BTE2313

Chapter 7: Function

by
Sulastri Abdul Manap
Faculty of Engineering Technology
sulastri@ump.edu.my

Objectives

- In this chapter, you will learn about:
  1. Create and apply user defined function
  2. Differentiate between standard library functions and user defined functions
  3. Able to use both types of functions

Introduction

- A ***function*** is a complete section (block) of C++ code with a definite start point and an end point and its own set of variables.
- Functions can be passed data values and they can return data.
- Functions are **called** from other functions, like `main()` to perform the task.
- Two types of function:

1. User defined function: The programmer writes their own function to use in the program.

2. Standard library function: Function already exist in the C++ standard libraries http://www.cplusplus.com/reference/

Introduction (cont.)

- Advantages of functions:

→Easier to solve complex task by dividing it into several smaller parts (structured programming)

→Functions separate the concept (*what* is done) from the implementation (*how* it is done)

→Functions can be called several times in the same program, allowing the code to be reused.

Three Important Questions

- *"What is the function supposed to do? (Who's job is it?"*)
  - If a function is to read all data from a file, that function should open the file, read the data, and close the file.

- *"What input values does the function need to do its job?"*
  - If your function is supposed to calculate the volume of a pond, you'd need to give it the pond dimensions.

- *"What will my function return to me?"*
  - What will the function give you when it has finished ·· task?

Functions: Basic Format

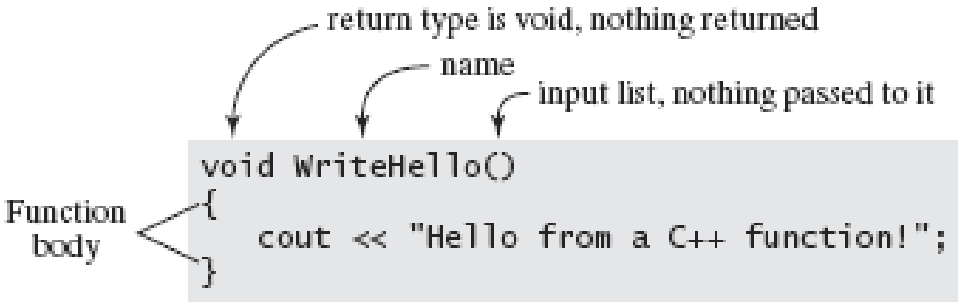
```
return_type  function_name(input parameter list)
{
   // Body of function
}
```

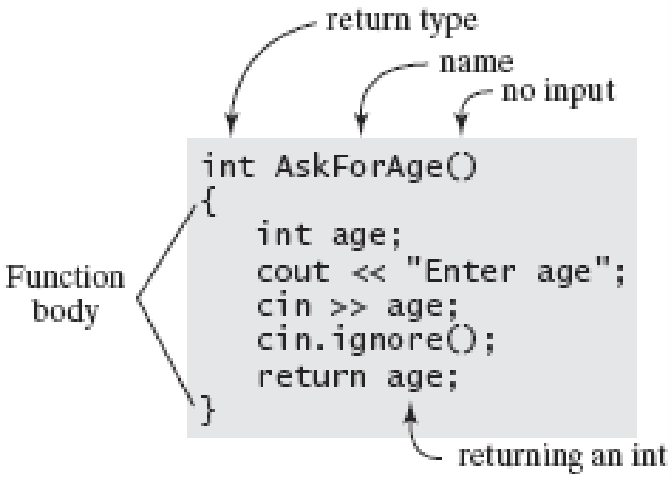
- The ***return_type*** is the data type of the value returned from the function.

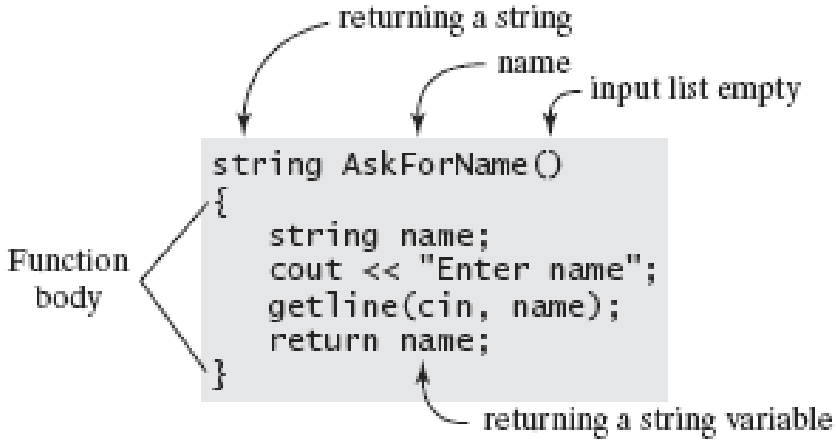
- The ***function_name*** is the identifier (name) of the function and is used to access the function.

