Course Title:	Programming Fundamentals Lab (CL1002)
Assignment Title:	Lab Manual 12 Tasks
Submitted to:	Sir Sandesh Kumar
Name:	Muhammad Usman Khan
Roll No:	25K-2038 BCY-1A
Date:	December 6th 2025

LAB TASKS [14 Marks]

1. Design a Fitness Tracker Data Analyzer where a fitness tracker stores the number of steps taken every hour. Write a program to dynamically allocate an integer array for hourly step data, allow user to add more hours (realloc). Use **pointer arithmetic** to compute: Max steps, total steps, hours with steps above a threshold and Save data in fitness_tracker.txt file.

2. Design a Student Exam Seating System where a university wants to assign exam seat numbers dynamically. Write a program to dynamically allocate an array of seat assignments. Each seat contains: `studentName`, `rollNumber`, `seatNumber`. Validate roll numbers using a **recursive function** (e.g., check if alphanumeric). Save final seating plan to seating .txt file. Handle memory reallocation for additional students.

3. Design a Temperature Monitoring & Alert System where a weather station records temperature readings throughout the day. Write a program to dynamically store temperature readings, allow user to append more readings using realloc. Compute: Highest and lowest temperature, number of readings crossing an alert threshold and write a temperature_summary.txt report to a file.

4. Design a Digital Library Checkout Logger where a librarian wants to track book checkouts for the day. Write a program to dynamically allocate an array for book checkout logs. Each record: `bookTitle`, `memberID`, `checkoutTime`, `returnDueDays`, automatically compute due dates and append data to a file in CSV format.

5. Design a Car Rental Duration & Cost Calculator where a car rental service stores rental durations dynamically. Write a program to dynamically store rental durations (hours or days). Expand the array whenever a new car is rented, using pointers, compute: Total rental time for the day, highest rental duration, cost per customer and write invoices to a Rental _Invoices.txt file.

6. Design a Real-Time Event Attendance Counter where an event venue wants to dynamically track attendance at different entry gates. Write a program to allocate an array storing headcounts at each gate, add more gates using realloc. Find: Total attendees **recursively,** gate with highest attendance. Save all data to attendance.txt file with timestamps.

7. Design a Medication Inventory Manager where a pharmacy wants to manage medication inventory. Write a program to use dynamic arrays to store: `medicineName`, `quantityAvailable`, `unitPrice`, Let user add/remove medicines by resizing arrays. Compute: Total inventory value, low-stock alerts and save all the data in medicine_inventory.txt file on exit.

Question 1

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    int *hours_steps = NULL;
    int total_hours = 0;
    int capacity = 2;
    int i;

    hours_steps = (int *)malloc(capacity * sizeof(int));
    if (hours_steps == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    int choice = 1;

    while (choice == 1) {
        if (total_hours == capacity) {
            capacity = capacity * 2;
            hours_steps = (int *)realloc(hours_steps, capacity * sizeof(int));
            if (hours_steps == NULL) {
                printf("Reallocation failed\n");
                return 1;
            }
        }

        printf("Enter steps for hour %d: ", total_hours + 1);
        scanf("%d", (hours_steps + total_hours));
        total_hours++;

        printf("Add more hours? (1 = yes, 0 = no): ");
        scanf("%d", &choice);
    }

    int total_steps = 0;
    int max_steps = *(hours_steps);
    int threshold;
    int above_threshold = 0;

    for (i = 0; i < total_hours; i++) {
        total_steps = total_steps + *(hours_steps + i);

        if (*(hours_steps + i) > max_steps) {
            max_steps = *(hours_steps + i);
        }
    }

    printf("Enter step threshold: ");
    scanf("%d", &threshold);

    for (i = 0; i < total_hours; i++) {
        if (*(hours_steps + i) > threshold) {
            above_threshold++;
        }
    }

