

C++ Chapter 5: Functions

Contents

C++ Chapter 5: Functions.....	1
DRY.....	1
Online Tutorials.....	2
Functions	2
Why Use Functions?.....	2
User Defined Functions	3
Function Prototypes.....	3
Example Function	3
Tutorial 5.1: Void Function	4
Tutorial 5.2: Function with a Single Parameter.....	6
Tutorial 5.3: Function with Multiple Parameters.....	8
Tutorial 5.4: Function with Return Value	10
Tutorial 5.5: Function Overloading.....	13
Tutorial 5.6: Input Function with Try Catch.....	15
Tutorial 5.7: Recursion	17
Tutorial 5.8: Pass by Reference.....	20
Tutorial 5.9: Default Arguments.....	22
Tutorial 5.10: Build a C++ Program with Multiple Files.....	24
Assignment 1: Function Header Conversion Program	26
Assignment Submission.....	26

Time required: 120 minutes

DRY

Don't Repeat Yourself (DRY) is a principle of software engineering aimed at reducing repetition of software patterns. If you are repeating any code, there is probably a better solution.

Online Tutorials

Go through the following tutorials before going through the tutorial assignments.

- [C++ Functions](#)
- [C++ Parameters/Arguments](#)
- [C++ Default Parameter](#)
- [C++ Multiple Parameters](#)
- [C++ Return Values](#)
- [C++ Function Overloading](#)
- [C++ Recursion](#)

Functions

- Functions are building blocks
- Also called modules, methods, procedures, or sub-procedures
- Like miniature programs
- Can be put together to form larger program
- One of two principle means of modularization in C++ (The other is classes)

Why Use Functions?

Divide and Conquer

- Allow complicated programs to be divided into manageable components
- Programmer can focus on just the function: develop it, debug it, and test it
- Various developers can work on different functions simultaneously

Reusability

- Can be used in more than one place in a program--or in different programs
- Avoids repetition of code, thus simplifying code maintenance
- Can be called multiple times from anywhere in the program

Components:

- Custom functions and classes that you write
- Prepackaged functions and classes available in the C++ Standard Library

User Defined Functions

There can be 4 different types of user-defined functions, they are:

- Function with no parameters and no return value
- Function with no parameters and a return value
- Function with parameters and no return value
- Function with parameters and a return value

Function Prototypes

A **function prototype** is a declaration of the function that tells the program about the type of the value which is to be returned by the function. It gives the number and type of parameters in the function. It allows the C++ compiler to find the function wherever it is in the program code.

The function prototype serves the following purposes –

1. Gives the return type of the data that the function will return.
2. Gives the number of arguments passed to the function.
3. Gives the data types of each of the passed arguments.
4. Gives the order in which the arguments are passed to the function.

The function prototype specifies the input/output interlace to the function i.e. what to give to the function and what to expect from the function.

The prototype of a function is also called the signature of the function.

Example Function

My code demonstrates a simple implementation of a function named **hello()** that prints out a greeting and a message to the console.

```

/**
 * Filename: sample_function.cpp
 * Written by:
 * Written on:
 * Purpose: Demonstrate a function in C++
 */

#include <iostream>
using namespace std;

// Function declaration for the hello() function
void hello();

int main()
{
    // Calls the hello() function
    hello();
    return 0;
}

void hello()
{
    // Prints "Hello!" followed by a newline
    cout << "Hello!" << endl;
    // Prints a message
    cout << "I am a sample function";
}

```

- **hello** is the name of the function.
- This code includes a function declaration for **hello()** at the top.
- **void** means that this function does not have a return value.
- In the **main()** function, it invokes **hello()**.
- The **hello()** function outputs **Hello!** and **I am a sample function** to the console.
- The **hello()** function exits and returns to the **main()** function.

Tutorial 5.1: Void Function

Time to create our first function. This program has two functions, **message()** and **main()**. All C++ program have a **main()** function where the program starts.

