CS1002 – Programming Fundamentals

Lecture # 21
Monday, November 14, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus

Muhammad Yousaf

Objectives

In this week/classes, you will:

- Learn about standard (predefined) functions and discover how to use them in a program
- Learn about user-defined functions
- Examine value-returning functions, including actual and formal parameters
- Explore how to construct and use a value-returning, user-defined function in a program

Modular Programming

- **Modular programming:** Breaking a program up into smaller, manageable functions or modules
- **Function:** A collection of statements to perform a task
- **Motivation for modular programming**
  - Improves maintainability of programs
  - improves readability of the programs
  - code reusability, Code can be **re-used** (even in different programs)
    - **Different people** can work on **different functions** simultaneously

Introduction

- **Boss to worker analogy**
  - A boss (the calling function or caller) asks a worker (the called function) to perform a task and return (i.e., report back) the results when the task is done

Introduction

- **Functions** are like **building blocks**
  - Called **modules**
  - Can be put together to form a larger program

Predefined Functions

- In algebra, a function is defined as a rule or correspondence between values, called the function's arguments, and the unique value of the function associated with the arguments
  - If f(x) = 2x + 5, then f(1) = 7, f(2) = 9, and f(3) = 11
  - 1, 2, and 3 are arguments
  - 7, 9, and 11 are the corresponding values

Predefined Functions (cont'd.)

- Some of the predefined mathematical functions are:
  - sqrt(x);
  - sqrt(4);
  - sqrt(3 - 6x);
  - pow(x, y);
  - floor(x);
- Predefined functions are organized into **separate libraries**
- I/O functions are in **iostream** header
- Math functions are in **cmath/math** header

Predefined Functions (cont'd.)

- **pow(x,y)** calculates $\mathbf{x^y}$
  - pow(2, 3) = 8.0
  - Returns a value of type **double**
  - x and y are the **parameters (or arguments)**
    - This function has two parameters
- **sqrt(x)** calculates the nonnegative square root of x, for x >= 0.0
  - sqrt(2.25) is 1.5
  - Type **double**

Predefined Functions (cont'd.)

- The floor function **floor(x)** calculates largest whole number not greater than x
  - floor(48.79) is 48.0
  - Type double
  - Has only one parameter

Predefined Functions (cont'd.)

TABLE 6-1 Predefined Functions

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Predefined Functions (cont'd.)

TABLE 6-1 Predefined Functions (continued)

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`islower(x)`	`<cctype>`	Returns `true` if x is a lowercase letter; otherwise, it returns `false`; `islower('h')` is `true`	`int`	`int`
`isupper(x)`	`<cctype>`	Returns `true` if x is a uppercase letter; otherwise, it returns `false`; `isupper('K')` is `true`	`int`	`int`
`pow(x, y)`	`<cmath>`	Returns x^y; if x is negative, y must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of x; x must be nonnegative: `sqrt(4.0) = 2.0`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of x if x is uppercase; otherwise, it returns x	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of x if x is lowercase; otherwise, it returns x	`int`	`int`

Predefined Functio

Example 6-1

```
//How to use predefined function
#include <iostream>
#include <cmath>
#include <cctype>
#include <cstdlib>
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;
}
```

