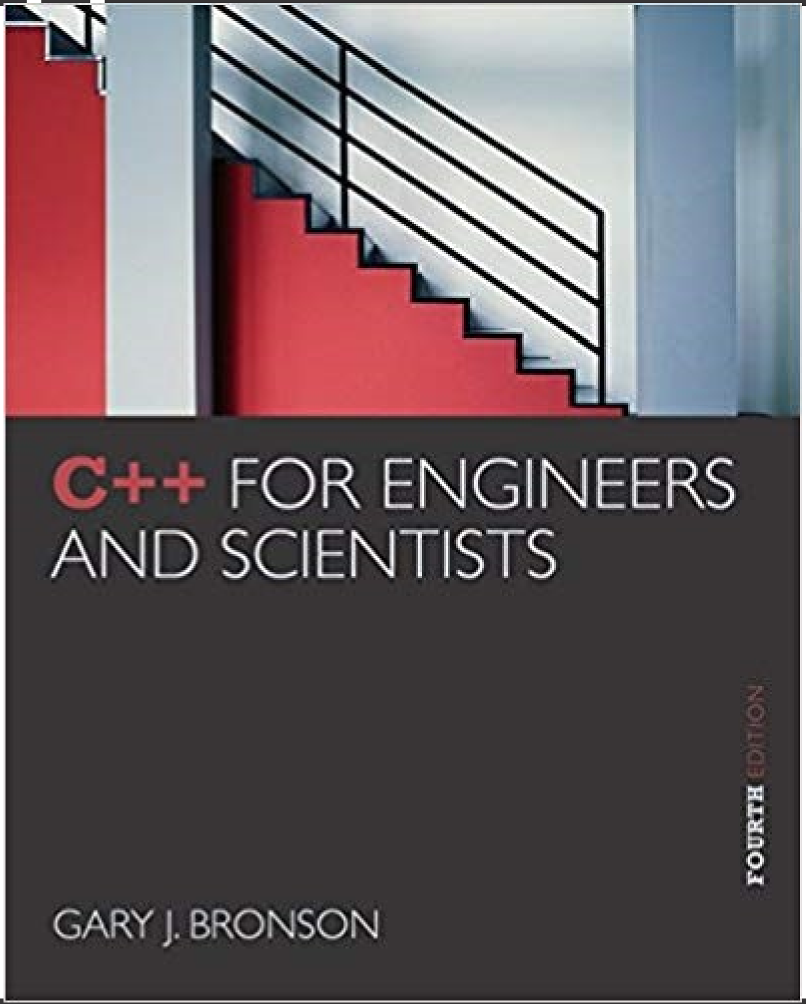
ELEG 1043
Computer Applications in Engineering



Chapter 6:
Modularity Using Functions

C++ FOR ENGINEERS
AND SCIENTISTS

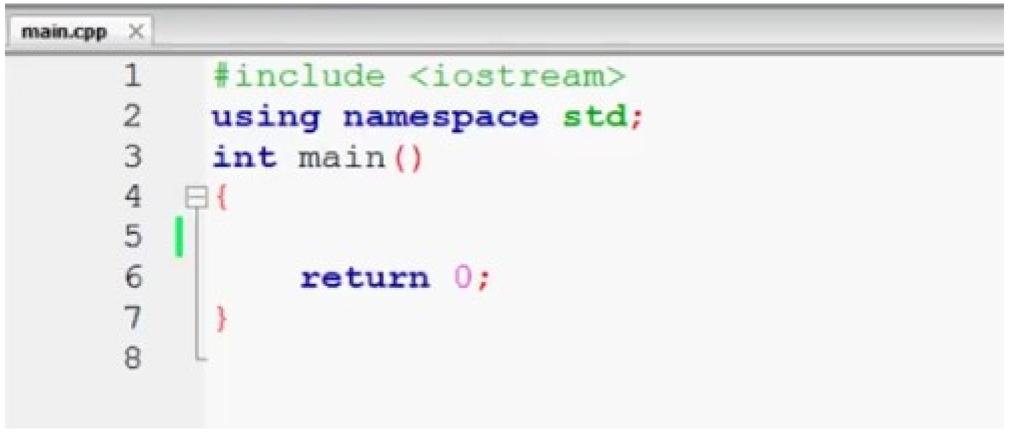
Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Objectives

- In this chapter, you will learn about:
  - Function and parameter declarations
  - Returning a single value
  - Returning multiple values
  - Rectangular to polar coordinate conversion
  - Variable scope
  - Variable storage categories
  - Common programming errors

Function



```
#include <iostream>
using namespace std;
int main()
{

    return 0;
}

