Logistics Management System

Course: CSC 1012 Introduction to Computer Programming

University: University of Sri Jayewardenepura

Faculty: Faculty of Applied Sciences

Student Name: Dimal Withanage

Student ID: AS20240406

Date: 26/10/2025

Table of Contents

1. Introduction

1.1. Project Objective

The primary objective of this project is to design and implement a menu-driven logistics management system using the C programming language. This system applies key programming concepts, including arrays, functions, loops, and conditionals, to simulate the management of delivery logistics.

1.2. Scope

The program is designed to perform the following core tasks:

- Manage a list of cities and the distances between them.
- Handle customer delivery requests, including source, destination, and package weight.
- Select the appropriate vehicle for delivery based on package weight.
- Calculate and estimate delivery costs (base, fuel, total) and profit.
- Estimate the time required for a delivery.
- Find the shortest delivery route using Exhaustive Search algorithm.
- Track completed deliveries and generate performance reports.
- Save and load system data (routes, deliveries) to and from text files for persistence.

1.3. GitHub Repository

The complete source code and commit history for this project are available on GitHub.

Repository URL: https://github.com/DimalWithanage/Logistics-Management-System

2. System Design

2.1. Data Structures

The system relies on several global arrays and constants to store and manage data during runtime:

- **Constants:**
    - MAX_CITIES (30): Maximum number of cities the system can manage.
    - MAX_NAME_LENGTH (50): Maximum character length for city names.
    - MAX_DELIVERIES (50): Maximum number of deliveries that can be stored in one session.
    - FUEL_PRICE (310.0): A constant representing the price of fuel (LKR per liter).
- **City and Route Data:**
    - char cities[MAX_CITIES][MAX_NAME_LENGTH]: A 2D array to store the names of all cities.
    - int cityCount: An integer to track the current number of cities added.
    - int distances[MAX_CITIES][MAX_CITIES]: A 2D array (adjacency matrix) to store the distances between cities. distances[i][j] holds the distance from city i to city j.
- **Vehicle Data:**
    - char vehicleTypes[3][20]: Stores the names of available vehicle types ("Van", "Truck", "Lorry").
    - int vehicleCapacity[3]: Stores the maximum weight capacity (kg) for each vehicle type.
    - int vehicleRatePerKm[3]: Stores the base operational cost (LKR per km) for each vehicle.
    - int vehicleAvgSpeed[3]: Stores the average travel speed (km/h) for each vehicle.
    - int vehicleFuelEfficiency[3]: Stores the fuel efficiency (km per liter) for each vehicle.

- **Delivery Data:**
  - Arrays prefixed with delivery... (e.g., deliverySource, deliveryDest, deliveryDistance, deliveryWeight, deliveryTotalCost, etc.) are used to store the details of each completed delivery. Each array is of size MAX_DELIVERIES, and the index corresponds to a specific delivery.
  - int deliveryCount: An integer to track the current number of deliveries completed.

2.2. Modular Structure

As required, the program is implemented in a modular manner. The logic is divided into well-defined functions, each responsible for a specific task. For example, addCity() handles only the addition of a new city, findLeastCostRoute () handles only the shortest path calculation, and generateReports() handles the reporting submenu. This approach improves code clarity, reusability, and maintainability.

2.3. File Handling

The system implements data persistence by saving data to and loading data from text files.

- routes.txt: Stores the cityCount, the list of city names, and the distance matrix.
- deliveries.txt: Stores the deliveryCount and the details of all completed deliveries.

These files are loaded when the program starts (if they exist) and are saved when the user exits the program, ensuring that data is not lost between sessions.

3. Implemented Functions

This section details the functions implemented in the main.c source file.

3.1. Main and Menu Functions

- int main():
  - This is the entry point of the program.
  - It first calls loadFromFile() to load any existing data.
  - It then enters a do-while loop that displays the main menu and prompts the user for a choice.
  - A switch statement is used to call the appropriate function based on the user's choice (cityManagement, distanceManagement, vehicleManagement,, performanceReports, etc.).

  - The loop continues until the user selects option 7 ("Exit"), at which point saveToFile() is called before the program terminates.

