1. Basic Global and Local Variable Usage Problem Statement: Write a program that declares a global variable and a local variable with the same name. Modify and print both variables to demonstrate their scope and accessibility.

```
#include <stdio.h>

int variable = 10;

void demonstrateScope() {
    int variable = 20;
    printf("Local variable inside function: %d\n", variable);
    extern int variable;
    variable = 30;
    printf("Global variable inside function: %d\n", variable);
}

int main() {
    printf("Global variable before function call: %d\n", variable);
    demonstrateScope();
    printf("Global variable after function call: %d\n", variable);
    return 0;
}
```

2. Global Variable Across Functions

- **Problem Statement**: Declare a global variable and create multiple functions to modify its value. Each function should perform a different operation (e.g., addition, subtraction) on the global variable and print its updated value.

```
#include <stdio.h>

// Declare a global variable
int globalVariable = 0;

void add(int value) {
    globalVariable += value;
    printf("After addition, globalVariable = %d\n", globalVariable);
}

void subtract(int value) {
    globalVariable -= value;
```

```
    printf("After subtraction, globalVariable = %d\n", globalVariable);
}
void multiply(int value) {
    globalVariable *= value;
    printf("After multiplication, globalVariable = %d\n", globalVariable);
void divide(int value) {
    if (value != 0) {
        globalVariable /= value;
        printf("After division, globalVariable = %d\n", globalVariable);
    } else {
        printf("Division by zero is not allowed.\n");
    }
}
int main() {
    printf("Initial globalVariable = %d\n", globalVariable);
    add(10);
    subtract(5);
    multiply(3);
    divide(2);

    return 0;
}
```

3. Local Variable Initialization • Problem Statement: Write a program with a function that declares a local variable and initializes it to a specific value. Call the function multiple times and observe how the local variable behaves with each call.

```
#include <stdio.h>
void demonstrateLocalVariable() {
    int localVar = 10;  // Local variable initialized to 10
    printf("Value of localVar: %d\n", localVar);
    localVar++;  // Increment the local variable
}
```

```
int main() {
    demonstrateLocalVariable(); // Call 1
    demonstrateLocalVariable(); // Call 2
    demonstrateLocalVariable(); // Call 3
    return 0;
}
```

4. Combining Global and Local Variables • Problem Statement: Write a program that calculates the sum of a global variable and a local variable inside a function. Print the result and explain the variable scope in comments.

```
#include <stdio.h>

int globalVar = 50; // Global variable

void calculateSum() {
    int localVar = 30;  // Local variable
    int sum = globalVar + localVar;
    printf("Sum of globalVar and localVar: %d\n", sum);
}

int main() {
    calculateSum();
    return 0;
}
```

5. Global Variable for Shared State • Problem Statement: Write a program that uses a global variable as a counter. Multiple functions should increment the counter and print its value. Demonstrate how global variables retain their state across function calls.

```
#include <stdio.h>
int counter = 0; // Global counter
```

```
void incrementCounter() {
    counter++;
    printf("Counter value: %d\n", counter);
}
int main() {
    incrementCounter(); // Call 1
    incrementCounter(); // Call 2
    incrementCounter(); // Call 3
    return 0;
}
```

6. Shadowing Global Variables • Problem Statement: Write a program where a local variable in a function shadows a global variable with the same name. Use the global scope operator to access the global variable and print both values.

```
#include <stdio.h>

int value = 100; // Global variable

void demonstrateShadowing() {
    int value = 50;  // Local variable shadows the global variable
    printf("Local value: %d\n", value);
    printf("Global value using scope operator: %d\n", ::value);
}

int main() {
    demonstrateShadowing();
    return 0;
}
```

7. Read-Only Global Variable • Problem Statement: Declare a global constant variable and write a program that uses it across multiple functions without modifying its value. Demonstrate the immutability of the global constant.

```
#include <stdio.h>
```

```
const int configValue = 100; // Global constant

void displayConfig() {
    printf("Configuration Value: %d\n", configValue);
}

int main() {
    displayConfig();
    // configValue = 200; // Uncommenting this will cause a compilation error
    return 0;
}
```

8. Global Variable for Configuration • Problem Statement: Use a global variable to store configuration settings (e.g., int configValue = 100). Write multiple functions that use this global configuration variable to perform operations.

```
#include <stdio.h>

int configValue = 100; // Global configuration variable

void performOperation1() {
    printf("Operation 1 with configValue: %d\n", configValue * 2);
}

void performOperation2() {
    printf("Operation 2 with configValue: %d\n", configValue + 50);
}

int main() {
    performOperation1();
    performOperation2();
    return 0;
}
```