Create a C++ file with the name **void_function.cpp**

Replace with your own favorite saying.

```
1  /**
2   * Name: void_function.cpp
3   * Written by:
4   * Written on:
5   * Purpose: Demonstrate C++ void function
6   */
7  #include <iostream>
8
9  // Function prototype
10 void message();
11
12 int main()
13 {
14     std::cout << "\nFirst function call" << std::endl;
15
16     // Function call
17     message();
18     std::cout << "\nSecond function call" << std::endl;
19
20     // Function call
21     message();
22     std::cout << "\nThird function call" << std::endl;
23
24     // Function call
25     message();
26
27     return 0;
28 }
```

```

30 // Void function
31 void message()
32 {
33     // Split a cout statement and hard coded string
34     // across multiple lines for readability
35     std::cout
36         << "I love deadlines.\n"
37         << "I like the whooshing sound they make as they fly by."
38         << std::endl;
39
40     std::cout << "Douglas Adams" << std::endl;
41 }

```

Example run:

```

First function call
I love deadlines.
I like the whooshing sound they make as they fly by.
Douglas Adams

Second function call
I love deadlines.
I like the whooshing sound they make as they fly by.
Douglas Adams

Third function call
I love deadlines.
I like the whooshing sound they make as they fly by.
Douglas Adams

```

Tutorial 5.2: Function with a Single Parameter

The following program has a parameter with a single parameter.

Create a C++ program named: **FunctionSingleParameter.cpp**

```

1  /**
2   * Name: FunctionSingleParameter.cpp
3   * Written by:
4   * Written on:
5   * Purpose: Function with 1 double parameter
6   */
7
8  #include <iostream>
9  // Import iomanip library for setprecision
10 #include <iomanip>
11 // Import Math library for sqrt method
12 #include <cmath>
13 // Function prototype
14 void findSquareRoot(double num);
15
16 int main()
17 {
18     // Initialize variables to hard coded values
19     double num1{4};
20     double num2{8.9};
21     double num3{112};
22
23     // Function calls with 1 double argument
24     findSquareRoot(num1);
25     findSquareRoot(num2);
26     findSquareRoot(num3);
27     return 0;
28 }
29
30 /***** FIND SQUARE ROOT *****/
31 * void function with 1 double parameter
32 */
33 void findSquareRoot(double num)
34 {
35     double squareRoot;
36     squareRoot = sqrt(num);
37     // Long line of code broken up into multiple lines
38     // setprecision is needed to set
39     // 15 digits of precision for double in C++ cout
40     std::cout
41         << std::setprecision(15)
42         << "The square root of "
43         << num << " is: " << squareRoot
44         << std::endl;
45 }

```

A double in C++ has 15 digits of precision. cout will normally truncate the display unless you use **setprecision**. **setprecision** is part of the **iomanip** library.

Example run without setprecision:

```
The square root of 4 is: 2
The square root of 8.9 is: 2.98329
The square root of 112 is: 10.583
```

Example run with setprecision:

```
The square root of 4 is: 2
The square root of 8.9 is: 2.98328677803526
The square root of 112 is: 10.5830052442584
```

Tutorial 5.3: Function with Multiple Parameters

The following program demonstrates a function with 2 datatypes and 3 parameters.

Create a C++ program named: **FunctionMultipleParameters.cpp**


```

1  /**
2   * Name: FunctionMultipleParameters.cpp
3   * Written by:
4   * Written on:
5   * Purpose: function with 2 double and one string parameter
6   */
7  #include <iostream>
8
9  // Function prototype
10 void multiply(double num1, double num2, std::string message);
11
12 int main()
13 {
14     // Initialize variables
15     std::string message = "\nFirst function call";
16     double num1{2};
17     double num2{3.29};
18     double num3{4};
19     double num4{313};
20
21     // Function call with 2 double and 1 string parameter
22     multiply(num1, num2, message);
23
24     message = "\nSecond function call";
25     // Function call with 2 double and 1 string parameter
26     multiply(num3, num4, message);
27     return 0;
28 }
29
30 /***** MULTIPLY *****/
31 * void method with 3 parameters
32 */
33 void multiply(double num1, double num2, std::string message)
34 {
35     double product;
36     product = num1 * num2;
37     std::cout << message << std::endl;
38     // Break cout statement at << for readability
39     std::cout << "The product of "
40         << num1 << " * "
41         << num2 << " is: "
42         << product << std::endl;
43 }

