Function Specification

|  |
| --- |
| 1. findTruckForShipment  char findTruckForShipment(struct Shipment s);  Description: Finds the best truck for a shipment based on shortest distance, availability, and validation of size and weight.  Parameters:   * s: Shipment from user input, including weight, box size, and destination information.   Returns:  An character representing route symbol of the assigned truck; 2 for blue truck, 4 for green truck, 8 for yellow truck and 0 if no truck is assigned |
| 2. truckLimitFactorComparison char truckLimitFactorComparison(struct Truck truck1, struct Truck truck2);  Description: Calculates the limit factor percentage and compares two trucks to determine the less full truck.  Parameters:   * truck1: The first truck to be compared, including total weight, box size, and route symbol information. * truck2: The second truck to be compared, including total weight, box size, and route symbol information.   Returns:  An character representing the route symbol of less full truck; 2 for blue truck, 4 for green truck, 8 for yellow truck and 0 if they have the same percentage. |
| 3. findBestRoute  int findBestRoute(struct Route\* routes, int size, struct Truck\* truck);  Description: Finds the best route by comparing possible routes from three different trucks and returns the index of the best route.  Parameters:   * routes: All potential routes available in the routes array after shortest path calculation. * size: Size of the routes array for looping purposes. * truck: The truck to which the best route will be added.   Returns:  An integer representing the index of the best route in the routes array. |
| 4. isDestination int isDestination(struct Route route, struct Shipment shipment);  Description: Checks if the destination point is included in the route by iterating through the route points.  Parameters:   * route: The route to be checked if any of the route points is the destination of the shipment. * shipment: Shipment destination to be compared.   Returns:  An integer returns 1 when the destination is found and included, and 0 when the destination is not found. |
| 5. printResult void printResult(struct Route bestRoute, struct Shipment shipment);  Description: Prints the result as output, including the best truck, diversion, and proposed route.  Parameters:   * bestRoute: The best route to be delivered will be printed. * shipment: Shipment data, including the destination, to be printed.   Returns: None (void) |
| 6. isValidPoint  Description: Validates if a given point is valid within the map. int isValidPoint(const struct Map\* map, const struct Point p1);  Parameters:   * map: The map of the delivery area with buildings on it. * p1: The point to be validated.   Returns:  int: An integer returns 1 if the point is valid, and 0 otherwise. |

|  |
| --- |
| 7. isValidWeight int isValidWeight(int weight);  Description: Validates if the weight of a shipment is within the maximum weight that the truck can hold up to.  Parameters:   * weight: The weight to be added to the truck.   Returns:  int: An integer returns 1 if the point is valid, and 0 otherwise. |
| 8. isValidSize  int isValidSize(double size);  Description: Validates if the box size is valid for a shipment and if the size added to the shipment is within the maximum capacity of the truck.  Parameters:   * size: The box size to be validated.   Returns:  int: An integer returns 1 if the point is valid, and 0 otherwise. |
| 9. addShipment  void addShipment(struct Shipment s);  Description: Adds the shipment (weight, box size, and destination) to the best available truck.  Parameters:   * s: Shipment from user input, including weight, box size, and destination information.   Returns: No return(void) |
| 10. allPossibleRoute void allPossibleRoute(struct Truck\* truck, struct shipment s, struct Route\* route[maxNoRoute], int\* size)  Description: Generates all possible paths from the three routes to a shipment destination. It loops through all the points on these three routes, gets the shortest possible path from those points to the shipment destination point, and stores the possible paths in the route array.  Parameters:   * truck: The truck in the three routes. * s: The shipment from the user input. * route: An array to store all the possible paths. * size: the size of the route array for iteration   Returns: No return(void) |
| 11. populateMap struct Map populateMap(void);  Description: Creates a map with the position of all buildings in it.  Parameters: No Parameters  Returns: A struct Map representing a map with the position of all buildings added to it. |
| 12. getNumRows  int getNumCols(const struct Map\* map);  Description: Retrieves the number of rows in a map.  Parameters:   * map: The map to query.   Returns: An integer representing the number of rows in the map. |
| 13. getNumCols  int getNumCols(const struct Map\* map);  Description: Retrieves the number of columns in a map.  Parameters:   * map: The map to query. * Returns: The number of columns in the map.   Returns: An integer representing the number of columns in the map. |