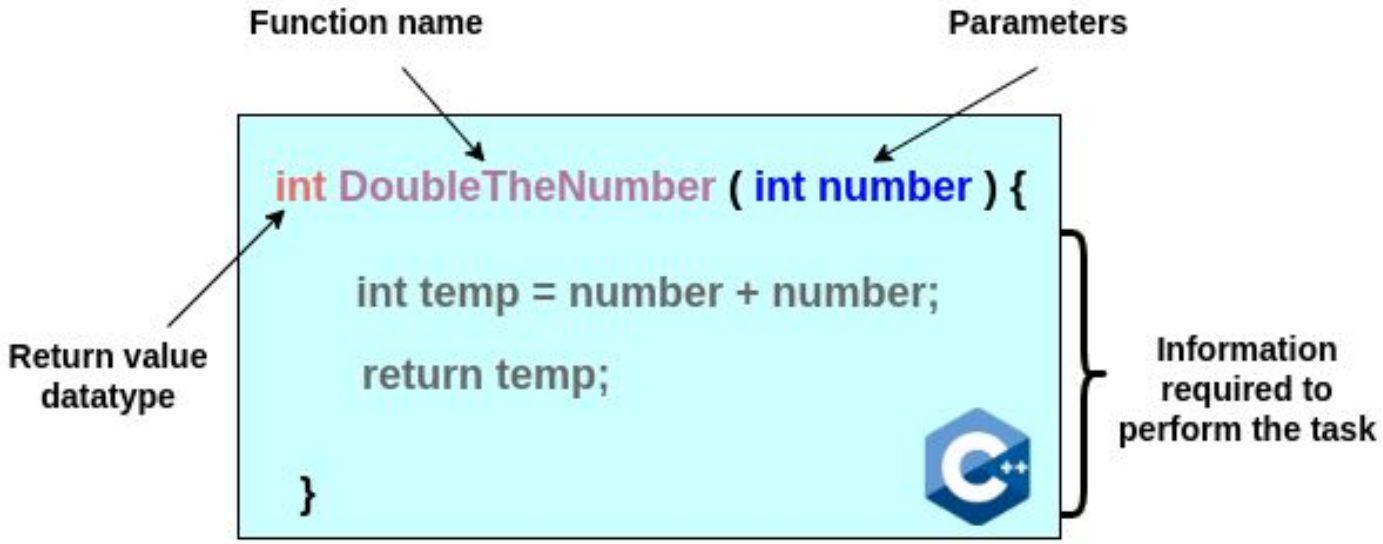
```
Select C:\Users\DELL\Documents\toupper.exe
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5 : 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15

--------------------------------
Process exited after 0.08357 seconds with return value 0
Press any key to continue . . .
```

User-Defined Functions

- **Value-returning functions:** have a return type
  - Return a value of a specific data type using the **return** statement function, called the type of the function
  - You need to add the following items :
    - The name of the function
    - The number of parameters, if any
    - The data type of each parameter
    - The data type of the value computed (that is, the value returned) by the Function
- **Void functions:** do not have a return type
  - Do not use a return statement to return a value

Syntax: Value-Returning function

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



Example:01

In the example below, the function `DoubleTheNumber` takes an integer variable `number` and *returns* an *integer* by doubling its value.

Remember: you can provide more parameters separated b

a comma (`,`).

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

// Function
int DoubleTheNumber ( int number )
{
  int temp = number + number;
  return temp;
}

int main() {
  // calling function
  cout << DoubleTheNumber(10);
  return 0;
}
```

How to pass parameters and arguments to a function in C++

In C++, a **parameter** is a variable that is defined while creating a function. When a function is called, these variables either take or receive arguments.

An **argument** in C++ is the value that is passed to a function whenever that specific function is called.

We declare a function `myfunction` and pass a parameter `myname` to the fu

We create a code block to be executed whenever the function is called.

We call the function, and this time it passes

the argument `"Awais Khan Aarayin"` to the function.

We call the function again, and this time it passes

the argument `"Abdullah"` to the function.

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

void myFunction(string myname)
{
  cout << myname << " Mama's baby!";
}
int main()
{
  // calling the function
  myFunction("Awais Khan Arrayin");
  myFunction("Abdullah");
  return 0;
}
```

Create a default parameter value for a function in C++

Whenever a function is called within a code without passing an argument to it,

the default argument passed to the default parameter when creating the function is returned.

To create a default parameter value, we make use of `=` to assign a default argument to that specific parameter.