- The ***input parameter list*** is the list of the input variable data types and names that the function receives.

Functions: Basic Format (cont.)







Three examples of simple functions.

Functions: Basic Format (cont.)

- Rules for declaring functions.
  - The function name must follow standard C++ naming conventions.
  - There may be one return type.
  - If there is no return type, the **`void`** data type is used.
  - The input argument list must have data type and names (separated by commas)
  - You may pass in as many arguments as you like.
  - If your list is empty (no arguments are passed), the parentheses will be empty () but still required

Basic Function: Example 1

- **`WriteHello()`** is passed with nothing, and returns nothing.

```
void WriteHello()
{
 cout << "\n Hello from a function!";
}
```

Basic Function: Example 2

- The **`AskForAge()`** function does not have input arguments, but it returns an integer value.

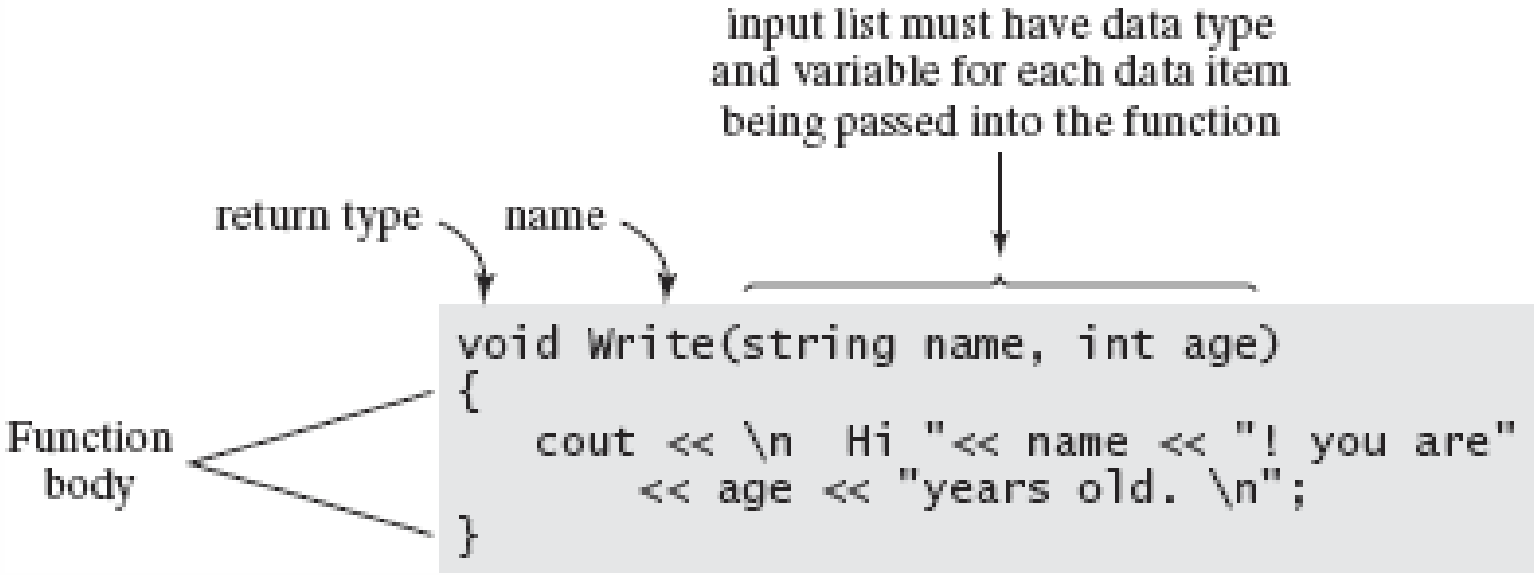
```
int AskForAge()
{
  int age;
  cout << "\n How old are you?  ";
  cin >> age;
  return age;
}
```

Basic Function: Example 3

- The **`AskForName()`** function does not have any input arguments. It only asks the user to enter the name.

```
string AskForName()
{
    string name;
    cout << "\n What is your name?  ";
    getline(cin,name);
    return name;
}
```

Basic Function: Example 4



- The **`Write`** function's job is to write the user's name and age to the screen.
- It doesn't "return" anything, but the input parameter list has two data items.

```
void Write(string name, int age)
{
   cout << "\n Hi "<< name << "! You are "
   << age << " years old.  \n";
}
```

