression(self):
        test_string = \
            """print (2 + 3)
"""
        correct_translation = \
            """print_to_console(((2) + (3)))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_times_expression(self):
        test_string = \
            """print (2 * 3)
"""
        correct_translation = \
            """print_to_console(((2) * (3)))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
```

```
        self.assertEqual(result[0], correct_translation)

    def test_minus_expression(self):
        test_string = \
            """print (2 - 3)
"""
        correct_translation = \
            """print_to_console(((2) - (3)))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_divide_expression(self):
        test_string = \
            """print (2 / 3)
"""
        correct_translation = \
            """print_to_console(((2) / (3)))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_all_expression(self):
        test_string = \
            """print (1 + 2 * (3 + 4))
"""
        correct_translation = \
            """print_to_console(((1) + (((2) * (((3) + (4)))))))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_assign_num_change(self):
        test_string = \
            """num is a number
set num to 10
set num to num*2
"""
        correct_translation = \
            """num = None
num = 10
num = ((num) * (2))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_assign_easy_num_change(self):
        test_string = \
            """num is a number
set num to 10
num2 is a number
set num2 to 20
set num to num2
"""
        correct_translation = \
            """num = None
num = 10
num2 = None
num2 = 20
num = num2
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_assign_word_print(self):
        test_string = \
            """color is a word
set color to "blue"
print color
"""
        correct_translation = \
            """color = None
color = "blue"
print_to_console(color)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_assign_word_print_complicated(self):
        test_string = \
            """color is a word
set color to "blue"
print color
c2 is a word
set c2 to "green"
set color to c2
"""
        correct_translation = \
            """color = None
color = "blue"
print_to_console(color)
c2 = None
c2 = "green"
color = c2
```

```
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_statement(self):
        test_string = \
            """if 1
{
   print "yay"
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_else_statement(self):
        test_string = \
            """if 1
{
   print "yay"
}
else
{
   print "no"
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
else:
   print_to_console("no")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_statement_nested(self):
        test_string = \
            """if 1
{
   print "yay"
   if 1
   {
       print "yahoo"
   }
```

```
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
   if 1:
       print_to_console("yahoo")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_else_statement_nested(self):
        test_string = \
            """if 1
{
   print "yay"
   if 1
   {
       print "yahoo"
   }
   else
   {
       print "oh no"
   }
}
else
{
   print "no"
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
   if 1:
       print_to_console("yahoo")
   else:
       print_to_console("oh no")
else:
   print_to_console("no")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_elseif_else_statement(self):
        test_string = \
            """if 1
{
   print "yay"
}
elseIf 2
{
   print "no"
}
```

```
else
{
   print "done"
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
elif 2:
   print_to_console("no")
else:
   print_to_console("done")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_elseif_else_statement_nested(self):
        test_string = \
            """if 1
{
   print "yay"
   if 1
   {
       print "yahoo"
   }
   elseIf 2
   {
       print "oh no"
   }
   else
   {
       print "here"
   }
}
elseIf 2
{
   print "no"
}
else
{
   print "done"
}
"""
        correct_translation = \
            """if 1:
   print_to_console("yay")
   if 1:
       print_to_console("yahoo")
   elif 2:
       print_to_console("oh no")
   else:
       print_to_console("here")
elif 2:
   print_to_console("no")
else:
   print_to_console("done")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)
```

```
650.
651.         saErrors = SemanticAnalyzer.analyzeStart(ast)
652.         self.assertEqual(len(saErrors), 0)
653.
654.         self.assertEqual(len(ast.errors), 0)
655.         self.assertEqual(result[0], correct_translation)
656.
657.     def test_if_if_else_complicated(self):
658.         test_string = \
659.             """if 1
660. {
661.    print "yay"
662.    if 1
663.    {
664.        print "yahoo"
665.    }
666.    else
667.    {
668.        print "oh no"
669.        if 1
670.        {
671.            print "good"
672.        }
673.        if 1
674.        {
675.            print "yay"
676.        }
677.        elseIf 2
678.        {
679.            print "no"
680.            if 1
681.            {
682.                print "hi"
683.            }
684.        }
685.        elseIf 3
686.        {
687.            print "yes"
688.        }
689.        else
690.        {
691.            if 5
692.            {
693.                print "works"
694.            }
695.            print "end"
696.        }
697.    }
698. }
699. else
700. {
701.    print "no"
702. }
703. """
704.         correct_translation = \
705.             """if 1:
706.    print_to_console("yay")
707.    if 1:
708.        print_to_console("yahoo")
709.    else:
710.        print_to_console("oh no")
711.        if 1:
712.            print_to_console("good")
713.        if 1:
714.            print_to_console("yay")
```

