

## **Function Overloading**

Multiple functions can have the same name with different parameters.

int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)

# Why Function Overloading?

- Easy to Use Same function name for different parameter types or counts.
- Improves Readability Cleaner and easier to understand code.
- **Reduces Duplication** No need to create multiple function names for similar tasks.
- **Supports Polymorphism** Enables compile-time polymorphism (flexible behavior).
- Better Organization Groups related logic under one name.

Consider the following example, which have two functions that add numbers of different type:

Example:

```
int plusFuncInt(int x, int y) {
  return x + y;
}
double plusFuncDouble(double x, double y) {
  return x + y;
}
int main() {
  int myNum1 = plusFuncInt(8, 5);
  double myNum2 = plusFuncDouble(4.3, 6.26);
}
```



```
cout << "Int: " << myNum1 << "\n";
cout << "Double: " << myNum2;
return 0;
```

Instead of defining two functions that should do the same thing, it is better to overload one.

In the example below, we overload the **plusFunc** function to work for both int and double:

#### **Example:**

```
int plusFunc(int x, int y) {
  return x + y;
}

double plusFunc(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFunc(8, 5);
  double myNum2 = plusFunc(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}</pre>
```

# Variable Scope

Now that you understand how functions work, it is important to learn how variables act inside and outside of functions. In C++, variables are only accessible inside the region they are created. This is called **scope**.

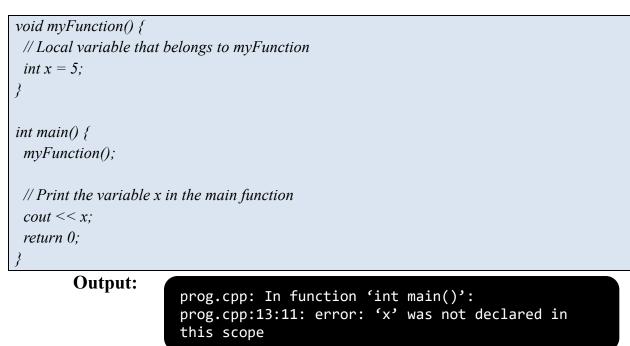


### **Local Scope**

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function:

```
void myFunction() {
    // Local variable that belongs to myFunction
    int x = 5;
    // Print the variable x
    cout << x;
}
int main() {
    myFunction();
    return 0;
}</pre>
```

**Note**: A **local variable** cannot be used outside the function it belongs to. If you try to access it outside the function, an error occurs:





### **Global Scope**

A variable created outside of a function, is called a global variable and belongs to the global scope. Global variables are available from within any scope, global and local:

```
// Global variable x
int x = 5;
void myFunction() {
    // We can use x here
    cout << x << "\n";
}
int main() {
    myFunction();
    // We can also use x here
    cout << x;
    return 0;
}</pre>
```

**Note:** If you operate with the same variable name inside and outside of a function, C++ will treat them as two separate variables; One available in the global scope (outside the function) and one available in the local scope (inside the function)

## **Example:**

```
// Global variable x
int x = 5;
void myFunction() {
    // Local variable with the same name as the global variable (x)
    int x = 22;
```



```
cout << x << "\n"; // Refers to the local variable x
}
int main() {
  myFunction();
  cout << x; // Refers to the global variable x
  return 0;
}</pre>
```

However, you should avoid using the same variable name for both globally and locally variables as it can lead to errors and confusion. In general, you should be careful with global variables, since they can be accessed and modified from any function:

#### **Example:**

```
// Global variable x
int x = 5;
void myFunction() {
    cout << ++x << "\n"; // Increment the value of x by 1 and print it
}
int main() {
    myFunction();
    cout << x; // Print the global variable x
    return 0;
}</pre>
```



Al-Mustaqbal University College of Science Artificial Intelligence Sciences Department

// The value of x is now 6 (no longer 5)