**CSC 1012 Introduction to Computer Programming**

**Assignment (Individual)**

**Logistics Management System**

Name: M.A.Binuri Vihangi

Index No: AS20240449

Registration No:122069

****

Table of Contents

[01 INTRODUCTION 3](#_Toc212323682)

[02.OBJECTIVES 5](#_Toc212323683)

[03.SYSTEM OVERVIEW 6](#_Toc212323684)

[04.DEVELOPMENT PROCESS 7](#_Toc212323685)

[05.CONSTANT & VARIABLES 8](#_Toc212323686)

[CONSTANT TABLE 8](#_Toc212323687)

[VARIABLES TABLE 8](#_Toc212323688)

[06.TEXT FILES 10](#_Toc212323689)

[07.FULL CODE 13](#_Toc212323690)

[08.FUNCTIONS & SUB FUNCTIONS 46](#_Toc212323691)

[main() Function 46](#_Toc212323692)

[cityManagement() Function 48](#_Toc212323693)

[SUB FUNCTION 50](#_Toc212323694)

[addCity() FUNCTION 50](#_Toc212323695)

[RenameCity() FUNCTION 51](#_Toc212323696)

[removeCity() FUNCTION 52](#_Toc212323697)

[viewCities() Function 53](#_Toc212323698)

[loadCities() & saveCities() 54](#_Toc212323699)

[distanceManagement() FUNCTION 55](#_Toc212323700)

[Sub FUNCTION 61](#_Toc212323701)

[loadDistances() FUNCTION 61](#_Toc212323702)

[SaveDistances() FUNCTION 62](#_Toc212323703)

[vehicleInfo() FUNCTION 62](#_Toc212323704)

[deliveryRequest() FUNCTION 62](#_Toc212323705)

[Sub FUNCTION 67](#_Toc212323706)

[saveDeliveries 67](#_Toc212323707)

[viewDeliveries() FUNCTION 67](#_Toc212323708)

[Sub FUNction 69](#_Toc212323709)

[loadDeliveries() 69](#_Toc212323710)

[routeFinder() FUNCTION 70](#_Toc212323711)

[reports FUNCTION 73](#_Toc212323712)

[help() FUNCTION 76](#_Toc212323713)

[exitProgram() FUNCTION 77](#_Toc212323714)

[09.GitHub repository link 80](#_Toc212323715)

# 01 INTRODUCTION

The Logistics Management System is a console based C program designed to manage transportation and delivery operations between cities. It allows users to add cities, manage distances, handle delivery requests, view delivery records, find the least-cost route, generate reports, and manage vehicle details efficiently.

This system simulates a basic logistics company operation including vehicle management, cost estimation, fuel calculation, and route optimization using simple text files for data storage.

The system's core functionality is built upon an integrated workflow that transforms user input into actionable logistical data. It begins with the foundational setup of a city network and its corresponding distance matrix. This data then serves as the basis for all subsequent operations, from generating accurate delivery quotations that factor in vehicle-specific rates and fuel costs, to calculating the shortest possible path between two locations for optimal route planning. This structured approach ensures reliable performance and provides users with a comprehensive toolset for end-to-end delivery management.

# 02.OBJECTIVES

* Manage cities and distances between them
* Handle customer delivery requests
* Estimate delivery time and cost
* Track completed deliveries and generate performance reports
* Vehicle management
* To provide a Help section
* To store all data persistently using .txt files
* To apply modular programming using functions and structures in C

# 03.SYSTEM OVERVIEW

The program provides a menu driven interface that allows users to choose from the following main modules:

1. City Management – Add, rename, remove, and view cities.
2. Distance Management – Record and view distances between cities.
3. Vehicle Information – Display vehicle types with capacity, rates, speed, and fuel efficiency.
4. Delivery Request – Calculate delivery cost, time, and profit.
5. View Deliveries – View all saved deliveries.
6. Route Finder – Find the least-cost route between cities.
7. Reports – Generate delivery statistics (revenue, profit, etc.)
8. Help-Contact informations
9. Exit -Exit the programme

# 04.DEVELOPMENT PROCESS

The system was developed using:

|  |  |
| --- | --- |
| Component | Description |
| Language | C |
| IDE Used | Code::Blocks |
| File Storage | Text files (.txt) |
| Programming Concepts | Arrays,structures,file handling,loops,switch case,modular programming |

# 05.CONSTANT & VARIABLES

## CONSTANT TABLE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Constant name | Value/Definition | Type | Description/Purpose | Where used |
| MAX-CITIES | 30 | Int | Maximum number of cities that can be stored in the system. | Used in city and distance management |
| CITY-N-LENTH | 50 | Int | Maximum length of each city name. | Used in city arrays |
| MAX-DELIVERIES | 50 | Int | Maximum number of delivery records allowed. | Used in delivery management |
| CITY-FILE | “cities.txt” | String | File used to store all city names. | Used in loadCities() and saveCities() |
| DIST-FILE | “distances.txt” | String | File used to store distance matrix between cities. | Used in loadDistances() and saveDistances() |
| DELIVERY-FILE | “deliveries.txt” | String | File used to store all delivery records. | Used in saveDelivery() and loadDelivery() |
| ROUTE-FILE | “least-cost-route.txt” | String | File used to store shortest or least cost route details | Used in routeFinder() |

## VARIABLES TABLE

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Type | Scope | Description/Purpose |
| vehicle-names | Char[3][10] | Global | Store available vehicle names (van,truck,lorry) |
| capacity | Int[3] | Global | Store vehicle weight capacities(in kg) |
| rate-per-km | Int[3] | Global | Stores per kilometer rate for each veicle type |
| speed | Int[3] | Global | Stores average speed(km/h)for each vehicle type. |
| fuel-efficiency | Int[3] | Global | Stores fuel efficiency(km per litre) of each vehicle type |
| fuel-price | Float | Global | Stores the current price per lireof fuel |
| struct Delivery | Struct | Gloal | Structure to represent delivery details(source,destination,vehicle,weight,cost,time) |
| choice | Int | Local(various functions) | Stores user’s menu selections in menus |
| cities | Char[MAX-CITIes][CITY-N-LENTH] | Local(city,distance,delivery functions) | Stores list of city names loaded from cities.txt |
| count | Int | Local | Keep tracks of number of cities or deliveries. |
| distance | Int[MAX-CITIES][MAX-CITIES] | Local | Distance matrix storing distances between cities |
| src,dest | Int | Local | Stores source and destination city indices for deliveries or route finding |
| vType | Int | Local | Stores user selected vehicle type |
| W | Float | Local | Stores delivery weight in kilograms |
| R,S,E | Float | Local | Temporary variable for rate,speed,efficiency,for cost calculations. |
| deliveryCost,fuelUsed,fuelCost,totalCost,profit,customerCharge | Float | Local(in deliveryRequest) | Used for calculating delivery and customer costs |
| timeHrs | Float | Local | Estimate travel time for delivery |
| Path[MAX-CITIES],cost[MAX-CITIES],visited[MAX-CITIES],prev[MAX-CITIES] | Int arrays | Local(in routeFinder) | Used in algorithm for finding least cost route |
| fp | FILE\* | Local | File pointer foe file handling |
| d | Struct Delivery |  | Temporary deliver record structure instance used for saving deliveries |

