Variable scopes

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 x=20;
void main()
{
   printf("Initial global variable value: %d\n",x);
   x=30;
   printf("Global variable after modification: %d\n",x);
   int x=10;
   printf("Local variable value: %d\n",x);
   x+=50;
   printf("Local variable after modification: %d\n",x);
}
```

O/P:

```
Initial global variable value: 20
Global variable after modification: 30
Local variable value: 10
Local variable after modification: 60
```

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 i=50;
void add(int n)
{
   i+=n;
   printf("After addition, i=%d\n",i);
}

void sub(int n)
{
   i-=n;
   printf("After subtraction, i=%d\n",i);
}

void mul(int n)
{
   i*=n;
   printf("After multiplication, i=%d\n",i);
}

void div(int n)
{
   if(n!=0)
```

```
    {
        i/=n;
        printf("After division, i=%d\n",i);
    }
    else
        printf("Division by zero is not allowed\n");
}

void main()
{
    printf("Global value=%d\n",i);
    add(10);
    sub(25);
    mul(2);
    div(5);
    div(0);
    printf("Updated global value:%d\n",i);
}
```

O/P:

```
Global value=50
After addition, i=60
After subtraction, i=35
After multiplication, i=70
After division, i=14
Division by zero is not allowed
Updated global value:14
```

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 fun()
{
   int i=20;
   printf("Before:%d\n",i);
   i+=30;
   printf("After:%d\n",i);
}

void main()
{
   fun();
   printf("----------\n");
   fun();
   printf("----------\n");
   fun();
   printf("----------\n");
   fun();
}
```

O/P:

```
Before:20
```

```
After:50
---------
Before:20
After:50
---------
Before:20
After:50
---------
Before:20
After:50
```

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 g=10;
void sum()
{
   int l=20,s;
   printf("Global variable:%d\n",g);
   printf("Local variable:%d\n",l);
   s=g+l;
   printf("Sum of global and local variables:%d\n",s);
}

void main()
```

```
{
   sum();
}
```

O/P:

```
Global variable:10
Local variable:20
Sum of global and local variables:30
```

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 c=0;
void inc()
{
   c++;
   printf("Counter after increment:%d\n",c);
}

void inc1(int n)
{
   c+=n;
   printf("Counter after incrementing by %d: %d\n",n,c);
}
```

```
void main()
{
   printf("Initial counter value:%d\n",c);
   inc();
   inc1(20);
   inc1(5);
   inc();
   printf("Updated counter value:%d\n",c);
}
```

O/P:

```
Initial counter value:0
Counter after increment:1
Counter after incrementing by 20: 21
Counter after incrementing by 5: 26
Counter after increment:27
Updated counter value:27
```

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 g=10;
void main()
{
```

```
    printf("Initial global variable:%d\n",g);
    int g=20;
    printf("Local variable:%d\n",g);
    printf("Global value after accessing using same name:%d\n",g);
}
```

O/P:

```
Initial global variable:10
Local variable:20
Global value after accessing using same name:20
```

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 g=10;
void fun()
{
   printf("Global constant in function:%d\n",g);
}

void main()
{
   printf("Global constant in main:%d\n",g);
   fun();
}
```

O/P:

Global constant in main:10

Global constant in function:10

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;
void addConfigValue(int i)
{
   configValue+=i;
   printf("Config value after adding by %d:%d\n",i,configValue);
}

void subConfigValue(int i)
{
   configValue-=i;
   printf("Config value after subtracting by %d:%d\n",i,configValue);
}

void main()
{
   printf("Config value:%d\n",configValue);
   addConfigValue(10);
```

```
    subConfigValue(5);
    printf("Config value:%d\n",configValue);
}
```

O/P:

```
Config value:100
Config value after adding by 10:110
Config value after subtracting by 5:105
Config value:105
```

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>
void main()
{
    int a=10;
    printf("Value of a: %d\n",a);
    if(a>8)
    {
        int b=20;
        printf("Inside if block: a = %d, b = %d\n",a,b);
    }
    //printf("Outside if block: b = %d\n", b); // compilation error
    for (int i=0;i<3;i++)
    {
```

```
        int c=i+20;
        printf("Inside for loop: i = %d, c = %d\n",i,c);
    }
    //printf("Outside for loop: i = %d, c = %d\n",i,c); // compilation error
}
```

O/P:

