init.h Function Spec(s)

**Function: void init()**

**Purpose:**

Initializes two global variables that will be used throughout the project:  
struct Map\* baseMap: which will contain the map featuring our delivery range, buildings, and base.

Struct Truck\* trucks: which will contain our 3 trucks, each assigned to the blue, green, or yellow route.

**Scope:**

This function is called in the int main(void) function to initialize global variables that will be needed for our functions.

**Parameters and Returns:**

* **void**: This function does not receive any parameters and returns void.

Shipments.h Function Spec(s)

**Function: void read()**

**Purpose:**

This function is used to get the user’s input from the input buffer, and create a shipment struct based on the user’s input. It reads the user-input into 3 variables: weight (int), size (double), and destination (char [4]), with each of the input separated by a space as the delimiter. The destination variable is later broken down into a row and column variable and converted from a human readable format (A1) to a computer-readable format (0, 0). This function acts as the main menu, which processes user input, sends input into validation, and then creates a shipment struct if input is valid and sends it to a function to get it assigned to the correct truck. If user input ‘0 0 x’ is detected, then the program will terminate.

**Scope:**

This function is called in the main() function and is used to show the main menu of the program.

**Parameters and Returns:**

* **void**: This function does not receive any parameters and returns void.

**Behavior:**

* This function prints out the main menu of the program and prompts users to enter their package’s information, and sends user input into validation, and valid packages to be assigned. It repeats this loop until the user input is ‘0 0 x’
* This function breaks down the user input into 3 variables: weight, size, and destination.
* If destination is determined to not be “x”, then it further breaks the destination down from the human-readable format (1A) to two variables: row and column. In this example, 1 becomes 0, and A becomes 0, which creates a Point struct of {0, 0}
* A shipment Struct will then be created if the variables passes the validation check done through the validate() function
* Valid shipments are passed onto the AssignTrucks() function.

**Function: int validate(double size, int weight, struct Point destination)**

**Purpose:**

This function is used to verify that the parameters received are valid parameters for the Shipment struct. This function is essential to ensure that all shipments have valid size, weight, and destinations. This ensures that all packages passed to AssignTruck are valid.

**Scope:**

This function gets used by read() to determine if the shipment parameters are valid.

**Parameters:**

* **double size**: The package size, which should only be 0.25, 0.50, or 1.00
* **int weight**: The package weight, which should be between 1 to 1000 kg.
* **struct Point destination**: The shipment’s delivery destination from {0, 0} to {24, 24}, and all destinations should be inside of a building.

**Returns:**

* **1**: If the package is valid.
* **-1**: If the size is invalid.
* **-2**: If the weight is invalid.
* **-3**: If the destination is invalid.
* **0**: If the exit code is entered.

**Behavior:**

* This function uses the populateMap() function to create a test map to check for valid destinations.
* The initialize a check integer variable to 1.
* The function will then go through a series of if and else if statements to check if each parameter is valid. If invalid, the check variable gets changed and an error message appears.
* It will then return the check variable.

Utils.h Function Spec(s)

**Function: getAllTruckPaths**

**Purpose:**

Get all the valid routes from the “shortestPath” function. This is necessary because there are points of a truck’s route that cannot reach the destination due to traveling backwards, reaching the edge of the map, etc.

This ensures that the assumption of Routes in the “routes” are valid - therefore choosing the smallest path will also be valid.

**Scope:**

This function is called in the global scope to populate the “routes” array which is also in the global scope.

**Parameters:**

* **s**: a Shipment struct containing the destination of a shipment
* **truck**: a Truck struct containing the route the truck travels on the map
* **map**: a const Map pointer to a struct containing all inbound points in the map
* **routes**: an array of Route structs that holds all valid routes
* **size**: a pointer to an integer that holds the size of the “routes” array

**Returns:**

* **void**: This function does not return anything, but it populates the "routes" array with all valid routes.