    FILE *fp = fopen("fitness_tracker.txt", "w");
    if (fp == NULL) {
        printf("File could not be opened\n");
        return 1;
    }

    fprintf(fp, "Fitness Tracker Data\n");
    fprintf(fp, "Total hours: %d\n", total_hours);
    fprintf(fp, "Total steps: %d\n", total_steps);
    fprintf(fp, "Maximum steps in an hour: %d\n", max_steps);
    fprintf(fp, "Hours above threshold: %d\n\n", above_threshold);

    for (i = 0; i < total_hours; i++) {
        fprintf(fp, "Hour %d: %d steps\n", i + 1, *(hours_steps + i));
    }

    fclose(fp);

    printf("\nAnalysis Complete\n");
    printf("Total steps: %d\n", total_steps);
    printf("Max steps in an hour: %d\n", max_steps);
    printf("Hours above threshold: %d\n", above_threshold);
    printf("Data saved in fitness_tracker.txt\n");

    free(hours_steps);
    return 0;
}
```

```
Enter steps for hour 1: 1200
Add more hours? (1 = yes, 0 = no): 1
Enter steps for hour 2: 3000
Add more hours? (1 = yes, 0 = no): 1
Enter steps for hour 3: 1800
Add more hours? (1 = yes, 0 = no): 0
Enter step threshold: 2000

Analysis Complete
Total steps: 6000
Max steps in an hour: 3000
Hours above threshold: 1
Data saved in fitness_tracker.txt

--------------------------------
Process exited after 26.31 seconds with return value 0
Press any key to continue . . .
```

Question 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Student {
    char student_name[50];
    char roll_number[20];
    int seat_number;
};

int check_roll_recursive(char *roll, int i) {
    if (roll[i] == '\0')
        return 1;

    if ((roll[i] >= 'A' && roll[i] <= 'Z') ||
        (roll[i] >= 'a' && roll[i] <= 'z') ||
        (roll[i] >= '0' && roll[i] <= '9')) {
        return check_roll_recursive(roll, i + 1);
    }

    return 0;
}

int main() {

    struct Student *students = NULL;
    int total_students = 0;
    int capacity = 2;
    int choice = 1;
    int i;

    students = (struct Student *)malloc(capacity * sizeof(struct Student));
    if (students == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    while (choice == 1) {

        if (total_students == capacity) {
            capacity = capacity * 2;
            students = (struct Student *)realloc(students, capacity * sizeof(struct Student));
            if (students == NULL) {
                printf("Reallocation failed\n");
                return 1;
            }
        }

        printf("Enter student name: ");
        getchar();
        fgets(students[total_students].student_name, 50, stdin);
        students[total_students].student_name[strcspn(students[total_students].student_name, "\n")] = '\0'

        do {
            printf("Enter roll number (alphanumeric only): ");
            fgets(students[total_students].roll_number, 20, stdin);
            students[total_students].roll_number[strcspn(students[total_students].roll_number, "\n")] = '\

            if (check_roll_recursive(students[total_students].roll_number, 0) == 0) {
                printf("Invalid roll number. Try again.\n");
            }
        } while (check_roll_recursive(students[total_students].roll_number, 0) == 0);

        students[total_students].seat_number = total_students + 1;

        total_students++;

        printf("Add more students? (1 = yes, 0 = no): ");
        scanf("%d", &choice);
    }
    FILE *fp = fopen("seating.txt", "w");
    if (fp == NULL) {
        printf("File open failed\n");
        return 1;
    }
    fprintf(fp, "Exam Seating Plan\n\n");

    for (i = 0; i < total_students; i++) {
        fprintf(fp, "Seat %d | Name: %s | Roll: %s\n",
                students[i].seat_number,
                students[i].student_name,
                students[i].roll_number);
    }
    fclose(fp);

    printf("\nSeating plan saved to seating.txt\n");

    free(students);
    return 0;
```

```
Enter student name: Usman Khan
Enter roll number (alphanumeric only): 25K2038
Add more students? (1 = yes, 0 = no): 1
Enter student name: Fazeel Zafar
Enter roll number (alphanumeric only): 25K-2065
Invalid roll number. Try again.
Enter roll number (alphanumeric only): 25K2065
Add more students? (1 = yes, 0 = no): 0

Seating plan saved to seating.txt

--------------------------------
Process exited after 43.07 seconds with return value 0
Press any key to continue . . .
```

```
Exam Seating Plan

Seat 1 | Name: sman Khan | Roll: 25K2038
Seat 2 | Name: Fazeel Zafar | Roll: 25K2065
```