```
Value of a: 10
Inside if block: a = 10, b = 20
Inside for loop: i = 0, c = 20
Inside for loop: i = 1, c = 21
Inside for loop: i = 2, c = 22
```

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 s=0;
void sum(int a[], int size)
{
   int s1=0,i;
   for(i=0;i<size;i++)
      s1+=a[i];
   printf("Local sum of array elements: %d\n",s1);
   s+=s1;
   printf("Updated global sum: %d\n",s);
```

```
}

void main()
{
   int a1[]={1, 2, 3, 4, 5}, a2[]={6, 7, 8, 9, 10}, ele1, ele2;
   ele1=sizeof(a1)/sizeof(a1[0]);
   ele2=sizeof(a2)/sizeof(a2[0]);
   printf("Sum of first array:\n");
   sum(a1,ele1);
   printf("--------------------------------\n");
   printf("Sum of second array:\n");
   sum(a2,ele2);
   printf("--------------------------------\n");
   printf("Total sum of global: %d\n",s);
}
```

O/P:

```
Sum of first array:
Local sum of array elements: 15
Updated global sum: 15
--------------------------------
Sum of second array:
Local sum of array elements: 40
Updated global sum: 55
--------------------------------
Total sum of global: 55
```

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 main()
{
   int i;
   for(i=1;i<=2;i++)
   {
      printf("Iteration %d:\n",i);
      static int cs=0;
      int s=0,i;
      for(i=1;i<=10;i++)
         s+=i;
      cs+=s;
      printf("Sum of numbers from 1 to 10: %d\n",s);
      printf("Cumulative sum: %d\n",cs);
   }
}
```

O/P:

```
Iteration 1:
Sum of numbers from 1 to 10: 55
```

Cumulative sum: 55

Iteration 2:

Sum of numbers from 1 to 10: 55

Cumulative sum: 110

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 main()
{
   int r,ci,i;
   static int t=0;
   do
   {
      printf("Enter the number of runs:\n");
      scanf("%d",&r);
      for(i=0;i<r;i++)
      {
         ci++;
         t++;
      }
      printf("Current run iterations: %d\n",ci);
      printf("Total iterations of all runs: %d\n",t);
   } while(r!=0);
}
```

O/P:

```
Enter the number of runs:
5
Current run iterations: 5
Total iterations of all runs: 5
Enter the number of runs:
6
Current run iterations: 11
Total iterations of all runs: 11
Enter the number of runs:
3
Current run iterations: 14
Total iterations of all runs: 14
Enter the number of runs:
0
Current run iterations: 14
Total iterations of all runs: 14
```

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 fun(int o,int i)
{
   int ce=0,j,k;
   static int te=0;
```

```
    for(j=0;j<o;j++)
    {
       for(k=0;k<i;k++)
       {
          ce++;
          te++;
       }
    }
    printf("Inner loop executions in current run: %d\n",ce);
    printf("Total inner loop executions across all runs: %d\n",te);
}

void main()
{
    fun(2,5);
    fun(3,4);
}
```

O/P:

```
Inner loop executions in current run: 10
Total inner loop executions across all runs: 10
Inner loop executions in current run: 12
Total inner loop executions across all runs: 22
```

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 run(int r)
{
   static int ec=0;
   int i=0;
   while(1)
   {
      printf("Current value: %d\n",i);
      if(i==r) {
         ec++;
         printf("Condition met\n");
         break;
      }
      i++;
   }
     printf("The loop has exited due to the condition being true %d times\n",ec);
}

void main()
{
   printf("First:\n");
   run(5);
   printf("Second:\n");
   run(2);
   printf("Third:\n");
   run(6);
```

```
}
```

O/P:

```
First:
Current value: 0
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Condition met
The loop has exited due to the condition being true 1 times
Second:
Current value: 0
Current value: 1
Current value: 2
Condition met
The loop has exited due to the condition being true 2 times
Third:
Current value: 0
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Current value: 6
Condition met
```

The loop has exited due to the condition being true 3 times

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 interrupt(int m,int i)
{
   static int c=0;
   int j=0;
   while(j<m)
   {
      printf("Current value: %d\n",j);
      if(j==i)
      {
         printf("Loop interrupted at %d\n",j);
         c++;
         break;
      }
      j++;
   }
   printf("The loop has been re-entered %d times after interruptions\n",c);
}

void main()
{
```

```
    printf("First:\n");
    interrupt(8,4);
    printf("Second:\n");
    interrupt(8,5);
    printf("Third:\n");
    interrupt(5,3);
}
```

O/P:

