Programming Principles (MT162)

Lecture 3

Dr. Ahmed Fathalla

Introduction

- Functions are like building blocks
- They allow complicated programs to be divided into manageable pieces
- Some advantages of functions:
  - **Different people can work on different functions simultaneously.**
  - **A programmer can focus on just that part of the program and construct it, debug it, and perfect it.**
  - **Can be re-used (even in different programs).**
  - **Enhance program readability.**

There are two types of function

- **Standard Library Functions:** Predefined in C++.
- **User-defined Function:** Created by users.

Predefined Functions

- Some of the predefined mathematical functions are:

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

- Predefined functions are organized into separate libraries.
- I/O functions are in `iostream` header.
- Math functions are in `math.h` or `cmath` header.

Predefined Functions

- **`Power Function - pow(x,y):`**
  - `pow(x,y)` calculates x^y
    - `pow(2, 3) = 8.0`
    - Returns a value of type `double`
    - `x` and `y` are the **parameters** (or **arguments**)
      - The function has two parameters

Predefined Functions

- **`Square Root Function - sqrt(x):`**
  - Square root function sqrt(x) has only one parameter
  - sqrt(x) returns value of type double
  - sqrt(x) calculates non-negative square root of x, for x >= 0.0: sqrt(2.25) = 1.5
- **`Floor Function - floor(x):`**
  - Floor function floor(x) has only one parameter
  - floor(x) returns value of type double
  - floor(x) calculates largest whole number not greater than x: floor(48.79) = 48.0

Predefined Functions (continued)

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`

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

using namespace std;

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

User-Defined Functions

- Two types:
  1. Void functions (non-value-returning): no return type, do not return a value
  2. Value-returning functions: have a data type, return only one value to caller
     - Return a value of a specific data type using the `return` statement

Function Declaration

- Syntax

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

- To define a function you must define 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 **function-type**

Void Functions

- Function definition syntax:

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

- Formal parameter list syntax:

```
dataType  variable, dataType  variable, ...
```

Must write the datatype of each single parameter

- Function call syntax:

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

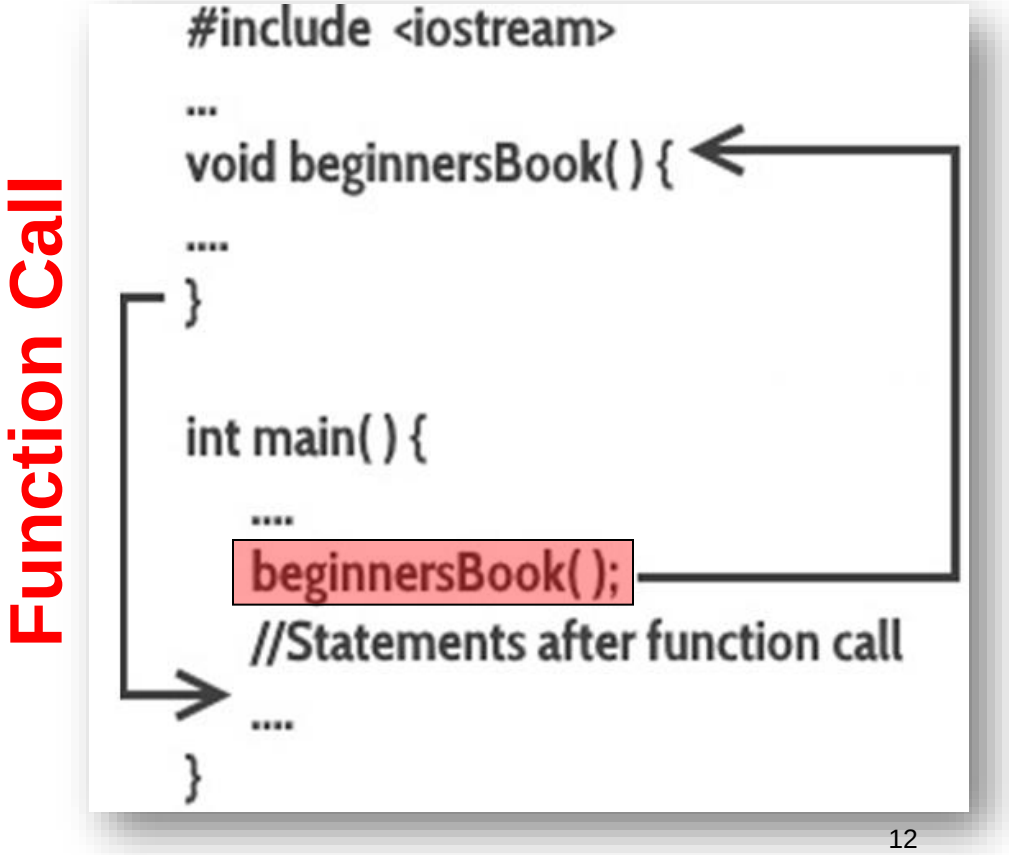
- Actual parameter list syntax:

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

Exercise

Write a function to print “Hello World” for N times, where N is a function parameter.

```
#include <iostream>
using namespace std;
void print_hello_world(int N)
   {
      for(int i=1;i<=N;i++)
         cout<<"Hello World\n";
      return;  // optional line
   }
int main()
   {
      int n=10;
      print_hello_world( n );   // Function Call
      return 0;
   }
```



Syntax: Value-Returning Function

- Syntax:

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

      return something;
}
```

- `Function-Type` is also called the **data type** or **return type**

Value-Returning Functions (continued)

- Formal Parameter: variable declared in the function definition

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

- Actual Parameter: variable or *expression* listed in the function call
  - Example: `x = pow(u, v)`

`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

Example: “get Max of two numbers”

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

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

    return max;
}
```

```
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.

Exercise

- Write a function that accept two input parameters and return the XOR value of the two numbers.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

XOR

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.

Exercise: get the max value of three numbers using the **“larger”** function

```
//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;
}
```

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

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 73

Line 6: The larger of 25 and 73 is 73
Line 7: The largest of 23, 34, and 12 is 34
```

Summary

- Functions (modules) are miniature programs
  - Divide a program into manageable tasks
- C++ provides the standard 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 (continued)

- In a function call, the number of actual parameters and their types must match with the formal parameters in the order given
- To call a function, use its name together with the actual parameter list
- Function heading and the body of the function are called the definition of the function
- If a function has no parameters, you need empty parentheses in heading and call
- A value-returning function returns its value via the `return` statement

Summary (continued)

- A prototype is the function heading without the body of the function; prototypes end with the semicolon
- Prototypes are placed before every function definition, including `main`
- User-defined functions execute only when they are called
- In a call statement, specify only the actual parameters, not their data types