Test Description

Test Name or ID: validatePackageWeight

Test Type: Black box

Description: This test aims to confirm that the validatePackageWeight function works as expected. This function checks if a package's weight falls within acceptable boundaries.

Setup: make a TruckStatus structure variable with the right amount of space and to test it with different weights for packages

Test Function: int validatePackageWeight(const struct TruckStatus *truckStatus, double weight);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T001	Package weight is within the available weight limit of the truck.	TruckStatus { availableWeight: 1200.0 } Package weight: 1000.0	1	1	Pass
T002	Package weight exceeds the available weight limit of the truck.	TruckStatus { availableWeight: 1200.0 } Package weight: 1300.0	0	0	Pass
T003	Package weight is zero.	TruckStatus { availableWeight: 1200.0 } Package weight: 0.0	0	0	Pass
T004	Package weight is negative.	TruckStatus { availableWeight: 1200.0 } Package weight: -500.0	0	0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID:validatePackageVolume

Test Type: Black box

Description: This test ensures that the function validatePackageVolume correctly validates the volume of a package1 against the available volume of a truck.

Setup: Create a TruckStatus struct instance with known availableVolume value. Create a package with a known volume. Call the validatePackageVolume function with these parameters.

Test Function: int validatePackageVolume(const struct TruckStatus *truckStatus, double volume)

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T005	Package volume is equal to the available volume limit of the truck.	TruckStatus { availableVolume: 50.0 } Package volume: 50.0	1	1	Pass
T006	Package volume is negative.	TruckStatus { availableVolume: 50.0 } Package volume: -30.0	0	0	Pass
T007	Package volume is within the available volume limit of the truck.	TruckStatus { availableVolume: 50.0 } Package volume: 40.0	1	1	Pass
T008	Package volume is within the available volume limit of the truck (zero volume).	TruckStatus { availableVolume: 0.0 } Package volume: 0.0	0	0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: isTruckAvailable

Test Type: Black box

Description: once we get the package information, we need to make sure if the truck can load the package based on the package weight and volume.

Setup: create a struct Package variable and a struct DeliveryStatus variable to checkout if the function works

Test Function: int isTruckAvailable(const struct Package package, const struct DeliveryStatus status, char truckColor);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T009	package too heavy to fit in the blue truck (Blue truck can load weight 1200.0 and volume 50.0)	Package { weight:1300.0, volume: 30.0 },	0	0	Pass

T010	package too big to fit in the blue truck (Blue truck can load weight 1200.0 and volume 50.0)	Package {
		weight:1000.0,
		volume: 55.0
		}
			0	0	Pass

T011	package too big and too heavy to fit in the blue truck (Blue truck can load weight 1200.0 and volume 50.0)	Package {
		weight:1300.0,
		volume: 55.0
		}
			0	0	Pass

T012	package can fit in the blue truck (Blue truck can load weight 1200.0 and volume 50.0)	Package {
		weight:1000.0,
		volume: 30.0
		}
			1	1	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID:

Test Type: Black box

Description: If there are two trucks with the same shortest path length, we need to compare the capacity

Setup: create two TruckStatus variable

Test Function: int compareTruckCapacity(const struct TruckStatus *truck1, const struct TruckStatus *truck2);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T013	Truck 1 has more capacity remaining	truck1: {availableWeight: 600, availableVolume: 25}, truck2: {availableWeight: 400, availableVolume: 20}	1	1	Pass
T014	Truck 2 has more capacity remaining	truck1: {availableWeight: 400, availableVolume: 20}, truck2: {availableWeight: 600, availableVolume: 25}	-1	-1	Pass
T015	Both trucks have the same capacity remaining	truck1: {availableWeight: 500, availableVolume: 25}, truck2: {availableWeight: 500, availableVolume: 25}	0	0	Pass
T016	Truck 1 is fully loaded	truck1: {availableWeight: 0, availableVolume: 0}, truck2: {availableWeight: 600, availableVolume: 30}	-1	-1	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID:

Test Type: Black box

Description: check the function can calculate map rows correctly

Setup: create a struct Map variable

Test Function: int getNumRows(const struct Map* map);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T017	Standard map with predefined number of rows	Map with numRows initialized to 25	25	25	Pass
T018	Map with less than the standard number of rows	Map with numRows initialized to 10	10	10	Pass
T019	Map with more rows than columns	Map with numRows initialized to 30	30	30	Pass
T020	Map with number of rows equal to zero	Map with numRows explicitly set to 0	0	0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: distance

