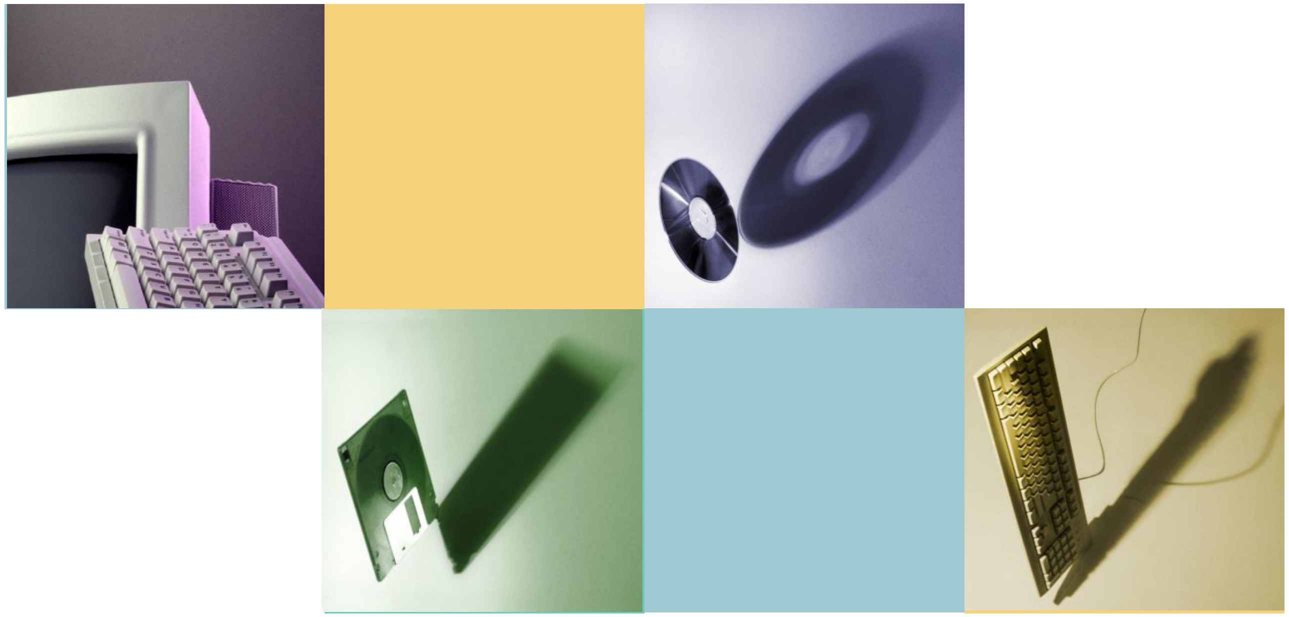
Object-Oriented Programming

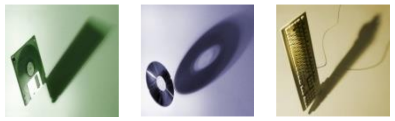


Chuan-Kang Ting

Dept of Computer Science and Information Engineering
National Chung Cheng University

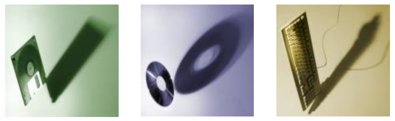
Chapter 3

Function Basics



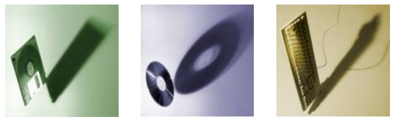
Outlines

- Functions
- Predefined Functions
- Programmer-Defined Functions
- Scope



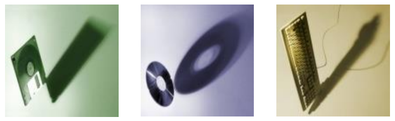
Functions

- **Building Blocks of Programs**
  - A program consists of functions
- **What is a function in C++?**
  - the same thing as procedure, subprogram, and method in other languages
- **I-P-O**
  - Input – Process – Output
  - Basic subparts to any program
  - Use **functions** for these "pieces”



Functions (cont’d)

- **Types of functions**
  - functions that return (produce) a value
  - functions that do NOT return a value → void function
    - All aspects are same as functions that "return a value"
    - They just don’t return a value!



