COMSATS University
Islamabad
(Lahore Campus)

CSC103- Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 6:
User-Defined Functions

Objectives

- In this chapter, you will:
  - Learn about standard (predefined) functions
  - Learn about user-defined functions
  - Examine value-returning functions
  - Explore how to construct and use a value-returning, user-defined function
  - Learn about function prototypes

Introduction

- Functions are often called modules
  - Modular programming refers to a programming approach, which makes heavy use of functions for solving problems.
- They are like miniature programs (sub-programs) that can be combined to form larger programs
- They allow complicated programs to be divided into manageable pieces.

Predefined Functions

- In C++, a function is similar to that of a function in algebra
  - It has a name
  - It does some computation
- Some of the predefined mathematical functions are:

```
sqrt(x)
pow(x, y)
floor(x)
```

Predefined Functions (cont'd.)

- Predefined functions are organized into separate libraries
  - I/O functions are in `iostream` header
  - Math functions are in `cmath` header
- To use predefined functions, you must include the header file using an `include` statement
- See Table 6-1 in the text for some common predefined functions

EXAMPLE 6-1

```
//How to use predefined functions.
#include <iostream>
#include <cmath>
#include <cctype>

using namespace std;

int main()
{
    int    x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static_cast<char>(toupper('a'))
         << endl;                                          //Line 1

    u = 4.2;                                               //Line 2
    v = 3.0;                                               //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;               //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                           //Line 5

    u = u + pow(3.0, 3);                                   //Line 6
    cout << "Line 7: u = " << u << endl;                   //Line 7

    x = -15;                                               //Line 8
    cout << "Line 9: Absolute value of " << x
         << " = " << abs(x) << endl;                       //Line 9

    return 0;
}
```

Sample Run:

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5: 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15
```

User-Defined Functions

- Value-returning functions: have a return type
  - Return a value of a specific data type using the `return` statement
- Void functions: do not have a return type
  - *Do not* use a `return` statement to return a value

Value-Returning Functions

To use these functions, you must:

- Include the appropriate header file in your program using the include statement
- Know the following items:
  - Name of the function
  - Number of parameters, if any
  - Data type of each parameter
  - Data type of the value returned: called the type of the function

Value-Returning Functions (cont'd.)

- Can use the value returned by a value-returning function by:
  - **Saving it for further calculation**
  - Using it in some calculation
  - Printing it
- A value-returning function is used in an assignment or in an output statement

Value-Returning Functions (cont'd.)

- Heading (or function header): first line of the function
  - Example: `int abs(int number)`
- **Formal parameter**: variable declared in the function heading
  - Example: `number`
- **Actual parameter/Actual argument**: variable or expression listed in a *call* to a function
  - Example: `x = pow(u, v)`

Syntax: Value-Returning Function

- Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

- `functionType` is also called the data type or return type

Syntax: Formal Parameter List

```
dataType identifier, dataType identifier, ...
```

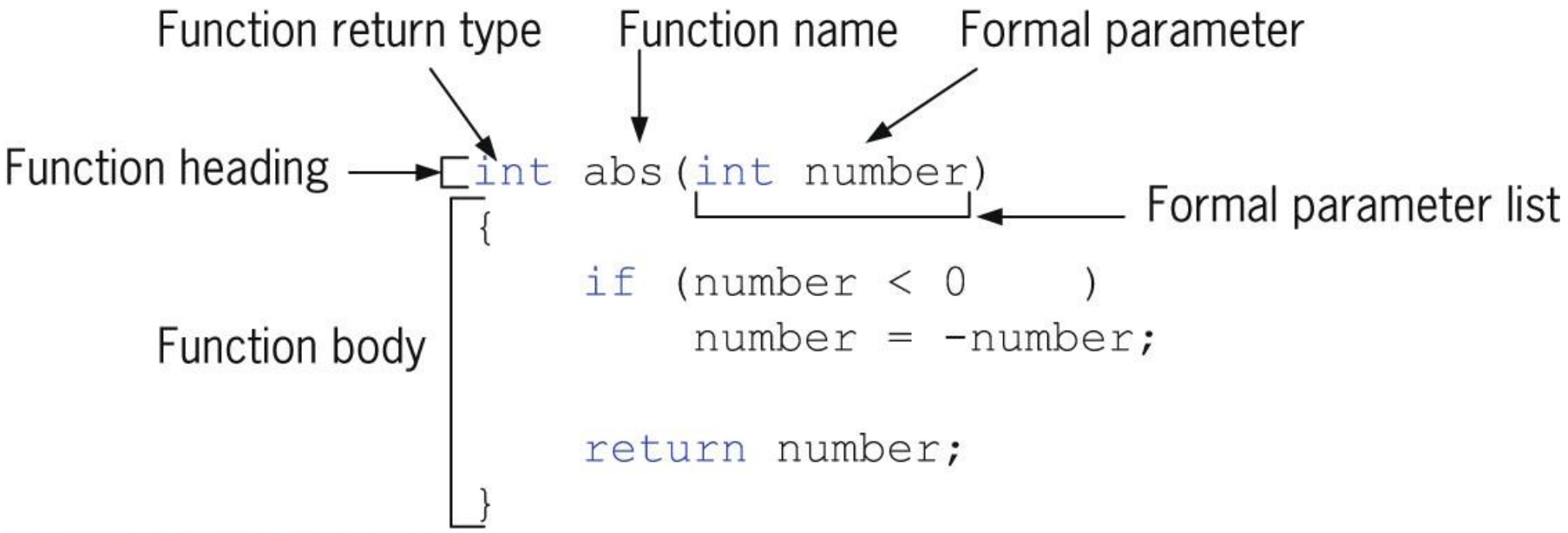

FIGURE 6-1 Various parts of the function abs

Function Call

- Syntax to call a value-returning function:

```
functionName(actual parameter list)
```

Syntax: Actual Parameter List

- Syntax of the actual parameter list:

```
expression or variable, expression or variable, ...
```

- Formal parameter list can be empty:

```
functionType functionName()
```

- A call to a value-returning function with an empty formal parameter list is:

```
functionName()
```

`return` Statement

- Function returns its value via the `return` statement
  - It passes this value outside the function

Syntax: `return` Statement

- Syntax:

```
return expr;
```

- In C++, `return` is a reserved word
- When a return statement executes
  - Function immediately terminates
  - Control goes back to the caller
- When a `return` statement executes in the function `main`, the program terminates

Syntax: `return` Statement (cont'd.)

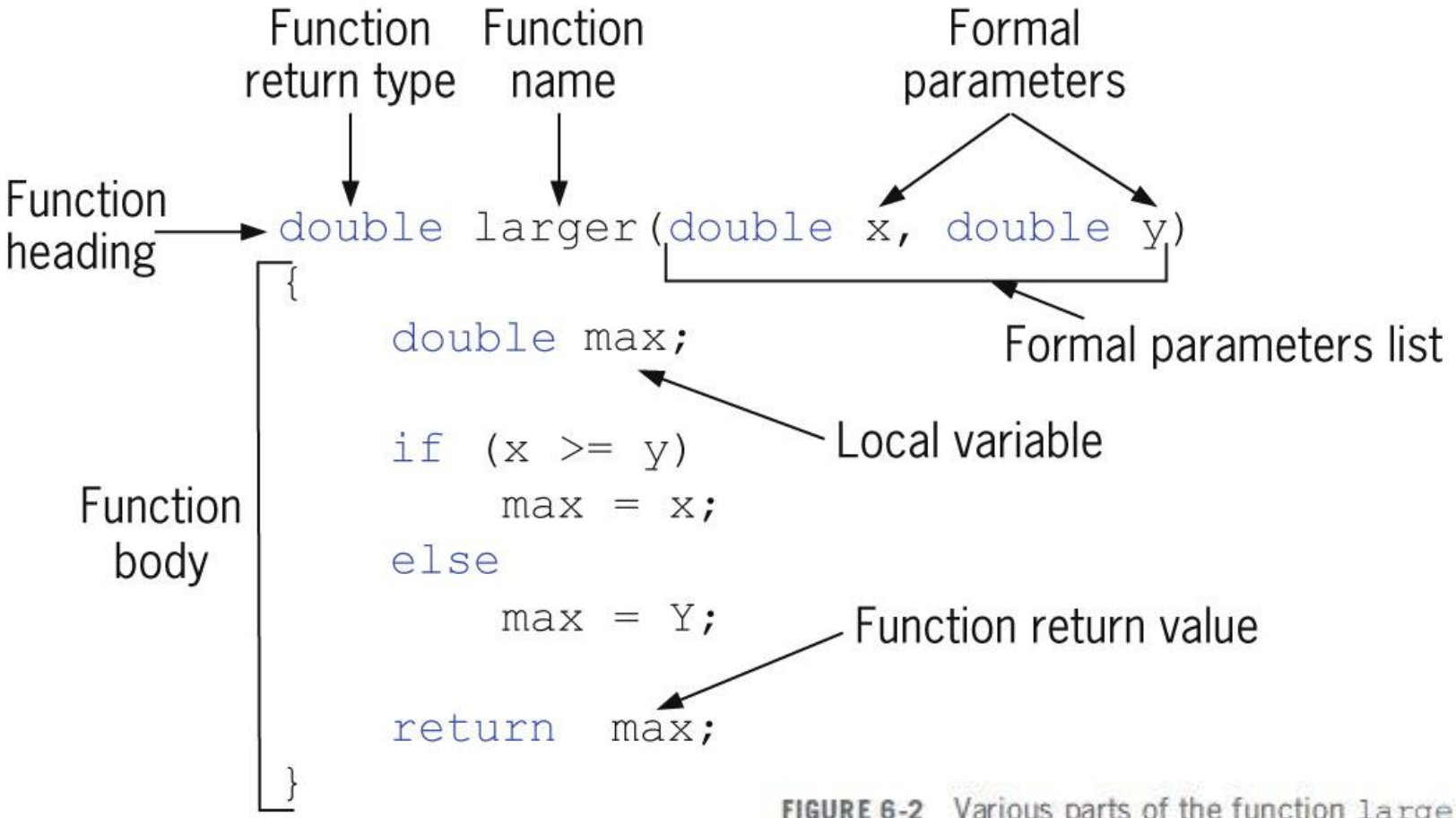

FIGURE 6-2 Various parts of the function `larger`

Syntax: `return` Statement (cont'd.)

Suppose that `num`, `num1`, and `num2` are `double` variables. Also suppose that `num1 = 45.75` and `num2 = 35.50`. Figure 6–2 shows various calls to the function `larger`.

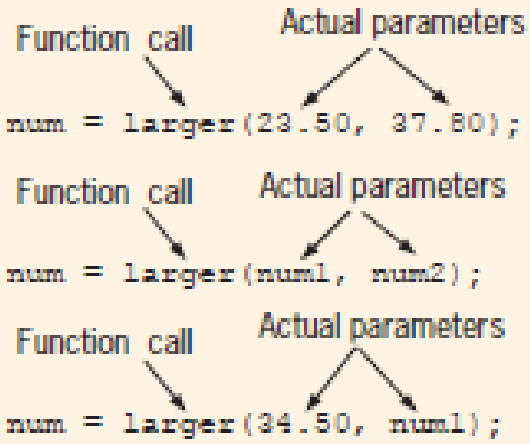

FIGURE 6-2 Function calls

Function Prototype

- Function prototype: function heading without the body of the function
- Syntax:

