# Test Description

**Test Name or ID**: TW01

**Test Type**: White Box

**Description**: Checks if the user input for the size of a shipment is within a valid range (0, 0.25, 0.5 and 1), then returns 1 and is invalid if this returns 0

**Setup:** Under the visual studio 2022 environment.

**Test Function**: validSize()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Invalid size option | size = 0.125 | 0 | 1 | Pass |
| Invalid size option | size = 0.75 | 0 | 1 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW02

**Test Type**: White Box

**Description**: Calculate total percentage of weight and space to decide which truck to load if they are the same distance from the delivery

**Setup:** Pass in a truck structure to calculate and return a combined total percentage of the weight and space

**Test Function**: CapPercentage()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Nominal input test | t.curSpace = 20 t.curWeight = 500 | 1.055 | 1.0556 | Fail |
| Input next margin | t.curSpace = 0 t.curWeight = 0 | 0 | 0 | Pass |
| Input next margin | t.curSpace = 36 t.curWeight = 1000 | 2 | 2 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW04

**Test Type**: White Box

**Description**: This test aims to verify if the “isTruckFull” function correctly determines if a truck can accommodate a shipment based on weight and space constraints.

**Setup:** Prepare a truck object and a shipment object with varying weights and dimensions.

**Test Function**: int isTruckFull(struct Truck truck, struct Shipment box);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test with an empty truck with a half full box | truck.curWeight = 0; truck.curSpace = 0; box.weight = 500; box.size = 18; | 1 | 1 | Pass |
| Test with an empty truck with a full box | truck.curWeight = 0; truck.curSpace = 0; box.weight = 1000; box.size = 36; | 1 | 1 | Pass |
| Test with an empty truck with an overweight box | truck.curWeight = 0; truck.curSpace = 0; box.weight = 2000; box.size = 18; | 0 | 0 | Pass |
| Test with an empty truck with an oversize box | truck.curWeight = 0; truck.curSpace = 0; box.weight = 500; box.size = 50; | 0 | 0 | Pass |
| Test with a half full truck with a small box | truck.curWeight = 500; truck.curSpace = 18; box.weight = 100; box.size = 10; | 1 | 1 | Pass |
| Test with a half full truck with a half full box | truck.curWeight = 500; truck.curSpace = 18; box.weight = 500; box.size = 18; | 1 | 1 | Pass |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW05

**Test Type**: White Box

**Description**: The function "calcShortPathDist" calculates the total distance of a given route and needs to be tested for accuracy.

**Setup:** Prepare test data with various route scenarios (e.g., empty route, straight line route, closed loop route) and set up a programming environment with access to the code. Use assertions to compare the actual total distance calculated by the function with the expected results.

**Test Function**: double calcShortPathDist (struct Route shortP);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test numPoints with the same points in shortP | shortP.numPoints = 4; shortP.points[0]={0,0}; shortP.points[1]={1,0}; shortP.points[2]={2,0}; shortP.points[3]={3,0}; | 3.0 | 3.0 | Pass |
| Test numPoints less than the points in shortP | shortP.numPoints = 2; shortP.points[0]={0,0}; shortP.points[1]={1,0}; shortP.points[2]={2,0}; shortP.points[3]={3,0}; | 1.0 | 1.0 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW06

**Test Type**: White Box

**Description**: The function "convLetter" takes a character representing a letter (uppercase) and converts it into its corresponding numeric value (0 to 25), representing its position in the alphabet.

**Setup:** Prepare test data with different uppercase letters (A to Z) as input to the function. Set up a programming environment with access to the code and use assertions to compare the actual numeric values returned by the function with the expected results (0 to 25 for valid inputs).

**Test Function**: int convLetter(char letterCord);

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test letterCord as A | letterCord = 'A'; | 0 | 0 | Pass |
| Test letterCord as Z | letterCord = 'Z'; | 25 | 25 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW08

**Test Type**: White Box

**Description**: The function "determineDivergencePath" aims to identify the points where a truck diverges from its main route to deliver a shipment. It iterates through the truck's s\_Path and compares each point with the points in the main route (truck->path). If a point in s\_Path is not present in the main route, it is considered a divergence point and added to the truck's divergence\_path.

