University of Sri Jayewardenepura - Faculty of Applied Sciences
BSc (General) Degree First Year
CSC 1012: Introduction to Computer Programming
Assignment (Individual)
Logistics Management System

Name: S S J Gunawardana

Index No: AS20240540

Project Report

1. Introduction

This project is a menu driven "Logistics Management System" created for CSC 1012: Introduction to Computer Programming course unit, following the assignment requirements. It's written in the C language and simulates a basic logistics and delivery management system.

The program allows a user to:

- Manage a list of cities.
- Set the direct road distances between cities.
- Handle customer delivery requests
- Automatically find the least cost route.
- Estimate delivery time and cost.
- Save all data to files (routes.txt, deliveries.txt) and load it automatically on startup.
- View summary reports of past deliveries.

To build this, I used core C programming concepts like 1D and 2D arrays, functions for modularity, loops, and conditional statements. I also focused on making the program robust by adding strong input validation (getSafeInput, fgets) to prevent crashes or errors if the user enters unexpected data. The program also includes file handling (saveData, loadData) to save and load all data, making it persistent between uses. User experience improvements like screen clearing (clearScreen) and pausing (pauseProgram) were also added. Furthermore, the system automatically records the date and time of each confirmed delivery for better record-keeping.

2. GitHub Repository

All the source code (Code::Blocks project folder containing the .c files) and the project's development history (commit messages) are available on GitHub, as required

- **Repository URL:** https://github.com/sobashss/logistics-management-system.git

3. System Design & Data Structures

The program uses several global variables defined at the top of main.c to keep track of all information:

- **Constants:** MAX_CITIES (30), NAME_LENGTH (60), FUEL_PRICE (310.0), MAX_DELIVERIES (50), and INF (999999) used for pathfinding calculations.
- **City Data:** char cityNames[MAX_CITIES][NAME_LENGTH] stores the names. int cityCount keeps track of how many cities have been added.
- **Distance Data:**
    - int distance[MAX_CITIES][MAX_CITIES]: Stores the *direct* road distance entered by the user. A value of -1 means the distance hasn't been set.
    - int leastDistance[MAX_CITIES][MAX_CITIES]: Stores the calculated *shortest possible path* between all cities, found using the Floyd-Warshall algorithm.
- **Vehicle Data:** Fixed information about the Van, Truck, and Lorry (capacity, rate per km, speed, fuel efficiency) is stored in parallel arrays (vehicleTypes, vehicleCapacity, vehicleSpeed) as specified.
- **Delivery Logbook:** A set of parallel arrays (deliverySource, deliveryDest, deliveryDistance, deliveryRevenue, deliveryDate, deliveryProfit, deliveryTime) stores the details of every completed delivery, used for generating reports. int deliveryCount tracks the number of completed deliveries.

4. Function Descriptions

The program is divided into functions, each performing a specific task, fulfilling the modular design requirement. Below is a description of every function implemented in main.c:

Main & Menu Functions

- int main(): The program's entry point. Initializes data by calling loadData(), then enters the main menu loop. Based on user input (validated by getSafeInput), it calls the appropriate sub-menu functions (manageCities, manageDistances, deliveryRequest, showReports). Calls saveData() before exiting on choice 5.
- void manageCities(): Clears the screen and displays the sub-menu for city management. Uses getSafeInput for menu navigation.
- void manageDistances(): Clears the screen and displays the sub-menu for distance management. Uses getSafeInput for menu navigation.

Input Validation & UI Helpers

- int getSafeInput(): A vital utility for robust input. Reads an integer using scanf. If scanf fails (non-numeric input), it prints an error, clears the invalid input from the buffer, and loops until a valid integer is entered. Returns the validated integer.
- void clearScreen(): Improves UI clarity by clearing the console window. Uses system("cls") on Windows and system("clear") on Linux/macOS. Called before displaying menus or major outputs.
- void pauseProgram(): Enhances user experience by pausing execution. Prints "Press Enter to continue..." and waits for the user to press Enter (using getchar()). Called after displaying messages, errors, or tables.

City Management

- void addCity(): Handles adding a new city. Prompts for the name and reads it safely using fgets (allowing spaces and preventing overflows). Checks if the city count limit (MAX_CITIES) is reached, if the name is empty, or if the name already exists. If valid, add the city to cityNames, increments cityCount, and initializes its distances in the distance matrix to -1 (except self-distance = 0).
- void listCities(): A simple helper function. Prints a numbered list (1-based index) of all currently stored cities. Used by other functions like renameCity, removeCity, and setDistance.
- void renameCity(): Allows renaming an existing city. Displays the city list, prompts for the index using getSafeInput, validates the index. Reads the new name using fgets, checks if it's empty or already used by another city. If valid, updates the name in the cityNames array.
- void removeCity(): Handles removing a city. Displays the list, gets the index using getSafeInput, validates it. Removes the city by shifting all subsequent elements in cityNames one position up. Crucially, it also shifts the corresponding rows and columns in the distance matrix to maintain data integrity. Decrements cityCount and calls leastDistanceRoute() to update paths.

Distance Management & Pathfinding

- void initializeDistances(): Sets up the initial distance matrix. Called by loadData only if routes.txt is not found. Set self-distances (e.g: distance[i][i]) to 0 and all other distances to -1 indicating they haven't been set by the user yet.
- void setDistance(): Allows the user to input or edit the *direct* distance between two cities. Prompts for source and destination indices using getSafeInput, validates them.

