

```
1. import random
2. import unittest
3. import Racecar.Compiler as Compiler
4. import Racecar.SymbolTable as SymbolTable
5. import Racecar.SemanticAnalyzer as SemanticAnalyzer
6. import Racecar.Parser as Parser
7.
8.
9. class TranslatorTests(unittest.TestCase):
10.     def test_empty_statement(self):
11.         test_string = \
12.             """
13.
14.             correct_translation = \
15.                 """
16.             result = Compiler.getPythonCode(test_string)
17.             ast = Parser.parseString(test_string)
18.
19.             saErrors = SemanticAnalyzer.analyzeStart(ast)
20.             self.assertEqual(len(saErrors), 0)
21.
22.             self.assertEqual(len(ast.errors), 0)
23.             self.assertEqual(result[0], correct_translation)
24.
25.     def test_drive_forwards(self):
26.         test_string1 = \
27.             """drive forwards 10 steps
28.
29.             test_string2 = \
30.                 """drive forward 10 steps
31.
32.             test_string3 = \
33.                 """drive forwards 10 step
34.
35.             test_string4 = \
36.                 """drive forward 10 step
37.
38.             correct_translation = \
39.                 """translate_car(10, CarDirection.FORWARDS)
40.
41.
42.             ast1 = Parser.parseString(test_string1)
43.             ast2 = Parser.parseString(test_string1)
44.             ast3 = Parser.parseString(test_string1)
45.             ast4 = Parser.parseString(test_string1)
46.
47.             self.assertEqual(len(ast1.errors), 0)
48.             self.assertEqual(len(ast2.errors), 0)
49.             self.assertEqual(len(ast3.errors), 0)
50.             self.assertEqual(len(ast4.errors), 0)
51.
52.             result1 = Compiler.getPythonCode(test_string1)
53.             result2 = Compiler.getPythonCode(test_string2)
54.             result3 = Compiler.getPythonCode(test_string3)
55.             result4 = Compiler.getPythonCode(test_string4)
56.
57.             saErrors1 = SemanticAnalyzer.analyzeStart(ast1)
58.             saErrors2 = SemanticAnalyzer.analyzeStart(ast2)
59.             saErrors3 = SemanticAnalyzer.analyzeStart(ast3)
60.             saErrors4 = SemanticAnalyzer.analyzeStart(ast4)
61.
62.             self.assertEqual(len(saErrors1), 0)
63.             self.assertEqual(len(saErrors2), 0)
64.             self.assertEqual(len(saErrors3), 0)
```

```

65.         self.assertEqual(len(saErrors4), 0)
66.
67.         self.assertEqual(result1[0], correct_translation)
68.         self.assertEqual(result2[0], correct_translation)
69.         self.assertEqual(result3[0], correct_translation)
70.         self.assertEqual(result4[0], correct_translation)
71.
72.     def test_drive_backwards(self):
73.         test_string1 = \
74.             """drive backwards 10 steps
75.         """
76.         test_string2 = \
77.             """drive backward 10 steps
78.         """
79.         test_string3 = \
80.             """drive backwards 10 step
81.         """
82.         test_string4 = \
83.             """drive backward 10 step
84.         """
85.         correct_translation = \
86.             """translate_car(10, CarDirection.BACKWARDS)
87.         """
88.
89.         ast1 = Parser.parseString(test_string1)
90.         ast2 = Parser.parseString(test_string1)
91.         ast3 = Parser.parseString(test_string1)
92.         ast4 = Parser.parseString(test_string1)
93.
94.         self.assertEqual(len(ast1.errors), 0)
95.         self.assertEqual(len(ast2.errors), 0)
96.         self.assertEqual(len(ast3.errors), 0)
97.         self.assertEqual(len(ast4.errors), 0)
98.
99.         result1 = Compiler.getPythonCode(test_string1)
100.        result2 = Compiler.getPythonCode(test_string2)
101.        result3 = Compiler.getPythonCode(test_string3)
102.        result4 = Compiler.getPythonCode(test_string4)
103.
104.        saErrors1 = SemanticAnalyzer.analyzeStart(ast1)
105.        saErrors2 = SemanticAnalyzer.analyzeStart(ast2)
106.        saErrors3 = SemanticAnalyzer.analyzeStart(ast3)
107.        saErrors4 = SemanticAnalyzer.analyzeStart(ast4)
108.
109.        self.assertEqual(len(saErrors1), 0)
110.        self.assertEqual(len(saErrors2), 0)
111.        self.assertEqual(len(saErrors3), 0)
112.        self.assertEqual(len(saErrors4), 0)
113.
114.        self.assertEqual(result1[0], correct_translation)
115.        self.assertEqual(result2[0], correct_translation)
116.        self.assertEqual(result3[0], correct_translation)
117.        self.assertEqual(result4[0], correct_translation)
118.
119.     def test_turn_left(self):
120.         test_string = \
121.             """turn left
122.         """
123.
124.         correct_translation = \
125.             """rotate_car(WheelDirection.LEFT)
126.         """
127.
128.         result = Compiler.getPythonCode(test_string)
129.         ast = Parser.parseString(test_string)
130.
131.         saErrors = SemanticAnalyzer.analyzeStart(ast)

```

```
130.         self.assertEqual(len(saErrors), 0)
131.
132.         self.assertEqual(len(ast.errors), 0)
133.         self.assertEqual(result[0], correct_translation)
134.
135.     def test_turn_right(self):
136.         test_string = \
137.             """turn right
138.
139.             correct_translation = \
140.                 """rotate_car(WheelDirection.RIGHT)
141.
142.             result = Compiler.getPythonCode(test_string)
143.             ast = Parser.parseString(test_string)
144.