Test Type: Black box

Description: To check if the function can calculate the distance between two points correctly

Setup: create two struct Point variable

Test Function: double distance(const struct Point* p1, const struct Point* p2);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T021	Two points at the same location	p1: {row: 5, col: 5}, p2: {row: 5, col: 5}	0 (distance is zero)	0	Pass
T022	Points with same row or column	p1: {row: 3, col: 4}, p2: {row: 3, col: 8}	4 (horizontal distance)	4	Pass
T023	Points diagonally placed	p1: {row: 1, col: 1}, p2: {row: 4, col: 5}	5 (3-4-5 triangle distance)	5	Pass

T024	Points with negative coordinates	p1: {row: -1, col: -1}, p2: {row: -4, col: -5}	5 (distance should be positive)	5	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

—----------white box —---

Test Name or ID: validatePackageWeight

Test Type: white box

Description: This test is meant to check if the validatePackageWeight function works correctly in white box test.

Setup: Create a TruckStatus structure variable with enough room and try it out with various package weights.

Test Function: int validatePackageWeight(double weight);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T025	To test with valid weight within range	weight = 600	1	1	Pass
T026	To test with weight equal to maximum allowed weight	weight = 1200	1	1	Pass
T027	To test with weight exceeding maximum allowed weight	weight = 1300	0	0	Pass
T028	To test with weight equal to 0	weight = 0	0	0	Pass

Bugs Found:

These scenarios cover various cases such as valid weights within range, weight equal to the maximum allowed, weight greater than the maximum allowed, and weight equal to 0. All tests passed without any bugs found.

Test Name or ID: validatePackageVolume

Test Type: white box

Description: To test the validatePackageVolume function's ability to correctly validate package volumes.

Setup: Create a TruckStatus object with a known available space. We also have a package with a known size. Now, we'll check if the package fits into the truck's available space.

Test Function: int validatePackageVolume(double volume);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T029	To test with valid volume	volume = 1.0	1	1	Pass
T030	To test with volume equal to maximum allowed volume (5.0)	volume = 5.0	1	1	Pass
T031	To test with volume exceeding maximum allowed volume	volume = 6.0	0	0	Pass
T032	To test with volume less than minimum allowed volume	volume = 0.2	0	0	Pass

Bugs Found:

This test scenario aims to thoroughly examine the validatePackageVolume function by testing it with various input volumes, including valid, maximum, and invalid values, to ensure that it correctly validates package volumes according to the specified criteria.

Test Name or ID:validateDestination

Test Type: white box

Description: To test the validateDestination function's ability to correctly validate destination points within the map boundaries.

Setup: create two struct Point variable

Test Function: int validateDestination(const struct Map *map, const struct Point destination);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T033	Test with a destination point within the map boundaries	destination (row=3, col=4)	1	1	Pass
T034	Test with a destination point outside the map boundaries	destination (row=10, col=15)	0	0	Pass
T035	Test with a destination point on the edge of the map boundaries	destination (row=0, col=5)	0	0	Pass
T036	Test with a destination point at the corner of the map boundaries	destination (row=5, col=0)	0	0	Pass

Bugs Found:

This test scenario aims to thoroughly examine the **validateDestination** function by testing it with various destination points, including points within and outside the map boundaries, as well as points on the edge and corner of the map, to ensure that it correctly validates destination points according to the specified criteria.

Test Name or ID: isTruckAvailable

Test Type: white box

Description: To test the isTruckAvailable function's ability to determine if a truck is available to fit a package based on its weight and volume.

Setup: Create a struct Package variable and a struct DeliveryStatus variable.

