第 17 章　处　理　流

到目前为止，我们一直使用 cout 将数据写到屏幕，使用 cin 从键盘读取数据，而对它们的工作原理没有全面的了解。

本章介绍以下内容：

- 什么是流？如何使用它们？
- 如何使用流来管理输入和输出？
- 如何使用流来读写文件？

17.1　流　概　述

C++没有定义如何将数据写入屏幕或文件，也没有定义如何将数据读入程序。然而，使用 C++编写程序时，这些是不可缺少的部分，标准 C++库包含用来方便输入和输出（I/O）的 iostream 库。

将输入和输出同语言分开并使用库来处理输入和输出的优点是，更容易使语言独立于平台。也就是说，可以在 PC 上编写 C++程序，然后在 Sun 工作站上重新编译并运行它们；或者在 Linux 上重新编译使用 Windows 编译器创建的代码，然后运行它。编译器厂商只需提供正确的库便万事大吉。至少从理论上说是这样的。

注意：库是一组扩展名为.obj 的文件，可将它们链接到程序以提供额外的功能。这是最基本的代码重用形式，从古老的程序员将 0 和 1 凿刻到洞穴壁上一直沿用至今。

当前，除对平面文件输入外，流对 C++编程来说不那么重要了。C++程序已经发展到使用操作系统或编译器厂商提供的图形用户界面（GUI）来同屏幕、文件和用户交互。这包括 Windows 库、X Windows 库以及 Borland 的 Windows 和 X Windows 用户界面抽象。由于这些库是专门针对操作系统的，并非 C++标准的组成部分，因此本书不讨论它们。

由于流是 C++标准的组成部分，因此本章将讨论它们。另外，为理解输入和输出的内部工作原理，最好理解流。然而，读者还应对操作系统或厂商提供的 GUI 库有大致的了解。

17.1.1　数据流的封装

文本输入和输出可使用 iostream 类来完成。Iostream 将数据流视为一个个字节接一个字节的流。如果流的目标是文件或控制台屏幕，数据源通常是程序的一部分。如果流的方向与此相反，数据可以来自键盘或磁盘文件并流入到数据变量中。

流的主要目标是，将从磁盘读取文件或将输入写入控制台屏幕的问题封装起来。创建流后，程序就可以使用它，流将负责处理所有的细节。图 17.1 说明了这种基本思想。

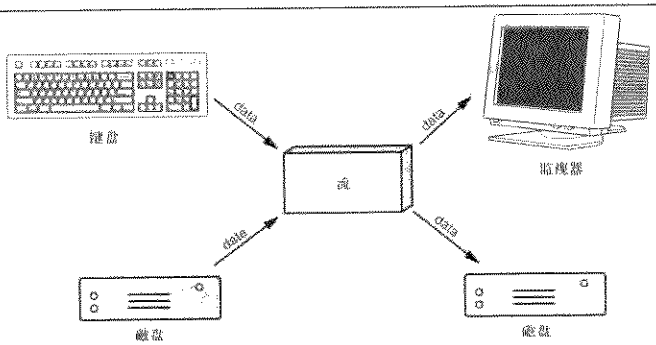

图 17.1　通过流进行封装

17.1.2　理解缓冲技术

将数据写入磁盘（和屏幕）是非常“昂贵”的。相对而言，将数据写入磁盘或从磁盘读取数据都要花很长的时间，读写磁盘可能妨碍程序的执行。为解决这个问题，流提供了缓冲技术。使用缓冲技术时，数据被写入到流中，而不立刻写入到磁盘中。不断地填充流的缓冲区，当缓冲区填满后，一次性将其中所有的数据写入磁盘。

这类似于不断向水箱里注水，但水不从底部流出来。图 17.2 说明了这个概念。

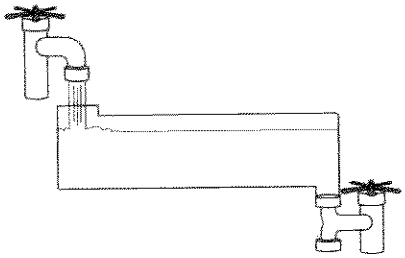

图 17.2　填充缓冲区

当水（数据）达到水箱顶部时，阀门打开，所有的水喷涌而出，如图 17.3 所示。

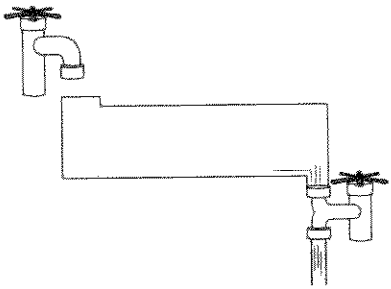

图 17.3　清空缓冲区

清空缓冲区后，底部阀门关上，顶部阀门打开；然后更多的水流入缓冲水箱，如图 17.4 所示。

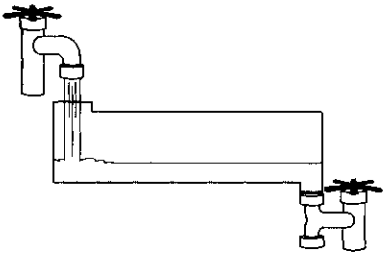

图 17.4 重新填充缓冲区

偶尔也需要在水箱未注满前就放掉水箱中的水。这被称为刷新缓冲区，如图 17.5 所示。

使用缓冲技术的危险之一是，如果数据下次输入/输出之前，没有刷新缓冲区，可能导致程序崩溃。在这种情况下，可能丢失数据。

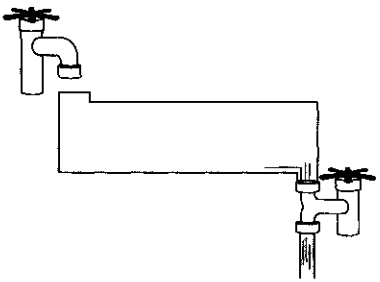

图 17.5 刷新缓冲区

17.2 流和缓冲区

正如读者可能预期到的，C++从面向对象的角度来实现流和缓冲区。它使用一系列的类和对象来完成这项任务：

- streambuf 类管理缓冲区，其成员函数提供了填充、清空、刷新和处理缓冲区的其他功能。
- ios 类是输入和输出流类的基类，它有一个成员变量为 streambuf 对象。
- istream 和 ostream 类是从 ios 类派生而来的，用于专门管理输入和输出行为。
- iostream 是从 istream 和 ostream 类派生而来的，提供了向屏幕写入数据的输入和输出方法。
- fstream 类提供了文件输入和输出功能。

17.3 标准 I/O 对象

包含 iostream 类的 C++程序启动时，将创建并初始化 4 个对象：

- cin：处理来自标准输入设备（键盘）的输入。
- cout：处理到标准输出设备（控制台屏幕）的输出。

- cerr：处理到标准错误设备（控制台屏幕）的非缓冲输出。由于这是非缓冲的，因此发送到 cerr 的任何数据都将立即写入标准错误设备，而不等到缓冲区填满或收到刷新命令。
- clog：处理输出到标准错误设备（控制台屏幕）的缓冲错误信息。这种输出通常被重定向到日志文件，这将在下一节介绍。

注意：编译器自动将 iostream 类库添加到程序中。要使用这些功能，只需在程序代码开头加入正确的 include 语句：

```
#include <iostream>
```

在前面的程序中，你一直在这样做。

17.4　重定向标准流

每种标准流（输入、输出和错误）都可以重定向到其他设备。标准错误流（ceer）经常被重定向到文件，而标准输入流（cin）和输出流（cout）可使用操作系统命令重定向到文件。

重定向指的是将输出（或输入）发送到不同于默认设备的地方。重定向是一种操作系统功能，而不是 iostream 库的功能。C++只提供了访问 4 种标准设备的途径，要重定向到其他设备，需要用户去完成。

DOS（Windows 命令提示符）和 UNIX 的重定向运算符是<（重定向输入）和>（重定向输出）。与 DOS（Windows 命令提示符）相比，UNIX 提供了更高级的重定向功能，但基本思想是相同的：将本来发送到屏幕的输出写入到文件或通过管道将其提供给另一个程序。同样，程序的输入也可从文件而不从键盘读取。

注意：管道技术（piping）指的是将一个程序的输出用作另一个程序的输入。

17.5　使用 cin 进行输入

全局对象 cin 负责输入，在程序中包含 iostream 后便可使用它。在以前的范例中，你使用重载的提取运算符（>>）将数据存储到程序变量中。这是如何工作的呢？你可能还记得，语法是这样的：

```
int someVariable;
cout << "Enter a number: ";
cin >> someVariable;
```