Terminology

Display 3.5 A Function Using a Random Number Generator

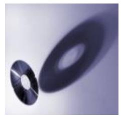
Alternate declaration:
double totalCost(int, double);

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

double totalCost(int numberParameter, double priceParameter);
//Computes the total cost, including 5% sales tax,
//on numberParameter items at a cost of priceParameter each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = totalCost(number, price);
```

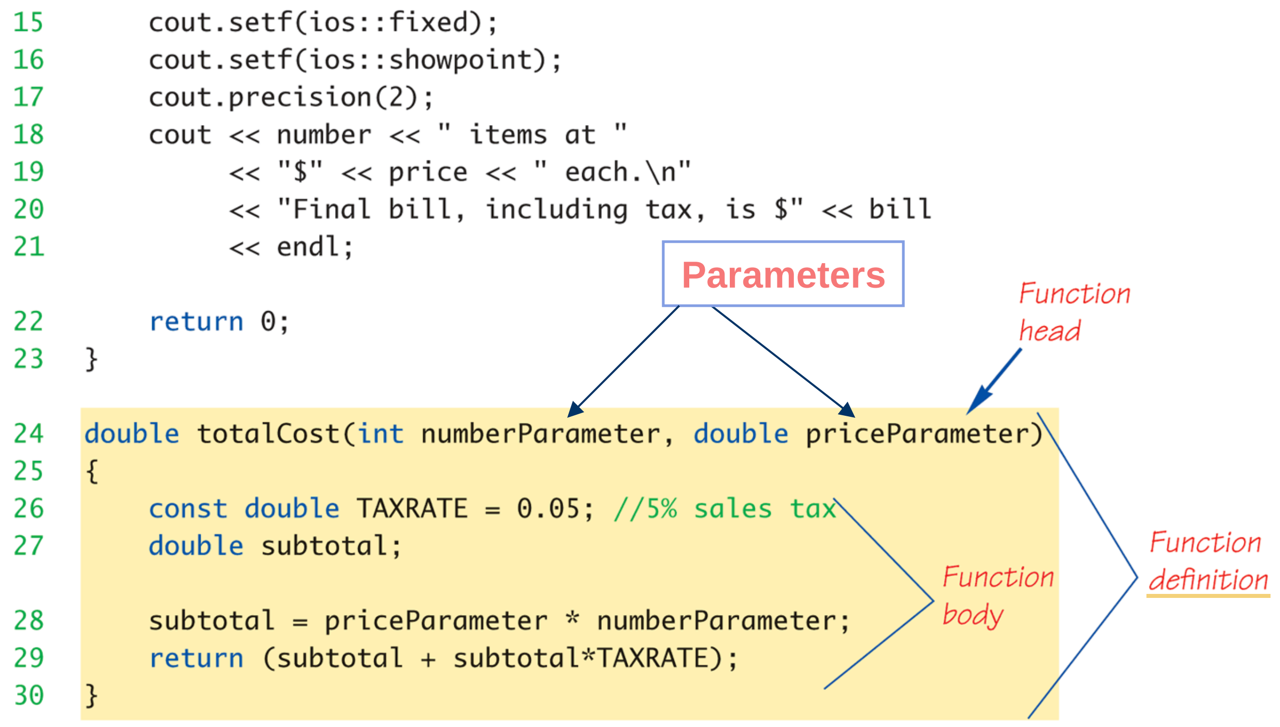
Function declaration; also called the function prototype

Function call (or function invocation)

Arguments



Terminology (cont'd)



```
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
         << "Final bill, including tax, is $" << bill
         << endl;

