**QUE 1 : What is a function in C++? Explain the concept of function declaration, definition, and calling**

A **function** in C++ is a block of code that performs a specific task. Functions are used to break down a complex program into smaller, more manageable tasks. They allow for code reuse and better organization of the program.

**1. Function Declaration:**

A **function declaration** (or function prototype) tells the compiler about the function's name, return type, and parameters (if any) before the function is actually used in the program. It doesn't contain the function's body (the actual code to be executed); it only provides the function's signature.

**Syntax**:

return\_type function\_name(parameter\_list);

**Example**:

int add(int a, int b); // Function declaration

**2. Function Definition:**

A **function definition** provides the actual body of the function. It contains the code that will be executed when the function is called.

**Syntax**:

return\_type function\_name(parameter\_list) {

// Function body

// Code to perform the task

}

**Example**:

int add(int a, int b) {

return a + b; // Function body

}

**3. Function Calling:**

To use a function in C++, you need to **call** it in the main() function (or other functions). When calling a function, you pass arguments (if any), and it returns a value (if it's not a void function).

**Syntax**:

function\_name(argument\_list);

**Example**:

int result = add(5, 3); // Calling the add function with arguments 5 and 3

cout << result; // Outputs the result of 5 + 3, which is 8

**QUE 2 : What is the scope of variables in C++? Differentiate between local and global scope.**

**1. Local Scope:**

A **local variable** is declared within a function or a block of code (such as loops or conditionals). The variable can only be accessed within the function or block where it is declared. Once the execution leaves that function or block, the variable is destroyed, and its memory is freed.

* **Declared**: Inside a function, block, or loop.
* **Accessible**: Only within the function or block where it is defined.
* **Lifetime**: Exists only during the execution of the block or function.

**2. Global Scope:**

A **global variable** is declared outside of all functions, typically at the top of the program. These variables are accessible throughout the entire program, in all functions, unless shadowed by a local variable with the same name. They exist for the entire duration of the program's execution.

* **Declared**: Outside of all functions, usually at the top of the program.
* **Accessible**: From any function within the program.
* **Lifetime**: Exists for the entire duration of the program.

**Key Differences Between Local and Global Scope:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Local Scope** | **Global Scope** |
| **Declaration** | Declared inside a function or block. | Declared outside any function, usually at the top. |
| **Accessibility** | Can only be accessed within the function or block where it's declared. | Can be accessed by any function throughout the program. |
| **Lifetime** | Exists only during the execution of the function or block. | Exists for the entire duration of the program. |
| **Memory Allocation** | Memory is allocated when the function/block is entered and deallocated when the function/block is exited. | Memory is allocated when the program starts and deallocated when the program ends. |
| **Shadowing** | Local variables can "shadow" global variables if they have the same name. | Global variables are visible everywhere unless shadowed by local variables. |

**Example:**

#include <iostream>

using namespace std;

int globalVar = 100; // Global variable

void myFunction() {

int localVar = 10; // Local variable

cout << "Local variable inside function: " << localVar << endl;

cout << "Global variable inside function: " << globalVar << endl;

}

int main() {

int localVar = 20; // Local variable in main function

cout << "Local variable inside main: " << localVar << endl;

cout << "Global variable inside main: " << globalVar << endl;

myFunction();

return 0;

}

**Output:**

Local variable inside main: 20

Global variable inside main: 100

Local variable inside function: 10

Global variable inside function: 100

**Explanation:**

* **Global Variable**: globalVar is declared outside of any function, making it a global variable. It can be accessed both in main() and myFunction().
* **Local Variable in main()**: The variable localVar inside main() has **local scope** and is only accessible within main().
* **Local Variable in myFunction()**: The variable localVar inside myFunction() is **local to that function** and can't be accessed outside of it.

**QUE 3 : Explain recursion in C++ with an example.**

**Recursion** is a technique where a function calls itself to solve a smaller instance of the same problem. The function keeps calling itself with a modified argument until it reaches a **base case**, which terminates the recursive calls.

In simple terms, recursion involves:

* A **base case**: The condition that stops the recursion.
* A **recursive case**: Where the function calls itself with a modified argument.

**Example of Recursion: Calculating Factorial**

Let’s use the **factorial** function to demonstrate recursion. The factorial of a number n is the product of all positive integers less than or equal to n. It is defined as:

* n! = n \* (n-1) \* (n-2) \* ... \* 1
* **Base case**: 0! = 1 (or 1! = 1).

**Recursive Factorial Function in C++:**

#include <iostream>

using namespace std;

// Function to calculate factorial using recursion

int factorial(int n) {

// Base case: If n is 0 or 1, return 1

if (n == 0 || n == 1) {

return 1;

}

else {

return n \* factorial(n - 1);

}

}

int main() {

int num;

cout << "Enter a number to calculate its factorial: ";

cin >> num;

cout << "Factorial of " << num << " is: " << factorial(num) << endl;

return 0;

}

**QUE 4 : What are function prototypes in C++? Why are they used?**

**Function Prototypes in C++ (Simplified)**

A **function prototype** is like a "preview" of a function. It tells the compiler the function's **name**, **return type**, and **parameters** before the function is actually written out (defined).

**Why Use Function Prototypes?**

1. **Tells the compiler about the function**: So the compiler knows how to use the function, even before it's fully defined.
2. **Allows calling functions before they're defined**: You can use functions in your main() or other functions before actually writing them out.
3. **Prevents errors**: Helps the compiler check if you're using the correct number of arguments and the right types when calling the function.

**Example:**

#include <iostream>

using namespace std;

// Function prototype

int add(int, int);

int main() {

cout << "Result: " << add(5, 3) << endl; // Calling function

return 0;

}

// Function definition

int add(int a, int b) {

return a + b;

}

**Key Points:**

* **Function prototype**: Tells the compiler about the function (int add(int, int);).
* **Function call**: You can use add(5, 3) in main() before the full definition of add().
* **Function definition**: Later, you write out how the function works (return a + b;).

In short, function prototypes let you use functions before defining them and help prevent errors.