145.             saErrors = SemanticAnalyzer.analyzeStart(ast)
146.             self.assertEqual(len(saErrors), 0)
147.
148.             self.assertEqual(len(ast.errors), 0)
149.             self.assertEqual(result[0], correct_translation)
150.
151.     def test_print(self):
152.         test_string = \
153.             """print "hello world"
154.
155.             correct_translation = \
156.                 """print_to_console("hello world")
157.
158.             result = Compiler.getPythonCode(test_string)
159.             ast = Parser.parseString(test_string)
160.
161.             saErrors = SemanticAnalyzer.analyzeStart(ast)
162.             self.assertEqual(len(saErrors), 0)
163.
164.             self.assertEqual(len(ast.errors), 0)
165.             self.assertEqual(result[0], correct_translation)
166.
167.     def test_declare(self):
168.         test_string = \
169.             """myNum is a number
170.
171.             correct_translation = \
172.                 """myNum = None
173.
174.             result = Compiler.getPythonCode(test_string)
175.             ast = Parser.parseString(test_string)
176.
177.             saErrors = SemanticAnalyzer.analyzeStart(ast)
178.             self.assertEqual(len(saErrors), 0)
179.
180.             self.assertEqual(len(ast.errors), 0)
181.             self.assertEqual(result[0], correct_translation)
182.
183.     def test_assign(self):
184.         test_string = \
185.             """set myVar to otherThing
186.
187.             correct_translation = \
188.                 """myVar = otherThing
189.
190.             result = Compiler.getPythonCode(test_string)
191.             ast = Parser.parseString(test_string)
192.
193.             saErrors = SemanticAnalyzer.analyzeStart(ast)
194.             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.
242.         result = Compiler.getPythonCode(test_string)
243.         ast = Parser.parseString(test_string)
244.
245.         saErrors = SemanticAnalyzer.analyzeStart(ast)
246.         self.assertEqual(len(saErrors), 0)
247.
248.         self.assertEqual(len(ast.errors), 0)
249.         self.assertEqual(result[0], correct_translation)
250.
251.     def test_function_invocation_with_one_parameter(self):
252.         test_string = \
253.             """move5Steps "forwards"
254. """
255.         correct_translation = \
256.             """move5Steps("forwards")
257. """
258.
259.         result = Compiler.getPythonCode(test_string)
260.         ast = Parser.parseString(test_string)
```

```
260.     saErrors = SemanticAnalyzer.analyzeStart(ast)
261.     self.assertEqual(len(saErrors), 0)
262.
263.     self.assertEqual(len(ast.errors), 0)
264.     self.assertEqual(result[0], correct_translation)
265.
266.     def test_function_invocation_with_two_parameters(self):
267.         test_string = \
268.             """define turnLeftThenDriveStraight using numStepsTurn \
269.                 (number) and numStepsDrive (number)
270. {
271. turn left
272. drive forward numStepsTurn steps
273. turn right
274. drive forward numStepsDrive steps
275. }
276. turnLeftThenDriveStraight 5 10
277. """
278.         correct_translation = \
279.             """def turnLeftThenDriveStraight(numStepsTurn, numStepsDrive):
280. rotate_car(WheelDirection.LEFT)
281. translate_car(numStepsTurn, CarDirection.FORWARDS)
282. rotate_car(WheelDirection.RIGHT)
283. translate_car(numStepsDrive, CarDirection.FORWARDS)
284. turnLeftThenDriveStraight(5, 10)
285. """
286.         result = Compiler.getPythonCode(test_string)
287.         ast = Parser.parseString(test_string)
288.
289.         saErrors = SemanticAnalyzer.analyzeStart(ast)
290.         self.assertEqual(len(saErrors), 0)
291.
292.         self.assertEqual(len(ast.errors), 0)
293.         self.assertEqual(result[0], correct_translation)
294.
295.     def test_plus_expression(self):
296.         test_string = \
297.             """print (2 + 3)
298. """
299.         correct_translation = \
300.             """print_to_console(((2) + (3)))
301. """
302.         result = Compiler.getPythonCode(test_string)
303.         ast = Parser.parseString(test_string)
304.
305.         saErrors = SemanticAnalyzer.analyzeStart(ast)
306.         self.assertEqual(len(saErrors), 0)
307.
308.         self.assertEqual(len(ast.errors), 0)
309.         self.assertEqual(result[0], correct_translation)
310.
311.     def test_times_expression(self):
312.         test_string = \
313.             """print (2 * 3)
314. """
315.         correct_translation = \
316.             """print_to_console(((2) * (3)))
317. """
318.         result = Compiler.getPythonCode(test_string)
319.         ast = Parser.parseString(test_string)
320.
321.         saErrors = SemanticAnalyzer.analyzeStart(ast)
322.         self.assertEqual(len(saErrors), 0)
323.
324.         self.assertEqual(len(ast.errors), 0)
```

```
325.         self.assertEqual(result[0], correct_translation)
326.
327.     def test_minus_expression(self):
328.         test_string = \
329.             """print (2 - 3)
330.
331.         correct_translation = \
332.             """print_to_console(((2) - (3)))
333.
334.         result = Compiler.getPythonCode(test_string)
335.         ast = Parser.parseString(test_string)
336.
337.         saErrors = SemanticAnalyzer.analyzeStart(ast)
338.         self.assertEqual(len(saErrors), 0)
339.
340.         self.assertEqual(len(ast.errors), 0)
341.         self.assertEqual(result[0], correct_translation)
342.
343.     def test_divide_expression(self):
344.         test_string = \
345.             """print (2 / 3)
346.
347.         correct_translation = \
348.             """print_to_console(((2) / (3)))
349.
350.         result = Compiler.getPythonCode(test_string)
351.         ast = Parser.parseString(test_string)
352.
