Function in C++

What is Function

- Functions are blocks of code designed to perform specific tasks. They allow code reusability, structure, and modular programming.
- Benefits of Functions:
  - Reusability
  - Simplicity
  - Maintainability
- Syntax:

```
return_type function_name(parameters) {
    // Function body
}
```

- Example:

```
int add(int a, int b) {
    return a + b;
}
```

Function Return Types and C++ Data Types

- Possible Return Types:
  - int: Integer values
  - double: Floating-point values
  - char: Single character
  - bool: True/False
  - void: No return value

Function Definition vs. Function Declaration

- Function Declaration (also known as the function prototype):
  - It tells the compiler about a function's name, return type, and parameters.
  - It allows the function to be called before it's defined.
  - Example:

```
int add(int, int); // Function declaration
```

- Function Definition:
  - It contains the actual code for the function, specifying what the function will do when called.
  - Must match the declaration.
  - Example:

```
int add(int a, int b) { // Function definition
    return a + b;
}
```

Passing Parameters to Functions

- Functions accept inputs called parameters or arguments, which allow customization of the function's behavior.
- Example with Multiple Parameters:

```
int multiply(int x, int y) {
    return x * y;
}
```

Two Ways of Parameter Passing

- We can pass parameter to a C++ function is the following ways:
  - Pass by Value
  - Pass by Reference
- Pass by Value:
  - When a parameter is passed by value, a copy of the actual value is passed to the function.
  - Changes made to the parameter inside the function do not affect the original value
  - Example:

```
#include <iostream>

void modifyValue(int num) {
    num = 100;  // Changes only the local copy
}

int main() {
    int x = 50;
    modifyValue(x);
    std::cout << "Value of x: " << x << std::endl;  // Output: 50
    return 0;
}
```

Pass by Reference

- When a parameter is passed by reference, the function receives a reference to the original variable.
- Changes made to the parameter inside the function affect the original variable.
- Example:

```
#include <iostream>

void modifyValue(int &num) {  // Reference to the original variable
    num = 100;  // Modifies the original variable
}

int main() {
    int x = 50;
    modifyValue(x);
    std::cout << "Value of x: " << x << std::endl;  // Output: 100
    return 0;
}
```

- What will happen if you call the function as

```
modifyValue(50);
```

Swap Function

- Look at this version of swap function:

```
void swap(int a, int b){
    int t;
    t=a;
    a=b;
    b=t;
}
```

- Call it from the main:

```
int main() {

    int p=10, q=5;
    swap(p,q);
    std::cout << "Value of p and q: " << p<<", "<<q<< std::endl;

    return 0;
}
```

New Swap Function

- Look at the new version of swap function:

```
void swap(int &a, int &b){
    int t;
    t=a;
    a=b;
    b=t;
}
```

- Call it from the main:

```
int main() {

    int p=10, q=5;
    swap(p,q);
    std::cout << "Value of p and q: " << p<<", "<<q<< std::endl;

    return 0;
}
```

Passing Arrays to Functions

```
void printArray(int arr[], int size) {
    for(int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
}
```

```
int arr[5] = {1, 2, 3, 4, 5};
int size = sizeof(arr) / sizeof(arr[0]);
printArray(arr, size);
```

Reusability

- Functions allow you to write code once and reuse it multiple times without duplication. This reduces errors and saves development time.

```
#include <iostream>
using namespace std;

// Function to calculate the square of a number
int square(int n) {
    return n * n;
}

int main() {
    cout << "Square of 4: " << square(4) << endl;
    cout << "Square of 7: " << square(7) << endl;
    cout << "Square of 10: " << square(10) << endl;  // Reusing
the same function
    return 0;
}
```

Simplicity

- Functions break complex problems into smaller, manageable parts. Each function performs a specific task, making the code easy to read and understand.

```
#include <iostream>
using namespace std;

// Function to check if a number is even
bool isEven(int n) {
    return n % 2 == 0;
}

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;

    if (isEven(num))
        cout << num << " is even." << endl;
    else
        cout << num << " is odd." << endl;

    return 0;
}
```

Maintainability

- Functions make code easy to update. If there’s a bug or an improvement needed, you can modify the function in one place without affecting the rest of the program.

```
#include <iostream>
using namespace std;

// Function to calculate tax
double calculateTax(double income) {
        double tax;
        if (income>=100000)
                tax = income * 0.25;
        else
                tax = income*0.20;
        return tax;
}

int main() {
        double yourIncome = 150000;
        cout << “Your tax: " << calculateTax(yourIncome) <<
endl;

    return 0;
}
```