Question 3

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    float *temp_readings = NULL;
    int total_readings = 0;
    int capacity = 3;
    int choice = 1;
    int i;

    float highest_temp;
    float lowest_temp;
    float alert_limit;
    int alert_count = 0;

    temp_readings = (float *)malloc(capacity * sizeof(float));
    if (temp_readings == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    printf("Enter alert temperature threshold: ");
    scanf("%f", &alert_limit);

    while (choice == 1) {

        if (total_readings == capacity) {
            capacity = capacity + 3;
            temp_readings = (float *)realloc(temp_readings, capacity * sizeof(float));
            if (temp_readings == NULL) {
                printf("Reallocation failed\n");
                return 1;
            }
        }

        printf("Enter temperature reading: ");
        scanf("%f", &temp_readings[total_readings]);

        total_readings++;

        printf("Add more readings? (1 = yes, 0 = no): ");
        scanf("%d", &choice);
    }

    highest_temp = temp_readings[0];
    lowest_temp = temp_readings[0];

    for (i = 0; i < total_readings; i++) {

        if (temp_readings[i] > highest_temp)
            highest_temp = temp_readings[i];

        if (temp_readings[i] < lowest_temp)
            lowest_temp = temp_readings[i];

        if (temp_readings[i] > alert_limit)
            alert_count++;
    }

    FILE *fp = fopen("temperature_summary.txt", "w");
    if (fp == NULL) {
        printf("File open failed\n");
        return 1;
    }

    fprintf(fp, "Temperature Monitoring Report\n\n");
    fprintf(fp, "Total Readings: %d\n", total_readings);
    fprintf(fp, "Highest Temperature: %.2f\n", highest_temp);
    fprintf(fp, "Lowest Temperature: %.2f\n", lowest_temp);
    fprintf(fp, "Alert Threshold: %.2f\n", alert_limit);
    fprintf(fp, "Readings Above Alert: %d\n", alert_count);

    fclose(fp);

    printf("\nSummary saved to temperature_summary.txt\n");

    free(temp_readings);
    return 0;
}
```

```
Enter alert temperature threshold: 37.5
Enter temperature reading: 36.8
Add more readings? (1 = yes, 0 = no): 1
Enter temperature reading: 38.2
Add more readings? (1 = yes, 0 = no): 1
Enter temperature reading: 37.0
Add more readings? (1 = yes, 0 = no): 1
Enter temperature reading: 39.1
Add more readings? (1 = yes, 0 = no): 0

Summary saved to temperature_summary.txt

--------------------------------
Process exited after 41.23 seconds with return value 0
Press any key to continue . . .
```

```
Temperature Monitoring Report

Total Readings: 4
Highest Temperature: 39.10
Lowest Temperature: 36.80
Alert Threshold: 37.50
Readings Above Alert: 2
```

Question 4

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct checkout_log {
    char book_title[100];
    int member_id;
    int checkout_day;
    int due_day;
    int return_due_days;
};

int main() {

    struct checkout_log *logs = NULL;
    int total_logs = 0;
    int capacity = 2;
    int choice = 1;

    logs = (struct checkout_log *)malloc(capacity * sizeof(struct checkout_log));
    if (logs == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    while (choice == 1) {

        if (total_logs == capacity) {
            capacity = capacity + 2;
            logs = (struct checkout_log *)realloc(logs, capacity * sizeof(struct checkout_log));
            if (logs == NULL) {
                printf("Reallocation failed\n");
                return 1;
            }
        }

        printf("Enter book title: ");
        getchar();
        fgets(logs[total_logs].book_title, 100, stdin);
        logs[total_logs].book_title[strcspn(logs[total_logs].book_title, "\n")] = '\0';

        printf("Enter member ID: ");
        scanf("%d", &logs[total_logs].member_id);

        printf("Enter checkout day (day number): ");
        scanf("%d", &logs[total_logs].checkout_day);

        printf("Enter return due days: ");
        scanf("%d", &logs[total_logs].return_due_days);

        logs[total_logs].due_day =
            logs[total_logs].checkout_day +
            logs[total_logs].return_due_days;

        total_logs++;

        printf("Add another checkout? (1 = yes, 0 = no): ");
        scanf("%d", &choice);
    }

    FILE *fp = fopen("checkout_logs.csv", "a");
    if (fp == NULL) {
        printf("File open failed\n");
        return 1;
    }

    fprintf(fp, "Book Title,Member ID,Checkout Day,Due Day\n");

    for (int i = 0; i < total_logs; i++) {
        fprintf(fp, "%s,%d,%d,%d\n",
            logs[i].book_title,
            logs[i].member_id,
            logs[i].checkout_day,
            logs[i].due_day
        );
    }

    fclose(fp);

    printf("\nCheckout logs saved to checkout_logs.csv\n");

    free(logs);
    return 0;
}
```

```
Enter book title: Clean Code
Enter member ID: 2038
Enter checkout day (day number): 5
Enter return due days: 14
Add another checkout? (1 = yes, 0 = no): 1
Enter book title: Introduction to Science
Enter member ID: 2065
Enter checkout day (day number): 10
Enter return due days: 7
Add another checkout? (1 = yes, 0 = no): 0

Checkout logs saved to checkout_logs.csv
```

```
1    Book Title,Member ID,Checkout Day,Due Day
2    lean Code,2038,5,19
3    Introduction to Science,2065,10,17
```