```

https://www.youtube.com/watch?v=S_82v5ZuCO4

Function and Parameter Declarations

- Interaction with a function includes:
  - Passing data (Arguments) to a function correctly when its called
  - Returning values from a function when it ceases operation
- A function is called by (1) giving the function's name and (2) passing arguments in the parentheses following the function name

$$\underbrace{\textit{function-name}}_{\substack{\text{This identifies the}\\ \text{called function}}}\ (\underbrace{\textit{data passed to function}}_{\substack{\text{This passes data}\\ \text{to the function}}});$$

Figure 6.1 Calling and passing data to a function

Function and Parameter Declarations (continued)

- Before a function is called, it must be declared to function that will do calling
- Declaration statement for a function is referred to as function prototype
- Function prototype tells calling function:
  - Type of value that will be formally returned, if any
  - Data type and order of the values (arguments) the calling function should transmit to the called function
- Function prototypes can be placed with the variable declaration statements above the calling function name or in a separate header file

Calling a Function

- Requirements when calling a function include:
  - Using the name of the function
  - Enclosing any data passed (Arguments) to the function in the parentheses following the function name, using the same order and type declared in the function prototype

Calling a Function (continued)

- The items enclosed in the parentheses are called **arguments** of the called function

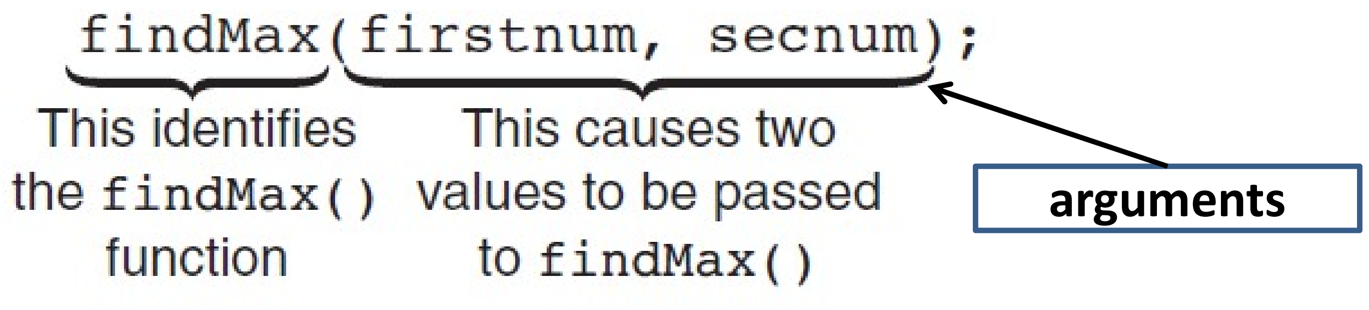

findMax(firstnum, secnum);
This identifies the findMax() function
This causes two values to be passed to findMax()
arguments

Figure 6.2 Calling and passing two values to `findMax()`

Calling a Function (continued)