全局对象 cout 将在本章后面讨论。现在，将重点放在第 3 行：cin >> someVariable;。有关 cin 你能猜到些什么呢？

显然，它必须是全局对象，因为你没有在自己的代码中定义它。根据以前的运算符使用经验，你知道 cin 重载了提取运算符（>>）：将存放在 cin 缓冲区中的数据写入到局部变量 someVarialbe 中。

读者可能还不知道的是，cin 为重载了接受各种参数的提取运算符，包括 int&、short&、long&、double&、float&、char&、char*等。遇到代码 cin >> someVariable;时，编译器将判断 someVariable 的类型。在上面的范例中，someVariable 是一个 int 变量，因此调用下面的函数：

```
istream & operator>> (int &)
```

注意，由于参数是按引用传递的，因此提取运算符能够对原始变量进行操作。程序清单 17.1 演示了 cin 的用法。

程序清单 17.1　cin 能处理不同的数据类型

```
0:  //Listing 17.1 - character strings and cin
1:
2:  #include <iostream>
```

```
 3: using namespace std;
 4:
 5: int main()
 6: {
 7:    int myInt;
 8:    long myLong;
 9:    double myDouble;
10:    float myFloat;
11:    unsigned int myUnsigned;
12:
13:    cout << "Int: ";
14:    cin >> myInt;
15:    cout << "Long: ";
16:    cin >> myLong;
17:    cout << "Double: ";
18:    cin >> myDouble;
19:    cout << "Float: ";
20:    cin >> myFloat;
21:    cout << "Unsigned: ";
22:    cin >> myUnsigned;
23:
24:    cout << "\n\nInt:\t" << myInt << endl;
25:    cout << "Long:\t" << myLong << endl;
26:    cout << "Double:\t" << myDouble << endl;
27:    cout << "Float:\t" << myFloat << endl;
28:    cout << "Unsigned:\t" << myUnsigned << endl;
29:    return 0;
30: }
```

输出：

```
Int: 2
Long: 70000
Double: 987654321
Float: 3.33
Unsigned: 25

Int:    2
Long:   70000
Double: 9.87654e+008
Float:  3.33
Unsigned:       25
```

分析：

第 7～11 行声明了各种类型的变量；第 13～22 行提示用户输入这些变量的值，第 24～28 行使用 cout 打印结果。

输出结果表明，值被输入到了“正确”的变量类型中，程序按期望的那样运行。

17.5.1 输入字符串

cin 还能够处理字符指针（char*）参数；因此，可创建一个字符缓冲区，并使用 cin 来填充它。例如，可以编写这样的代码：

```
char YourName[50]
```

```
cout << "Enter your name: ";
cin >> YourName;
```

如果输入 Jesse，数组 YourName 将包含字符 J、e、s、s、e 和\0。最后一个为空字符；cin 自动在字符串末尾加上一个空字符，因此缓冲区必须有足够的空间容纳整个字符串和空字符。对于 cin 对象来说，空字符表示字符串结束。

17.5.2　字符串的问题

使用 cin 成功完成上述操作后，当你试图输入全名时可能感到惊讶：cin 将空格视为分隔符，因此无法读取全名。遇到空格或换行符后，cin 认为输入结束；如果输入的是字符串，它将加上一个空字符。程序清单 17.2 演示了这种问题。

程序清单 17.2　试图向 cin 写入多个单词

```
 0: //Listing 17.2 - character strings and cin
 1:
 2: #include <iostream>
 3:
 4: int main()
 5: {
 6:    char YourName[50];
 7:    std::cout << "Your first name: ";
 8:    std::cin >> YourName;
 9:    std::cout << "Here it is: " << YourName << std::endl;
10:    std::cout << "Your entire name: ";
11:    std::cin >> YourName;
12:    std::cout << "Here it is: " << YourName << std::endl;
13:    return 0;
14: }
```

输出：

```
Your first name: Jesse
Here it is: Jesse
Your entire name: Jesse Liberty
Here it is: Jesse
```

分析：

第 6 行创建了一个名为 YourName 的字符数组，用于存储用户的输入。第 7 行提示用户输入名，从输出可知，名被正确存储。

第 10 行再次提示用户输入全名。cin 读取输入，遇到姓和名之间的空格时，在第一个词后加上一个空字符并结束输入。这显然不是我们的本意。

要理解 cin 为何这样做，请看程序清单 17.3，它演示了多个变量的输入。

程序清单 17.3　输入多个变量

```
 0: //Listing 17.3 - character strings and cin
 1:
 2: #include <iostream>
 3: using namespace std;
 4:
 5: int main()
 6: {
```

```
 7:    int myInt;
 8:    long myLong;
 9:    double myDouble;
10:    float myFloat;
11:    unsigned int myUnsigned;
12:    char myWord[50];
13:
14:    cout << "Int: ";
15:    cin >> myInt;
16:    cout << "Long: ";
17:    cin >> myLong;
18:    cout << "Double: ";
19:    cin >> myDouble;
20:    cout << "Float: ";
21:    cin >> myFloat;
22:    cout << "Word: ";
23:    cin >> myWord;
24:    cout << "Unsigned: ";
25:    cin >> myUnsigned;
26:
27:    cout << "\n\nInt:\t" << myInt << endl;
28:    cout << "Long:\t" << myLong << endl;
29:    cout << "Double:\t" << myDouble << endl;
30:    cout << "Float:\t" << myFloat << endl;
31:    cout << "Word: \t" << myWord << endl;
32:    cout << "Unsigned:\t" << myUnsigned << endl;
33:
34:    cout << "\n\nInt, Long, Double, Float, Word, Unsigned: ";
35:    cin >> myInt >> myLong >> myDouble;
36:    cin >> myFloat >> myWord >> myUnsigned;
37:    cout << "\n\nInt:\t" << myInt << endl;
38:    cout << "Long:\t" << myLong << endl;
39:    cout << "Double:\t" << myDouble << endl;
40:    cout << "Float:\t" << myFloat << endl;
41:    cout << "Word: \t" << myWord << endl;
42:    cout << "Unsigned:\t" << myUnsigned << endl;
43:
44:    return 0;
45: }
```

输出：

```
Int: 2
Long: 30303
Double: 393939397834
Float: 3.33
Word: Hallo
Unsigned: 85

Int:    2
Long:   30303
Double: 3.93939e+011
Float:  3.33
```

```
Word:   Hello
Unsigned:       85

Int, Long, Double, Float, Word, Unsigned: 3 304938 393847473 6.66 bye -2

Int:    3
Long:   304938
Double: 3.93847e+008
Float:  6.66
Word:   bye
Unsigned: 4294967294
```

分析：

这里也创建了多个变量，这次包括一个 char 数组。提示用户输入，并正确地打印输出。

第 34 行提示用户一次性提供全部输入，然后输入的每个“单词”被赋给相应的变量。为实现这种多重赋值，cin 必须将输入的每个单词视为每个变量的完整输入。如果 cin 认为整个输入是变量输入的一部分，将不能实现这种级联输入。

注意，第 42 行要求的最后一个对象是无符号整数，但用户输入的是–2。由于 cin 认为它显示的是一个无符号整数，因此按无符号整数计算–2 的位模式，这样使用 cout 输出时，显示的是 4294967294。无符号值 4294967294 与有符号值–2 的位模式完全相同。

本章后面将讨论如何将包含多个单词的字符串输入到缓冲区。现在的问题是：提取运算符如何处理这种级联方式？

17.5.3　>>的返回值

>>的返回值是一个 istream 对象引用。由于 cin 本身是 istream 对象，因此提取操作的返回值可用作下一次提取操作的输入。

```
int varOne, varTwo, varThree;
cout << "Enter three numbers: "
cin >> varOne >> varTwo >> varThree;
```

对于代码 cin >> VarOne >> varTwo >> varThree;，首先执行的提取操作为 cin >> VarOne，其返回值是另一个 istream 对象，该对象的提取运算符将变量 varTwo 作为参数，就像代码为：

```
((cin >> varOne) >> varTwo) >> varThree;
```

