cin & cout

1.2 初识输入输出

C++语言并未定义任何输入输出（IO）语句，取而代之，包含了一个全面的**标准库**（standard library）来提供 IO 机制（以及很多其他设施）。对于很多用途，包括本书中的示例来说，我们只需了解 IO 库中一部分基本概念和操作。

本书中的很多示例都使用了 **iostream** 库。iostream 库包含两个基础类型 **istream** 和 **ostream**，分别表示输入流和输出流。一个流就是一个字符序列，是从 IO 设备读出或写入 IO 设备的。术语“流”（stream）想要表达的是，随着时间的推移，字符是顺序生成或消耗的。

标准输入输出对象

6

标准库定义了 4 个 IO 对象。为了处理输入，我们使用一个名为 **cin**（发音为 see-in）的 istream 类型的对象。这个对象也被称为**标准输入**（standard input）。对于输出，我们使用一个名为 **cout**（发音为 see-out）的 ostream 类型的对象。此对象也被称为**标准输出**（standard output）。标准库还定义了其他两个 ostream 对象，名为 **cerr** 和 **clog**（发音分别为 see-err 和 see-log）。我们通常用 cerr 来输出警告和错误消息，因此它也被称为**标准错误**（standard error）。而 clog 用来输出程序运行时的一般性信息。

系统通常将程序所运行的窗口与这些对象关联起来。因此，当我们读取 cin，数据将从程序正在运行的窗口读入，当我们向 cout、cerr 和 clog 写入数据时，将会写到同一个窗口。

一个使用 IO 库的程序

在书店程序中，我们需要将多条记录合并成单一的汇总记录。作为一个相关的，但更简单的问题，我们先来看一下如何将两个数相加。通过使用 IO 库，我们可以扩展 main 程序，使之能提示用户输入两个数，然后输出它们的和：

```
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1 = 0, v2 = 0;
    std::cin >> v1 >> v2;
    std::cout << "The sum of " << v1 << " and " << v2
              << " is " << v1 + v2 << std::endl;
    return 0;
}
```

这个程序开始时在用户屏幕打印

```
Enter two numbers:
```

然后等待用户输入。如果用户键入

```
3 7
```

然后键入一个回车，则程序产生如下输出：

```
The sum of 3 and 7 is 10
```

程序的第一行

```
#include <iostream>
```

告诉编译器我们想要使用 iostream 库。尖括号中的名字（本例中是 iostream）指出了一个**头文件**（header）。每个使用标准库设施的程序都必须包含相关的头文件。#include 指令和头文件的名字必须写在同一行中。通常情况下，#include 指令必须出现在所有函数之外。我们一般将一个程序的所有#include 指令都放在源文件的开始位置。

7

向流写入数据

main 的函数体的第一条语句执行了一个**表达式**（expression）。在 C++中，一个表达式产生一个计算结果，它由一个或多个运算对象和（通常是）一个运算符组成。这条语句中的表达式使用了**输出运算符**（<<）在标准输出上打印消息：

```
std::cout << "Enter two numbers:" << std::endl;
```

<<运算符接受两个运算对象：左侧的运算对象必须是一个 ostream 对象，右侧的运算对象是要打印的值。此运算符将给定的值写到给定的 ostream 对象中。输出运算符的计算结果就是其左侧运算对象。即，计算结果就是我们写入给定值的那个 ostream 对象。

我们的输出语句使用了两次<<运算符。因为此运算符返回其左侧的运算对象，因此第一个运算符的结果成为了第二个运算符的左侧运算对象。这样，我们就可以将输出请求连接起来。因此，我们的表达式等价于

```
(std::cout << "Enter two numbers:") << std::endl;
```

链中每个运算符的左侧运算对象都是相同的，在本例中是 std::cout。我们也可以用两条语句生成相同的输出：

```
std::cout << "Enter two numbers:";
std::cout << std::endl;
```

第一个输出运算符给用户打印一条消息。这个消息是一个**字符串字面值常量**（string literal），是用一对双引号包围的字符序列。在双引号之间的文本被打印到标准输出。

第二个运算符打印 endl，这是一个被称为**操纵符**（manipulator）的特殊值。写入 endl 的效果是结束当前行，并将与设备关联的**缓冲区**（buffer）中的内容刷到设备中。缓冲刷新操作可以保证到目前为止程序所产生的所有输出都真正写入输出流中，而不是仅停留在内存中等待写入流。

