# Test Description Function 1 – struct Shipment getUserInput()

**Test Name or ID**:

**Test Type**: White Box

**Description**: Gets input about the shipment details validates the input and displays error prompts for incorrect input

**Setup:** N/A

**Test Function**: struct Shipment getUserInput()

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with valid weight, box size, and destination. | 500.0 10.0 4D | A valid struct Shipment object with weight = 500.0, volume = 10.0, destination row = 4, destination col = 'D'. | Pass |
| 2 | Test with input to stop the shipment entry | 0 0 x | A valid struct Shipment object with weight = 0, volume = 0, destination row and col are ignored as the input is 'x'. | Pass |
| 3 | Test with weight outside the valid range. | 1001.0 20.0 3C | Invalid weight (must be 1-1000 Kg.) | Fail |
| 4 | Test with box size outside the valid range. | 800.0 0.5 25Y | Invalid size | Fail |
| 5 | Test with an invalid destination format. | 700.0 30.0 0E | Invalid destination | Fail |
| 6 | Test with leading zeros in weight and box size. | 012.0 005.0 10A | A valid struct Shipment object with weight = 12.0, volume = 5.0, destination row = 10, destination col = 'A'. | Pass |
| 7 | Test with extra spaces between inputs. | 200.0 25.0 5B | A valid struct Shipment object with weight = 200.0, volume = 25.0, destination row = 5, destination col = 'B'. | Pass |

**Bugs Found**:

N/A

# Test Description Function 2 – int checkWeight(double weight)

**Test Name or ID**:

**Test Type**: White Box

**Description**: Checks if the weight input is valid or not

**Setup:** N/A

**Test Function**: int checkWeight(double weight)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with a weight within the valid range. | 500.0 | 1 | Pass |
| 2 | Test with the minimum allowed weight (lower bound test). | 1.0 | 1 | Pass |
| 3 | Test with the maximum allowed weight (upper bound test). | 1000.0 | 1 | Pass |
| 4 | Test with a weight of zero. | 0.0 | 0 | Pass |
| 5 | Test with a negative weight. | -100.0 | 0 | Pass |
| 6 | Test with a weight greater than the maximum allowed weight. | 1500 | 0 | Pass |
| 7 |  |  |  |  |

**Bugs Found**:

N/A

# Test Description Function 3 – int checkBox(double boxSize)

**Test Name or ID**:

**Test Type**: White Box

**Description**: Checks if the box volume input is valid or not

**Setup:** N/A

**Test Function**: int checkBox(double boxSize)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with a valid box size from the available options. | 0.5 | 1 | Pass |
| 2 | Test with the smallest allowed box size (lower bound test). | 0.25 | 1 | Pass |
| 3 | Test with the largest allowed box size (upper bound test). | 1.0 | 1 | Pass |
| 4 | Test with an invalid box size that is not in the available options. | 0.75 | 0 | Pass |
| 5 | Test with a box size that is on the boundary between available options (boundary test). | 0.25 | 1 | Pass |
| 6 | Test with a negative box size. | -0.5 | 0 | Pass |
| 7 | Test with a fractional box size. | 0.75 | 0 | Pass |

**Bugs Found**:

N/A

# Test Description Function 4 – int checkDestination(int row, char col)

**Test Name or ID**:

**Test Type**: White Box

**Description**: Checks if the destination input is valid or not

**Setup:** N/A

**Test Function**: int checkDestination(int row, char col)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with a valid destination that exists in the map. | row = 3, col = 'C' | 1 | Pass |
| 2 | Test with the minimum allowed row (1) and col ('A') values (lower bound test). | row = 1, col = 'A' | 1 | Pass |
| 3 | Test with the maximum allowed row (maximum row number in the map) and col (maximum column letter in the map) values (upper bound test). | row = 25, col = 'Y' | 1 | Pass |
| 4 | Test with an invalid destination that does not exist in the map. | row = 26, col = 'Z' | 0 | Pass |
| 5 | Test with a row that is on the boundary of the map (boundary test). | row = 13, col = 'Y' | 1 | Pass |
| 6 | Test with a column letter that is on the boundary of the map (boundary test). | row = 0, col = 'C' | 1 | Pass |
| 7 | Test with an invalid row value that is less than the minimum allowed value (1). | row = 4, col = 'Z' | 0 | Pass |

**Bugs Found**:

N/A

# Test Description Function 5 - struct DivertedRoute getDivertedRoute(struct Map\* map, struct Truck truck,const struct Point destination)

