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 var = 10;
void function() {
   int var = 20;
   printf("Local var inside function: %d\n", var);
}
int main() {
   printf("Global var in main: %d\n", var);
   function();
   return 0;
}
```

Output:

```
Global var in main: 10
Local var inside function: 20
```

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 globalVar = 10;
void add() {
   globalVar += 5;
   printf("After adding: %d\n", globalVar);
}
void subtract() {
   globalVar -= 3;
   printf("After subtracting: %d\n", globalVar);
}
int main() {
   printf("Initial globalVar: %d\n", globalVar);
   add();
   subtract();
   return 0;
}
```

Output:

```
Initial globalVar: 10
After adding: 15
After subtracting: 12
```

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 testLocalVariable() {
   int localVar = 5;
   printf("Local variable value: %d\n", localVar);
}
int main() {
   testLocalVariable();
   testLocalVariable();
   return 0;
}
```

Output:

```
Local variable value: 5
Local variable value: 5
```

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 = 200;

void sum() {

   int localVar = 45;

   int total = globalVar + localVar;

   printf("Sum of globalVar and localVar: %d\n", total);

}

int main() {

   sum();

   return 0;

}
```

Output:

```
Sum of globalVar and localVar: 245
```

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 increment() {

   counter++;

   printf("Counter after increment: %d\n", counter);

}

void decrement() {

   counter--;

   printf("Counter after decrement: %d\n", counter);

}

int main() {

   increment();
```

```
    increment();
    decrement();
    return 0;
}
```

Output:

```
Counter after increment: 1
Counter after increment: 2
Counter after decrement: 1
```

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 var = 10;
void shadowingFunction() {
    int var = 20;
    printf("Local var inside function: %d\n", var);
    printf("Global  var inside function: %d\n",var);
}
int main() {
    printf("Global  var in main: %d\n", var);
    shadowingFunction();
    return 0;
}
```

Output:

```
Global var in main: 10
Local var inside function: 20
Global var inside function: 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 CONSTANT_VAR = 400;

void printConstant() {

   printf("Constant variable value: %d\n", CONSTANT_VAR);

}

int main() {

   printConstant();

   return 0;

}
```

Output:

```
Constant variable value: 400
```

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 updateConfig() {

   configValue += 50;

   printf("Updated config value: %d\n", configValue);

}

void showConfig() {

   printf("Current config value: %d\n", configValue);

}

int main() {

   showConfig();

   updateConfig();

   showConfig();
```

```
    return 0;
}
```

Output:

```
Current config value: 100
Updated config value: 150
Current config value: 150
```

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 localVar = 42;
        printf("localVar inside block: %d\n", localVar);
    }
    return 0;
}
```

Output:

```
localVar inside block: 42
```

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 globalTotal = 0;
void calculateSum(int arr[], int size) {
    int localSum = 0;
    for (int i = 0; i < size; i++) {
        localSum += arr[i];
    }
    globalTotal += localSum;
    printf("Local sum: %d, Global total: %d\n", localSum, globalTotal);

}

int main() {

    int arr[] = {1, 2, 3, 4, 5};

    calculateSum(arr, 5);

    return 0;

}
```

Output:

```
Local sum: 15, Global total: 15
```

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 calculateSum() {

    static int sum = 0;

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

        sum += i;

    }

    printf("Cumulative sum: %d\n", sum);

}

int main() {

    calculateSum();

    calculateSum();

    return 0;

}
```

Output:

```
Cumulative sum: 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 countIterations() {

    static int count = 0;

    for (int i = 1; i <= 5; i++) {

        count++;

    }

    printf("Total iterations executed so far: %d\n", count);

}

int main() {

    countIterations();

    countIterations();

    return 0;

}
```

Output:

```
Total iterations executed so far: 5

Total iterations executed so far: 10
```

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 countInnerLoop() {

    static int innerLoopCount = 0;

    for (int i = 1; i <= 3; i++) {

        for (int j = 1; j <= 2; j++) {

            innerLoopCount++;