程序员常常在调试时添加打印语句。这类语句应该保证“一直”刷新流。否则，如果程序崩溃，输出可能还留在缓冲区中，从而导致关于程序崩溃位置的错误推断。

使用标准库中的名字

细心的读者可能会注意到这个程序使用了 std::cout 和 std::endl，而不是直接的 cout 和 endl。前缀 std::指出名字 cout 和 endl 是定义在名为 **std** 的**命名空间**（namespace）中的。命名空间可以帮助我们避免不经意的名字定义冲突，以及使用库中相同名字导致的冲突。标准库定义的所有名字都在命名空间 std 中。

8

通过命名空间使用标准库有一个副作用：当使用标准库中的一个名字时，必须显式说明我们想使用来自命名空间 std 中的名字。例如，需要写出 std::cout，通过使用**作用域运算符**（::）来指出我们想使用定义在命名空间 std 中的名字 cout。3.1 节（第 74 页）将给出一个更简单的访问标准库中名字的方法。

从流读取数据

在提示用户输入数据之后，接下来我们希望读入用户的输入。首先定义两个名为 v1 和 v2 的变量（variable）来保存输入：

```
int v1 = 0, v2 = 0;
```

我们将这两个变量定义为 int 类型，int 是一种内置类型，用来表示整数。还将它们初始化（initialize）为 0。初始化一个变量，就是在变量创建的同时为它赋予一个值。

下一条语句是

```
std::cin >> v1 >> v2;
```

它读入输入数据。**输入运算符**（**>>**）与输出运算符类似，它接受一个 istream 作为其左侧运算对象，接受一个对象作为其右侧运算对象。它从给定的 istream 读入数据，并存入给定对象中。与输出运算符类似，输入运算符返回其左侧运算对象作为其计算结果。因此，此表达式等价于

```
(std::cin >> v1) >> v2;
```

由于此运算符返回其左侧运算对象，因此我们可以将一系列输入请求合并到单一语句中。本例中的输入操作从 std::cin 读入两个值，并将第一个值存入 v1，将第二个值存入 v2。换句话说，它与下面两条语句的执行结果是一样的

```
std::cin >> v1;
std::cin >> v2;
```

完成程序

剩下的就是打印计算结果了：

```
std::cout << "The sum of " << v1 << " and " << v2
          << " is " << v1 + v2 << std::endl;
```

这条语句虽然比提示用户输入的打印语句更长，但原理上是一样的，它将每个运算对象打印在标准输出上。本例一个有意思的地方在于，运算对象并不都是相同类型的值。某些运算对象是字符串字面值常量，例如"The sum of "。其他运算对象则是 int 值，如 v1、v2 以及算术表达式 v1+v2 的计算结果。标准库定义了不同版本的输入输出运算符，来处理这些不同类型的运算对象。

comments

9

1.2 节练习

练习 1.3：编写程序，在标准输出上打印 Hello, World。

练习 1.4：我们的程序使用加法运算符+来将两个数相加。编写程序使用乘法运算符*，来打印两个数的积。

练习 1.5：我们将所有输出操作放在一条很长的语句中。重写程序，将每个运算对象的打印操作放在一条独立的语句中。

练习 1.6：解释下面程序片段是否合法。

```
std::cout << "The sum of " << v1;
          << " and " << v2;
          << " is " << v1 + v2 << std::endl;
```

如果程序是合法的，它输出什么？如果程序不合法，原因何在？应该如何修正？

1.3 注释简介

在程序变得更复杂之前，我们应该了解一下 C++是如何处理注释（comments）的。注释可以帮助人类读者理解程序。注释通常用于概述算法，确定变量的用途，或者解释晦涩难懂的代码段。编译器会忽略注释，因此注释对程序的行为或性能不会有任何影响。

虽然编译器会忽略注释，但读者并不会。即使系统文档的其他部分已经过时，程序员也倾向于相信注释的内容是正确可信的。因此，错误的注释比完全没有注释更糟糕，因为它会误导读者。因此，当你修改代码时，不要忘记同时更新注释！

C++中注释的种类

C++中有两种注释：单行注释和界定符对注释。单行注释以双斜线（//）开始，以换行符结束。当前行双斜线右侧的所有内容都会被编译器忽略，这种注释可以包含任何文本，包括额外的双斜线。

另一种注释使用继承自 C 语言的两个界定符（/*和*/）。这种注释以/*开始，以*/结束，可以包含除*/外的任意内容，包括换行符。编译器将落在/*和*/之间的所有内容都当作注释。