3.2. Route Management Functions

- void addCity():
  - Checks if the cityCount is less than MAX_CITIES.
  - If there is space, it prompts the user to enter a new city name.
  - The new city name is stored in the cities array, and cityCount is incremented.
  - If the city limit is reached, it displays an error message.
- void inputDistance ():
  - Prompts the user to enter the distances between all pairs of cities currently in the system.
  - It uses nested loops to iterate through the distances matrix.
  - It ensures that the distance from a city to itself (distances[i][i]) is always 0.
  - It assumes distances are symmetrical and sets distances[j][i] = distances[i][j].
- void displayCities():
  - Checks if cityCount is zero. If so, prints a message that no cities are available.
  - If cities exist, it iterates through the cities array and prints each city name with its corresponding index number.
- void displayDistanceMatrix():
  - Checks if cityCount is zero.
  - If cities exist, it prints the distance matrix in a formatted table, showing the city names as headers for rows and columns.

3.3. Shortest Path Functions

- void findLeastCostRoute ():
  - This function serves as the user interface for finding the shortest path.
  - It prompts the user to enter a source city and a destination city.

    - It validates the indices and then calls calculateAndDisplayCost () to perform the calculation.

3.4. Delivery Management Functions

- void deliveryRequestHandling ():
    - Handles all the steps to create a new delivery.
    - Prompts the user for source city, destination city, and package weight.
    - Calls displayCities() to validate cities.
    - Calls findLeastCostRoute () to get the shortest distance for the delivery.
    - If a valid path and vehicle are found, it calls calculateAndDisplayCost ().
    - It then prints a detailed summary of the delivery, including costs, profit, and estimated time.
    - Finally, it stores all delivery details in the delivery arrays and increments deliveryCount.
- void vehicleManagement():
    - It iterates through the vehicleCapacity array to find the first (smallest) vehicle that can handle the weight.
    - Returns the index of the selected vehicle (0 for Van, 1 for Truck, 2 for Lorry).
    - If the weight exceeds the largest vehicle's capacity, it prints an error.
- void calculateAndDisplayCost(int source, int dest, int weight, int vehicleType):
    - Calculates all costs for the delivery at the given index.
    - baseCost = distances * vehicleRatePerKm
    - fuelCost = fuelUsed * FUEL_PRICE
    - totalCost = baseCost+ fuelCost
    - profit= baseCost* 0.25 (25% profit margin)
    - customerCharge= totalCost+ profit
    - The calculated values are stored in their respective delivery arrays.

3.5. Reporting Functions

- void performanceReports ():
  - Displays a sub-menu for different performance reports.
  - Uses a for loop to call the specific reporting functions.
  - Prints the current value of deliveryCount.
  - Iterates through the deliveryDistance array and sums all distances.
  - Prints the total distance covered for all completed deliveries.
  - Calculates the sum of all deliveryEstimatedTime values and divides by deliveryCount.
  - Prints the average delivery time.
  - Calculates and prints the sum of all deliveryCustomerCharge (Total Revenue) and deliveryProfit (Total Profit).
  - Initializes min and max distances with the first delivery's distance.
  - Iterates through the totalDistance array to find the shortest and longest delivery routes completed.
  - Prints the min and max distances.

3.6. File Handling Functions

- void saveToFile():
  - Opens routes.txt in write mode ("w").
  - Writes cityCount, all city names, and the entire distances matrix to the file.
  - Opens deliveries.txt in write mode.
  - Writes deliveryCount and then loops through all completed deliveries, saving all their details.
  - Prints a success message.

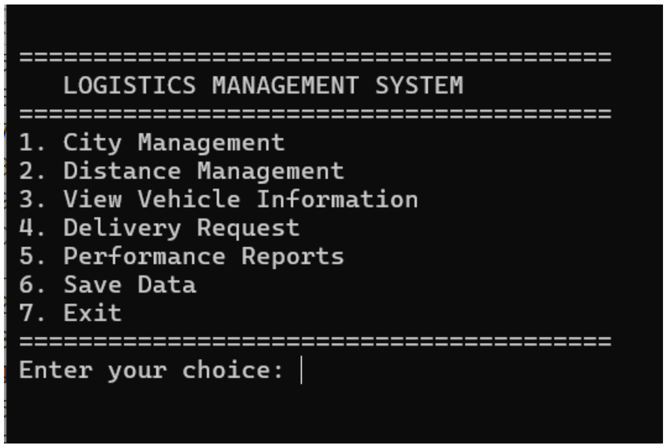
- void loadFromFile():
  - Attempts to open routes.txt in read mode ("r").
  - If the file exists, it reads cityCount, the city names, and the distances matrix from the file into the global arrays.
  - Attempts to open deliveries.txt in read mode.
  - If the file exists, it reads deliveryCount and then uses scanf with a specific format string to parse the pipe-delimited data for each delivery back into the delivery arrays.
  - Prints success messages if files are loaded.