```
functionType functionName(parameter list);
```

- Not necessary to specify the variable name in the parameter list
- Data type of each parameter must be specified

```
double larger(double x, double y);
```

Value-Returning Functions: Some Peculiarities

- Make sure a value-returning function always returns a value in all cases.

```
int secret(int x)
{
    if (x > 5)              //Line 1
        return 2 * x;       //Line 2
}
```

A correct definition of the function secret is:

```
int secret(int x)
{
    if (x > 5)              //Line 1
        return 2 * x;       //Line 2

    return x;               //Line 3
}
```

Value-Returning Functions: Some Peculiarities (cont'd.)

- A function can return only 1 value using the return statement, i.e. `return x, y;  //only the value of y will be returned`

```
int funcRet1()
{
    int x = 45;

    return 23, x;   //only the value of x is returned
}

int funcRet2(int z)
{
    int a = 2;
    int b = 3;

    return 2 * a + b, z + b; //only the value of z + b is returned
}
```

Flow of Compilation and Execution

- Execution always begins at the first statement in the function `main`
- Other functions are executed only when called
- Function prototypes appear before any function definition
  - Compiler translates these first
- Compiler can then correctly translate a function call

Flow of Compilation and Execution (cont'd.)

- Function call transfers control to the first statement in the body of the called function
- When the end of a called function is executed, control is passed back to the point immediately following the function call
  - Function's returned value replaces the function call statement

Summary

- Functions (modules) divide a program into manageable tasks
- C++ provides standard, predefined functions
- Two types of user-defined functions: value-returning functions and void functions
- Variables defined in a function heading are called formal parameters
- Expressions, variables, or constant values in a function call are called actual parameters

Summary (cont'd.)

- Function heading and the body of the function are called the definition of the function
- A value-returning function returns its value via the `return` statement
- A prototype is the function heading without the body of the function
- User-defined functions execute only when they are called