1. Student Grade Management System

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

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

A const global variable for the maximum number of grades.

A volatile variable to simulate an external grade update process.

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>

#define MAX_GRADES 5

static int studentCount = 0;

volatile int externalGradeUpdate = 0;

const int MAX_MARKS = 100;

void processGrades() {
   int marks[MAX_GRADES];
   char grade;
   for (int i = 0; i < MAX_GRADES; i++) {
      printf("Enter marks for student %d: ", i + 1);
      scanf("%d", &marks[i]);
      if (marks[i] >= 90) {
         grade = 'A';
      } else if (marks[i] >= 75) {
         grade = 'B';
      } else if (marks[i] >= 50) {
         grade = 'C';
      } else {
         grade = 'F';
      }
      printf("Grade for student %d: %c\n", i + 1, grade);
      studentCount++;
   }
```

```
    if (externalGradeUpdate) {
        printf("External grade update received.\n");
    }
}

int main() {
    processGrades();
    printf("Total students processed: %d\n", studentCount);
    return 0;
}
```

Output:

```
Enter marks for student 1: 67
Grade for student 1: C
Enter marks for student 2: 98
Grade for student 2: A
Enter marks for student 3: 23
Grade for student 3: F
```

2. Prime Number Finder

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

A const variable for the upper limit N.

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

Nested for loops for the prime-checking logic.

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

```
#include <stdio.h>
const int N = 100;
static int primeCount = 0;
int isPrime(int num) {
    for (int i = 2; i * i <= num; i++) {
```

```
        if (num % i == 0)
            return 0;
    }
    return 1;
}
int main() {
    for (int i = 2; i <= N; i++) {
        if (isPrime(i)) {
            printf("%d ", i);
            primeCount++;
        }
    }
    printf("\nTotal prime numbers found: %d\n", primeCount);
    return 0;
}
```

Output:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Total prime numbers found: 25

3. Dynamic Menu-Driven Calculator

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

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

A const pointer to hold operation names.

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>
static int operationCount = 0;
const char *operations[] = {"Addition", "Subtraction", "Multiplication", "Division"};
void addition() {
    int a, b;
```

```
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("Result: %d\n", a + b);
}
void subtraction() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("Result: %d\n", a - b);
}
void multiplication() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("Result: %d\n", a * b);
}
void division() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    if (b != 0) {
        printf("Result: %.2f\n", (float)a / b);
    } else {
        printf("Division by zero error\n");
    }
}
int main() {
    int choice;
    do {
        printf("Select operation:\n1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5.
Exit\n");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1: addition(); break;
            case 2: subtraction(); break;
            case 3: multiplication(); break;
            case 4: division(); break;
            case 5: break;
            default: printf("Invalid choice\n");
        }
        operationCount++;
    } while (choice != 5);
    printf("Total operations performed: %d\n", operationCount);
    return 0;
}
```

Output:

Select operation:

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

3

Enter two numbers: 23 13

Result: 299

4. Configuration-Based Matrix Operations

Problem Statement: Perform matrix addition and multiplication. Use:

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

static variables to hold intermediate results.

if statements to check for matrix compatibility.

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

static int result_add[MAX_SIZE][MAX_SIZE];

static int result_mul[MAX_SIZE][MAX_SIZE];

void matrix_addition(int A[MAX_SIZE][MAX_SIZE], int B[MAX_SIZE][MAX_SIZE], int rows, int cols) {
    if (rows <= MAX_SIZE && cols <= MAX_SIZE) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result_add[i][j] = A[i][j] + B[i][j];
            }
        }
    } else {
        printf("Matrix dimensions are incompatible for addition.\n");
    }
}

void matrix_multiplication(int A[MAX_SIZE][MAX_SIZE], int B[MAX_SIZE][MAX_SIZE], int A_rows, int A_cols, int B_rows, int B_cols) {
    if (A_cols == B_rows) {
        for (int i = 0; i < A_rows; i++) {
            for (int j = 0; j < B_cols; j++) {
                result_mul[i][j] = 0; // Clear previous values
                for (int k = 0; k < A_cols; k++) {
                    result_mul[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    } else {
        printf("Matrix dimensions are incompatible for multiplication.\n");
    }
}
```

```
void print_matrix(int matrix[MAX_SIZE][MAX_SIZE], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int rowsA = 3, colsA = 3;
    int rowsB = 3, colsB = 3;
    int A[MAX_SIZE][MAX_SIZE] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int B[MAX_SIZE][MAX_SIZE] = {
        {9, 8, 7},
        {6, 5, 4},
        {3, 2, 1}
    };
    matrix_addition(A, B, rowsA, colsA);
    printf("Matrix Addition Result:\n");
    print_matrix(result_add, rowsA, colsA);
    matrix_multiplication(A, B, rowsA, colsA, rowsB, colsB);
    printf("\nMatrix Multiplication Result:\n");
    print_matrix(result_mul, rowsA, colsB);
    return 0;
}
```

Output:

Matrix Addition Result:

10 10 10

10 10 10

10 10 10

Matrix Multiplication Result:

30 24 18

84 69 54

138 114 90

5. Temperature Monitoring System

Problem Statement: Simulate a temperature monitoring system using:

A volatile variable to simulate temperature input.

A static variable to hold the maximum temperature recorded.

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

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>
volatile int currentTemp = 0;
static int maxTemp = -100;
void checkTemperature() {
    if (currentTemp > maxTemp) {
        maxTemp = currentTemp;
    }
    if (currentTemp > 30) {
        printf("Warning: High Temperature\n");
    } else if (currentTemp < 0) {
        printf("Warning: Low Temperature\n");
    }
}
int main() {
```

```
    while (1) {
        printf("Enter current temperature: ");
        scanf("%d", &currentTemp);
        checkTemperature();
        printf("Max Temperature Recorded: %d\n", maxTemp);
    }
    return 0;
}
```