353.         saErrors = SemanticAnalyzer.analyzeStart(ast)
354.         self.assertEqual(len(saErrors), 0)
355.
356.         self.assertEqual(len(ast.errors), 0)
357.         self.assertEqual(result[0], correct_translation)
358.
359.     def test_all_expression(self):
360.         test_string = \
361.             """print (1 + 2 * (3 + 4))
362.
363.         correct_translation = \
364.             """print_to_console(((1) + (((2) * (((3) + (4)))))))
365.
366.         result = Compiler.getPythonCode(test_string)
367.         ast = Parser.parseString(test_string)
368.
369.         saErrors = SemanticAnalyzer.analyzeStart(ast)
370.         self.assertEqual(len(saErrors), 0)
371.
372.         self.assertEqual(len(ast.errors), 0)
373.         self.assertEqual(result[0], correct_translation)
374.
375.     def test_assign_num_change(self):
376.         test_string = \
377.             """num is a number
378. set num to 10
379. set num to num*2
380.
381.         correct_translation = \
382.             """num = None
383. num = 10
384. num = ((num) * (2))
385.
386.         result = Compiler.getPythonCode(test_string)
387.         ast = Parser.parseString(test_string)
388.
389.         saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
390.         self.assertEqual(len(saErrors), 0)
391.
392.         self.assertEqual(len(ast.errors), 0)
393.         self.assertEqual(result[0], correct_translation)
394.
395.     def test_assign_easy_num_change(self):
396.         test_string = \
397.             """num is a number
398. set num to 10
399. num2 is a number
400. set num2 to 20
401. set num to num2
402. """
403.         correct_translation = \
404.             """num = None
405. num = 10
406. num2 = None
407. num2 = 20
408. num = num2
409. """
410.         result = Compiler.getPythonCode(test_string)
411.         ast = Parser.parseString(test_string)
412.
413.         saErrors = SemanticAnalyzer.analyzeStart(ast)
414.         self.assertEqual(len(saErrors), 0)
415.
416.         self.assertEqual(len(ast.errors), 0)
417.         self.assertEqual(result[0], correct_translation)
418.
419.     def test_assign_word_print(self):
420.         test_string = \
421.             """color is a word
422. set color to "blue"
423. print color
424. """
425.         correct_translation = \
426.             """color = None
427. color = "blue"
428. print_to_console(color)
429. """
430.         result = Compiler.getPythonCode(test_string)
431.         ast = Parser.parseString(test_string)
432.
433.         saErrors = SemanticAnalyzer.analyzeStart(ast)
434.         self.assertEqual(len(saErrors), 0)
435.
436.         self.assertEqual(len(ast.errors), 0)
437.         self.assertEqual(result[0], correct_translation)
438.
439.     def test_assign_word_print_complicated(self):
440.         test_string = \
441.             """color is a word
442. set color to "blue"
443. print color
444. c2 is a word
445. set c2 to "green"
446. set color to c2
447. """
448.         correct_translation = \
449.             """color = None
450. color = "blue"
451. print_to_console(color)
452. c2 = None
453. c2 = "green"
454. color = c2
```

```
455. """
456.     result = Compiler.getPythonCode(test_string)
457.     ast = Parser.parseString(test_string)
458.
459.     saErrors = SemanticAnalyzer.analyzeStart(ast)
460.     self.assertEqual(len(saErrors), 0)
461.
462.     self.assertEqual(len(ast.errors), 0)
463.     self.assertEqual(result[0], correct_translation)
464.
465.     def test_if_statement(self):
466.         test_string = \
467.             """if 1
468. {
469.     print "yay"
470. }
471. """
472.         correct_translation = \
473.             """if 1:
474. print_to_console("yay")
475. """
476.         result = Compiler.getPythonCode(test_string)
477.         ast = Parser.parseString(test_string)
478.
479.         saErrors = SemanticAnalyzer.analyzeStart(ast)
480.         self.assertEqual(len(saErrors), 0)
481.
482.         self.assertEqual(len(ast.errors), 0)
483.         self.assertEqual(result[0], correct_translation)
484.
485.         def test_if_else_statement(self):
486.             test_string = \
487.                 """if 1
488. {
489.     print "yay"
490. }
491. else
492. {
493.     print "no"
494. }
495. """
496.             correct_translation = \
497.                 """if 1:
498.     print_to_console("yay")
499. else:
500.     print_to_console("no")
501. """
502.             result = Compiler.getPythonCode(test_string)
503.             ast = Parser.parseString(test_string)
504.
505.             saErrors = SemanticAnalyzer.analyzeStart(ast)
506.             self.assertEqual(len(saErrors), 0)
507.
508.             self.assertEqual(len(ast.errors), 0)
509.             self.assertEqual(result[0], correct_translation)
510.
511.         def test_if_statement_nested(self):
512.             test_string = \
513.                 """if 1
514. {
515.     print "yay"
516.     if 1
517.     {
518.         print "yahoo"
519.     }
```



```
520. }
521. """
522.     correct_translation = \
523.         """if 1:
524.     print_to_console("yay")
525.     if 1:
526.         print_to_console("yahoo")
527. """
528.     result = Compiler.getPythonCode(test_string)
529.     ast = Parser.parseString(test_string)
530.
531.     saErrors = SemanticAnalyzer.analyzeStart(ast)
532.     self.assertEqual(len(saErrors), 0)
533.
534.     self.assertEqual(len(ast.errors), 0)
535.     self.assertEqual(result[0], correct_translation)
536.