# 06.TEXT FILES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| File Name | Data Stored | Type of Data | Accessed By | Used in Functions | Purpose |
| cities.txt | City names | Text | Automatically created if not found | loadCities(),saveCities(),addCity(),removeCity() | Stores all added cities permanently |
| distances.txt | Distance matrix | Numeric text | Automatically created if not found | loadDistances(),saveDistance() | Stores distance (in km) |
| deliveries.txt | Delivery records | text | Automatically created on first save | saveDelivery(),loadDeliveries() | Stores all delivery transactions |
| least-cost-route.txt | Route details | Text | Created by program | routeFinder() | Saves shortest/least cost routes found by the user |

cities.txt

distances.txt

deliveries.txt

least\_cost\_route.txt

# 07.FULL CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_CITIES 30

#define CITY\_N\_LENTH 50

#define MAX\_DELIVERIES 50

#define CITY\_FILE "cities.txt"

#define DIST\_FILE "distances.txt"

#define DELIVERY\_FILE "deliveries.txt"

#define ROUTE\_FILE "least\_cost\_route.txt"

char vehicle\_names[3][10] = {"Van", "Truck", "Lorry"};

int capacity[3] = {1000, 5000, 10000};

int rate\_per\_km[3] = {30, 40, 80};

int speed[3] = {60, 50, 45};

int fuel\_efficiency[3] = {12, 6, 4};

float fuel\_price = 310.0;

struct Delivery {

char source[50];

char destination[50];

char vehicle[10];

float weight;

float cost;

float time;

};

//function declaration

void cityManagement();

void addCity(char cities[][CITY\_N\_LENTH],int\*count);

void renameCity(char cities[][CITY\_N\_LENTH], int count);

void removeCity(char cities[][CITY\_N\_LENTH], int \*count);

void viewCities(char cities[][CITY\_N\_LENTH], int count);

void loadCities(char cities[][CITY\_N\_LENTH], int \*count);

void saveCities(char cities[][CITY\_N\_LENTH], int count);

void distanceManagement();

void vehicleInfo();

void deliveryRequest();

void viewDeliveries();

void routeFinder();

void reports();

void help();

void save();

void exitProgram();

void loadDistances(int distance[MAX\_CITIES][MAX\_CITIES],int cityCount);

void saveDistances(int distance[MAX\_CITIES][MAX\_CITIES],int cityCount);

void saveDelivery(struct Delivery d);

void loadDeliveries(struct Delivery deliveries[],int\*count);

int main()

{

int choice;

printf("==============================\n");

printf("LOGISTICS MANAGEMENT SYSTEM\n");

printf("==============================\n");

printf("1. City Management\n");

printf("2. Distance Management\n");

printf("3. Vehicle Information\n");

printf("4. Delivery Request\n");

printf("5. View All Deliveries\n");

printf("6. Route Finder\n");

printf("7. Reports\n");

printf("8. Help\n");

printf("9. Exit\n");

printf("Enter your Choice (1-9):\n");

scanf("%d",&choice);

switch(choice){

case 1:

cityManagement();

break;

case 2:

distanceManagement();

break;

case 3:

vehicleInfo();

break;

case 4:

deliveryRequest();

break;

case 5:

viewDeliveries();

break;

case 6:

routeFinder();

break;

case 7:

reports();

break;

case 8: help();

break;

case 9:

exitProgram();

return 0;

break;

default:

printf("Invalid choice!\n");

}

return 0;

}

void cityManagement() {

int choice;

int count=0;

char cities[MAX\_CITIES][CITY\_N\_LENTH];

loadCities(cities, &count);

printf("\nCITY MANAGEMENT\n");

printf("\n==========================\n");

printf("1. Add City\n");

printf("2. Rename City\n");

printf("3. Remove City\n");

printf("4. View All Cities\n");

printf("5. Return to Main Menu\n");

printf("6. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addCity(cities, &count);

break;

case 2:

renameCity(cities, count);

break;

case 3:

removeCity(cities, &count);

break;

case 4:

viewCities(cities, count);

break;

case 5:

saveCities(cities, count);

printf("Returning to main menu...\n");

main();

break;

case 6:

exitProgram();

break;

default:

printf("Invalid choice! Try again.\n");

}

}

void distanceManagement(){

int distance[MAX\_CITIES][MAX\_CITIES];

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int count = 0;

int choice;

memset(distance,0,sizeof(distance));

loadCities(cities, &count);

if (count < 2) {

printf("At least two cities are needed!\n");

return;

}

loadDistances(distance, count);

printf(" DISTANCE MANAGEMENT \n");

printf("=============================\n");

printf("1. Input or Edit Distance\n");

printf("2. View Distance Table\n");

printf("3. Return to Main Menu\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

int i, j, d;

printf("\nAvailable Cities:\n");

for (int k = 0; k < count; k++) {

printf("%d. %s\n", k + 1, cities[k]);

}

printf("Enter source city number: ");

scanf("%d", &i);

printf("Enter destination city number: ");

scanf("%d", &j);

if (i < 1 || i > count || j < 1 || j > count) {

printf("Invalid city numbers!\n");

break;

}

if (i == j) {

printf("Distance from a city to itself is 0 km.\n");

distance[i - 1][j - 1] = 0;

saveDistances(distance,count);

break;

}

printf("Enter distance between %s and %s (in km): ", cities[i - 1], cities[j - 1]);

scanf("%d", &d);

distance[i - 1][j - 1] = d;

distance[j - 1][i - 1] = d;

saveDistances(distance, count);

printf("Distance added successfully and saved!\n");

break;

}

case 2: {

printf("\n%-20s", " ");

for (int i = 0; i < count; i++) {

printf("%-15s", cities[i]);

}

printf("\n");

for (int i = 0; i < count; i++) {

printf("%-20s", cities[i]);

for (int j = 0; j < count; j++) {

printf("%-15d", distance[i][j]);

}

printf("\n");

}

break;

}

case 3:

printf("Returning to main menu...\n");

main();

return;

case 4:

exitProgram();

return;

default:

printf("Invalid choice! Try again.\n");

}

}

void vehicleInfo(){

int choice;

while (1) {

printf("\n===== VEHICLE INFORMATION =====\n");

printf("1. View all vehicles\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\n=== VEHICLES ===\n");

printf("%-10s %-15s %-20s %-20s %-20s\n", "Type", "Capacity(kg)", "Rate per km(LKR)", "Avg Speed(km/h)", "Fuel Efficiency(km/l)");

for (int i = 0; i < 3; i++) {

printf("%-10s %-15d %-20d %-20d %-20d\n",vehicle\_names[i],capacity[i],rate\_per\_km[i],speed[i],fuel\_efficiency[i]);

}

break;

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

}

void deliveryRequest(){

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int distance[MAX\_CITIES][MAX\_CITIES]={0};

int cityCount = 0;

loadCities(cities, &cityCount);

loadDistances(distance, cityCount);

int choice;

printf("\n===== DELIVERY REQUEST HANDLING =====\n");

printf("1. New Delivery\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

int src, dest, vType;

float W;

printf("\nAvailable Cities:\n");

for (int i = 0; i < cityCount; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

printf("\nEnter Source City Number: ");

scanf("%d", &src);

printf("Enter Destination City Number: ");

scanf("%d", &dest);

if (src < 1 || src > cityCount || dest < 1 || dest > cityCount) {

printf("Invalid city numbers!\n");

break;

}

if (src == dest) {

printf("Source and destination cannot be the same!\n");

break;

}

printf("\nSelect Vehicle Type:\n1. Van\n2. Truck\n3. Lorry\nEnter type: ");

scanf("%d", &vType);

if (vType < 1 || vType > 3) {

printf("Invalid vehicle type!\n");

break;

}

printf("Enter Weight (in kg): ");

scanf("%f", &W);

if (W > capacity[vType - 1]) {

printf("Weight exceeds capacity of %s (%d kg)!\n",

vehicle\_names[vType - 1], capacity[vType - 1]);

break;