**Test Name or ID**:

**Test Type**: White Box

**Description**: Gets the shortest possible path when a truck's route needs to be diverted for a shipment **Setup:** N/A

**Test Function**: struct DivertedRoute getDivertedRoute(struct Map\* map, struct Truck truck,const struct Point destination)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with a valid truck route and destination. Verify that the diverted route is calculated correctly. | truck.truckRoute = { {1, 1}, {2, 2}, {3, 3} }, truck.numPoints = 3, destination = {4, 4} | The divertedRoute should be the shortest path from the closest point in the truck route to the destination. | Pass |
| 2 | Test with an empty truck route and a valid destination. | truck.truckRoute = {}, truck.numPoints = 0, destination = {4, 4} | The divertedRoute should be empty since there are no points in the truck route to calculate the diverted route. | Pass |
| 3 | Test with a valid truck route and an empty destination. | truck.truckRoute = { {1, 1}, {2, 2}, {3, 3} }, truck.numPoints = 3, destination = {} | The divertedRoute should be empty since there is no destination point to calculate the diverted route. | Pass |
| 4 | Test with an invalid or null map pointer. | map = NULL, truck.truckRoute = { {1, 1}, {2, 2}, {3, 3} }, truck.numPoints = 3, destination = {4, 4} | The function should handle the null map pointer gracefully and return an empty divertedRoute. | Pass |
| 5 | Test with a truck route containing the maximum allowed number of points (upper bound test). | truck.truckRoute = { /\* Maximum allowed points in the truck route \*/ }, truck.numPoints = MAX\_ROUTE, destination = {4, 4} | The divertedRoute should be the shortest path from the closest point in the truck route to the destination. | Pass |
| 6 | Test with a truck route containing a large number of points. | truck.truckRoute = { {1, 1}, {2, 2}, {3, 3}, ... }, truck.numPoints = large\_value, destination = {4, 4} | The divertedRoute should be the shortest path from the closest point in the truck route to the destination. | Pass |
| 7 | Test with a destination point that has large coordinates. | truck.truckRoute = { {1, 1}, {2, 2}, {3, 3} }, truck.numPoints = 3, destination = {large\_value, large\_value} | The divertedRoute should be the shortest path from the closest point in the truck route to the destination. | Pass |

**Bugs Found**:

N/A

# Test Description Function 6 - int selectTruck(struct Map \*map, struct Truck truckArr[], int numOfTrucks, struct Shipment shipment)

**Test Name or ID**:

**Test Type**: White Box

**Description**: Finds the best truck for shipment after comparing, routes, shortest diverted paths, load already on the trucks and percentages of the limiting factors. If the shipment cannot be delivered it returns a negative value

**Setup:** N/A

**Test Function**: int selectTruck(struct Map \*map, struct Truck truckArr[], int numOfTrucks, struct Shipment shipment)

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Description | Test Data | Expected Result | Pass/Fail |
| 1 | Test with an invalid destination that does not exist in the map. | truckArr[0] with a route that does not include the destination point of the shipment.  shipment = { weight = 20, volume = 0.5, destination = {28, 'x'} } | The function should return -1 since there is no valid truck available to carry the shipment due to an invalid destination. | Pass |
| 2 | Test with an invalid box size that is not in the available options. | truckArr[1] with a route that includes the destination point of the shipment.  shipment = { weight = 20, volume = 2.0, destination = {12, 'L'} } | The function should return -1 since there is no valid truck available to carry the shipment due to an invalid box size. | Pass |
| 3 | Test with an invalid weight that exceeds the maximum allowed weight. | truckArr[2] with a route that includes the destination point of the shipment.  shipment = { weight = 1005, volume = 0.5, destination = {12, 'L'} } | The function should return -1 since there is no valid truck available to carry the shipment due to an invalid weight. | Pass |
| 4 | Test with a shipment destination that exists on the truck's current route without the need for diversion. | truckArr[3] with a route that includes the destination point of the shipment.  shipment = { weight = 20, volume = 0.5, destination = {12, 'L'} } | The function should return 3 since truckArr[3] is already on-route and has enough capacity to carry the shipment without diversion. | Pass |
| 5 | Test with a shipment destination that requires a diverted route for one of the trucks. | truckArr[4] with a route that does not include the destination point of the shipment.  shipment = { weight = 200, volume = 1.0, destination = {8, 'Y'} } | The function should return 4 since truckArr[4] has the shortest diverted route and enough capacity to carry the shipment. | Pass |
| 6 | Test with a shipment destination that requires a diverted route for another truck. | truckArr[5] with a route that includes the destination point of the shipment.  shipment = { weight = 500, volume = 1.0, destination = {8, 'Y'} } | The function should return 5 since truckArr[5] has the shortest diverted route and enough capacity to carry the shipment. | Pass |

**Bugs Found**:

N/A