537.     def test_if_else_statement_nested(self):
538.         test_string = \
539.             """if 1
540. {
541.     print "yay"
542.     if 1
543.     {
544.         print "yahoo"
545.     }
546.     else
547.     {
548.         print "oh no"
549.     }
550. }
551. else
552. {
553.     print "no"
554. }
555. """
556.         correct_translation = \
557.             """if 1:
558.         print_to_console("yay")
559.         if 1:
560.             print_to_console("yahoo")
561.         else:
562.             print_to_console("oh no")
563.     else:
564.         print_to_console("no")
565. """
566.         result = Compiler.getPythonCode(test_string)
567.         ast = Parser.parseString(test_string)
568.
569.         saErrors = SemanticAnalyzer.analyzeStart(ast)
570.         self.assertEqual(len(saErrors), 0)
571.
572.         self.assertEqual(len(ast.errors), 0)
573.         self.assertEqual(result[0], correct_translation)
574.
575.     def test_if_elseif_else_statement(self):
576.         test_string = \
577.             """if 1
578. {
579.     print "yay"
580. }
581. elseif 2
582. {
583.     print "no"
584. }
```

```
585. else
586. {
587.     print "done"
588. }
589. """
590.         correct_translation = \
591.             """if 1:
592.                 print_to_console("yay")
593. elif 2:
594.                 print_to_console("no")
595. else:
596.                 print_to_console("done")
597. """
598.         result = Compiler.getPythonCode(test_string)
599.         ast = Parser.parseString(test_string)
600.
601.         saErrors = SemanticAnalyzer.analyzeStart(ast)
602.         self.assertEqual(len(saErrors), 0)
603.
604.         self.assertEqual(len(ast.errors), 0)
605.         self.assertEqual(result[0], correct_translation)
606.
607.     def test_if_elseif_else_statement_nested(self):
608.         test_string = \
609.             """if 1
610. {
611.     print "yay"
612.     if 1
613.     {
614.         print "yahoo"
615.     }
616.     elif 2
617.     {
618.         print "oh no"
619.     }
620.     else
621.     {
622.         print "here"
623.     }
624. }
625. elif 2
626. {
627.     print "no"
628. }
629. else
630. {
631.     print "done"
632. }
633. """
634.         correct_translation = \
635.             """if 1:
636.                 print_to_console("yay")
637.                 if 1:
638.                     print_to_console("yahoo")
639.                 elif 2:
640.                     print_to_console("oh no")
641.                 else:
642.                     print_to_console("here")
643. elif 2:
644.                 print_to_console("no")
645. else:
646.                 print_to_console("done")
647. """
648.         result = Compiler.getPythonCode(test_string)
649.         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.         elif 2
678.         {
679.             print "no"
680.             if 1
681.             {
682.                 print "hi"
683.             }
684.         }
685.         elif 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")
```

```

715.         elif 2:
716.             print_to_console("no")
717.             if 1:
718.                 print_to_console("hi")
719.         elif 3:
720.             print_to_console("yes")
721.         else:
722.             if 5:
723.                 print_to_console("works")
724.                 print_to_console("end")
725.     else:
726.         print_to_console("no")
727.     """
728.     result = Compiler.getPythonCode(test_string)
729.     ast = Parser.parseString(test_string)
730.
731.     saErrors = SemanticAnalyzer.analyzeStart(ast)
732.     self.assertEqual(len(saErrors), 0)
733.
734.     self.assertEqual(len(ast.errors), 0)
735.     self.assertEqual(result[0], correct_translation)
736.
737.     def test_comment_singleline(self):
738.         test_string = \
739.             """:) this is a single line comment
740.         """
741.         correct_translation = \
742.             """
743.         result = Compiler.getPythonCode(test_string)
744.         ast = Parser.parseString(test_string)
745.
746.         saErrors = SemanticAnalyzer.analyzeStart(ast)
747.         self.assertEqual(len(saErrors), 0)
748.
749.         self.assertEqual(len(ast.errors), 0)
750.         self.assertEqual(result[0], correct_translation)
751.
752.     def test_comment_multiline(self):
753.         test_string = \
754.             """:- ( this is
755. a multiline
756. comment
757. :-)
758.         """
759.         correct_translation = \
760.             """
761.         result = Compiler.getPythonCode(test_string)
762.         ast = Parser.parseString(test_string)
763.
764.         saErrors = SemanticAnalyzer.analyzeStart(ast)
765.         self.assertEqual(len(saErrors), 0)
766.
767.         self.assertEqual(len(ast.errors), 0)
768.         self.assertEqual(result[0], correct_translation)
769.
770.     def test_loop_for(self):
771.         test_string = \
772.             """myCounter is a number
773. set myCounter to 10
774. repeat myCounter times
775. {
776.     drive forward 1 step
777.     print myCounter
778. }
779.         """

```

```
780.         correct_translation = \
781.             """myCounter = None
782. myCounter = 10
783. for x in range(myCounter):
784.     translate_car(1, CarDirection.FORWARDS)
785.     print_to_console(myCounter)
786. """
787.         result = Compiler.getPythonCode(test_string)
788.         ast = Parser.parseString(test_string)
789.
790.         saErrors = SemanticAnalyzer.analyzeStart(ast)
791.         self.assertEqual(len(saErrors), 0)
792.
793.         self.assertEqual(len(ast.errors), 0)
794.         self.assertEqual(result[0], correct_translation)
795.
796.     def test_loop_for_nested(self):
797.         test_string = \
798.             """myCounter is a number
799. set myCounter to 10
800. myCounter2 is a number
801. set myCounter2 to 10
802. repeat myCounter times
803. {
804.     drive forward 1 step
805.     repeat myCounter2 times
806.     {
807.         drive forward 1 step
808.     }
809. }
810. """
811.         correct_translation = \
812.             """myCounter = None
813. myCounter = 10
814. myCounter2 = None
815. myCounter2 = 10
816. for x in range(myCounter):
817.     translate_car(1, CarDirection.FORWARDS)
818.     for x in range(myCounter2):
819.         translate_car(1, CarDirection.FORWARDS)
820. """
821.         result = Compiler.getPythonCode(test_string)
822.         ast = Parser.parseString(test_string)
823.