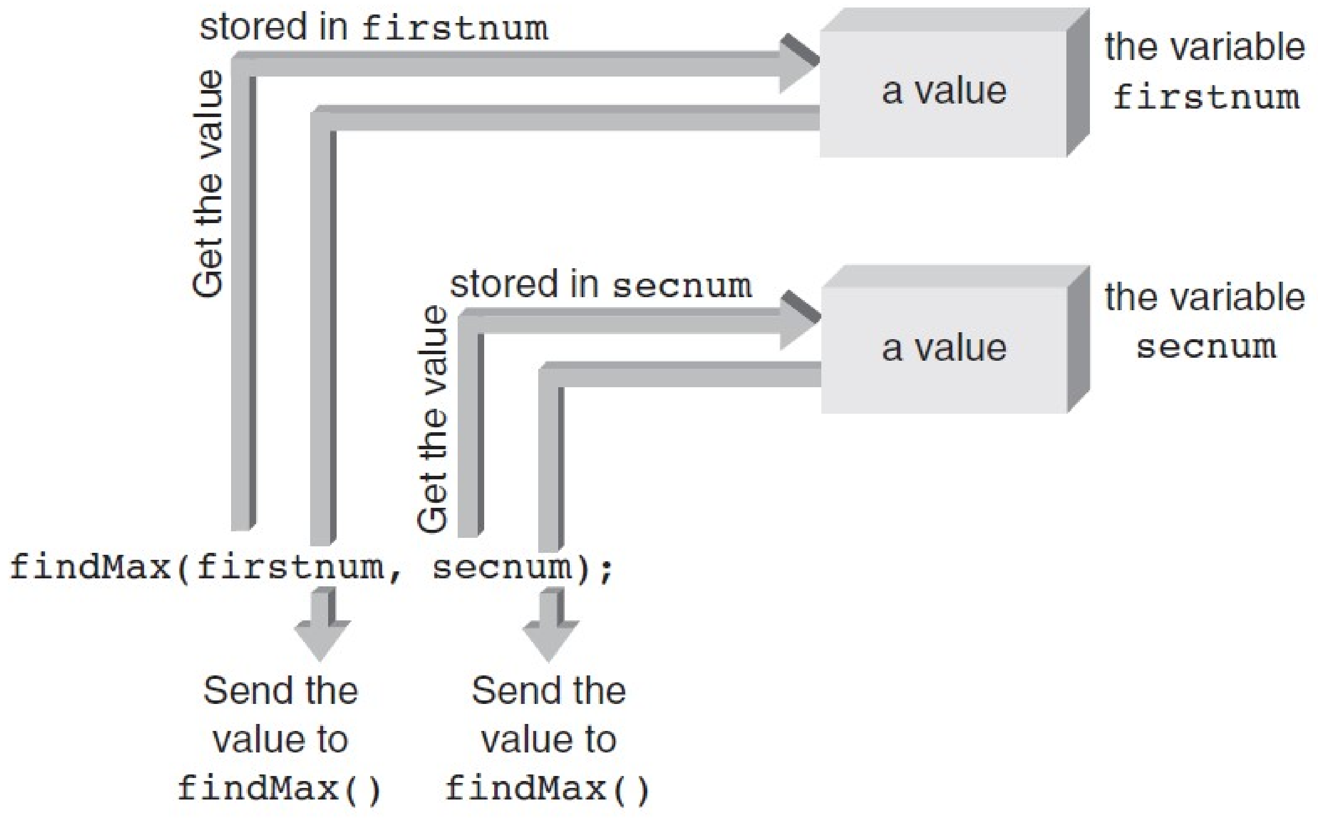


Figure 6.3 The `findMax()` function receives actual values

Defining a Function

- Every C++ function consists of two parts:
  - **Function header**
  - **Function body**
- Function header's purpose:
  - Identify data type of value function returns, provide function with name, and specify number, order, and type of arguments function expects
- Function body's purpose:
  - To operate on passed data and return, at most, one value directly back to the calling function