Calling and Called Functions

- The terms ***calling function*** and ***called function*** are often used when referring to functions.
- When one function (Function1) accesses another function (Function2), it is said that Function1 calls Function2.
- That is, Function1 is the calling function and Function2 is the called function.
- In order to invoke (make use of) the function, you'll need to have a statement that calls it.
- The ***call statement*** is the code located in the program where the function is to be used or accessed.
- Depending on the requirements for the function (input list) and return type—your function calls may vary in appearance

Function Call: Example

```
//Write a greeting
WriteHello();        //call, no inputs, no return value

//Ask for the user's name
name = AskForName();    //call, returns name

//Ask for age
age = AskForAge();      //call, value assign into age

//Write information
Write(name, age);       //call, pass name, age to Write
```

Requirements for Writing Functions

- Every function must have a
  - ***function prototype/declaration***
  - ***function definition/body***.
  - The function prototype statement tells the compiler the function name, and its return and input types.
  - The function definition contains the actual code that performs the function's task.

- The first line of the function definition is the ***function header***. That line contains the return type, function name, and input parameter list.

Concentrate on the Function First

- Concentrate on one function at a time.
- Work as if you can’t see other parts of the program.
- Each function is its own block of code, with its own variables and control statements.
- Write the prototype, see if it can be called, then write the function.

C++ HOTEL Example

source code files

```
# includes go here

// function declarations or prototypes go here

int main()
{
  .
  .
  return 0;
}

int HowManyPeople()
{
  int num;
  :    :
  return num;
}

int HowManyNights()
{
  int num;
  :    :
  return num;
}

float CalcCost(int numPeople, int numNights, string roomtype)
{
  float cost;
  :    :
  return cost;
}

void WriteCostSummary(int numPeople, int numNights,
                      string roomtype, float cost)
{
  .
  .

}
```

All call statements in main()

main() function

HowManyPeople function

HowManyNights function

CalcCost function

WriteCostSummary function

See how functions must be laid out in the source code file.

Function Declaration/Prototype

- Before a function can be called, the calling function must know about the called function.
- The prototype may be declared in the calling function before the call. It can appear above the calling function or it can appear in an `include` file.
- The important point is that the compiler must have seen the function prototype before the function is called.

Function Declaration/Prototype (cont.)

- The form of the function prototype is:

```
return_type function_name(input param type
  list);
```

- These statements must be seen before they are called.
- Example: 5 prototypes in the C++ Hotel program.

```
int HowManyPeople();
int HowManyNights();
string WhatTypeRoom();
float CalcCost(int people,int nights,
        string roomType);
void WriteCostSummary(int people, int nights,
        string roomType, float cost);
```

`return` Statement

- The ***return statement*** serves three purposes in a C++ function.
  - It is required when the function is returning a value to the calling function.
  - The return statement causes the program control to exit the function and to return to the calling function.
  - An expression may be evaluated within the parentheses format of the return statement.
- The return statement is way to terminate the function and return to the calling function at any point in the function body.
- The return statement in **`CalcCost()`** can be written as:

```
totalCost = (rate + addPersonCost)* nights
return totalCost;
```

- OR it could be written like this:

```
return ((rate + addPersonCost)* nights);
```

`return` Statement (cont.)

- The function **`WriteGreeting()`** does not return a value to the calling function, so no return statement is required:

```
void WriteGreeting()
{
 cout << "\n Welcome to the C++”
    << “ Hotel Rate Program"
    << "\n we offer a fine hotel on”          << “
the beach. \n";
}
```

Function Calls

- The ***call statement*** is the C++ statement where the called function is accessed.
- When a function is called, control is passed to the called function, and the statements inside that function are performed.
- Control is returned to the calling function when the function tasks are completed.
- The function calls in the C++ Hotel program are:

```
numPeople = HowManyPeople();
numNights = HowManyNights();
roomType = WhatTypeRoom();
totalCost = CalcCost(numPeople, numNights,
   roomType);
WriteCostSummary(numPeople, numNights,
   roomType, totalCost);
```

Types of Function Calls

No Inputs and No Return Value

- When a function does not have any inputs, nor does it return anything to the calling function, the call statement is very simple.

- The **`WriteGreeting()`** function is like this, and must be called like so:

```
WriteGreeting();
```

Types of Function Calls (cont.)

No Inputs but has a Return Value

- When a function does not have any inputs, but it does return a value, you have to be sure the call statement has an assign operator so that the returned value is placed in a variable.