9. Local Variables with Limited Scope • Problem Statement: Write a program where local variables are declared inside a block (e.g., if or for block). Demonstrate that they are inaccessible outside the block.

```
#include <stdio.h>


int main() {
   if (1) {
      int blockVar = 10; // Local variable inside the block
      printf("blockVar inside block: %d\n", blockVar);
   }
   // printf("blockVar outside block: %d\n", blockVar); // Uncommenting this will cause an error
   return 0;
}
```

10. Combining Local and Global Variables in Loops • Problem Statement: Write a program that uses a global variable to track the total sum and a local variable to store the sum of elements in an array. Use a loop to calculate the local sum, then add it to the global total.

```
#include <stdio.h>


int totalSum = 0; // Global variable


void calculateLocalSum(int arr[], int size) {
   int localSum = 0; // Local variable
   for (int i = 0; i < size; i++) {
      localSum += arr[i];
   }
   printf("Local sum: %d\n", localSum);
   totalSum += localSum; // Add local sum to global sum
}


int main() {
   int arr[] = {1, 2, 3, 4, 5};
   int size = sizeof(arr) / sizeof(arr[0]);
```

```
    calculateLocalSum(arr, size);
    printf("Global totalSum: %d\n", totalSum);

    return 0;
}
```

Problem statements on Static Storage classes

1. Static Variable in a Loop • Problem Statement: Write a program that uses a static variable inside a loop to keep track of the cumulative sum of numbers from 1 to 10. The loop should run multiple times, and the variable should retain its value between iterations.

```
#include <stdio.h>

void cumulativeSum() {
    static int sum = 0; // Static variable to retain its value between function calls
    for (int i = 1; i <= 10; i++) {
        sum += i;
    }
    printf("Cumulative sum: %d\n", sum);
}

int main() {
    cumulativeSum(); // First call
    cumulativeSum(); // Second call
    cumulativeSum(); // Third call
    return 0;
}
```

2. Static Variable to Count Iterations • Problem Statement: Use a static variable inside a loop to count the total number of iterations executed across multiple runs of the loop. Print the count after each run.

```
#include <stdio.h>
```

```
void countIterations() {
    static int totalIterations = 0; // Static variable to track iterations
    for (int i = 0; i < 5; i++) {
        totalIterations++;
    }
    printf("Total iterations so far: %d\n", totalIterations);
}

int main() {
    countIterations(); // First run
    countIterations(); // Second run
    countIterations(); // Third run
    return 0;
}
```

3. Static Variable in Nested Loops • Problem Statement: Use a static variable in a nested loop structure to count the total number of times the inner loop has executed across multiple runs of the program.

```
#include <stdio.h>

void nestedLoopCounter() {
    static int innerLoopCount = 0; // Static variable to count inner loop executions
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            innerLoopCount++;
        }
    }
    printf("Inner loop executed %d times so far.\n", innerLoopCount);
}

int main() {
    nestedLoopCounter(); // First run
    nestedLoopCounter(); // Second run
```

```
    nestedLoopCounter(); // Third run
    return 0;
}
```

4. Static Variable to Track Loop Exit Condition • Problem Statement: Write a program where a loop executes until a specific condition is met. Use a static variable to track and display the number of times the loop exited due to the condition being true.

```
#include <stdio.h>

void trackExitCondition() {
    static int exitCount = 0; // Static variable to track loop exits
    int i = 0;
    while (1) {
        if (i >= 5) {
            exitCount++;
            break; // Exit condition met
        }
        i++;
    }
    printf("Loop exited %d times due to condition.\n", exitCount);
}

int main() {
    trackExitCondition(); // First run
    trackExitCondition(); // Second run
    trackExitCondition(); // Third run
    return 0;
}
```

5. Static Variable to Track Loop Re-entry • Problem Statement: Write a program where a static variable keeps track of how many times the loop is re-entered after being interrupted (e.g., using a break statement).

```
#include <stdio.h>
```

```
void trackReEntry() {
    static int reEntryCount = 0; // Static variable to track loop re-entries
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            reEntryCount++;
            break; // Interrupt the loop
        }
    }
    printf("Loop re-entered %d times.\n", reEntryCount);
}

int main() {
    trackReEntry(); // First run
    trackReEntry(); // Second run
    trackReEntry(); // Third run
    return 0;
}
```

