# 7.1 复习函数的基本知识

要使用C++函数，必须完成如下工作：

* 提供函数定义；
* 提供函数原型；
* 调用函数。

库函数是已经定义和编译好的函数，同时可以使用标准库头文件提供其原型，因此只需正确地调用这种函数即可。例如，标准C库中有一个strlen()函数，可用来确定字符串的长度。相关的标准头文件cstring包含了strlen()和其他一些与字符串相关的函数的原型。这些预备工作使程序员能够在程序中随意使用strlen()函数。

然而，创建自己的函数时，必须自行处理这3个方面——定义、提供原型和调用。程序清单7.1用一个简短的示例演示了这3个步骤。

**程序清单7.1 calling.cpp**

// calling.cpp -- 函数定义，函数原型和函数调用

#include <iostream>

void simple(); // 函数原型

int main()

{

 using namespace std;

 cout << "main() will call the simple() function:\n";

 simple(); // 函数调用

 cout << "main() is finished with the simple() function.\n";

 cin.get();

 return 0;

}

// 函数定义

void simple()

{

 using namespace std;

 cout << "I'm but a simple function.\n";

}

下面是该程序的输出：

main() will call the simple() function:

I’m but a simple function.

main() is finished with the simple() function.

执行函数simple()时，将暂停执行main()中的代码：等函数simple()执行完毕后，继续执行main()中的代码。在每个函数定义中．都使用一条using编译指令，因为每个函数都使用了cout。另一种方法是，在函数定义之前放一条using编译指令或在函数中使用std::cout。

下面详细介绍这3个步骤。

## 7.1.1定义函数

可以将函数分成两类：没有返回值的函数和有返回值的函数。没有返回位的函数被称为void函数，其通用格式如下：

void functionName(parameterList)

{

 statement(s)

return; // 可选

}

其中，parameterList指定了传递给函数的参数类型和数量，本章后面将更详细题介绍该列表。可选的返回语句标记了函数的结尾;否则，函数将在有花括号处结束。通常，可以用void函数来执行某种操作。例如，将Cheers!打印指定次数（n）的函数如下：

void cheers(int n) // 没有返回值

{

 for(int i=0;i<n;i++)

std::cout << “Cheers”;

std::cout << std::endl;

}

参数列表int意味着调用函数cheers()时，应将一个int值作为参数传递给它。

有返回值的函数将生成一个值，并将它返回给调用函数。换句话来说，如果函数返回9.0平方根（sqrt(9.0)），则该函数调用的值3.0。这种函数的类型被声明为返回值的类型，其通用格式如下：

typeName functionName(parameterList)

{

statements

return value; // value的类型被转换为typeName

}

对于有返回值的函数，必须使用返回语句，以便将值返回给调用函数。值本身可以是常量、变量，也可以是表达式，只是其结果的类型必须为typeNmae类型或可以被转换为typeName（例如，如果声明的返回类型为double，而函数返回一个int表达式，则该int值将被强制转换为double类型）。然后，函数将最终的值返回给调用函数。C++对于返回值的类型有—定的限制：不能是数组，但可以是其他任何类型——整数、浮点数、指针，甚至可以是结构和对象！（有趣的是，虽然C++函数不能直接返回数组，但可以将数组作为结构或对象组成部分来返回。）

作为一名程序员，并不需要知道函数是如何返回值的，但是对这个问题有所了解将有助于澄清概念。通常，函数通过将返回值复制到指定的CPU寄存器或内存单元来将其返回。随后，调用程序将查看该内存单元。返回函数和调用函数必须就该内存单元中存储的数据的类型达成一致。函数原型将返回值类型告知调用程序，而函数定义命令被调用函数应返回什么类型的数据。在原型中提供与定义相同的信息似乎有些多余，但这样做确实有道理。要让信差从办公室的办公桌上取走一些物品，则向信差和办公室中的同事交代自己的意图将提高信差顺利完成这项工作的概率。

函数在执行返回语句后结束。如果函数包含多条返回语句（例如，它们位于不同的if else选项中），则函数在执行遇到的第一条返回语句后结束。

int bigger(int a,int b)

{

 if(a > b)

 return a; // 如果a>b，函数就在这里结束

 else

 return b; // 否则，函数在这里结束

}

有返回值的函数向调用程序返回一个值，然后调用程序可以将其赋给变量、显示或将其用于别的用途。下面是一个简单的例子，函数返回double值的立方：

double cube (double x) // x乘以x乘以x

{

return x \* x \* x;

}

例如，函数调用cube(1.2)将返回1.728。上述返回语句使用了一个表达式，函数将计算该表达式的值（这里为1.728），并将其返回。

## 7.1.2 函数原型与函数调用

函数原型经常隐藏在include文件中。程序清单7.2在一个程序中使用了函数cheer()和cube()。请留意其中的函数原型。

**程序清单7.2 protos.cpp**

// protos.cpp -- 使用函数原型和函数调用

#include <iostream>

void cheers(int); // 函数原型：没有返回值

double cube(double x); // 函数原型：返回一个double类型

int main()

{

 using namespace std;

 cheers(5); // 函数调用

 cout << "Give me a number: ";

 double side;

 cin >> side;

 double volume = cube(side); // 函数调用

 cout << "A " << side <<"-foot cube has a volume of ";

 cout << volume << " cubic feet.\n";

 cheers(cube(2)); // prototype protection at work

 cin.get();

 cin.get();

 return 0;

}

void cheers(int n)

{

 using namespace std;

 for (int i = 0; i < n; i++)

 cout << "Cheers! ";

 cout << endl;

}

double cube(double x)

{

 return x \* x \* x;

}

在程序清单7.2的程序中，只需要使用名称空间std中成员的函数中使用了编译指令using。下面是该程序的运行情况：

Cheers!Cheers!Cheers!Cheers!Cheers!