注释界定符可以放置于任何允许放置制表符、空格符或换行符的地方。注释界定符可以跨越程序中的多行，但这并不是必须的。当注释界定符跨越多行时，最好能显式指出其内部的程序行都属于多行注释的一部分。我们所采用的风格是，注释内的每行都以一个星号开头，从而指出整个范围都是多行注释的一部分。

10

程序中通常同时包含两种形式的注释。注释界定符对通常用于多行解释，而双斜线注释常用于半行和单行附注。

```
#include <iostream>
/*
 * 简单主函数：
 * 读取两个数，求它们的和
 */
int main()
{
```

```
    // 提示用户输入两个数
    std::cout << "Enter two numbers:" << std::endl;
    int v1 = 0, v2 = 0;      // 保存我们读入的输入数据的变量
    std::cin >> v1 >> v2;    // 读取输入数据
    std::cout << "The sum of " << v1 << " and " << v2
              << " is " << v1 + v2 << std::endl;
    return 0;
}
```

在本书中，我们用楷体来突出显示注释。在实际程序中，注释文本的显示形式是否区别于程序代码文本的显示，依赖于你所使用的程序设计环境是否提供这一特性。

注释界定符不能嵌套

界定符对形式的注释是以/*开始，以*/结束的。因此，一个注释不能嵌套在另一个注释之内。编译器对这类问题所给出的错误信息可能是难以理解、令人迷惑的。例如，在你的系统中编译下面的程序，就会产生错误：

```
/*
 * 注释对/* */不能嵌套。
 * "不能嵌套"几个字会被认为是源码，
 * 像剩余程序一样处理
 */
int main()
{
    return 0;
}
```

我们通常需要在调试期间注释掉一些代码。由于这些代码可能包含界定符对形式的注释，因此可能导致注释嵌套错误，因此最好的方式是用单行注释方式注释掉代码段的每一行。

```
// /*
// * 单行注释中的任何内容都会被忽略
// * 包括嵌套的注释对也一样会被忽略
// */
```

1.3 节练习

11

练习 1.7：编译一个包含不正确的嵌套注释的程序，观察编译器返回的错误信息。

练习 1.8：指出下列哪些输出语句是合法的（如果有的话）：

```
std::cout << "/*";
std::cout << "*/";
std::cout << /* "*/" */;
std::cout << /* "*/" /* "/*" */;
```

预测编译这些语句会产生什么样的结果，实际编译这些语句来验证你的答案（编写一个小程序，每次将上述一条语句作为其主体），改正每个编译错误。

While & for & if loop

1.4 控制流

语句一般是顺序执行的：语句块的第一条语句首先执行，然后是第二条语句，依此类推。当然，少数程序，包括我们解决书店问题的程序，都可以写成只有顺序执行的形式。但程序设计语言提供了多种不同的控制流语句，允许我们写出更为复杂的执行路径。

1.4.1 while 语句

while 语句反复执行一段代码，直至给定条件为假为止。我们可以用 while 语句编写一段程序，求 1 到 10 这 10 个数之和：

```
#include <iostream>
int main()
{
    int sum = 0, val = 1;
    // 只要 val 的值小于等于 10，while 循环就会持续执行
    while (val <= 10) {
        sum += val;  // 将 sum + val 赋予 sum
        ++val;       // 将 val 加 1
    }
    std::cout << "Sum of 1 to 10 inclusive is "
              << sum << std::endl;
    return 0;
}
```

我们编译并执行这个程序，它会打印出

```
Sum of 1 to 10 inclusive is 55
```

与之前的例子一样，我们首先包含头文件 iostream，然后定义 main。在 main 中我们定义两个 int 变量：sum 用来保存和；val 用来表示从 1 到 10 的每个数。我们将 sum 的初值设置为 0，val 从 1 开始。

12

这个程序的新内容是 while 语句。while 语句的形式为

```
while (condition)
    statement
```

while 语句的执行过程是交替地检测 *condition* 条件和执行关联的语句 *statement*，直至 *condition* 为假时停止。所谓**条件**（condition）就是一个产生真或假的结果的表达式。只要 *condition* 为真，*statement* 就会被执行。当执行完 *statement*，会再次检测 *condition*。如果 *condition* 仍为真，*statement* 再次被执行。while 语句持续地交替检测 *condition* 和执行 *statement*，直至 *condition* 为假为止。

