LOGISTIC MANAGEMENT SYSTEM



Name : K.T.A.T.N.Kasthuriratne

Index : As20240635

Table of contents

a).Introduction

b).Methodology

c).Planining

d).Describing the code

1. Include and define
2. Global arrays and function declaration
3. Main function
4. Display menu function
5. Add city function
6. Manage city function
7. Manage distance function
8. Display distance table function
9. Find shortest path function
10. Display delivery estimate function
11. Handle delivery function
12. Show report function
13. Save to file and Load from file functions

e).Code outputs

INTRODUCTION

This project is a simple logistic management system developed using the C programming language. It is a software application designed to manage the operations of a Delivery system between cities. This system is very important to users to reduce manual work and improve accuracy by digitally tracking . It is aimed at beginners who wish to understand how to manage real-world tasks using fundamental C programming concepts such as arrays, loops, functions, and conditional statements.

METODOLOGY

This Logistics Management System uses a structured procedural programming approach with C. The code features a modular design that breaks down complex logistics operations into specialized functions for managing cities, distances, deliveries, and reports. It uses static arrays for data storage and an adjacency matrix to represent the city network as a graph. The main pathfinding feature uses a recursive depth-first search (DFS) algorithm with backtracking to find the shortest delivery route between cities. The system includes input validation at every step to maintain data integrity and prevent errors. Business logic is implemented through detailed cost calculations that consider distance, weight, vehicle type, fuel consumption, and profit margins. File-based persistence ensures data continuity across sessions by saving city networks and delivery records to text files. The menu-driven interface offers an intuitive user experience, making the system easy to use for small-to-medium scale logistics operations. This method emphasizes simplicity, maintainability, and educational clarity rather than performance optimization, making it well-suited for learning basic concepts in data structures, algorithms, and business application development.

PLANNING

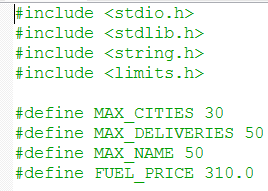
The planning phase began with the analysis of the business problem: management of a delivery network consisting of multiple cities, calculation of the best routes, and estimating the cost based on different vehicle types. The system was designed based on a modular approach where each major function (city management, distance management, delivery processing, reporting) is handled by dedicated functions following the single responsibility principle. Data structures were chosen carefully: a 2D adjacency matrix to represent the city network as a graph, static arrays to store city names and delivery records, and parallel arrays to store vehicle specs. Algorithm choice was focused on the use of recursive depth-first search with backtracking for pathfinding, suitable for small-scale networks (up to 30 cities). A menu-driven interface was created to guide the user through operations step by step, with rigorous input validation at every phase to prevent errors. Business logic design incorporated real-world factors: distance-based pricing, weight fluctuation, fuel efficiency calculations, and a 25% markup. The file persistence strategy uses two different text files for routes and deliveries to achieve data persistence between sessions. Development was planned in three phases: starting with initial CRUD operations, then core delivery and pathfinding logic, and concluding with reporting and persistence functionality. Error handling was built into the design from the start, including boundary checking, duplicate prevention, and capacity checking. This methodical design process ensured the final code would be simple, maintainable, and helpful for small-to-medium logistics companies while also being an excellent instructional example of applying data structures and algorithms to real business problems.

DISCRIBING THE CODE

1.Include and define

Include texts are using for user input and output commanding data access the computer.

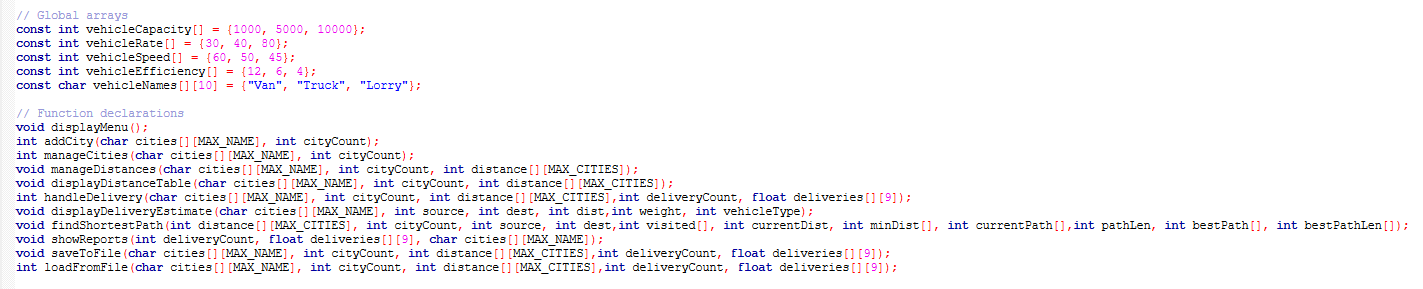
Define is use for input fixable value.