Question 5

```c
#include <stdio.h>
#include <stdlib.h>

struct rental {
    int customer_id;
    int rental_hours;
    int cost;
};

int main() {

    struct rental *rentals = NULL;
    int total_customers = 0;
    int size = 2;
    int choice = 1;

    int rate_per_hour = 500;
    int total_hours = 0;
    int max_hours = 0;

    rentals = (struct rental *)malloc(size * sizeof(struct rental));
    if (rentals == NULL) {
        printf("Memory not allocated\n");
        return 1;
    }

    while (choice == 1) {

        if (total_customers == size) {
            size = size + 2;
            rentals = (struct rental *)realloc(rentals, size * sizeof(struct rental));
            if (rentals == NULL) {
                printf("Reallocation failed\n");
                return 1;
            }
        }

        printf("Enter customer id: ");
        scanf("%d", &rentals[total_customers].customer_id);

        printf("Enter rental duration in hours: ");
        scanf("%d", &rentals[total_customers].rental_hours);

        rentals[total_customers].cost =
            rentals[total_customers].rental_hours * rate_per_hour;
            rentals[total_customers].rental_hours * rate_per_hour;

        total_hours = total_hours + rentals[total_customers].rental_hours;

        if (rentals[total_customers].rental_hours > max_hours)
            max_hours = rentals[total_customers].rental_hours;

        total_customers++;

        printf("Add another rental? (1 = yes, 0 = no): ");
        scanf("%d", &choice);
    }

    FILE *fp = fopen("Rental_Invoices.txt", "w");
    if (fp == NULL) {
        printf("File error\n");
        return 1;
    }

    fprintf(fp, "Car Rental Invoices\n\n");

    for (int i = 0; i < total_customers; i++) {
        fprintf(fp, "Customer ID: %d\n", rentals[i].customer_id);
        fprintf(fp, "Rental Hours: %d\n", rentals[i].rental_hours);
        fprintf(fp, "Total Cost: %d\n\n", rentals[i].cost);
    }

    fprintf(fp, "Total Rental Hours Today: %d\n", total_hours);
    fprintf(fp, "Highest Rental Duration: %d\n", max_hours);

    fclose(fp);

    printf("\nInvoices written to Rental_Invoices.txt\n");

    free(rentals);
    return 0;
}
```

```
Enter customer id: 2038
Enter rental duration in hours: 4
Add another rental? (1 = yes, 0 = no): 1
Enter customer id: 3021
Enter rental duration in hours: 7
Add another rental? (1 = yes, 0 = no): 0

Invoices written to Rental_Invoices.txt

--------------------------------
ds with return value 0
```

```
Car Rental Invoices

Customer ID: 2038
Rental Hours: 4
Total Cost: 2000

Customer ID: 3021
Rental Hours: 7
Total Cost: 3500

Total Rental Hours Today: 11
Highest Rental Duration: 7
```

Question 6

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int* gate_attendance = NULL;
int gate_count = 0;

void add_gate(int count) {
    int* temp = realloc(gate_attendance, (gate_count + count) * sizeof(int));
    if (!temp) {
        printf("Memory allocation failed!\n");
        return;
    }
    gate_attendance = temp;
    for (int i = gate_count; i < gate_count + count; i++) {
        gate_attendance[i] = 0;
    }
    gate_count += count;
    printf("%d gate(s) added successfully.\n", count);
}

void record_attendance() {
    if (gate_count == 0) {
        printf("No gates available. Add gates first.\n");
        return;
    }
    for (int i = 0; i < gate_count; i++) {
        printf("Enter number of attendees at Gate %d: ", i+1);
        scanf("%d", &gate_attendance[i]);
    }
}

int total_attendance_recursive(int index) {
    if (index < 0) return 0;
    return gate_attendance[index] + total_attendance_recursive(index - 1);
}

int gate_with_highest_attendance() {
    if (gate_count == 0) return -1;
    int max_index = 0;
    for (int i = 1; i < gate_count; i++) {
        if (gate_attendance[i] > gate_attendance[max_index]) {
            max_index = i;
        }
    }
    return max_index;
}