- We create a function, `myfunction`, with a parameter `myname`
  a. that has "`John`" as a default argument.
- We write a block to return whenever the `myfunction()` function is called in the
- We call the `myfunction` function without passing an argument to it.
  a. This returns the default argument that was passed to the function earlier.
- We call the function again and this time, we pass the argument `"Sheikh"` to it.
- We call the function a third time and this time, we pass the argument `"Jatt"` to

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

void myfunction(string myname = "John")
{
  cout << myname << " is mama's baby! \n";
}

int main() {
  // calling the function without an argument
  myfunction();

  // calling the function with arguments
  myfunction("Sheikh");
  myfunction("Jatt");
  return 0;
}
```

Function Header

- The function header consists of
  - The function return type
  - The function name
  - The function parameter list
- Example:

```
int main()
```

Function Return Type

- If a function returns a value, the type of the value must be indicated

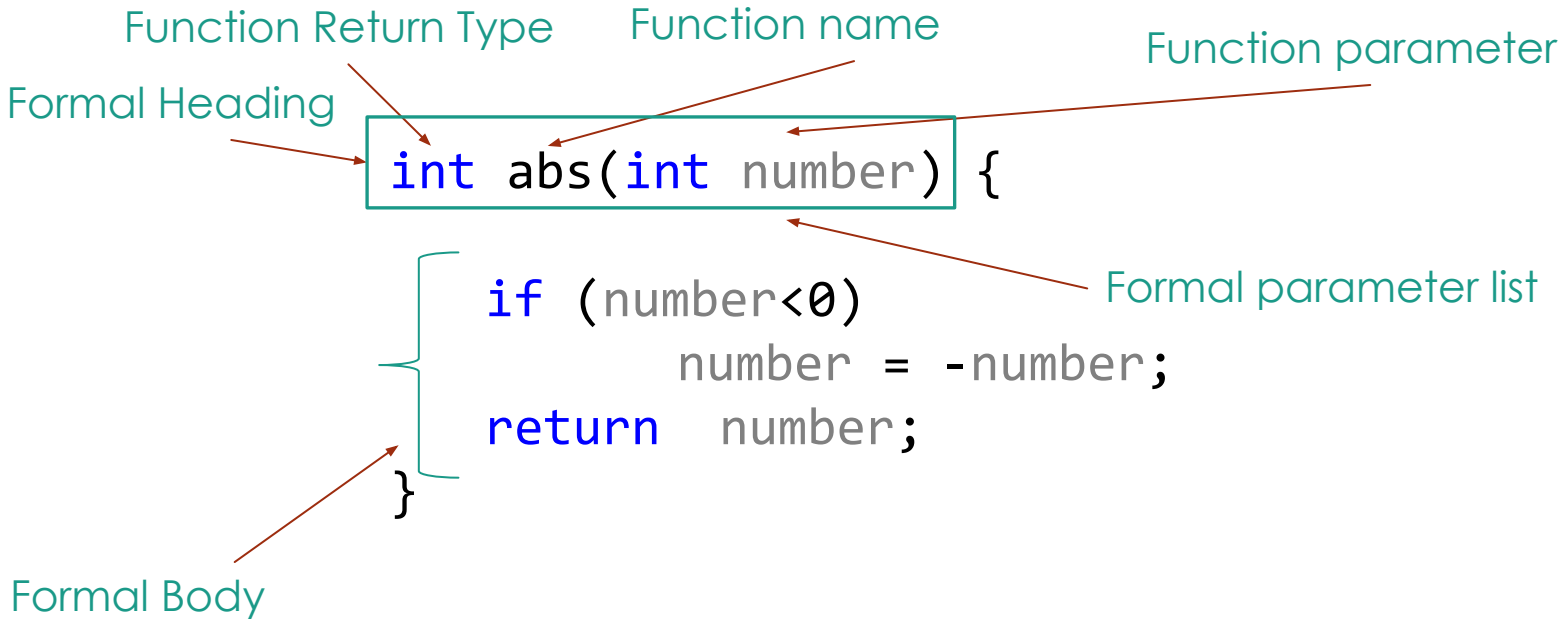
```
int main()
```

- If a function does not return a value, its return type is void

```
void printHeading()
{
    cout << "\tMonthly Sales\n";
}
```

Syntax: Formal Parameter List

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



Defining and Calling Functions

- **Function call:** Statement that causes a function to execute
- **Function definition:** Statements that make up a function

