Assignment 5

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.

Sol: 1. #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;
}
```

O/p: 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.

Sol:

```
#include <stdio.h>

int globalValue = 0;

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

void subtract(int value) {
    globalValue -= value;
    printf("After subtraction, globalValue: %d\n", globalValue);
}

void multiply(int value) {
    globalValue *= value;
    printf("After multiplication, globalValue: %d\n", globalValue);
}

void divide(int value) {
    if (value != 0) {
        globalValue /= value;
        printf("After division, globalValue: %d\n", globalValue);
```

```
    } else {
        printf("Division by zero is not allowed.\n");
    }
}
int main() {
    printf("Initial globalValue: %d\n", globalValue);
   add(10);
    subtract(5);
    multiply(3);
    divide(2);

    return 0;
}
```

O/p: Initial globalValue: 0

After addition, globalValue: 10

After subtraction, globalValue: 5

After multiplication, globalValue: 15

After division, globalValue: 7

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.

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

void localVariableDemo() {
    int localValue = 0;
    localValue++;
    printf("Local variable value: %d\n", localValue);
}

int main() {
    printf("Calling the function first time:\n");
    localVariableDemo();
    printf("Calling the function second time:\n");
    localVariableDemo();
 printf("Calling the function third time:\n");
    localVariableDemo();
 return 0;
}
```

O/p:Calling the function first time:

Local variable value: 1

Calling the function second time:

Local variable value: 1

Calling the function third time:

Local variable value: 1

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.

Sol: #include <stdio.h>

int globalValue = 50;

void calculateSum() {

    int localValue = 30;

    int sum = globalValue + localValue;

    printf("Global variable: %d\n", globalValue);

    printf("Local variable: %d\n", localValue);

    printf("Sum of global and local variables: %d\n", sum);

}

int main() {

    calculateSum();

```
    return 0;
}
```

Sol: Global variable: 50

Local variable: 30

Sum of global and local variables: 80

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.

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

int counter = 0;

void incrementByOne() {
    counter++;
    printf("Counter after incrementing by 1: %d\n", counter);
}

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

```
int main() {
    printf("Initial counter value: %d\n", counter);
    incrementByOne();
    incrementByOne();
    incrementByValue(5);
    incrementByOne();

    return 0;
}
```

O/p: Initial counter value: 0

Counter after incrementing by 1: 1

Counter after incrementing by 1: 2

Counter after incrementing by 5: 7

Counter after incrementing by 1: 8

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.

Sol: #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;
}
```

O/p: 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.

Sol: #include <stdio.h>

const int globalConstant = 42;

void printConstant() {

printf("The value of the global constant is: %d\n", globalConstant);

}

int main() {

printConstant();

return 0;

}

O/p: The value of the global constant is: 42

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.

Sol: #include <stdio.h>

int configValue = 100;

void printConfig() {

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

}

```
void calculateWithConfig(int input) {
    int result = input * configValue;
    printf("Result of input (%d) multiplied by configValue: %d\n", input, result);
}

void temporaryConfigChange(int newConfig, int input) {
    printf("Temporarily changing configValue to: %d\n", newConfig);
    int originalConfig = configValue;
    configValue = newConfig;
  calculateWithConfig(input);
 configValue = originalConfig;
    printf("Restored configValue to: %d\n", configValue);
}

int main() {
   printConfig();
    calculateWithConfig(5);
    temporaryConfigChange(50, 10);
    printConfig();
```

```
    return 0;
}
```

O/p: Current configuration value: 100

Result of input (5) multiplied by configValue: 500

Temporarily changing configValue to: 50

Result of input (10) multiplied by configValue: 500

Restored configValue to: 100

Current configuration value: 100

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.

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


int main() {
    int a = 10;


    if (a > 5) {
        int blockVar = 20;
        printf("Inside the if block: blockVar = %d\n", blockVar);
    }
```

```
    for (int i = 0; i < 3; i++) {
        int loopVar = i * 10;
        printf("Inside the for loop: loopVar = %d\n", loopVar);
    }
    return 0;
}
```

O/p: Inside the if block: blockVar = 20

Inside the for loop: loopVar = 0

Inside the for loop: loopVar = 10

Inside the for loop: loopVar = 20

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.