void save_attendance() {
    FILE *fp = fopen("attendance.txt", "a");
    if (!fp) {
        printf("Cannot open file for writing!\n");
        return;
    }

    time_t now = time(NULL);
    fprintf(fp, "Attendance report at %s", ctime(&now));
    for (int i = 0; i < gate_count; i++) {
        fprintf(fp, "Gate %d: %d attendees\n", i+1, gate_attendance[i]);
    }
    fprintf(fp, "Total attendees: %d\n", total_attendance_recursive(gate_count-1));
    int max_gate = gate_with_highest_attendance();
    if (max_gate != -1) {
        fprintf(fp, "Gate with highest attendance: Gate %d (%d attendees)\n\n",
                max_gate+1, gate_attendance[max_gate]);
    }

    fclose(fp);
    printf("Attendance data saved to attendance.txt\n");
}

void free_memory() {
    if (gate_attendance) free(gate_attendance);
    gate_attendance = NULL;
    gate_count = 0;
}

int main() {
    int choice, count;
    while (1) {
        printf("\n==== Real-Time Event Attendance Counter ====\n");
        printf("1. Add Gate(s)\n");
        printf("2. Record Attendance\n");
        printf("3. Show Total Attendance\n");
        printf("4. Show Gate with Highest Attendance\n");
        printf("5. Save Attendance Report\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Enter number of gates to add: ");
            scanf("%d", &count);
            add_gate(count);
        }
        else if (choice == 2) {
            record_attendance();
        }
        else if (choice == 3) {
            int total = total_attendance_recursive(gate_count-1);
            printf("Total attendees so far: %d\n", total);
        }
        else if (choice == 4) {
            int max_gate = gate_with_highest_attendance();
            if (max_gate != -1) {
                printf("Gate with highest attendance: Gate %d (%d attendees)\n",
                       max_gate+1, gate_attendance[max_gate]);
            } else {
                printf("No gates added yet.\n");
            }
        }
        else if (choice == 5) {
            save_attendance();
        }
        else if (choice == 6) {
            save_attendance();
            free_memory();
            printf("Exiting program. Memory freed.\n");
            break;
        }
        else {
            printf("Invalid choice. Try again.\n");
        }
    }

    return 0;
}
```

```
==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 1
Enter number of gates to add: 3
3 gate(s) added successfully.

==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 2
Enter number of attendees at Gate 1: 50
Enter number of attendees at Gate 2: 120
Enter number of attendees at Gate 3: 75

==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 3
Total attendees so far: 245

==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 245
Invalid choice. Try again.
==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 4
Gate with highest attendance: Gate 2 (120 attendees)

==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 5
Attendance data saved to attendance.txt

==== Real-Time Event Attendance Counter ====
1. Add Gate(s)
2. Record Attendance
3. Show Total Attendance
4. Show Gate with Highest Attendance
5. Save Attendance Report
6. Exit
Enter your choice: 6
Attendance data saved to attendance.txt
Exiting program. Memory freed.
```

```
Attendance report at Sun Dec 07 22:16:43 2025
Gate 1: 50 attendees
Gate 2: 120 attendees
Gate 3: 75 attendees
Total attendees: 245
Gate with highest attendance: Gate 2 (120 attendees)