}

int D = distance[src - 1][dest - 1];

if (D <= 0) {

printf("No valid distance found between %s and %s.\n",

cities[src - 1], cities[dest - 1]);

break;

}

float R = rate\_per\_km[vType - 1];

float S = speed[vType - 1];

float E = fuel\_efficiency[vType - 1];

float fuel\_price = 310;

float deliveryCost = D \* R \* (1 + (W/10000));

float fuelUsed = D / E;

float fuelCost = fuelUsed \* fuel\_price;

float totalCost = deliveryCost + fuelCost;

float profit = deliveryCost \* 0.25;

float customerCharge = totalCost + profit;

float timeHrs =D / S;

printf("\n======================================================\n");

printf("DELIVERY COST ESTIMATION\n");

printf("------------------------------------------------------\n");

printf("From: %s\n", cities[src - 1]);

printf("To: %s\n", cities[dest - 1]);

printf("Minimum Distance: %d km\n", D);

printf("Vehicle: %s\n", vehicle\_names[vType - 1]);

printf("Weight: %.0f kg\n", W);

printf("------------------------------------------------------\n");

printf("Base Cost: %d \* %.0f \* (1 + %.0f/10000) = %.2f LKR\n",

D, R, W, deliveryCost);

printf("Fuel Used: %.2f L\n", fuelUsed);

printf("Fuel Cost: %.2f LKR\n", fuelCost);

printf("Operational Cost: %.2f LKR\n", totalCost);

printf("Profit: %.2f LKR\n", profit);

printf("Customer Charge: %.2f LKR\n", customerCharge);

printf("Estimated Time: %.2f hours\n", timeHrs);

printf("======================================================\n");

struct Delivery d;

strcpy(d.source, cities[src - 1]);

strcpy(d.destination, cities[dest - 1]);

strcpy(d.vehicle, vehicle\_names[vType - 1]);

d.weight = W;

d.cost = customerCharge;

d.time = timeHrs;

saveDelivery(d);

printf("Delivery saved successfully!\n");

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

void viewDeliveries() {

struct Delivery deliveries[MAX\_DELIVERIES];

int count = 0;

loadDeliveries(deliveries, &count);

int choice;

printf("\n===== VIEW ALL DELIVERIES =====\n");

printf("1. View all deliveries\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (count == 0) {

printf("No deliveries recorded yet!\n");

return;

}

printf("\n%-5s %-15s %-15s %-10s %-10s %-10s\n",

"No", "Source", "Destination", "Vehicle", "Weight", "Cost(LKR)");

printf("-------------------------------------------------------------\n");

for (int i = 0; i < count; i++) {

printf("%-5d %-15s %-15s %-10s %-10.1f %-10.2f\n",

i + 1,

deliveries[i].source,

deliveries[i].destination,

deliveries[i].vehicle,

deliveries[i].weight,

deliveries[i].cost);

}

printf("-------------------------------------------------------------\n");

break;

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

void routeFinder() {

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int distance[MAX\_CITIES][MAX\_CITIES];

int cityCount = 0;

loadCities(cities, &cityCount);

loadDistances(distance, cityCount);

if (cityCount < 2) {

printf("Not enough cities to find a route!\n");

return;

}

int choice;

printf("\n===== LEAST-COST ROUTE FINDER =====\n");

printf("1. Find Least-Cost Route\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

printf("\nAvailable Cities:\n");

for (int i = 0; i < cityCount; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

int src, dest;

printf("\nEnter Source City Number: ");

scanf("%d", &src);

printf("Enter Destination City Number: ");

scanf("%d", &dest);

if (src < 1 || src > cityCount || dest < 1 || dest > cityCount) {

printf("Invalid city numbers!\n");

break;

}

src--;

dest--;

int cost[MAX\_CITIES], visited[MAX\_CITIES], prev[MAX\_CITIES];

for (int i = 0; i < cityCount; i++) {

cost[i] = 999999;

visited[i] = 0;

prev[i] = -1;

}

cost[src] = 0;

for (int i = 0; i < cityCount - 1; i++) {

int min = 999999, u = -1;

for (int j = 0; j < cityCount; j++) {

if (!visited[j] && cost[j] < min) {

min = cost[j];

u = j;

}

}

if (u == -1) break;

visited[u] = 1;

for (int v = 0; v < cityCount; v++) {

if (distance[u][v] > 0 && !visited[v] &&

cost[u] + distance[u][v] < cost[v]) {

cost[v] = cost[u] + distance[u][v];

prev[v] = u;

}

}

}

if (cost[dest] == 999999) {

printf("\nNo path found between %s and %s.\n", cities[src], cities[dest]);

break;

}

int path[MAX\_CITIES], count = 0;

for (int v = dest; v != -1; v = prev[v]) {

path[count++] = v;

}

printf("\nShortest route from %s to %s:\n", cities[src], cities[dest]);

for (int i = count - 1; i >= 0; i--) {

printf("%s", cities[path[i]]);

if (i > 0) printf(" -> ");

}

printf("\nTotal Distance: %d km\n", cost[dest]);

FILE \*fp = fopen("least\_cost\_route.txt", "a");

if (fp != NULL) {

fprintf(fp, "From %s to %s : ", cities[src], cities[dest]);

for (int i = count - 1; i >= 0; i--) {

fprintf(fp, "%s", cities[path[i]]);

if (i > 0) fprintf(fp, " -> ");

}

fprintf(fp, " | Distance: %d km\n", cost[dest]);

fclose(fp);

printf("Route saved to 'least\_cost\_route.txt' successfully!\n");

} else {

printf("Error saving route to file!\n");

}

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("\nInvalid choice! Please enter 1–3.\n");

}

}

void reports() {

struct Delivery deliveries[MAX\_DELIVERIES];

int count = 0;

loadDeliveries(deliveries, &count);

int choice;

printf("\n===== REPORTS =====\n");

printf("1. View Reports\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

if (count == 0) {

printf("\nNo delivery records found!\n");

break;

}

float totalDistance = 0, totalTime = 0, totalRevenue = 0;

float longest = 0, shortest = 999999;

for (int i = 0; i < count; i++) {

float distance = deliveries[i].cost / 100;

totalDistance += distance;

totalTime += deliveries[i].time;

totalRevenue += deliveries[i].cost;

if (distance > longest)

longest = distance;

if (distance < shortest)

shortest = distance;

}

float avgTime = totalTime / count;

float profit = totalRevenue \* 0.25;

printf("\n========== DELIVERY REPORT ==========\n");

printf("Total Deliveries Completed : %d\n", count);

printf("Total Distance Covered : %.2f km\n", totalDistance);

printf("Average Delivery Time : %.2f hours\n", avgTime);

printf("Total Revenue : %.2f LKR\n", totalRevenue);

printf("Total Profit (25%%) : %.2f LKR\n", profit);

printf("Longest Route : %.2f km\n", longest);

printf("Shortest Route : %.2f km\n", shortest);

printf("====================================\n");

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

void help(){

int choice;

printf("\n======= Help & Support =======\n");

printf("If you experience any issues or need assistance,\n");

printf("please contact our support team:\n\n");

printf("Email : support@flashlogistics.com\n");

printf("Phone : +94 77 123 4567\n");

printf("Address : Flash Logistics Management (Pvt) Ltd,\n");

printf(" No. 25, Wijerama,\n");

printf(" Colombo, Sri Lanka.\n");

printf("==============================\n");

printf("1. Return to Main Menu\n");

printf("2. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nReturning to main menu...\n");

main();

return;

case 2:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

void exitProgram(){

char choice;

printf("\nAre you sure you want to exit? (Y/N): ");

scanf(" %c", &choice);

if (choice == 'Y' || choice == 'y')