2.Global arrays and function declaration

A **global array** is an array that is declared outside of all functions.Because of this, it can be accessed by all functions in the program.

A function declaration tells the compiler about a function’s name, return type, and parameters before the function is used in the program.



3.Main function

The main() function in this Logistics Management System acts as the central control unit of the entire program. It begins by declaring and initializing several arrays to store city names, distances between cities, and delivery records. The program then attempts to load previously saved data from a file using the loadFromFile() function and updates the number of cities and deliveries accordingly. After displaying a welcome message, the program enters an infinite loop that presents a menu to the user, allowing them to manage cities, enter distances, handle deliveries, or generate reports. Each menu option triggers a specific function, such as manageCities(), manageDistances(), or handleDelivery(), to perform the chosen task. When the user selects the exit option, the program saves all updated data back to the file and terminates gracefully. Overall, this main() function efficiently manages program flow, user interaction, and data handling in a structured and organized manner

4.Display menu function

The displayMenu() function is a simple user-interface function designed to show the main options available in the Logistics Management System. When this function is called, it prints a neatly formatted menu on the screen. its sole purpose is to make the program more user-friendly by displaying all available actions in an organized and easy-to-read format.

5.Add city function

The addCity() function is responsible for adding a new city to the city list in the Logistics Management System. It takes two parameters: a 2D character array cities, which stores the names of all existing cities, and an integer cityCount, which keeps track of how many cities are currently stored.

6.Manage city function

This C function, manageCities(), offers a simple menu-based system for handling a list of city names stored in a 2D array. It lets the user perform four main tasks: add a new city, rename an existing city, remove a city, or list all cities. Depending on the user's choice, it calls the right function, like addCity, or carries out the action directly. The function also checks for errors to avoid invalid inputs, such as trying to rename or delete when there are no cities or entering an invalid city number. After any changes, it returns the updated cityCount.

7.Manage distance function

In a city management system, the user can set and control the distances between city pairs using the manageDistances() function. Before continuing, it makes sure there are at least two cities. After displaying the list of cities that are available, the function asks the user to choose two cities and enter the kilometers that separate them. It avoids establishing a distance between the same city and uses validation to stop invalid or negative distances. It guarantees that the distance between cities A and B is equal to the distance between cities B and A by storing the distance symmetrically in a 2D distance matrix after validation. Lastly, it shows the distance entered to validate the update.

8.Display distance table function

The displayDistanceTable() function neatly displays a distance table showing the distances between all pairs of cities. It first checks whether any cities exist and stops if there are none. Then, it prints a formatted table where the first row lists all city names as column headers, and each subsequent row shows a city followed by its distances to every other city. The data is taken from the 2D distance array, which stores the distance values between cities. This function helps users easily visualize the distance relationships among all cities in a clear, tabular format.

9.Find shortest path function

The findShortestPath() function finds the shortest route between two cities using a recursive depth-first search approach. It explores all possible paths between the source and destination cities by checking connected routes in the distance matrix. During the process, it keeps track of the total distance and updates the shortest path whenever a shorter one is found. The function uses backtracking to unmark visited cities after exploring each route, ensuring all possible paths are considered efficiently.

10.Display delivery estimate function

The displayDeliveryEstimate() function calculates and displays a detailed delivery cost and time estimate between two cities. It uses data such as distance, package weight, and vehicle type to determine the total delivery charge. The function computes key values including base cost, fuel consumption, fuel cost, total operational cost, profit, and customer charge. It also estimates the delivery time based on the vehicle’s speed. Finally, it presents all these details in a neatly. formatted report, giving a clear summary of the delivery cost breakdown and time estimate for the selected route.

11.Handle delivery function

The handleDelivery() function manages the process of recording a new delivery between two cities. It begins by checking whether there are enough cities and available delivery slots. The user is prompted to select a source and destination city, enter the package weight, and choose a vehicle type (van, truck, or lorry). The function validates inputs to ensure that city numbers, vehicle types, and weights are within valid limits. It then calls the findShortestPath() function to determine the shortest route between the cities and displays the optimal route using displayDeliveryEstimate(). Finally, it calculates important delivery details such as distance, cost, fuel usage, time, and profit, storing them in the deliveries array for record-keeping.