Attendance report at Sun Dec 07 22:16:57 2025
Gate 1: 50 attendees
Gate 2: 120 attendees
Gate 3: 75 attendees
Total attendees: 245
Gate with highest attendance: Gate 2 (120 attendees)
```

Question 7

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Medicine {
    char medicine_name[100];
    int quantity_available;
    float unit_price;
};

struct Medicine *inventory = NULL;
int medicine_count = 0;

void add_medicine() {
    struct Medicine temp;

    printf("Enter Medicine Name: ");
    getchar();
    fgets(temp.medicine_name, sizeof(temp.medicine_name), stdin);
    temp.medicine_name[strcspn(temp.medicine_name, "\n")] = '\0';

    printf("Enter Quantity Available: ");
    scanf("%d", &temp.quantity_available);

    printf("Enter Unit Price: ");
    scanf("%f", &temp.unit_price);

    struct Medicine *new_arr = realloc(inventory, (medicine_count + 1) * sizeof(struct Medicine));
    if (!new_arr) {
        printf("Memory allocation failed!\n");
        return;
    }
    inventory = new_arr;
    inventory[medicine_count] = temp;
    medicine_count++;
    printf("Medicine added successfully.\n");
}

void remove_medicine() {
    if (medicine_count == 0) {
        printf("No medicines in inventory.\n");
        return;
    }

    char name[100];
    printf("Enter Medicine Name to remove: ");
    getchar();
    fgets(name, sizeof(name), stdin);
    name[strcspn(name, "\n")] = '\0';

    int found = -1;
    for (int i = 0; i < medicine_count; i++) {
        if (strcmp(inventory[i].medicine_name, name) == 0) {
            found = i;
            break;
        }
    }

    if (found == -1) {
        printf("Medicine not found.\n");
        return;
    }

    for (int i = found; i < medicine_count - 1; i++) {
        inventory[i] = inventory[i + 1];
    }

    medicine_count--;
    if (medicine_count > 0) {
        struct Medicine *new_arr = realloc(inventory, medicine_count * sizeof(struct Medicine));
        if (new_arr) inventory = new_arr;
    } else {
        free(inventory);
        inventory = NULL;
    }

    printf("Medicine removed successfully.\n");
}

void display_inventory() {
    if (medicine_count == 0) {
        printf("Inventory is empty.\n");
        return;
    }

    printf("\nCurrent Inventory:\n");
    for (int i = 0; i < medicine_count; i++) {
        printf("%d. %s | Qty: %d | Price: %.2f\n",
               i + 1,
               inventory[i].medicine_name,
               inventory[i].medicine_name,
               inventory[i].quantity_available,
               inventory[i].unit_price);
    }
}

void compute_inventory_value() {
    float total = 0;
    printf("\nLow Stock Medicines (Qty < 5):\n");
    int low_found = 0;

    for (int i = 0; i < medicine_count; i++) {
        total += inventory[i].quantity_available * inventory[i].unit_price;
        if (inventory[i].quantity_available < 5) {
            printf("- %s | Qty: %d\n", inventory[i].medicine_name, inventory[i].quantity_available);
            low_found = 1;
        }
    }

    if (!low_found) printf("None\n");
    printf("Total Inventory Value: %.2f\n", total);
}

void save_inventory() {
    FILE *fp = fopen("medicine_inventory.txt", "w");
    if (!fp) {
        printf("Cannot open file for writing.\n");
        return;
    }

    for (int i = 0; i < medicine_count; i++) {
        fprintf(fp, "%s,%d,%.2f\n",
                inventory[i].medicine_name,
                inventory[i].quantity_available,
                inventory[i].unit_price);
    }
    fclose(fp);
    printf("Inventory saved to medicine_inventory.txt\n");
}

void free_memory() {
    if (inventory) free(inventory);
    inventory = NULL;
    medicine_count = 0;
}

int main() {
    int choice;

    while (1) {
        printf("\n==== Medication Inventory Manager ====\n");
        printf("1. Add Medicine\n");
        printf("2. Remove Medicine\n");
        printf("3. Display Inventory\n");
        printf("4. Compute Inventory Value & Low-Stock Alerts\n");
        printf("5. Save & Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) add_medicine();
        else if (choice == 2) remove_medicine();
        else if (choice == 3) display_inventory();
        else if (choice == 4) compute_inventory_value();
        else if (choice == 5) {
            save_inventory();
            free_memory();
            printf("Exiting program.\n");
            break;
        }
        else printf("Invalid choice. Try again.\n");
    }

    return 0;
}
```

```
==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 1
Enter Medicine Name: Paracetamol
Enter Quantity Available: 10
Enter Unit Price: 2.5
Medicine added successfully.

==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 1
Enter Medicine Name: Leozin
Enter Quantity Available: 3
Enter Unit Price: 5
Medicine added successfully.

==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 3

Current Inventory:
1. Paracetamol | Qty: 10 | Price: 2.50
2. Leozin | Qty: 3 | Price: 5.00

==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 4

Low Stock Medicines (Qty < 5):
- Leozin | Qty: 3
Total Inventory Value: 40.00
```

```
==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 2
Enter Medicine Name to remove: Paracetamol
Medicine removed successfully.

==== Medication Inventory Manager ====
1. Add Medicine
2. Remove Medicine
3. Display Inventory
4. Compute Inventory Value & Low-Stock Alerts
5. Save & Exit
Enter your choice: 5
Inventory saved to medicine_inventory.txt
Exiting program.

--------------------------------
Process exited after 173.7 seconds with return value 0
Press any key to continue . . .
```

```
1    Leozin,3,5.00
```