Sol:

```
#include <stdio.h>

int totalSum = 0;

void calculateLocalSum(int arr[], int size) {

    int localSum = 0;
```

```
    for (int i = 0; i < size; i++) {
        localSum += arr[i];
    }
    totalSum += localSum;
    printf("Local sum of the array: %d\n", localSum);
}

int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {6, 7, 8, 9, 10};

    calculateLocalSum(arr1, 5);
    calculateLocalSum(arr2, 5);

    printf("Total sum (global variable): %d\n", totalSum);

    return 0;
}
```

O/p: Local sum of the array: 15

Local sum of the array: 40

Total sum (global variable): 55

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.

Sol: #include <stdio.h>

```
int main() {

    static int cumulative_sum = 0;

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

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

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

            cumulative_sum += i;

        }

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

    }



    return 0;

}
```

O/p: Iteration 1:

Cumulative sum after iteration 1: 55

Iteration 2:

Cumulative sum after iteration 2: 110

Iteration 3:

Cumulative sum after iteration 3: 165

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.

Sol: #include <stdio.h>

```
void count_iterations() {
    static int count = 0;
    for (int i = 1; i <= 5; i++) {

        count++;
        printf("Iteration %d\n", i);
    }
 printf("Total iterations after this run: %d\n", count);
}
int main() {
```

```
    count_iterations();
    count_iterations();
    count_iterations();
   return 0;
}
```

O/p:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Total iterations after this run: 5
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Total iterations after this run: 10
Iteration 1
Iteration 2
```

Iteration 3

Iteration 4

Iteration 5

Total iterations after this run: 15

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.

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

 void count_inner_loop_executions() {

   static int total_inner_loop_count = 0;

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

     for (int j = 1; j <= 4; j++) {

        total_inner_loop_count++;

        printf("Outer loop %d, Inner loop %d\n", i, j);

     }

  }

  printf("Total inner loop executions so far: %d\n",
total_inner_loop_count);

}

int main() {
```

```
    count_inner_loop_executions();
    count_inner_loop_executions();
    count_inner_loop_executions();

    return 0;
}
```

O/p:

```
Outer loop 1, Inner loop 1
Outer loop 1, Inner loop 2
Outer loop 1, Inner loop 3
Outer loop 1, Inner loop 4
Outer loop 2, Inner loop 1
Outer loop 2, Inner loop 2
Outer loop 2, Inner loop 3
Outer loop 2, Inner loop 4
Outer loop 3, Inner loop 1
Outer loop 3, Inner loop 2
Outer loop 3, Inner loop 3
Outer loop 3, Inner loop 4
```

Total inner loop executions so far: 12

Outer loop 1, Inner loop 1

Outer loop 1, Inner loop 2

Outer loop 1, Inner loop 3

Outer loop 1, Inner loop 4

Outer loop 2, Inner loop 1

Outer loop 2, Inner loop 2

Outer loop 2, Inner loop 3

Outer loop 2, Inner loop 4

Outer loop 3, Inner loop 1

Outer loop 3, Inner loop 2

Outer loop 3, Inner loop 3

Outer loop 3, Inner loop 4

Total inner loop executions so far: 24

Outer loop 1, Inner loop 1

Outer loop 1, Inner loop 2

Outer loop 1, Inner loop 3

Outer loop 1, Inner loop 4

Outer loop 2, Inner loop 1

Outer loop 2, Inner loop 2

Outer loop 2, Inner loop 3

Outer loop 2, Inner loop 4

Outer loop 3, Inner loop 1

Outer loop 3, Inner loop 2

Outer loop 3, Inner loop 3

Outer loop 3, Inner loop 4

Total inner loop executions so far: 36

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.