    return 0;
}

double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal*TAXRATE);
}
```

Sample Dialogue

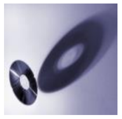
```
Enter the number of items purchased: 2
Enter the price per item: $10.10
2 items at  $10.10 each.
Final bill, including tax, is $21.21
```

Parameters

- **(Formal) Parameters**
  - A formal parameter is used as a **placeholder** to stand in for the argument
    - When you write a function declaration, you don't know what the arguments will be, so you use the formal parameters in place of the arguments
    - When the function is called, the argument values are *plugged in* for the parameters

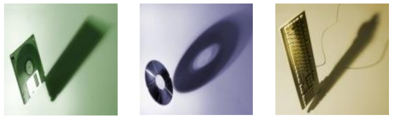
• **Parameters** →	Function declaration Function definition
• **Argument** →	Function call

Placeholder:
a symbol in a logical or math expression that can be replaced by the name of any element of a set – Webster



Using Functions (0)

- **3 Pieces**
  - Function Declaration
    - Information for compiler
    - To properly interpret calls
  - Function Definition
    - Actual implementation/code for what function does and actions
  - Function Call
    - Transfer control to function

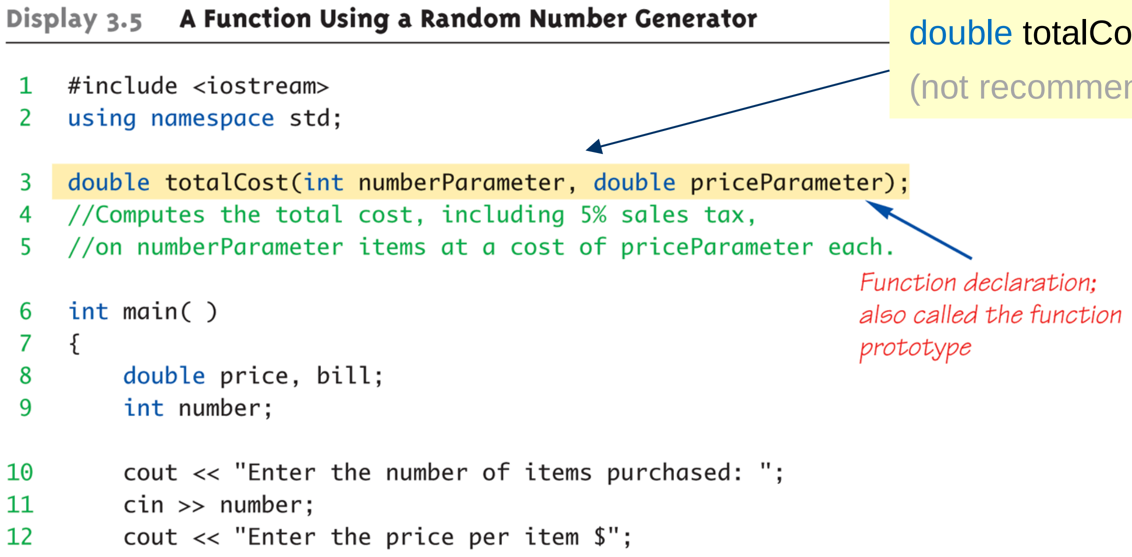


Using Functions (1)

- **Function Declaration**
  - Tells compiler how to interpret calls
  - Syntax:
    - <return_type> FnName(<formal-parameter-list>);
  - e.g.

Alternate declaration:
double totalCost(int, double);
(not recommend!)

Place:
before any calls

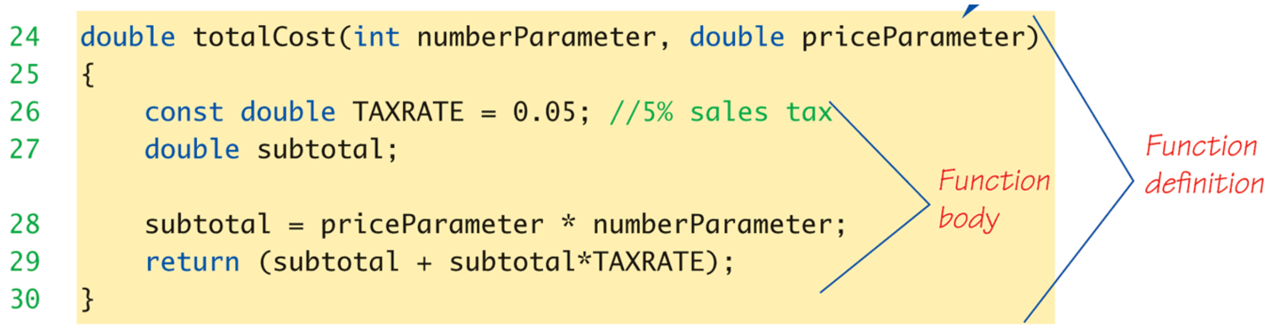


Display 3.5 A Function Using a Random Number Generator

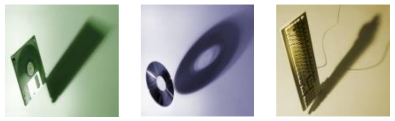
```
#include <iostream>
using namespace std;

double totalCost(int numberParameter, double priceParameter);
//Computes the total cost, including 5% sales tax,
//on numberParameter items at a cost of priceParameter each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
```

Using Functions (2)

- **Function Definition**
  - Implementation of function
  - Consists of head and body
  - Formal parameters
    - "Placeholders" for data sent in
  - `return` statement
    - sends data and transfers control back to caller
  - e.g.



