Chapter 1

开始学习C++

1.1 进入 C++

1.1.1 main() 函数

C++ 和 C 一样，也是用终止符（terminator），而不是分隔符。终止符是一个分号，他是语句的计数标记，是语句的组成部分，而不是语句之间的标记。结论是：在 C++ 中，不能省略分号。

作为接口的函数头

通常，C++ 函数可被其他函数激活或调用，函数头描述了函数与调用它的函数之间的接口。位于函数名前面的部分叫做函数返回类型，它描述的是从函数返回给调用它的函数的信息。函数名后括号中的部分叫做形参列表（argument list）或参数列表（parameter list）；它描述的是从调用函数传递给被调用函数的信息。

在括号中使用关键字参数 void 显示地指出，函数不接受任何参数。在 C++ 中（不是 C）中，让括号空着与在括号中使用 void 等效（在 C 中，让括号空着意味着对是否接受参数保持沉默）。

表 1.1: 头文件命名约定

头文件类型	约定	示例	说明
C++ 旧式风格	以 .h 结尾	iostream.h	C++ 程序可以使用
C 旧式风格	以 .h 结尾	math.h	C、 C++ 程序可以使用
C++ 新式风格	没有拓展名	iostream	C++ 程序可以使用，使用 namespace std
转化后的 C	加上前缀 c，没有拓展名	cmath	C++ 程序可以使用，可以使用不是 C 的特性，如 namespace std

1.1.2 C++ 预处理器和 iostream 文件

1.1.3 头文件名

1.1.4 名称空间

如果使用 iostream，而不是 iostream.h，则应使用下面的名称空间编译指令来使 iostream 中的定义对程序可用：

```
using namespace std;
```

1.1.5 使用 cout 进行 C++ 输出

从概念上看，输出是一个流，即从程序流出的一系列字符。cout 对象表示这种流，其属性是在iostream文件中定义的。cout 的对象属性包括一个插入运算符（<<），它可以将其右侧的信息插入到流中。

控制符 endl

endl 是一个特殊的 C++ 符号，表示一个重要的概念：重起一行。在输出流中插入 endl 将导致屏幕光标移到下一行开头。诸如 endl 等对于 cout 来

说有特殊含义的特殊符号被称为控制符（manipulator）。和 cout 一样，endl 也是在头文件 iostream 中定义的，且位于名称空间 std 中。

1.2 C++ 语句

C++ 程序是一组函数，而每个函数又是一组语句。声明语句创建变量，赋值语句给该变量提供一个值。

1.2.1 声明语句和变量

为什么变量必须声明？有些语言（最典型的是 BASIC）在使用新名称时创建新的变量，而不用显式地进行声明。这看上去对用户比较友好，事实上从短期上说确实如此。问题是，如果错误地拼写了变量名，将在不知情的情况下创建一个新的变量。

程序中的声明语句叫做**定义声明**（defining declaration）语句，简称为定义（definition）。这意味着它将导致编译器为变量分配内存空间。在较为复杂的情况下，还可能有**引用声明**（reference declaration）。这些声明命令计算机使用在其他地方定义的变量。

1.2.2 cout 的新花样

cout 的智能行为源自 C++ 的面向对象特性。实际上，C++ 插入运算符（<<）将根据其后的数据类型相应地调整其行为，这是一个运算符重载的例子。

1.3 其他 C++ 语句

1.3.1 使用 cin

就像 C++ 将输出看作是流出程序的字符流一样，它也将输入看作是流入程序的字符流。iostream 文件将 cin 定义为一个表示这种流的对象。输出时，¡¡运算符将字符串插入到输出流中；输入时，cin 使用 >> 运算符从输入流中抽取字符。通常，需要在运算符右侧提供一个变量，以接收抽取的信息（符号 << 和 >> 被选择用来指示信息流的方向）。

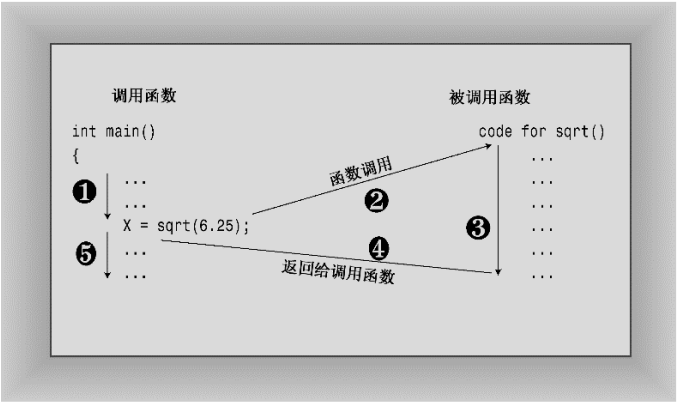

图 1.1: 调用函数

1.3.2 类简介

类是用户定义的一种数据类型。要定义类，需要描述它能够表示什么信息和可对数据执行哪些操作。类之于对象就像类型之于变量。也就是说，类定义描述的是数据格式及其用法，而对象则是根据数据格式规范创建的实体。