```
       elif 2:
           print_to_console("no")
           if 1:
               print_to_console("hi")
       elif 3:
           print_to_console("yes")
       else:
           if 5:
               print_to_console("works")
           print_to_console("end")
else:
   print_to_console("no")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_comment_singleline(self):
        test_string = \
            """:) this is a single line comment
"""
        correct_translation = \
            """"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_comment_multiline(self):
        test_string = \
            """:-( this is
a multiline
comment
:-)
"""
        correct_translation = \
            """"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_loop_for(self):
        test_string = \
            """myCounter is a number
set myCounter to 10
repeat myCounter times
{
   drive forward 1 step
   print myCounter
}
"""
```

```
        correct_translation = \
            """myCounter = None
myCounter = 10
for x in range(myCounter):
   translate_car(1, CarDirection.FORWARDS)
   print_to_console(myCounter)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_loop_for_nested(self):
        test_string = \
            """myCounter is a number
set myCounter to 10
myCounter2 is a number
set myCounter2 to 10
repeat myCounter times
{
   drive forward 1 step
   repeat myCounter2 times
   {
       drive forward 1 step
   }
}
"""
        correct_translation = \
            """myCounter = None
myCounter = 10
myCounter2 = None
myCounter2 = 10
for x in range(myCounter):
   translate_car(1, CarDirection.FORWARDS)
   for x in range(myCounter2):
       translate_car(1, CarDirection.FORWARDS)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_loop_while(self):
        test_string = \
            """myCounter is a number
set myCounter to 1
repeat if myCounter is not 5
{
   drive forward 1 step
   set myCounter to myCounter + 1
}
"""
        correct_translation = \
            """myCounter = None
myCounter = 1
while myCounter != 5:
   translate_car(1, CarDirection.FORWARDS)
```

```
   myCounter = ((myCounter) + (1))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

#**********************************stopped SA tests here

    def test_loop_while_nested(self):
        test_string = \
            """myCounter is a number
set myCounter to 1
myCounter2 is a number
repeat if myCounter is not 5
{
   drive forward 1 step
   set myCounter2 to 0
   repeat if myCounter2 is not 5
   {
       drive forward 1 step
       set myCounter2 to myCounter2 + 1
   }
   set myCounter to myCounter + 1
}
"""
        correct_translation = \
            """myCounter = None
myCounter = 1
myCounter2 = None
while myCounter != 5:
   translate_car(1, CarDirection.FORWARDS)
   myCounter2 = 0
   while myCounter2 != 5:
       translate_car(1, CarDirection.FORWARDS)
       myCounter2 = ((myCounter2) + (1))
   myCounter = ((myCounter) + (1))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_boolean_opeartors(self):
        test_string = \
            """if 1 < 2
{
   print "yes"
}
elseIf 1 is 2
{
   print "yes"
}
if 1 >= 2
{
   print "no"
}
```

```
elseIf 1 is not 2
{
   print "yes"
}
elseIf 1 > 2
{
   print "yes"
}
elseIf 1 < 2
{
   print "yes"
}
"""
        correct_translation = \
            """if 1 < 2:
   print_to_console("yes")
elif 1 == 2:
   print_to_console("yes")
if 1 >= 2:
   print_to_console("no")
elif 1 != 2:
   print_to_console("yes")
elif 1 > 2:
   print_to_console("yes")
elif 1 < 2:
   print_to_console("yes")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_string_concatenation(self):
        test_string = \
            """print "hey" ++ myWord
"""
        correct_translation = \
            """print_to_console((str("hey") + str(myWord)))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_string_concatenation_complicated(self):
        test_string = \
            """print "hey" ++ myWord ++ "now"
"""
        correct_translation = \
            """print_to_console((str((str("hey") + str(myWord))) + str("now")))
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_get_car_position(self):
        test_string = \
            """print getCarPosition
"""
        correct_translation = \
            """print_to_console(getCurrentPosition())
```

```
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_can_move(self):
        test_string = \
            """if canDrive forward 5 steps
{
   drive forward 5 steps
}
"""
        correct_translation = \
            """if can_move(5, CarDirection.FORWARDS):
   translate_car(5, CarDirection.FORWARDS)
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_if_plus(self):
        test_string = \
            """if 2 + 5
{
   print "yay"
}
"""
        correct_translation = \
            """if ((2) + (5)):
   print_to_console("yay")
"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)

    def test_template(self):
        test_string = \
            """
"""
        correct_translation = \
            """"""
        result = Compiler.getPythonCode(test_string)
        ast = Parser.parseString(test_string)

        self.assertEqual(len(ast.errors), 0)
        self.assertEqual(result[0], correct_translation)


class SymbolTableTests(unittest.TestCase):

    def test_symbol_table_add_entry(self):
        '''Tests the SymbolTableEntry.addEntry) function.'''

