# Test Description

**Test Name or ID**: ValidatePackageWeight

**Test Type**: Black box

**Description**: This test verifies that the validatePackageWeight function correctly identifies valid and invalid package weights. The function should return 1 for valid weights (1 to 2500 kg) and 0 for any other weight.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: validatePackageWeight

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Validate maximum package weight (2500 kg) | 2500 | 1 | 1 | Pass |
| Validate minimum package weight (1 kg) | 1 | 1 | 1 | Pass |
| Validate over maximum weight (3000 kg) | 3000 | 0 | 0 | Pass |
| Validate zero weight | 0 | 0 | 0 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**: Initial implementation did not check for minimum weight. The issue was fixed by adding a check for weights greater than or equal to 1 kg.

The original Code:   
int validatePackageWeight(double num) {  
return (num <= MAX\_CARGO\_WEIGHT) ? 1 : 0;  
}

Fixed Code:   
int validatePackageWeight(double num) { //fixed code after black box testing

return (num >= 1 && num <= MAX\_CARGO\_WEIGHT) ? 1 : 0;

}

Test Description

**Test Name or ID**: ValidatePackageBox

**Test Type**: Black box

**Description**: This test validates that the validatePackageBox function correctly identifies valid and invalid package sizes. The function should return 1 for valid sizes (1, 3, 5) and 0 for any other size.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: validatePackageBox

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Valid package size 1 | 1 | 1 | 1 | Pass |
| Valid package size 3 | 3 | 1 | 1 | Pass |
| Valid package size 5 | 5 | 1 | 1 | Pass |
| Invalid package size 2 | 2 | 0 | 0 | Pass |
| Invalid package size 4 | 4 | 0 | 0 | Pass |
| Invalid package size 6 | 6 | 0 | 0 | Pass |
|  | 0 |  |  |  |

null

**Bugs Found**: No bugs found in this function.

Test Description

**Test Name or ID**: ReturnInt

**Test Type**: Black box

**Description**: This test verifies that the returnInt function accurately converts alphabetic characters to their corresponding integer values based on their position in the alphabet.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: returnInt

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Convert 'A' to 0 | 'A' | 0 | 0 | Pass |
| Convert 'B' to 1 | 'B' | 1 | 1 | Pass |
| Invalid character 'a' | 'a' | -1 | -1 | Pass |
| Invalid character '1' | '1' | -1 | -1 | Pass |
| Invalid character '@' | '@' | -1 | -1 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**: No bugs found in this function.

Test Description

**Test Name or ID**: IsDestinationValid

**Test Type**: Black box

**Description**: This test verifies that the isDestinationValid function correctly identifies valid and invalid destination coordinates on the map.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: isDestinationValid

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Valid destination (1, 4) | (1, 4) | 1 | 1 | Pass |
| Valid destination (7, 18) | (7, 18) | 1 | 1 | Pass |
| Invalid destination with negative row | (-1, 0) | 0 | 0 | Pass |
| Invalid destination with negative column | (0, -1) | 0 | 0 | Pass |
| Invalid destination out of bounds | (30, 30) | 0 | 0 | Pass |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**: No bugs found in this function.

Test Description

**Test Name or ID**: CheckDestInput

**Test Type**: Black box

**Description**: This test verifies that the checkDestInput function correctly parses the destination input string to extract the numeric and character components.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: checkDestInput

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Parse "12A" correctly | "12A" | num: 12, character: 'A' | num: 12, character: 'A' | Pass |
| Parse "5B" correctly | "5B" | num: 5, character: 'B' | num: 5, character: 'B' | Pass |
| Parse "X" correctly | "X" | num: 0, character: 'X' | num: 0, character: 'X' | Pass |
| Parse "24Z" correctly | "24Z" | num: 24, character: 'Z' | num: 24, character: 'Z' | Pass |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**: No bugs found in this function.

Test Description

**Test Name or ID**: ReturnSymb

**Test Type**: Black box

**Description**: This test checks the returnSymb function, which returns the symbol at a specific location on the map. The function should correctly convert numeric values in the map to their corresponding symbols.

**Setup:** 1. Ensure the development environment is set up with all necessary files (Truck.c, Truck.h, mapping.c, mapping.h).

2. Include the test file BlackBoxTest.cpp in the Visual Studio project.

3. Configure the project for unit testing using CppUnitTestFramework.

**Test Function**: returnSymb

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Convert 1 to 'X' at (1,1) | (1,1) | 'X' | 'X' | Pass |
| Convert 0 to ' ' at (0,0) | (0,0) | ' ' | ' ' | Pass |
| Convert 1 to 'X' at (1,4) | (1,4) | 'X' | 'X' | Pass |
| Convert 0 to ' ' at (3,0) | (3,0) | ' ' | ' ' | Pass |
| Convert 0 to ' ' at (24,24) | (24,24) | ' ' | ' ' | Pass |
|  |  |  |  |  |
|  |  |  |  |  |

**Bugs Found**: Initial implementation did not correctly map the numeric values to symbols. The issue was fixed by adding proper symbol mapping logic.

The original code:   
char returnSymb(const struct Map\* routeMap, int destRow, int destCol) {  
 size\_t maxRow = MAP\_ROWS;  
 size\_t maxCol = MAP\_COLS;  
 char destinationSymbol = routeMap->squares[destRow][destCol];  
 return destinationSymbol;

}

The fixed code:

char returnSymb(const struct Map\* routeMap, int destRow, int destCol) { //fixed code after black box testing

size\_t maxRow = MAP\_ROWS;

size\_t maxCol = MAP\_COLS;

char symbols[] = { ' ', 'X' };

char destinationSymbol = routeMap->squares[destRow][destCol];

return symbols[destinationSymbol];

}