Output:

Enter current temperature: 345

Warning: High Temperature

Max Temperature Recorded: 345

6. Password Validator

Problem Statement: Implement a password validation program. Use:

A static variable to count the number of failed attempts.

A const variable for the maximum allowed attempts.

if-else and switch statements to handle validation rules.

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>
static int failedAttempts = 0;
const int MAX_ATTEMPTS = 3;
int validatePassword(char *password) {
    return strcmp(password, "password123") == 0;
}
int main() {
    char password[50];
    int attempts = 0;
```

```
    do {
        printf("Enter password: ");
        scanf("%s", password);
        if (validatePassword(password)) {
            printf("Password validated\n");
            break;
        } else {
            printf("Invalid password\n");
            failedAttempts++;
            attempts++;
        }
    } while (failedAttempts < MAX_ATTEMPTS);
    if (failedAttempts == MAX_ATTEMPTS) {
        printf("Max attempts reached\n");
    }
    return 0;
}
```

Output:

```
Enter password: sd2re245
Invalid password
Enter password: 123
Invalid password
Enter password: password123
Password validated
```

7. Bank Transaction Simulator

Problem Statement: Simulate bank transactions. Use:

A static variable to maintain the account balance.

A const variable for the maximum withdrawal limit.

if-else statements to check transaction validity.

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>
static int accountBalance = 1000;
const int MAX_WITHDRAWAL = 500;

void deposit(int amount) {
    accountBalance += amount;
}
void withdraw(int amount) {
    if (amount <= accountBalance && amount <= MAX_WITHDRAWAL) {
        accountBalance -= amount;
    } else {
        printf("Transaction failed\n");
    }
}
int main() {
    int choice, amount;
    do {
        printf("Enter 1 for Deposit, 2 for Withdraw, 3 for Exit: ");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Enter deposit amount: ");
            scanf("%d", &amount);
            deposit(amount);
        } else if (choice == 2) {
            printf("Enter withdrawal amount: ");
            scanf("%d", &amount);
            withdraw(amount);
        }
        printf("Account balance: %d\n", accountBalance);
```

```
    } while (choice != 3);
    return 0;
}
```

Output:

```
Enter 1 for Deposit, 2 for Withdraw, 3 for Exit: 1
Enter deposit amount: 1000
Account balance: 2000
Enter 1 for Deposit, 2 for Withdraw, 3 for Exit: 2
Enter withdrawal amount: 500
Account balance: 1500
Enter 1 for Deposit, 2 for Withdraw, 3 for Exit: 3
Account balance: 1500
```

8. Digital Clock Simulation

Problem Statement: Simulate a digital clock. Use:

volatile variables to simulate clock ticks.

A static variable to count the total number of ticks.

Nested for loops for hours, minutes, and seconds.

if statements to reset counters at appropriate limits.

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

```
#include <stdio.h>
volatile int seconds = 0, minutes = 0, hours = 0;
static int tickCount = 0;
void updateClock() {
    seconds++;
    if (seconds == 60) {
        seconds = 0;
        minutes++;
        if (minutes == 60) {
            minutes = 0;
```

```
            hours++;
            if (hours == 24) {
                hours = 0;
            }
        }
    }
    tickCount++;
}
int main() {
    while (1) {
        updateClock();
        printf("%02d:%02d:%02d\n", hours, minutes, seconds);
        if (tickCount >= 10) break;
    }
    printf("Total ticks: %d\n", tickCount);
    return 0;
}
```

Output:

00:00:01

00:00:02

00:00:03

00:00:04

00:00:05

00:00:06

00:00:07

00:00:08

00:00:09

00:00:10

Total ticks: 10

9. Game Score Tracker

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

A static variable to maintain the current score.

A const variable for the winning score.

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

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>

static int currentScore = 0;

const int WINNING_SCORE = 10;

int main() {
   int roundScore;
   while (currentScore < WINNING_SCORE) {
      printf("Enter score for the round: ");
      scanf("%d", &roundScore);
      currentScore += roundScore;
      if (currentScore >= WINNING_SCORE) {
         printf("You win!\n");
      } else {
         printf("Current score: %d\n", currentScore);
      }
   }
   return 0;
}
```

Output:

```
Enter score for the round: 8
Current score: 8
Enter score for the round: 9
You win!
```