**Behavior:**

* This function populates the "routes" array with all valid routes that reach the destination of the shipment.
  + A valid route must contain the destination point in its array of Point structs
    - Valid routes have the destination point at the end of their array
  + Note: invalid routes do not have the destination routes in their array of Points
* The "routes" array should be an array of Route structs that holds all valid routes.
* The "size" parameter is a pointer to an integer that holds the size of the "routes" array. It is used to keep track of the number of valid routes found.
* The function uses the "Truck" struct and the "Map" struct to determine the valid routes. The "Truck" struct contains the route the truck travels on the map, and the "Map" struct contains all inbound points in the map.
* The function populates the "routes" array by:
  + Travel to each point of a trucks route
  + Calculate the shortest route at each point the truck travels to
  + Check if the calculate route is valid (i.e. is able to get to the destination)
  + Appends calculated route to “routes” array if valid
* The function modifies the "size" variable to reflect the number of valid routes found by the function.

**Function: getBestRoute**

**Purpose:**

Get the shortest route from the array of valid Routes. The shortest route will be assigned to deliver a package.

**Scope:**

This function is called in the global scope find the index of the shortest path Route in the array of Routes (i.e. routes array)

**Parameters:**

* **routes:** an array of Route structs that holds all valid routes
* **s0**:an integer that holds the size of the "routes" array
* **s**: a Shipment struct containing the destination of the shipment

**Returns:**

* **int**: index of the Route in the "routes" array with the shortest distance

**Behavior:**

* This function loops through all valid routes in the "routes" array and get the routes length stored in “numPoints”
* The "s0" parameter is an integer that holds the size of the "routes" array. It is used to ensure that the function only loops through valid routes.
* The "s" parameter is a Shipment struct that contains the destination of the shipment.
* The function compares the ith Route in the “routes” array and compares if it travels less than the current shortest route
  + If the ith route is shorter, replace the index of previous shortest route with the new shortest route index
* The function keeps track of the index of the route with the shortest distance found so far.
* After looping through all valid routes, the function returns the index of the route with the shortest distance. If there are no valid routes, the function returns 0

**Function: containsDestination**

**Purpose:**

Filtering function to evaluate if a single route is valid

**Scope:**

This function is called in **getAllTruckPaths** assess if each calculated path if valid

**Parameters:**

* **route**: a Route struct representing a route traveled by a truck
* **s**: a Shipment struct containing the destination of the shipment

**Returns:**

* **int**: 1 if the destination is in the route, 0 if not

**Behavior:**

* This function checks if the given route contains the shipment destination.
* Valid routes have similar behavior where the destination point will always be the last element in the array of points
* Check if the routes last point (row, col) is equal to the destinations point (row, col), and return 1 (i.e. true) if they are equal

**Function: printRoute**

**Purpose:**

Output the divergent points of a truck to the console.

**Scope:**

This function is called in the global scope find the index of the shortest path Route in the array of Routes (i.e. routes array)

**Parameters:**

* **Parameters:**
* **route**: a Route struct containing the points in the given route

**Returns:**

* **void**: does not return a value, only prints to console

**Behavior:**

* This function loops through all the points in the route, and displays each one in the format (row, col)

Truck.h Function Spec(s)

**Function: calculateUtilizationScore**

**Purpose:**

Assign a “utilization” score to a truck. A utilization score is a metric to measure how full a truck is.

Note: utilization score = (cargoWeight / MAX\_CARGO\_WEIGHT) / (cargoVol / MAX\_CARGO\_VOL);

**Scope:**

This function is called in **getBestRoute**, reassigning the best route when volume or weight is the deciding factor between trucks that are equidistant to a package destination.

**Parameters:**

* **t**: a Truck struct representing a truck, holds the current weight, and current volume of the truck

**Returns:**

* **float**: the utilization score, which ranges from 0 to 1, where 0 means the truck is empty and 1 means the truck is full

**Behavior:**

* This function calculates the utilization score for a given truck.
* The function calculates the utilization score by dividing the cargo weight of the truck by the maximum cargo weight that the truck can carry and dividing the cargo volume of the truck by the maximum cargo volume allowed. These two ratios are then divided to get the final utilization score.