Assignment 6

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.

```
Sol: #include <stdio.h>
        const int MAX_GRADES = 5;
volatile int externalGradeUpdate = 0;
static int totalStudentsProcessed = 0;
char determineGrade(int marks) {
    char grade;
    if (marks >= 90) {
        grade = 'A';
    } else if (marks >= 80) {
        grade = 'B';
    } else if (marks >= 70) {
        grade = 'C';
    } else if (marks >= 60) {
        grade = 'D';
    } else {
        grade = 'F';
    }
    switch (grade) {
        case 'A':
        case 'B':
        case 'C':
        case 'D':
            printf("Pass Grade: %c\n", grade);
            break;
        case 'F':
            printf("Fail Grade: %c\n", grade);
            break;
        default:
            printf("Invalid Grade\n");
    }

    return grade;
}

int main() {
    int numberOfStudents;
    printf("Enter the number of students: ");
    scanf("%d", &numberOfStudents);
   char studentNames[numberOfStudents][50];
    int marks[numberOfStudents];
    char grades[numberOfStudents];
    for (int i = 0; i < numberOfStudents; i++) {
        printf("Enter the name of student %d: ", i + 1);
        scanf("%s", studentNames[i]);
     printf("Enter marks of %s: ", studentNames[i]);
        scanf("%d", &marks[i]);
        grades[i] = determineGrade(marks[i]);
        if (externalGradeUpdate) {
            marks[i] += 5; // Example: Adding 5 marks externally
            grades[i] = determineGrade(marks[i]);
        }
        totalStudentsProcessed++;
    }
    printf("\n--- Student Grades ---\n");
    for (int i = 0; i < numberOfStudents; i++) {
        printf("Student: %s, Marks: %d, Grade: %c\n", studentNames[i],
marks[i], grades[i]);
    }
    printf("\nTotal students processed: %d\n", totalStudentsProcessed);
    return 0;
}
```

```
O/p: Enter the number of students: 3
Enter the name of student 1: likitha
Enter marks of likitha: 85
Pass Grade: B
Enter the name of student 2: john
Enter marks of john: 90
Pass Grade: A
Enter the name of student 3: xyz
Enter marks of xyz: 35
Fail Grade: F

--- Student Grades ---
Student: likitha, Marks: 85, Grade: B
Student: john, Marks: 90, Grade: A
Student: xyz, Marks: 35, Grade: F

Total students processed: 3
```

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.

```
Sol: #include <stdio.h>
     int main() {
    const int N = 100;
    static int primeCount = 0;
    for (int num = 2; num <= N; num++) {
        int isPrime = 1;
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) {
                isPrime = 0;
                break;
            }
        }
        if (isPrime) {
            printf("%d ", num);
            primeCount++;
        }
    }
    printf("\nTotal number of primes found: %d\n", primeCount);

    return 0;
}
```

```
O/p: 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 number of primes 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.

```
Sol: #include <stdio.h>
    int main() {
    static int operationCount = 0;
    const char *operations[] = {"Addition", "Subtraction", "Multiplication", "Division"};

    int choice;
    double num1, num2, result;
    do {
        printf("\nMenu:\n");
        printf("1. %s\n", operations[0]);
        printf("2. %s\n", operations[1]);
        printf("3. %s\n", operations[2]);
        printf("4. %s\n", operations[3]);
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);
        if (choice == 5) {
            break;
        }
        printf("Enter two numbers: ");
        scanf("%lf %lf", &num1, &num2);
        switch(choice) {
            case 1:
                result = num1 + num2;
                printf("Result of Addition: %.2lf\n", result);
                break;
            case 2:
                result = num1 - num2;
                printf("Result of Subtraction: %.2lf\n", result);
                break;
            case 3:
                result = num1 * num2;
                printf("Result of Multiplication: %.2lf\n", result);
                break;
            case 4:
                if (num2 != 0) {
```

```
                    result = num1 / num2;
                    printf("Result of Division: %.2lf\n", result);
                } else {
                    printf("Error: Division by zero is not allowed!\n");
                }
                break;
            default:
                printf("Invalid choice! Please choose a valid operation (1-5).\n");
                continue;
        }
        operationCount++;

    } while(choice != 5);
    printf("\nTotal operations performed: %d\n", operationCount);

    return 0;
}
```

O/p:

```
Menu:
1. Addition
2. Subtraction
3. Multiplication
```

```
4. Division
5. Exit
Enter your choice (1-5): 1
Enter two numbers: 12 31
Result of Addition: 43.00

Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 3
Enter two numbers: 31 89
Result of Multiplication: 2759.00

Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 5
```

```
Total operations performed: 2
```

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.