4. Program Workflow and Screenshots

This section provides a walkthrough of the program's functionality with screenshots.

4.1. Main Menu

When the program starts, it loads existing data and displays the main menu, which acts as the central navigation hub.



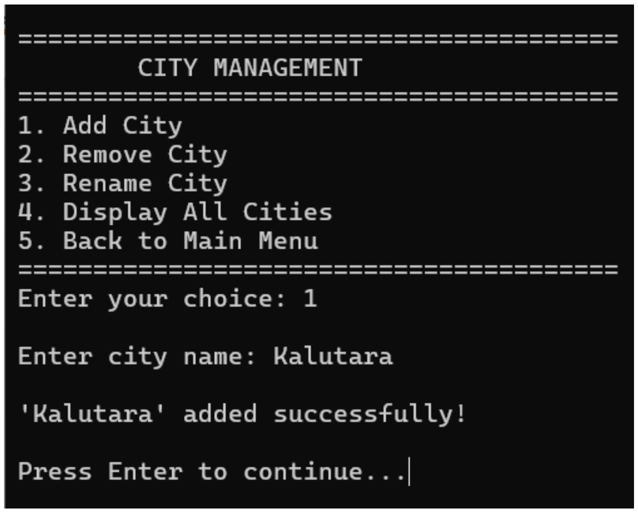
```
========================================
   LOGISTICS MANAGEMENT SYSTEM
========================================
1. City Management
2. Distance Management
3. View Vehicle Information
4. Delivery Request
5. Performance Reports
6. Save Data
7. Exit
========================================
Enter your choice: 
```

4.2. Route Management

Users can add cities and define the distances between them.

Adding a City:

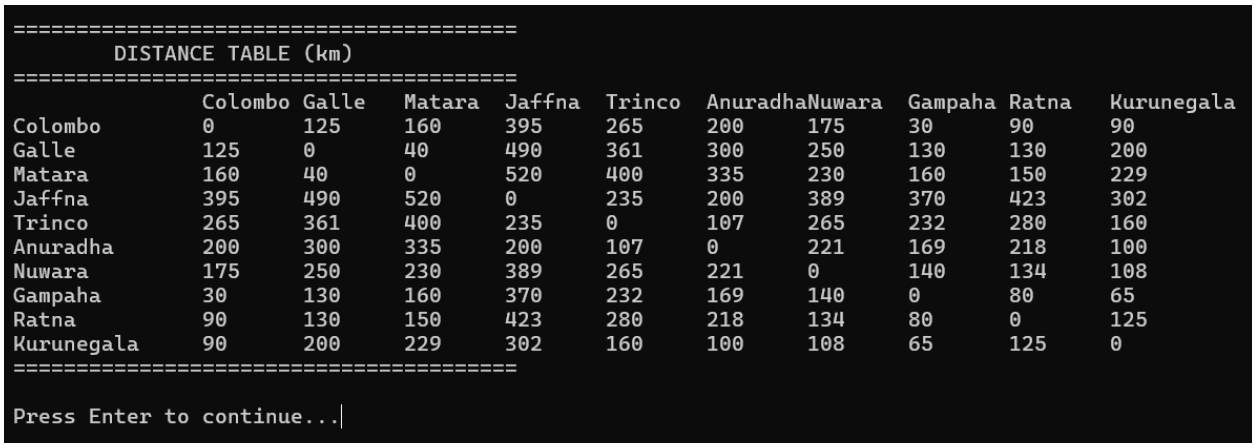
Selecting option 1 allows the user to add a new city to the system.



```
========================================
        CITY MANAGEMENT
========================================
1. Add City
2. Remove City
3. Rename City
4. Display All Cities
5. Back to Main Menu
========================================
Enter your choice: 1

Enter city name: Kalutara

'Kalutara' added successfully!

Press Enter to continue...
```