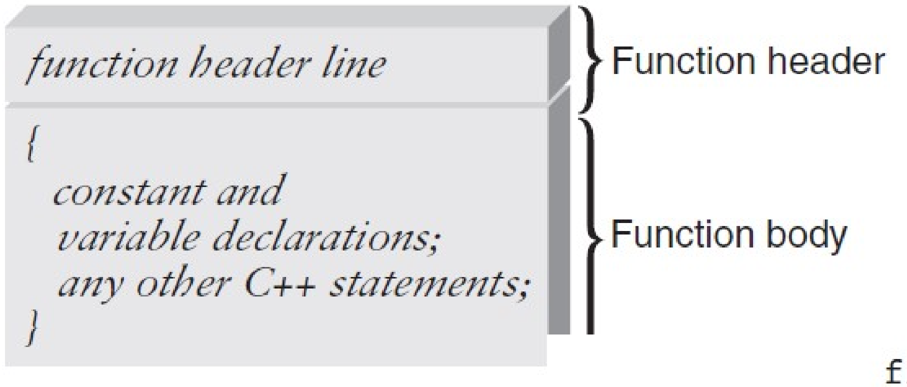
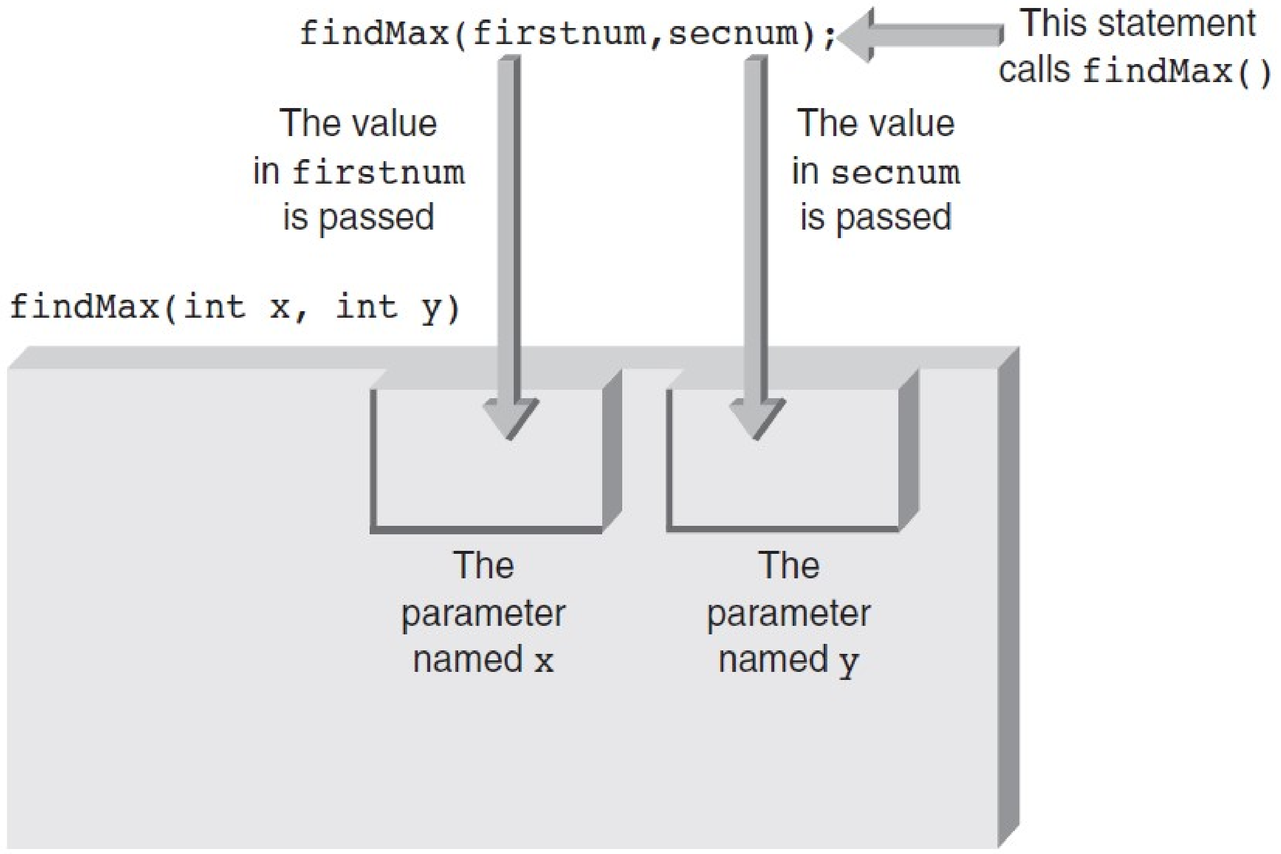


Figure 6.4 The general format of a function



findMax(firstnum,secnum);
This statement calls findMax()
The value in firstnum is passed
The value in secnum is passed
findMax(int x, int y)
The parameter named x
The parameter named y

Figure 6.5 Storing values ir parameters

Placement of Statements

- General rule for placing statements in a C++ program:
  - All preprocessor directives, named constants, variables, and functions must be declared or defined before they can be used
    - Preprocessor directives are lines included in the code of programs preceded by a hash sign (#)
      - Example: #define TABLE_SIZE 100
  - Although this rule permits placing both preprocessor directives and declaration statements throughout the program, doing so results in poor program structure

Functions with Empty Parameter Lists

- Although useful functions having an empty parameter list are extremely limited, they can occur
- Function prototype for such a function requires writing the keyword **`void`** or nothing at all between the parentheses following the function’s name
- Examples:

```
int display();
int display(void);
```

Default Arguments

- C++ provides **default arguments** in a function call for added flexibility
  - Primary use: to extend parameter list of existing functions without requiring any change in calling parameter lists already used in a program
  - Listed in the function prototype and transmitted automatically to the called function when the corresponding arguments are omitted from the function call
  - Example: Function prototype with default arguments

```
void example(int, int = 5, double = 6.78)
```

Reusing Function Names (Overloading)

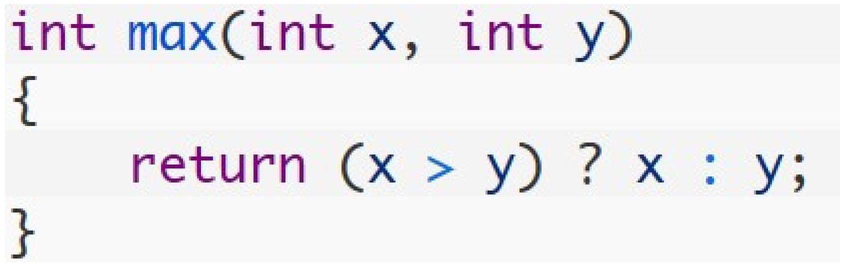
- C++ provides the capability of using the same function name for more than one function
  - Referred to as **function overloading**
- Only requirement for creating more than one function with same name:
  - Compiler must be able to determine which function to use based on the parameters' data types (not the data type of the return value, if any)
  - Which of the functions is called depends on the argument type supplied at the time of the call