```
int abs(int number);
Similarly the function abs might have the following definition:

int abs(int number)
{
    if(number < 0)
        number = -number;
    return number;
}
```

Function Definition

- **Definition includes**
  - **return type:** Data type of the value the function returns to the part of the program that called it
  - **name:** Name of the function. Function names follow same rules as variable names
  - **parameter list:** Variables that hold the values passed to the function
  - **body:** Statements that perform the function's task

Calling a Function

- To call a function, use the function name followed by () and ;

  printHeading();

- When a function is called, the program executes the **body of the function**
- After the **function terminates**, execution resumes in the calling function at the **point of call**

- **main()** is automatically called when the program starts
- **main()** can call any number of functions
- Functions can call other functions

Function Call

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

Syntax: Actual Parameter List

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

```
functionType functionName()
```

Can be zero parameter

Actual Parameter Vs Formal Parameter

Suppose that the heading of the function **pow** is:

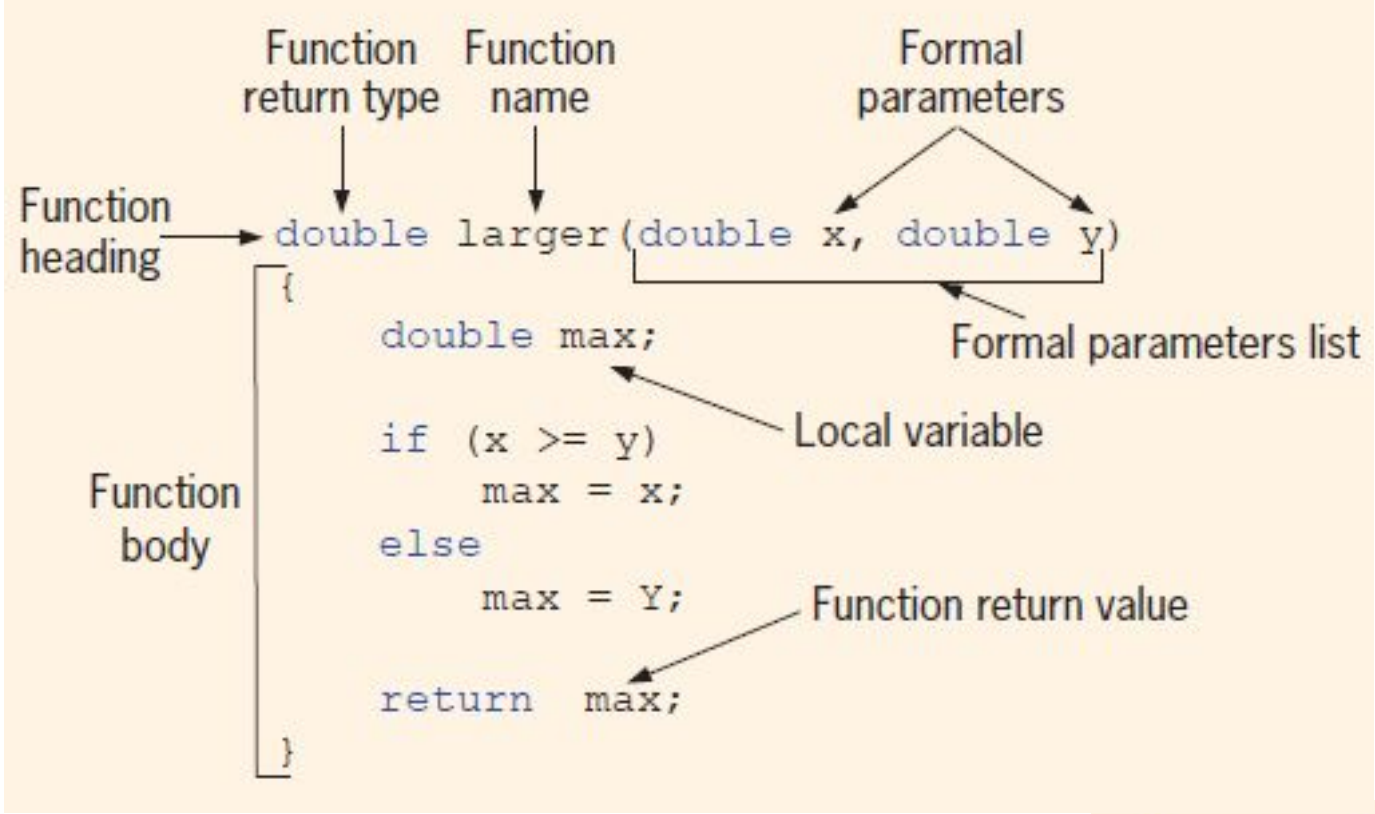
```
double pow(double base, double exponent);
```

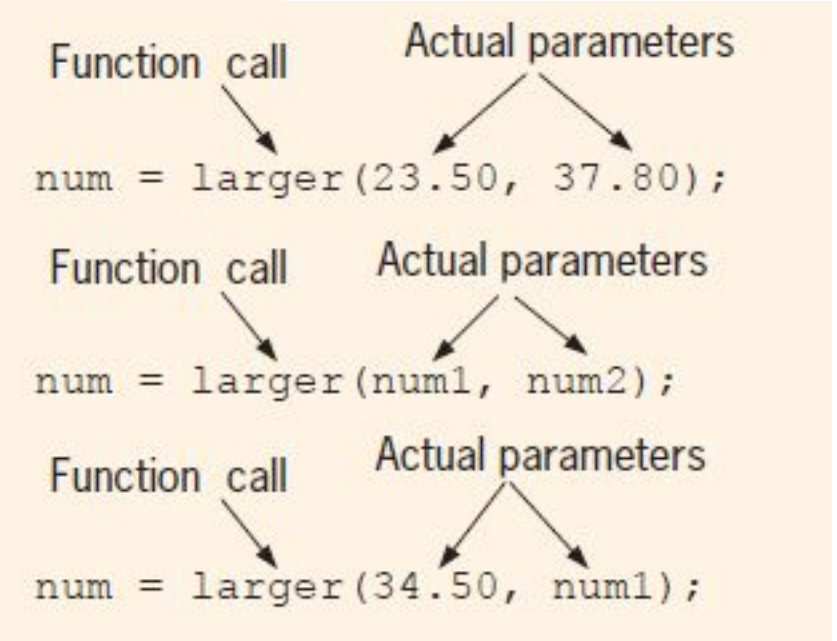
From the heading of the function pow, it follows that the formal parameters of pow are base and exponent. Consider the following statements:

