# 2.4 函数

函数用于创建C++程序的模块。C++函数分两种：有返回值的和没有返回值的。

## 2.4.1 使用有返回值的函数

有返回值的的函数将生成一个值，而这个值可赋给变量或在其他表达式中使用。例如，标准C/C++库包含一个名为sqrt()的函数，它返回平方根。假设要计算6.25的平方根，并将这个值赋给变量x，则可以使用下面的语句：

x = sqrt (6.25); // returns the valua 2.5 and assigns it to x

表达式sqrt(6.25)将调用sqrt()函数。表达式sqrt(6.25)被称为函数调用，被调用的函数叫做**被调用函数（called function）**，包含函数调用的函数叫做**调用函数（calling function）**。

圆括号中的值（这里为6.25）是发送给函数的信息，这被称为**传递给函数**。以这种方式发送给函数的值叫做**参数**。函数sqrt()得到的结果为2.5，并将这个值发送给调用函数；发送回去的值叫做函数的**返回值（return value）**。可以这么认为，函数执行完毕后，语句中的函数调用部分将被替换为返回的值。因此，这个例子将返回值赋给变量x。简而言之，参数是发送给函数的信息，返回值是从函数中发送回去的值。

在使用函数之前，C++编译器必须知道函数的参数类犁和返回值类型。如果缺少这些信息，编译器将不知道如何解释返回值。C++提供这种信息的方式是使用**函数原型语句**。C++程序应当为程序中使用的每个函数提供原型。

函数原型之于函数就像变量声明之于变量——指出涉及的类型。sqrt()的函数原型像这样：

double sqrt(double) //function prototype

第一个double意味着sqrt()将返回一个double值。括号中的double意味着sqrt()需要一个double参数。

原型结尾的分号表明它是一条语句，这使得它是一个原型，而不是函数头。如果省略分号，编译器将把这行代码解释为一个函数头，并要求接着提供定义该函数的函数体。

在程序中使用sqrt()时，也必须提供原型。可以用两种方法来实现：

* 在源代码文件中输入函数原型：
* 包含头文件cmath（老系统为math.h），其中定义了原型。

第二种方法更好，因为头文件更有可能使原型正确。

不要混淆函数原型和函数定义。原型只描述函数接口。也就是说，它描述的是发送给函数的信息和返回的信息。而定义中包含了函数的代码，如计算平方根的代码。C和C++将库函数的这两项特性（原型和定义）分开了。库文件中包含了函数的编译代码，而头文件中则包含了原型。

应在首次使用函数之前提供其原型。通常的做法是把原型放到main()函数定义的前面。程序清单2.4演示了库函数sqrt()的用法，它通过包含cmath文件来提供该函数的原型：

**程序清单2.4 sqrt.cpp**

// sqrt.cpp -- using the sqrt() function

#include <iostream>

#include <cmath> // or math.h

int main()

{

using namespace std;

double area;

cout << "Enter the floor area, in square feet, of your home: ";

cin >> area;

double side;

side = sqrt(area);

cout << "That's the equivalent of a square " << side

<< " feet to the side." << endl;

cout << "How fascinating!" << endl;

cin.get();

cin.get();

return 0;

}

下面是该程序的运行情况：

Enter the floor area, in square feet,of your home: 1536

That’s the equivalent of a square 39.1918 feet to the side.

How fascinating!

C++允许在程序的任何地方声明新变量，因此sqrt.cpp在要使用side时才声明它。C++还允许在创建变量时对它进行赋值，因此也可以这样做：

double side = sqr(area);

这个过程叫做**初始化（initialization）**。

## 2.4.2函数变体

有些函数需要多项信息。这些函数使用多个参数，参数间用逗号分开。例如，数学函数pow()接受两个参数，返回值为以第一个参数为底，第二个参数为指数的幂，该函数的原型如下：

double pow(double,double); //prototype of a function with two arguments

要计算5的8次方，可以这样使用该函数：

answer=pow(5.0, 8.0); //function call with a list of arguments

另外一些函数不接受任何参数。例如，有一个C库（与cstdlib或stdlib.h头文件相关的库）包含一个rand()函数，该函数不接受任何参数，并返回一个随机整数。该函数的原型如下：