后面讨论 cout 时将再次使用这种技术。

17.6　cin 的其他成员函数

除了重载运算符>>外，cin 还有很多其他成员函数。它们用于对输入进行细致的控制，让你能够：

- 读取单个字符；
- 读取字符串；
- 忽略输入；
- 查看缓冲区中的下一个字符；
- 将数据放回缓冲区。

17.6.1　单字符输入

运算符>>接受一个字符引用作为参数，可用于从标准输入中读取单个字符。成员函数 get()也可用于获取

单个字符，且有两种调用方式：不提供任何参数并使用其返回值；使用一个字符引用参数来调用。

1. 使用不接受任何参数的 get( )

get( )的第一种形式不接受任何参数，它返回找到的字符值，如果达到文件末尾则返回 EOF（文件末尾）。不接受任何参数的 get( )不常用。

这种形式的 get( )不能像 cin 那样用于级联多次输入，因为其返回值不是 iostream 对象。所以下面的语句是非法的：

```
cin.get() >>myVarOne >> myVarTwo; // illegal
```

cin.get( )>>myVarOne 的返回值是一个整数，而不是 iostream 对象。

程序清单 17.4 演示了不接受参数的 get( )的常见用法。

程序清单 17.4　使用不接受参数的 get( )

```
 0:  // Listing 17.4 - Using get() with no parameters
 1:
 2:  #include <iostream>
 3:
 4:  int main()
 5:  {
 6:     char ch;
 7:     while ( (ch = std::cin.get()) != EOF)
 8:      {
 9:        std::cout << "ch: " << ch << std::endl;
10:      }
11:     std::cout << "\nDone!\n";
12:     return 0;
13:  }
```

提示：要退出该程序，必须通过键盘输入文件末尾标记。为此，在 DOS 计算机上按 Ctrl+Z，在 UNIX 上按 Ctrl+D。

输出：

```
Hello
ch: H
ch: e
ch: l
ch: l
ch: o
ch:

Wor_d
ch: W
ch: o
ch: r
ch: l
ch: d
ch:

^Z (ctrl-z)

Done!
```

分析：

第 6 行声明了一个局部字符变量 ch。while 循环将 cin.get( )读取的输入赋给 ch，如果该字符不是 EOF，就打印出它。

然而，输出被缓冲，直到遇到换行符。遇到 EOF（在 DOS 机器上按 Ctrl+Z 或在 UNIX 机器上按 Ctrl+D）后退出循环。

注意，并非每种 istream 实现都支持这种版本的 get( )，虽然它现在是 ANSI/ISO 标准的组成部分。

2. 使用接受字符引用参数的 get( )

用字符变量作为参数来调用 get( )时，将把输入流中的下一个字符赋给该字符变量。返回值是一个 iostream 对象，因此这种形式的 get( )可以级联，如程序清单 17.5 所示。

程序清单 17.5　使用接受参数的 get( )

```
0:  // Listing 17.5 - Using get() with parameters
1:
2:  #include <iostream>
3:
4:  int main()
5:  {
6:     char a, b, c;
7:
8:     std::cout << "Enter three letters: ";
9:
10:    std::cin.get(a).get(b).get(c);
11:
12:    std::cout << "a: " << a << "\nb: ";
13:    std::cout << b << "\nc: " << c << std::endl;
14:    return 0;
15: }
```

输出：

```
Enter three letters: one
a: o
b: n
c: e
```

分析：

第 6 行创建了 3 个字符变量：a、b 和 c。第 10 行以级联方式调用了 cin.get( )3 次。首先调用 cin.get(a)，将第一个字符赋给 a 并返回 cin。这样接下来将调用 cin.get(b)，将下一个字符赋给 b。最后，调用 cin.get(c)，将第 3 个字符赋给 c。

由于 cin.get(a)的结果为 cin，因此可以这样编写代码：

```
cin.get(a) >> b;
```

在这种情况下，cin.get(a)的结果为 cin，因此第二部分为 cin>>b;。

应该：

需要跳过空白时，应使用提取运算符（>>）。

需要检查包括空白在内的每个字符时应使用接受一个字符参数的 get( )。

不应该：

如果不清楚要做什么，不要级联 cin 语句来读取多个输入。使用多条 cin 语句更合适，它比级联 cin 语句

更容易理解。

17.6.2 从标准输入读取字符串

要将输入存储到字符数组中，可使用提取运算符（>>）、成员函数 get( )的第三个版本或 getline( )。

get( )的第三个版本接受 3 个参数：

```
get( pCharArray, StreamSize, TermChar );
```

第一个参数（pCharArray）是字符数组指针，第二个参数（StreamSize）是要读取的最大字符数加 1，第三个参数（TermChar）是结束字符。

如果第二个参数为 20，get( )将读取 19 个字符，然后在存储在第一个参数中的字符串末尾加上空字符。第三个参数（结束字符）默认为换行符（'\n'）。如果 get( )在读取最大字符数之前遇到结束字符，将添加空字符，并将结束字符留在缓冲区中。

程序清单 17.6 演示了这种格式的 get( )的用法。

程序清单 17.6 使用接受字符数组作为参数的 get( )

```
0:  // Listing 17.6 - Using get() with a character array
1:
2:  #include <iostream>
3:  using namespace std;
4:
5:  int main()
6:  {
7:     char stringOne[256];
8:     char stringTwo[256];
9:
10:    cout << "Enter string one: ";
11:    cin.get(stringOne,256);
12:    cout << "stringOne: " << stringOne << endl;
13:
14:    cout << "Enter string two: ";
15:    cin >> stringTwo;
16:    cout << "StringTwo: " << stringTwo << endl;
17:    return 0;
18: }
```

输出：

```
Enter string one: Now is the time
stringOne: Now is the time
Enter string two: For all good
StringTwo: For
```

分析：

第 7 行和第 8 行创建了两个字符数组。第 10 行提示用户输入一个字符串，第 11 行调用了 cin.get( )。第一个参数是要填充的缓冲区，第二个参数是 get( )将读取的最大字符数加 1（多出来的那个位置用于存储空字符（'\0'）），第三个参数默认为换行符。

用户输入“Now is the time”。由于用户以换行符结束该短语，因此该短语被存储到 stringOne 中，并在末尾加上一个空字符。

第14 行提示用户再输入一个字符串，这次使用提取运算符（>>）来读取。由于提取运算符读取第一个空白字符前的所有字符，因此第二个字符串中存储的是“For”和空字符，这显然不是我们的本意。

接受 3 个参数的 get( )非常适合用来读取字符串，然而这并非惟一的解决方案。解决这种问题的另一种方

法是使用 getline( )，如程序清单 17.7 所示。

程序清单 17.7　使用 getline( )

```
// Listing 17.7 - Using getline()

#include <iostream>
using namespace std;

int main()
{
   char stringOne[256];
   char stringTwo[256];
   char stringThree[256];

   cout << "Enter string one: ";
   cin.getline(stringOne,256);
   cout << "stringOne: " << stringOne << endl;

   cout << "Enter string two: ";
   cin >> stringTwo;
   cout << "stringTwo: " << stringTwo << endl;

   cout << "Enter string three: ";
   cin.getline(stringThree,256);
   cout << "stringThree: " << stringThree << endl;
   return 0;
}
```

输出：

```
Enter string one: one two three
stringOne: one two three
Enter string two: four five six
stringTwo: four
Enter string three: stringThree: five six
```

分析：

这个范例值得仔细研究，其中可能有一些令人意外的地方。第 7～9 行声明了 3 个字符数组。

第 11 行提示用户输入一个字符串，然后使用 getline( )读取它。和 get( )一样，getline( )接受一个缓冲区和最大字符数作为参数；但与 get( )不同的是，它读取换行符并将其丢弃。get( )不丢弃换行符，将其留在输入缓冲区中。

第 15 行再次提示用户输入一个字符串，这次使用提取运算符来读取它。从输出可知，用户输入了“four five six”，但只有第一个单词（four）被存储到 stringTwo 中。然后程序第二次提示用户输入一个字符串（显示“Enter string three”），并再次调用 getline( )来读取它。由于“five six”还留在输入缓冲区中，因此 getline( )读取它，并在遇到换行符后结束，第 21 行打印存储在 stringThree 中的字符串。

用户根本没有机会输入第 3 个字符，因为原来留在输入缓冲区的数据已经满足了第二次读取要求。

第 16 行对 cin 的调用没有读取输入缓冲区中的全部内容。第 16 行的提取运算符（>>）读取到第一个空格为止，并将第一个单词存储到字符数组中。

get( ) 和 getline( )

成员函数 get( )被重载。在第一个版本中，它不接受任何参数并返回读取的字符的值。在第二个版本中，

它接受一个字符引用参数，并按引用返回 istream 对象。

在第三个也最后一个版本中，get( )接受一个字符数组、要读取的最大字符数和结束字符（默认为换行符）作为参数。这种版本的 get( )将字符读入数组，直到读入的字符数比指定的最大字符数少 1 或遇到结束字符。遇到结束字符后，get( )将该字符留在输入缓冲区中并结束读取。