```
double u = 2.5;
double v = 3.0;
double x, y;
x = pow(u, v);                                          //Line1
y = pow(2.0, 3.2) + 5.1;                                //Line2
cout << u << " to the power of 7 = " << pow(u, 7) << endl;          //Line3
```

Formal Parameter: A variable declared in the function heading.

Actual Parameter: A variable or expression listed in a call to function.


Function return type
Function name
Formal parameters
Function heading
double larger(double x, double y)
{
double max;
Formal parameters list
Local variable
Function body
if (x >= y)
max = x;
else
max = Y;
Function return value
return max;
}


Function call
Actual parameters
num = larger(23.50, 37.80);
Function call
Actual parameters
num = larger(num1, num2);
Function call
Actual parameters
num = larger(34.50, num1);

Function Call Notes

- Value of **argument** is **copied** into parameter when the function is called
- Function can have **>= 0** parameter(s)
- There must be a **data type** listed in the **prototype ()** and an argument declaration in the **function heading ()** for each parameter
- Arguments will be **promoted/demoted** as necessary to match parameters

Calling Functions with Multiple Arguments

- When calling a function with multiple arguments:
  - The number of arguments in the call must match the function prototype and definition
  - The first argument will be copied into the first parameter, the second argument into the second parameter, etc.

Calling Functions with Multiple Arguments – an Illustration

```
void displayData(int h, int w)//heading
{
   cout << "Height = " << h << endl;
   cout << "Weight = " << w << endl;
}

displayData(height, weight);     //Call
```

Value-Returning Functions -- Example

