chapter17 输入、输出和文件

1）c++的输入与输出概论

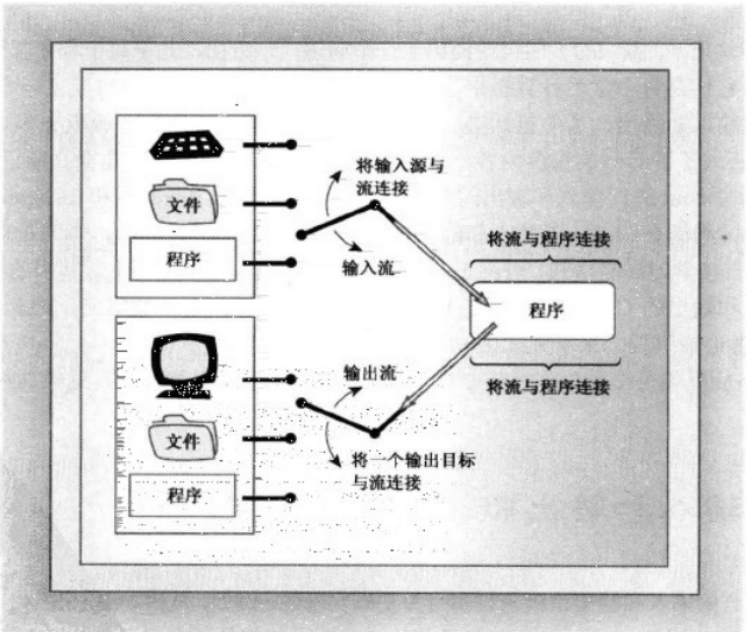
（1）流和缓冲区

【1】流:

C++程序把输入和输出看作字节流。输入时，程序从输入流中抽取字节；输出时，程序将字节插入到输出流中。对于面向文本的程序，每个字节代表一个字符，更通俗地说，字节可以构成字符或数值数据的二进制表示。输入流中的字节可能来自键盘，也可能来自存储设备（如硬盘）或其他程序。同样，输出流中的字节可以流向屏幕、打印机、存储设备或其他程序。流充当了程序和流源或流目标之间的桥梁。这使得C++程序可以以相同的方式对待来自键盘的输入和来自文件的输入。C++程序只是检查字节流，而不需要知道字节来自何方。同理，通过使用流，C++程序处理输出的方式将独立于其去向。因此管理输入包含两步：