成员函数 getline( )也接受 3 个参数：要填充的缓冲区、要读取的最大字符数加 1 以及结束字符。getline( )函数使用这 3 个参数的方式和 get( )相同，只是它丢弃结束字符。

17.6.3 使用 cin.ignore( )

有时可能需要忽略行尾（EOL）或文件尾（EOF）之前的剩余字符，成员函数 ignore( )提供了这种功能。它接受两个参数：要忽略的最大字符数和结束字符。代码 ignore(80、'\n') 将最多忽略 80 个字符，直到遇到换行符。然后丢弃换行符，ignore( )语句结束。程序清单 17.8 演示了 ignore( )的用法。

程序清单 17.8 使用 ignore( )

```
// Listing 17.8 - Using ignore()
#include <iostream>
using namespace std;

int main()
{
      char stringOne[255];
      char stringTwo[255];

   cout << "Enter string one:";
   cin.get(stringOne,255);
   cout << "String one: " << stringOne << endl;

   cout << "Enter string two: ";
   cin.getline(stringTwo,255);
   cout << "String two: " << stringTwo << endl;

   cout << "\n\nNow try again...\n";

   cout << "Enter string one: ";
   cin.get(stringOne,255);
   cout << "String one: " << stringOne<< endl;

   cin.ignore(255, '\n');

   cout << "Enter string two: ";
   cin.getline(stringTwo,255);
   cout << "String Two: " << stringTwo<< endl;
   return 0;
}
```

输出：

```
Enter string one: once upon a time
String one: once upon a time
Enter string two: String two:


Now try again...
```

```
Enter string one: once upon a time
String one: once upon a time
Enter string two: there was a
String Two: there was a
```

分析：

第 6 和第 7 行创建了两个字符数组。第 9 行提示用户输入，用户输入“once upon a time”后按回车键。第 10 行使用 get( )来读取该字符串。get( )将换行符之前的内容存储到 stringOne 中，并将换行符留在输入缓冲区中。

第 13 行再次提示用户输入，但第 14 行使用 getline( )来读取缓冲区中换行符之前的内容。由于之前调用 get( )时将一个换行符留在了缓冲区中，因此用户还未输入任何内容之前，第 14 行已经结束。

第 19 行再次提示用户输入，用户的输入与第一次完全相同。然而这次使用 ignore( )来删除换行符以清空输入流（第 23 行）。这样，当程序执行到第 26 行的 getline( )调用时，输入缓冲区是空的，因此用户能够输入下一行字符串。

17.6.4　查看和插入字符：peek( )和 putback( )

输入对象 cin 有两个使用起来非常方便的方法：peek( )和 putback( )。peek( )查看但不提取下一个字符，putback( )将一个字符插入到输入流中。程序清单 17.9 演示了它们的用法。

程序清单 17.9　使用 peek( )和 putback( )

```
 0:  // Listing 17.9 - Using peek() and putback()
 1:  #include <iostream>
 2:  using namespace std;
 3:
 4:  int main()
 5:  {
 6:     char ch;
 7:     cout << "enter a phrase: ";
 8:     while ( cin.get(ch) != 0 )
 9:     {
10:        if (ch == '!')
11:           cin.putback('$');
12:        else
13:           cout << ch;
14:        while (cin.peek() == '#')
15:           cin.ignore(1, '#');
16:     }
17:     return 0;
18:  }
```

输出：

```
enter a phrase: Now!is#the!time#for!fun#!
Now$isthe$timefor$fun$
```

分析：

第 6 行声明了一个字符变量 ch，第 7 行提示用户输入一条短语。该程序旨在将感叹号（!）替换为美元符号（$）并删除所有的英镑符号（#）。

遇到文件末尾字符（在 Windows 计算机上为 Ctrl + C，在其他操作系统上为 Ctrl + Z 或 Ctrl + D）之前，将不断执行第 8～16 行的循环（cin.get( )到达文件末尾后，将返回 0）。如果当前字符为感叹号，则将其丢弃，并将美元符号（$）插入输入缓冲区，下次循环将读取该符号。如果当前字符不是感叹号，则将其打印出来（第

13 行）。

第 14 行查看下一个字符，如果是英镑符号，则使用 ignore( )方法将其删除，如第 15 行所示。

虽然该程序并非处理这些问题的最有效方式（同时，如果第一个字符为#，该程序将发现不了），但它演示了这些方法的工作原理。

提示：peek( )和 putback( )通常用于分析字符串和其他数据，如编写编译器时。

17.7 使用 cout 进行输出

你使用过 cout 和重载的插入运算符（<<）将字符串、整数和其他数值写入到屏幕。还可以对数据进行格式化，对齐列以及以十进制和十六进制书写数值数据，本节介绍如何完成这种任务。

17.7.1 刷新输出

正如你知道的，endl 写入一个换行符并刷新输出缓冲区。endl 调用 cout 的成员函数 flush( )，后者输出被缓冲的所有数据。也可以直接调用 flush( )：

```
cout << flush();
```

需要清空输出缓冲区并将其中的内容写入屏幕时，该方法非常方便。

17.7.2 执行输出的函数

就像 get( )和 getline( )是提取运算符的补充一样，put( )和 write( )是插入运算符的补充。

1. 使用 put( )写入字符

函数 put( )用于将单个字符写入输出设备。put( )返回一个 ostream 引用，而 cout 是一个 ostream 对象，因此可以像级联插入运算符一样级联 put( )，如程序清单 17.10 所示。

程序清单 17.10 使用 put( )

```
0:  // Listing 17.10 - Using put()
1:
2:  #include <iostream>
3:
4:  int main()
5:  {
6:    std::cout.put('H').put('e'). put('l'). put('l'). put('o'). put('\n');
7:    return 0;
8:  }
```

输出：

```
Hello
```

注意：有些非标准编译器可能不支持上述打印代码。如果你的编译器不能打印单词 Hello，建议跳过该程序清单。

分析：

第 6 行的执行过程是这样的：cout.put('H') 将字母 H 写入屏幕并返回一个 cout 对象。这样，还未执行的代码如下：

```
cout.put('e').put('l').put('l').put('o').put('\n');
```

然后写入字母 e 并返回一个 cout 对象。这样，还未执行的代码如下：

```
cout.put('l').put('l').put('o').put('\n');
```

重复上述过程：将每个字母写入屏幕并返回 cout 对象。将最后一个字符（'\n'）写入屏幕后函数返回。

2. 使用 write( )写入更多字符

函数 write( )的工作原理和插入运算符（<<）相同，只是它接受一个指出最多写入多少个字符的参数：

```
cout.write(Text, Size)
```

write( )的第一个参数是要打印的文本，第二个参数（Size）指定要打印 Text 中的多少个字符。注意，Size 可能大于或小于 Text 的实际长度。如果大于，将输出内存中 Text 后面的值。程序清单 17.11 演示了 write( ) 的用法。

程序清单 17.11　使用 write( )

```
 0: // Listing 17.11 - Using write()
 1: #include <iostream>
 2: #include <string.h>
 3: using namespace std;
 4:
 5: int main()
 6: {
 7:    char One[] = "One if by land";
 8:
 9:    int fullLength = strlen(One);
10:    int tooShort = fullLength -4;
11:    int tooLong = fullLength + 6;
12:
13:    cout.write(One,fullLength) << endl;
14:    cout.write(One,tooShort) << endl;
15:    cout.write(One,tooLong) << endl;
16:    return 0;
17: }
```

输出：

```
One if by land
One if by
One if by land ?!
```

注意：在你的计算机上，最后一行的输出可能与此不同，因为访问了未被初始化的变量占用的内存。

分析：

该程序清单打印一个短语的内容，它每次打印的内容量不同。第 7 行创建了一个短语。第 9 行使用全局方法 strlen( )将 int 变量 fullLength 设置为短语的长度，该方法是通过第 2 行的编译指令包含进来的；tooShort 被设置为短语的长度减 4；tooLong 被设置为短语的长度加 6。

第 13 行使用 write( )打印整个短语。长度被设置为短语的实际长度，因此打印了正确的短语。

第 14 行再次打印，但输出表明，比整个短语少 4 个字符。

第 15 行再次打印，但这次让 write( )多写入 6 个字符。写入该短语后，继续写入后续内存中的 6 个字节。该内存中的内容是不确定的，因此读者的输出可能与前面列出的不同。

