Extra Questions

Q1. 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>

int const maximun_grades=5;

char get_grade(int total_marks);



char get_grade(int total_marks) {

    char grade;



    if (total_marks >= 90) {

        grade = 'A';

    } else if (total_marks >= 80) {

        grade = 'B';

    } else if (total_marks >= 70) {

        grade = 'C';

    } else if (total_marks >= 60) {

        grade = 'D';

    } else {

        grade = 'F';

    }
```

```
    return grade;
}

int main()
{
    int student_count;

    int static total_students=0;
    int volatile update=0;
    int num_students;
    printf("Enter the number of students: ");
    scanf("%d", &num_students);

    for(int i=0;i<num_students;i++){
        int total_marks;
    char grade;
    printf("Enter marks for student :\n");
    scanf("%d",&total_marks);
    grade = get_grade(total_marks);
    printf("Student %d's grade: %c\n", i + 1, grade);
    student_count++;

    }

    printf("Total students processed: %d\n", student_count);
```

```
    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 upperLimit; // 'const' used correctly for constant upper limit

    static int count = 0; // Static variable to count primes

    printf("Enter upper limit N:\n");

    scanf("%d", &upperLimit);

    if (upperLimit > 1) {

        for (int i = 2; i <= upperLimit; i++) {

            int isPrime = 1; // Assume the number is prime

            for (int j = 2; j <= i / 2; j++) {

                if (i % j == 0) { // If divisible, it's not a prime

                    isPrime = 0;

                    break;

                }

            }
```

```
            if (isPrime) {
                count++;
            }
        }
        printf("Number of prime numbers = %d\n", count);
    } else {
        printf("Invalid input. Please enter a number greater than 1.\n");
    }
    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>

int main()

{

   int static count=0;

   char *const operaation[]={"Addition","Subtraction","Multiplication","Division"};

   int choice;

   int a,b;

    do{

   printf("Enter 2 numbers:\n");
```

```
    scanf("%d %d",&a,&b);
    printf("Select Operation:\n
1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Exit");
    scanf("%d",&choice);
        if(choice==5)
        break;
    switch (choice){
        case 1:{
        printf("%d+%d=%d\n",a,b,a+b);
        break;
        }
        case 2:{
        printf("%d-%d=%d\n",a,b,a-b);
        break;
        }
        case 3:{
        printf("%d*%d=%d\n",a,b,a*b);
        break;
        }
        case 4:{
            if(b==0){
                printf("Division by 0 is not possible\n");
                break;
            }
            else{
```

```
            printf("%d/%d=%d\n",a,b,a/b);
            break;
        }
      }
      default:
      printf("Invalid!Enter valid Operation\n");
    }
    count++;
    }while(1);
    printf("Total No: of operations: %d\n",count);
}
```

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>

const int max_size = 5; // Const global variable for matrix size

void inputMatrix(int rows, int cols, int matrix[max_size][max_size]);

void printMatrix(int rows, int cols, int matrix[max_size][max_size]);

void addMatrices(int rows, int cols, int matrix1[max_size][max_size], int
matrix2[max_size][max_size], int result[max_size][max_size]);

void multiplyMatrices(int rows1, int cols1, int matrix1[max_size][max_size], int
rows2, int cols2, int matrix2[max_size][max_size], int result[max_size][max_size]);
```

```
int main() {
    int rows1, cols1, rows2, cols2;
    printf("Enter rows and columns for the first matrix (max %d): ", max_size);
    scanf("%d%d", &rows1, &cols1);

    printf("Enter rows and columns for the second matrix (max %d): ", max_size);
    scanf("%d%d", &rows2, &cols2);

    int matrix1[max_size][max_size], matrix2[max_size][max_size];
    static int additionResult[5][5];    // Static variable to hold addition results
    static int multiplicationResult[5][5]; // Static variable to hold multiplication results

    printf("Enter elements for the first matrix:\n");
    inputMatrix(rows1, cols1, matrix1);

    printf("Enter elements for the second matrix:\n");
    inputMatrix(rows2, cols2, matrix2);

    // Matrix addition
    if (rows1 == rows2 && cols1 == cols2) {
        addMatrices(rows1, cols1, matrix1, matrix2, additionResult);
        printf("Matrix Addition Result:\n");
        printMatrix(rows1, cols1, additionResult);
    } else {
        printf("Matrix addition not possible: dimensions do not match.\n");
```

```
    }

    // Matrix multiplication
    if (cols1 == rows2) {
        multiplyMatrices(rows1, cols1, matrix1, rows2, cols2, matrix2,
multiplicationResult);
        printf("Matrix Multiplication Result:\n");
        printMatrix(rows1, cols2, multiplicationResult);
    } else {
        printf("Matrix multiplication not possible: incompatible dimensions.\n");
    }
    return 0;
}

void inputMatrix(int rows, int cols, int matrix[max_size][max_size]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Enter element [%d][%d]: ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }
}

void printMatrix(int rows, int cols, int matrix[max_size][max_size]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
```

```
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void addMatrices(int rows, int cols, int matrix1[max_size][max_size], int
matrix2[max_size][max_size], int result[max_size][max_size]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
}

void multiplyMatrices(int rows1, int cols1, int matrix1[max_size][max_size], int
rows2, int cols2, int matrix2[max_size][max_size], int result[max_size][max_size]) {
    // Initialize result matrix
    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
            result[i][j] = 0;
        }
    }

    for (int i = 0; i < rows1; i++) {
        for (int j = 0; j < cols2; j++) {
```

```
                for (int k = 0; k < cols1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }
    }
```