Sol: #include <stdio.h>

#include <stdlib.h>

#include <time.h>

void track_loop_exit() {

  static int exit_count = 0;

  int threshold = 50;

 while (1) {

        int random_number = rand() % 100 + 1;

```
        printf("Generated number: %d\n", random_number);
 if (random_number > threshold) {


            exit_count++;
            break;
        }
    }
 printf("The loop exited %d time(s) due to the condition being true.\n",
exit_count);
}
int main() {


    srand(time(NULL));


    track_loop_exit();
    track_loop_exit();
    track_loop_exit();


    return 0;
```

}

O/p: Generated number: 16

Generated number: 20

Generated number: 70

The loop exited 1 time(s) due to the condition being true.

Generated number: 77

The loop exited 2 time(s) due to the condition being true.

Generated number: 50

Generated number: 31

Generated number: 24

Generated number: 79

The loop exited 3 time(s) due to the condition being true.

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).

Sol: #include <stdio.h>

#include <stdlib.h>

#include <time.h>

void track_loop_reentry() {

```
    static int reentry_count = 0;

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

        int random_number = rand() % 10 + 1;

        printf("Loop iteration %d, Generated number: %d\n", i,
random_number);

 if (random_number <= 5) {

            printf("Random number is %d, breaking the loop!\n",
random_number);

            reentry_count++;

            break;

        }

    }

  printf("The loop has been re-entered %d time(s) due to the break
statement.\n", reentry_count);

}


int main() {


    srand(time(NULL));

    track_loop_reentry();

    track_loop_reentry();
```

```
    track_loop_reentry();


    return 0;
}
```

O/p: Loop iteration 1, Generated number: 9

Loop iteration 2, Generated number: 6

Loop iteration 3, Generated number: 2

Random number is 2, breaking the loop!

The loop has been re-entered 1 time(s) due to the break statement.

Loop iteration 1, Generated number: 4

Random number is 4, breaking the loop!

The loop has been re-entered 2 time(s) due to the break statement.

Loop iteration 1, Generated number: 8

Loop iteration 2, Generated number: 10

Loop iteration 3, Generated number: 6

Loop iteration 4, Generated number: 1

Random number is 1, breaking the loop!

The loop has been re-entered 3 time(s) due to the break statement.

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.

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

void count_steps(int step_size) {

  static int total_steps = 0;

for (int i = 1; i <= 100; i += step_size) {

      printf("Current step: %d\n", i);

      total_steps++;

   }

 printf("Total steps taken so far: %d\n", total_steps);

}

int main() {

   count_steps(2);

   count_steps(5);

   count_steps(10);


   return 0;

}
```

O/p:

```
Current step: 1
```

Current step: 3

Current step: 5

Current step: 7

Current step: 9

Current step: 11

Current step: 13

Current step: 15

Current step: 17

Current step: 19

Current step: 21

Current step: 23

Current step: 25

Current step: 27

Current step: 29

Current step: 31

Current step: 33

Current step: 35

Current step: 37

Current step: 39

Current step: 41

Current step: 43

Current step: 45

Current step: 47

Current step: 49

Current step: 51

Current step: 53

Current step: 55

Current step: 57

Current step: 59

Current step: 61

Current step: 63

Current step: 65

Current step: 67

Current step: 69

Current step: 71

Current step: 73

Current step: 75

Current step: 77

Current step: 79

Current step: 81

Current step: 83

Current step: 85

Current step: 87

Current step: 89

Current step: 91

Current step: 93

Current step: 95

Current step: 97

Current step: 99

Total steps taken so far: 50

Current step: 1

Current step: 6

Current step: 11

Current step: 16

Current step: 21

Current step: 26

Current step: 31

Current step: 36

Current step: 41

Current step: 46

Current step: 51

Current step: 56

Current step: 61

Current step: 66

Current step: 71

Current step: 76

Current step: 81

Current step: 86

Current step: 91

Current step: 96

Total steps taken so far: 70

Current step: 1

Current step: 11

Current step: 21

Current step: 31

Current step: 41

Current step: 51

Current step: 61

Current step: 71

Current step: 81

Current step: 91

Total steps taken so far: 80

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.

Sol: #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;
}
```

O/p: 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.