Test Function: int isTruckAvailable(const struct Package package, const struct DeliveryStatus status, char truckColor);

Test Scenarios:

TEST ID	Description	Test Data	Expected Res1ult	Actual Result	Pass/Fail
T037	Test when the specified truck has enough weight and volume for the package	package(weight=800, volume=10, destination(row=3, col=4), building(...)),	1	1	Pass

		status(blueTruckStatus(availableWeight=900, availableVolume=20), greenTruckStatus(availableWeight=700, availableVolume=30), yellowTruckStatus(availableWeight=1000, availableVolume=25)), truckColor='B'			
T038	Test when the specified truck does not have enough weight for the package	package(weight=1500, volume=20, destination(row=5, col=8), building(...)), status(blueTruckStatus(availableWeight=900, availableVolume=20), greenTruckStatus(availableWeight=700, availableVolume=30), yellowTruckStatus(availableWeight=1000, availableVolume=25)), truckColor='G'	0	0	Pass
T039	Test when the specified truck does not have enough volume for the package	package(weight=700, volume=40, destination(row=2, col=6), building(...)), status(blueTruckStatus(availableWeight=900, availableVolume=20), greenTruckStatus(availableWeight=700, availableVolume=30), yellowTruckStatus(availableWeight=1000, availableVolume=25)), truckColor='Y'	0	0	Pass
T040	Test when the specified truck color is invalid	package(weight=1000, volume=15, destination(row=7, col=3), building(...)), status(blueTruckStatus(availableWeight=900, availableVolume=20), greenTruckStatus(availableWeight=700, availableVolume=30), yellowTruckStatus(availabl	0	0	Pass

		eWeight=1000, availableVolume=25)), truckColor='X'			

Bugs Found:

I conduct the tests and fill in the "Actual Result" column accordingly

Test Name or ID: isNextToDes

Test Type: white box

Description: Checks if a given point is adjacent to the destination point on a grid.

Setup: Create two struct Point

Test Function: int isNextToDes(struct Point p1, struct Point des)

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T041	Adjacent Column Test	p1 = {0, 0}, p2 = {0, 1}	1	1	Pass
T042	Adjacent Row Test:	p1 = {0, 0}, p2 = {1, 0}	1	1	Pass
T043	Non Adjacent Test	p1 = {0, 0}, p2 = {2, 2}	0	0	Pass
T044	Same Position Test	p1 = {1, 1}, p2 = {1, 1}	1	1	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: decideTruckForDelivery

Test Type: white box

Description: Decides which truck should be used for delivery based on the shortest path to the destination

Setup: create struct Package, DeliveryStatus, Map, Route

Test Function: int decideTruckForDelivery(const struct Package package,

struct DeliveryStatus status, const struct Map *map, const struct Route routes[3]);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T045	All Trucks Available, Different Route Lengths	package = {500, .5 ,8Y}	1	1	Pass
T046	Tie on shortest route with different capacities	package = {500, .5 ,3B} blue truck preload a package	1	1	Pass
T047	One truck is shortest but not available	package = {1200, .5 ,7E} yellow truck is full	0	0	Pass
T048	NoTrucksAvailable	package = {1200, .5 ,7E} all truckStatus is full	-1	-1	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: truckShortestPath

Test Type: white box

Description: Calculates the shortest path from any point in a given truck route to a specified destination

Setup: create struct Route, Map, Point, Building variable

Test Function: struct ShortestRouteInfo truckShortestPath(struct Route truckRoute, const struct Map *map, const struct Point destination, const struct Building building);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T049	destination on the blue route	blueRoute, map, destination{11,11}, building {{11,11, 11,16}, {15,11}, {15,16}}	blueShortestRouteInfo.shortestRoute.numPoints = 0	blueShortestRouteInfo.shortestRoute.numPoints = 0	Pass
T050	destination not the route (shortest divert)	blueRoute, map, destination = {6,1}, building = {{6,1}, {6,0}, {7,1}, {7,0}}	blueShortestRouteInfo.shortestRoute.numPoints = 1	blueShortestRouteInfo.shortestRoute.numPoints = 1	Pass
T051	will not go under building left button when it will reach edge	blueRoute, map, destination = {7,24}, building = {{7,24}, {7,22}, {8,24}, {8,22}}	blueShortestRouteInfo.shortestRoute.numPoints = 14	blueShortestRouteInfo.shortestRoute.numPoints = 14	Pass
T052	start equal destination	blueRoute, map, destination = {0,0}, building = {{7,24}, {7,22}, {8,24}, {8,22}}	blueShortestRouteInfo.shortestRoute.numPoints = 0	blueShortestRouteInfo.shortestRoute.numPoints = 0	Pass
T53	destination on the yellow route	yellowRoute, map, destination{6,4}, building {{6,5, {6,4}, {7,5}, {7,4}}	yellowShortestRouteInfo.shortestRoute.numPoints = 0	yellowShortestRouteInfo.shortestRoute.numPoints = 0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: updateTruckStatus