```
int abs(int number)
{
   if(number < 0)
      number = -number;

   return number;
}
```

return **Statement**

- Once a value-returning function computes the value, the function returns this value via the return statement
  - It passes this value outside the function via the return statement

Syntax: return **Statement**

- The **return** statement has the following 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

  e.g return 0;

Returning a Value From a Function

- **return** statement can be used to return a value from the function to **the module that made the function call**
- **Prototype and definition must indicate data type** of return value (not void)
- **Calling function should use return value**
  - assign it to a variable
  - send it to cout
  - use it in an arithmetic computation
  - use it in a relational expression
  - Pass it as a parameter to another function

Syntax: return Statement (cont'd.)

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

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}
```

You can also write this function as follows:

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```

NOTE

1. In the definition of the function `larger`, `x` and `y` are formal parameters.
2. The `return` statement can appear anywhere in the function. Recall that once a `return` statement executes, all subsequent statements are skipped. Thus, it's a good idea to return the value as soon as it is computed.

Example

Now that the function `larger` is written, the following C++ code illustrates how to use it.

```
double one = 13.00;
double two = 36.53;
double maxNum;
```

Consider the following statements:

```
cout << "The larger of 5 and 6 is " << larger(5, 6)
     << endl;                                          //Line 1

cout << "The larger of " << one << " and " << two
     << " is " << larger(one, two) << endl;            //Line 2

cout << "The larger of " << one << " and 29 is "
     << larger(one, 29) << endl;                       //Line 3

maxNum = larger(38.45, 56.78);                         //Line 4
```

NOTE

In a function call, you specify only the actual parameter, not its data type. For example, in Example 6-2, the statements in Lines 1, 2, 3, and 4 show how to call the function `larger` with the actual parameters. However, the following statements contain incorrect calls to the function `larger` and would result in syntax errors. (Assume that all variables are properly declared.)

```
x = larger(int one, 29);              //illegal
y = larger(int one, int 29);          //illegal
cout << larger(int one, int two);     //illegal
```

Function to compare three numbers

```
double compareThree(double x, double y, double z)
{
        return larger(x, larger(y, z));
}
```

In the function heading, `x`, `y`, and `z` are formal parameters.

Let us take a look at the expression:

```
larger(x, larger(y, z))
```

Returning a Boolean Value

- Function can return **true** or **false**
- Declare return type in function prototype and heading as **bool**
- Function body must contain **return** statement(s) that return **true** or **false**
- Calling function can use return value in a **relational expression**

Boolean return Example

```
bool isValid(int);              // prototype

bool isValid(int val)           // heading
{
    int min = 0, max = 100;
    if (val >= min && val <= max)
        return true;
    else
        return false;
}

if (isValid(score))         // call
   …
```

Function Prototypes

- The compiler must know the following about a function before it is called
  - name
  - return type
  - number of parameters
  - data type of each parameter

Function Prototype

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

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

- It is **not** necessary to specify the **variable name in** the parameter list
- The **data type** of each **parameter must** be specified

Function Prototype

- The function heading without the body of the function.

Syntax: Function Prototype

The general syntax of the function prototype of a value-returning function is:

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

(Note that the function prototype ends with a semicolon.)

For the function `larger`, the prototype is:

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

NOTE

When writing the function prototype, you do not have to specify the variable name in the parameter list. However, you must specify the data type of each parameter.

You can rewrite the function prototype of the function `larger` as follows:

```
double larger(double, double);
```

Prototype Notes

- Place prototypes near **top** of program
- **Program must include either prototype or full function definition before any call to the function**, otherwise a compiler error occurs
- When using prototypes, **function definitions can be placed in any order in the source file**. Traditionally, **main** is placed first.
- Use a **function prototype** (similar to the heading of the function
  - **Header:** void printHeading()
  - **Prototype:** void printHeading();