```

Example run:

```
First function call  
The product of 2 * 3.29 is: 6.58  
  
Second function call  
The product of 4 * 313 is: 1252
```

Tutorial 5.4: Function with Return Value

Functions can do some work and return a value.

Example:

```

/**
 * Name: FunctionAddTwoNumbers.cpp
 * Written by:
 * Written on:
 * Purpose: two parameters and one return
 */

#include <iostream>
// Function prototype with two int parameters and an int return value
int sum(int, int);

int main()
{
    int num1;
    int num2;
    int output;

    std::cout << "Enter a number";
    std::cin >> num1;
    std::cout << "Enter a number";
    std::cin >> num2;

    // Method call with two arguments and one return value
    output = sum(num1, num2);
    std::cout << num1 << " + " << num2 << " = " << output << std::endl;
    return 0;
}

// Two parameters coming in
int sum(int x, int y)
{
    // Return value
    return x + y;
}

```

Create a C++ program named: **FunctionReturnValue.cpp**

```

1  /**
2   * Name: FunctionReturnValue.cpp
3   * Written by:
4   * Written on:
5   * Purpose: Function with a return value
6   */
7  #include <iostream>
8  // Import cmath library for ceil method
9  #include <cmath>
10 // Function one double parameter one double return
11 double getCeil(double num);
12
13 int main()
14 {
15     double num1{4.54};
16     double num2{8.2};
17     double numCeil;
18
19     std::cout << "\nPrint function return value directly"
20               << std::endl;
21     // function call with 1 argument and return value
22     std::cout << getCeil(num1) << std::endl;
23
24     std::cout << "Assign return value to variable" << std::endl;
25     // Function call with 1 argument, assign return value to variable
26     numCeil = getCeil(num2);
27     std::cout << numCeil;
28     return 0;
29 }
30
31 // Function with 1 double parameter
32 // and 1 double return value
33 double getCeil(double num)
34 {
35     double numCeil;
36     // cmath.ceil() method rounds a floating point
37     // up to the nearest integer value
38     numCeil = ceil(num);
39     return numCeil;
40 }

```

Example run:

```
Print function return value directly
5
Assign return value to variable
9
```

Tutorial 5.5: Function Overloading

Function overloading occurs when a function is defined multiple times with different signatures.

We will also introduce default parameters. If an argument is not passed, the default parameter value is used. This is only setup in the function prototype, it is not included in the function definition.

```

1  /**
2   * Name: FunctionOverloading.cpp
3   * Written by:
4   * Written on:
5   * Purpose: Overload function with different signatures
6   */
7  #include <iostream>
8  // Needed for stoi method to parse integer from string
9  #include <string>
10 // Function prototypes
11 // If this function is called without an argument, 24 is used
12 int getResult(int num = 24);
13 int getResult(int num1, int num2);
14 int getResult(int num, std::string value);
15 std::string getResult(std::string str1, std::string str2);
16
17 int main()
18 {
19     // Call method with a single int argument
20     int result = getResult(5);
21     std::cout << result << std::endl;
22
23     // Call method with no arguments, use the default parameter
24     result = getResult();
25     std::cout << result << std::endl;
26
27     // Call method with two int arguments
28     result = getResult(5, 4);
29     std::cout << result << std::endl;
30
31     // Call method with an int and a string argument
32     result = getResult(5, "12");
33     std::cout << result << std::endl;
34
35     // Call method with two string arguments
36     std::cout << getResult("Billy", "Idol") << std::endl;
37     return 0;
38 }

