// 1. Student Grade Management System

// •    Problem Statement: Create a program to manage student grades. Use:

// o    A static variable to keep track of the total number of students processed.

// o    A const global variable for the maximum number of grades.

// o    A volatile variable to simulate an external grade update process.

// o    Use if-else and switch to determine grades based on marks and a for loop to process multiple students.

// •    Key Concepts Covered: Storage classes (static, volatile), Type qualifiers (const), Decision-making (if-else, switch), Looping (for).

// #include <stdio.h>

// const int MAX\_GRADES = 100;

// static int total\_students = 0;

// void calculateGrade(int marks);

// void simulateExternalGradeUpdate(volatile int \*marks);

// int main() {

//     int num\_students;

//     printf("Enter the number of students to process (max %d): ", MAX\_GRADES);

//     scanf("%d", &num\_students);

//     if (num\_students > MAX\_GRADES) {

//         printf("Exceeds maximum number of grades allowed (%d). Please try again.\n", MAX\_GRADES);

//         return 1;

//     }

//     for (int i = 0; i < num\_students; i++) {

//         int marks;

//         printf("\nEnter marks for student %d: ", i + 1);

//         scanf("%d", &marks);

//         volatile int external\_marks = marks;

//         simulateExternalGradeUpdate(&external\_marks);

//         printf("Processing updated marks: %d\n", external\_marks);

//         calculateGrade(external\_marks);

//         total\_students++;

//     }

//     printf("\nTotal students processed: %d\n", total\_students);

//     return 0;

// }

// void calculateGrade(int marks) {

//     char grade;

//     if (marks >= 90) {

//         grade = 'A';

//     } else if (marks >= 75) {

//         grade = 'B';

//     } else if (marks >= 50) {

//         grade = 'C';

//     } else {

//         grade = 'F';

//     }

//     // Switch-case for additional remarks

//     switch (grade) {

//         case 'A':

//             printf("Grade: A - Excellent!\n");

//             break;

//         case 'B':

//             printf("Grade: B - Good!\n");

//             break;

//         case 'C':

//             printf("Grade: C - Average!\n");

//             break;

//         case 'F':

//             printf("Grade: F - Needs Improvement!\n");

//             break;

//         default:

//             printf("Invalid grade.\n");

//     }

// }

// // Function to simulate an external grade update process

// void simulateExternalGradeUpdate(volatile int \*marks) {

//     // Simulate an update (e.g., moderation or error correction)

//     \*marks += 5; // Add 5 marks as an external adjustment

//     if (\*marks > 100) {

//         \*marks = 100; // Cap marks at 100

//     }

// }

// 2. Prime Number Finder

// •    Problem Statement: Write a program to find all prime numbers between 1 and a given number N. Use:

// o    A const variable for the upper limit N.

// o    A static variable to count the total number of prime numbers found.

// o    Nested for loops for the prime-checking logic.

// •    Key Concepts Covered: Type qualifiers (const), Storage classes (static), Looping (for).

// #include <stdio.h>

// int main() {

//     const int N;

//     static int total\_primes = 0;

//     printf("Enter the upper limit (N): ");

//     scanf("%d", &N);

//     printf("Prime numbers between 1 and %d are:\n", N);

//     for (int num = 2; num <= N; num++) {

//         int is\_prime = 1;

//         for (int i = 2; i \* i <= num; i++) {

//             if (num % i == 0) {

//                 is\_prime = 0;

//                 break;

//             }

//         }

//         if (is\_prime) {

//             printf("%d ", num);

//             total\_primes++;

//         }

//     }

//     printf("\n\nTotal prime numbers found: %d\n", total\_primes);

//     return 0;

// }

// 3. Dynamic Menu-Driven Calculator

// •    Problem Statement: Create a menu-driven calculator with options for addition, subtraction, multiplication, and division. Use:

// o    A static variable to track the total number of operations performed.

// o    A const pointer to hold operation names.

// o    A do-while loop for the menu and a switch case for operation selection.

// •    Key Concepts Covered: Storage classes (static), Type qualifiers (const), Decision-making (switch), Looping (do-while).