Give me a number:5

A 5-foot cube has a volume of 125 cubic feet.

Cheers!Cheers!Cheers!Cheers!Cheers!Cheers!Cheers!Cheers!

main()使用函数名和参数（后而跟一个分号）来调用void类型的函数：cheers(5);，这是一个函数调用语句。但由于cube()有返回值，因此main()可以将其用在赋值语句中：

double volume=cube(side);

读者应将重点放在原型上。首先，需要知道C++要求提供原型的原因。其次，由于C++要求提供原型，因此还应知道正确的语法。下面依次介绍这几点，将程序清单7.2作为讨论的基础。

### 1．为什么需要原型

原型描述了函数到编译器的接口，也就是说，它将函数返回值的类型（如果有的话）以及参数的类型和数量告诉编译器。例如，请看原型将如何影响程序清单7.2中下述函数调用：

double volume=cube(side);

首先，原型告诉编译器，cube()有一个double参数。如果程序没有提供这样的参数，原型将让编译器能够捕获这种错误。其次，cube()函数完成计算后，将把返回值放置在指定的位置——可能是CPU寄存器，也可能是内存中。然后调用函数（这里为main()）将从这个位置取得返回值。由于原型指出了cube()的类型为double，因此编译器知道应检索多少个字节以及如何解释它们。

为何编译器需要原型？难道它就不能在文件中进一步查找，以了解函数是如何定义的吗？这种方法的一个问题是效率不高。编译器在搜索文件的剩余部分时将必须停止对main()的编译。一个更严重的问题是，函数甚至可能并不在文件中。C++允许将一个程序放在多个文件中，单独编译这些文件，然后再将它们组合起来。在这种情况下，编译器在编译main()时，可能无权访问函数代码。如果函数位于库中，情况也将如此。避免使用函数原型的唯一办法是，在首次使用函数之前定义它，但这并不总是可行的。另外，C++的编程风格是将main()放在最前面，因为它通常提供了程序的整体结构。

### 2．原型的语法

函数原型是一条语句，因此必须以分号结束。获得原型最简单的方法是，复制函数定义中的函数头，并添加分号。对于cube()，程序清单7.2中的程序正是这样做的：

double cube(double x); // 在函数头之后添加;就可以获得原型

然而，函数原型不要求提供变量名，有类型列表就足够了。对于cheer()的原型，该程序只提供了参数类型：

void cheers(int); // 在原型中不提供变量名也可以

通常，在原型的参数列表中，可以包括变量名，也可以不包括。原型中的变量名相当于占位符，因此不必与函数定义中的变量名相同。

### 3．原型的功能

原型可以帮助编译器完成许多工作；对程序员而言，它们可以极大地降低程序出错的几率。具体来说，原型确保以下几点：

* 编译器正确处理函数返回值；
* 编译器检查使用的参数数目是否正确；
* 编译器检测使用的参数类型是否正确。如果不正确，则转换为正确的类型（如果可能的话）。

前面已经讨论了如何正确处理返回值。下面来看一看参数数目不对时将发生的情况。例如，假设进行如下调用：

double z = cube();

如果没有函数原型，编译器将允许它通过。当函数被调用时，它将找到cube()调用存放值的位置，并使用这里的值。这正是ANSIC从C++借鉴原型之前，C语言的工作方式。由于对于ANSI C来说，原型是可选的，因此有些C语言程序正是这样工作的。但在C++中，原型不是可选的，因此可以确保不会发生这类错误。

接下来，假设提供了一个参数，但其类型不正确。在C语言中，这将造成奇怪的错误。例如，如果函数需要一个int值（假设占16位），而程序员传递了一个double值（假设占64位），则函数将只检查64位中的前16位，并试图将它们解释为一个int值。但C++自动将传递的值转换为原型中指定的类型，条件是两者都是算术类型。例如，程序清单7.2将能够应付下述语句中两次出现的类型不匹配的情况：

cheers(cube(2});

首先，程序将int的值2传递给cube()，而后者期望的是double类型。编译器注意到，cube()原型指定了一个double类型参数，因此将2转换为2.0——一个double值。接下来，cube()返叫一个double值(8.0)，这个值被用作cheer()的参数。编译器将再一次检查原型，并发现cheer()要求一个int参数，因此它将返回值转换为整数8。通常，原型自动将被传递的参数强制转换为期望的类型。（但第8章将介绍的函数重载可能导致二义性，因此不允许某些自动强制类型转换。）

自动类型转换并不能避免所有可能的错误。例如．如果将8.33E27传递给期望一个int值的函数，则这样大的值将不能被正确转换为int值。当较大的类型被自动转换为较小的类型时，有些编译器将发出警告，指出这可能会丢失数据。

仅当有意义时，原型化才会导致类型转换。例如，原型不会将整数转换为结构或指针。

在编译阶段进行的原型化被称为**静态类型检查（static type checking）**，可以看出，静态类型检查捕获许多在运行阶段非常难以捕获的错误。