824.         saErrors = SemanticAnalyzer.analyzeStart(ast)
825.         self.assertEqual(len(saErrors), 0)
826.
827.         self.assertEqual(len(ast.errors), 0)
828.         self.assertEqual(result[0], correct_translation)
829.
830.     def test_loop_while(self):
831.         test_string = \
832.             """myCounter is a number
833. set myCounter to 1
834. repeat if myCounter is not 5
835. {
836.     drive forward 1 step
837.     set myCounter to myCounter + 1
838. }
839. """
840.         correct_translation = \
841.             """myCounter = None
842. myCounter = 1
843. while myCounter != 5:
844.     translate_car(1, CarDirection.FORWARDS)
```

```

845.     myCounter = ((myCounter) + (1))
846. """
847.         result = Compiler.getPythonCode(test_string)
848.         ast = Parser.parseString(test_string)
849.
850.         saErrors = SemanticAnalyzer.analyzeStart(ast)
851.         self.assertEqual(len(saErrors), 0)
852.
853.         self.assertEqual(len(ast.errors), 0)
854.         self.assertEqual(result[0], correct_translation)
855.
856. #####stopped SA tests here
857.
858.     def test_loop_while_nested(self):
859.         test_string = \
860.             """myCounter is a number
861. set myCounter to 1
862. myCounter2 is a number
863. repeat if myCounter is not 5
864. {
865.     drive forward 1 step
866.     set myCounter2 to 0
867.     repeat if myCounter2 is not 5
868.     {
869.         drive forward 1 step
870.         set myCounter2 to myCounter2 + 1
871.     }
872.     set myCounter to myCounter + 1
873. }
874. """
875.         correct_translation = \
876.             """myCounter = None
877. myCounter = 1
878. myCounter2 = None
879. while myCounter != 5:
880.     translate_car(1, CarDirection.FORWARDS)
881.     myCounter2 = 0
882.     while myCounter2 != 5:
883.         translate_car(1, CarDirection.FORWARDS)
884.         myCounter2 = ((myCounter2) + (1))
885.     myCounter = ((myCounter) + (1))
886. """
887.         result = Compiler.getPythonCode(test_string)
888.         ast = Parser.parseString(test_string)
889.
890.         saErrors = SemanticAnalyzer.analyzeStart(ast)
891.         self.assertEqual(len(saErrors), 0)
892.
893.         self.assertEqual(len(ast.errors), 0)
894.         self.assertEqual(result[0], correct_translation)
895.
896.     def test_boolean_opeartors(self):
897.         test_string = \
898.             """if 1 < 2
899. {
900.     print "yes"
901. }
902. elif 1 is 2
903. {
904.     print "yes"
905. }
906. if 1 >= 2
907. {
908.     print "no"
909. }

```

```
910. elif 1 is not 2
911. {
912.     print "yes"
913. }
914. elif 1 > 2
915. {
916.     print "yes"
917. }
918. elif 1 < 2
919. {
920.     print "yes"
921. }
922. """
923.         correct_translation = \
924.             """if 1 < 2:
925.                 print_to_console("yes")
926. elif 1 == 2:
927.                 print_to_console("yes")
928. if 1 >= 2:
929.                 print_to_console("no")
930. elif 1 != 2:
931.                 print_to_console("yes")
932. elif 1 > 2:
933.                 print_to_console("yes")
934. elif 1 < 2:
935.                 print_to_console("yes")
936. """
937.         result = Compiler.getPythonCode(test_string)
938.         ast = Parser.parseString(test_string)
939.
940.         self.assertEqual(len(ast.errors), 0)
941.         self.assertEqual(result[0], correct_translation)
942.
943.     def test_string_concatenation(self):
944.         test_string = \
945.             """print "hey" ++ myWord
946. """
947.         correct_translation = \
948.             """print_to_console((str("hey") + str(myWord)))
949. """
950.         result = Compiler.getPythonCode(test_string)
951.         ast = Parser.parseString(test_string)
952.
953.         self.assertEqual(len(ast.errors), 0)
954.         self.assertEqual(result[0], correct_translation)
955.
956.     def test_string_concatenation_complicated(self):
957.         test_string = \
958.             """print "hey" ++ myWord ++ "now"
959. """
960.         correct_translation = \
961.             """print_to_console((str((str("hey") + str(myWord))) + str("now")))
962. """
963.         result = Compiler.getPythonCode(test_string)
964.         ast = Parser.parseString(test_string)
965.
966.         self.assertEqual(len(ast.errors), 0)
967.         self.assertEqual(result[0], correct_translation)
968.
969.     def test_get_car_position(self):
970.         test_string = \
971.             """print getCarPosition
972. """
973.         correct_translation = \
974.             """print_to_console(getCurrentPosition())
```

```

975. """
976.         result = Compiler.getPythonCode(test_string)
977.         ast = Parser.parseString(test_string)
978.
979.         self.assertEqual(len(ast.errors), 0)
980.         self.assertEqual(result[0], correct_translation)
981.
982.     def test_can_move(self):
983.         test_string = \
984.             """if canDrive forward 5 steps
985. {
986.     drive forward 5 steps
987. }
988. """
989.         correct_translation = \
990.             """if can_move(5, CarDirection.FORWARDS):
991. translate_car(5, CarDirection.FORWARDS)
992. """
993.         result = Compiler.getPythonCode(test_string)
994.         ast = Parser.parseString(test_string)
995.
996.         self.assertEqual(len(ast.errors), 0)
997.         self.assertEqual(result[0], correct_translation)
998.