// #include <stdio.h>

// int main() {

//     static int total\_operations = 0;

//     int choice;

//     float num1, num2, result;

//     do {

//         printf("\nMenu-Driven Calculator\n");

//         printf("1. Addition\n");

//         printf("2. Subtraction\n");

//         printf("3. Multiplication\n");

//         printf("4. Division\n");

//         printf("5. Exit\n");

//         printf("Enter your choice: ");

//         scanf("%d", &choice);

//         if (choice >= 1 && choice <= 4) {

//             printf("Enter two numbers: ");

//             scanf("%f %f", &num1, &num2);

//         }

//         switch (choice) {

//             case 1:

//                 result = num1 + num2;

//                 printf("Result: %.2f\n", result);

//                 total\_operations++;

//                 break;

//             case 2:

//                 result = num1 - num2;

//                 printf("Result: %.2f\n", result);

//                 total\_operations++;

//                 break;

//             case 3:

//                 result = num1 \* num2;

//                 printf("Result: %.2f\n", result);

//                 total\_operations++;

//                 break;

//             case 4:

//                 if (num2 != 0) {

//                     result = num1 / num2;

//                     printf("Result: %.2f\n", result);

//                 } else {

//                     printf("Error: Division by zero is not allowed.\n");

//                 }

//                 total\_operations++;

//                 break;

//             case 5:

//                 printf("Exiting program.\n");

//                 break;

//             default:

//                 printf("Invalid choice. Please try again.\n");

//         }

//     } while (choice != 5);

//     printf("\nTotal operations performed: %d\n", total\_operations);

//     return 0;

// }

// 4. Configuration-Based Matrix Operations

// •    Problem Statement: Perform matrix addition and multiplication. Use:

// o    A const global variable to define the maximum size of the matrix.

// o    static variables to hold intermediate results.

// o    if statements to check for matrix compatibility.

// o    Nested for loops for matrix calculations.

// •    Key Concepts Covered: Type qualifiers (const), Storage classes (static), Decision-making (if), Looping (nested for).

#include <stdio.h>

#define MAX\_SIZE 10 // Maximum size of the matrix

int main() {

    int mat1[MAX\_SIZE][MAX\_SIZE], mat2[MAX\_SIZE][MAX\_SIZE], result[MAX\_SIZE][MAX\_SIZE];

    int rows, cols, choice;

    // Input matrix size

    printf("Enter the number of rows and columns for the matrices: ");

    scanf("%d %d", &rows, &cols);

    // Input matrices

    printf("Enter elements of Matrix 1:\n");

    for (int i = 0; i < rows; i++) {

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

            scanf("%d", &mat1[i][j]);

        }

    }

    printf("Enter elements of Matrix 2:\n");

    for (int i = 0; i < rows; i++) {

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

            scanf("%d", &mat2[i][j]);

        }

    }

    do {

        // Menu

        printf("\nMatrix Operations:\n");

        printf("1. Addition\n");

        printf("2. Multiplication\n");

        printf("3. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        if (choice == 1) {

            // Matrix Addition

            printf("\nMatrix Addition Result:\n");

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

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

                    result[i][j] = mat1[i][j] + mat2[i][j];

                    printf("%d ", result[i][j]);

                }

                printf("\n");

            }

        } else if (choice == 2) {

            // Matrix Multiplication

            printf("\nMatrix Multiplication Result:\n");

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

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

                    result[i][j] = 0;

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

                        result[i][j] += mat1[i][k] \* mat2[k][j];

                    }

                    printf("%d ", result[i][j]);

                }

                printf("\n");

            }

        } else if (choice != 3) {

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

        }

    } while (choice != 3);

    printf("Program exited.\n");

    return 0;

}

// 5. Temperature Monitoring System

// •    Problem Statement: Simulate a temperature monitoring system using:

// o    A volatile variable to simulate temperature input.

// o    A static variable to hold the maximum temperature recorded.

// o    if-else statements to issue warnings when the temperature exceeds thresholds.

// o    A while loop to continuously monitor and update the temperature.

// •    Key Concepts Covered: Storage classes (volatile, static), Decision-making (if-else), Looping (while).