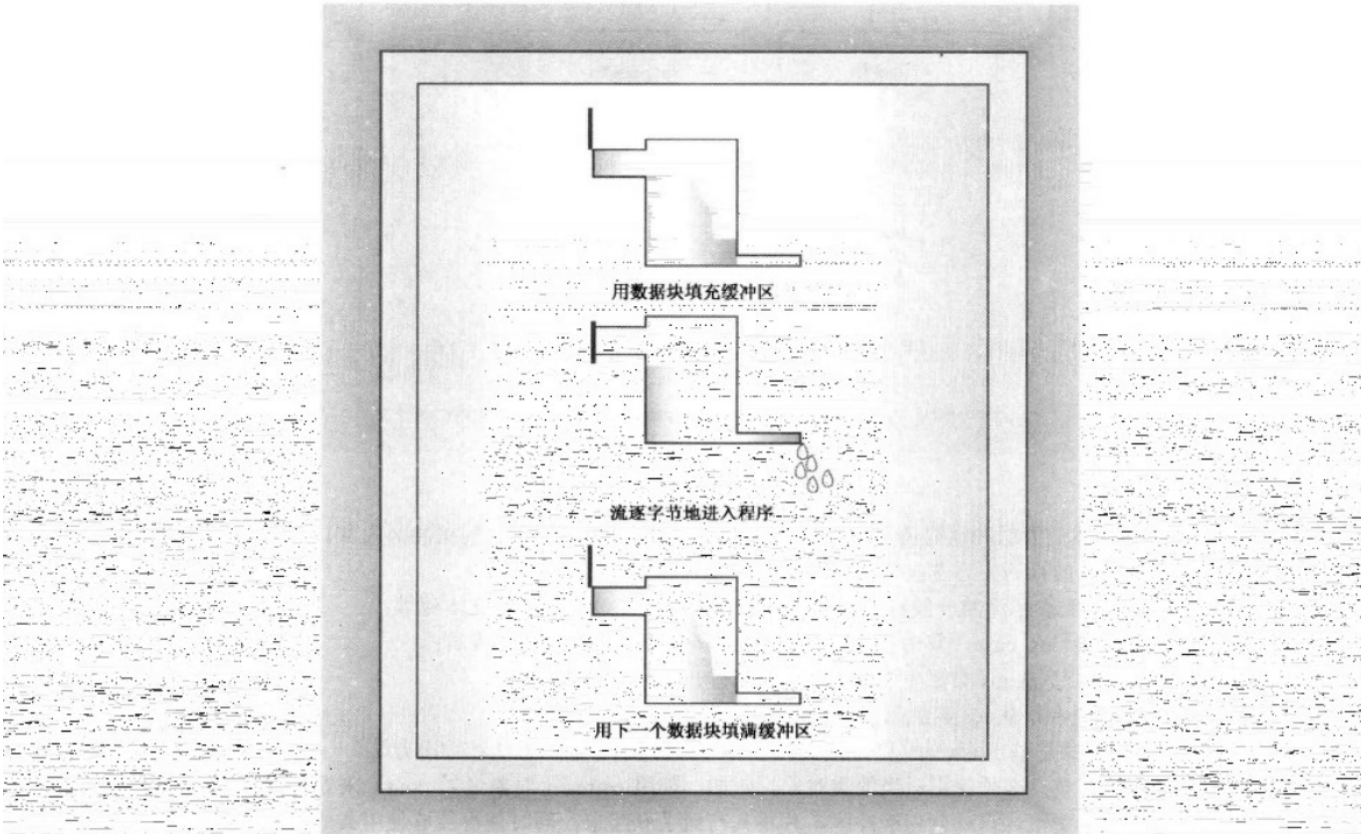
- 将流与输入去向的程序关联起来。
- 将流与文件连接起来。

换句话说，输入流需要两个连接，每端各一个。文件端部连接提供了流的来源，程序端连接将流的流出部分转储到程序中（文件端连接可以是文件，也可以是设备，如键盘）。同样，对输出的管理包括将输出流连接到程序以及将输出目标与流关联起来。这就像将字节（而不是水）引入到水管中（参见图 17.1）。



【2】缓冲区:

通常，通过使用缓冲区可以更高效地处理输入和输出。缓冲区是用作中介的内存块，它是将信息从设备传输到程序或从程序传输给设备的临时存储工具。通常，像磁盘驱动器这样的设备以 512 字节（或更多）的块为单位来传输信息，而程序通常每次只能处理一个字节的信息。缓冲区帮助匹配这两种不同的信息传输速率。例如，假设程序要计算记录在硬盘文件中的金额。程序可以从文件中读取一个字符，处理它，再从文件中读取下一个字符，再处理，依此类推。从磁盘文件中每次读取一个字符需要大量的硬件活动，速度非常慢。缓冲方法则从磁盘上读取大量信息，将这些信息存储在缓冲区中，然后每次从缓冲区里读取一个字节。因为从内存中读取单个字节的速度比从硬盘上读取快很多，所以这种方法更快，也更方便。当然，到达缓冲区尾部后，程序将从磁盘上读取另一块数据。这种原理与水库在暴风雨中收集几兆加仑流量的水，然后以比较文明的速度给您家里供水是一样的（见图 17.2）。输出时，程序首先填满缓冲区，然后把整块数据传输给硬盘，并清空缓冲区，以备下一批输出使用。这被称为刷新缓冲区（flushing the buffer）。



> ps：c++程序常在用户按下回车键时刷新输入缓冲区，常在用户发送换行符时刷新输出缓冲区

（2）管理流和缓冲区的工作

【1】类别：

> （1）streambuf类：为缓冲区提供了内存，并提供了用于填充缓冲区、访问缓冲区内容、刷新缓冲区、管理缓冲区内存的类方法
> （2）ios_base类：表示流的一般特征，如：是否可以读取、是二进制流还是文本流...
> （3）ios类：基于ios_base，其中新增了一个指向streambuf对象的指针成员
> （4）ostream类：从ios类中派生出来的，提供了输出方法
> （5）istream类：从ios类中派生出来的，提供了输入方法
> （6）iostream类：继承istream与ostream类

> ps：要使用上述工具，必须使用适当的类对象。例如：使用ostream对象(如：cout)来处理输出。创建这样的对象将打开一个流，自动创建缓冲区，并将其与流关联起来，同时使得能够使用类成员函数

【2】默认的传入/传出设备解析：

> （1）cin对象对应于*标准输入流*，默认时这个流被关联到标准输入设备（通常为键盘）
> （2）cout对象对应于*标准输出流*，默认时这个流被关联到标准输出设备（通常为显示器）
> （3）cerr对象对应于*标准错误流*，可用于显示错误信息。默认时这个流被关联到标准输出设备（通常为显示器）。**这个流没有被缓冲**，这意味着消息被直接发送给屏幕，而不会等到缓冲区填满或新的换行符
> （4）clog对象对应于*标准错误流*，可用于显示错误信息。默认时这个流被关联到标准输出设备（通常为显示器）。**但是这个流被缓冲**！

