```
#include <stdio.h>

int main()
{
    int const data1 = 10;
    data1 = 50;


    return 0;
}
```

POINTERS

```
#include <stdio.h>

int main()
{
    int const data1 = 10;

    printf("001Data1=%d \n",data1);

    int *Ptr = &data1;

    *Ptr=500;

    printf("002Data1=%d \n",data1);


    return 0;
}
```

AFTER TYPE CASTING

```
#include <stdio.h>

int main()
{
    int const data1 = 10;

    printf("001Data1=%d \n",data1);

    int *Ptr = (int *) &data1;

    *Ptr=500;

    printf("002Data1=%d \n",data1);


    return 0;
}


#include <stdio.h>
    int const data2= 10;

int main()
{
    printf("001Data1=%d \n",data2);

    int *Ptr = (int *) &data2;

    *Ptr=500;

    printf("002Data1=%d \n",data2);


    return 0;
}
```

case 2 modified ptr value const

Pointer value reinitialize,WAP TO USE POINTERS WHERE DATA CANNOT BE MODIFIED

```
#include <stdio.h>
    int const data2= 10;

int main()
{

   int a = 10;
   int b = 20;
   int const *Ptr= &a;
   printf("Address of a is= %p \n",&a);
   printf("001 address of Ptr = %p \n",Ptr);
    Ptr= &b;
     printf("Address of b is= %p \n",&b);

     printf("001 address of Ptr = %p",Ptr);


    return 0;
}
```

case 3 const ptr modified value
----

not allowing pointer to re initialize

```
#include <stdio.h>
    int const data2= 10;

int main()
{

   int a = 10;
   int b = 20;
   int *const Ptr= &a;
   ---------------------------------
   printf("Address of a is= %p \n",&a);
   printf("001 address of Ptr = %p \n",Ptr);
   Ptr= &b;
   printf("Address of b is= %p \n",&b);
   printf("001 address of Ptr = %p",Ptr);
```

```
    return 0;
}
```

value can be modified

```
#include <stdio.h>
    int const data2= 10;

int main()
{

   int a = 10;
   int b = 20;
   int *const Ptr= &a;
   printf("001 a= %d \n",a);
     *Ptr=40;
   printf("002 a= %d \n",a);


    return 0;
}
```

case 4:const ptr and constant value

```
#include <stdio.h>
    int const data2= 10;

int main()
{

   int a = 10;
   int b = 20;
   int const *const Ptr= &a;
   printf("001 a= %d \n",a);
   *Ptr=40;
   printf("002 a= %d \n",a);


    return 0;
}
```

LOCAL AND GLOBAL VARIBLES PROBLEMS

1.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 = 12;
int main()
{
    int a = 15;
    printf("In a  local a = : %d\n",a);
    printf("In a global a = %d\n",a);
    return 0;
}
```

2.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 num = 10;

void add() {
    num += 5;
    printf("After addition, num = %d\n", num);
}

void subtract() {
    num -= 3;
    printf("After subtraction, num = %d\n", num);
}


int main() {
    printf("Initial value of num = %d\n", num);

    add();
    subtract();

    return 0;
}
```

3.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 F1_a() {
    int a = 5;
    printf("Local variable initialized to %d in this function call.\n", a);
}

int main() {
    F1_a();
    F1_a();
    F1_a();

    return 0;
}
```

4.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 add() {
    int b = 5;

    int result = a + b;
    printf("The sum of the global and local variables is: %d\n", result);
}

int main() {
    add();

    printf("Global variable value outside the function: %d\n", a);

    return 0;
}
```

5.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 f1()
 {
    counter++;
    printf("Value of counter : %d\n",counter);
 }
  void f2()
 {
    counter++;
     printf("Value of counter : %d\n",counter);
 }
 void f3()
 {
    counter++;
     printf("Value of counter : %d\n",counter);
 }
 void f4()
 {
    counter++;
     printf("Value of counter : %d\n",counter);
 }

 void main(){
    f1();
    f2();
    f3();
    f4();

 }
```

7.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 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.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.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 = 10;
        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.: 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[] = {1, 2, 3, 4, 5};
    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;
}
```