Test Type: white box

Description: To check if it update the truck status based on the package to be delivered correctly

Setup: truckIndex, struct Package, struct DeliveryStatus

Test Function: void updateTruckStatus(int truckIndex, const struct Package package,

struct DeliveryStatus *status);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T054	Update Blue Truck	truckIndex: 0 package: {weight: 100, volume: 10}	blueTruckStatus.availableWeight= 1100, blueTruckStatus.availableVolume=40	blueTruckStatus.availableWeight= 1100, blueTruckStatus.availableVolume=40	Pass
T055	Update Green Truck	truckIndex: 1 package: {weight: 100, volume: 10}	greenTruckStatus.availableWeight= 1100, greenTruckStatus.availableVolume=40	greenTruckStatus.availableWeight = 1100, greenTruckStatus.availableVolume =40	Pass
T056	Update Yellow Truck	truckIndex: 2 package: {weight: 100, volume: 10}	yellowTruckStatus.availableWeight= 1100, yellowTruckStatus.availableVolume=40	yellowTruckStatus.availableWeight = 1100, yellowTruckStatus.availableVolume =40	Pass
T057	Invalid Truck Index	truckIndex: -1 package: {weight: 100, volume: 10}	all trucks remain same	all trucks remained the same	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: compareTruckCapacity

Test Type: white box

Description: Checking the remaining capacity of two trucks and seeing if the function returns the one with the greater available space

Setup: create two struct TruckStatus variables

Test Function: int compareTruckCapacity(const struct TruckStatus* truck1, const struct TruckStatus* truck2);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail

T058	Two valid truck statuses	t1 = {1000, 40}
		t2 = {700, 30}
			1	1	Pass

T059	Equal truck statuses	t1 = {1000, 40}
		t2 = {1000, 40}
			0	0	Pass

T060	Negative truck statuses	t1 = {-1000, -40}
		t2 = {-800, -30}
			-1	-1	Pass

T061	Blank truck status	t1 = {1000, 40}
		t2 = {}
			1	1	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: printDeliveryRoute

Test Type: white box

Description: To check if the function can show the right instructions based on the information given

Setup: create a struct truckIndex, struct package and struct deliverystatus variable

Test Function: void printDeliveryRoute(int truckIndex, struct ShortestRouteInfo routeInfo);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail

T062	Positive, valid inputs	t = 0 s = { points= {1,1}, {2,2}, {3,3}, numPoints = 3, route symbol = a}	ship on GREEN LINE, divert : 2B, 3C, 4D	ship on GREEN LINE, divert : 2B, 3C, 4D	Pass
T063	No shortest route information provided	t = 0 s = {}	ship on GREEN LINE, no diversion	ship on GREEN LINE, no diversion	Pass
T064	truck index is out of range	t = -1 s = { points= {1,1}, {2,2}, {3,3}, numPoints = 3, route symbol = a}	No trucks available right now. divert : 2B, 3C, 4D	No trucks available right now. divert : 2B, 3C, 4D	Pass
T065	Out of range and no route information	t = -1 s = {}	No trucks available right now. no diversion	No trucks available right now. no diversion	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: shortestIndex

Test Type: white box

Description: Checking the shortest index returned by the function

Setup: create an array of lengths

Test Function: int shorstestIndex(int *lenghtArray);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T066	Positive Lengths Array	A = {1, 2, 3}	0	0	Pass
T067	Negative Length Array	A = {-1, -2, -3}	2	2	Pass
T068	Zero Length Array	A = {0, 0, 0}	0	0	Pass
T069	Lowest number appears twice in array	A = {2, 4, 2}	0	0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: tieOnTheShortest

Test Type: white box

Description: Checking if function can identify ties

Setup: Create an array of lengths

Test Function: int tieOnTheShortest(int *lenghtArray);

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T070	First element is shortest	A = {1, 2, 3}	0	0	Pass
T071	First and third element is shortest	A = {1, 2, 1}	1	1	Pass
T072	All element are equal	A = {0, 0, 0}	1	1	Pass
T073	Two elements are not the same but they are not the smallest number	A = {0, 1, 1}	0	0	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

—----------integration test---

Test Name or ID: Validate Package and Select Truck

Test Type: integration test

Description: Ensure that the system correctly validates a package's weight and volume, then selects the appropriate truck based on availability and capacity.

