# BlackBox Test Description (MS03 Submission)

**Test Name or ID**: test\_populateMap(), test\_printMap()

**Test Type**: BlackBox

**Description**:   
The testing codes mentioned below can be found in blackbox\_tester.c. They are used to create test cases for populating and printing the map correctly when compiled. These testing codes adhere to blackbox testing principles, where the tester user is unaware of the code's functionality and specifications. But only the description of Milestone specification. In Group 4, we are testing the basic functionality of the program, ensuring that it correctly prints and populates the map as described in the Milestone description.

**Setup:** Creating testing file and running with gcc in visual studio code. All blackbox testing codes will be testing on the mapping.c since we have not completed the implementation of functions in new h file created in MS02.

**Test Function**:

void test\_populateMap()

{

    struct Map map = populateMap();

*// Verify the number of rows and columns*

    if (map.numRows == 25 && map.numCols == 25)

        printf("populateMap test case 1 passed!\n");

    else

        printf("populateMap test case 1 failed!\n");

}

For test\_populateMap function , numRows and numCols will be Test Data that we change for each test.

void test\_printMap() {

    struct Map map = populateMap();

*// Test Case 1*

    printf("printMap test case 1:\n");

    printMap(&map, 1, 1); *// Start row with 1 and column with 'A'*

*// Check if the row starts with 1*

    if (map.squares[3][0] == '1') *// Adjust the index to [0][0]*

        printf("Row starts with 1.\n");

    else

        printf("Row does not start with 1.\n");

*// Check if the column starts with 'A'*

    if (map.squares[0][3] == 'A') *// Adjust the index to [0][1]*

        printf("Column starts with 'A'.\n");

    else

        printf("Column does not start with 'A'.\n");

}

For the test\_printMap() function, we can assign different values to map.squares[row][column] to test specific conditions in the if statements. The aim is to achieve the desired output of "Row starts with 1" and "Column starts with 'A'", indicating that the test case has passed.  
map.square[row].[column]

A screenshot of a computer

Description automatically generated  
Considering the desired output, we observe that the numbers start after the alphabet and the dash character '-'. The alphabet starts after the numbers, followed by the vertical bar '|'. In black box testing, where we assume indexing starts from [0], the first numbers and letters will be in the third index. Therefore, map.squares[3][0] and map.squares[0][3] would be correct choices to test the conditions.

By assigning appropriate values to these indices, we can validate if the row starts with 1 and the column starts with 'A' as expected.  
Moreover we can modify the function to check the end position as well with map.squares[24][0] and map.squares[0][24]

A computer screen shot of a code

Description automatically generated

**Test Scenarios:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test\_populateMap() | Minimum int = 0,0 | populateMap test case 1 failed! | populateMap test case 1 failed! | PASS |
| Test\_populateMap() | Maximum int = 25,25 City represented as 25 x 25 grid | populateMap test case 1 passed! | populateMap test case 1 passed! | PASS |
| Test\_printMap() | Considering the fact that index position start with 0 map.square[3].[0] to see if third row in first column matches as 1. And for Second if statement map.square[0].[3] to see if third column and first row matches as ‘A’. | printMap test case 1:  Row starts with 1.  Column starts with 'A'. | printMap test case 1:  ABCDEFGHIJKLMNOPQRSTUVWXY  -------------------------  1| XX  2| XX XX XX XX XX  3| XX XX XX X XX XX XX  4|  5|  6|  7|XX XX XX X XX  8|XX XX X XXX X X XXX  9| X XXX X X XXX  10|  11|  12|X XXX XXXXXX XXX XXX  13|X XXXX XXXXXX XXX XXX  14|X XXX XXXX XXXXXX XXX XXX  15|X XXX XXXXXX XXX XXX  16|X XXX XXXXXX XXX XXX  17| XXX  18| XXX  19| XXX XXXXXXXXXXXX  20|  21|  22| XXXXXXX XX XX XXXX  23| XXXXXXX XX XX XXXX XXXX  24| XXXXXXX XX XX XXXX  25| XXXXXXX XX XX  Row does not start with 1.  Column does not start with 'A'. | FAIL |
| Test\_printMap() | Testing end position as map.squares[24][0] and map.squares[0][24] | "Row ends with 25.” "Column ends with 'Y'” | printMap test case 1:  ABCDEFGHIJKLMNOPQRSTUVWXY  -------------------------  1| XX  2| XX XX XX XX XX  3| XX XX XX X XX XX XX  4|  5|  6|  7|XX XX XX X XX  8|XX XX X XXX X X XXX  9| X XXX X X XXX  10|  11|  12|X XXX XXXXXX XXX XXX  13|X XXXX XXXXXX XXX XXX  14|X XXX XXXX XXXXXX XXX XXX  15|X XXX XXXXXX XXX XXX  16|X XXX XXXXXX XXX XXX  17| XXX  18| XXX  19| XXX XXXXXXXXXXXX  20|  21|  22| XXXXXXX XX XX XXXX  23| XXXXXXX XX XX XXXX XXXX  24| XXXXXXX XX XX XXXX  25| XXXXXXX XX XX  Row does not end with 25.  Column does not end with 'Y'. | FAIL |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# BlackBox Test Debugging (MS04 Submission)