Function Prototype (Illustration)

```
//Program: Largest of three numbers

#include <iostream>

using namespace std;

double larger(double x, double y);
double compareThree(double x, double y, double z);

int main()
{
    double one, two;                                        //Line 1

    cout << "Line 2: The larger of 5 and 10 is "
         << larger(5, 10) << endl;                          //Line 2

    cout << "Line 3: Enter two numbers: ";                  //Line 3
    cin >> one >> two;                                      //Line 4
    cout << endl;                                           //Line 5

    cout << "Line 6: The larger of " << one
         << " and " << two << " is "
         << larger(one, two) << endl;                       //Line 6

    cout << "Line 7: The largest of 23, 34, and "
         << "12 is " << compareThree(23, 34, 12)
         << endl;                                           //Line 7

    return 0;
}
```

Function Prototype (Illustration cont'd.)

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

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}

double compareThree (double x, double y, double z)
{
    return larger(x, larger(y, z));
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 2: The larger of 5 and 10 is 10
Line 3: Enter two numbers: 25.6 73.85

Line 6: The larger of 25.6 and 73.85 is 73.85
Line 7: The largest of 43.48, 34.00, and 12.65 is 43.48
```

Example Program I

```
1   // C++ Program
2   // Creating and using a programmer-defined function.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   int square(int);   // function prototype
9
10  int main()
11  {
12     // loop 10 times and calculate and output
13     // square of x each time
14     for (int x = 1; x <= 10; x++)
15        cout << square(x) << "  ";  // function call
16
17     cout << endl;
18
19     return 0;  // indicates successful termination
20
21  } // end main
22
23  // square function definition returns square of an integer
24  int square(int y)  // y is a copy of argument to function
25  {
26     return y * y;     // returns square of y as an int
27
28  } // end function square
```

Function prototype: specifies data types of arguments and return values. **`square`** expects and **`int`**, and returns an **`int`**.

Parentheses **`()`** cause function to be called. When done, it returns the result.

Definition of **`square`**. **`y`** is a copy of the argument passed. Returns **`y * y`**, or **`y`** squared.

Output

1 4 9 16 25 36 49 64 81 100

Example Program II

```
1   // C ++ Program
2   // Finding the maximum of three floating-point numbers.
3   #include <iostream>
4
5   using std::cout;
6   using std::cin;
7   using std::endl;
8
9   double maximum(double, double, double); // function prototype
10
11  int main()
12  {
13     double number1;
14     double number2;
15     double number3;
16
17     cout << "Enter three floating-point numbers: ";
18     cin >> number1 >> number2 >> number3;
19
20     // number1, number2 and number3 are arguments to
21     // the maximum function call
22     cout << "Maximum is: "
23          << maximum(number1, number2, number3) << endl;
24
25     return 0;  // indicates successful termination
26
27  } // end main
28
```

Function **maximum** takes 3 arguments (all **double**) and returns a **double**.

Example Program II

```
29    // function maximum definition;
30    // x, y and z are parameters
31    double maximum(double x, double y, double z)
32    {
33       double max = x;    // assume x is largest
34
35       if (y > max)     // if y is larger,
36          max = y;        // assign y to max
37
38       if (z > max)     // if z is larger,
39          max = z;        // assign z to max
40
41       return max;        // max is largest value
42
43    } // end function maximum
```

Comma separated list for multiple parameters.

Sample run

Enter three floating-point numbers: 99.32 37.3 27.1928

Maximum is: 99.32

Enter three floating-point numbers: 1.1 3.333 2.22

Maximum is: 3.333

Enter three floating-point numbers: 27.9 14.31 88.99

Maximum is: 88.99

Value-Returning Functions: Some Peculiarity

```
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 Peculiarity (cont'd.)

```
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
}
```

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

EXAMPLE 6-3

In this example, we write the definition of function `courseGrade`. This function takes as a parameter an `int` value specifying the score for a course and returns the grade, a value of type `char`, for the course. (We assume that the test score is a value between `0` and `100` inclusive.)

```
char courseGrade(int score)
{
    switch (score / 10)
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        return 'F';
    case 6:
        return 'D';
    case 7:
        return 'C';
    case 8:
        return 'B';
    case 9:
    case 10:
        return 'A';
    }
}
```

Questions