Function Templates

- Function template: Single complete function that serves as a model for a family of functions
  - Function from the family that is actually created depends on the specific function call
- Generalize the writing of functions that perform essentially the same operation, but on different parameter data types
- Make it possible to write a general function that handles all cases but where the compiler can set parameters, variables, and even return type based on the actual function call

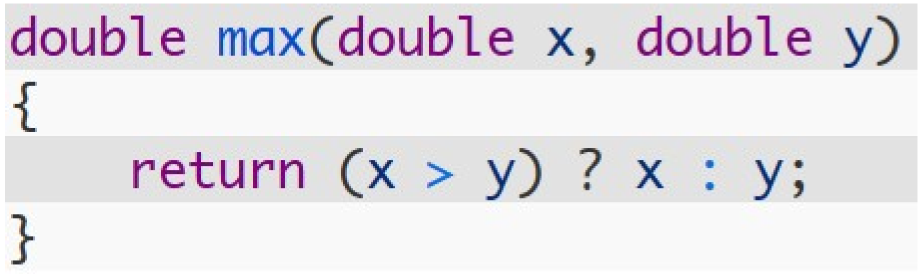
Function Templates

(1)



```
int max(int x, int y)
{
    return (x > y) ? x : y;
}
```

(2)

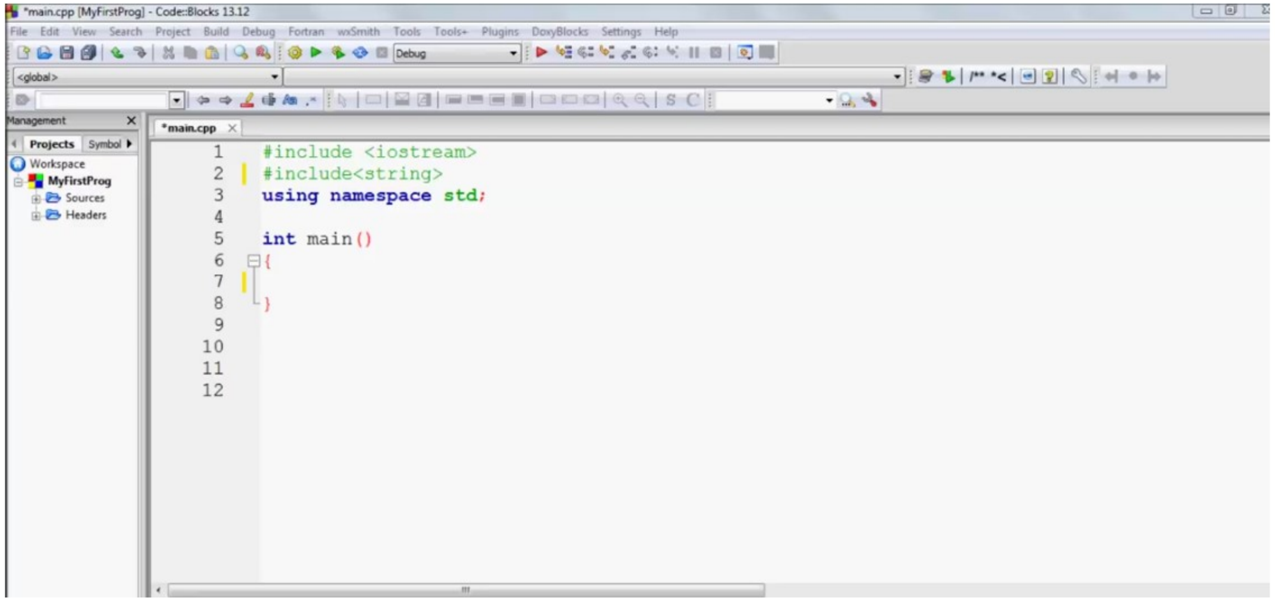


```
double max(double x, double y)
{
    return (x > y) ? x : y;
}
```

Function Templates

```
template <typename T>
T max (T a, T b)
{
        T result;
        result = (a>b)? a : b;
        return (result);
}
```

Example: Function Templates


*main.cpp [MyFirstProg] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbol
Workspace
MyFirstProg
Sources
Headers
*main.cpp

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

int main()
{

}
```

https://www.youtube.com/watch?v=HTy3D98C188