```
Sol: #include <stdio.h>
 #define MAX_SIZE 3
void matrixAddition(int A[MAX_SIZE][MAX_SIZE], int
B[MAX_SIZE][MAX_SIZE], int result[MAX_SIZE][MAX_SIZE], int
rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = A[i][j] + B[i][j]; // Add corresponding elements
        }
    }
}
void matrixMultiplication(int A[MAX_SIZE][MAX_SIZE], int
B[MAX_SIZE][MAX_SIZE], int result[MAX_SIZE][MAX_SIZE], int
rowsA, int colsA, int rowsB, int colsB) {
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            result[i][j] = 0;
            for (int k = 0; k < colsA; k++) {
                result[i][j] += A[i][k] * B[k][j];            }
        }
    }
}

int main() {
    static int additionResult[MAX_SIZE][MAX_SIZE];
    static int multiplicationResult[MAX_SIZE][MAX_SIZE];

    int A[MAX_SIZE][MAX_SIZE], B[MAX_SIZE][MAX_SIZE];
    int rowsA, colsA, rowsB, colsB;
    printf("Enter the number of rows and columns for Matrix A: ");
    scanf("%d %d", &rowsA, &colsA);
    printf("Enter elements for Matrix A (%d x %d):\n", rowsA, colsA);
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsA; j++) {
            scanf("%d", &A[i][j]);
        }
    }
    printf("Enter the number of rows and columns for Matrix B: ");
    scanf("%d %d", &rowsB, &colsB);
    printf("Enter elements for Matrix B (%d x %d):\n", rowsB, colsB);
    for (int i = 0; i < rowsB; i++) {
        for (int j = 0; j < colsB; j++) {
            scanf("%d", &B[i][j]);
        }
    }
    if (rowsA == rowsB && colsA == colsB) {
        matrixAddition(A, B, additionResult, rowsA, colsA);
        printf("\nMatrix Addition Result:\n");
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsA; j++) {
                printf("%d ", additionResult[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("\nMatrix Addition is not possible. Matrices must have the same dimensions.\n");
    }
    if (colsA == rowsB) {
        matrixMultiplication(A, B, multiplicationResult, rowsA, colsA, rowsB, colsB);
        printf("\nMatrix Multiplication Result:\n");
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                printf("%d ", multiplicationResult[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("\nMatrix Multiplication is not possible. The number of columns in Matrix A must equal the number of rows in Matrix B.\n");
    }

    return 0;
}
```

O/p:

```
Enter the number of rows and columns for Matrix A: 2 2
Enter elements for Matrix A (2 x 2):
1 2
3 4
Enter the number of rows and columns for Matrix B: 2 2
Enter elements for Matrix B (2 x 2):
5 6
7 8

Matrix Addition Result:
```

```
6 8

10 12


Matrix Multiplication Result:

19 22

43 50
```

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.

```
Sol: #include <stdio.h>
        int main() {
    volatile int temperature;
    static int maxTemperature = -1000;
    int lowerThreshold = 15;
    int upperThreshold = 30;

    while (1) {
                printf("Enter the current temperature: ");
        scanf("%d", &temperature);
```

```
                if (temperature > maxTemperature) {
            maxTemperature = temperature;
        }
        if (temperature < lowerThreshold) {
            printf("Warning: Temperature is too low!\n");
        } else if (temperature > upperThreshold) {
            printf("Warning: Temperature is too high!\n");
        } else {
            printf("Temperature is within the normal range.\n");
        }
        printf("Maximum temperature recorded: %d\n\n", maxTemperature);
        if (temperature < 0) {
            printf("Exiting temperature monitoring system...\n");
            break;
        }
    }

    return 0;
}
```

```
O/p: Enter the current temperature: 25
Temperature is within the normal range.
```

```
Maximum temperature recorded: 25

Enter the current temperature: 42
Warning: Temperature is too high!
Maximum temperature recorded: 42

Enter the current temperature: -2
Warning: Temperature is too low!
Maximum temperature recorded: 42

Exiting temperature monitoring system...
```

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.