```

```

40 int getResult(int num)
41 {
42     return num * 2;
43 }
44
45 int getResult(int num1, int num2)
46 {
47     return num1 * num2;
48 }
49
50 int getResult(int num1, std::string string_value)
51 {
52     // std::stoi is a string library function
53     // that parses an integer from a string
54     return num1 * std::stoi(string_value);
55 }
56
57 std::string getResult(std::string str1, std::string str2)
58 {
59     return str1 + " " + str2;
60 }

```

Example run:

```

10
48
20
60
Billy Idol

```

Tutorial 5.6: Input Function with Try Catch

This method takes a prompt string, does error checking and/or input validation, then returns a value.

Create a C++ program named: **InputFunction.cpp**

```

1  /**
2   * Name: input_function.cpp
3   * Written by:
4   * Written on:
5   * Purpose: Recursive function for valid user input
6   */
7  #include <iostream>
8  // Include string library for stod convert string to double
9  #include <string>
10 // Function prototpye
11 double getDouble(std::string prompt);
12
13 int main()
14 {
15     double number;
16     // Call method with String prompt parameter
17     number = getDouble("Enter number: ");
18     std::cout << "Number: " << number << std::endl;
19 }

```



```

21  /**
22   * Prompt the user for input, return value
23   *
24   * @param String prompt the user for the value to be entered
25   */
26  double getDouble(std::string prompt)
27  {
28      std::string line_input;
29      double double_value{0};
30      // Print the prompt parameter
31      std::cout << prompt;
32
33      // Get the entire input line into a string
34      getline(std::cin, line_input);
35
36      // Catch any input mismatch exceptions
37      try
38      {
39          // Convert string to a double
40          double_value = std::stod(line_input);
41          // Return value
42          return (double_value);
43      }
44      // Catch any exceptions
45      catch (...)
46      {
47          // Return method will recurse until the user enters a number
48          return (getDouble("Enter a number: "));
49      }
50  }

```

Example run:

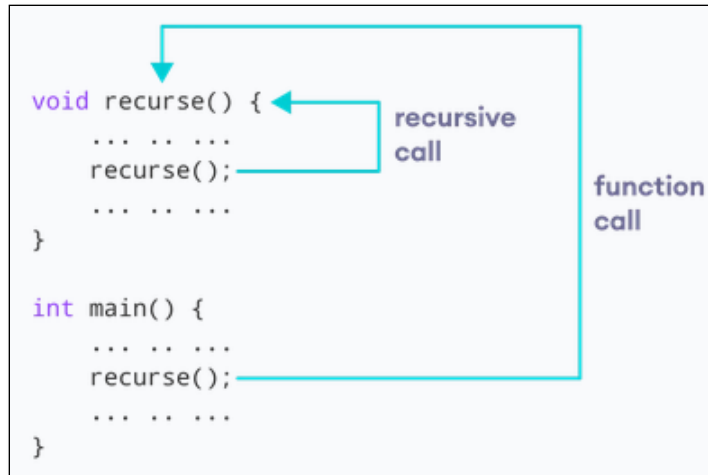
```

Enter number: w
Enter a number: d
Enter a number: 56
Number: 56

```

Tutorial 5.7: Recursion

A function can call other functions or itself. Recursion is when a function calls itself to accomplish a task. Recursion should be used sparingly.



```

1  /**
2   * @file    count_down.cpp
3   * @author  William A Loring
4   * @version V1.0.0
5   * @date   Revised: 10/17/2022
6   * @brief  Recursion demo
7   */
8
9  #include <iostream>
10 // Function prototypes allow functions to be placed in any order
11 void countDownFrom(int num);
12 void countUpTo(int from, int to);
13
14 int main()
15 {
16     std::cout << "Countdown" << std::endl;
17     countDownFrom(10);
18     std::cout << std::endl;
19
20     std::cout << "Countup" << std::endl;
21     countUpTo(0, 10);
22     std::cout << std::endl;
23 }
24
25 /**
26  * @param int num to countdown from
27  */
28 void countDownFrom(int num)
29 {
30     if (num >= 0)
31     {
32         std::cout << num << " ";
33         // Recursive call to itself
34         countDownFrom(num - 1);
35     }
36 }