int rand(void); //prototype of a function that takes no arguments

关键字void明确指出，该函数不接受任何参数。如果省略void，让括号为空，则C++将其解释为一个不接受任何参数的隐式声明。可以这样使用该函数：

myGuess=rand(); //function call with no arguments

还有一些函数没有返回值。例如，假设编写了一个函数，它按美元、美分格式显示数字。当向它传递参数23.5，它将在屏幕上显示$23.50。由于这个函数把值发送给屏幕，而不是调用程序，因此不需要返回值。可以在原型中使用关键字void来指定返回类型，以指出函数没有返回值：

void bucks(double); //prototype for function with no return valua

由于它不返回值，因此不能将该函数调用放在赋值语句或其他表达式中。相反，应使用一条纯粹的函数调用语句：

bucks(1234.56); //function call, no return value

在有些语言中，有返回值的函数被称为函数（function）；没有返回值的函数被称为过程（procedure）或子程序（subroutine）。但C++与C一样，这两种变体都被称为函数。

## 2.4.3用户定义的函数

每个C++程序都必须有一个main()函数，用户必须对它进行定义。假设需要添加另一个用户定义的函数。和库函数一样，也可以通过函数名来调用用户定义的函数。对于库函数，在使用之前必须提供其原型，通常把原型放到main()定义之前。

现在必须提供新函数的源代码。最简单的方法是，将代码放在main()的后面。程序清单2.5演示了这些元素。

**程序清单2.5 ourfunc.cpp**

// ourfunc.cpp -- defining your own function

#include <iostream>

void simon(int); // function prototype for simon()

int main()

{

using namespace std;

simon(3); // call the simon() function

cout << "选择一个整数： ";

int count;

cin >> count;

simon(count); // call it again

cout << "完成！" << endl;

cin.get();

cin.get();

return 0;

}

void simon(int n) // define the simon() function

{

using namespace std;

cout << "Simon says touch your toes " << n << " times." << endl;

} // void functions don't need return statements

main()函数两次调用simon()函数，一次的参数为3，另一次的参数为变量count。在这两次调用之间，用户输入一个整数，用来设置count的值。这个例子没有在cout提示消息中使用换行符。这样将导致用户输入与提示出现在同一行中。下面是运行情况：

Simon says touch your toes3 times.

Pick an integer: 512

Simon says touch your toes 512 times.

Done!

### 1．函数格式

在程序清单2.5中，simon()函数的定义与main()的定义采用的格式相同。首先，有一个函数头；然后是花括号中的函数体。可以把函数的格式统一为如下的情形：

type functionname(argumentlist)

{

statements

}

定义simon()的源代码位于main()的后面。C++不允许将函数定义嵌套在另一个函数定义中。每个函数定义都是独立的，所有函数的创建都是平等的。

### 2．函数头

在程序清单2.5中，simon()函数的函数头如下：

void simon（int n）

开头的void表明simon()没有返回值。因此，第一个函数调用方式如下：

simon(3); // ok for void functions

括号中的int表明，使用simon()时，应提供一个int参数。n是一个新的变量，函数调用时传递的值将被赋给它。因此，下面的函数调用将3赋给simon()函数头中定义的变最n。

当函数体中的cout语句使用n时，将使用函数调用时传递的值。这就是为什么simon (3)在输出中显示3的原因所在。在示例运行中，函数调用simon(count)导致函数显示512，因为这正是赋给count的值。简而言之，simon( )的函数头表明，该函数接受一个int参数，不返回任何值。

对于main()的函数，开头的int表明，main()返回一个整数值：空括号（其中可以包含void）表明，main()没有参数。对于有返回值的函数，应使用关键字return来提供返回值，并结束函数。这就是为什么要在main()结尾使用下述语句的原因：

return 0;

那这个值返回到哪里了呢？答案是，可以将计算机操作系统（如UNIX或Windows）看作调用程序。因此，main()的返回值并不是返回给程序的其他部分，而是返回给操作系统。很多操作系统都可以使用程序的返回值。例如，UNIX外壳脚本和Windows命令行批处理文件都被设计成运行程序，并测试它们的返回值（通常叫做退出值）。通常的约定是，退出值为0则意味着程序运行成功，为非零则意味着存在问题。因此，如果C++程序无法打开文件，可以将它设计为返回一个非零值。然后，便可以设计一个外壳脚本或批处理文件来运行该程序，如果该程序发出指示失败的消息，则采取其他措施。