```
Sol: #include <stdio.h>
#include <string.h>
static int failedAttempts=0;
const int MAX_ATTEMPTS =3;
int validatePassword(char *password){
    return strcmp(password,"likitha123")==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;
}
```

O/p: enter password:1234

```
invalid password
enter password:password
invalid password
enter password:likitha123
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.

```
Sol: #include <stdio.h>
 #define MAX_WITHDRAWAL_LIMIT 1000
int main() {
    static int balance = 5000;
    int choice, amount;
    char cont;
     printf("Welcome to the Bank Transaction Simulator!\n");
    do {
        printf("\n1. Deposit\n2. Withdraw\n3. Check Balance\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter amount to deposit: ");
                scanf("%d", &amount);
                if (amount > 0) {
                    balance += amount;
                    printf("Amount deposited successfully! Current balance:
%d\n", balance);
                } else {
                    printf("Error: Deposit amount must be positive.\n");
                }
                break;

            case 2:
                printf("Enter amount to withdraw: ");
                scanf("%d", &amount);
                if (amount > MAX_WITHDRAWAL_LIMIT) {
                    printf("Error: Cannot withdraw more than %d at once.\n",
MAX_WITHDRAWAL_LIMIT);
                } else if (amount > balance) {
                    printf("Error: Insufficient balance.\n");
                } else if (amount <= 0) {
                    printf("Error: Withdrawal amount must be positive.\n");
                } else {
                    balance -= amount;
                    printf("Amount withdrawn successfully! Current balance: %d\n", balance);
                }
                break;

            case 3:
                printf("Current balance: %d\n", balance);
                break;

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

        printf("Do you want to perform another transaction? (y/n): ");
        scanf(" %c", &cont);
    } while (cont == 'y' || cont == 'Y');
    printf("Thank you for using the Bank Transaction Simulator!\n");
    return 0;
}
```

O/p:

```
Welcome to the Bank Transaction Simulator!
```

```
1. Deposit
2. Withdraw
3. Check Balance
Enter your choice: 1
Enter amount to deposit: 139075
Amount deposited successfully! Current balance: 144075
Do you want to perform another transaction? (y/n): y

1. Deposit
2. Withdraw
3. Check Balance
Enter your choice: 2
Enter amount to withdraw: 10000
Error: Cannot withdraw more than 1000 at once.
Do you want to perform another transaction? (y/n): y

1. Deposit
2. Withdraw
3. Check Balance
Enter your choice: 2
Enter amount to withdraw: 1000
Amount withdrawn successfully! Current balance: 143075
Do you want to perform another transaction? (y/n): y
```

```
1. Deposit
2. Withdraw
3. Check Balance
Enter your choice: 3
Current balance: 143075
Do you want to perform another transaction? (y/n): n
Thank you for using the Bank Transaction Simulator!
```

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.

Sol:

```
#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;
}
```

```
O/p: 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.

```
Sol: #include <stdio.h>
#define WINNING_SCORE 100
int main() {
    static int currentScore = 0;
    int roundScore;
    char playAgain;
    printf("Welcome to the Game Score Tracker!\n");
    printf("Reach a score of %d to win.\n", WINNING_SCORE);
 while (1) {
        printf("\nEnter score for this round: ");
        scanf("%d", &roundScore);
        if (roundScore < 0) {
            printf("Invalid score. Score must be positive.\n");
            continue;
        }

        currentScore += roundScore;
        printf("Current score: %d\n", currentScore);
            if (currentScore >= WINNING_SCORE) {
            printf("Congratulations! You have won the game with a score of %d!\n", currentScore);
            break;
        }
          printf("Do you want to play another round? (y/n): ");
        scanf(" %c", &playAgain);

        if (playAgain != 'y' && playAgain != 'Y') {
            printf("Game over. Final score: %d.\n", currentScore);
            break;
        }
    }

    return 0;
```

```
}
O/p: Welcome to the Game Score Tracker!
Reach a score of 100 to win.

Enter score for this round: 60
Current score: 60
Do you want to play another round? (y/n): y

Enter score for this round: 50
Current score: 110
Congratulations! You have won the game with a score of 110!
```