Objectives

- In this chapter, you will learn about:
  - Sorting 3 numbers review
  - Function overview
  - Function and parameter declarations
  - Function with empty parameter lists
  - Default arguments
  - Put sorting into a function
Sorting of three numbers

- Solution 1 using if-else
  - Check all possible combinations
- Solution 2 using if
  - Find the first minimum and swap the results
  - Find the second minimum,
  - ...

- Questions, what if we want to reuse the algorithm and sort another three numbers?
Functions

- Functions allow us to organize complex programs
- Allow use to reuse code
- A function should have a well defined operation
- It takes values as inputs (arguments)
- It outputs some value (return value)
- i.e.
  - `setw(5)`;
  - `setprecision(2)`;
  - `ceil(3.5)`;
  - `pow(3.0, 4.0)`;
  - `log10(1231)`;
  - ...
Function

- Function prototype
  - Declaration statement
- Function body
  - Function implementation
- Function call
  - Action summoning
Function Prototype

double my_function( int , int );
double my_function(int x, int y)
{
    double result;
    result = x+y;
    return result;
}
Function Call

```c
int main()
{
    double a, b, z;
    ...
    z = my_function(a, b);
    ...
}
```

Another function

Argument list

Function call
Function and Parameter Declarations

• Interaction with a function includes:
  – **Passing** data to a function correctly when its called
  – **Returning** values from a function when it ceases operation

• A function is called by giving the function’s name and passing arguments in the parentheses following the function name

```
function-name (data passed to function);
```

This identifies the called function
This passes data to the function

**Figure 6.1** Calling and passing data to a function
Function and Parameter Declarations (continued)

Program 6.1

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

void findMax(int, int); // the function declaration (prototype)

int main()
{
    int firstnum, secnum;

    cout << "\nEnter a number: ";
    cin >> firstnum;
    cout << "Great! Please enter a second number: ";
    cin >> secnum;

    findMax(firstnum, secnum); // the function is called here

    return 0;
}
```
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 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 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

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

Figure 6.3 The findMax() function receives actual values
Calling a Function (continued)

• Figure 6.3:
  – The `findMax()` function does not receive the variables named `firstnum` and `secnum` and has no knowledge of the variable names
  – The function receives the values in these variables and must then determine where to store those values before it does anything else
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
Defining a Function (continued)

**Figure 6.4** The general format of a function

```
function header line
{
    constant and variable declarations; any other C++ statements;
}
```

**Figure 6.5** Storing values in parameters

```
findMax(firstnum, secnum);

findMax(int x, int y)
```

This statement calls `findMax()`.
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
  – 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 parenthesis following the function’s name.

• i.e.

```cpp
void foo();
void foo(void);
```
Default Arguments

• C++ provides **default arguments** in a function call for added flexibility

• Primary use of default arguments is to extend parameter list of existing functions without requiring any change in calling parameter lists already used in a program

• Default arguments are listed in the function prototype and transmitted automatically to the called function when the corresponding arguments are omitted from the function call
Default Arguments (continued)

• Example: Function prototype with default arguments

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

• Sample valid calls to the `example` function

```c
example(7, 2, 9.3) // no defaults used
example(7, 2)      // same as example(7, 2, 6.78)
example(7)         // same as example(7, 5, 6.78)
```
Sorting 3 numbers using a function

- Function prototype
  - void sort(int, int, int);
- Function body
  - void sort(int a, int b, int c);
- Function call
  - sort(a, b,c);