| 14. printMap  void printMap(const struct Map\* map, const int base1, const int alphaCols);  Description: Prints the map using symbols to represent different elements on the map, such as buildings, routes, and paths.  Parameters:   * map: The map to print. * base1: If true, print row indices from 1 up; otherwise, print from 0 up. * alphaCols: If true, print column headers as letters; otherwise, print as numbers.   Returns: No return(void) |
| 15. addRoute  struct Map addRoute(const struct Map\* map, const struct Route\* route);  Description: Adds a route to a map using the indicated symbol.  Parameters:   * map: The map to add the route to. * route: The route to add to the map.   Returns: A struct of Route representing a copy of the original map with the route added to it. |
| 16. addPtToRoute  void addPtToRoute(struct Route\* route, struct Points pt);  Description: Adds a point to a route.  Parameters:   * route: The route to which the point should be added. * pt: The point to add to the route.   Returns: No return(void) |
| 17. addPointToRouteIfNot void addPointToRouteIfNot(struct Route\* route, const int row, const int col, const struct Points notThis);  Description: Adds a point to a route if it is not equal to another point.  Parameters:   * route: The route to which the point should be added. * row: The row of the point to add to the route. * col: The column of the point to add to the route. * notThis: The point will be added to the route ONLY if it is not equal to this point.   Returns: No return(void) |
| 18. addPointToRoute  void addPointToRoute(struct Route\* route, const int row, const int col);  Description: Adds a point to a route.  Parameters:   * route: The route to which the point should be added. * row: The row of the point to add to the route. * col: The column of the point to add to the route.   Returns: No return(void) |
| 19. getBlueRoute  struct Route getYellowRoute(void);  Description: Builds and returns the route for the blue trucks.  Parameters: No Parameters  Returns: A struct of Route representing the route for the blue trucks. |
| 20. getGreenRoute  struct Route getYellowRoute(void);  Description: Builds and returns the route for the green trucks.  Parameters: No Parameters  Returns: A struct of Route representing the route for the green trucks. |
| 21. getYellowRoute  struct Route getYellowRoute(void);  Description: Builds and returns the route for the yellow trucks.  Parameters: No Parameters  Returns: A struct of Route representing the route for the yellow trucks. |
| 22. distance double distance(const struct Points\* p1, const struct Points\* p2);  Description: Calculates the Euclidean distance between two points.  Parameters:   * p1: The first point. * p2: The second point.   Returns: A double representing the distance between p1 and p2. |
| 23. shortestPath struct Route shortestPath(const struct Map\* map, const struct Points start, const struct Points dest);  Description: Calculates the shortest path between two points on a map, ensuring that the path does not pass through buildings.  Parameters:   * map: The map showing the location of buildings. * start: The point to start from. * dest: The point to go to.   Returns: A struct of Route representing the shortest path from start to dest. If there is no path, then a Route of zero length is returned. If start and dest are the same point, it also returns a Route of zero length. |
| 24. getPossibleMoves  struct Route getPossibleMoves(const struct Map\* map, const struct Points p1, const struct Points backpath);  Description: Calculates all adjacent squares to a given point, ensuring that the squares do not overlap with buildings and do not include the backpath.  Parameters:   * map: The map showing the location of buildings. * p1: The point to calculate possible moves for. * backpath: The previous point visited on the path traveled, so backward movement is excluded.   Returns: A struct of Route representing the collection of adjacent points that could potentially be moved to. |
| 25: getClosestPoint  int getClosestPoint(const struct Route\* route, const struct Points pt);  Description: Calculates the Euclidean distance from every point in a route to a single point and returns the index of the point in the route that is closest to the specified point.  Parameters:   * route: The route to use to find the closest point. * pt: The point to find the member of the route that is closest to this point.   Returns: An integer representing the index of the closest point on the route to the specified point. If the route is empty, it returns -1. |
| 26: eqPt  int eqPt(const struct Points p1, const struct Points p2);  Description: Compares two points for equality.  Parameters:   * p1: The first point. * p2: The second point.   Returns: An integer representing 1 is true if p1 is equal to p2. |