Setup:

1. Initialize the delivery status with initializeDeliveryStatus().
2. For each package data, validate weight and volume using validatePackageWeight() and validatePackageVolume().
3. If validation passes, use decideTruckForDelivery() to select the appropriate truck.

Test Function: validatePackageWeight, validatePackageVolume, decideTruckForDelivery

Test Scenarios:

1.Package with valid weight and volume, destination accessible, blue truck available with the highest capacity.
2. Package with valid weight and volume, destination accessible, green truck available with the highest capacity.
3. Package with weight exceeding `MAX_WEIGHT`, should fail validation.
4. Package with volume exceeding `MAX_VOLUME`, should fail validation.

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail
T074	Package with valid weight and volume, destination accessible, blue truck available with the highest capacity.	Weight: 100 kg, Volume: 0.5 cubic meters,	Blue truck selected for delivery.	Blue truck selected for delivery.	Pass
T075	Package with valid weight and volume, destination accessible, green truck available with the highest capacity.	Weight: 100 kg, Volume: 0.5 cubic meters,	Green truck selected for delivery.	Green truck selected for delivery.	Pass
T076	Package with weight exceeding MAX_WEIGHT, should fail validation.	Weight: 1500 kg	Validation failure.	Validation failure.	Pass

T077	Package with volume exceeding MAX_VOLUME, should fail validation.	Volume: 10 cubic meters	Validation failure.	Validation failure.	Pass

Bugs Found:

Test Name or ID: Validate Package and Select Truck

Test Type: integration test

Description: Test the system's ability to check truck availability and select the correct route based on the package's destination and truck status.

Setup:

1. Initialize delivery status.
2. create a package
3. Use `isTruckAvailable()` to check the availability of each truck.
4. Use `decideTruckForDelivery()` to determine the best truck for delivery based on the package's requirements and truck availability.

Test Function: `isTruckAvailable(),decideTruckForDelivery()`

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail

T078	Package with a destination requiring blue truck; blue truck available	struct DeliveryStatus dstatus = initializeDeliveryStatus ();
		struct Point destination = {7, 10};
			Blue truck selected for delivery	Blue truck selected for delivery	Pass

		struct Building building = {{7, 12}, {7, 10}, {8, 12}, {8, 10}};
		package = {100, 50, destination, building}
		truckColor = ‘B’
		Blue truck available w/ max capacity			

T079	Package requiring the green route, green truck is not available	struct DeliveryStatus dstatus = initializeDeliveryStatus() - blue truck;
		struct Point destination = {7, 10};
		building = findBuildingBoundaries(map, destination);
		Package = { 100, 50, destination, building }
		truckColor = ‘G’
		Green truck not available
			Blue Truck is selected	Blue Truck is selected	Pass

T080	Package requiring the yellow route, all routes available	struct DeliveryStatus = dstatus = initialize DeliveryStatus() building = findBuildingBoundaries(map, destination) package = {100, 50} truckColor = ‘Y’	Truck with highest capacity should be chosen	Error	Fail

T081	Large volume and heavy package that only one truck can accommodate	destination = { 19, 22 }
		building = findBuildingBoundaries(map, destination)
		Package = {1100, 40, destination, building }
		truckColor = ‘Y’
			Truck with greatest capacity chosen	Yellow Truck Selected	Pass

Bugs Found:

- decideTruckForDelivery() asks for blue truck despite the program asking for yellow truck. whenever truck colour and route changed to blue, decideTruckForDelivery() returns yellow truck.

Test Name or ID: Shortest Path Calculation and Delivery Route

Test Type: integration test

Description: Verify that the system calculates the shortest path correctly and updates the truck status accordingly after a delivery.

Setup:

1. Initialize the map and delivery status.
2. Use truckShortestPath() to calculate the shortest path for each package's destination.
3. Select the appropriate truck with decideTruckForDelivery()
4. After delivery, update the truck status using updateTruckStatus() .

Test Function: truckShortestPath(),decideTruckForDelivery(), updateTruckStatus().