999.     def test_if_plus(self):
1000.         test_string = \
1001.             """if 2 + 5
1002. {
1003.     print "yay"
1004. }
1005. """
1006.         correct_translation = \
1007.             """if ((2) + (5)):
1008. print_to_console("yay")
1009. """
1010.         result = Compiler.getPythonCode(test_string)
1011.         ast = Parser.parseString(test_string)
1012.
1013.         self.assertEqual(len(ast.errors), 0)
1014.         self.assertEqual(result[0], correct_translation)
1015.
1016.     def test_template(self):
1017.         test_string = \
1018.             """
1019. """
1020.         correct_translation = \
1021.             """
1022. """
1023.         result = Compiler.getPythonCode(test_string)
1024.         ast = Parser.parseString(test_string)
1025.
1026.         self.assertEqual(len(ast.errors), 0)
1027.         self.assertEqual(result[0], correct_translation)
1028.
1029. class SymbolTableTests(unittest.TestCase):
1030.
1031.     def test_symbol_table_add_entry(self):
1032.         '''Tests the SymbolTableEntry.addEntry() function.'''
1033.
1034.         table = SymbolTable.SymbolLookupTable()
1035.
1036.         entry1 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
1037.         entry2 = SymbolTable.SymbolTableEntry("name1", "word", [0], None, [])
1038.
1039.         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.
```

```

1104.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1105.         self.assertEqual(len(saErrors), 1)
1106.
1107.     def test_set_undeclared_var(self):
1108.         test_string = \
1109.             """set myNum to 10
1110.
1111.
1112.         ast = Parser.parseString(test_string)
1113.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1114.
1115.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1116.         #should have an error saying mySecondNum doesnt exist
1117.         self.assertEqual(len(saErrors), 1)
1118.
1119.     def test_general_access_undeclared_var(self):
1120.         test_string = \
1121.             """myNum is a number
1122. set myNum to mySecondNum
1123.
1124.
1125.         ast = Parser.parseString(test_string)
1126.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1127.
1128.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1129.         #should have an error saying mySecondNum doesnt exist
1130.         self.assertEqual(len(saErrors), 1)
1131.
1132.     def test_var_declared_in_if(self):
1133.         test_string = \
1134.             """if 1
1135. {
1136.     myNum is a number
1137.     set myNum to 10
1138.     print myNum
1139. }
1140.
1141.
1142.         ast = Parser.parseString(test_string)
1143.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1144.
1145.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1146.         self.assertEqual(len(saErrors), 0)
1147.
1148.     def test_var_declared_in_if_accessed_outside(self):
1149.         test_string = \
1150.             """if 1
1151. {
1152.     myNum is a number
1153.     set myNum to 10
1154.     print myNum
1155. }
1156. print myNum
1157.
1158.
1159.         ast = Parser.parseString(test_string)
1160.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1161.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1162.         #should have an error about printing myNum after the if
1163.         self.assertEqual(len(saErrors), 1)
1164.
1165.     def test_var_declared_in_func(self):
1166.         test_string = \
1167.             """define moveForwardFive
1168. {

```

```
1169. myNum is a number
1170. set myNum to 5
1171. drive forward myNum steps
1172. }
1173. """
1174.
1175.         ast = Parser.parseString(test_string)
1176.
1177.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1178.         self.assertEqual(len(saErrors), 0)
1179.
1180.     def test_var_declared_in_func_accessed_outside(self):
1181.         test_string = \
1182.             """define moveForwardFive
1183. {
1184.     myNum is a number
1185.     set myNum to 5
1186.     drive forward myNum steps
1187. }
1188. moveForwardFive
1189. drive forward myNum steps
1190. """
1191.
1192.         ast = Parser.parseString(test_string)
1193.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1194.
1195.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1196.         self.assertEqual(len(saErrors), 1)
1197.
1198.     def test_access_passed_in_var_in_func(self):
1199.         test_string = \
1200.             """define moveForward using numSteps (number)
1201. {
1202.     drive forward numSteps steps
1203. }
1204. """
1205.
1206.         ast = Parser.parseString(test_string)
1207.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1208.
1209.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1210.         self.assertEqual(len(saErrors), 0)
1211.
1212.     def test_access_passed_in_var_in_if_in_func(self):
1213.         test_string = \
1214.             """define moveForwardFive using numSteps (number)
1215. {
1216.     drive forward numSteps steps
1217.     if 1
1218.     {
1219.         myCounter is a number
1220.         set myCounter to 10
1221.         drive forward 10 steps
1222.     }
1223. }
1224. """
1225.
1226.         ast = Parser.parseString(test_string)
1227.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1228.
1229.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1230.         self.assertEqual(len(saErrors), 0)
1231.
1232.     def test_call_func_anywhere(self):
1233.         test_string = \
```

```
1234.         """moveForwardFive
1235. define moveForwardFive
1236. {
1237.     drive forward 5 steps
1238. }
1239. moveForwardFive
1240. """
1241.
1242.         ast = Parser.parseString(test_string)
1243.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1244.
1245.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1246.         self.assertEqual(len(saErrors), 0)
1247.
1248.         def test_declare_func_in_func(self):
1249.             test_string = \
1250.                 """define moveForwardFive
1251. {
1252.     drive forward 5 steps
1253.     define moveForwardTen
1254.     {
1255.         drive forward 10 steps
1256.     }
1257. }
1258. """
1259.
1260.             ast = Parser.parseString(test_string)
1261.             self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1262.
1263.             saErrors = SemanticAnalyzer.analyzeStart(ast)
1264.             self.assertEqual(len(saErrors), 1)
1265.
1266.             def test_call_funciton_with_param(self):
1267.                 test_string = \
1268.                     """define moveForwardFive using numSteps (number)
1269. {
1270.     drive forward numSteps steps
1271. }
1272. myNum is a number
1273. set myNum to 10
1274. moveForwardFive myNum
1275. """
1276.