{

printf("\nExiting program...\n");

exit(0);

}

else if (choice == 'N' || choice == 'n')

{

printf("\nReturning to main menu...\n");

main();

}

else

{

printf("Invalid input! Please enter Y or N.\n");

}

}

//cityManagement function's sub function

void loadCities(char cities[][CITY\_N\_LENTH],int \*count){

FILE \*fp = fopen(CITY\_FILE,"r");

if (fp==NULL)return;

while (fgets(cities[\*count], CITY\_N\_LENTH, fp)) {

cities[\*count][strcspn(cities[\*count], "\n")] = '\0';

(\*count)++;

}

fclose(fp);

}

void saveCities(char cities[][CITY\_N\_LENTH],int count){

FILE \*fp = fopen(CITY\_FILE, "w");

if (fp == NULL) {

printf("SAVING ERROR!\n");

return;

}

for (int i = 0; i < count; i++) {

fprintf(fp, "%s\n", cities[i]);

}

fclose(fp);

}

void addCity(char cities[][CITY\_N\_LENTH], int \*count) {

if (\*count >= MAX\_CITIES) {

printf("LIMIT EXCEEDED (MAX %d)!\n", MAX\_CITIES);

return;

}

printf("Enter new city name: ");

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

if (fgets(cities[\*count], CITY\_N\_LENTH, stdin) == NULL) {

printf("Input error!\n");

return;

}

cities[\*count][strcspn(cities[\*count], "\n")] = '\0';

(\*count)++;

saveCities(cities, \*count);

printf("City added successfully!\n");

}

void renameCity(char cities[][CITY\_N\_LENTH],int count){

if (count == 0) {

printf("No cities available!.\n");

return;

}

viewCities(cities, count);

int index;

printf("Enter city number to rename: ");

scanf("%d", &index);

if (index < 1 || index > count) {

printf("Invalid Index!\n");

return;

}

printf("Enter new city name: ");

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

fgets(cities[index - 1], CITY\_N\_LENTH, stdin);

cities[index - 1][strcspn(cities[index - 1], "\n")] = '\0';

saveCities(cities, count);

printf("City renamed successfully!\n");

}

void removeCity(char cities[][CITY\_N\_LENTH],int \*count){

if (\*count == 0) {

printf("No cities available!.\n");

return;

}

viewCities(cities,\*count);

int index;

printf("Enter city number to remove: ");

scanf("%d", &index);

if (index < 1 || index > \*count) {

printf("Invalid number!\n");

return;

}

for (int i = index - 1; i < \*count - 1; i++) {

strcpy(cities[i], cities[i + 1]);

}

(\*count)--;

saveCities(cities, \*count);

printf("City removed successfully!\n");

}

void viewCities(char cities[][CITY\_N\_LENTH],int count){

if (count == 0) {

printf("No cities available!.\n");

return;

}

printf("\n====List of Cities====\n");

for (int i = 0; i < count; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

}

// Distance management sub functions

void loadDistances(int distance[MAX\_CITIES][MAX\_CITIES], int cityCount) {

FILE \*fp = fopen(DIST\_FILE, "r");

if (fp == NULL) {

for (int i = 0; i < cityCount; i++)

for (int j = 0; j < cityCount; j++)

distance[i][j] = 0;

return;

}

for (int i = 0; i < cityCount; i++) {

for (int j = 0; j < cityCount; j++) {

if (fscanf(fp, "%d", &distance[i][j]) != 1) {

distance[i][j] = 0;

}

}

}

fclose(fp);

}

void saveDistances(int distance[MAX\_CITIES][MAX\_CITIES], int cityCount) {

FILE \*fp = fopen(DIST\_FILE, "w");

if (fp == NULL) {

printf("Error saving distances!\n");

return;

}

for (int i = 0; i < cityCount; i++) {

for (int j = 0; j < cityCount; j++) {

fprintf(fp, "%d ", distance[i][j]);

}

fprintf(fp, "\n");

}

fclose(fp);

}

// delivery request function"s sub function

void saveDelivery(struct Delivery d) {

FILE \*fp = fopen(DELIVERY\_FILE, "a");

if (fp == NULL) {

printf("Error saving delivery!\n");

return;

}

fprintf(fp, "%s %s %s %.2f %.2f %.2f\n",

d.source, d.destination, d.vehicle, d.weight, d.cost, d.time);

fclose(fp);

}

//view delivery function's sub function

void loadDeliveries(struct Delivery deliveries[], int \*count) {

FILE \*fp = fopen(DELIVERY\_FILE, "r");

if (fp == NULL) {

\*count = 0;

return;

}

while (fscanf(fp, "%s %s %s %f %f %f",

deliveries[\*count].source,

deliveries[\*count].destination,

deliveries[\*count].vehicle,

&deliveries[\*count].weight,

&deliveries[\*count].cost,

&deliveries[\*count].time) == 6) {

(\*count)++;

if (\*count >= MAX\_DELIVERIES)

break;

}

fclose(fp);

}

# 08.FUNCTIONS & SUB FUNCTIONS

## main() Function

* Display the main menu of the system.
* Based on user input,calls the respective module function
* Uses switch case for menu selection

int main()

{

int choice;

printf("==============================\n");

printf("LOGISTICS MANAGEMENT SYSTEM\n");

printf("==============================\n");

printf("1. City Management\n");

printf("2. Distance Management\n");

printf("3. Vehicle Information\n");

printf("4. Delivery Request\n");

printf("5. View All Deliveries\n");

printf("6. Route Finder\n");

printf("7. Reports\n");

printf("8. Help\n");

printf("9. Exit\n");

printf("Enter your Choice (1-9):\n");

scanf("%d",&choice);

switch(choice){

case 1:

cityManagement();

break;

case 2:

distanceManagement();

break;

case 3:

vehicleInfo();

break;

case 4:

deliveryRequest();

break;

case 5:

viewDeliveries();

break;

case 6:

routeFinder();

break;

case 7:

reports();

break;

case 8: help();

break;

case 9:

exitProgram();

return 0;

break;

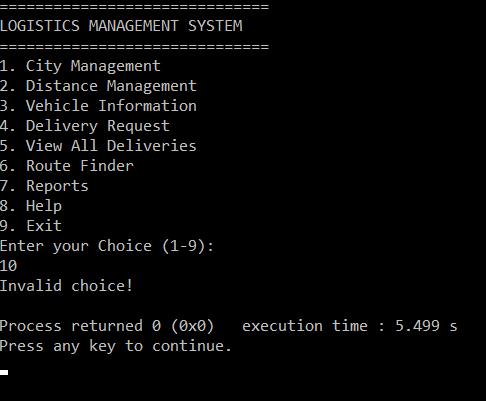
default:

printf("Invalid choice!\n");

}

return 0;

}



Invalid choice

## cityManagement() Function

* Allows users to manage the list of cities

void cityManagement() {

int choice;

int count=0;

char cities[MAX\_CITIES][CITY\_N\_LENTH];

loadCities(cities, &count);

printf("\nCITY MANAGEMENT\n");

printf("\n==========================\n");

printf("1. Add City\n");

printf("2. Rename City\n");

printf("3. Remove City\n");

printf("4. View All Cities\n");

printf("5. Return to Main Menu\n");

printf("6. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

addCity(cities, &count);

break;

case 2:

renameCity(cities, count);

break;

case 3:

removeCity(cities, &count);

break;

case 4:

viewCities(cities, count);

break;

case 5:

saveCities(cities, count);

printf("Returning to main menu...\n");

main();

break;

case 6:

exitProgram();

