TEST CASES WITH FUNCTION-

struct Package createPackage(int packageId, int weight, double size, struct Point destination);

CASE1: Test for valid package creation with fractional size:

Input: int packageId = 1;

int weight = 750;

double size = 0.25;

Point destination = {10, 20};

Expected output: Test for valid package creation with fractional size passed. (0 or 1)

CASE2: Test for package creation with large weight:

Input: int packageId = 2;

int weight = 10000; // Large weight is valid

double size = 0.5;

Point destination = {30, 40};

Expected output: Test for package creation with large weight passed. (0 or 1)

CASE3: Test for package creation with destination at the origin:

Input: int packageId = 3;

int weight = 500;

double size = 1.0;

Point destination = {0, 0};

Expected output: Test for package creation with destination at the origin passed. (0 or 1)

TEST CASES WITH FUNCTION-

int validatePackage(int weight, double size, struct Point destination);

CASE1: Test for a valid package:

Input: int weight = 500;

double size = 0.75;

struct Point destination = {10, 20};

Expected output: Test for valid package passed. (0 or 1)

CASE2: Test for an invalid package with zero weight:

Input: int weight = 0; (invalid weight)

double size = 0.5;

struct Point destination = {30, 40};

Expected output: Test for invalid package with zero weight may pass. (0 or 1)

CASE3: Test for an invalid package with negative size:

Input int weight = 250;

double size = -0.25; (invalid size)

struct Point destination = {50, 60};

Expected output: Test for invalid package with negative size passed. (0 or 1)

TEST CASES WITH FUNCTION-

int assignPackage(struct Package package, struct Truck \*truck);

CASE1: Test for a valid package assignment:

Input: int truckId = 1;

int truckWeight = 1000;

double truckSize = 1.0;

struct Route truckRoute = {};

struct Package truckPackages[10];

Expected output: Test for a valid package assignment passed.

CASE2: Test for an invalid package assignment (truck has no space):

Input: int truckId = 2;

int truckWeight = 400;

double truckSize = 0.5;

struct Route truckRoute = {};

struct Package truckPackages[10];

Expected output: Test for an invalid package assignment (truck has no space) passed.

CASE3: Test for an invalid package assignment with NULL truck pointer

assignPackage(package, NULL);

Expected output:

Test for an invalid package assignment with NULL truck pointer passed.

TEST CASES WITH FUNCTION-

struct Truck createTruck(int truckId, int weight, double size, struct Route route, struct Package \*packages);

CASE1: Test for a valid truck that can carry the package:

Input: int truckId = 1;

int truckWeight = 1000;

double truckSize = 1.0;

struct Route truckRoute = {};

struct Package truckPackages[10];

Expected output: Test for a valid truck that can carry the package passed. (0 or 1)

CASE2: Test for an invalid truck (insufficient weight capacity):

Input: int truckId = 2;

int truckWeight = 500; (Insufficient weight capacity)

double truckSize = 1.0;

struct Route truckRoute = {};

struct Package truckPackages[10];

Expected output: Test for an invalid truck (insufficient weight capacity) passed. (0 or 1)

CASE3: Test for an invalid truck (insufficient size capacity):

Input: int truckId = 3;

int truckWeight = 800;

double truckSize = 1.0; (Insufficient size capacity)

struct Route truckRoute =

struct Package truckPackages[10];

Expected output: Test for an invalid truck (insufficient size capacity) passed. (0 or 1)

TEST CASES WITH FUNCTION-

int validateTruck(struct Truck truck, struct Package package);

CASE1: Test for valid user input and route display:

Input: Please enter the route of the truck: A B C

Expected output: Test for valid user input and route display passed.

CASE2: Test for displaying an empty route (no user input):

Input: Please enter the route of the truck:

Expected output: Test for displaying an empty route passed.

CASE3: Test for displaying a route with special characters:

Input: Please enter the route of the truck: @$

Expected output: Test for displaying a route with special characters passed.