Prompts for the distance using getSafeInput, validates it (must be > 0). Stores the distance symmetrically (distance[i][j] = distance[j][i]). Calls leastDistanceRoute() afterwards, as changing a direct distance requires recalculating all shortest paths.

- void leastDistanceRoute(): Implements the **Floyd-Warshall algorithm** to calculate the shortest possible path between all pairs of cities. It first initializes the leastDistance matrix based on the current distance matrix (copying direct distances and using INF for unset paths). Then, using three nested loops (k, i, j), it checks if going via an intermediate city k provides a shorter route from i to j. Updates the leastDistance matrix accordingly. This function contains the core logic to find the least cost route.
- void displayDistanceTable(): Clears the screen and displays the *direct* distances (from the distance matrix) in a formatted table. It dynamically calculates the width of the first column based on the longest city name to ensure perfect alignment, regardless of name length. Prints "N/A" for unset distances (-1).

Delivery Request & Calculations

- int selectVehicle(): Clears the screen, displays details (capacity, rate, speed) for the Van, Truck, and Lorry. Uses a while loop and getSafeInput to ensure the user enters a valid choice (1, 2, or 3) and returns the corresponding 0-based index (0, 1, or 2).
- void deliveryRequest(): The main function for handling a new delivery.
  I. Clears screen, checks if MAX_DELIVERIES limit reached or if less than 2 cities exist.
  II. Displays city list, gets source/destination indices using getSafeInput and validates them.
  III. Calls selectVehicle() to get the vehicle choice.
  IV. Gets delivery weight using getSafeInput and validates it against vehicle capacity.
  V. Calls leastDistanceRoute() to ensure shortest paths are up-to-date.
  VI. Retrieves the shortest distance dist from the leastDistance matrix. Check if a route exists (dist != INF).
  VII. Calls the calculation functions (deliveryCost, estimatedDeliveryTime, etc.) with the retrieved distance and other parameters.
  VIII. Displays the formatted quote, matching the "Sample Output".
  IX. Enters a loop prompting the user to confirm ('y'/'n'). Uses scanf to read the character safely.
  X. If 'y'/'Y', records the current date & time using <time.h> functions (time(), localtime(), strftime()) formatted as 'YYYY-MM-DD_HH:MM' and saves all delivery details

(sourceIdx, destIdx, dist, dtime, charge, prof, deliveryDate) into the global logbook arrays. Increment deliveryCount.

XI. If 'n'/'N', prints a cancellation message.

XII. If invalid input, print an error and loops again.

- float deliveryCost(float distance, float rate, float weight): Calculates the base delivery cost using the formula: distance * rate * (1 + weight / 10000.0). Returns the result.
- float estimatedDeliveryTime(float distance, float speed): Calculates estimated time using the formula: distance / speed. Returns the result in hours.
- float fuelConsumption(float distance, float efficiency): Calculates fuel needed using the formula: distance / efficiency. Returns the result in liters.
- float fuelCost(float consumption): Calculates the cost of fuel using the formula: consumption * FUEL_PRICE. Returns the result in LKR.
- float totalOperationalCost(float deliveryCost, float fuelCost): Calculates the total operational cost by summing deliveryCost + fuelCost. Returns the result.
- float profit(float deliveryCost): Calculates the profit margin using the formula: deliveryCost * 0.25 (25%). Returns the result.
- float finalCharge(float totalCost, float profit): Calculates the final charge to the customer by summing totalCost + profit. Returns the result.

Reporting & Data Persistence

- void showReports(): Clear the screen. Check if deliveryCount is 0. If deliveries exist, it iterates through the logbook arrays (deliveryDistance, deliveryTime, etc.), calculating totals (distance, time, revenue, profit) and finding extremes (longest/shortest routes). Finally, display these statistics in a formatted summary table.
- void saveData(): Opens **routes.txt** and deliveries.txt in write mode ("w"). Writes cityCount, all city names, and the distance matrix to routes.txt. Writes deliveryCount and all recorded delivery details (source, dest, distance, time, revenue, profit and date) to **deliveries.txt**. Called just before exiting main.
- void loadData(): Called once at the start of main. Attempts to open **routes.txt** and **deliveries.txt** in read mode ("r").
  - If routes.txt doesn't exist, calls initializeDistances().
  - If it exists, reads cityCount, then uses fgets in a loop to read city names (handling spaces correctly), then reads the distance matrix using fscanf.

- If deliveries.txt exists, reads deliveryCount and then uses fscanf in a loop to read all the stored delivery records (including the date string) back into the logbook arrays.
- Prints status messages. Finally, calls leastDistanceRoute() to calculate shortest paths based on the loaded distances.

5. Assumptions

- **File Integrity:** The program assumes the user does not manually edit routes.txt or deliveries.txt in a way that breaks the expected format (e.g: adding letters where numbers should be). Basic checks are included (like checking if files open successfully and reading counts correctly).
- **User Indexing:** The program shows users lists starting from 1 (e.g: "1. Colombo") but converts these to 0-based indices internally when accessing arrays.
- **Date Format:** The date and time are saved with an underscore (_) separating the date and time (YYYY-MM-DD_HH:MM) to work correctly with the fscanf used in loadData for the delivery records.

6. Conclusion

This project successfully demonstrates the use of C programming fundamentals to create a functional Logistics Management System. It meets all the requirements outlined in the assignment brief, including managing cities and distances, calculating delivery costs based on the shortest route (found using Floyd-Warshall), generating reports, and saving/loading data persistently. The addition of robust input validation (getSafeInput, fgets) and UI helpers (clearScreen, pauseProgram) makes the program stable and user-friendly.