Viewing Distances:

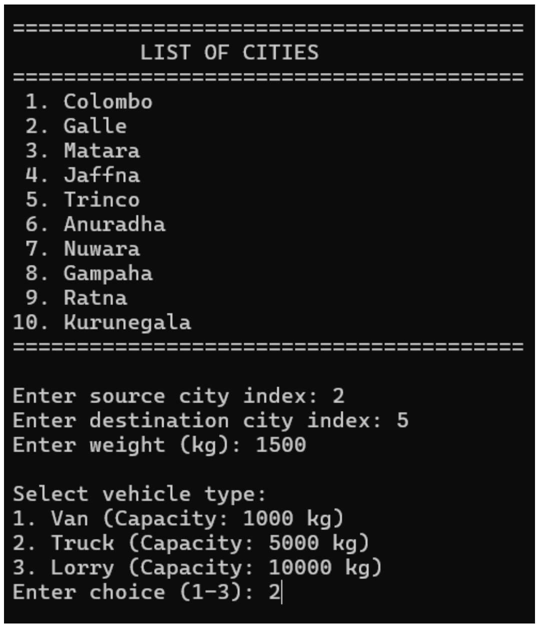
Option 2 displays the current distance matrix.



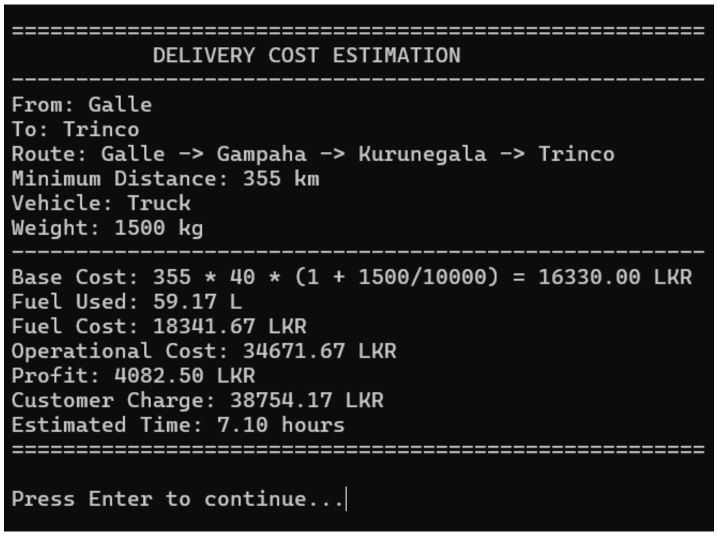
```
========================================
        DISTANCE TABLE (km)
========================================
               Colombo Galle   Matara  Jaffna  Trinco  AnuradhaNuwara  Gampaha Ratna   Kurunegala
Colombo        0       125     160     395     265     200     175     30      90      90
Galle          125     0       40      490     361     300     250     130     130     200
Matara         160     40      0       520     400     335     230     160     150     229
Jaffna         395     490     520     0       235     200     389     370     423     302
Trinco         265     361     400     235     0       107     265     232     280     160
Anuradha       200     300     335     200     107     0       221     169     218     100
Nuwara         175     250     230     389     265     221     0       140     134     108
Gampaha        30      130     160     370     232     169     140     0       80      65
Ratna          90      130     150     423     280     218     134     80      0       125
Kurunegala     90      200     229     302     160     100     108     65      125     0
========================================

Press Enter to continue...
```

4.3. Handling a Delivery and Finding Shortest Path

Option 4 is the core function for processing a new delivery. It asks for source, destination, and weight, then provides a full cost and time estimation.