- Note how the input list parentheses are empty, but the return values are each assigned to one of main's variables.

```
numPeople = HowManyPeople();
numNights = HowManyNights();
roomType = WhatTypeRoom();
```

Types of Function Calls (cont.)

Input and No Return Values

- When a calling function must pass information to the called function, but the function doesn't return anything, the call statement does not have an assign operator.

- When this program writes the resultant cost summary to the screen, this task is done by the **`WriteCostSummary()`** function. The function doesn't return any value to us.

```
WriteCostSummary( numPeople, numNights,
                  roomType,totalCost);
```

Types of Function Calls (cont.)

Input and Return Values

- When a calling function must pass information to the called function, and that function returns a value to us, we need be have an assign statement to obtain that value.

- Here is the call to **`CalcCost()`**, which returns to us the total cost of the stay.

```
totalCost = CalcCost(numPeople, numNights,
        roomType);
```

Passing Values to Function

- Pass by Value:

→ Only pass the value (not the variable) to the parameter in the function.

→ The variable's value in `main()` does not change.

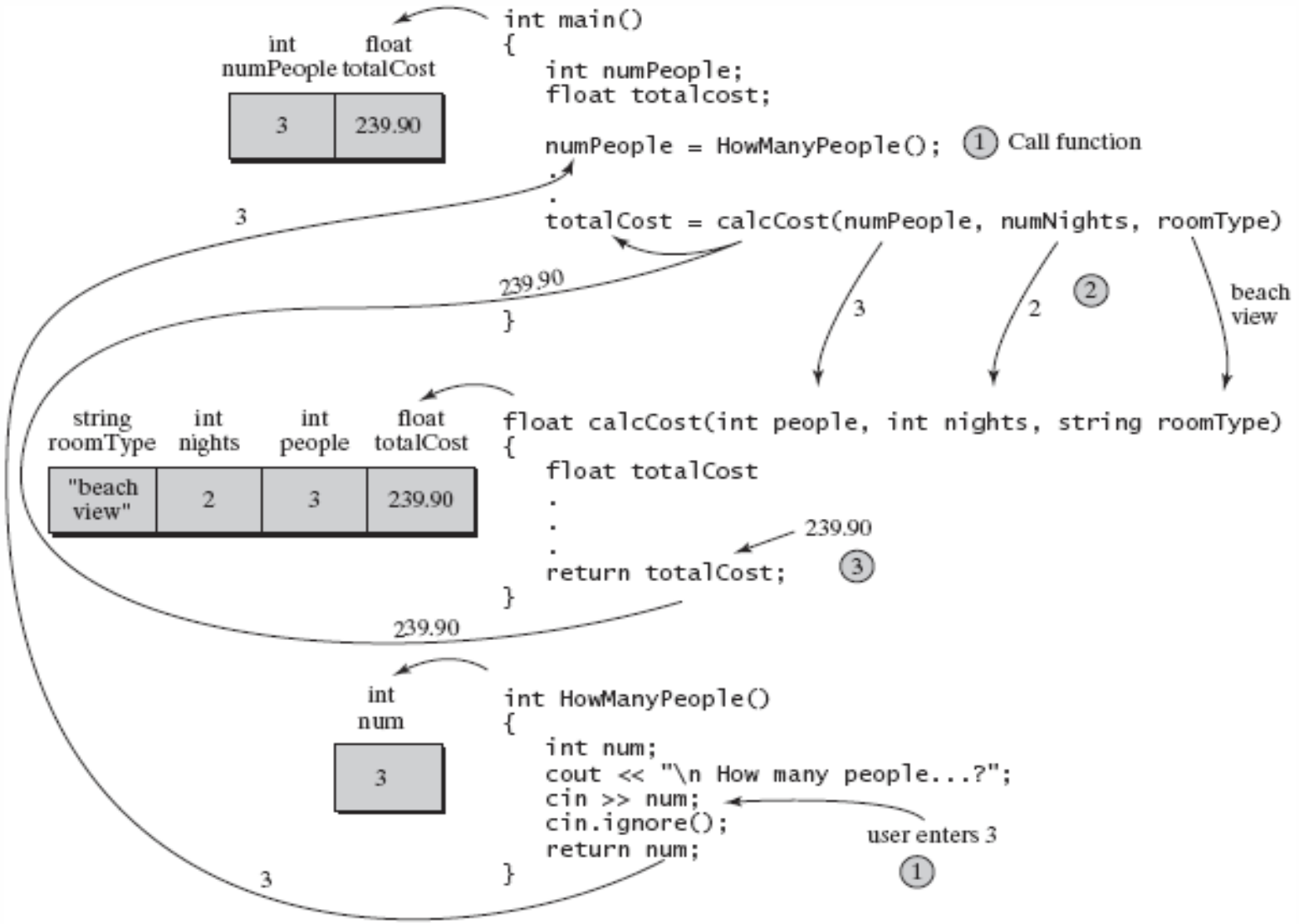
→ The value is copied into the parameter of the function

- Pass by reference:

→ Ask the parameter in the function to refer to the variable in `main()`.