## 2.4.4用户定义的有返回值的函数

下面编写一个使用返回语句的函数。在函数头中指出返回类型，在函数体结尾处使用return。可以用这种形式为在英国观光的人解决重量的问题。在英国，很多浴室都以英石（stone）为单位，不像美国以磅或公斤为单位。一英石等于14磅，程序清单2.6使用一个函数来完成这样的转换。

**程序清单2.6 convert.cpp**

// convert.cpp -- converts stone to pounds

#include <iostream>

int stonetolb(int); // function prototype

int main()

{

using namespace std;

int stone;

cout << "Enter the weight in stone: ";

cin >> stone;

int pounds = stonetolb(stone);

cout << stone << " stone = ";

cout << pounds << " pounds." << endl;

cin.get();

cin.get();

return 0;

}

int stonetolb(int sts)

{

return 14 \* sts;

}

下面是该程序的运行情况：

Enter the weight in stone:15

15 stone=210 pounds.

在main()中，程序使用cin来给整型变量stone提供一个值。这个值被作为参数传递给stonetolb()函数，在该函数中，这个值被赋给变量sts。然后，stonetolb()用关键字return将14\*sts返回给main()。这表明return后面并非一定得跟一个简单的数字。这里通过使用较为复杂的表达式，避免了创建一个新变量，将结果赋给该变量，然后将它返回。程序将计算表达式的值（这里为210），并将其返回。如果返回表达式的值很麻烦，可以采取更复杂的方式：

int stonetolb(int sts)

{

int pounds = 14 \* sts;

return pounds;

}

这两个版本返回的结果相同，但第二个版本更容易理解和修改，因为它将计算和返回分开了。

通常，在可以使用一个简单常量的地方，都可以使用一个返回值类型与该常量相同的函数。例如，stonetolb()返回一个int值，这意味着可以以下面的方式使用该函数：

int aunt = stonetolb(20);

int aunts = aunt+stonetolb(10);

cout << “Ferdie weighs” << stonetolb(16) << “pounds” << endl;

在上述任何一种情况下，程序都将计算返回值，然后在语句中使用这个值。

这些例子表明，函数原型描述了函数接口，即函数如何与程序的其他部分交互。参数列表指出了何种信息将被传递给函数，函数类型指出了返回值的类型。程序员有时将函数比作一个由出入它们的信息所指定的黑盒子（black boxes）（电工用语）。函数原型将这种观点诠释得淋漓尽致。

函数stonetolb()短小、简单，但包含了全部的函数特性：

* 有函数头和函数体；
* 接受一个参数；
* 返回一个值；
* 需要一个原型。

## 2.4.5在多函数程序中使用using编译指令

在程序清单2.5中，两个函数中都包含下面一条using编译指令：

using namespace std;

这是因为每个函数都使用了cout，因此需要能够访问位于名称空间std中的cout定义。

在程序清单2.5中，可以采用另一种方法让两个函数都能够访问名称空间std．即将编译指令放在函数的外面，且位于两个函数的前面：

// ourfunc.cpp -- repositioning the using directive

#include <iostream>

using namespace std; //affects all function definitions in this file

void simon(int); // function prototype for simon()

int main()

{

simon(3); // call the simon() function

cout << "选择一个整数： ";

int count;

cin >> count;

simon(count); // call it again

cout << "完成！" << endl;

cin.get();

cin.get();

return 0;

}

void simon(int n) // define the simon() function

{

cout << "Simon says touch your toes " << n << " times." << endl;

}

当前通行的理念是，只让需要访问名称空间std的函数访问它是更好的选择。例如，在程序清单2.6中，只有main()函数使用cout，因此没有必要让函数stonetolb()能够访问名称空间std。因此编译指令using被放在函数main()中，使得只有该函数能够访问名称空间std。

总之，让程序能够访问名称空间std的方法有多种，下面是其中的4种。

* 将using namespace std;放在函数定义之前，让文件中所有的函数都能够使用名称空间std中所有的元素。
* 将using namespace std;放在特定的函数定义中，让该函数能够使用名称空间std中的所有元素。
* 在特定的函数中使用类似using std::cout;这样的编译指令，而不是using namespace std;，让该函数能够使用指定的元素，如cout。
* 完全不使用编译指令using，而在需要使用名称空间std中的元素时，使用前缀std::，如下所示：std::cout << “I'm using cout and endl from the std namespace” << std::endl;