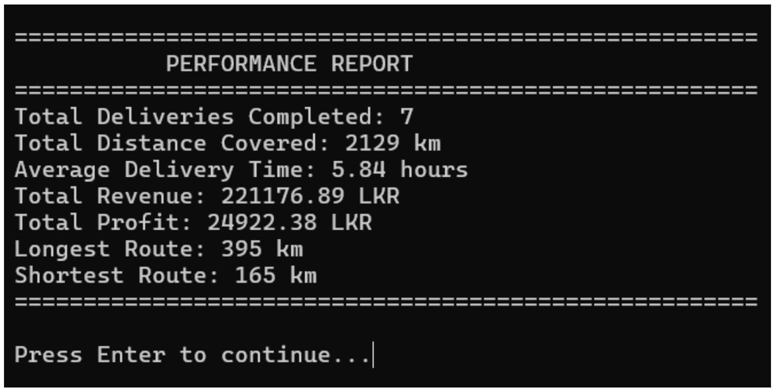
```
========================================
          LIST OF CITIES
========================================
 1. Colombo
 2. Galle
 3. Matara
 4. Jaffna
 5. Trinco
 6. Anuradha
 7. Nuwara
 8. Gampaha
 9. Ratna
10. Kurunegala
========================================

Enter source city index: 2
Enter destination city index: 5
Enter weight (kg): 1500

Select vehicle type:
1. Van (Capacity: 1000 kg)
2. Truck (Capacity: 5000 kg)
3. Lorry (Capacity: 10000 kg)
Enter choice (1-3): 2
```



```
======================================================
           DELIVERY COST ESTIMATION
------------------------------------------------------
From: Galle
To: Trinco
Route: Galle -> Gampaha -> Kurunegala -> Trinco
Minimum Distance: 355 km
Vehicle: Truck
Weight: 1500 kg
------------------------------------------------------
Base Cost: 355 * 40 * (1 + 1500/10000) = 16330.00 LKR
Fuel Used: 59.17 L
Fuel Cost: 18341.67 LKR
Operational Cost: 34671.67 LKR
Profit: 4082.50 LKR
Customer Charge: 38754.17 LKR
Estimated Time: 7.10 hours
======================================================

Press Enter to continue...
```

4.4. Generating Reports

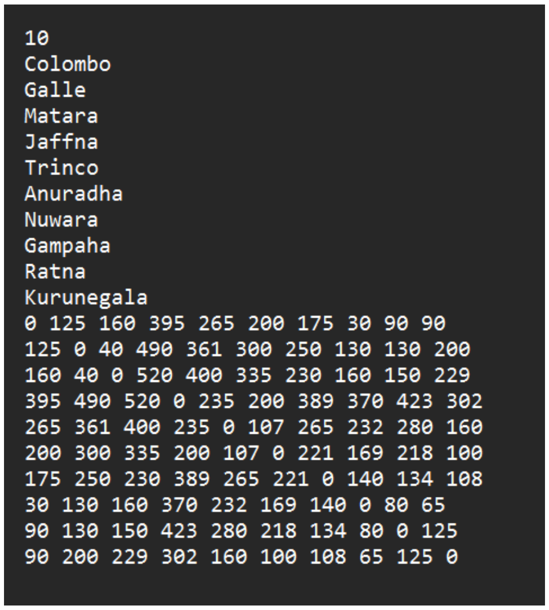
Option 5 leads to a sub-menu where the user can view various performance metrics.



```
======================================================
           PERFORMANCE REPORT
======================================================
Total Deliveries Completed: 7
Total Distance Covered: 2129 km
Average Delivery Time: 5.84 hours
Total Revenue: 221176.89 LKR
Total Profit: 24922.38 LKR
Longest Route: 395 km
Shortest Route: 165 km
======================================================

Press Enter to continue...
```

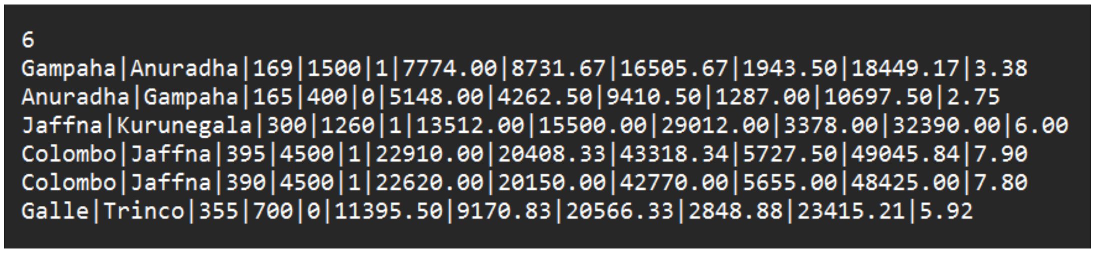
4.5. Data Persistence (File Content)