```
double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal*TAXRATE);
}
```



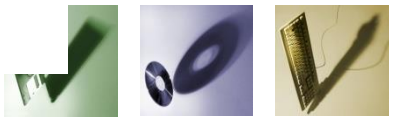
Using Functions (3)

- **Function Call**
  - Transfer control to function
  - Return / not return (void function) a value with the defined data type
  - Arguments
    - The data send in formal parameters of a function
  - e.g.

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

double totalCost(int numberParameter, double priceParameter);
//Computes the total cost, including 5% sales tax,
//on numberParameter items at a cost of priceParameter each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = totalCost(number, price);
```

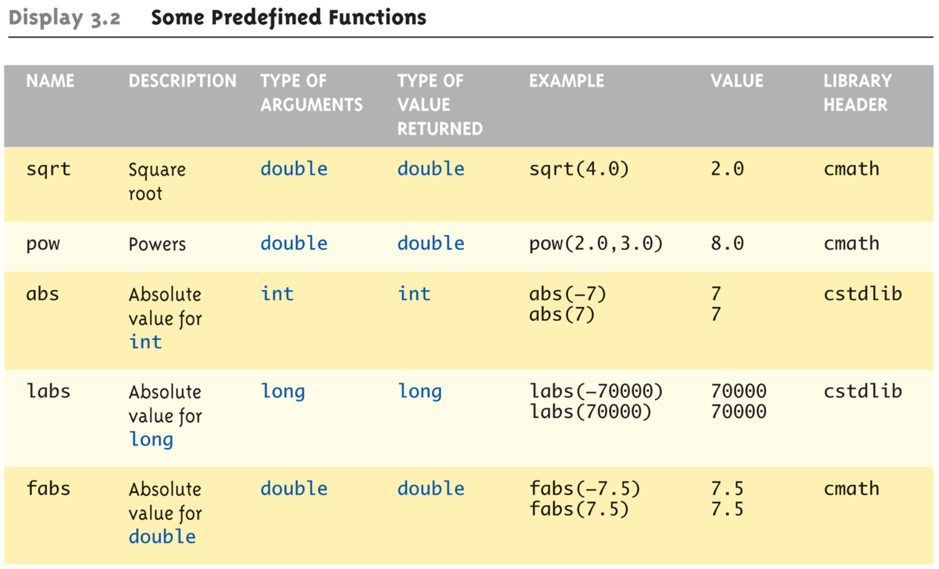
Function declaration; also called the function prototype

Function call



Predefined Functions

- **"Predefined" functions**
  - C++ comes with libraries of predefined functions
    - One can hardly write a program without these predefined functions
    - Do not reinvent the wheel
  - `#include` directive and `std` namespace



Display 3.2 **Some Predefined Functions**

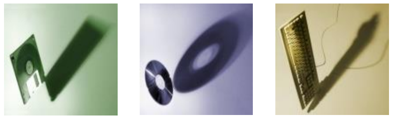
NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0,3.0)	8.0	cmath

abs	Absolute value for int	int	int	abs(-7)
				abs(7)
					7
					7
						cstdlib

labs	Absolute value for long	long	long	labs(-70000)
				labs(70000)
					70000
					70000
						cstdlib

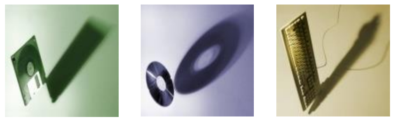
fabs	Absolute value for double	double	double	fabs(-7.5)
				fabs(7.5)
					7.5
					7.5
						cmath

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



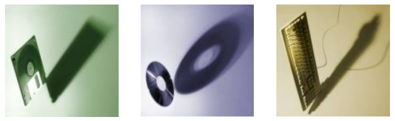
`main( )`

- `main()` is a function
  - One and only one function called main() will exist in a program
  - Automatically called when program is executed
- Who calls `main()` ?
  - Operating system
  - Tradition holds it should have return statement
    - Value returned to "caller" → Here: operating system
    - Should return "int" or "void"