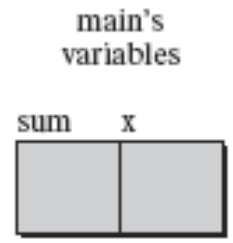
→ Written with ampersand (&) in the function's parameter.

Pass by Value

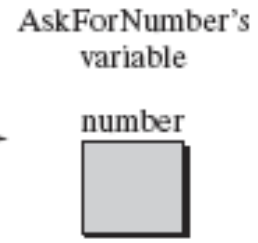


Each function has its own copy of the data variables.

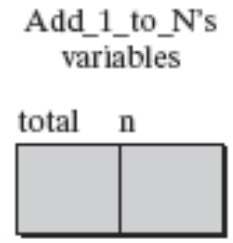
```
int main()
{
   int x, sum;
   x = AskforNumber();
   sum = Add_1_to_N(x);
   return 0;
}
```




```
int AskforNumber()
{
   int number;
   cout << "\n Enter a number.";
   cin >> number;
   return number;
}
```




```
int Add_1_to_N(int n)
{
   int total;
   //
   return total;
}
```



Functions have their own copies of the variables.
These variables are referred to as local variables.

Standard Library Function (cont.)

- Exist in the standard libraries in C++
- Need to include the standard library (where function is located)
  `#include <libraryname>`
- Example: To use function named `pow` to calculate 2^3 (function `pow` is located in the standard library `math.h`)

```
#include<iostream>
#include<math>
using namespace std;
int main ()
{
    double result;
    result = pow(2,3); //2^3
    cout << "2 to the power of 3 is" << result;
    return 0;
}
```

Standard Library Function (cont.)

- Example of Standard Library: `math.h` contains functions to compute common mathematical operations and transformations
- http://www.cplusplus.com/reference/cmath/

Function	Parameters	Example	Answer	Decription
pow	2 double	pow (2,3);	8	2 to the power of 3 equals to 8
sqrt	1 double	sqrt (25);	5	Square root of 25 is 5
cos	1 double	cos (0);	1	Cosine for 0 is 1
sin	1 double	sin (0);	0	Sine for 0 is 0
tan	1 double	tan (0)	0	Tangent for 0 is 0

Common Errors With Functions

Compiler Error: function does not take __ parameters.

```
prototype:      void Write();

call:           int main()
                {
                   .
                   .
                   .
                   Write(name,age);
                   .
                   .
                   .
                }

function        void Write(string name, int age)
                {
                   cout<<"\n"<<name<<"is"<<age"years old.";
                }
```

Compiler sees the prototype with no inputs

Call statement has two inputs
Hence you can get this error.

HOW TO FIX : Make sure prototype, call, and function header match in the input lists!

The function prototype and call statements do not match in the input lists.

Link Error: unresolved external: void _decl Write(class string, int age)

```
void Write(string name, int age);

int main()
{
   .
   .
   .
   Write(name,age);
   .
   .
   .
}

// function not written yet
   OR if function didn't match prototype

void Write(string name)
{
   cout<<"\n"<<name;
}
```

Here is the prototype

Call statement - parameters match prototype compiles without error but

Linker is looking for the Write function that matches the prototype

FIX : If you have a prototype and call to a function, make sure the function is written too!

The prototype, call statement and function header lines have to match.

Compile Error: missing function header (old-style formal list?)

```
void Write(string name, int age);
```

Prototype correctly written with ending;

Function Definition

```
void Write(string name, int age);
{
  cout<<"\n"<<name<<"you are"<<age<<"years old.";
}
```

Problem here no ; on function header line

FIX : Remove ; from function header line.

Leaving the semi-colon on the function header line generates this error.

Summary

- A function has a name, a list of parameters (which may be empty), and a result type (which may be `void`).
- Callers communicate information to a function via its parameters (also known as arguments)
- Callers must pass the correct number and types of parameters that the function expects.
- If a function is declared to return a value, it must execute a return statement with a value of the declared return type.
- C++ standard library provides a collection of routines that can be incorporated into the codes that you write.
- The function is a standard unit of reuse in C++
- When faced with the choice of using a standard library function or user defined function to solve the same problem, choose the library function. The standard function will be tested thoroughly, well documented, and likely more efficient.