6.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 f1(void)
{
    int n = 10;
    return n;
}
void main(){
    printf("The global value of n is %d and local values of n is %d \n",n,f1());
}
```

STORAGE CLASS PROBLEMS

1.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 _sum() {
    static int sum = 0;

    for (int i = 1; i <= 10; i++) {
        sum += i;

        printf("Cumulative sum after adding %d: %d\n", i, sum);
    }
}

int main() {
    printf("Running first iteration:\n");
    _sum();

    printf("\nRunning second iteration:\n");
    _sum();

    return 0;
}
```

2.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 count1() {
    static int tI = 0;//tI is total iteration

    int num = 5;
    for (int i = 0; i < num; ++i) {
        tI++;
    }
    printf("Total iterations after this run: %d\n", tI);
}
int main() {
    count1();
    count1();
    count1();
    return 0;
}
```

3.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 cIE() {
    static int tE = 0;  //tE is total Executions

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 5; j++) {
            tE++;
        }
    }

    printf("Total inner loop executions: %d\n", tE);
}

int main() {
    cIE();  //cIE is count inner loop iteration
    cIE();
    cIE();

    return 0;
}
```

4.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 exit_count = 0;
    int c = 0;
    int limit = 5;
    while (1) {
        printf("Current counter: %d\n", c);
        c++;
        if (c == limit) {
            exit_count++;
            printf("Condition met! Exiting loop. Loop exited %d times.\n", exit_count);
            break;
        }
    }
}

int main() {
    condition();
    condition();
    condition();

    return 0;
}
```

5.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  interruption() {
    static int interrupt_count = 0;
    for (int i = 0; i < 5; i++) {
        printf("Loop iteration: %d\n", i);
        // Break the loop at iteration 2 to simulate an interruption
        if (i == 2) {
            printf("Loop interrupted!\n");
            interrupt_count++;
            break;
        }
    }
    printf("Loop interrupted %d times.\n", interrupt_count);
}
int main() {
    interruption();
    interruption();
    interruption();

    return 0;
}
```

6.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 increment(int stepsize, int limit) {
    static int totalsteps = 0;

    int count = 0;
    for (int i = 0; i < limit; i += stepsize) {
        count++;
        printf("Current value: %d, Count for this run: %d\n", i, count);
    }

    totalsteps += count;
    printf("Total steps so far (across all runs): %d\n", totalsteps);
}

int main() {
    increment(2, 10);
    increment(3, 15);
    increment(1, 5);

    return 0;
}
```

PROGRAMS USING CONST

1.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};

    for (int i = 0; i < 5; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);

        // Attempt to modify an element (this will cause an error)
        // arr[i] = arr[i] + 1;  // Uncommenting this line will cause a compile-time error
    }

    return 0;
}
```

2.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 upperLimit = 10;

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

    return 0;
}
```

3.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 outerLimit = 5;
    const int innerLimit = 4;

    int totalIterations = 0;

    for (int i = 0; i < outerLimit; ++i) {
        for (int j = 0; j < innerLimit; ++j) {
            totalIterations++;
            printf("Outer loop iteration %d, Inner loop iteration %d\n", i + 1, j + 1);
        }
    }

    printf("Total iterations: %d\n", totalIterations);

    return 0;
}
```

4.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};

    const int *ptr = arr;

    for (int i = 0; i < 5; i++) {
        printf("Element %d: %d\n", i + 1, *ptr);

        ptr++;
    }

    return 0;
}
```

5.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>

#define PI 3.14159

int main() {
    int i;
    double radius, area;

    for (radius = 1; radius <= 10; radius++) {
        area = PI * radius * radius;
        printf("Radius: %.2f, Area: %.2f\n", radius, area);
    }

    return 0;
}
```

6.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 MI = 10;
    int count = 0;

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

    printf("Loop terminated after %d iterations.\n", MI);

    return 0;
}
```

7.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 step_size = 3;

    for (int i = 0; i <= 30; i += step_size) {
        printf("%d ", i);
    }

    return 0;
}
```

8.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 cols = 10;

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

    return 0;
}
```