在本程序中，while 语句是这样的

```
// 只要 val 的值小于等于 10，while 循环就会持续执行
while (val <= 10) {
    sum += val;  // 将 sum + val 赋予 sum
    ++val;       // 将 val 加 1
}
```

条件中使用了**小于等于运算符**（**<=**）来比较 val 的当前值和 10。只要 val 小于等于 10，条件即为真。如果条件为真，就执行 while 循环体。在本例中，循环体是由两条语句组

成的语句块：

```
{
    sum += val;       // 将 sum + val 赋予 sum
    ++val;            // 将 val 加 1
}
```

所谓语句块（block），就是用花括号包围的零条或多条语句的序列。语句块也是语句的一种，在任何要求使用语句的地方都可以使用语句块。在本例中，语句块的第一条语句使用了**复合赋值运算符**（+=）。此运算符将其右侧的运算对象加到左侧运算对象上，将结果保存到左侧运算对象中。它本质上与一个加法结合一个**赋值**（assignment）是相同的：

```
sum = sum + val; // 将 sum + val 赋予 sum
```

因此，语句块中第一条语句将 val 的值加到当前和 sum 上，并将结果保存在 sum 中。

下一条语句

```
++val; // 将 val 加 1
```

使用**前缀递增运算符**（++）。递增运算符将运算对象的值增加 1。++val 等价于 val=val+1。

执行完 while 循环体后，循环会再次对条件进行求值。如果 val 的值（现在已经增加了）仍然小于等于 10，则 while 的循环体会再次执行。循环连续检测条件、执行循环体，直至 val 不再小于等于 10 为止。

一旦 val 大于 10，程序跳出 while 循环，继续执行 while 之后的语句。在本例中，继续执行打印输出语句，然后执行 return 语句完成 main 程序。

1.4.1 节练习

13

练习 1.9：编写程序，使用 while 循环将 50 到 100 的整数相加。

练习 1.10：除了++运算符将运算对象的值增加 1 之外，还有一个递减运算符（—）实现将值减少 1。编写程序，使用递减运算符在循环中按递减顺序打印出 10 到 0 之间的整数。

练习 1.11：编写程序，提示用户输入两个整数，打印出这两个整数所指定的范围内的所有整数。

1.4.2 for 语句

在我们的 while 循环例子中，使用了变量 val 来控制循环执行次数。我们在循环条件中检测 val 的值，在 while 循环体中将 val 递增。

这种在循环条件中检测变量、在循环体中递增变量的模式使用非常频繁，以至于 C++ 语言专门定义了第二种循环语句——**for 语句**，来简化符合这种模式的语句。可以用 for 语句来重写从 1 加到 10 的程序：

```
#include <iostream>
int main()
{
    int sum = 0;
    // 从 1 加到 10
    for (int val = 1; val <= 10; ++val)
```

```
        sum += val; // 等价于 sum = sum + val
    std::cout << "Sum of 1 to 10 inclusive is "
              << sum << std::endl;
    return 0;
}
```

与之前一样，我们定义了变量 sum，并将其初始化为 0。在此版本中，val 的定义是 for 语句的一部分：

```
for (int val = 1; val <= 10; ++val)
    sum += val;
```

每个 for 语句都包含两部分：循环头和循环体。循环头控制循环体的执行次数，它由三部分组成：一个初始化语句（*init-statement*）、一个循环条件（*condition*）以及一个表达式（*expression*）。在本例中，初始化语句为

```
int val = 1
```

它定义了一个名为 val 的 int 型对象，并为其赋初值 1。变量 val 仅在 for 循环内部存在，在循环结束之后是不能使用的。初始化语句只在 for 循环入口处执行一次。循环条件

```
val <= 10
```

14 比较 val 的值和 10。循环体每次执行前都会先检查循环条件。只要 val 小于等于 10，就会执行 for 循环体。表达式在 for 循环体之后执行。在本例中，表达式

```
++val
```

使用前缀递增运算符将 val 的值增加 1。执行完表达式后，for 语句重新检测循环条件。如果 val 的新值仍然小于等于 10，就再次执行 for 循环体。执行完循环体后，再次将 val 的值增加 1。循环持续这一过程直至循环条件为假。

