```c
#include <stdio.h>

int main()
{
   int const data1=10;

   printf("01 data = %d\n",data1);

   int *ptr = &data1;

   *ptr = 500;

    printf("02 data = %d\n",data1);

   return 0;
}
```

//// case:  modifiable pointer and constant data----------

```c
#include <stdio.h>

int main()
{
   int a = 10;
   int b = 20;

   int const *Ptr = &a;

   printf("address of a = %p\n",&a);
```

```
    printf("1. adress of ptr =%p\n",Ptr);



    Ptr = &b;
    printf("address of b = %p\n",&b);
    printf("2. address pf ptr= %p",Ptr);

    return 0;
}



///----------Case : modifiable data constant data



#include <stdio.h>

int main()
{
    int a = 10;
    int b = 20;

    int *const Ptr = &a;



    printf("1. a =%d\n",a);
    *Ptr = 40;
    printf("2. a= %d",a);

    return 0;
}
```

/////// case: constant data constant pointer----------

```
#include <stdio.h>

int main()
{
   int a = 10;
   int b = 20;

   int const *const Ptr = &a;
   Ptr = &b;
   printf("1. a =%d\n",a);
   *Ptr = 40;
   printf("2. a= %d",a);

   return 0;
}
```

**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 a = 10;
void modifyVariable()
{
   int a =20;
   printf("local Variable = %d\n",a);
    a = 30;
```

```
    printf(" modified local Variable = %d\n",a);
    {
        extern int a;
        printf("global Variable = %d\n",a);
        a = 40;
         printf("Modified global Variable = %d\n",a);
    }
}
int main()
{
    printf(" initial global variable = %d\n",a);

    modifyVariable();

    printf("Modified global Variable = %d\n",a);

    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>
int a = 10;
void  add()
{
    a+= 5;
    printf("After addition a = %d\n",a);
}
```

```
void sub()
{
   a-=5;
   printf("After subtraction a = %d\n",a);
}
void mul()
{
   a*=5;
   printf("After multiplcaton a = %d\n",a);
}
void divi()
{
a=a/2;
printf("After division a = %d\n",a);
}
int main()
{
   printf("a=%d",a);
   add();
   sub();
   mul();
   divi();
   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  fun1()
{
   int a = 10;
   printf("function call of local variable initialized to %d\n",a);
}
int main()
{
   fun1();
   fun1();
   fun1();
   fun1();
   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 a = 10;
void  sum()
{
   int b = 20;
   int result = a+b;
   printf("sum of global variable %d and local vairalble %d is %d",a,b,result);
}
int main()
{
   sum();
   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;
void incrementCounter()
   {
   counter++;
   printf("Counter value after increment: %d\n", counter);
   }
void incrementCounterBy(int value)
{
   counter = counter+value;
   printf("Counter value after increment by %d is: %d\n",value, counter);
}
int main()
{
   printf("initial counter value =%d\n",counter);
   incrementCounter();
   incrementCounterBy(5);
   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 n = 5;
int main()
{
   int n=10;
   {
      extern int n;
      printf("global Value of n is %d\n",n);
   }
    printf("Local Value of n is %d\n",n);
    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>

 int const GLOBAL_CONSTANT = 100;

void print_constant() {
   printf("The value of GLOBAL_CONSTANT in print_constant function: %d\n", GLOBAL_CONSTANT);
}

void modify_constant() {
}
```

```
int main() {
    print_constant();

    modify_constant();
    printf("The value of GLOBAL_CONSTANT in main function: %d\n", GLOBAL_CONSTANT);

    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 cV = 100;

void setConfigValue(int value) {
    cV = value;
}

int getConfigValue() {
    return cV;
}

void printConfigValue() {
```

```
    printf("Current config value: %d\n", cV);
}

void doubleConfigValue() {
    cV *= 2;
}

int main() {
    printConfigValue();

    setConfigValue(200);
    printConfigValue();

    doubleConfigValue();
    printConfigValue();

    int currentValue = getConfigValue();
    printf("Config value retrieved: %d\n", currentValue);

    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 x = 15;
    printf("Inside if block: x = %d\n", x);
  }

  // Uncommenting the next line will cause a compile-time error because 'x' is out of scope
  // printf("Outside if block: x = %d\n", x);  // Error! 'x' is not accessible outside the block

  for (int i = 0; i < 1; i++) {
    int y = 20;
    printf("Inside for loop: y = %d\n", y);
  }