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>

int max_threshold = 50;

int main() {
    volatile float temperature_reading = 0;
    static float maximum_temp = 0;
    while (1) {
        temperature_reading += 5;

        if (temperature_reading > max_threshold) {
            printf("Warning: Temperature exceeds maximum threshold\n");
            maximum_temp = temperature_reading;

        } else {
            printf("Temperature: %f\n", temperature_reading);
```

```
        }

        printf("Maximum Temperature Recorded: %f\n", maximum_temp);

        // Simulate a delay (roughly 1 second)

        for (int i = 0; i < 100000000; i++) {

        }

    }

  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>

int main()

{

    int const max_attempt=5;

    char const correct_password[]="567abc";

    char password[7];

    int static count=0;

    do{


        printf("Enter password:\n");
```

```
        scanf("%s", password);
        if(strcmp(correct_password,password)!=0){
            printf("Try again\n");
        }
        else{
            printf("password is correct\n");
            break;
        }
        count++;
    }while(count<max_attempt);
    if(count==max_attempt)
    printf("Attempts exceeds the maximum limit\n");
    printf("Number of attempts:%d\n",count);
    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>
int main()
{
    static float account_balance=0;
    float const withdraw_max=5000;
```

```
int choice;
float amount;

do{
  printf("\nEnter Choice:\n1.Deposit\n2.Withrawal\n3.Check Balance\n4.Exit\n");
  scanf("%d",&choice);
  switch (choice){
      case 1:
      printf("Enter amount to deposit: ");
      scanf("%f",&amount);
      account_balance+=amount;
      printf("\n%f deposited to the account",amount);
      break;
      case 2:
      printf("Enter amount to withdraw: ");
      scanf("%f",&amount);
      if(account_balance<=0)
      {
         printf("\nTransaction not possible:minimum balance");
      }
      else if(amount>withdraw_max){
         printf("\nTransaction not possible:Exceeds maximum withdraw limit");
         break;
      }
      else{
```

```
            account_balance-=amount;
        printf("\nAmount withdraw: %f",amount);
        }
        break;
        case 3:
        printf("\nBalace amount=%f",account_balance);
        break;

        default:
        printf("\n Invalid Entry");
     }
   }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>
int main() {
    static int total_ticks = 0;
    volatile int tick = 0;
```

```
    int hours, minutes, seconds;
    for (hours = 0; hours < 24; hours++) {
        for (minutes = 0; minutes < 60; minutes++) {
            for (seconds = 0; seconds < 60; seconds++) {
                // Increment total ticks for each second
                total_ticks++;
                printf("Time: %02d:%02d:%02d\n", hours, minutes, seconds);
                sleep(1);
                tick++;
                if (tick >= 60) {
                    tick = 0;
                }
            }
        }
    }
    printf("Total ticks: %d\n", total_ticks);
    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>
void playRound() {
   static int score = 0;
```

```
    score += 10;
    printf("Current Score: %d\n", score);
    if (score >= 50) {
        printf("Congratulations! You've won the game.\n");
    } else {
        printf("Keep playing!\n");
    }
}

int main() {
    const int winningScore = 50;
    int round = 1;
    printf("Welcome to the Game!\n");
    printf("Reach %d points to win.\n", winningScore);
    while (1) {
        printf("\n--- Level %d ---\n", round);
        playRound();
        char choice;
        printf("Do you want to play the next round? (y/n): ");
        scanf(" %c", &choice);

        if (choice == 'n' || choice == 'N') {
            printf("Exiting the game. Thanks for playing!\n");
            break;
        }
        round++;
    }

    return 0;
}
```