在此循环中，for 循环体执行加法

```
sum += val; // 等价于 sum = sum + val
```

简要重述一下 for 循环的总体执行流程：

1. 创建变量 val，将其初始化为 1。
2. 检测 val 是否小于等于 10。若检测成功，执行 for 循环体。若失败，退出循环，继续执行 for 循环体之后的第一条语句。
3. 将 val 的值增加 1。
4. 重复第 2 步中的条件检测，只要条件为真就继续执行剩余步骤。

1.4.2 节练习

练习 1.12：下面的 for 循环完成了什么功能？sum 的终值是多少？

```
int sum = 0;
for (int i = -100; i <= 100; ++i)
    sum += i;
```

练习 1.13：使用 for 循环重做 1.4.1 节中的所有练习（第 11 页）。

1.4.3 读取数量不定的输入数据

在前一节中，我们编写程序实现了 1 到 10 这 10 个整数求和。扩展此程序一个很自然的方向是实现对用户输入的一组数求和。在这种情况下，我们预先不知道要对多少个数求和，这就需要不断读取数据直至没有新的输入为止：

```
#include <iostream>                                          15
int main()
{
    int sum = 0, value = 0;
    // 读取数据直到遇到文件尾，计算所有读入的值的和
    while (std::cin >> value)
        sum += value; // 等价于 sum = sum + value
    std::cout << "Sum is: " << sum << std::endl;
    return 0;
}
```

如果我们输入

```
3 4 5 6
```

则程序会输出

```
Sum is: 18
```

main 的首行定义了两个名为 sum 和 value 的 int 变量，均初始化为 0。我们使用 value 保存用户输入的每个数，数据读取操作是在 while 的循环条件中完成的：

```
while (std::cin >> value)
```

while 循环条件的求值就是执行表达式

```
std::cin >> value
```

此表达式从标准输入读取下一个数，保存在 value 中。输入运算符（参见 1.2 节，第 7 页）返回其左侧运算对象，在本例中是 std::cin。因此，此循环条件实际上检测的是 std::cin。

当我们使用一个 istream 对象作为条件时，其效果是检测流的状态。如果流是有效的，即流未遇到错误，那么检测成功。当遇到文件结束符（end-of-file），或遇到一个无效输入时（例如读入的值不是一个整数），istream 对象的状态会变为无效。处于无效状态的 istream 对象会使条件变为假。

因此，我们的 while 循环会一直执行直至遇到文件结束符（或输入错误）。while 循环体使用复合赋值运算符将当前值加到 sum 上。一旦条件失败，while 循环将会结束。我们将执行下一条语句，打印 sum 的值和一个 endl。

从键盘输入文件结束符

当从键盘向程序输入数据时，对于如何指出文件结束，不同操作系统有不同的约定。在 Windows 系统中，输入文件结束符的方法是敲 Ctrl+Z（按住 Ctrl 键的同时按 Z 键），然后按 Enter 或 Return 键。在 UNIX 系统中，包括 Mac OS X 系统中，文件结束符输入是用 Ctrl+D。

16

再探编译

编译器的一部分工作是寻找程序文本中的错误。编译器没有能力检查一个程序是否按照其作者的意图工作，但可以检查形式（form）上的错误。下面列出了一些最常见的编译器可以检查出的错误。

语法错误（syntax error）：程序员犯了 C++语言文法上的错误。下面程序展示了一些常见的语法错误；每条注释描述了下一行中语句存在的错误：

```
// 错误：main 的参数列表漏掉了
int main ( {
    // 错误：endl 后使用了冒号而非分号
    std::cout << "Read each file." << std::endl:
    // 错误：字符串字面常量的两侧漏掉了引号
    std::cout << Update master. << std::endl;
    // 错误：漏掉了第二个输出运算符
    std::cout << "Write new master." std::endl;
    // 错误：return 语句漏掉了分号
    return 0
}
```

类型错误（type error）：C++中每个数据项都有其类型。例如，10 的类型是 int（或者更通俗地说，“10 是一个 int 型数据”）。单词"hello"，包括两侧的双引号标记，则是一个字符串字面值常量。一个类型错误的例子是，向一个期望参数为 int 的函数传递了一个字符串字面值常量。

声明错误（declaration error）：C++程序中的每个名字都要先声明后使用。名字声明失败通常会导致一条错误信息。两种常见的声明错误是：对来自标准库的名字忘记使用 std::、标识符名字拼写错误：

```
#include <iostream>
int main()
{
    int v1 = 0, v2 = 0;
    std::cin >> v >> v2; // 错误：使用了"v"而非"v1"
    // 错误：cout 未定义；应该是 std::cout
    cout << v1 + v2 << std::endl;
    return 0;
}
```