1277.                 ast = Parser.parseString(test_string)
1278.                 self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1279.
1280.                 saErrors = SemanticAnalyzer.analyzeStart(ast)
1281.                 self.assertEqual(len(saErrors), 0)
1282.
1283.                 def test_access_func_param_outside(self):
1284.                     test_string = \
1285.                         """define moveForwardFive using numSteps (number)
1286. {
1287.     drive forward numSteps steps
1288. }
1289. myNum is a number
1290. set myNum to 10
1291. moveForwardFive myNum
1292. print numSteps
1293. """
1294.
1295.                     ast = Parser.parseString(test_string)
1296.                     self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1297.
1298.                     saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
1299.         #should fail since numSteps is only inside the function
1300.         self.assertEqual(len(saErrors), 1)
1301.
1302.     def test_assignments_var_to_var(self):
1303.         test_string = \
1304.             """myNum is a number
1305. set myNum to 10
1306. myWord is a word
1307. set myWord to "hello"
1308. set myNum to myWord
1309. set myWord to myNum
1310. """
1311.
1312.         ast = Parser.parseString(test_string)
1313.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1314.
1315.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1316.         self.assertEqual(len(saErrors), 2)
1317.
1318.
1319.     def test_assignments_var_number(self):
1320.         test_string = \
1321.             """myWord is a word
1322. set myWord to 10
1323. """
1324.
1325.         ast = Parser.parseString(test_string)
1326.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1327.
1328.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1329.         self.assertEqual(len(saErrors), 1)
1330.
1331.     def test_assignments_var_str_literal(self):
1332.         test_string = \
1333.             """myNum is a number
1334. set myNum to "hello"
1335. """
1336.
1337.         ast = Parser.parseString(test_string)
1338.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1339.
1340.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1341.         self.assertEqual(len(saErrors), 1)
1342.
1343.     def test_compare_vars_num_to_string(self):
1344.         test_string = \
1345.             """myNum is a number
1346. set myNum to 10
1347. myWord is a word
1348. set myWord to "hello"
1349. if myWord > myNum
1350. {
1351.     print "bad"
1352. }
1353. """
1354.
1355.         ast = Parser.parseString(test_string)
1356.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1357.
1358.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1359.         self.assertEqual(len(saErrors), 1)
1360.
1361.     def test_compare_num_to_stringliteral(self):
1362.         test_string = \
1363.             """if 10 > "hello"
```

```
1364. {
1365.     print "bad"
1366. }
1367. """
1368.
1369.     ast = Parser.parseString(test_string)
1370.     self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1371.
1372.     saErrors = SemanticAnalyzer.analyzeStart(ast)
1373.     self.assertEqual(len(saErrors), 1)
1374.
1375.     def test_compare_num_to_num(self):
1376.         test_string = \
1377.             """if 10 > 5
1378. {
1379.     print "good"
1380. }
1381. """
1382.
1383.         ast = Parser.parseString(test_string)
1384.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1385.
1386.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1387.         self.assertEqual(len(saErrors), 0)
1388.
1389.     def test_compare_strL_to_strL(self):
1390.         test_string = \
1391.             """if "hello" is "hi"
1392. {
1393.     print "good"
1394. }
1395. """
1396.
1397.         ast = Parser.parseString(test_string)
1398.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1399.
1400.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1401.         self.assertEqual(len(saErrors), 0)
1402.
1403.     def test_compare_strL_to_strL_not(self):
1404.         test_string = \
1405.             """if "hello" is not "hi"
1406. {
1407.     print "good"
1408. }
1409. """
1410.
1411.         ast = Parser.parseString(test_string)
1412.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1413.
1414.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1415.         self.assertEqual(len(saErrors), 0)
1416.
1417.     def test_compare_num_str_is(self):
1418.         test_string = \
1419.             """if 10 is "hello"
1420. {
1421.     print "bad"
1422. }
1423. """
1424.
1425.         ast = Parser.parseString(test_string)
1426.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1427.
1428.         saErrors = SemanticAnalyzer.analyzeStart(ast)
```

```
1429.         self.assertEqual(len(saErrors), 1)
1430.
1431.     def test_num_params_passing(self):
1432.         test_string = \
1433.             """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1434. {
1435. }
1436. moveForwardFiveAndTurn 10 "left"
1437. """
1438.
1439.         ast = Parser.parseString(test_string)
1440.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1441.
1442.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1443.         self.assertEqual(len(saErrors), 0)
1444.
1445.     def test_num_params_passing_2(self):
1446.         test_string = \
1447.             """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1448. {
1449. }
1450. moveForwardFiveAndTurn 10 "hi" "extra"
1451. """
1452.
1453.         ast = Parser.parseString(test_string)
1454.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1455.
1456.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1457.         self.assertEqual(len(saErrors), 1)
1458.
1459.     def test_num_params_passing_3(self):
1460.         test_string = \
1461.             """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1462. {
1463. }
1464. moveForwardFiveAndTurn 10 15
1465. """
1466.
1467.         ast = Parser.parseString(test_string)
1468.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1469.
1470.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1471.         self.assertEqual(len(saErrors), 1)
1472.
1473.     def test_num_params_passing_4(self):
1474.         test_string = \
1475.             """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1476. {
1477.     print numSteps
1478.     print direction
1479. }
1480. moveForwardFiveAndTurn "hello" "hi"
1481. """
1482.
1483.         ast = Parser.parseString(test_string)
1484.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1485.
1486.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1487.         self.assertEqual(len(saErrors), 1)
1488.
