ASSIGNMENT

---

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’; // Excellent
    } else if (marks >= 80) {
        return ’B’; // Good
    } else if (marks >= 70) {
        return ’C’; // Average
    } else if (marks >= 60) {
        return ’D’; // Pass
    } else {
        return ’F’; // Fail
    }
}

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 = 100;

    static int primeCount = 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);
            primeCount++;
        }
    }
    printf("\nTotal prime numbers found: %d\n", primeCount);

    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>

#define ADDITION "Addition"
#define SUBTRACTION "Subtraction"
#define MULTIPLICATION "Multiplication"
#define DIVISION "Division"

static int operationCount = 0;

const char* operationNames[] = { ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION };

void displayMenu() {
    printf("\nMenu:\n");
    printf("1. %s\n", operationNames[0]);
    printf("2. %s\n", operationNames[1]);
    printf("3. %s\n", operationNames[2]);
    printf("4. %s\n", operationNames[3]);
    printf("5. Exit\n");
}

void performOperation(int choice) {
    double num1, num2, result;

    printf("Enter two numbers: ");
    scanf("%lf %lf", &num1, &num2);

    switch (choice) {
    case 1:
        result = num1 + num2;
        printf("Result: %.2f\n", result);
        break;
    case 2:
        result = num1 - num2;
        printf("Result: %.2f\n", result);
        break;
    case 3:
        result = num1 * num2;
        printf("Result: %.2f\n", result);
        break;
    case 4:
        if (num2 != 0) {
            result = num1 / num2;
            printf("Result: %.2f\n", result);
        } else {
            printf("Error: Division by zero!\n");
        }
        break;
```

```
    default:
        printf("Invalid choice. Try again!\n");
        return;
    }

    operationCount++;
}

void displayOperationCount() {
    printf("Total operations performed: %d\n", operationCount);
}

int main() {
    int choice;

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

        if (choice != 5) {
            performOperation(choice);
        }

        displayOperationCount();

    } while (choice != 5);

    printf("Exiting program. Goodbye!\n");
    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: ");
  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;

static int maxTemperature = -999;

int readTemperature() {
    return (rand() % 61) - 10;
}

int main() {
    srand(time(NULL));

    while(1) {
        currentTemperature = readTemperature();

        if(currentTemperature > 30) {
            printf("Warning: High Temperature! Current: %d°C\n", currentTemperature);
        } else if(currentTemperature < 0) {
            printf("Warning: Low Temperature! Current: %d°C\n", currentTemperature);
        } else {
            printf("Temperature is normal. Current: %d°C\n", currentTemperature);
        }
```

```
        if(currentTemperature > maxTemperature) {
            maxTemperature = currentTemperature;
            printf("New Max Temperature Recorded: %d°C\n", maxTemperature);
        }

        sleep(1);
    }

    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) {
    const char correctPassword[] = "Secure123";

    if (strcmp(password, correctPassword) == 0) {
        return 1;
        } else {
        return 0;
    }
}

int main() {
    static int failedAttempts = 0;

    char password[50];
    int valid = 0;

    do {
        printf("Enter password (Attempt %d of %d): ", failedAttempts + 1, MAX_ATTEMPTS);
        scanf("%s", password);

        valid = validatePassword(password);

        if (!valid) {
            failedAttempts++;
            printf("Incorrect password. You have %d attempts left.\n", MAX_ATTEMPTS - failedAttempts);
        }
    } while (!valid && failedAttempts < MAX_ATTEMPTS);
```

```
    if (valid) {
        printf("Password accepted! Access granted.\n");
    } else {
        printf("Maximum attempts reached. Access denied.\n");
    }

    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>

void deposit(double *balance, double amount) {
    if (amount > 0) {
        *balance += amount;
        printf("Deposited: %.2f\n", amount);
    } else {
        printf("Deposit amount must be greater than 0.\n");
    }
}

void withdraw(double *balance, double amount, const double MAX_WITHDRAWAL_LIMIT) {
    if (amount > 0 && amount <= *balance && amount <= MAX_WITHDRAWAL_LIMIT) {
        *balance -= amount;
        printf("Withdrew: %.2f\n", amount);
    } else if (amount > MAX_WITHDRAWAL_LIMIT) {
        printf("Amount exceeds maximum withdrawal limit of %.2f\n", MAX_WITHDRAWAL_LIMIT);
    } else if (amount > *balance) {
        printf("Insufficient funds. Available balance: %.2f\n", *balance);
    } else {
        printf("Invalid withdrawal amount.\n");
    }
}

void displayBalance(double balance) {
    printf("Current Balance: %.2f\n", balance);
}

int main() {
    static double balance = 0.0;
    const double MAX_WITHDRAWAL_LIMIT = 1000.0;

    int choice;
    double amount;

    do {
```

```
        printf("\n--- Bank Transaction 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);

        switch (choice) {
            case 1:
                printf("Enter deposit amount: ");
                scanf("%lf", &amount);
                deposit(&balance, amount);
                break;
            case 2:
                printf("Enter withdrawal amount: ");
                scanf("%lf", &amount);
                withdraw(&balance, amount, MAX_WITHDRAWAL_LIMIT);
                break;
            case 3:
                displayBalance(balance);
                break;
            case 4:
                printf("Exiting the program...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

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>
#include <unistd.h>

volatile int hour = 0;
volatile int minute = 0;
volatile int second = 0;

static int total_ticks = 0;

int main() {
    while (1) {
        sleep(1);
```

```
        total_ticks++;

        second++;

        if (second >= 60) {
            second = 0;
            minute++;
        }

        if (minute >= 60) {
            minute = 0;
            hour++;
        }

        if (hour >= 24) {
            hour = 0;
        }

        printf("%02d:%02d:%02d\n", hour, minute, second);

}

    return 0;
}
```

9.Game Score Tracker

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 = 10;

int playRound() {
    int roundScore;
    printf("Enter score for this round (1 for win, 0 for lose): ");
    scanf("%d", &roundScore);
    return roundScore;
}

int main() {
    static int currentScore = 0;

    while (currentScore < WINNING_SCORE) {
        printf("\nCurrent score: %d\n", currentScore);

        if (playRound() == 1) {
            currentScore++;
        }
```

```
        if (currentScore >= WINNING_SCORE) {
            printf("\nCongratulations! You've won the game with a score of %d!\n", currentScore);
            break;
        } else {
            printf("Keep playing! Your score is %d.\n", currentScore);
        }
    }

    return 0;
}
```