12.Show report function

The showReports() function generates a summary report of all recorded deliveries. It calculates important performance metrics such as the total number of deliveries, total distance covered, average delivery time, total revenue, and total profit. It also identifies the longest and shortest delivery routes based on distance. The function then displays these results in a neatly formatted report, giving a clear overview of the system’s overall performance and efficiency. If no deliveries are recorded, it notifies the user accordingly.

13.Save to file and Load from file functions

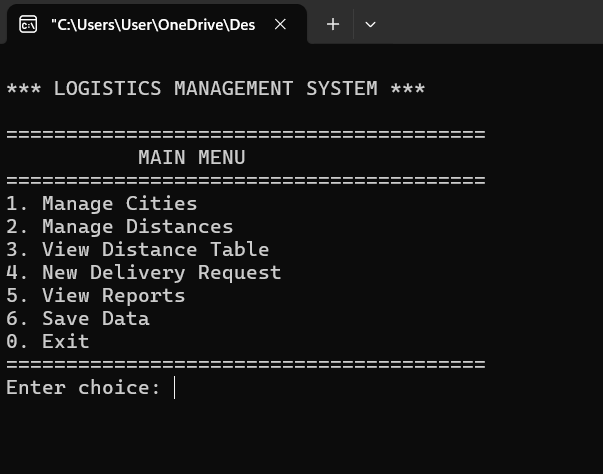
The saveToFile() and loadFromFile() functions handle the saving and loading of program data to and from files, ensuring that city, distance, and delivery information are preserved between runs.

The saveToFile() function writes all current data to two text files: routes.txt for storing city names and distance matrices, and deliveries.txt for saving delivery details such as source, destination, distance, weight, cost, and profit.

The loadFromFile() function reads data back from these files when the program starts. It restores the city list, distance table, and delivery records into memory, allowing the user to continue from where they left off. If files are missing, it safely returns without crashing. Together, these functions provide simple and effective data persistence for the delivery management system.