1489.     def test_num_params_passing_5(self):
1490.         test_string = \
```

```
1494.         """define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1495.     {
1496.         print numSteps
1497.         print direction
1498.     }
1499. moveForwardFiveAndTurn "left" 10
1500. """
1501.
1502.         ast = Parser.parseString(test_string)
1503.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1504.
1505.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1506.         self.assertEqual(len(saErrors), 1)
1507.
1508.     def test_built_in_functions_params(self):
1509.         test_string = \
1510.             """drive forwards five steps
1511. """
1512.
1513.         ast = Parser.parseString(test_string)
1514.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1515.
1516.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1517.         self.assertEqual(len(saErrors), 1)
1518.
1519.     def test_while_loop(self):
1520.         test_string = \
1521.             """repeat 5 times
1522. {
1523.     print "hi"
1524. }
1525. """
1526.
1527.         ast = Parser.parseString(test_string)
1528.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1529.
1530.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1531.         self.assertEqual(len(saErrors), 0)
1532.
1533.     def test_while_loop_bad(self):
1534.         test_string = \
1535.             """repeat five times
1536. {
1537.     print "hi"
1538. }
1539. """
1540.
1541.         ast = Parser.parseString(test_string)
1542.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1543.
1544.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1545.         self.assertEqual(len(saErrors), 1)
1546.
1547.     def test_calling_nonexistant_function(self):
1548.         test_string = \
1549.             """fullTurn "left"
1550. """
1551.
1552.         ast = Parser.parseString(test_string)
1553.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1554.
1555.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1556.         self.assertEqual(len(saErrors), 1)
1557.
1558.     def test_course2(self):
```



```
1559.     test_string = \  
1560.         """define fullTurn using direction (word)  
1561. {  
1562.     if direction is "left"  
1563.     {  
1564.         turn left  
1565.         turn left  
1566.     }  
1567.     else  
1568.     {  
1569.         turn right  
1570.         turn right  
1571.     }  
1572. }  
1573.  
1574. define driveThenFullTurn using numSteps (number) and direction (word)  
1575. {  
1576.     drive forward numSteps steps  
1577.     fullTurn direction  
1578. }  
1579.  
1580. driveThenFullTurn 0 "right"  
1581. driveThenFullTurn 25 "left"  
1582. driveThenFullTurn 18 "left"  
1583.  
1584. repeat 2 times  
1585. {  
1586.  
1587.     driveThenFullTurn 50 "right"  
1588.     driveThenFullTurn 15 "right"  
1589.     driveThenFullTurn 50 "left"  
1590.     driveThenFullTurn 16 "left"  
1591. }  
1592.  
1593. driveThenFullTurn 50 "right"  
1594. driveThenFullTurn 15 "right"  
1595. driveThenFullTurn 50 "left"  
1596. driveThenFullTurn 10 "left"  
1597. """  
1598.  
1599.     ast = Parser.parseString(test_string)  
1600.     self.assertEqual(len(ast.errors), 0, "Test failed at parser.")  
1601.  
1602.     saErrors = SemanticAnalyzer.analyzeStart(ast)  
1603.     self.assertEqual(len(saErrors), 0)  
1604.  
1605.     def test_var_uninitialized(self):  
1606.         test_string = \  
1607.             """myNum is a number  
1608. drive forward myNum steps  
1609. """  
1610.  
1611.         ast = Parser.parseString(test_string)  
1612.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")  
1613.  
1614.         saErrors = SemanticAnalyzer.analyzeStart(ast)  
1615.         self.assertEqual(len(saErrors), 1)  
1616.  
1617.     def test_call_function_in_if_and_while(self):  
1618.         test_string = \  
1619.             """moveForwardFiveAndTurn 10 "left"  
1620. if 1  
1621. {  
1622.     moveForwardFiveAndTurn 10 "left"  
1623. }
```

```
1624. repeat 2 times
1625. {
1626.     moveForwardFiveAndTurn 10 "left"
1627. }
1628. define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1629. {
1630.     print numSteps
1631.     print direction
1632.     fullTurn direction
1633. }
1634. define fullTurn using direction (word)
1635. {
1636.     if direction is "left"
1637.     {
1638.         turn left
1639.         turn left
1640.     }
1641.     else
1642.     {
1643.         turn right
1644.         turn right
1645.     }
1646.     moveForwardFiveAndTurn 5 direction
1647. }
1648. """
1649.
1650.         ast = Parser.parseString(test_string)
1651.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1652.
1653.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1654.         self.assertEqual(len(saErrors), 0)
1655.
1656.         def test_call_function_before_declared(self):
1657.             test_string = \
1658.                 """moveForwardFiveAndTurn 10 "left"
1659. if 1
1660. {
1661.     moveForwardFiveAndTurn 10 "left"
1662. }
1663. repeat 2 times
1664. {
1665.     moveForwardFiveAndTurn 10 "left"
1666. }
1667. drive forward 5 steps
1668. define moveForwardFiveAndTurn using numSteps (number) and direction (word)
1669. {
1670.     print numSteps
1671.     print direction
1672. }
1673. """
1674.
1675.         ast = Parser.parseString(test_string)
1676.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1677.
1678.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1679.         self.assertEqual(len(saErrors), 0)
1680.
1681.         def test_template(self):
1682.             test_string = \
1683.                 """
1684. """
1685.
1686.         ast = Parser.parseString(test_string)
1687.         self.assertEqual(len(ast.errors), 0, "Test failed at parser.")
1688.
```

```
1689.         saErrors = SemanticAnalyzer.analyzeStart(ast)
1690.         self.assertEqual(len(saErrors), 0)
1691.
1692. if __name__ == '__main__':
1693.     suite = unittest.TestLoader().loadTestsFromTestCase(TranslatorTests)
1694.     unittest.TextTestRunner(verbosity=2).run(suite)
```