17.7.3　控制符、标记和格式化指令

输出流包含很多状态标记，用于指定计数方式（十进制或十六进制）、字段宽度和字段填充字符。状态标

记长 1 字节，其中的每一位都有特殊含义。用这种方式操纵位将在第 21 章讨论。每个 ostream 标记都可以用成员函数和控制符来设置。

1. 使用 cout.width()

输出的默认宽度为刚好能够容纳输出缓冲区中的数字、字符或字符串。可以使用 width()来修改默认宽度设置。

width()是成员函数，必须通过 cout 对象来调用。它只修改下一个输出字段的宽度，然后字段宽度设置立刻恢复到默认值。程序清单 17.12 演示了 width()的用法。

程序清单 17.12　调整输出的宽度

```
 0:  // Listing 17.12 - Adjusting the width of output
 1:  #include <iostream>
 2:  using namespace std;
 3:
 4:  int main()
 5:  {
 6:     cout << "Start >";
 7:     cout.width(25);
 8:     cout << 123 << "< End\n";
 9:
10:     cout << "Start >";
11:     cout.width(25);
12:     cout << 123<< "< Next >";
13:     cout << 456 << "< End\n";
14:
15:     cout << "Start >";
16:     cout.width(4);
17:     cout << 123456 << "< End\n";
18:
19:     return 0;
20:  }
```

输出：

```
Start >                    123< End
Start >                    123< Next >456< End
Start >123456< End
```

分析：

第一次输出（第 6～8 行）在宽度被设置为 25（第 7 行）的字段中打印数字 123，如第 1 行输出所示。

第二次输出首先在宽度被设置为 25 的字段中打印 123，然后打印 456。注意，456 被打印在一个宽度刚刚好的字段中；正如前面指出的，width()的作用仅持续到下一次输出。

最后一次输出表明，将宽度设置为小于输出内容与设置为刚刚好等效。当宽度被设置为比输出内容小时，不会导致输出被截短。

2. 设置填充字符

通常情况下，cout 用空格填充字段，如前面所示。有时可能想用其他字符（如星号）来填充。为此，可调用 fill()，并将要用来填充的字符作为参数传递给它，如程序清单 17.13 所示。

程序清单 17.13　使用 fill()

```
 0:  // Listing 17.13 - fill()
```

```
 1:
 2:  #include <iostream>
 3:  using namespace std;
 4:
 5:  int main()
 6:  {
 7:     cout << "Start >";
 8:     cout.width(25);
 9:     cout << 123 << "< End\n";
10:
11:     cout << "Start >";
12:     cout.width(25);
13:     cout.fill('*');
14:     cout << 123 << "< End\n";
15:
16:     cout << "Start >";
17:     cout.width(25);
18:     cout << 456 << "< End\n";
19:
20:     return 0;
21:  }
```

输出：

```
Start >                  123< End
Start >******************123< End
Start >******************456< End
```

分析：

第 7～9 行的功能与前一个范例中完全相同：在一个宽度为 25 的字段中打印 123。第 11～14 行重复这种操作，但第 13 行将填充字符设置为星号，如输出所示。需要注意的是，不同于函数 width()那样只对下一次输出有效，fill()设置的填充字符将一直有效，直到你修改为止。第 16～18 行的输出证明了这一点。

3. 管理输出状态：设置标记

当对象的部分或全部数据成员表示可在程序执行期间修改的条件时，该对象被认为是有状态的。例如，可以设置是否显示末尾的零（以免 20.00 被截短为 20）。

lostream 对象使用标记来记录其状态。可以调用 setf()并传递一个预定义的枚举常量来设置这些标记。例如，要显示末尾的零，可调用 setf(ios::showpoint)。

这些枚举常量的作用域为 iostream 类（ios），因此将其作为 setf()的参数时，需要采用全限定方式 ios::flagname，如 ios::showpoint。表 17.1 列出了一些可使用的标记。要使用这些标记，必须在程序中包含 iostream。另外，要使用那些需要参数的标记，还必须包含 iomanip。

表 17.1　　一些 iostream 设置标记

标　记	用　途
showpoint	根据精度设置显示小数点和末尾的零
showpos	在正数前面加上正号（+）
left	让输出左对齐
right	让输出右对齐
internal	让符号左对齐并让数值右对齐
scientific	用科学表示法显示浮点数

续表

标　记	用　途
fixed	以小数点表示法显示浮点数
showbase	在十六进制数前加上 0x，指出这是十六进制值
Uppercase	在十六进制和科学表示法中使用大写字母
dec	以十进制方式显示
oct	以八进制方式显示
hex	以十六进制方式显示

表 17.1 中的标记也可与插入运算符 一起使用。程序清单 17.14 演示了这些设置，还使用了设置宽度的 setw 控制符，该控制符也可与插入运算符一起使用。

程序清单 17.14　使用 setf

```
0:  // Listing 17.14 - Using setf
1:  #include <iostream>
2:  #include <iomanip>
3:  using namespace std;
4:
5:  int main()
6:  {
7:     const int number = 185;
8:     cout << "The number is " << number << endl;
9:
10:    cout << "The number is " << hex <<  number << endl;
11:
12:    cout.setf(ios::showbase);
13:    cout << "The number is " << hex <<  number << endl;
14:
15:    cout << "The number is " ;
16:    cout.width(10);
17:    cout << hex << number << endl;
18:
19:    cout << "The number is " ;
20:    cout.width(10);
21:    cout.setf(ios::left);
22:    cout << hex << number << endl;
23:
24:    cout << "The number is " ;
25:    cout.width(10);
26:    cout.setf(ios::internal);
27:    cout << hex << number << endl;
28:
29:    cout << "The number is " << setw(10) << hex << number << endl;
30:    return 0;
31: }
```

输出：

```
The number is 185
The number is b9
The number is 0xb9
The number is        0xb9
```

```
The number is 0xb9
The number is 0x       b9
The number is:0x       b9
```

分析：

第 7 行将 int 常量 number 初始化为 185，第 8 行按正常方式显示这个值。

第 10 行再次显示这个值，但使用了控制符 hex，将这个值以十六进制方式显示为 b9。

注意：在十六进制中，b 表示 11；11*16 = 176，176 + 9 = 185。

第 12 行设置标记 showbase，这导致在所有十六进制数前加上 0x，如输出所示。

第 16 行将宽度设置为 10，并采用默认的右对齐。第 20 行再次将宽度设置为 10，但设置为左对齐，因此数字被打印在最左边。

第 25 行再次将宽度设置为 10，但采用居中对齐。因此 0x 被打印在最左边，而 b9 被打印在最右边。

最后，第 29 行结合使用插入运算符和 setw( )，将宽度设置为 10，并再次打印这个值。

从这个程序清单可知，与插入运算符一起使用时，无需对标记进行全限定：要采用十六进制表示只需使用 hex 即可；而用作函数 setf( )的参数时，必须对标记进行全限定：要采用十六进制表示必须传递 ios::hex。从第 17 和 21 行可以看到这种区别。

17.8　流和 printf( )函数之比较

大多数 C++程序还提供了标准 C 语言 I/O 库，其中包括函数 printf( )。虽然 printf( )在某些方面比 cout 使用起来更容易，但应尽量避免使用它。

printf( )没有提供类型安全性，很容易无意间命令它将整数当作字符显示或反过来。printf( )也不支持类，因此无法告诉它如何打印类数据，而必须将类成员逐个传递给 printf( )。

由于有很多使用 printf( )的代码，因此本节简要地介绍一下 printf( )的用法。然而，建议你在 C++程序中使用该函数。

要使用 printf( )，务必包含头文件 stdio.h。在最简单的情况下，printf( )将一个格式化字符串作为其第一个参数，并将一系列的值作为其他参数。

格式化字符串是用引号括起的文本和转换说明符。所有转换说明符都必须以百分号（%）打头。表 17.2 列出了常见的转换说明符。

表 17.2　　**常见的转换说明符**

说 明 符	用　于
%s	字符串
%d	整型
%l	长整型
%ld	双精度
%f	浮点数

每个转换说明符还可以以浮点数的方式指定宽度和精度，其中小数点左边的数字表示总宽度，小数点右边的数字表示浮点数的精度。因此 5%d 表示宽度为 5 位的整数，%15.5 表示宽度为 15 位，其中最后 5 位为小数部分的浮点数。程序清单 17.15 演示了 printf( )的各种用法。