        table = SymbolTable.SymbolLookupTable()

        entry1 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
        entry2 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])

        table.addEntry(entry1)
```

```
1040.
1041.         self.assertFalse(table.addEntry(entry2))
1042.
1043.         entry3 = SymbolTable.SymbolTableEntry("name2", "word", [0], None, [])
1044.
1045.         self.assertTrue(entry3.validateWithTableEntry(entry3))
1046.
1047.     def test_symbol_table_verify(self):
1048.         '''Tests the SymbolLookupTable.verifyEntry() function.'''
1049.         table = SymbolTable.SymbolLookupTable()
1050.
1051.         entry1 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
1052.         entry2 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
1053.         entry3 = SymbolTable.SymbolTableEntry("name2", "word", [0], None, [])
1054.
1055.         table.addEntry(entry1)
1056.         table.addEntry(entry3)
1057.
1058.         self.assertTrue(table.verifyEntry(entry2))
1059.
1060.     def test_symbol_table_get_entry(self):
1061.         '''Tests the SymbolLookupTable.getEntry() function.'''
1062.         table = SymbolTable.SymbolLookupTable()
1063.
1064.         entry1 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
1065.
1066.         table.addEntry(entry1)
1067.
1068.         self.assertEqual(table.getEntry(SymbolTable.SymbolTableEntry("name1", None, [0],
    None, [])), entry1)
1069.
1070.
1071. class SemanticAnalyzerTests(unittest.TestCase):
1072.     def test_basic(self):
1073.         test_string = \
1074.             """drive forward 5 steps
1075. """
1076.
1077.         ast = Parser.parseString(test_string)
1078.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1079.
1080.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1081.         self.assertEqual(len(saErrors), 0)
1082.
1083.     def test_print_var(self):
1084.         test_string = \
1085.             """myNum is a number
1086. set myNum to 10
1087. print myNum
1088. """
1089.
1090.         ast = Parser.parseString(test_string)
1091.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1092.
1093.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1094.         self.assertEqual(len(saErrors), 0)
1095.
1096.     def test_print_undeclared_var(self):
1097.         test_string = \
1098.             """print myNum
1099. """
1100.
1101.         ast = Parser.parseString(test_string)
1102.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1103.
```

```
        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_set_undeclared_var(self):
        test_string = \
            """set myNum to 10
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        #should have an error saying mySecondNum doesnt exist
        self.assertEqual(len(saErrors), 1)

    def test_general_access_undeclared_var(self):
        test_string = \
            """myNum is a number
set myNum to mySecondNum
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        #should have an error saying mySecondNum doesnt exist
        self.assertEqual(len(saErrors), 1)

    def test_var_declared_in_if(self):
        test_string = \
            """if 1
{
   myNum is a number
   set myNum to 10
   print myNum
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_var_declared_in_if_accessed_outside(self):
        test_string = \
            """if 1
{
   myNum is a number
   set myNum to 10
   print myNum
}
print myNum
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
        saErrors = SemanticAnalyzer.analyzeStart(ast)
        #should have an error about printing myNum after the if
        self.assertEqual(len(saErrors), 1)

    def test_var_declared_in_func(self):
        test_string = \
            """define moveForwardFive
{
```

```
   myNum is a number
   set myNum to 5
   drive forward myNum steps
}
"""

        ast = Parser.parseString(test_string)

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_var_declared_in_func_accessed_outside(self):
        test_string = \
            """define moveForwardFive
{
   myNum is a number
   set myNum to 5
   drive forward myNum steps
}
moveForwardFive
drive forward myNum steps
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_access_passed_in_var_in_func(self):
        test_string = \
            """define moveForward using numSteps (number)
{
   drive forward numSteps steps
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_access_passed_in_var_in_if_in_func(self):
        test_string = \
            """define moveForwardFive using numSteps (number)
{
   drive forward numSteps steps
   if 1
   {
       myCounter is a number
       set myCounter to 10
       drive forward 10 steps
   }
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_call_func_anywhere(self):
        test_string = \
```