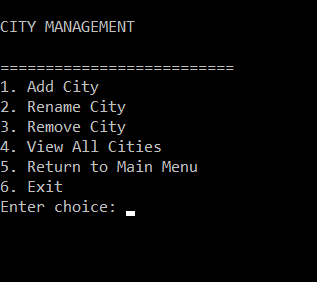
break;

default:

printf("Invalid choice! Try again.\n");

}

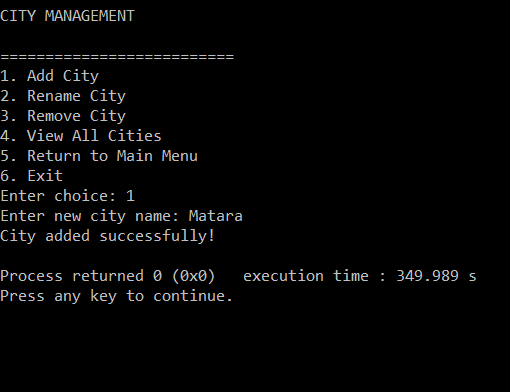
}



## SUB FUNCTION

### addCity() FUNCTION

* Adds a new city to the file.



void addCity(char cities[][CITY\_N\_LENTH], int \*count) {

if (\*count >= MAX\_CITIES) {

printf("LIMIT EXCEEDED (MAX %d)!\n", MAX\_CITIES);

return;

}

printf("Enter new city name: ");

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

if (fgets(cities[\*count], CITY\_N\_LENTH, stdin) == NULL) {

printf("Input error!\n");

return;

}

cities[\*count][strcspn(cities[\*count], "\n")] = '\0';

(\*count)++;

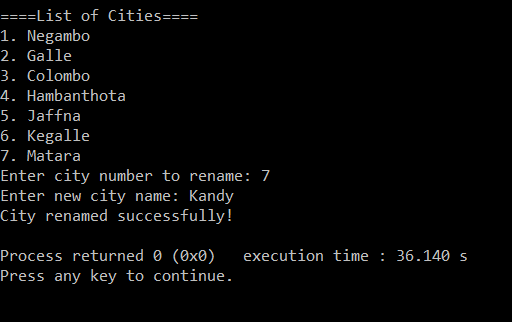
saveCities(cities, \*count);

printf("City added successfully!\n");

}

### RenameCity() FUNCTION

* Renames an existing city from the list.



void renameCity(char cities[][CITY\_N\_LENTH],int count){

if (count == 0) {

printf("No cities available!.\n");

return;

}

viewCities(cities, count);

int index;

printf("Enter city number to rename: ");

scanf("%d", &index);

if (index < 1 || index > count) {

printf("Invalid Index!\n");

return;

}

printf("Enter new city name: ");

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

fgets(cities[index - 1], CITY\_N\_LENTH, stdin);

cities[index - 1][strcspn(cities[index - 1], "\n")] = '\0';

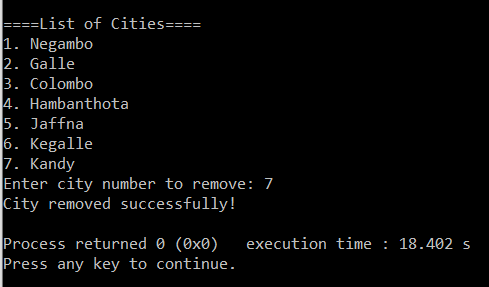
saveCities(cities, count);

printf("City renamed successfully!\n");

}

### removeCity() FUNCTION

* Removes a selected city.



void removeCity(char cities[][CITY\_N\_LENTH],int \*count){

if (\*count == 0) {

printf("No cities available!.\n");

return;

}

viewCities(cities,\*count);

int index;

printf("Enter city number to remove: ");

scanf("%d", &index);

if (index < 1 || index > \*count) {

printf("Invalid number!\n");

return;

}

for (int i = index - 1; i < \*count - 1; i++) {

strcpy(cities[i], cities[i + 1]);

}

(\*count)--;

saveCities(cities, \*count);

printf("City removed successfully!\n");

}

### viewCities() Function

* Display all cities currently available



void viewCities(char cities[][CITY\_N\_LENTH],int count){

if (count == 0) {

printf("No cities available!.\n");

return;

}

printf("\n====List of Cities====\n");

for (int i = 0; i < count; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

}

### loadCities() & saveCities()

* Handle file input/output for cities.txt

void loadCities(char cities[][CITY\_N\_LENTH],int \*count){

FILE \*fp = fopen(CITY\_FILE,"r");

if (fp==NULL)return;

while (fgets(cities[\*count], CITY\_N\_LENTH, fp)) {

cities[\*count][strcspn(cities[\*count], "\n")] = '\0';

(\*count)++;

}

fclose(fp);

}

void saveCities(char cities[][CITY\_N\_LENTH],int count){

FILE \*fp = fopen(CITY\_FILE, "w");

if (fp == NULL) {

printf("SAVING ERROR!\n");

return;

}

for (int i = 0; i < count; i++) {

fprintf(fp, "%s\n", cities[i]);

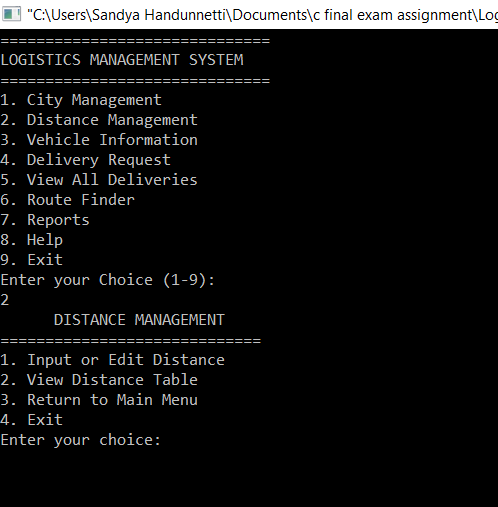
}

fclose(fp);

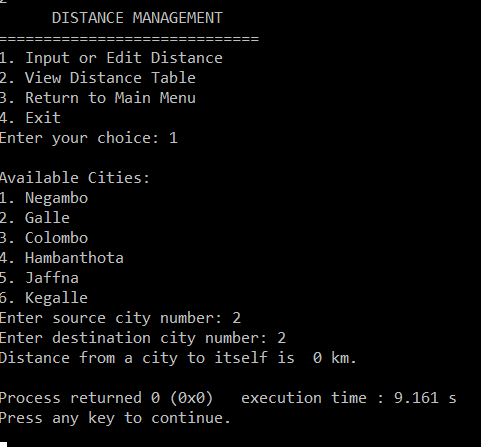
}

## distanceManagement() FUNCTION

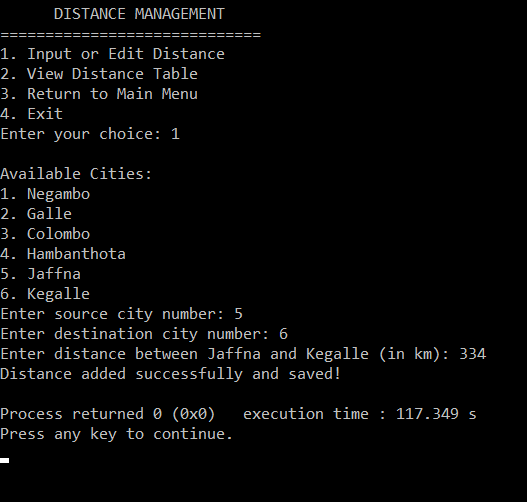
* Handle the distances between cities.
* Allows the user to input edit or view city to city distance table