```

```

38  /**
39   * @param int from to countdown from
40   * @param int to to countdown to
41   */
42  void countUpTo(int from, int to)
43  {
44      if (from <= to)
45      {
46          std::cout << from << " ";
47          // Recursive call to itself
48          countUpTo(from + 1, to);
49      }
50  }

```

Example run:

```

Countdown
10 9 8 7 6 5 4 3 2 1 0
Countup
0 1 2 3 4 5 6 7 8 9 10

```

Tutorial 5.8: Pass by Reference

We have been passing arguments by copying the value of the variable. The original variable is unchanged. A reference variable is a "reference" to an existing variable. The reference is created with the & operator. The & operator refers to the memory location of the variable, not the name.

Passing by reference in C++ allows a function to directly access and modify the original data stored in variables outside the function's scope. When you pass a variable by reference to a function, you're providing the function with the memory address of the original variable rather than a copy of its value.

In C++, references are created using the & symbol when declaring function parameters. For example:

```
void modifyValue(int &x);
```

Here, **int &x** indicates that the function `modifyValue` accepts an integer reference as its parameter.

```
void modifyValue(int &x) {  
    // Changes the original variable's value to 10  
    x = 10;  
}  
  
int main() {  
    int value = 5;  
    // Passes value by reference  
    modifyValue(value);  
    // Now, value is 10 as modified inside the function modifyValue  
    return 0;  
}
```

Passing by reference offers two main advantages:

Direct Modification: Any changes made to the reference parameter within the function affect the original variable. This allows functions to alter variable values directly, avoiding the need to return values or use global variables.

Efficiency: Passing large data structures by reference avoids unnecessary copying of data, which can improve performance and memory efficiency, especially when dealing with complex data types like arrays or objects.

The following code shows how passing variables by reference allows functions to directly manipulate the original data rather than working with copies.

```

1  /**
2   * Filename: pass_by_reference.cpp
3   * Written by:
4   * Written on:
5   * Purpose: & pass variables by reference
6   */
7
8  #include <iostream>
9  // Function declaration taking two integer references as parameters
10 void aFunction(int &num1, int &num2);
11
12 int main()
13 {
14     int num1{5};
15     int num2{7};
16
17     // Call aFunction, pass num1 and num2 by reference
18     aFunction(num1, num2);
19
20     std::cout << num1 << ", " << num2 << std::endl;
21     return 0;
22 }
23
24 // Modifies the values of num1 and num2 received by reference
25 void aFunction(int &num1, int &num2)
26 {
27     // These parameters are passed by reference,
28     // the original variables in main() will be modified
29     // The name doesn't matter, these are positional arguments
30     num1 = 2;
31     num2 = 2;
32 }

```

Example run:

```
2, 2
```

Tutorial 5.9: Default Arguments

It is possible to call a function and not have to pass all the arguments.

Here is an example function prototype with a default argument. If the second argument is not passed, the value of repeats is 1.

```
greeting(string greet, int repeats = 1);
```

Create the following example of a program with default arguments.

```
/**
 * Name: function_default_arguments.cpp
 * Written by:
 * Written on:
 * Purpose: Demonstrate default arguments in C++
 */
#include <iostream>
// Function prototype includes the default parameter
void greeting(std::string greet, int repeats = 1);

int main(void)
{
    greeting("Hello", 4);
    // Function call with one argument, default argument will be used
    greeting("Good morning");
    greeting("Hola amigo!", 10);
    return 0;
}

// Function definition does not include the default parameter
void greeting(std::string greet, int repeats)
{
    for (int i = 0; i < repeats; i++)
        std::cout << greet << std::endl;
}
```

Example run:

```
Hello
Hello
Hello
Hello
Good morning
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
Hola amigo!
```

Tutorial 5.10: Build a C++ Program with Multiple Files

As our code becomes increasingly complex, it is helpful to organize our code into separate files.