**Setup:** Prepare test data with different scenarios of truck routes and divergence points. Set up a programming environment with access to the code. Use assertions to compare the actual divergence\_path generated by the function with the expected results.

**Test Function**: determineDivergencePath()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test for no divergence points | truck->path: A linear path from (0,0) to (0,4).  truck->s\_Path: The same linear path as truck->path. | truck->divergence\_path should be empty. | truck->divergence\_path : empty. | PASS |
| Test for one divergence point | truck->path: A linear path from (0,0) to (0,4).  truck->s\_Path: A diverging path from (0,0) to (0,4) with a divergence point at (0,2). | truck->divergence\_path should contain the divergence point (0,2). | truck->divergence\_path contains the divergence point (0,2). | PASS |
| Test for multiple divergence points | truck->path: A linear path from (0,0) to (0,4).  truck->s\_Path: A diverging path from (0,0) to (0,4) with divergence points at (0,1) and (0,3). | truck->divergence\_path should contain the divergence points (0,1) and (0,3). | truck->divergence\_path contains the divergence points (0,1) and (0,3). | PASS |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW09

**Test Type**: White Box

**Description**: The function "setValidDefaultMinTruck" aims to find the default minimum truck index based on the list of trucks. The function checks each truck's s\_Path (shortest path) and returns the index of the first valid truck that has a non-empty path. If no valid truck is found, the function returns -1.

**Setup:** Prepare test data with different scenarios of truck routes, including trucks with valid and invalid paths. Set up a programming environment with access to the code. Use assertions to compare the actual minimum truck index returned by the function with the expected results.

**Test Function**: "setValidDefaultMinTruck ()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test for all invalid trucks | truckList[0].s\_Path.numPoints = 0  truckList[1].s\_Path.numPoints = 0  truckList[2].s\_Path.numPoints = 0 | -1 (no valid truck found) | -1 | PASS |
| Test for one valid truck | truckList[0].s\_Path.numPoints = 0 (invalid)  truckList[1].s\_Path.numPoints = 6 (valid)  truckList[2].s\_Path.numPoints = 0 (invalid) | 1 (index of the valid truck) | 1 | PASS |
| Test for multiple valid trucks | truckList[0].s\_Path.numPoints = 0 (invalid)  truckList[1].s\_Path.numPoints = 8 (valid)  truckList[2].s\_Path.numPoints = 12 (valid) | 1 (index of the first valid truck with a non-empty path) | 1 | PASS |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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

**Test Name or ID**: TW10

**Test Type**: White Box

**Description**: The function "findBestTruckForShipment" aims to determine the best truck to carry a shipment to its destination. It calculates the shortest distance between the trucks and the destination and considers factors such as truck capacity and current load to make the best selection.

**Setup:** Prepare test data with different scenarios of truck routes, package destinations, and truck loads. Set up a programming environment with access to the code. Use assertions to compare the actual selected truck index returned by the function with the expected results.

**Test Function**: findBestTruckForShipment ()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test for all invalid trucks | truckList[0].s\_Path.numPoints = 0  truckList[1].s\_Path.numPoints = 0  truckList[2].s\_Path.numPoints = 0  package.destination = (5,5) (any valid destination)  package.weight = 100 (any valid weight)  package.size = 0.5 (any valid size) | -1 (no valid truck found) | -1 | PASS |
| Test for one valid truck | truckList[0].s\_Path.numPoints = 0 (invalid)  truckList[1].s\_Path.numPoints = 6 (valid)  truckList[2].s\_Path.numPoints = 0 (invalid)  package.destination = (3,3) (any valid destination)  package.weight = 500 (any valid weight)  package.size = 0.25 (any valid size) | 1 (index of the valid truck) | 1 | PASS |
| Test for multiple valid trucks with different capacities | truckList[0].s\_Path.numPoints = 8 (valid)  truckList[1].s\_Path.numPoints = 10 (valid)  truckList[2].s\_Path.numPoints = 12 (valid)  package.destination = (8,8) (any valid destination)  package.weight = 800 (any valid weight)  package.size = 0.5 (any valid size) | 0 (index of the truck with the shortest distance and sufficient capacity) |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**:

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