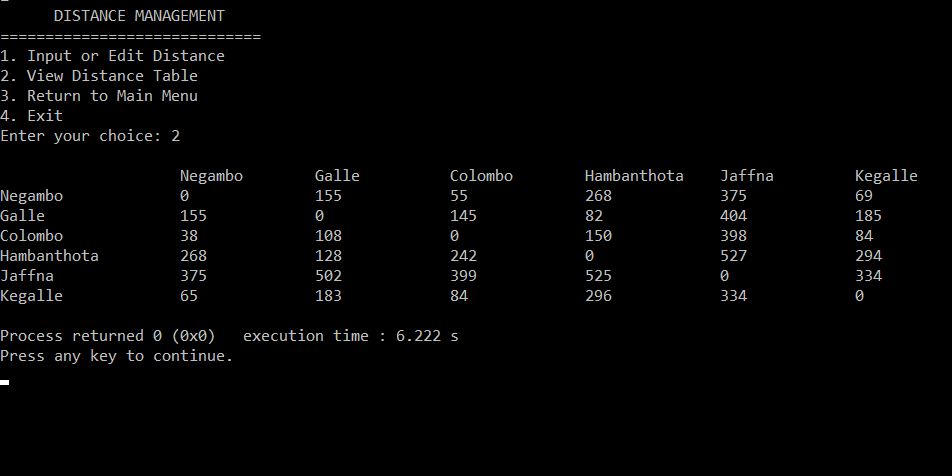
DISTANCE MANAGEMENT interface



Distance between same city



## Input & Edit distance



View Distance Table

void distanceManagement(){

int distance[MAX\_CITIES][MAX\_CITIES];

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int count = 0;

int choice;

memset(distance,0,sizeof(distance));

loadCities(cities, &count);

if (count < 2) {

printf("At least two cities are needed!\n");

return;

}

loadDistances(distance, count);

printf(" DISTANCE MANAGEMENT \n");

printf("=============================\n");

printf("1. Input or Edit Distance\n");

printf("2. View Distance Table\n");

printf("3. Return to Main Menu\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

int i, j, d;

printf("\nAvailable Cities:\n");

for (int k = 0; k < count; k++) {

printf("%d. %s\n", k + 1, cities[k]);

}

printf("Enter source city number: ");

scanf("%d", &i);

printf("Enter destination city number: ");

scanf("%d", &j);

if (i < 1 || i > count || j < 1 || j > count) {

printf("Invalid city numbers!\n");

break;

}

if (i == j) {

printf("Distance from a city to itself is 0 km.\n");

distance[i - 1][j - 1] = 0;

saveDistances(distance,count);

break;

}

printf("Enter distance between %s and %s (in km): ", cities[i - 1], cities[j - 1]);

scanf("%d", &d);

distance[i - 1][j - 1] = d;

distance[j - 1][i - 1] = d;

saveDistances(distance, count);

printf("Distance added successfully and saved!\n");

break;

}

case 2: {

printf("\n%-20s", " ");

for (int i = 0; i < count; i++) {

printf("%-15s", cities[i]);

}

printf("\n");

for (int i = 0; i < count; i++) {

printf("%-20s", cities[i]);

for (int j = 0; j < count; j++) {

printf("%-15d", distance[i][j]);

}

printf("\n");

}

break;

}

case 3:

printf("Returning to main menu...\n");

main();

return;

case 4:

exitProgram();

return;

default:

printf("Invalid choice! Try again.\n");

}

}

## Sub FUNCTION

### loadDistances() FUNCTION

* Loads distance values from distances.txt.

void loadDistances(int distance[MAX\_CITIES][MAX\_CITIES], int cityCount) {

FILE \*fp = fopen(DIST\_FILE, "r");

if (fp == NULL) {

for (int i = 0; i < cityCount; i++)

for (int j = 0; j < cityCount; j++)

distance[i][j] = 0;

return;

}

for (int i = 0; i < cityCount; i++) {

for (int j = 0; j < cityCount; j++) {

if (fscanf(fp, "%d", &distance[i][j]) != 1) {

distance[i][j] = 0;

}

}

}

fclose(fp);

}

### SaveDistances() FUNCTION

* Saves updated distances to the same file

void saveDistances(int distance[MAX\_CITIES][MAX\_CITIES], int cityCount) {

FILE \*fp = fopen(DIST\_FILE, "w");

if (fp == NULL) {

printf("Error saving distances!\n");

return;

}

for (int i = 0; i < cityCount; i++) {

for (int j = 0; j < cityCount; j++) {

fprintf(fp, "%d ", distance[i][j]);

}

fprintf(fp, "\n");

}

fclose(fp);

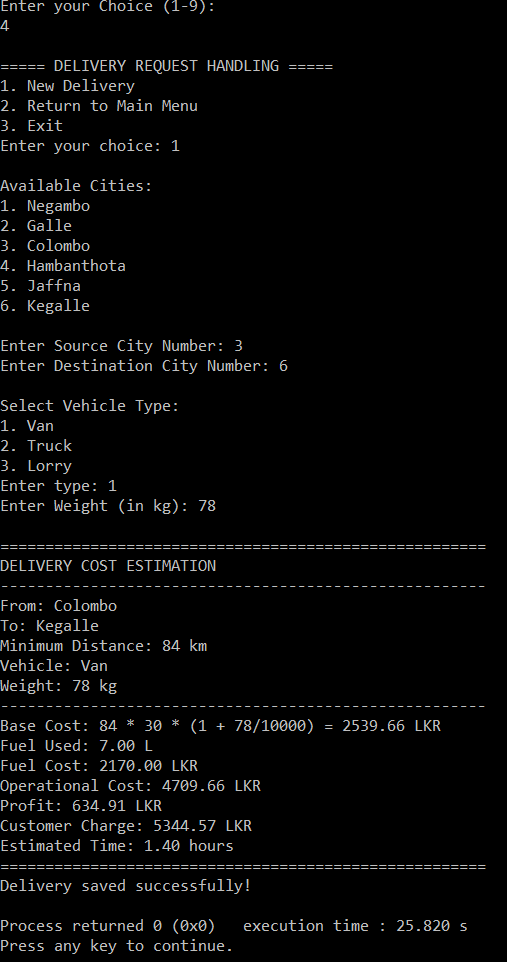
}

## vehicleInfo() FUNCTION

* Shows all vehicle types with details such as capacity,rate,speed,and fuel efficiency.

## deliveryRequest() FUNCTION

* Handle delivery creation and cost calculation.
* Calculates distance,vehicle cost,fuel cost,total cost,profit,and estimated time.
* Save delivery details to deliveries,txt.