This example demonstrates the separation of a function into separate files.

1. Create a separate folder for your MyFunction project.
2. Open the folder with VSCode.

Create the following files.

main.cpp - includes the header file and uses the function.

```
1  // Include the header file containing function declaration
2  #include "my_function.h"
3
4  int main() {
5      // Call the function defined in my_function.cpp
6      myFunction();
7      return 0;
8  }
```

my_function.cpp - defines the function's implementation.


```

1  #ifndef MYFUNCTION_H
2  #define MYFUNCTION_H
3
4  // Function declaration
5  void myFunction();
6
7  #endif // MYFUNCTION_H header guard
8  // Prevents the header from being imported more than once.

```

my_function.h – header file that contains the function declaration.

```

1  #include <iostream>
2  // Include the header file containing function declaration
3  #include "my_function.h"
4
5  // Function definition
6  void myFunction()
7  {
8      std::cout << "This is my function!" << std::endl;
9  }

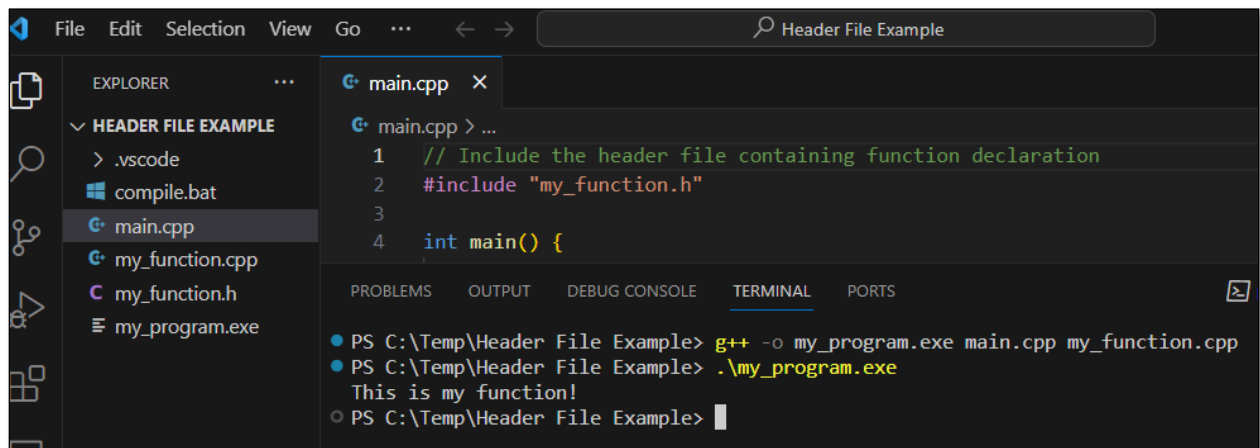
```

Compiling multiple files can be done in a few different ways. The easiest is to use a batch file.

1. In VSCode → Create a new file named **compile.bat**
2. Copy and paste or type the following.

```
g++ -o main.exe main.cpp my_function.cpp
```

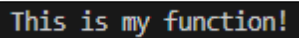
-o <output> specifies the output file name. For example, **-o main.exe** sets the output file name as **my_program.exe**



1. Go to the **Terminal** menu → new **Terminal**
2. At the terminal prompt → start typing the first couple of letters of **main.exe** → press **Tab** until you see the right filename. Press Enter.

NOTE: For future C++ programs with multiple files, you can copy, modify and reuse this batch file.

Example run:



```
This is my function!
```

Assignment 1: Function Header Conversion Program

Create a C++ program that does any measurement conversion with separate function header file, function file, and main file.

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.