程序清单 17.15　使用 printf( )进行打印

```
0:  //17.15 Printing with printf()
```

```
 1: #include <stdio.h>
 2:
 3: int main()
 4: {
 5:    printf("%s","hello world\n");
 6:
 7:    char *phrase = "Hello again!\n";
 8:    printf("%s",phrase);
 9:
10:    int x = 5;
11:    printf("%d\n",x);
12:
13:    char *phraseTwo = "Here's some value: ";
14:    char *phraseThree = " and also these: ";
15:    int y = 7, z = 35;
16:    long longVar = 98456;
17:    float floatVar = 8.8f;
18:
19:    printf("%s %d %d", phraseTwo, y, z);
20:    printf("%s %ld %f\n",phraseThree,longVar,floatVar);
21:
22:    char *phraseFour = "Formatted: ";
23:    printf("%s %5d %10d  %10.5f\n",phraseFour,y,z,floatVar);
24:
25:    return 0;
26: }
```

输出：

```
hello world
Hello again!
5
Here's some values: 7 35  and also these: 98456 8.800000
Formatted:     7       35     8.800000
```

分析：

第一条 printf( )语句（第 5 行）使用标准格式：用引号括起的转换说明符（这里为%s）和要插入到转换说明符中的值。

%s 表示这是一个字符串，这里为用引号括起的“hello world”。

第二条 printf( )语句（第 8 行）与第一条相同，但这次使用的 char 指针，而不是用引号括起的字符串。

第三条 printf( )语句（第 11 行）使用整数转换说明符（%d），值为 int 变量 x。第四条 printf( )语句（第 19 行）要复杂得多：同时打印 3 个值。首先是每个转换说明符，然后是用逗号分开的值。第 20 行与第 19 行类似，但使用的说明符和值不同。

最后，第 23 行使用了格式说明来指定宽度和精度。正如读者所看到的，这些使用起来比控制符更容易。

然而，正如前面指出的，这里的缺点在于没有类型检查，同时 printf( )也不能被声明为类的友元或成员函数。因此要打印类的各个成员数据，必须在参数列表中将每个存取器方法传递给 printf( )函数。

FAQ

能总结一下如何控制输出吗？

答：在 C++中，要格式化输出，需要结合使用特殊字符、输出控制符和标记。

可使用插入运算符将包含下述特殊字符的输出字符串传递给 cout：

\n：换行符；
\r：回车；
\t：制表符；
\\：反斜杠；
\ddd（八进制数）：ASCII 字符；
\a：振铃。
例如，下面的代码振铃、打印错误消息并移到下一个制表位：

```
cout << "\aAn error occurred\t";
```

控制符和 cout 运算符一起使用。要使用接受参数的控制符，必须在程序中包含 iomanip。
下面是不需要包含头文件 iomanip 就可以使用的控制符：
flush：刷新输出缓冲区；
endl：换行并刷新输出缓冲区；
oct：采用八进制；
dcc：采用十进制；
hex：采用十六进制。
下面是需要包含头文件 iomanip 才能使用的控制符：
setbase(base)：设置计数方式（0 为十进制，8 为八进制，10 为十进制，16 为十六进制）；
setw (width)：设置最小输出字段宽度；
setfill(ch)：指定了字段宽度时使用的填充字符；
setprecision (p)：设置浮点数的精度；
setiosflags (f)：设置一个或多个 ios 标记。
resetiosflags (f)：重置一个或多个 ios 标记。
例如，下面的代码将字段宽度设置为 12，将填充字符设置为“#”，将计数方式设置为十六进制，然后打印变量 x 的值，将一个换行符放入缓冲区并刷新缓冲区：

```
cout << setw(12) << setfill( '#' ) << hex << x << endl;
```

除 flush、endl 和 setw 外，所有控制符在被修改或程序结束前都一直有效。setw 在当前 cout 结束后恢复到默认值。
很多标记可用作 setiosflags 和 resetiosflags 控制符的参数，前面的表 17.1 列出了它们。
更详细的信息请参阅头文件 ios 和编译器文档。

17.9　文件输入和输出

在处理来自键盘或硬盘的数据以及输出到屏幕或硬盘的数据方面，流提供了统一的方法。不管在哪种情况下，都可以使用插入和提取运算符以及其他相关函数和控制符。要打开和关闭文件，需要创建 ifstream 和 ofstream 对象，这将在接下来的几节中介绍。

17.10　使用 ofstream

用于读写文件的对象被称为 ofstream 对象。这些对象是从前面一直在使用的 iostream 对象派生出来。
要开始写入文件，首先必须创建一个 ofstream 对象，然后将其与磁盘上的文件关联起来。要使用 ofstream 对象，必须在程序中包含头文件 fstream。

注意：由于 fstream 包含了 iostream，因此不需要再显式地包含 iostream。

17.10.1 条件状态

iostream 对象包含报告输入和输出状态的标记。可以使用布尔函数 eof()、bad()、fail()和 good()来检查这些标记。iostream 对象遇到文件末尾（EOF）时，函数 eof()返回 true；如果你试图进行非法操作，函数 bad()将返回 true；在 bad()返回 true 或操作失败时，函数 fail()将返回 true；最后，在其他 3 个函数都返回 false 时，函数 good()返回 true。

17.10.2 打开文件进行输入和输出

要使用文件，必须首先打开它。要使用 ofstream 打开文件 myfile.cpp，声明一个 ofstream 实例，并将该文件名作为参数传递给它：

```
ofstream fout("myfile.cpp");
```

上述代码打开该文件以便进行输出。打开该文件以便进行输入的方法完全相同，只是需要使用 ifstream 对象：

```
ifstream fin("myfile.cpp");
```

fout 和 fin 是你指定的名称。这里使用 fout 旨在说明它与 cout 相似，而使用 fin 旨在说明它与 cin 相似。也可以使用指出它们要访问的文件的名称。

稍后就将使用的一个重要文件流函数是 close()。你创建的每个文件流对象都打开一个文件，以便进行读、写或读写。完成读写后应关闭文件，这非常重要，它确保文件不会受损以及将缓冲区的数据写入到磁盘。

将流对象同文件关联起来后，可以像使用其他流对象一样使用它们。程序清单 17.16 演示了这一点。

程序清单 17.16 打开文件以便进行读写

```
 0: //Listing 17.16 Opening Files for Read and Write
 1: #include <fstream>
 2: #include <iostream>
 3: using namespace std;
 4:
 5: int main()
 6: {
 7:    char fileName[80];
 8:    char buffer[255];    // for user input
 9:    cout << "File name: ";
10:    cin >> fileName;
11:
12:    ofstream fout(fileName);  // open for writing
13:    fout << "This line written directly to the file...\n";
14:    cout << "Enter text for the file: ";
15:    cin.ignore(1, '\n');   //eat the newline after the file name
16:    cin.getline(buffer,255);  // get the user's input
17:    fout << buffer << "\n";   // and write it to the file
18:    fout.close();             // close the file, ready for reopen
19:
20:    ifstream fin(fileName);    // reopen for reading
21:    cout << "Here's the contents of the file:\n";
22:    char ch;
23:    while (fin.get(ch))
24:       cout << ch;
25:
26:    cout << "\n***End of file contents.***\n";
```

```
27:
28:    fin.close();          // always pays to be tidy
29:    return 0;
30: }
```

输出：

```
File name: test1
Enter text for the file: This text is written to the file!
Here's the contents of the file:
This line written directly to the file...
This text is written to the file!

***End of file contents.***
```

分析：

第 7 行创建了一个用于存储文件名的字符数组；第 8 行创建了一个用于存储用户输入的字符数组。第 9 行提示用户输入一个文件名，输入被存储到数组 fileName 中。第 12 行创建一个名为 fout 的 ofstream 对象，并将其与用户输入的文件名关联起来。这将打开该文件；如果该文件已经存在，将删除其内容。

第 13 行将一个文本字符串直接写入文件。第 14 行提示用户输入文件内容。第 15 行使用函数 ignore( )删除用户输入文件名时留下的换行符；第 16 行将用户输入的文件内容存储到数组 buffer 中。第 17 行将用户的输入和一个换行符写入文件，然后第 18 行关闭文件。

第 20 行重新打开该文件，但这次使用 ifstream 对象以输入模式打开。然后第 23 和 24 行以每次一个字符的方式读取文件的内容。

17.10.3　修改 ofstream 打开文件时的默认行为

打开文件时的默认行为是，如果文件不存在则创建它，如果文件已经存在则删除其内容。如果不想采用打开文件时的默认行为，可以给 ofstream 类的构造函数提供第二个参数。