User interface & Slip printing

void deliveryRequest(){

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int distance[MAX\_CITIES][MAX\_CITIES]={0};

int cityCount = 0;

loadCities(cities, &cityCount);

loadDistances(distance, cityCount);

int choice;

printf("\n===== DELIVERY REQUEST HANDLING =====\n");

printf("1. New Delivery\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

int src, dest, vType;

float W;

printf("\nAvailable Cities:\n");

for (int i = 0; i < cityCount; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

printf("\nEnter Source City Number: ");

scanf("%d", &src);

printf("Enter Destination City Number: ");

scanf("%d", &dest);

if (src < 1 || src > cityCount || dest < 1 || dest > cityCount) {

printf("Invalid city numbers!\n");

break;

}

if (src == dest) {

printf("Source and destination cannot be the same!\n");

break;

}

printf("\nSelect Vehicle Type:\n1. Van\n2. Truck\n3. Lorry\nEnter type: ");

scanf("%d", &vType);

if (vType < 1 || vType > 3) {

printf("Invalid vehicle type!\n");

break;

}

printf("Enter Weight (in kg): ");

scanf("%f", &W);

if (W > capacity[vType - 1]) {

printf("Weight exceeds capacity of %s (%d kg)!\n",

vehicle\_names[vType - 1], capacity[vType - 1]);

break;

}

int D = distance[src - 1][dest - 1];

if (D <= 0) {

printf("No valid distance found between %s and %s.\n",

cities[src - 1], cities[dest - 1]);

break;

}

float R = rate\_per\_km[vType - 1];

float S = speed[vType - 1];

float E = fuel\_efficiency[vType - 1];

float fuel\_price = 310;

float deliveryCost = D \* R \* (1 + (W/10000));

float fuelUsed = D / E;

float fuelCost = fuelUsed \* fuel\_price;

float totalCost = deliveryCost + fuelCost;

float profit = deliveryCost \* 0.25;

float customerCharge = totalCost + profit;

float timeHrs =D / S;

printf("\n======================================================\n");

printf("DELIVERY COST ESTIMATION\n");

printf("------------------------------------------------------\n");

printf("From: %s\n", cities[src - 1]);

printf("To: %s\n", cities[dest - 1]);

printf("Minimum Distance: %d km\n", D);

printf("Vehicle: %s\n", vehicle\_names[vType - 1]);

printf("Weight: %.0f kg\n", W);

printf("------------------------------------------------------\n");

printf("Base Cost: %d \* %.0f \* (1 + %.0f/10000) = %.2f LKR\n",

D, R, W, deliveryCost);

printf("Fuel Used: %.2f L\n", fuelUsed);

printf("Fuel Cost: %.2f LKR\n", fuelCost);

printf("Operational Cost: %.2f LKR\n", totalCost);

printf("Profit: %.2f LKR\n", profit);

printf("Customer Charge: %.2f LKR\n", customerCharge);

printf("Estimated Time: %.2f hours\n", timeHrs);

printf("======================================================\n");

struct Delivery d;

strcpy(d.source, cities[src - 1]);

strcpy(d.destination, cities[dest - 1]);

strcpy(d.vehicle, vehicle\_names[vType - 1]);

d.weight = W;

d.cost = customerCharge;

d.time = timeHrs;

saveDelivery(d);

printf("Delivery saved successfully!\n");

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

## Sub FUNCTION

### saveDeliveries

* Store delivery details

void saveDelivery(struct Delivery d) {

FILE \*fp = fopen(DELIVERY\_FILE, "a");

if (fp == NULL) {

printf("Error saving delivery!\n");

return;

}

fprintf(fp, "%s %s %s %.2f %.2f %.2f\n",

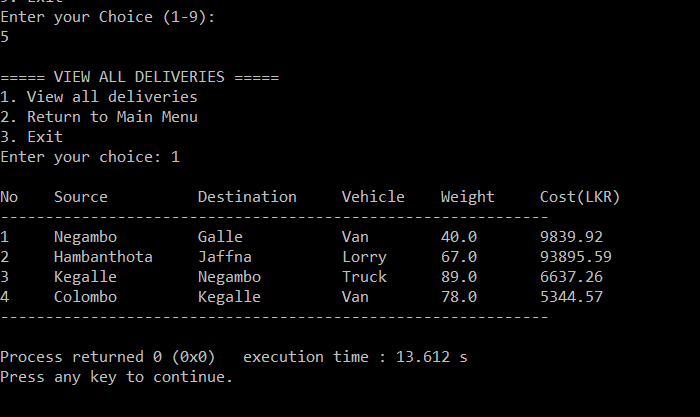
d.source, d.destination, d.vehicle, d.weight, d.cost, d.time);

fclose(fp);

}

## viewDeliveries() FUNCTION

* View all past deliveries saved in deliveries.txt



void viewDeliveries() {

struct Delivery deliveries[MAX\_DELIVERIES];

int count = 0;

loadDeliveries(deliveries, &count);

int choice;

printf("\n===== VIEW ALL DELIVERIES =====\n");

printf("1. View all deliveries\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

if (count == 0) {

printf("No deliveries recorded yet!\n");

return;

}

printf("\n%-5s %-15s %-15s %-10s %-10s %-10s\n",

"No", "Source", "Destination", "Vehicle", "Weight", "Cost(LKR)");

printf("-------------------------------------------------------------\n");

for (int i = 0; i < count; i++) {

printf("%-5d %-15s %-15s %-10s %-10.1f %-10.2f\n",

i + 1,

deliveries[i].source,

deliveries[i].destination,

deliveries[i].vehicle,

deliveries[i].weight,

deliveries[i].cost);

}

printf("-------------------------------------------------------------\n");

break;

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

printf("Invalid choice! Please enter 1–3.\n");

}

}

## Sub FUNction

### loadDeliveries()

* Loads and reads all delivery records from the file.

void loadDeliveries(struct Delivery deliveries[], int \*count) {

FILE \*fp = fopen(DELIVERY\_FILE, "r");

if (fp == NULL) {

\*count = 0;

return;

}

while (fscanf(fp, "%s %s %s %f %f %f",

deliveries[\*count].source,

deliveries[\*count].destination,

deliveries[\*count].vehicle,

&deliveries[\*count].weight,

&deliveries[\*count].cost,

&deliveries[\*count].time) == 6) {

(\*count)++;

if (\*count >= MAX\_DELIVERIES)

break;

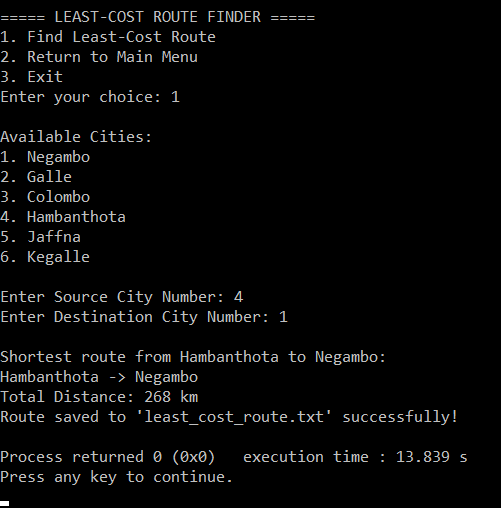
}

fclose(fp);

}

## routeFinder() FUNCTION

* Finds the shortest route between two cities.
* Saves the route and total distance to least\_cost\_route.txt



void routeFinder() {

char cities[MAX\_CITIES][CITY\_N\_LENTH];

int distance[MAX\_CITIES][MAX\_CITIES];

int cityCount = 0;

loadCities(cities, &cityCount);

loadDistances(distance, cityCount);

if (cityCount < 2) {

printf("Not enough cities to find a route!\n");

return;

}

int choice;

printf("\n===== LEAST-COST ROUTE FINDER =====\n");

printf("1. Find Least-Cost Route\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

printf("\nAvailable Cities:\n");

for (int i = 0; i < cityCount; i++) {

printf("%d. %s\n", i + 1, cities[i]);

}

int src, dest;

printf("\nEnter Source City Number: ");

scanf("%d", &src);

printf("Enter Destination City Number: ");

scanf("%d", &dest);

if (src < 1 || src > cityCount || dest < 1 || dest > cityCount) {

printf("Invalid city numbers!\n");

break;

}

src--;

dest--;

int cost[MAX\_CITIES], visited[MAX\_CITIES], prev[MAX\_CITIES];

for (int i = 0; i < cityCount; i++) {

cost[i] = 999999;

visited[i] = 0;

prev[i] = -1;