注意：类描述了一种数据类型的全部属性（包括可使用它执行的操作），对象是根据这些描述创建的实体。

类描述指定了可对类对象执行的所有操作。要对特定对象执行这些允许的操作，需要给该对象发送一条消息。C++提供了两种发送消息的方式：一种方式是使用类方法（本质上就是函数调用）；另一种方式是重新定义运算符，cin 和 cout 采用的就是这种方式。

1.4 函数

C++函数分两种：有返回值的和没有返回值的。

1.4.1 使用有返回值的函数

有返回值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。

表达式 `sqrt(6.25)` 将调用 `sqrt()`函数。表达式 `sqrt(6.25)` 被称为函

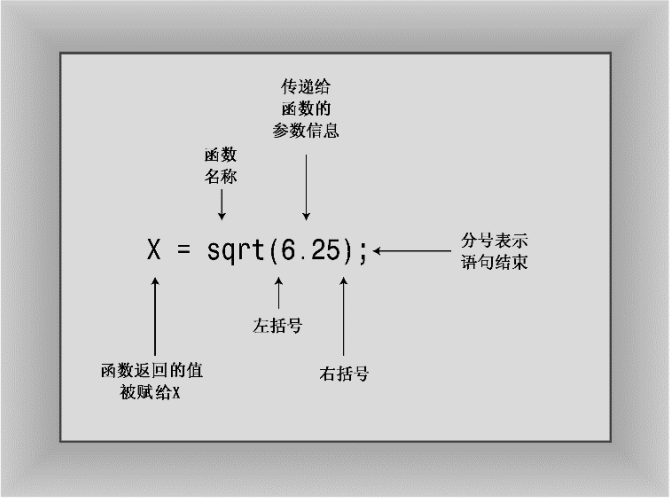

图 1.2: 函数调用的句法

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，参见Figure 1.1）。

在使用函数之前，C++ 编译器必须知道函数的参数类型和返回值类型。也就是说，函数是返回整数、字符、小数、有罪裁决还是别的什么东西？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。因此该原型对 sqrt() 的描述和下面代码中使用的函数相同：

```
double x; // declare x as a type double variable
x = sqrt(6.25);
```

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。

不要混淆函数原型和函数定义。可以看出，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。

1.4.2 函数变体

1.4.3 用户定义的函数

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。但现在您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。

函数格式

注意，定义 simon() 的源代码位于main( )的后面。和 C 一样（但不同于 Pascal），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（参见 Figure 1.3）。

main()返回一个int值，而程序员要求它返回整数0。但可能会产生疑问，将这个值返回到哪里了呢？毕竟，程序中没有哪个地方可以看出对main()的调用：

```
squeeze = main(); // absent from our programs
```

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。

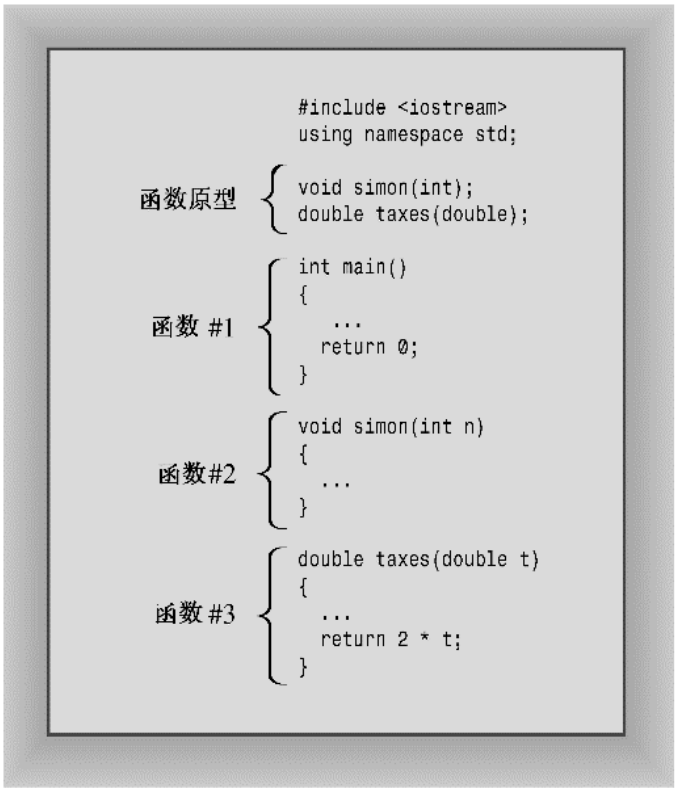

图 1.3: 函数定义在文件中依次出现

1.4.4 用户定义的有返回值的函数

main() 函数已经揭示了有返回值的函数的格式：在函数头中指出返回类型，在函数体结尾处使用 return。

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。

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。

1.4.5 在多函数程序中使用 using 编译指令

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。

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。

1. 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。
2. 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。
3. 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。
4. 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;
```