        }

    }

    printf("Inner loop executed %d times.\n", innerLoopCount);

}
```

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

Output:

```
Inner loop executed 6 times.
Inner loop executed 12 times.
```

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;
  int i = 0;
  while (i < 10) {
    i++;
    if (i == 5) {
      exitCount++;
      break;
    }
  }
  printf("Loop exited %d times.\n", exitCount);
}
int main() {
  trackExitCondition();
  trackExitCondition();
  return 0;
}
```

Output:

```
Loop exited 1 times.

Loop exited 2 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 trackReentry() {

   static int reentryCount = 0;

   for (int i = 0; i < 3; i++) {

      if (i == 1) {

         reentryCount++;

         printf("Loop re-entered after break.\n");

         continue;

      }

      printf("In loop: %d\n", i);

   }

   printf("Loop re-entered %d times.\n", reentryCount);

}

int main() {

   trackReentry();

   trackReentry();

   return 0;

}
```

Output:

```
In loop: 0

Loop re-entered after break.

In loop: 2

Loop re-entered 1 times.

In loop: 0

Loop re-entered after break.

In loop: 2
```

Loop re-entered 2 times.

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 trackSteps(int stepSize) {
    static int totalSteps = 0;
    for (int i = 0; i <= 10; i += stepSize) {
        totalSteps++;
    }
    printf("Total steps taken so far: %d\n", totalSteps);
}
int main() {
    trackSteps(2);
    trackSteps(3);
    return 0;
}
```

Output:

```
Total steps taken so far: 6
Total steps taken so far: 10
```

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[] = {10, 20, 30, 40, 50};
    int i;
    for (i = 0; i < 5; i++) {
        printf("Element %d: %d\n", i, arr[i]);
    }
    return 0;
}
```

Output:

```
Element 0: 10
Element 1: 20
Element 2: 30
Element 3: 40
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>
int main() {
    const int limit = 5;
    int i;
    for (i = 0; i < limit; i++) {
        printf("Iteration count: %d\n", i);
    }
    return 0;
}
```

Output:

```
Iteration count: 0
Iteration count: 1
Iteration count: 2
Iteration count: 3
Iteration count: 4
```

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;
    const int cols = 4;
    int i, j;
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("Row %d, Column %d\n", i, j);
        }
    }
    return 0;
}
```

Output:

```
Row 0, Column 0
Row 0, Column 1
Row 0, Column 2
Row 0, Column 3
Row 1, Column 0
Row 1, Column 1
Row 1, Column 2
Row 1, Column 3
Row 2, Column 0
Row 2, Column 1
Row 2, Column 2
Row 2, Column 3
```

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;
    for (int i = 0; i < 5; i++) {
        printf("Value at arr[%d]: %d\n", i, *ptr);
        ptr++;
    }
    return 0;
}
```

Output:

```
Value at arr[0]: 10
Value at arr[1]: 20
Value at arr[2]: 30
Value at arr[3]: 40
Value at arr[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>


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

Output:

```
Radius: 1, Area: 3.14
```

```
Radius: 2, Area: 12.56

Radius: 3, Area: 28.26

Radius: 4, Area: 50.24

Radius: 5, Area: 78.50
```

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;

  int count = 0;

  while (count < maxIterations) {

    printf("Iteration count: %d\n", count);

    count++;

  }

  return 0;

}
```

Output:

```
Iteration count: 0

Iteration count: 1

Iteration count: 2

Iteration count: 3

Iteration count: 4
```

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;

  const int limit = 20;

  for (int i = 0; i < limit; i += stepSize) {

    printf("i: %d\n", i);
  }
  return 0;
}
```

Output:

```
i: 0
i: 3
i: 6
i: 9
i: 12
i: 15
i: 18
```

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 = 4;
  const int cols = 6;
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      printf("* ");
    }
    printf("\n");
  }
  return 0;
}
```

Output:

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