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

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

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.

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

int main() {

const int outerLimit = 3;

const int innerLimit = 4;

int iterationCount = 0;

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

    for (int j = 0; j < innerLimit; j++) {

        iterationCount++;

        printf("Iteration count: %d (Outer: %d, Inner: %d)\n", iterationCount, i, j);

    }

}

 printf("\nTotal number of iterations: %d\n", iterationCount);

  return 0;

}
```

O/p:

Iteration count: 1 (Outer: 0, Inner: 0)

Iteration count: 2 (Outer: 0, Inner: 1)

Iteration count: 3 (Outer: 0, Inner: 2)

Iteration count: 4 (Outer: 0, Inner: 3)

Iteration count: 5 (Outer: 1, Inner: 0)

Iteration count: 6 (Outer: 1, Inner: 1)

Iteration count: 7 (Outer: 1, Inner: 2)

Iteration count: 8 (Outer: 1, Inner: 3)

Iteration count: 9 (Outer: 2, Inner: 0)

Iteration count: 10 (Outer: 2, Inner: 1)

Iteration count: 11 (Outer: 2, Inner: 2)

Iteration count: 12 (Outer: 2, Inner: 3)

Total number of iterations: 12

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.

Sol: #include <stdio.h>

int main() {

int arr[] = {10, 20, 30, 40, 50};

int size = sizeof(arr) / sizeof(arr[0]);

int *const ptr = arr;

```
    printf("Array elements: \n");
    for (int i = 0; i < size; i++) {
        printf("Value at ptr + %d: %d\n", i, *(ptr + i));
    }
  return 0;
}
```

O/p: Array elements:

Value at ptr + 0: 10

Value at ptr + 1: 20

Value at ptr + 2: 30

Value at ptr + 3: 40

Value at ptr + 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.

Sol:

```
#include <stdio.h>
        int main() {


    const double PI = 3.14159;
    printf("Radius\t\tArea of Circle\n");
```

```
    printf("-------------------------\n");
   for (int radius = 1; radius <= 10; radius++) {
       double area = PI * radius * radius;
         printf("%d\t\t%.2f\n", radius, area);
    }
  return 0;
}
```

O/p: Radius Area of Circle

1	3.14
2	12.57
3	28.27
4	50.27
5	78.54
6	113.10
7	153.94
8	201.06
9	254.47
10	314.16

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.

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

int main() {

    const int MAX_ITERATIONS = 5;

    int count = 0;

    while (count < MAX_ITERATIONS) {

        printf("Iteration %d\n", count + 1);

        count++;

    }

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

  return 0;

}
```

O/p: Iteration 1

Iteration 2

Iteration 3

Iteration 4

Iteration 5

Loop terminated after 5 iterations.

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.

Sol: #include <stdio.h>

```
   int main() {
  const int STEP_SIZE = 3;
  printf("Numbers with a step size of %d:\n", STEP_SIZE);
  for (int i = 0; i <= 20; i += STEP_SIZE) {
     printf("%d ", i);    }
 printf("\n");
  return 0;
}
```

O/p: Numbers with a step size of 3:

0 3 6 9 12 15 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.

Sol: #include <stdio.h>

```
int main() {
     const int ROWS = 4;
    const int COLUMNS = 6;
    printf("Rectangular pattern (%d rows, %d columns):\n", ROWS,
COLUMNS);
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
            printf("* ");
        }
        printf("\n");
    }


    return 0;
}
```

O/p:

Rectangular pattern (4 rows, 6 columns):

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