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>

// Global variable

int num = 10;

void demonstrateScope() {

// Local variable with the same name

int num = 20;

// Print the local variable

printf("Inside function - Local variable 'num': %d\n", num);

// Print the global variable

printf("Inside function - Global variable 'num': %d\n", ::num);

}

int main() {

printf("Before function call - Global variable 'num': %d\n", num);

demonstrateScope();

printf("After function call - Global variable 'num': %d\n", num);

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>

// Global variable

int num = 10;

// Function to add a value to the global variable

void addValue(int value) {

num += value;

printf("After addition, global variable 'num' is: %d\n", num);

}

// Function to subtract a value from the global variable

void subtractValue(int value) {

num -= value;

printf("After subtraction, global variable 'num' is: %d\n", num);

}

// Function to multiply the global variable by a value

void multiplyValue(int value) {

num \*= value;

printf("After multiplication, global variable 'num' is: %d\n", num);

}

// Function to divide the global variable by a value

void divideValue(int value) {

num /= value;

printf("After division, global variable 'num' is: %d\n", num);

}

int main() {

printf("Initial value of global variable 'num': %d\n", num);

addValue(5); // Add 5 to the global variable

subtractValue(3); // Subtract 3 from the global variable

multiplyValue(2);

}

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>

// Function that declares and initializes a local variable

void functionWithLocalVariable() {

int num = 5; // Local variable initialized to 5

printf("Local variable 'num' is initialized to: %d\n", num);

}

int main() {

printf("Calling the function multiple times:\n");

// Calling the function multiple times

functionWithLocalVariable();

functionWithLocalVariable();

functionWithLocalVariable();

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>

// Global variable

int globalNum = 10;

void calculateSum() {

// Local variable

int localNum = 20;

int sum = globalNum + localNum;

// Print the sum

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

// Local variable is only accessible within this function

printf("Inside function - Local variable 'localNum': %d\n", localNum);

// Global variable is accessible inside the function

printf("Inside function - Global variable 'globalNum': %d\n", globalNum);

}

int main() {

// Call the function to calculate and print the sum

calculateSum();

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>

// Global variable as a counter

int counter = 0;

// Function to increment the counter

void incrementCounter() {

counter++; // Increment the global counter

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

}

// Function to reset the counter to 0

void resetCounter() {

counter = 0; // Reset the counter

printf("Counter has been reset to: %d\n", counter);

}

int main() {

printf("Initial value of counter: %d\n", counter);

// Increment the counter multiple times

incrementCounter();

incrementCounter();

incrementCounter();

// Reset the counter

resetCounter();

// Increment again to show it starts from 0 after reset

incrementCounter();

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>

// Global variable

int num1 = 10;

void shadowingExample() {

// Local variable with the same name as the global variable

int num = 20;

// Print the local variable

printf("Inside function - Local variable 'num': %d\n", num);

// Access and print the global variable by directly referring to it

printf("Inside function - Global variable 'num': %d\n", num1);

}

int main() {

printf("Before function call - Global variable 'num': %d\n", num1);

// Call the function where shadowing happens

shadowingExample();

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>

// Global constant variable

const int a = 100;

void printGlobalConstant() {

// Access and print the global constant

printf("Inside function : %d\n", a);

}

void anotherFunction() {

// Access and print the global constant again

printf("Inside another function %d\n", a);

}

int main() {

// Access and print the global constant in the main function

printf("Inside main : %d\n",a);

// Call the functions that use the global constant

printGlobalConstant();

anotherFunction();

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>

// Global variable for configuration

int configValue = 100;

// Function to modify the configuration value

void modifyConfigValue(int newConfigValue) {

configValue = newConfigValue;

printf("Configuration value modified to: %d\n", configValue);

}

// Function to use the configuration value in an operation (e.g., addition)

void performOperation() {

int result = configValue + 50; // Example operation using the configValue

printf("Performing operation: %d\n", result);

}

// Function to print the current configuration value

void printConfigValue() {

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

}

int main() {

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

// Perform operation using the global configValue

performOperation();

// Modify the global configValue

modifyConfigValue(200);

// Perform operation again with the modified configuration value

performOperation();

// Print the final configuration value

printConfigValue();

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() {

// Variable declared inside the main function

int outerVar = 10;

// Block 1: Inside an if statement

if (outerVar > 5) {

int innerVar = 20; // Local variable inside the if block

printf("Inside if block: %d\n", innerVar);

}

// Uncommenting the line below would result in an error since innerVar is out of scope

// printf("Outside if block: innerVar = %d\n"); // This will cause a compile-time error

// Block 2: Inside a for loop

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

int loopVar = i \* 10; // Local variable inside the for loop

printf("Inside for loop: %d\n", loopVar);

}

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.

has context menu

#include <stdio.h>

// Global variable to track the total sum

int totalSum = 5;

// Function to calculate the sum of an array using a local variable

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

// Local variable to store the sum of the array elements

int localSum = 0;

// Loop to calculate the sum of array elements

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

localSum += arr[i]; // Add each element to the local sum

}

// Add the local sum to the global totalSum

totalSum += localSum;

// Print the local sum

printf("Local sum of array elements: %d\n", localSum);

}

int main() {

// Array of integers

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

// Size of the array

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

printf("Initial totalSum: %d\n", totalSum);

// Calculate the sum of the array and add it to the global total

calculateAndAddSum(arr, size);

// Print the updated global totalSum

printf("Updated totalSum after adding local sum: %d\n", totalSum);

return 0;

}

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 cumulativeSum() {

static int sum = 0; // Static variable to retain value

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

sum += i;

}

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

}

int main() {

for (int i = 0; i < 3; i++) { // Loop multiple times

cumulativeSum();

}

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 countIterations(int loopRuns) {

static int totalIterations = 0; // Static variable to retain value

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

totalIterations++;

}

printf("Total Iterations so far: %d\n", totalIterations);

}

int main() {

countIterations(5); // Run 1: 5 iterations

countIterations(3); // Run 2: 3 iterations

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 nestedLoopCounter() {

static int innerLoopCount = 0; // Static variable to retain value

for (int i = 0; i < 2; i++) { // Outer loop

for (int j = 0; j < 3; j++) { // Inner loop

innerLoopCount++;

}

}

printf("Total Inner Loop Executions: %d\n", innerLoopCount);

}

int main() {

nestedLoopCounter(); // Run 1

nestedLoopCounter(); // Run 2

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 trackExitCondition() {

static int exitCount = 0; // Static variable to retain value

int sum = 0;

while (sum < 50) {

sum += 10;

if (sum >= 50) {

exitCount++;

break;

}

}

printf("Number of times loop exited due to condition: %d\n", exitCount);

}

int main() {

trackExitCondition(); // Run 1

trackExitCondition(); // Run 2

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 trackReentry() {

static int reentryCount = 0; // Static variable to retain value

for (int i = 0; i < 10; i++) {

if (i == 5) {

reentryCount++;

break; // Interrupt the loop

}

}

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

}

int main() {

trackReentry(); // Run 1

trackReentry(); // Run 2

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.

has context menu

#include <stdio.h>

void countSteps(int stepSize) {

static int totalSteps = 0; // Static variable to retain value

for (int i = 0; i < 20; i += stepSize) {

totalSteps++;

}

printf("Total Steps so far: %d\n", totalSteps);

}

int main() {

countSteps(2); // Run 1: Step size = 2

countSteps(3); // Run 2: Step size = 3

return 0;

}