**[Win23-SFT221-NFF-4](https://activity-1-11.atlassian.net/browse/WSN4" \t "_self)**

CREATED BY- **YAKSHIT [GROUP NUMBER-4 ID(175307214)]**

**F1 BLACKBOX TESTCODE-**

#include <stdio.h>

#include <limits.h>

#include <float.h>

int isValidPackage(int weight, double size);

int main() {

// Test case 1: minimum valid package weight and size

int weight1 = 1;

double size1 = 0.001;

int result1 = isValidPackage(weight1, size1);

printf("Test case 1 - Expected output: 1, Actual output: %d\n", result1);

// Test case 2: maximum valid package weight and size

int weight2 = 100;

double size2 = 1.0;

int result2 = isValidPackage(weight2, size2);

printf("Test case 2 - Expected output: 1, Actual output: %d\n", result2);

// Test case 3: invalid package weight

int weight3 = 0;

double size3 = 0.01;

int result3 = isValidPackage(weight3, size3);

printf("Test case 3 - Expected output: 0, Actual output: %d\n", result3);

// Test case 4: invalid package size

int weight4 = 10;

double size4 = -0.1;

int result4 = isValidPackage(weight4, size4);

printf("Test case 4 - Expected output: 0, Actual output: %d\n", result4);

// Test case 5: weight and size both zero

int weight5 = 0;

double size5 = 0;

int result5 = isValidPackage(weight5, size5);

printf("Test case 5 - Expected output: 0, Actual output: %d\n", result5);

// Test case 6: weight as maximum integer value

int weight6 = INT\_MAX;

double size6 = 0.5;

int result6 = isValidPackage(weight6, size6);

printf("Test case 6 - Expected output: 0, Actual output: %d\n", result6);

// Test case 7: size as maximum double value

int weight7 = 20;

double size7 = DBL\_MAX;

int result7 = isValidPackage(weight7, size7);

printf("Test case 7 - Expected output: 0, Actual output: %d\n", result7);

// Test case 8: weight and size as non-numeric values

int weight8 = 'a';

double size8 = 'b';

int result8 = isValidPackage(weight8, size8);

printf("Test case 8 - Expected output: 0, Actual output: %d\n", result8);

// Test case 9: weight as negative value

int weight9 = -5;

double size9 = 0.1;

int result9 = isValidPackage(weight9, size9);

printf("Test case 9 - Expected output: 0, Actual output: %d\n", result9);

// Test case 10: size as zero and weight greater than zero

int weight10 = 10;

double size10 = 0;

int result10 = isValidPackage(weight10, size10);

printf("Test case 10 - Expected output: 0, Actual output: %d\n", result10);

return 0;

}

**F2 BLACKBOX TESTCODE-**

**// Blackbox testcode for convertDestinationNametoPoint()**

// Test case for valid destination name:

const char destName1[] = "New York";

struct Point expectedPoint1(40.7128, -74.0060);

struct Point actualPoint1 = convertDestinationNametoPoint(destName1);

assert(actualPoint1.latitude == expectedPoint1.latitude && actualPoint1.longitude == expectedPoint1.longitude);

// Test case for invalid destination name:

const char destName2[] = "Atlantis";

struct Point expectedPoint2(-1, -1);

struct Point actualPoint2 = convertDestinationNametoPoint(destName2);

assert(actualPoint2.latitude == expectedPoint2.latitude && actualPoint2.longitude == expectedPoint2.longitude);

// Test case for empty destination name:

const char destName3[] = "";

struct Point expectedPoint3(-1, -1);

struct Point actualPoint3 = convertDestinationNametoPoint(destName3);

assert(actualPoint3.latitude == expectedPoint3.latitude && actualPoint3.longitude == expectedPoint3.longitude);

// Test case for destination name with only spaces:

const char destName4[] = " ";

struct Point expectedPoint4(-1, -1);

struct Point actualPoint4 = convertDestinationNametoPoint(destName4);

assert(actualPoint4.latitude == expectedPoint4.latitude && actualPoint4.longitude == expectedPoint4.longitude);

// Test case for destination name with leading and trailing spaces:

const char destName5[] = " Paris ";

struct Point expectedPoint5(48.8566, 2.3522);

struct Point actualPoint5 = convertDestinationNametoPoint(destName5);

assert(actualPoint5.latitude == expectedPoint5.latitude && actualPoint5.longitude == expectedPoint5.longitude);

// Test case for destination name with special characters:

const char destName6[] = "Los Angeles!";

struct Point expectedPoint6(34.0522, -118.2437);

struct Point actualPoint6 = convertDestinationNametoPoint(destName6);

assert(actualPoint6.latitude == expectedPoint6.latitude && actualPoint6.longitude == expectedPoint6.longitude);

// Test case for destination name with mixed case:

const char destName7[] = "LoNdOn";

struct Point expectedPoint7(51.5074, -0.1278);

struct Point actualPoint7 = convertDestinationNametoPoint(destName7);

assert(actualPoint7.latitude == expectedPoint7.latitude && actualPoint7.longitude == expectedPoint7.longitude);