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Simulate a volatile temperature input (changes frequently)

volatile int current\_temperature = 0;

// Function to generate a random temperature value

int generateTemperature() {

    return (rand() % 101) - 20; // Generates temperature between -20 to 80

}

int main() {

    static int max\_temperature = -100; // Static variable to hold the maximum temperature recorded

    int threshold\_warning = 50;       // Warning threshold

    int threshold\_critical = 70;      // Critical threshold

    srand(time(0)); // Seed for random number generation

    printf("Temperature Monitoring System\n");

    printf("Press Ctrl+C to exit the program.\n\n");

    while (1) {

        // Simulate temperature input

        current\_temperature = generateTemperature();

        // Update maximum temperature

        if (current\_temperature > max\_temperature) {

            max\_temperature = current\_temperature;

        }

        // Display current temperature and issue warnings if needed

        printf("Current Temperature: %d°C\n", current\_temperature);

        if (current\_temperature > threshold\_critical) {

            printf("CRITICAL WARNING: Temperature exceeds %d°C!\n", threshold\_critical);

        } else if (current\_temperature > threshold\_warning) {

            printf("WARNING: Temperature exceeds %d°C!\n", threshold\_warning);

        }

        printf("Maximum Temperature Recorded: %d°C\n\n", max\_temperature);

        // Delay for readability

        \_sleep(1000); // Sleep for 1 second (use sleep() on Linux)

    }

    return 0;

}

// 6. Password Validator

// •    Problem Statement: Implement a password validation program. Use:

// o    A static variable to count the number of failed attempts.

// o    A const variable for the maximum allowed attempts.

// o    if-else and switch statements to handle validation rules.

// o    A do-while loop to retry password entry.

// •    Key Concepts Covered: Storage classes (static), Type qualifiers (const), Decision-making (if-else, switch), Looping (do-while).

#include <stdio.h>

#include <string.h>

// Function to check password validity

int validatePassword(const char \*password) {

    int length = strlen(password);

    int hasUpper = 0, hasLower = 0, hasDigit = 0;

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

        if (password[i] >= 'A' && password[i] <= 'Z')

            hasUpper = 1;

        else if (password[i] >= 'a' && password[i] <= 'z')

            hasLower = 1;

        else if (password[i] >= '0' && password[i] <= '9')

            hasDigit = 1;

    }

    if (length >= 8 && hasUpper && hasLower && hasDigit) {

        return 1; // Password is valid

    } else {

        return 0; // Password is invalid

    }

}

int main() {

    static int failed\_attempts = 0; // Static variable to track failed attempts

    const int max\_attempts = 3;    // Const variable for the maximum allowed attempts

    char password[20];

    int is\_valid;

    printf("Password Validator\n");

    printf("Rules: At least 8 characters, 1 uppercase letter, 1 lowercase letter, and 1 digit.\n");

    do {

        printf("Enter your password: ");

        scanf("%s", password);

        is\_valid = validatePassword(password);

        if (is\_valid) {

            printf("Password is valid. Access granted!\n");

            break;

        } else {

            printf("Invalid password. Try again.\n");

            failed\_attempts++;

        }

        switch (failed\_attempts) {

            case 1:

                printf("Hint: Check the password length.\n");

                break;

            case 2:

                printf("Hint: Ensure it includes uppercase, lowercase, and a digit.\n");

                break;

            default:

                break;

        }

    } while (failed\_attempts < max\_attempts);

    if (failed\_attempts == max\_attempts) {

        printf("Maximum attempts reached. Access denied.\n");

    }

    return 0;

}

// 7. Bank Transaction Simulator

// •    Problem Statement: Simulate bank transactions. Use:

// o    A static variable to maintain the account balance.

// o    A const variable for the maximum withdrawal limit.

// o    if-else statements to check transaction validity.

// o    A do-while loop for performing multiple transactions.

// •    Key Concepts Covered: Storage classes (static), Type qualifiers (const), Decision-making (if-else), Looping (do-while).

#include <stdio.h>

// Function prototypes

void deposit(float amount);

void withdraw(float amount);

// Static variable to maintain the account balance

static float account\_balance = 1000.0; // Initial balance

int main() {

    const float max\_withdrawal\_limit = 500.0; // Maximum withdrawal limit per transaction

    int choice;

    float amount;

    printf("Welcome to Bank Transaction Simulator!\n");

    printf("Initial Account Balance: ₹%.2f\n\n", account\_balance);

    do {

        // Display menu

        printf("Menu:\n");

        printf("1. Deposit\n");

        printf("2. Withdraw\n");

        printf("3. Check Balance\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        if (choice == 1) {

            // Deposit transaction

            printf("Enter amount to deposit: ₹");

            scanf("%f", &amount);

            deposit(amount);

        } else if (choice == 2) {

            // Withdraw transaction

            printf("Enter amount to withdraw: ₹");

            scanf("%f", &amount);

            if (amount > max\_withdrawal\_limit) {

                printf("Error: Withdrawal amount exceeds the limit of ₹%.2f\n", max\_withdrawal\_limit);