**Bugs Found**:

We have found that the testing function Test\_printMap() does not work as expected. Even when we input correct values for positions, the test case fails. Therefore, we need to evaluate it further in order to make the test case pass. After investigation, we realized that there is a problem within the testing function. As a result, we have extended it as shown below.

Please note that the correct implementation for blackbox testing is part of MS04, not MS03. However, the group has decided to continue development this week and push it to the repository later, once the whitebox testing for MS04 is complete.

*// Test 2: printMap()*

*// Helper function to check if the map coordinates are correct*

int checkMapCoordinates(*const* struct Map\* map) {

    int rowMax = map->numRows;

    int colMax = map->numCols;

    for (int r = 0; r < rowMax; r++) {

        if (r + 1 < 1 || r + 1 > 25) {

            printf("Row coordinate out of range: %d\n", r + 1);

            return 0;

        }

        for (int c = 0; c < colMax; c++) {

            if (c < 0 || c >= 25) {

                printf("Column coordinate out of range: %d\n", c);

                return 0;

            }

        }

    }

    return 1;

}

*// Helper function to print the map*

void testPrintMap(*const* struct Map\* map) {

    char sym*[]* = " XB?G?.?Y?-?\*?+?P";

    int rowMax = map->numRows;

    printf("%4s", " ");

    for (int c = 0; c < map->numCols; c++) {

        printf("%c", 'A' + c);

    }

    printf("\n");

    printf("%4s", " ");

    for (int c = 0; c < map->numCols; c++) {

        printf("-");

    }

    printf("\n");

    for (int r = 1; r <= rowMax; r++) {

        printf("%3d|", r);

        for (int c = 0; c < map->numCols; c++) {

            printf("%c", sym[map->squares[r - 1][c]]);

        }

        printf("\n");

    }

}

*// Test printMap function*

void test\_printMap() {

*// Test case 1: base1 = 1, alphaCols = 1*

    struct Map map1 = populateMap();

    printf("\*\*\* PrintMap Test case 1:\n");

    printMap(&map1, 1, 1);

    printf("\n");

    if (checkMapCoordinates(&map1))

        printf(": printMap Test case 1 passed!\n\n");

    else

        printf(": printMap Test case 1 failed!\n\n");

*// Test case 2: base1 = 10, alphaCols = 0*

    struct Map map2 = populateMap();

    printf("\*\*\* PrintMap Test case 2:\n");

    printMap(&map2, 10, 0);

    printf("\n");

    if (!checkMapCoordinates(&map2))

        printf(": printMap Test case 2 passed!\n\n");

    else

        printf(": printMap Test case 2 failed!\n\n");

*// Test case 3: base1 = 20, alphaCols = 1*

    struct Map map3 = populateMap();

    printf("\*\*\* PrintMap Test case 3:\n");

    printMap(&map3, 20, 1);

    printf("\n");

    if (!checkMapCoordinates(&map3))

        printf(": printMap Test case 3 passed!\n\n");

    else

        printf(": printMap Test case 3 failed!\n\n");

}

Testing output for further development

populateMap test passed!

\*\*\* PrintMap Test case 1:

    ABCDEFGHIJKLMNOPQRSTUVWXY

    -------------------------

  1|    XX

  2| XX XX XX       XX  XX

  3| XX XX XX X XX  XX  XX

  4|

  5|

  6|

  7|XX  XX XX      X XX

  8|XX  XX X  XXX  X  X   XXX

  9|       X  XXX  X  X   XXX

 10|

 11|

 12|X XXX      XXXXXX XXX XXX

 13|X     XXXX XXXXXX XXX XXX

 14|X XXX XXXX XXXXXX XXX XXX

 15|X XXX      XXXXXX XXX XXX

 16|X     XXX  XXXXXX XXX XXX

 17|      XXX

 18|      XXX

 19|      XXX    XXXXXXXXXXXX

 20|

 21|

 22| XXXXXXX  XX XX XXXX

 23| XXXXXXX  XX XX XXXX XXXX

 24| XXXXXXX  XX XX      XXXX

 25| XXXXXXX  XX XX

: printMap Test case 1 passed!