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail

T082	Package with a straightforward route to the destination.	struct Point destination = {7, 10};
		struct Building building = {{7, 12}, {7, 10}, {8, 12}, {8, 10}};
		struct Package package = {100, 0.5, destination, building};
			blue truck is selected and update the available weight and volume	blue truck is selected and update the available weight and volume	Pass

T083	Package requiring a diversion from the standard truck route	struct DeliveryStatus deliveryStat = initializeDeliveryStatus();
		// 12L
		struct Point destination = {11, 11};
		struct Building building = {{11, 16}, {11, 11}, {15, 16}, {15, 11}};
			blue truck is selected and update the available weight and volume	blue truck is selected and update the available weight and volume	Pass

		struct Package package = {100, 0.5, destination, building};			

T084	Package with a destination that requires a complex route avoiding obstacles	struct DeliveryStatus deliveryStat = initializeDeliveryStatus(); // 7F
		struct Point destination = {6, 5};
		struct Building building = {{6, 5}, {6, 4}, {7,5}, {7, 4}};
		struct Package package = {100, 0.5, destination, building};
			blue truck is selected and update the available weight and volume	blue truck is selected and update the available weight and volume	Pass

T085	Package that updates the truck status to near its capacity limits	struct DeliveryStatus deliveryStat = initializeDeliveryStatus();
		// 8k
		struct Point destination = {7, 10};
		struct Building building = {{7, 12}, {7, 10}, {8, 12}, {8, 10}};
		struct Package package = {1499, 0.5, destination, building};
			blue truck is selected and update the available weight and volume	blue truck is selected and update the available weight and volume	Pass

Bugs Found:

Description of each bug found above and how to reproduce it.

Test Name or ID: Comprehensive Delivery Scenario

Test Type: integration test

Description: This test are designed to comprehensively evaluate the functionality and reliability of the delivery system. By ensuring the system passes this test, I can be confident it meets the requirements and is ready for real-world deployment. I demonstrate the result as the pdf required.

Setup:

1. Enter the different input as in the pdf
2. Compare the output

Test Function: truckShortestPath(),decideTruckForDelivery(), updateTruckStatus(),validatePackageWeight, validatePackageVolume

Test Scenarios:

TEST ID	Description	Test Data	Expected Result	Actual Result	Pass/Fail

T086	Each Package Choose Shortest Path	struct Point destination1 = {7, 10};
		struct Building building1 = findBuildingBoundaries(map, destination1);
		struct Package package1 = {100, 0.5, destination, building};
		
		struct Point destination2 = {8, 24};
		struct Building building2 = findBuildingBoundaries(map, destination2);
		struct Package package2 = {100, 0.5,
			Blue truck is selected for package 1.
			Green truck is selected for package 2.
			Yellow truck is selected for package 3.
				Blue truck is selected for package 1.
				Green truck is selected for package 2.
				Yellow truck is selected for package 3.
					Pass

		destination2, building2};
		struct Point destination3 = {6, 4};
		struct Building building3 = findBuildingBoundaries(map, destination3);
		struct Package package3 = {100, 0.5, destination3, building3};			

T087	Same ShortestPath Choose Truck Base on Capacity	// 2B, a destination that all truck can delivery without devision
		struct Point destination = {1, 1};
		struct Building building = findBuildingBoundaries(map, destination);
		struct Package package = {100, 0.5, destination, building};
		deliver it 3 times
			Blue truck is selected for first delivery. Green truck is selected for second delivery. Yellow truck is selected for third delivery	Blue truck is selected for first delivery. Green truck is selected for second delivery. Yellow truck is selected for third delivery	Pass

T088	The Truck With Shortest Path Is Full	// 9Y
		struct Point destination = {8, 24};
		struct Building building = findBuildingBoundaries(map, destination2);
		struct Package package = {1200, 0.5, destination, building};
		deliver it 2 times
			Green truck is selected for first delivery. Blue truck is selected for second delivery.	Green truck is selected for first delivery. Blue truck is selected for second delivery.	Pass

T089	All Trucks Are Full	struct Point destination = {8, 24};
		struct Building building = findBuildingBoundaries(map, destination2);
		struct Package
			none of the truck is selected for the fourth delivery	none of the truck is selected for the fourth delivery	Pass

		package = {1200, 0.5, destination, building}; deliver it 4 times			

Bugs Found:

Description of each bug found above and how to reproduce it.