该参数的合法取值包括：

- ios::app：附加到已有文件的末尾，而不是删除其内容。
- ios::ate：跳到文件末尾，但可以在文件的任何地方写入数据。
- ios::trunc：默认值。删除已有文件的内容。
- ios::nocreate：如果文件不存在，打开操作失败。
- ios::noreplace：如果文件已经存在，打开操作失败。

注意，app 是 append 的缩写；ate 是 at end 的缩写，而 trunc 是 truncate 的缩写。程序清单 17.17 重新打开程序清单 17.16 使用的文件并追加内容，演示了追加模式的用法。

程序清单 17.17　在文件末尾添加内容

```
 0: //Listing 17.17 Appending to the End of a File
 1: #include <fstream>
 2: #include <iostream>
 3: using namespace std;
 4:
 5: int main()   // returns 1 on error
 6: {
 7:    char fileName[80];
 8:    char buffer[255];
 9:    cout << "Please reenter the file name: ";
10:    cin >> fileName;
11:
```

```
12:    ifstream fin(fileName);
13:    if (fin)              // already exists?
14:    {
15:       cout << "Current file contents:\n";
16:       char ch;
17:       while (fin.get(ch))
18:          cout << ch;
19:       cout << "\n***End of file contents.***\n";
20:    }
21:    fin.close();
22:
23:    cout << "\nOpening " << fileName << " in append mode...\n";
24:
25:    ofstream fout(fileName,ios::app);
26:    if (!fout)
27:    {
28:       cout << "Unable to open " << fileName << " for appending.\n";
29:       return(1);
30:    }
31:
32:    cout << "\nEnter text for the file: ";
33:    cin.ignore(1, '\n');
34:    cin.getline(buffer,255);
35:    fout << buffer << "\n";
36:    fout.close();
37:
38:    fin.open(fileName);  // reassign existing fin object!
39:    if (!fin)
40:    {
41:       cout << "Unable to open " << fileName << " for reading.\n";
42:       return(1);
43:    }
44:    cout << "\nHere's the contents of the file:\n";
45:    char ch;
46:    while (fin.get(ch))
47:       cout << ch;
48:    cout << "\n***End of file contents.***\n";
49:    fin.close();
50:    return 0;
51: }
```

输出：

```
Please reenter the file name: test1
Current file contents:
This line written directly to the file...
This text is written to the file!

***End of file contents.***

Opening test1 in append mode...

Enter text for the file: More text for the file!
```

```
Here's the contents of the file:
This line written directly to the file...
This text is written to the file!
More text for the file!

***End of file contents.***
```

分析：

和前一个程序清单一样，第 9 和 10 行也提示用户输入文件名，但第 12 行创建了一个输入文件流对象。第 13 行对打开操作进行测试，如果文件已经存在，则第 15～19 行打印其内容。注意，if(fin)和 if(four.good( ))等效。

然后关闭该文件，第 25 行重新打开该文件，但使用追加模式。在这次（以及每次）打开文件后，都对文件进行测试以核实它被正确打开。注意，if(!fout)和测试 if(fout.fail( ))等效。如果文件没有打开，第 28 行打印一条错误消息，然后使用一条 return 语句结束程序。如果文件被成功打开，则提示用户输入文件内容，然后第 36 行再次关闭文件。

最后，和程序清单 17.16 一样，以读取模式再次打开文件，但这次不必再次声明 fin，而只是将相同的文件与其关联起来。同样，第 39 行对文件打开操作进行测试，如果一切正常，将文件内容打印到屏幕上，然后关闭文件。

应该：

每次打开文件时都应进行测试以核实文件被成功打开。

应重用已有的 ifstream 和 ofstream 对象。

使用完 fstream 对象后应关闭它们。

不应该：

不要企图关闭 cin 和 cout 或给它们重新赋值。

尽可能不要在 C++程序中使用 printf( )。

17.11　二进制文件和文本文件

有些操作系统（如 DOS）区分二进制文件和文本文件。文本文件将所有内容都保存为文本（读者可能猜到了），因此像 54325 这样的大数被保存为一串数字（'5'、'4'、'3'、'2'、'5'）。这样做效率很低，但优点是可以用诸如 DOS 命令和 Windows 命令行程序 type 等简单程序来阅读文本。

为帮助文件系统区分二进制文件和文本文件，C++提供了标记 ios:binary。在很多系统上该标记都被忽略，因为所有数据都以二进制方式存储。在一些非常严谨的系统上，ios::binary 标记是非法的，甚至不能通过编译。

二进制文件不仅能够存储整数和字符串，还能够存储整个数据结构。可以使用 fstream 的 write( )方法一次性写入所有的数据。

如果使用 write( )写入，可以用 read( )来读取。然而，这两个函数都接受一个字符指针作为参数，因此必须将类的地址强制转换为字符指针。

这些函数的第二个参数是要读写的字符数，这可以使用 sizeof( )来确定。注意，写入的只是类的数据，而不包括方法；读取的也只有数据。程序清单 17.18 演示了如何将对象的内容写入文件。

程序清单 17.18　将对象的内容写入文件

```
0:  //Listing 17.18 Writing a Class to a File
1:  #include <fstream>
```

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

class Animal
{
  public:
    Animal(int weight,long days):itsWeight(weight),DaysAlive(days) {}
    ~Animal(){}

    int GetWeight()const { return itsWeight; }
    void SetWeight(int weight) { itsWeight = weight; }

    long GetDaysAlive()const { return DaysAlive; }
    void SetDaysAlive(long days) { DaysAlive = days; }

  private:
    int itsWeight;
    long DaysAlive;
};

int main()   // returns 1 on error
{
   char fileName[80];


   cout << "Please enter the file name: ";
   cin >> fileName;
   ofstream fout(fileName,ios::binary);
   if (!fout)
   {
     cout << "Unable to open " << fileName << " for writing.\n";
     return(1);
   }

   Animal Bear(50,100);
   fout.write((char*) &Bear,sizeof Bear);

   fout.close();

   ifstream fin(fileName,ios::binary);
   if (!fin)
   {
     cout << "Unable to open " << fileName << " for reading.\n";
     return(1);
   }

   Animal BearTwo(1,1);

   cout << "BearTwo weight: " << BearTwo.GetWeight() << endl;
   cout << "BearTwo days: " << BearTwo.GetDaysAlive() << endl;

   fin.read((char*) &BearTwo, sizeof BearTwo);
```

```
54:
55:     cout << "BearTwo weight: " << BearTwo.GetWeight() << endl;
56:     cout << "BearTwo days: " << BearTwo.GetDaysAlive() << endl;
57:     fin.close();
58:     return 0;
59: }
```

输出：

```
Please enter the file name: Animals
BearTwo weight: 1
BearTwo days: 1
BearTwo weight: 50
BearTwo days: 100
```

分析：

第 5～20 行声明了一个简化的 Animal 类。第 24～34 行创建了一个文件，并以二进制模式打开它进行输出。第 36 行创建了一个体重为 50、年龄是 100 天的 Animal 对象，第 37 行将其数据写入文件。

第 39 行关闭该文件，第 41 行以二进制模式重新打开该文件以便进行读取。第 48 行创建了第二个 Animal 对象，其体重为 1，年龄只有 1 天。第 53 行将文件中的数据读入新 Animal 对象中，这将用文件中的数据替换原来的数据，输出证明了这一点。

17.12　命令行处理

很多操作系统（如 DOS 和 UNIX）都允许用户在启动程序时给它传递参数。这些参数被称为命令行选项，通常在命令行上用空格将它们分开。例如：

```
SomeProgram Param1 Param2 Param3
```

这些参数并不直接传递给 main( )函数。相反，每个程序的 main( )函数都被传入两个参数。第一个参数是一个整数，它指定了命令行参数数目，其中包括程序名本身，因此每个程序至少有一个参数。前面给出的命令行范例包含 4 个参数（程序名 SomeProgram 和 3 个参数，这样总共是 4 个命令行参数）。

传递给函数 main( )的第二个参数是一个字符串指针数组。由于数组名是一个指向数组第一个元素的常量指针，因此可以将参数声明为指向字符指针的指针、指向字符数组的指针或字符数组的数组。

第一个参数通常名为 argc（参数数目），但也可以给它指定其他名称；第二个参数通常名为 argv（参数向量），但这只是一种约定。

通常应检测 argc 以核实 main( )函数接受到预期的参数数目，并使用 argv 来访问字符串本身。注意 argv[0]是程序名，argv[1]是以字符串表示的程序的第一个参数。如果程序接受两个数字作为参数，需要将这些数字转换为字符串。第 21 章将介绍如何使用标准库转换。程序清单 17.19 演示了如何使用命令行参数。

程序清单 17.19　使用命令行参数