// Test case for destination name with multiple words:

const char destName8[] = "Rio de Janeiro";

struct Point expectedPoint8(-22.9068, -43.1729);

struct Point actualPoint8 = convertDestinationNametoPoint(destName8);

assert(actualPoint8.latitude == expectedPoint8.latitude && actualPoint8.longitude == expectedPoint8.longitude);

// Test case for destination name with non-ASCII characters:

const char destName9[] = "München";

struct Point expectedPoint9(48.1351, 11.5820);

struct Point actualPoint9 = convertDestinationNametoPoint(destName9);

assert(actualPoint9.latitude == expectedPoint9.latitude && actualPoint9.longitude == expectedPoint9.longitude);

// Test case for destination name with leading/trailing numbers:

const char destName10[] = "123 Sydney 456";

struct Point expectedPoint10(-33.8651, 151.2099);

struct Point actualPoint10 = convertDestinationNametoPoint(destName10);

assert(actualPoint10.latitude == expectedPoint10.latitude && actualPoint10.longitude == expectedPoint10.longitude);

**F3 BLACKBOX TESTCODE-**

#include <stdio.h>

#include <string.h>

#include <float.h>

struct Point {

double latitude;

double longitude;

};

/\*\*

\* This function will convert a point to readable destination name

\* pt - point

\*

\* returns array of character of destination name. for example: 8Y

\*/

char\* convertPointToDestinationName(const struct Point pt) {

if (pt.latitude == 90.0) {

return "North Pole";

} else if (pt.latitude == -90.0) {

return "South Pole";

} else if (pt.latitude >= 37.0902 && pt.latitude <= 49.3845 && pt.longitude >= -95.7 && pt.longitude <= -74.71) {

return "New York";

} else if (pt.latitude >= 51.5072 && pt.latitude <= 55.9986 && pt.longitude >= -0.3213 && pt.longitude <= 0.6895) {

return "London";

} else if (pt.latitude >= 48.8156 && pt.latitude <= 48.9166 && pt.longitude >= 2.2258 && pt.longitude <= 2.4699) {

return "Paris";

} else if (pt.latitude >= -34.02 && pt.latitude <= -32.0 && pt.longitude >= 115.5 && pt.longitude <= 151.2) {

return "Sydney";

} else if (pt.latitude >= 35.5494 && pt.latitude <= 35.816 && pt.longitude >= 139.5467 && pt.longitude <= 139.9293) {

return "Tokyo";

} else if (pt.latitude >= -26.4917 && pt.latitude <= -25.9633 && pt.longitude >= 27.9344 && pt.longitude <= 28.3299) {

return "Johannesburg";

} else if (pt.latitude >= -23.7154 && pt.latitude <= -23.3569 && pt.longitude >= -46.8857 && pt.longitude <= -46.3658) {

return "São Paulo";

} else if (pt.latitude >= 45.6856 && pt.latitude <= 51.0543 && pt.longitude >= 7.1299 && pt.longitude <= 14.6047) {

return "München";

} else {

return "Unknown Destination";

}

}

// Test cases

int main() {

// Test case for valid point in the USA:

struct Point pt1 = {40.7128, -74.0060};

char\* destName1 = convertPointToDestinationName(pt1);

printf("Destination name for New York: %s\n", destName1); // expected output: New York

// Test case for valid point in Europe:

struct Point pt2 = {48.8566, 2.3522};

char\* destName2 = convertPointToDestinationName(pt2);

printf("Destination name for Paris: %s\n", destName2); // expected output: Paris

// Test case for valid point in Australia:

struct Point pt3 = {-33.8651, 151.2099};

char\* destName3 = convertPointToDestinationName(pt3);

printf("Destination name for Sydney: %s\n", destName3); // expected output: Sydney

// Test case for valid point in Asia:

struct Point pt4 = {35.6895, 139.6917};

char\* destName4 = convertPointToDestinationName(pt4);Top of Form

printf("Destination name for Asia: %s\n", destName3); // expected output: Tokyo

// Test case for valid point in Africa:

struct Point pt5 = {-26.2041, 28.0473)};

char\* destName5 = convertPointToDestinationName(pt5);Top of Form

printf("Destination name for Africa: %s\n", destName3); // expected output: Johannesburg

**F4 BLACKBOX TESTCODE-**

Here's the complete black box test code for the reachedDestination function:

#include <stdio.h>

#include <math.h>

struct Point {

double x;

double y;

};

int reachedDestination(const struct Point p1, const struct Point p2) {

double distance = sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

if(distance <= 1.00) {

return 1;

}

else {

return 0;

}

}

int main() {

struct Point p1, p2;

int expected\_output, output;