Scope (1)

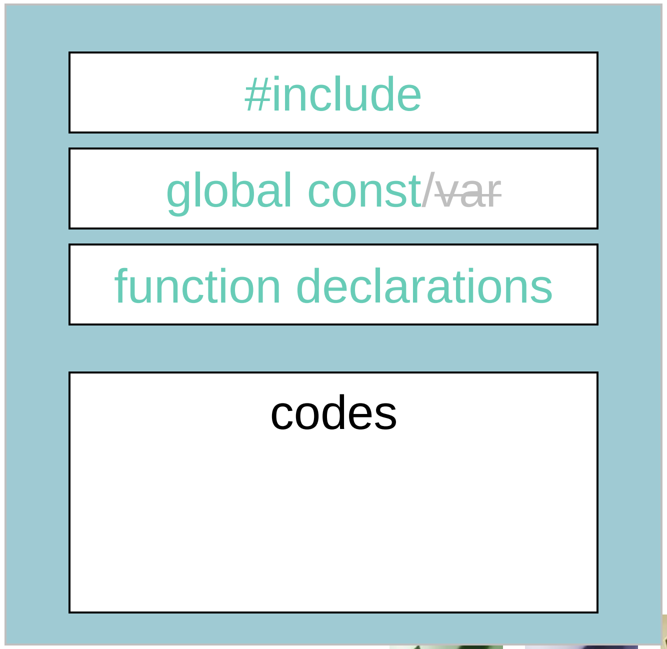
- **Local variables**
  - The variables that are declared *within* the body of a function definition
  - These variables are said to be **local** to that function
  - Can only be accessed within that function



Scope (2)

- **Global variables and constants**
  - The variables (or constants) that are declared outside the body of all the functions
  - These variables can be accessed anywhere, after they are declared
  - Program style
    - To aid readability:

However, global variables can make a program harder to understand and maintain,
so we urge you to **AVOID using global variables**



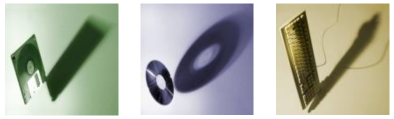
Scope (3)

- **Block**
  - A variable declared inside a compound statement (i.e. inside “{” and “}”) is local to the compound statement
  - A compound statement with declarations is usually called a block. (Notice that the body of a function definition is a block)
- **Variables declared in a `for` loop**

```
for (int i=0; i<=10; i++)
{
   ...              //scope of i
}
```

Pre/Post-conditions

- **Preconditions**
  - States the condition for executing the function
    - The function should not be used and cannot be expected to perform correctly unless the precondition holds
- **Postconditions**
  - Describes the effect of the function call
    - The value returned
    - The changes made to the values of arguments



Pre/Post-conditions (cont'd)

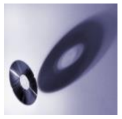
- **Comment function declaration:**

```
void showInterest(double balance, double rate);
//Precondition: balance is nonnegative account
//balance. rate is interest rate as percentage.
//Postcondition: amount of interest on given
//balance at given rate
```

- **You do NOT need to know its definition in order to use the function**
  - All you need to know is given by the precondition and postcondition

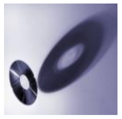
Procedural Abstraction

- **Idea: Do you need to know the code details when you use a program or function?**
- **No!**
  - You only need to know **what** the function’s job is
    - function declaration
    - accompanying comments (pre/post-condition)
  - You do NOT need to know **how** the function does its job
    - function definition



Procedural Abstraction (cont'd)

- **Black box**
  - If a function is well designed, the programmer can use the function as it were a black box
  - That is, you don't need to know what happens inside this box
- **Procedural abstraction**
  - Writing and using functions as if they were black boxes
- **Information hiding**
  - Designing a function so that it can be used as a black box
    - Hides **how** the job is done; only reveals **what** is done
  - a.k.a. *principle of procedural abstraction* or *black box principle*



Summary

- **Terminology**
  - Arguments vs. Parameters
  - Function declaration and function definition
- **Predefined and programmer-defined functions**
- **Scope**
  - Local / global
  - Block
  - in a `for` loop
- **Procedural abstraction**
  - Information hiding