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>


const int MAX_GRADES = 100;


volatile int externalGradeUpdate = 0;


static int totalStudentsProcessed = 0;


char getGrade(int marks)
{
  if (marks >= 90)
  {
    return 'A';
  } else if (marks >= 80)
  {
    return 'B';
  } else if (marks >= 70)
  {
    return 'C';
  } else if (marks >= 60)
  {
    return 'D';
  } else {
    return 'F';
  }
}

int main() {
  int numberOfStudents;

  printf("Enter the number of students: ");
  scanf("%d", &numberOfStudents);

  if (numberOfStudents > MAX_GRADES) {
    printf("Maximum number of students allowed is %d.\n", MAX_GRADES);
    numberOfStudents = MAX_GRADES;
  }

  for (int i = 0; i < numberOfStudents; ++i)
  {
    int marks;
    printf("Enter marks for student %d: ", i + 1);
    scanf("%d", &marks);

    char grade = getGrade(marks);

    printf("Student %d has grade: %c\n", i + 1, grade);

    if (externalGradeUpdate != 0)
    {
        printf("External grade update detected. Adjusting marks...\n");
        marks += externalGradeUpdate;
        printf("Adjusted marks: %d\n", marks);
        grade = getGrade(marks);
        printf("Adjusted grade for student %d: %c\n", i + 1, grade);
        externalGradeUpdate = 0;
    }

    totalStudentsProcessed++;
  }

  printf("\nTotal students processed: %d\n", totalStudentsProcessed);

  return 0;
}
```

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>

int main() {
    const int N = 50;

    static int Count = 0;



    for (int i = 2; i <= N; ++i)
    {
        int isPrime = 1;

        for (int j = 2; j * j <= i; ++j)
        {
            if (i % j == 0) {
                isPrime = 0;
                break;
            }
        }

        if (isPrime) {
            printf("%d ", i);
            Count++;
        }
    }
    printf("\nTotal prime numbers: %d\n", Count);

    return 0;
}
```

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;
```

```
void add();
void sub();
void mul();
void divi();
void add()
{
   int num1, num2;
printf("Enter two numbers to add: ");
scanf("%d %d", &num1, &num2);
printf("Result: %d\n", num1 + num2);
operationCount++;
}
void sub()
{
   int num1, num2;
printf("Enter two numbers to subtract: ");
scanf("%d %d", &num1, &num2);
printf("Result: %d\n", num1 - num2);
operationCount++;
}
void mul()
{
   int num1, num2;
   printf("Enter two numbers to multiply: ");
   scanf("%d %d", &num1, &num2);
   printf("Result: %d\n", num1 * num2);
   operationCount++;
   }
   void divi()
   {
      int num1, num2;
```

```
    printf("Enter two numbers to divide: ");
    scanf("%d %d", &num1, &num2);

    if (num2 == 0)
    {
        printf("Error: Division by zero is not allowed.\n");
    }
    else
    {
        printf("Result: %d\n", num1 / num2);
        }
    }
int main()


{
const char *menuOptions = "1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5. Exit\n";
int choice;
do {

     printf("Menu:\n%s", menuOptions);

    printf("Enter your choice: "); scanf("%d", &choice);

    switch (choice)
    {
    case 1: add();
    break;
    case 2: sub();
    break;
    case 3: mul();
    break;
    case 4: divi();
    break;

    default:
    printf("Invalid choice. Please try again.\n");
    }
   } while (choice != 5);

    printf("Total number of operations performed: %d\n", operationCount);



    return 0;
}
```

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

int A[MAX_SIZE][MAX_SIZE], B[MAX_SIZE][MAX_SIZE], C[MAX_SIZE][MAX_SIZE],
D[MAX_SIZE][MAX_SIZE];

void matrix_addition(int rows, int cols) {
    static int sum[MAX_SIZE][MAX_SIZE];

    if (rows <= MAX_SIZE && cols <= MAX_SIZE) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sum[i][j] = A[i][j] + B[i][j];
            }
        }
        printf("\nMatrix Addition Result:\n");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                printf("%d ", sum[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Matrix dimensions are incompatible for addition.\n");
    }
}

void matrix_multiplication(int rows_A, int cols_A, int rows_B, int cols_B) {
    static int product[MAX_SIZE][MAX_SIZE];

    if (cols_A == rows_B) {
        for (int i = 0; i < rows_A; i++) {
            for (int j = 0; j < cols_B; j++) {
                product[i][j] = 0;
                for (int k = 0; k < cols_A; k++) {
                    product[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        printf("\nMatrix Multiplication Result:\n");
        for (int i = 0; i < rows_A; i++) {
            for (int j = 0; j < cols_B; j++) {
                printf("%d ", product[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Matrix dimensions are incompatible for multiplication.\n");
    }
}

int main() {
    int rows_A, cols_A, rows_B, cols_B;

    printf("Enter number of rows and columns for matrix A: \n");
    scanf("%d %d", &rows_A, &cols_A);

    printf("Enter elements of matrix A:\n");
    for (int i = 0; i < rows_A; i++) {
        for (int j = 0; j < cols_A; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    printf("Enter number of rows and columns for matrix B: ");
    scanf("%d %d", &rows_B, &cols_B);

    printf("Enter elements of matrix B:\n");
    for (int i = 0; i < rows_B; i++) {
        for (int j = 0; j < cols_B; j++) {
            scanf("%d", &B[i][j]);
        }
    }

    if (rows_A == rows_B && cols_A == cols_B) {
        matrix_addition(rows_A, cols_A);
    } else {
        printf("Matrix dimensions are incompatible for addition.\n");
    }

    matrix_multiplication(rows_A, cols_A, rows_B, cols_B);

    return 0;
}
```