}

cost[src] = 0;

for (int i = 0; i < cityCount - 1; i++) {

int min = 999999, u = -1;

for (int j = 0; j < cityCount; j++) {

if (!visited[j] && cost[j] < min) {

min = cost[j];

u = j;

}

}

if (u == -1) break;

visited[u] = 1;

for (int v = 0; v < cityCount; v++) {

if (distance[u][v] > 0 && !visited[v] &&

cost[u] + distance[u][v] < cost[v]) {

cost[v] = cost[u] + distance[u][v];

prev[v] = u;

}

}

}

if (cost[dest] == 999999) {

printf("\nNo path found between %s and %s.\n", cities[src], cities[dest]);

break;

}

int path[MAX\_CITIES], count = 0;

for (int v = dest; v != -1; v = prev[v]) {

path[count++] = v;

}

printf("\nShortest route from %s to %s:\n", cities[src], cities[dest]);

for (int i = count - 1; i >= 0; i--) {

printf("%s", cities[path[i]]);

if (i > 0) printf(" -> ");

}

printf("\nTotal Distance: %d km\n", cost[dest]);

FILE \*fp = fopen("least\_cost\_route.txt", "a");

if (fp != NULL) {

fprintf(fp, "From %s to %s : ", cities[src], cities[dest]);

for (int i = count - 1; i >= 0; i--) {

fprintf(fp, "%s", cities[path[i]]);

if (i > 0) fprintf(fp, " -> ");

}

fprintf(fp, " | Distance: %d km\n", cost[dest]);

fclose(fp);

printf("Route saved to 'least\_cost\_route.txt' successfully!\n");

} else {

printf("Error saving route to file!\n");

}

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

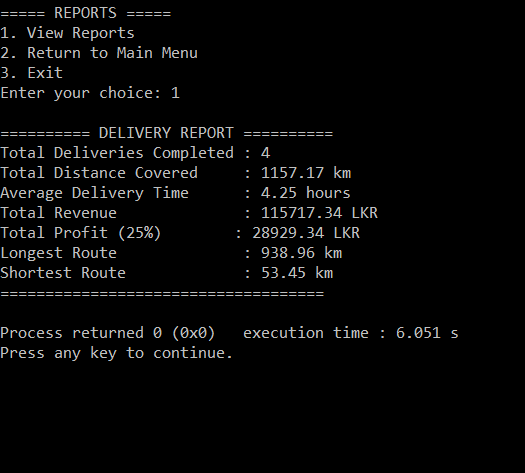
printf("\nInvalid choice! Please enter 1–3.\n");

}

}

## reports FUNCTION

* Generates summary statistics abut deliveries
* Total deliveries.
* Total revenue and profit
* Longest and shortest routes
* Average time per delivery.



void reports() {

struct Delivery deliveries[MAX\_DELIVERIES];

int count = 0;

loadDeliveries(deliveries, &count);

int choice;

printf("\n===== REPORTS =====\n");

printf("1. View Reports\n");

printf("2. Return to Main Menu\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

if (count == 0) {

printf("\nNo delivery records found!\n");

break;

}

float totalDistance = 0, totalTime = 0, totalRevenue = 0;

float longest = 0, shortest = 999999;

for (int i = 0; i < count; i++) {

float distance = deliveries[i].cost / 100;

totalDistance += distance;

totalTime += deliveries[i].time;

totalRevenue += deliveries[i].cost;

if (distance > longest)

longest = distance;

if (distance < shortest)

shortest = distance;

}

float avgTime = totalTime / count;

float profit = totalRevenue \* 0.25;

printf("\n========== DELIVERY REPORT ==========\n");

printf("Total Deliveries Completed : %d\n", count);

printf("Total Distance Covered : %.2f km\n", totalDistance);

printf("Average Delivery Time : %.2f hours\n", avgTime);

printf("Total Revenue : %.2f LKR\n", totalRevenue);

printf("Total Profit (25%%) : %.2f LKR\n", profit);

printf("Longest Route : %.2f km\n", longest);

printf("Shortest Route : %.2f km\n", shortest);

printf("====================================\n");

break;

}

case 2:

printf("\nReturning to main menu...\n");

main();

return;

case 3:

exitProgram();

return;

default:

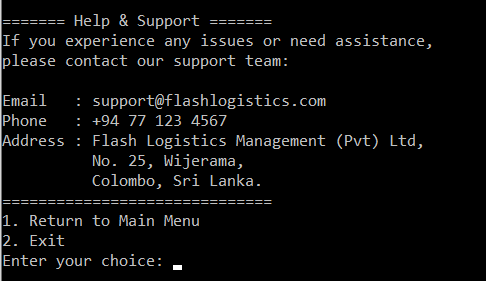
printf("Invalid choice! Please enter 1–3.\n");

}

}

## help() FUNCTION

* Display contact information for system support.



void help(){

int choice;

printf("\n======= Help & Support =======\n");

printf("If you experience any issues or need assistance,\n");

printf("please contact our support team:\n\n");

printf("Email : support@flashlogistics.com\n");

printf("Phone : +94 77 123 4567\n");

printf("Address : Flash Logistics Management (Pvt) Ltd,\n");

printf(" No. 25, Wijerama,\n");

printf(" Colombo, Sri Lanka.\n");

printf("==============================\n");

printf("1. Return to Main Menu\n");

printf("2. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nReturning to main menu...\n");

main();

return;

case 2:

exitProgram();

return;

default:

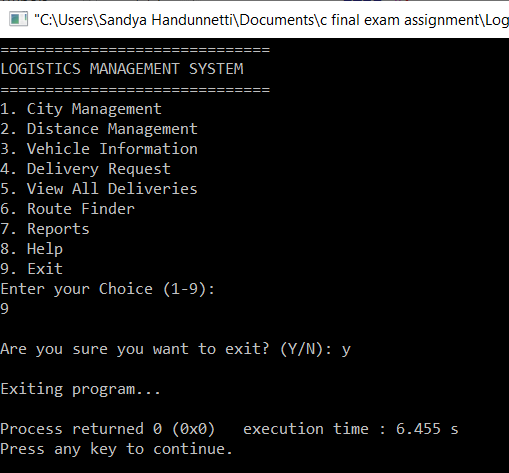
printf("Invalid choice! Please enter 1–3.\n");

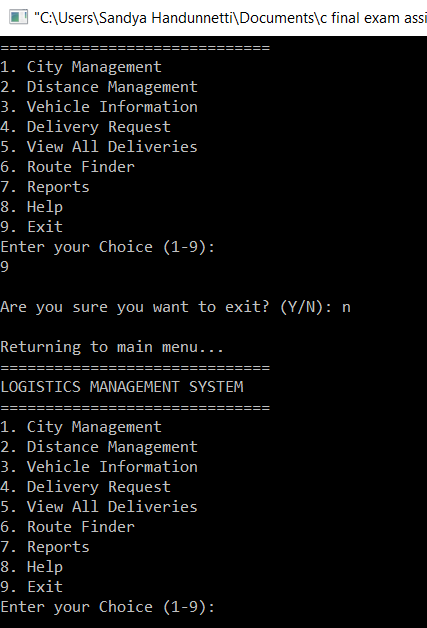
}

}

## exitProgram() FUNCTION

* Confirms exit and terminates the program safely





void exitProgram(){

char choice;

printf("\nAre you sure you want to exit? (Y/N): ");

scanf(" %c", &choice);

if (choice == 'Y' || choice == 'y')

{

printf("\nExiting program...\n");

exit(0);

}

else if (choice == 'N' || choice == 'n')

{

printf("\nReturning to main menu...\n");

main();

}

else

{

printf("Invalid input! Please enter Y or N.\n");

}

}

# 09.GitHub repository link

# 