Integration Tests

List of Functions

init.h	Truck.h	Shipment.h	utils.h
1. init()	2. AssignTruck() 3. calculateUtilizationScore()	4. read() 5. validate()	6. getAllTruckPaths() 7. getBestRoute() 8. containsDestination() 9. printRoute() 10. isBuilding()

The above table shows our list of functions. Since function #2 “AssignTruck()” uses all the functions inside utils.h (6-10), and uses function #3, the only remaining function we have to test is #1, 4, and 5. Because Function #4 is a function that reads from input buffer, and does not read from input stream or return any values, the read() function will not be tested.
Therefore, for our integration tests, we will be utilizing init(), validate(), and AssignTruck() functions to test all of our functions. *Integration Tests T01-T10 are ran through the Unit Test Module automatically.
Expected Return Values:
2 = Blue Truck 4 = Green Truck 8 = Yellow Truck
0 = Cannot deliver today -1 = invalid package

Integration Test #	Description	Test Data	Expected Result
T01	Tests with all valid variables and package destination closest to green truck	init() struct Shipment package = {10, 0.5, {7, 24}}	4
T02	Tests with all valid variables and package destination closest to blue truck	Init() struct Shipment Package = {10, 0.5, {11,11}}	2
T03	Tests with all valid variables and package destination closest to yellow truck	Init() struct Shipment Package = {10, 0.5, {14,3}}	8
T04	Tests with invalid package weight and valid destination closest to yellow truck	Init() struct Shipment Package = {2000, 0.5, {14,3}}	-1
T05	Tests with invalid package volume and valid destination closest to yellow truck	Init() struct Shipment Package = {400, 0.42, {14,3}}	-1
T06	Tests with invalid package destination (not inside building), but destination closet to yellow truck	Init() struct Shipment Package = {400, 0.5, {20,0}}	-1

T07	Tests with all valid variables and package destinations. The truck closest to package is green (4) but is full, so second closest truck is blue (2)	Init() Trucks[1].cargoWeight=999; struct Shipment Package = {400, 0.5, {7,24}}	2
T08	Tests with all valid variables and package destinations. The truck closest to package is yellow (8) but is full, so second closest truck is blue (2)	Init() Trucks[2].cargoWeight=999; struct Shipment Package = {400, 0.5, {7,24}}	2
T09	Tests with all valid variables, but the closest truck to the package (blue - 2) is full, and then second closest truck is yellow	Init() Trucks[0].cargoWeight = 999; struct Shipment Package = {400, 0.5, {15,11}};	8
T10	Tests with all valid variables and package destinations. The truck closest to package is blue (2) but all three trucks are full	Init() Trucks[0].cargoWeight=999 Trucks[1].cargoWeight=999 Trucks[2].cargoWeight=999 struct Shipment Package = {400, 0.5, {15,11}}	0

Manual Integration Testing

In order to test the program with the read() function, I had to manually test the program. Therefore, I ran the program and inputted the following sample data:

20 .5 28x
20 2 12L
1005 .5 12L
200 1.0 8Y
500 1.0 8Y
500 1.0 8Y
0 0 X
0 0 x

```
=================
Seneca Deliveries
=================
Enter shipment weight, box size and destination (0 0 x to stop): 20 .5 28x
Invalid destination
Enter shipment weight, box size and destination (0 0 x to stop): 20 2 12L
Invalid size
Enter shipment weight, box size and destination (0 0 x to stop): 1005 .5 12L
Invalid weight (must be 1-1000 Kg.)
Enter shipment weight, box size and destination (0 0 x to stop): 200 1.0 8Y
Ship on GREEN LINE, 7Y, 8Y
Enter shipment weight, box size and destination (0 0 x to stop): 500 1.0 8Y
Ship on GREEN LINE, 7Y, 8Y
Enter shipment weight, box size and destination (0 0 x to stop): 500 1.0 8Y
Ship on BLUE LINE, 7Y, 8Y, 9Y, 10Y, 11Y, 12Y, 13Y, 14Y, 15Y, 16Y
Enter shipment weight, box size and destination (0 0 x to stop): 0 0 X
Invalid weight (must be 1-1000 Kg.)
Enter shipment weight, box size and destination (0 0 x to stop): 0 0 x
Thank you for shipping with Seneca!
```