错误信息通常包含一个行号和一条简短描述，描述了编译器认为的我们所犯的错误。按照报告的顺序来逐个修正错误，是一种好习惯。因为一个单个错误常常会具有传递效应，导致编译器在其后报告比实际数量多得多的错误信息。另一个好习惯是在每修

正一个错误后就立即重新编译代码，或者最多是修正了一小部分明显的错误后就重新编译。这就是所谓的“编辑-编译-调试”（edit-compile-debug）周期。

1.4.3 节练习

练习 1.16：编写程序，从 cin 读取一组数，输出其和。

1.4.4 if 语句

与大多数语言一样，C++也提供了 **if 语句**来支持条件执行。我们可以用 if 语句写一个程序，来统计在输入中每个值连续出现了多少次：

```
#include <iostream>
int main()
{
    // currVal 是我们正在统计的数；我们将读入的新值存入 val
    int currVal = 0, val = 0;
    // 读取第一个数，并确保确实有数据可以处理
    if (std::cin >> currVal) {
        int cnt = 1;                    // 保存我们正在处理的当前值的个数
        while (std::cin >> val) {       // 读取剩余的数
            if (val == currVal)         // 如果值相同
                ++cnt;                  // 将 cnt 加 1
            else {                      // 否则，打印前一个值的个数
                std::cout << currVal << " occurs "
                          << cnt << " times" << std::endl;
                currVal = val;          // 记住新值
                cnt = 1;                // 重置计数器
            }
        } // while 循环在这里结束
        // 记住打印文件中最后一个值的个数
        std::cout << currVal << " occurs "
                  << cnt << " times" << std::endl;
    } // 最外层的 if 语句在这里结束
    return 0;
}
```

如果我们输入如下内容：

```
42 42 42 42 42 55 55 62 100 100 100
```

则输出应该是：

```
42 occurs 5 times
55 occurs 2 times
62 occurs 1 times
100 occurs 3 times
```

有了之前多个程序的基础，你对这个程序中的大部分代码应该比较熟悉了。程序以两个变量 val 和 currVal 的定义开始：currVal 记录我们正在统计出现次数的那个数；val 则保存从输入读取的每个数。与之前的程序相比，新的内容就是两个 if 语句。第一条 if 语句

```
18      if (std::cin >> currVal) {
            // ...
        } //最外层的 if 语句在这里结束
```

保证输入不为空。与 while 语句类似，if 也对一个条件进行求值。第一条 if 语句的条件是读取一个数值存入 currVal 中。如果读取成功，则条件为真，我们继续执行条件之后的语句块。该语句块以左花括号开始，以 return 语句之前的右花括号结束。

如果需要统计出现次数的值，我们就定义 cnt，用来统计每个数值连续出现的次数。与上一小节的程序类似，我们用一个 while 循环反复从标准输入读取整数。

while 的循环体是一个语句块，它包含了第二条 if 语句：

```
if (val == currVal)                 // 如果值相同
    ++cnt;                          // 将 cnt 加 1
else {                              // 否则，打印前一个值的个数
    std::cout << currVal << " occurs "
              << cnt << " times" << std::endl;
    currVal = val;                  // 记住新值
    cnt = 1;                        // 重置计数器
}
```

这条 if 语句中的条件使用了**相等运算符**（==）来检测 val 是否等于 currVal。如果是，我们执行紧跟在条件之后的语句。这条语句将 cnt 增加 1，表明我们再次看到了 currVal。

如果条件为假，即 val 不等于 currVal，则执行 else 之后的语句。这条语句是一个由一条输出语句和两条赋值语句组成的语句块。输出语句打印我们刚刚统计完的值的出现次数。赋值语句将 cnt 重置为 1，将 currVal 重置为刚刚读入的值 val。

C++用=进行赋值，用==作为相等运算符。两个运算符都可以出现在条件中。一个常见的错误是想在条件中使用==（相等判断），却误用了=。

1.4.4 节练习

练习 1.17：如果输入的所有值都是相等的，本节的程序会输出什么？如果没有重复值，输出又会是怎样的？

练习 1.18：编译并运行本节的程序，给它输入全都相等的值。再次运行程序，输入没有重复的值。

练习 1.19：修改你为 1.4.1 节练习 1.10（第 11 页）所编写的程序（打印一个范围内的数），使其能处理用户输入的第一个数比第二个数小的情况。