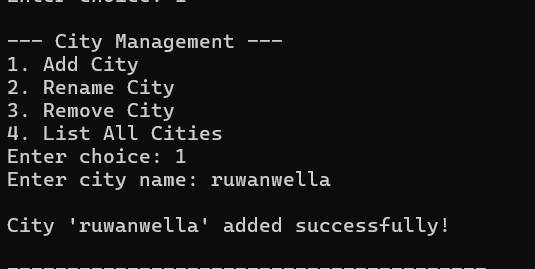
CODE OUTPUTS

1.Main menu

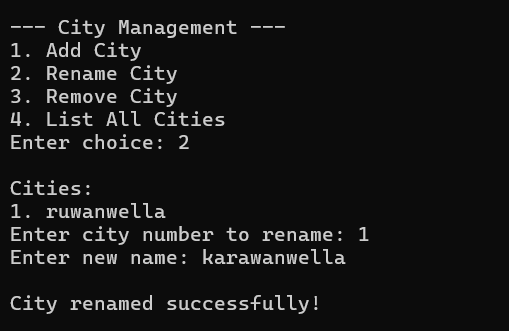


2.City management

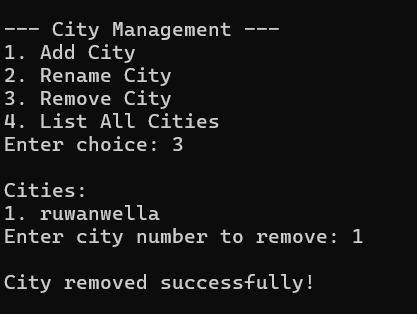
Add city



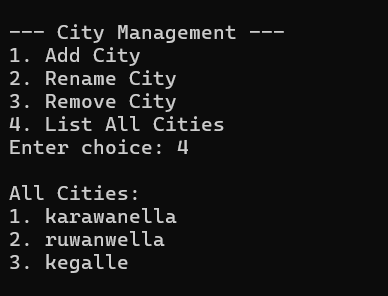
Rename city



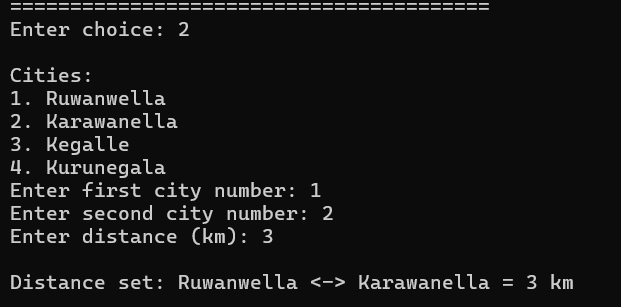
Remove city



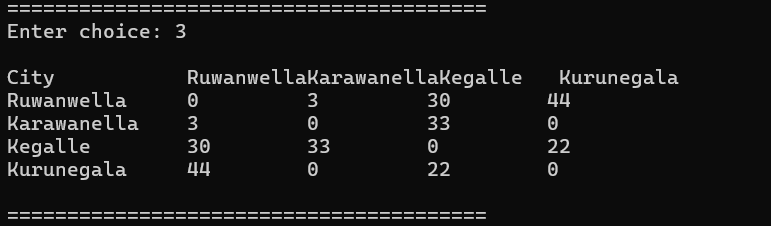
List all cities



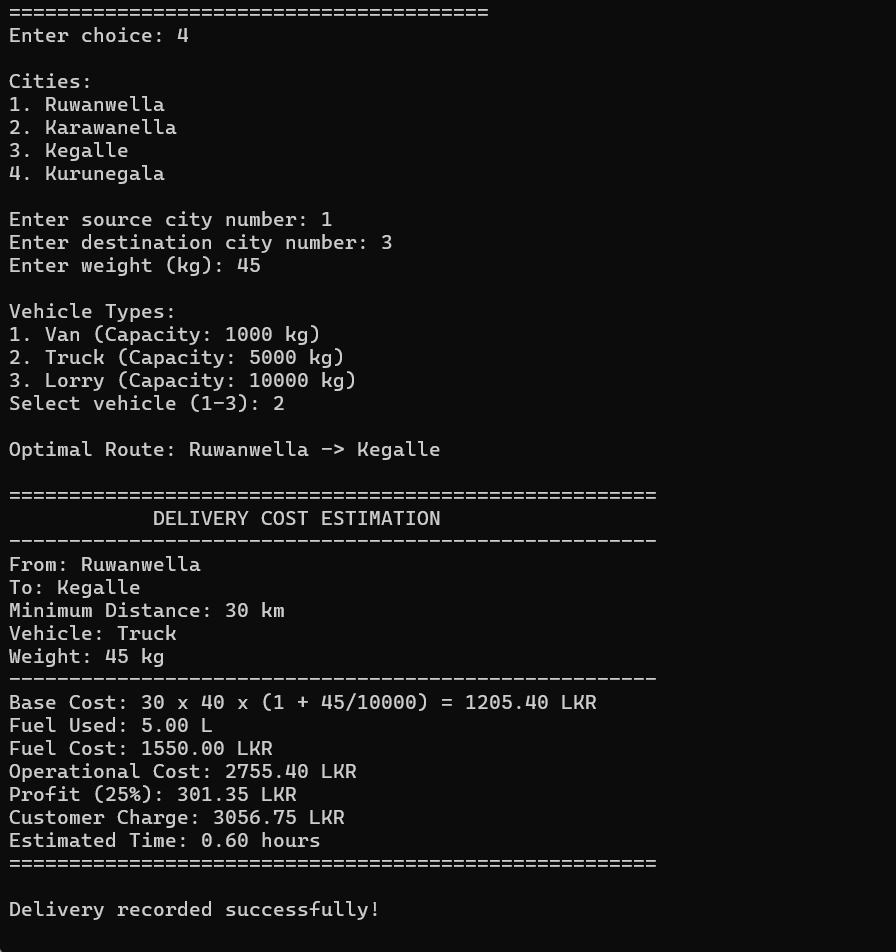
3.Manage distance



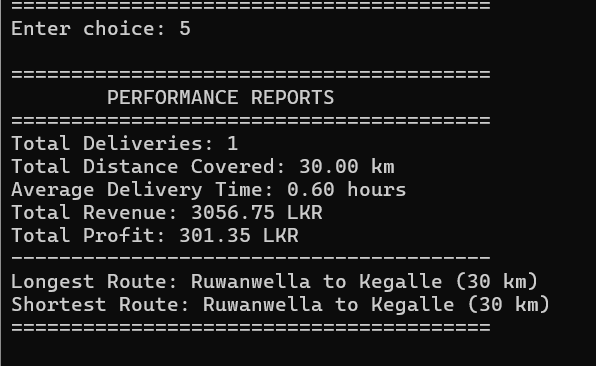
4.View distance table



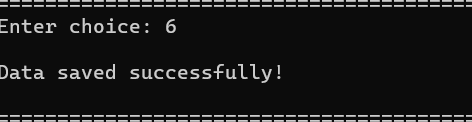
5.New delivery request



6.View reports



7.Save data



8.Exit