Data is saved to routes.txt and deliveries.txt upon exit.



```
10
Colombo
Galle
Matara
Jaffna
Trinco
Anuradha
Nuwara
Gampaha
Ratna
Kurunegala
0 125 160 395 265 200 175 30 90 90
125 0 40 490 361 300 250 130 130 200
160 40 0 520 400 335 230 160 150 229
395 490 520 0 235 200 389 370 423 302
265 361 400 235 0 107 265 232 280 160
200 300 335 200 107 0 221 169 218 100
175 250 230 389 265 221 0 140 134 108
30 130 160 370 232 169 140 0 80 65
90 130 150 423 280 218 134 80 0 125
90 200 229 302 160 100 108 65 125 0
```

routes.txt



```
6
Gampaha|Anuradha|169|1500|1|7774.00|8731.67|16505.67|1943.50|18449.17|3.38
Anuradha|Gampaha|165|400|0|5148.00|4262.50|9410.50|1287.00|10697.50|2.75
Jaffna|Kurunegala|300|1260|1|13512.00|15500.00|29012.00|3378.00|32390.00|6.00
Colombo|Jaffna|395|4500|1|22910.00|20408.33|43318.34|5727.50|49045.84|7.90
Colombo|Jaffna|390|4500|1|22620.00|20150.00|42770.00|5655.00|48425.00|7.80
Galle|Trinco|355|700|0|11395.50|9170.83|20566.33|2848.88|23415.21|5.92
```

deliveries.txt

5. Assumptions and Limitations

5.1. Assumptions

The following assumptions were made during the design and implementation of the system:

1. **Constant Fuel Price:** The fuel price is fixed at 310.0 LKR per liter and does not change during runtime.
2. **Symmetrical Distances:** The distance from City A to City B is assumed to be the same as the distance from City B to City A.
3. **Average Speeds:** Vehicle speeds are averages and do not account for traffic, weather, or road conditions.
4. **Fixed Profit Margin:** The profit for every delivery is calculated as a fixed 25% of the total operational cost.
5. **Positive Distances:** All distances entered are non-negative, as required for algorithm to function correctly.
6. **Vehicle Selection:** The smallest available vehicle that can accommodate the weight is always chosen.
7. **Direct Routes:** The system assumes the "distance" between two cities is the road distance and calculates fuel and time based on this single value.

5.2. Limitations

The system has the following limitations:

1. **Fixed Array Sizes:** The program uses fixed-size arrays. It cannot handle more than 30 cities or 50 deliveries per session.
2. **Static Data:** All vehicle information (capacity, speed, etc.) is hard-coded and cannot be changed by the user.
3. **No Real-Time Tracking:** The system only provides estimations and does not perform any real-time tracking of deliveries.
4. **Simple Error Handling:** Error handling is basic. For example, it does not robustly handle incorrect data types entered by the user (e.g., entering text instead of a number).

6. Conclusion

6.1. Summary

This project successfully achieved the objective of creating a functional logistics management system in C. The program effectively manages cities, distances, and delivery requests. It provides core logistics functionalities such as cost/time estimation, vehicle selection, and shortest path finding using exhaustive search algorithm. The system also includes reporting features and data persistence through file handling.

6.2. Learnings

This project provided practical experience in several key areas of C programming:

- **Data Management:** Using parallel arrays to manage related data for cities, vehicles, and deliveries.
- **Modular Programming:** Structuring a larger program into smaller, manageable functions.
- **Algorithm Implementation:** Implementing a complex algorithm like exhaustive search from scratch.
- **File I/O:** Reading from and writing to text files to persist data between sessions.
- **Software Design:** Planning the logic and data structures for a complete, menu-driven application.

6.3. Future Improvements

While functional, the system could be enhanced with several features:

- **Dynamic Data Structures:** Using dynamic memory allocation (malloc, realloc) or linked lists to remove the fixed limits on cities and deliveries.
- **Database Integration:** Storing data in an SQL database (like SQLite) instead of text files for more robust and scalable data management.
- **Advanced Features:** Adding features like user accounts, real-time delivery tracking (simulation), and inventory management.
- **Enhanced Error Handling:** Implementing more robust input validation to prevent crashes from invalid user input.