  // Uncommenting the next line will also cause a compile-time error because 'y' is out of scope
  // printf("Outside for loop: y = %d\n", y);  // Error! 'y' is not accessible outside the block

  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 Tsum = 0;

void calculate_sum(int arr[], int size)
{
  int sum = 0;

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

```
        sum += arr[i];
    }

      Tsum += sum;

    printf("Local sum of this array: %d\n", sum);
}

int main() {
    int arr1[] = {2, 4, 6, 8, 10};
    int arr2[] = {10, 20, 30};

    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);

    calculate_sum(arr1, size1);
    calculate_sum(arr2, size2);

    printf("Global total sum: %d\n", Tsum);

    return 0;
}
```

**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 cummilativeSum()
{
  static int sum=0;
```

```
  for(int i=0;i<10;i++)
  {
     sum+=i;
  }
  printf("cummilative sum = %d \n",sum);
}

int main()
{
   printf("after 1 st iteration : \n");
   cummilativeSum();
   printf("after 2nd iteration : \n");
   cummilativeSum();

   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 count()
{
   static int count = 0;
   for(int i =0;i<=5 ; i++)
   {
   count++;
   }
   printf("total iterations = %d\n",count);
}
```

```
int main()
{

   count();
   count();
   count();
   count();

   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 netsed()
{
   static int count = 0;
   for(int i =0;i<=5 ; i++)
   {
    for(int j=0;j<=4;j++)
    {
       count++;

    }
   }
   printf("inner loop execution = %d\n",count);
}
int main()
{
```

```
    netsed();
    netsed();
    netsed();
    netsed();
    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 condition()
{
    static int count = 0;
    int c = 0;
    int limit = 5;
    while (1) {
        printf("loop executed %d\n", c);
        c++;
        if (c == limit)
        {
            count++;
            printf(" Loop exited %d times.\n", count);
            break;
        }
    }
}

int main() {
```

```
    condition();
    condition();
    condition();

    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 loop() {
    static int Count = 0;
    int i;
    for (i = 0; i < 10; i++)
    {
        if (i == 5)
        {
            Count++;
            printf("Loop interrupted at i = %d, re-entry count: %d\n", i, Count);
            break;
        }
        printf("i = %d\n", i);
    }
}

int main()
{
```

```
    for (int j = 0; j < 3; j++)
    {
        loop();
    }
    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 loop(int stepSize)
{
    static int totalSteps = 0;
    int steps = 0;

    for (int i = 0; i < 100; i += stepSize)
    {
        steps++;
    }

    totalSteps += steps;
    printf("Steps = %d\n", steps);
    printf("Total steps= %d\n", totalSteps);
}

int main()
{
    loop(5);
```

```
    loop(10);
    loop(20);

    return 0;
}
```

**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};
    int size = sizeof(arr)/sizeof(arr[0]);
    for(int i=0;i<=4;i++)
    {
        printf("%d elemnt is %d:\n",i,arr[i]);
    }
    for(int i=0;i<=4;i++)
    {
        arr[i] = arr[i]+1;
    }
     return 0;
}
```

```
///----------------------Result------------

error: assignment of read-only location ‘arr[i]’
```

**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=10;
   for(int i=0;i<=limit;i++)
   {
      printf("iteration count : %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 limit=4;
   const int innerlmt=3;
   int total=0;
   for(int i=0;i<=limit;i++)
   {
      for(int j=0;j<=innerlmt;j++)
      {
         total++;

      }
   }
```

```
    printf("total itration = %d",total);
    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[] = { 1, 2, 3, 4, 5};
   int size = sizeof(arr)/sizeof(arr[0]);
   int const *Ptr = arr;
   for(int i=0;i<=size;i++)
   {
      printf("value at index %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 float PI = 3.14;
   float r,a;
   for(r=1;r<=10;r++)
```

```
    {
        a=PI*r*r;
        printf("radius %.2f, Area = %.2f \n",r,a);
    }
     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 count = 5;
    int i=1;
    while(i<=count)
    {
        printf("iteration count = %d\n",i);
        i++;
    }
     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 = 3;
```

```
    for (int i = 0; i <= 30; i += stepSize)
    {
        printf("%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 = 5;
    const int columns = 10;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```