```
0: //Listing 17.19 Using Command-line Arguments
1: #include <iostream>
2: int main(int argc, char **argv)
3: {
4:    std::cout << "Received " << argc << " arguments...\n";
5:    for (int i=0; i<argc; i++)
6:       std::cout << "argument " << i << ": " << argv[i] << std::endl;
7:    return 0;
8: }
```

输出：

```
TestProgram Teach Yourself C++ In 21 Days
Received 7 arguments...
argumnet 0: TestProgram
argument 1: Teach
argument 2: Yourself
argument 3: C++
argument 4: In
argument 5: 21
argument 6: Days
```

注意：必须在命令行（也就是 DOS 窗口中）运行上述代码，或者在编译器中设置命令行参数（请参阅编译器文档）。

分析：

函数 main( )接受两个参数：argc 是一个表示命令行参数个数的整数；argv 是一个指向字符串数组的指针。在 argv 指向的数组中，每个字符串都是一个命令行参数。注意，也可以将 argv 声明为 char *argv[ ]或 char argv[ ][ ]。如何声明 argv 是一个编程风格问题，虽然该程序将它声明为指向指针的指针，但仍可以使用下标表示法来访问各个字符串。

第 4 行使用 argc 来打印命令行参数个数：总共有 7 个，其中包括程序名本身。

第 5 和第 6 行打印每个命令行参数：使用下标表示法将以空字符结尾的字符串传递给 cout。

程序清单 17.20 演示了一种更常见的命令行参数的用途，它修改了程序清单 17.18，以命令行参数的方式来获取文件名。

程序清单 17.20　使用命令行参数来获取文件名

```
 0: //Listing 17.20 Using Command-line Arguments
 1: #include <fstream>
 2: #include <iostream>
 3: using namespace std;
 4:
 5: class Animal
 6: {
 7:   public:
 8:     Animal(int weight,long days):itsWeight(weight),DaysAlive(days){}
 9:     ~Animal(){}
10:
11:     int GetWeight()const { return itsWeight; }
12:     void SetWeight(int weight) { itsWeight = weight; }
13:
14:     long GetDaysAlive()const { return  DaysAlive; }
15:     void SetDaysAlive(long days) { DaysAlive = days; }
16:
17:   private:
18:     int itsWeight;
19:     long DaysAlive;
20: };
21:
22: int main(int argc, char *argv[])  // returns 1 on error
23: {
24:    if (argc != 2)
25:    {
```

```
26:       cout << "Usage: " << argv[0] << " <filename>" << endl;
27:       return(1);
28:    }
29:
30:    ofstream fout(argv[1],ios::binary);
31:    if (!fout)
32:    {
33:       cout << "Unable to open " << argv[1] << " for writing.\n";
34:       return(1);
35:    }
36:
37:    Animal Bear(50,100);
38:    fout.write((char*) &Bear,sizeof Bear);
39:
40:    fout.close();
41:
42:    ifstream fin(argv[1],ios::binary);
43:    if (!fin)
44:    {
45:       cout << "Unable to open " << argv[1] << " for reading.\n";
46:       return(1);
47:    }
48:
49:    Animal BearTwo(1,1);
50:
51:    cout << "BearTwo weight: " << BearTwo.GetWeight() << endl;
52:    cout << "BearTwo days: " << BearTwo.GetDaysAlive() << endl;
53:
54:    fin.read((char*) &BearTwo, sizeof BearTwo);
55:
56:    cout << "BearTwo weight: " << BearTwo.GetWeight() << endl;
57:    cout << "BearTwo days: " << BearTwo.GetDaysAlive() << endl;
58:    fin.close();
59:    return 0;
60: }
```

输出：

```
BearTwo weight: 1
BearTwo days: 1
BearTwo weight: 50
BearTwo days: 100
```

分析：

Animal 类的声明和程序清单 17.18 完全相同，但这次没有提示用户输入文件名，而是使用命令行参数来获取。第 22 行将 main( )函数声明为接受两个参数：命令行参数个数和指向命令行参数字符串数组的指针。

在第 24～28 行，程序确保收到预期的参数个数（两个）。如果用户没有提供一个文件名，则打印一条错误消息：

```
Usage TestProgram <filename>
```

然后程序退出。注意，通过使用 argv[0]而不是程序名，可以把将该程序编译为任何名称，而不会妨碍该语句的执行结果。甚至可以将编译后的可执行文件重命名，该语句仍将正确执行。

在第 30 行，程序试图以二进制输出模式打开指定的文件。没有理由将文件名复制到局部临时变量中，可以通过访问 argv[1]来直接使用它。

第 42 行再次使用了这种技巧，重新打开该文件进行输入。当文件不能打开时，错误状态语句也使用了这种技巧（第 33 行和第 45 行）。

17.13 小　　结

本章介绍了流，讨论了全局对象 cin 和 cout。istream 和 ostream 对象旨在封装写入设备驱动程序以及缓冲输入和输出的工作。

在每个程序中都创建了 4 个标准流对象：cout、cin、cerr 和 clog。在很多操作系统中，这些对象都可以被重定向。

istream 对象 cin 用于输入，它通常与重载的提取运算符（>>）一起使用；ostream 对象 cout 用于输出，它通常与重载的插入运算符（<<）一起使用。

这些对象还包含很多其他的成员函数，如 get( )和 put( )。由于这些成员函数的常用版本返回一个流对象引用，因此可以级联这些运算符和函数。

可以使用控制符来修改流对象的状态。它们可以设置流对象的格式化和显示特征以及其他各种属性。

可以使用从流类派生而来的 fstream 类来实现文件 I/O。除支持插入和提取运算符外，这些对象还支持用于存储和检索大型二进制对象的函数 read( )和 write( )。

17.14 问 与 答

问：如何确定何时使用插入和提取运算符，何时使用流类的其他成员函数？

答：一般而言，使用插入和提取运算符较容易，在其行为能够满足需求时应首选它们。在这些运算符不能满足需求的非常情况下（如读取包含多个单词的字符串），可使用其成员函数。

问：cerr 和 clog 之间的区别是什么？

答：ceer 不被缓冲，写入 cerr 的内容立即被输出。需要将错误消息写入屏幕时，这很合适；但将日志写入磁盘时，这将极大地影响程序的性能。Clog 对输出进行缓冲，因此效率更高，但风险是如果程序崩溃将丢失部分日志数据。

问：既然 printf( )很管用为什么还要创建流？

答：printf( )不支持 C++的强类型系统，也不支持用户定义的类。对 printf( )的支持实际上是对 C 编程语言的传承。

问：什么时候使用 putback( )？

答：在使用读操作来确定字符是否有效，但另一个读操作（可能由另一个对象执行）需要该字符位于缓冲区中时。这种情况大都发生在分析文件时，例如，C++编译器可能使用 putback( )。

问：我的朋友在 C++程序中使用 printf( )，我也可以这样做吗？

答：不可以。现在应将 printf( )视为被摒弃的。

17.15 作　　业

作业包括测验和练习，前者帮助加深读者对所学知识的理解，后者提供了使用新学知识的机会。请尽量先完成测验和练习题，然后再对照附录 D 中的答案，继续学习下一章之前，请务必弄懂这些答案。

17.15.1 测验

1. 什么是插入运算符？其作用是什么？

2. 什么是提取运算符？其作用是什么？
3. cin.get( )的 3 种形式是什么？它们之间有什么区别？
4. cin.read( )和 cin.getline( )之间有什么不同？
5. 使用插入运算符输出 long 时默认宽度是多少？
6. 插入运算符的返回值是什么？
7. ofstream 的构造函数接受什么参数？
8. 参数 ios::ate 有何作用？

17.15.2　练习

1. 编写一个程序，将数据写入 4 个标准 iostream 对象：cin、cout、cerr 和 clog。
2. 编写一个程序，提示用户输入其全名，然后将其显示在屏幕上。
3. 重写程序清单 17.9，使其功能保持不变，但不使用 putback( )和 ignore( )。
4. 编写一个程序，它接受一个文件名作为参数，并打开这个文件进行读取。读取该文件的每个字符，但只将字母和标点符号显示到屏幕上（忽略所有的非打印字符），然后关闭文件并退出。
5. 编写一个程序，按相反的顺序显示其命令行参数，但不显示程序名。