6. Static Variable for Step Count in Loops • Problem Statement: Create a program with a loop that increments by a variable step size. Use a static variable to count and retain the total number of steps taken across multiple runs of the loop.

```
#include <stdio.h>
void countSteps() {
    static int totalSteps = 0; // Static variable to count steps
    int stepSize = 2;
    for (int i = 0; i < 10; i += stepSize) {
        totalSteps++;
    }
    printf("Total steps so far: %d\n", totalSteps);
}

int main() {
    countSteps(); // First run
```

```
    countSteps(); // Second run
    countSteps(); // Third run
    return 0;
}
```

Problem statement on const Type specifier

1. Using const for Read-Only Array

- **Problem Statement**: Declare an array of integers as const and use a loop to print each element of the array. Attempt to modify an element inside the loop and explain the result.

```
#include <stdio.h>

int main() {
    const int arr[] = {1, 2, 3, 4, 5}; // Declare a const array
    int size = sizeof(arr) / sizeof(arr[0]);

    for (int i = 0; i < size; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
        // arr[i] = 10; // Uncommenting this will cause a compilation error
    }

    return 0;
}
```

2. const Variable as a Loop Limit

- **Problem Statement**: Declare a const integer variable as the upper limit of a loop. Write a loop that runs from 0 to the value of the const variable and prints the iteration count.

```
#include <stdio.h>

int main() {
    const int limit = 5; // const variable as loop limit

    for (int i = 0; i < limit; i++) {
        printf("Iteration: %d\n", i);
```

```
    }


    return 0;
}
```

3. Nested Loops with const Limits

- **Problem Statement**: Use two const variables to define the limits of nested loops. Demonstrate how the values of the constants affect the total number of iterations.

```
#include <stdio.h>


int main() {
    const int rows = 3; // Number of rows
    const int cols = 4; // Number of columns


    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("(%d, %d) ", i, j);
        }
        printf("\n");
    }


    return 0;
}
```

4. const for Read-Only Pointer in Loops

- **Problem Statement**: Declare a const pointer to an integer and use it in a loop to traverse an array. Print each value the pointer points to.

```
#include <stdio.h>


int main() {
    int arr[] = {10, 20, 30, 40, 50};
    const int *ptr = arr; // const pointer to read-only integers


    for (int i = 0; i < 5; i++) {
```

```
    printf("Value at ptr[%d]: %d\n", i, *ptr);
    ptr++;
  }

  return 0;
}
```

5. const for Loop-Invariant Variable

- **Problem Statement**: Declare a const variable that holds a mathematical constant (e.g., PI = 3.14). Use this constant in a loop to calculate and print the areas of circles for a range of radii.

```
#include <stdio.h>

int main() {
  const double PI = 3.14; // const variable for PI
  for (int radius = 1; radius <= 5; radius++) {
    double area = PI * radius * radius;
    printf("Radius: %d, Area: %.2f\n", radius, area);
  }
  return 0;
}
```

6. const Variable in Conditional Loops

- **Problem Statement**: Use a const variable as a termination condition for a while loop. The loop should terminate when the iteration count reaches the value of the const variable.

```
#include <stdio.h>

int main() {
  const int maxIterations = 5; // const variable as loop condition
  int count = 0;

  while (count < maxIterations) {
    printf("Iteration: %d\n", count);
    count++;
```

```
    }


    return 0;
}
```

7. const and Immutable Loop Step Size

- **Problem Statement**: Declare a const variable as the step size of a for loop. Use this step size to iterate through a range of numbers and print only every nth number.

```
#include <stdio.h>

int main() {
    const int stepSize = 2; // const variable for step size
    for (int i = 0; i <= 10; i += stepSize) {
        printf("i = %d\n", i);
    }
    return 0;
}
```

8. const Variable for Nested Loop Patterns

- **Problem Statement**: Use two const variables to define the number of rows and columns for printing a rectangular pattern using nested loops. The dimensions of the rectangle should be based on the const variables.

```
#include <stdio.h>


int main() {
    const int rows = 3; // const variable for rows
    const int cols = 5; // const variable for columns


    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("* ");
        }
        printf("\n");
    }
```

```
    return 0;
}
```