```
First:
Current value: 0
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Loop interrupted at 4
The loop has been re-entered 1 times after interruptions
Second:
Current value: 0
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Loop interrupted at 5
The loop has been re-entered 2 times after interruptions
Third:
```

Current value: 0

Current value: 1

Current value: 2

Current value: 3

Loop interrupted at 3

The loop has been re-entered 3 times after interruptions

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 func(int s,int e,int size)
{
   static int ts=0;
   int cs=0,i;
   for(i=s;i<e;i+=size)
   {
      printf("Current value: %d\n",i);
      cs++;
      ts++;
   }
   printf("Steps in this run: %d\n",cs);
   printf("Total steps across all runs: %d\n",ts);
}

void main()
```

```
{
    printf("First run:\n");
    func(2,8,1);
    printf("Second run:\n");
    func(5,10,2);
    printf("Third run:\n");
    func(0,6,3);
}
```

O/P:

```
First run:
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Current value: 6
Current value: 7
Steps in this run: 6
Total steps across all runs: 6
Second run:
Current value: 5
Current value: 7
Current value: 9
Steps in this run: 3
Total steps across all runs: 9
Third run:
Current value: 0
```

Current value: 3

Steps in this run: 2

Total steps across all runs: 11

‘const’ type qualifier

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>
void main()
{
   int i;
   const int a[]={10, 20, 30, 40, 50};
   for(i=0;i<5;i++)
   {
      printf("Array element %d: %d\n",i,a[i]);
      //a[i]=a[i]+1;  //compilation error
   }
}
```

O/P:

Array element 0: 10

Array element 1: 20

Array element 2: 30

Array element 3: 40

Array element 4: 50

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>
void main()
{
   int i;
   const int ul=5;
   for(i=0;i<=ul;i++)
      printf("Iteration count: %d\n",i);
}
```

O/P:

Iteration count: 0

Iteration count: 1

Iteration count: 2

Iteration count: 3

Iteration count: 4

Iteration count: 5

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>
void main()
{
   int i,j;
   const int ol=2,il=3;
   for(i=0;i<ol;i++)
   {
      for(j=0;j<il;j++)
         printf("Outer loop iteration: %d, Inner loop iteration: %d\n",i,j);
   }
}
```

O/P:

```
Outer loop iteration: 0, Inner loop iteration: 0
Outer loop iteration: 0, Inner loop iteration: 1
Outer loop iteration: 0, Inner loop iteration: 2
Outer loop iteration: 1, Inner loop iteration: 0
Outer loop iteration: 1, Inner loop iteration: 1
Outer loop iteration: 1, Inner loop iteration: 2
```

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>
void main()
{
   int a[]={10, 20, 30, 40, 50},i;
```

```
    const int *p=a;
    for(i=0;i<5;i++)
    {
        printf("Array value at %d: %d\n",i,*p);
        p++;
    }
}
```

O/P:

```
Array value at 0: 10
Array value at 1: 20
Array value at 2: 30
Array value at 3: 40
Array value at 4: 50
```

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>
void main()
{
    int r;
    const double PI=3.14;
    double a;
    for(r=1;r<=5;r++)
    {
```

```
        a=PI*r*r;
        printf("Radius: %d, Area: %f\n",r,a);
    }
}
```

O/P:

Radius: 1, Area: 3.140000

Radius: 2, Area: 12.560000

Radius: 3, Area: 28.260000

Radius: 4, Area: 50.240000

Radius: 5, Area: 78.500000

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>
void main()
{
   const int n=5;
   int c=0;
   while(c<n)
   {
      printf("Iteration: %d\n",c+1);
      c++;
   }
}
```

O/P:

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

Iteration: 5

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>
void main()
{
   int i;
   const int s=4;
   for(i=0;i<=10;i+=s)
      printf("%d ",i);
   printf("\n");
}
```

O/P:

0 4 8

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>
void main()
{
   int i,j;
   const int r=3, c=6;
   for(i=0;i<r;i++)
   {
      for(j=0;j<c;j++)
         printf("* ");
      printf("\n");
   }
}
```

O/P:

```
* * * * * *
* * * * * *
* * * * * *
```