\*\*\* PrintMap Test case 2:

    0123456789012345678901234

    -------------------------

 10|    XX

 11| XX XX XX       XX  XX

 12| XX XX XX X XX  XX  XX

 13|

 14|

 15|

 16|XX  XX XX      X XX

 17|XX  XX X  XXX  X  X   XXX

 18|       X  XXX  X  X   XXX

 19|

 20|

 21|X XXX      XXXXXX XXX XXX

 22|X     XXXX XXXXXX XXX XXX

 23|X XXX XXXX XXXXXX XXX XXX

 24|X XXX      XXXXXX XXX XXX

 25|X     XXX  XXXXXX XXX XXX

 26|      XXX

 27|      XXX

 28|      XXX    XXXXXXXXXXXX

 29|

 30|

 31| XXXXXXX  XX XX XXXX

 32| XXXXXXX  XX XX XXXX XXXX

 33| XXXXXXX  XX XX      XXXX

 34| XXXXXXX  XX XX

: printMap Test case 2 failed!

\*\*\* PrintMap Test case 3:

    ABCDEFGHIJKLMNOPQRSTUVWXY

    -------------------------

 20|    XX

 21| XX XX XX       XX  XX

 22| XX XX XX X XX  XX  XX

 23|

 24|

 25|

 26|XX  XX XX      X XX

 27|XX  XX X  XXX  X  X   XXX

 28|       X  XXX  X  X   XXX

 29|

 30|

 31|X XXX      XXXXXX XXX XXX

 32|X     XXXX XXXXXX XXX XXX

 33|X XXX XXXX XXXXXX XXX XXX

 34|X XXX      XXXXXX XXX XXX

 35|X     XXX  XXXXXX XXX XXX

 36|      XXX

 37|      XXX

 38|      XXX    XXXXXXXXXXXX

 39|

 40|

 41| XXXXXXX  XX XX XXXX

 42| XXXXXXX  XX XX XXXX XXXX

 43| XXXXXXX  XX XX      XXXX

 44| XXXXXXX  XX XX

: printMap Test case 3 failed!

# Black Box Test Debugging Continue(MS04 Submission)

**Test Name or ID**: test\_populateMap(), test\_printMap()

**Test Type**: BlackBox

As stated above, black box testing has been debugged and implemented with new code and matrix ticket DEF1 and DEF2 has been resolved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| Test\_printMap() | printMap(&map1, 1, 1); | printMap Test case 1 passed! | printMap Test case 1 passed! | PASS |
| Test\_printMap() | printMap(&map2, 10, 0); | printMap Test case 2 failed! | printMap Test case 2 failed! | PASS |
| Test\_printMap() | printMap(&map3, 20, 1); | printMap Test case 3 failed! | printMap Test case 3 failed! | PASS |

Output of tests in console including the map example, is same as above where we demonstrated the further development last week. Now black\_box\_tester.c has been updated and pushed. Testing compile was done with gcc with mapping.c.

# White Box Test Implementation (MS04 Submission)

## whitebox\_mapping\_tester.c

White Box testing below is added to the source file folder C file named as whitebox\_mapping\_tester.c

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| test\_addRoute() | struct Route route = { 'X', 1, 1, 5, 'E' }; | printMap Test case 1 passed! | addRoute test case1 failed! | Fail |
|  |  |  |  |  |
|  |  |  |  |  |

As of MS04 our team has been developing different types of white box test cases, testing internal functions like adding routes to the truck paths. Specifically this function is testing on mapping.c that was given at the beginning however the test did not pass. Ticket has been created on matrix, and updated and tick has been assigned to one of team member for debugging.

## whitebox\_finder\_tester.c

This testing file contains various test cases that are testing towards the function implementation that we did on function specification that was due on MS03.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Test Data | Expected Result | Actual Result | Pass/Fail |
| test\_hasDestination() | shipment.destination.row = 5;      shipment.destination.col = 5; | test\_hasDestination test passed! | test\_hasDestination test failed! | Fail |
| test\_isTruckOverloaded() | struct Truck truck;  *// Initialize the truck as required for testing*      struct Shipment ship;  *// Initialize the shipment as required for testing*  *// Test the function*      int result = isTruckOverloaded(truck, ship); | test\_isTruckOverloaded test passed! | test\_isTruckOverloaded test passed! | Pass |
| test\_isBoxSizeExceeded | float boxSize = 5.0; | Box size is valid. | Box size is valid. | Pass |
| test\_validCargo() | float boxSize = 0.5; | test\_validCargo test passed! | test\_validCargo test passed! | Pass |