```
            """moveForwardFive
define moveForwardFive
{
   drive forward 5 steps
}
moveForwardFive
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_declare_func_in_func(self):
        test_string = \
            """define moveForwardFive
{
   drive forward 5 steps
   define moveForwardTen
   {
       drive forward 10 steps
   }
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_call_funciton_with_param(self):
        test_string = \
            """define moveForwardFive using numSteps (number)
{
   drive forward numSteps steps
}
myNum is a number
set myNum to 10
moveForwardFive myNum
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_access_func_param_outside(self):
        test_string = \
            """define moveForwardFive using numSteps (number)
{
   drive forward numSteps steps
}
myNum is a number
set myNum to 10
moveForwardFive myNum
print numSteps
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
        #should fail since numSteps is only inside the function
        self.assertEqual(len(saErrors), 1)

    def test_assignments_var_to_var(self):
        test_string = \
            """myNum is a number
set myNum to 10
myWord is a word
set myWord to "hello"
set myNum to myWord
set myWord to myNum
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 2)


    def test_assignments_var_number(self):
        test_string = \
            """myWord is a word
set myWord to 10
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_assignments_var_str_literal(self):
        test_string = \
            """myNum is a number
set myNum to "hello"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_compare_vars_num_to_string(self):
        test_string = \
            """myNum is a number
set myNum to 10
myWord is a word
set myWord to "hello"
if myWord > myNum
{
   print "bad"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_compare_num_to_stringliteral(self):
        test_string = \
            """if 10 > "hello"
```

```
{
   print "bad"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_compare_num_to_num(self):
        test_string = \
            """if 10 > 5
{
   print "good"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_compare_strL_to_strL(self):
        test_string = \
            """if "hello" is "hi"
{
   print "good"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_compare_strL_to_strL_not(self):
        test_string = \
            """if "hello" is not "hi"
{
   print "good"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_compare_num_str_is(self):
        test_string = \
            """if 10 is "hello"
{
   print "bad"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
        self.assertEqual(len(saErrors), 1)

    def test_num_params_passing(self):
        test_string = \
            """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{

}
moveForwardFiveAndTurn 10 "left"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_num_params_passing_2(self):
        test_string = \
            """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{

}
moveForwardFiveAndTurn 10 "hi" "extra"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_num_params_passing_3(self):
        test_string = \
            """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{

}
moveForwardFiveAndTurn 10 15
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_num_params_passing_4(self):
        test_string = \
            """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{
   print numSteps
   print direction
}
moveForwardFiveAndTurn "hello" "hi"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_num_params_passing_5(self):
        test_string = \
```

```
            """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{
   print numSteps
   print direction
}
moveForwardFiveAndTurn "left" 10
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_built_in_functions_params(self):
        test_string = \
            """drive forwards five steps
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_while_loop(self):
        test_string = \
            """repeat 5 times
{
   print "hi"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_while_loop_bad(self):
        test_string = \
            """repeat five times
{
   print "hi"
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_calling_nonexistant_function(self):
        test_string = \
            """fullTurn "left"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_course2(self):
```

```
        test_string = \
            """define fullTurn using direction (word)
{
   if direction is "left"
   {
       turn left
       turn left
   }
   else
   {
       turn right
       turn right
   }
}

define driveThenFullTurn using numSteps (number) and direction (word)
{
   drive forward numSteps steps
   fullTurn direction
}

driveThenFullTurn 0 "right"
driveThenFullTurn 25 "left"
driveThenFullTurn 18 "left"

repeat 2 times
{

   driveThenFullTurn 50 "right"
   driveThenFullTurn 15 "right"
   driveThenFullTurn 50 "left"
   driveThenFullTurn 16 "left"
}

driveThenFullTurn 50 "right"
driveThenFullTurn 15 "right"
driveThenFullTurn 50 "left"
driveThenFullTurn 10 "left"
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_var_uninitialized(self):
        test_string = \
            """myNum is a number
drive forward myNum steps
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 1)

    def test_call_function_in_if_and_while(self):
        test_string = \
            """moveForwardFiveAndTurn 10 "left"
if 1
{
   moveForwardFiveAndTurn 10 "left"
}
```

```
repeat 2 times
{
   moveForwardFiveAndTurn 10 "left"
}
define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{
   print numSteps
   print direction
   fullTurn direction
}
define fullTurn using direction (word)
{
   if direction is "left"
   {
       turn left
       turn left
   }
   else
   {
       turn right
       turn right
   }
   moveForwardFiveAndTurn 5 direction
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_call_function_before_declared(self):
        test_string = \
            """moveForwardFiveAndTurn 10 "left"
if 1
{
   moveForwardFiveAndTurn 10 "left"
}
repeat 2 times
{
   moveForwardFiveAndTurn 10 "left"
}
drive forward 5 steps
define moveForwardFiveAndTurn using numSteps (number) and direction (word)
{
   print numSteps
   print direction
}
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

    def test_template(self):
        test_string = \
            """
"""

        ast = Parser.parseString(test_string)
        self.assertEqual(len(ast.errors), 0, "Test failed at parser.")

```

```
        saErrors = SemanticAnalyzer.analyzeStart(ast)
        self.assertEqual(len(saErrors), 0)

if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(TranslatorTests)
    unittest.TextTestRunner(verbosity=2).run(suite)
```