            } else if (amount > account\_balance) {

                printf("Error: Insufficient balance.\n");

            } else {

                withdraw(amount);

            }

        } else if (choice == 3) {

            // Check balance

            printf("Current Account Balance: ₹%.2f\n", account\_balance);

        } else if (choice != 4) {

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

        }

    } while (choice != 4);

    printf("Thank you for using the Bank Transaction Simulator!\n");

    return 0;

}

// Function to deposit money

void deposit(float amount) {

    if (amount > 0) {

        account\_balance += amount;

        printf("₹%.2f deposited successfully! Current Balance: ₹%.2f\n", amount, account\_balance);

    } else {

        printf("Invalid deposit amount. Please try again.\n");

    }

}

// Function to withdraw money

void withdraw(float amount) {

    if (amount > 0) {

        account\_balance -= amount;

        printf("₹%.2f withdrawn successfully! Current Balance: ₹%.2f\n", amount, account\_balance);

    } else {

        printf("Invalid withdrawal amount. Please try again.\n");

    }

}

// 8. Digital Clock Simulation

// •    Problem Statement: Simulate a digital clock. Use:

// o    volatile variables to simulate clock ticks.

// o    A static variable to count the total number of ticks.

// o    Nested for loops for hours, minutes, and seconds.

// o    if statements to reset counters at appropriate limits.

// •    Key Concepts Covered: Storage classes (volatile, static), Decision-making (if), Looping (nested for).

 #include <stdio.h>

#include <unistd.h> // For sleep function (Linux/macOS)

// Function prototypes

void simulateClock();

int main() {

    printf("Digital Clock Simulation\n");

    simulateClock(); // Call the function to simulate the clock

    return 0;

}

void simulateClock() {

    volatile int hours = 0, minutes = 0, seconds = 0; // Volatile variables to simulate clock ticks

    static int total\_ticks = 0; // Static variable to count total number of ticks

    while (1) {

        // Display current time

        printf("\r%02d:%02d:%02d", hours, minutes, seconds);

        fflush(stdout); // Ensures that the output is printed immediately

        // Simulate 1-second tick

        sleep(1); // Sleep for 1 second

        // Increment the seconds

        seconds++;

        // Check if seconds exceed 59 and reset, and increment minutes

        if (seconds >= 60) {

            seconds = 0;

            minutes++;

        }

        // Check if minutes exceed 59 and reset, and increment hours

        if (minutes >= 60) {

            minutes = 0;

            hours++;

        }

        // Check if hours exceed 23 and reset

        if (hours >= 24) {

            hours = 0;

        }

        // Increment the total\_ticks counter

        total\_ticks++;

        // Optional: Display the total ticks every 60 seconds (1 minute)

        if (total\_ticks % 60 == 0) {

            printf("\nTotal Ticks: %d\n", total\_ticks);

        }

    }

}

// 9. Game Score Tracker

// •    Problem Statement: Track scores in a simple game. Use:

// o    A static variable to maintain the current score.

// o    A const variable for the winning score.

// o    if-else statements to decide if the player has won or lost.

// o    A while loop to play rounds of the game.

// •    Key Concepts Covered: Storage classes (static), Type qualifiers (const), Decision-making (if-else), Looping (while).

#include <stdio.h>

// Function prototypes

void playRound();

int main() {

    printf("Game Score Tracker\n");

    playRound();  // Start playing rounds of the game

    return 0;

}

void playRound() {

    static int current\_score = 0; // Static variable to maintain current score

    const int winning\_score = 10; // Const variable for the winning score

    int points;

    printf("Target winning score: %d\n", winning\_score);

    printf("Current score: %d\n", current\_score);

    while (current\_score < winning\_score) {

        // Simulate playing a round and adding points to score

        printf("\nEnter points earned this round (1-5): ");

        scanf("%d", &points);

        // Check if points are valid

        if (points < 1 || points > 5) {

            printf("Invalid points! Enter a value between 1 and 5.\n");

            continue; // Skip this round and prompt again

        }

        // Add points to the current score

        current\_score += points;

        // Display updated score

        printf("Current score: %d\n", current\_score);

        // Check if the player has won

        if (current\_score >= winning\_score) {

            printf("Congratulations! You have won the game with %d points!\n", current\_score);

            break;

        }

    }

    if (current\_score < winning\_score) {

        printf("Game Over. Try again!\n");

    }

}