// Test case for the truck just reaching the destination

p1 = (struct Point) {40.7128, -74.0060};

p2 = (struct Point) {40.7129, -74.0060};

expected\_output = 1;

output = reachedDestination(p1, p2);

printf("Test Case 1: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck not reaching the destination

p1 = (struct Point) {48.8566, 2.3522};

p2 = (struct Point) {40.7128, -74.0060};

expected\_output = 0;

output = reachedDestination(p1, p2);

printf("Test Case 2: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck being at the same position as the destination

p1 = (struct Point) {40.7128, -74.0060};

p2 = (struct Point) {40.7128, -74.0060};

expected\_output = 1;

output = reachedDestination(p1, p2);

printf("Test Case 3: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck being very far away from the destination

p1 = (struct Point) {-33.8651, 151.2099};

p2 = (struct Point) {40.7128, -74.0060};

expected\_output = 0;

output = reachedDestination(p1, p2);

printf("Test Case 4: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck being very close to the destination

p1 = (struct Point) {51.5074, -0.1278};

p2 = (struct Point) {51.5074, -0.1277};

expected\_output = 1;

output = reachedDestination(p1, p2);

printf("Test Case 5: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck being at the North Pole

p1 = (struct Point) {90.0, 0.0};

p2 = (struct Point) {90.0, 0.1};

expected\_output = 1;

output = reachedDestination(p1, p2);

printf("Test Case 6: Expected Output = %d, Output = %d\n", expected\_output, output);

// Test case for the truck being at the South Pole

struct Point p1 = {-90.0, 0.0};

struct Point p2 = {-90.0, 0.1};

int expected = 1;

int result = reachedDestination(p1, p2);

if (result == expected) {

printf("Test case 7 PASSED\n");

} else {

printf("Test case 7 FAILED\n");

}

//TEST CASE FOR TRUCK BEING AT THE EQUATOR

struct Point p1 = {0.0, 0.0};

struct Point p2 = {0.0, 1.0};

int expected = 1;

int result = reachedDestination(p1, p2);

if (result == expected) {

printf("Test case 8 PASSED\n");

} else {

printf("Test case 8 FAILED\n");

}

**F5 BLACKBOX TEST CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Truck {

int id;

int capacity;

double size;

};

typedef struct Truck Truck;

Truck\* getTrucksWithCapacity(Truck trucks[], int numberOfTrucks, int weight, double size) {

if (numberOfTrucks == 0 || weight <= 0 || size <= 0) {

return NULL;

}

int count = 0;

Truck\* result = (Truck\*) malloc(sizeof(Truck) \* numberOfTrucks);

for (int i = 0; i < numberOfTrucks; i++) {

if (trucks[i].capacity >= weight && trucks[i].size >= size) {

result[count] = trucks[i];

count++;

}

}

if (count == 0) {

free(result);

return NULL;

}

return result;

}

int main() {

// Test case for when there are no trucks available

Truck empty\_trucks[] = {};

int empty\_trucks\_size = 0;

int empty\_weight = 10;

double empty\_size = 0.5;

Truck\* empty\_result = getTrucksWithCapacity(empty\_trucks, empty\_trucks\_size, empty\_weight, empty\_size);

printf("Test case 1 result: ");

if (empty\_result == NULL) {

printf("Pass\n");

} else {

printf("Fail\n");

}

free(empty\_result);

// Test case for when all trucks are below capacity

Truck below\_trucks[] = {

{1, 500, 20},

{2, 600, 25},

{3, 700, 30}

};

int below\_trucks\_size = 3;

int below\_weight = 100;

double below\_size = 5.0;

Truck\* below\_result = getTrucksWithCapacity(below\_trucks, below\_trucks\_size, below\_weight, below\_size);

printf("Test case 2 result: ");

if (below\_result == NULL) {

printf("Pass\n");

} else {

printf("Fail\n");

}

free(below\_result);

// Test case for when some trucks have exactly enough capacity

Truck some\_trucks[] = {

{1, 500, 20},

{2, 600, 25},

{3, 700, 30}

};

int some\_trucks\_size = 3;

int some\_weight = 200;

double some\_size = 10.0;

Truck\* some\_result = getTrucksWithCapacity(some\_trucks, some\_trucks\_size, some\_weight, some\_size);

printf("Test case 3 result: ");

if (some\_result != NULL && some\_result[0].id == 2 && some\_result[1].id == 3) {

printf("Pass\n");

} else {

printf("Fail\n");

}

free(some\_result);

// Test case for when only one truck has enough capacity

Truck one\_truck[] = {

{1, 500, 20},

{2, 600, 25},

{3, 700, 30}

};

int one\_truck\_size = 3;

int one\_weight = 250;

double one\_size = 12.5;

Truck\* one\_result = getTrucksWithCapacity(one\_truck, one\_truck\_size, one\_weight, one\_size);

printf("Test case 4 result: ");

if (some\_result != NULL && some\_result[0].id == 2 && some\_result[1].id == 3) {

printf("Pass\n");

} else {

printf("Fail\n");

}

free(some\_result);