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>
#include <stdlib.h>
#include <time.h>

volatile int currentTemperature;
int generateTemperature()
```

```
{
   return rand() % 100;
}
void monitorTemperature()
{
   static int maxTemperature = 0;
   const int highThreshold = 70;
   const int criticalThreshold = 90;
   while (1)
   {
   currentTemperature = generateTemperature();
   if (currentTemperature > maxTemperature)
   {
      maxTemperature = currentTemperature;
      }
      if (currentTemperature >= criticalThreshold)
      { printf("Critical Warning! Current temperature: %d°C\n", currentTemperature);
      }
      else if (currentTemperature >= highThreshold)
      { printf("High Temperature Warning. Current temperature: %d°C\n", currentTemperature);
      }
      else {
         printf("Temperature is normal. Current temperature: %d°C\n", currentTemperature);


      }
      printf("Maximum temperature recorded so far: %d°C\n", maxTemperature);


      sleep(1);
      }
      }
```

```
    int main()
    {
       srand(time(0));
       monitorTemperature();
       return 0;
       }
```

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>
const int MAX_ATTEMPTS = 3;

int validatePassword(char password[])
{
   if (strcmp(password, "password123") == 0)
   {
       return 1;
   } else {
       return 0;
   }
}
```

```
int main()
{
    static int failedAttempts = 0;
    char password[50];
    int isValid;

    do {
        printf("Enter your password: ");
        scanf("%s", password);

        isValid = validatePassword(password);

        if (isValid) {
            printf("Password is correct!\n");
            break;
        } else {
            printf("Incorrect password. Try again.\n");
            failedAttempts++;
        }

        switch (failedAttempts) {
            case 1:
                printf("Warning: You have 2 attempts left.\n");
                break;
            case 2:
                printf("Warning: You have 1 attempt left.\n");
                break;
            case 3:
                printf("Error: Maximum attempts reached. Access denied.\n");
                break;
        }
    } while (failedAttempts < MAX_ATTEMPTS);

    return 0;
}
```

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>

const int MAX_WITHDRAWAL_LIMIT = 1000;

void deposit(float amount);
void withdraw(float amount);
void displayBalance();
static float accountBalance = 0.0;

int main()
{
    int choice;
    float amount;
    do {
    printf("\nBank Transaction Menu:\n");
    printf("1. Deposit\n");
    printf("2. Withdraw\n");
    printf("3. Display Balance\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
       case 1: printf("Enter the amount to deposit: ");
       scanf("%f", &amount);
       deposit(amount);
       break;
       case 2: printf("Enter the amount to withdraw: ");
       scanf("%f", &amount);
       withdraw(amount);
       break;
       case 3:
       displayBalance();
       break;
       case 4:
       printf("Exiting the transaction menu...\n");
       break;
       default: printf("Invalid choice. Please try again.\n");

    }
    } while (choice != 4);
    return 0;
    }
    void deposit(float amount)
    {
       if (amount > 0)
    {
       accountBalance += amount;
```

```
        printf("Successfully deposited %.2f. Current balance: %.2f\n", amount, accountBalance);
        }
        else {
            printf("Invalid deposit amount.\n");
        }
    }
    void withdraw(float amount)
    {
        if (amount > 0 && amount <= MAX_WITHDRAWAL_LIMIT)
        {
            if (amount <= accountBalance)
            {
                accountBalance -= amount;
                printf("Successfully withdrew %.2f. Current balance: %.2f\n", amount, accountBalance);
                }
                else
                {
                    printf("Insufficient balance for the withdrawal.\n");
                    }
                    }
                    else {
                        printf("Invalid withdrawal amount. The amount must be positive and not exceed the
maximum withdrawal limit of %d.\n", MAX_WITHDRAWAL_LIMIT);


                    }
                    }
                    void displayBalance()
                    {
                        printf("Current account balance: %.2f\n", accountBalance);
                        }
```

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 tick = 0;
void simulateTick()
{
  tick = 1;
}
void digitalClock()
{


  static int totalTicks = 0;
  for (int hours = 0; hours < 24; hours++)
  {
    for (int minutes = 0; minutes < 60; minutes++)
    {


      for (int seconds = 0; seconds < 60; seconds++)
      {
      simulateTick();
      totalTicks++;
      tick = 0;
      printf("Time: %02d:%02d:%02d\n", hours, minutes, seconds);


      }
```

```
        }
        }
        printf("Total number of ticks: %d\n", totalTicks);
        }
        int main()
        {

          digitalClock();
          return 0;
          }
```

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>
const int WINNING_SCORE = 50;
void playGame();

int main()
{

  playGame();
  return 0;
}
```

```
void playGame()
{
   static int currentScore = 0;
   int points;

   printf("Welcome to the game! Reach %d points to win.\n", WINNING_SCORE);

   while (1)
   {

      printf("\nEnter points scored in this round (positive to gain, negative to lose): ");
      scanf("%d", &points);


      currentScore += points;


      if (currentScore >= WINNING_SCORE)
      {
         printf("\nCongratulations! You have won the game with %d points!\n", currentScore);
         break;
      } else if (currentScore < 0)
      {
         printf("\nSorry! You have lost the game with %d points.\n", currentScore);
         break;
      } else {
         printf("Current Score: %d. Keep playing!\n", currentScore);
      }
   }
}
```