【3】重定向：
能够改变标准输入与标准输出
<File_Name：改变了输入流的流入端连接，但是流出端仍然和程序相连
>File_Name：改变了输出流的流出端连接，但是流入端仍然和程序相连

2）使用cout进行输出

（poi）刷新输出缓冲区：
【1】问题引入：

如果程序使用 cout 将字节发送给标准输出，情况将如何？由于 ostream 类对 cout 对象处理的输出进行缓冲，所以输出不会立即发送到目标地址，而是被存储在缓冲区中，直到缓冲区填满。然后，程序将刷新（flush）缓冲区，把内容发送出去，并清空缓冲区，以存储新的数据。通常，缓冲区为 512 字节或其整数倍。当标准输出连接的是硬盘上的文件时，缓冲可以节省大量的时间。毕竟，不希望程序为发送 512 个字节，而存取磁盘 512 次。将 512 个字节收集到缓冲区中，然后一次性将它们写入硬盘的效率要高得多。

然而，对于屏幕输出来说，首先填充缓冲区的重要性要低得多。如果必须重述消息“Press any key to continue”以便使用 512 个字节来填充缓冲区，实在是太不方便了。所幸的是，在屏幕输出时，程序不必等到缓冲区被填满。例如，将换行符发送到缓冲区后，将刷新缓冲区。另外，正如前面指出的，多数 C++实现都会在输入即将发生时刷新缓冲区。也就是说，假设有下面的代码：

```
cout << "Enter a number: ";
float num;
cin >> num;
```

程序期待输入这一事实，将导致它立刻显示 cout 消息（即刷新“Enter a number:”消息），即使输出字符串中没有换行符。如果没有这种特性，程序将等待输入，而无法通过 cout 消息来提示用户。

如果实现不能在所希望时刷新输出，可以使用两个控制符中的一个来强行进行刷新。控制符 flush 刷新缓冲区，而控制符 endl 刷新缓冲区，并插入一个换行符。这两个控制符的使用方式与变量名相同：

【2】知识导入：

```
（1）cout<<flush; 刷新缓冲区

（2）cout<<endl;刷新缓冲区 并 插入一个换行符
```

3）使用cin进行输入

（1）cin>>如何检查输入：

不同版本的抽取运算符查看输入流的方法是相同的。它们跳过空白（空格、换行符和制表符），直到遇到非空白字符。即使对于单字符模式（参数类型为 char、unsigned char 或 signed char），情况也是如此，但对于 C 语言的字符输入函数，情况并非如此（参见图 17.5）。在单字符模式下，>>运算符将读取该字符，将它放置到指定的位置。在其他模式下，>>运算符将读取一个指定类型的数据。也就是说，它读取从非空白字符开始，到与目标类型不匹配的第一个字符之间的全部内容。

```
char philosophy[20];
int distance;
char initial;

cin >> philosophy >> distance >> initial;
```

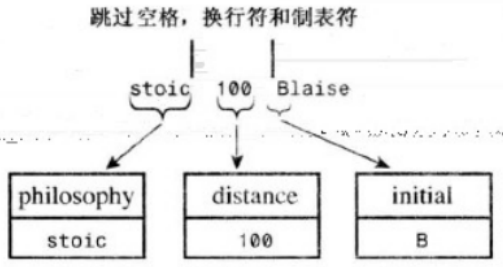


图 17.5　cin>>跳过空白

例如，对于下面的代码：

```
int elevation;
cin >> elevation;
```

假设键入下面的字符：

```
-123Z
```

运算符将读取字符-、1、2 和 3，因为它们都是整数的有效部分。但 Z 字符不是有效字符，因此输入中最后一个可接受的字符是 3。Z 将留在输入流中，下一个 cin 语句将从这里开始读取。与此同时，运算符将字符序列-123 转换为一个整数值，并将它赋给 elevation。

（2）流状态：

[1] cin 或 cout对象包含一个描述**流状态(stream state)** 的数据成员
[2] 流状态由三个 ios_base 元素组成：eofbit、badbit、failbit（每个元素都是一位，要么是1【设置】，要么是0【清除】）

表 17.4 流状态

成　员	描　述
eofbit	如果到达文件尾，则设置为 1
badbit	如果流被破坏，则设置为 1；例如，文件读取错误
failbit	如果输入操作未能读取预期的字符或输出操作没有写入预期的字符，则设置为 1
goodbit	另一种表示 0 的方法
good( )	如果流可以使用（所有的位都被清除），则返回 true
eof( )	如果 eofbit 被设置，则返回 true
bad( )	如果 badbit 被设置，则返回 true
fail( )	如果 badbit 或 failbit 被设置，则返回 true
rdstate( )	返回流状态
exceptions( )	返回一个位掩码，指出哪些标记导致异常被引发
exceptions(iostate ex)	设置哪些状态将导致 clear( )引发异常；例如，如果 ex 是 eofbit，则如果 eofbit 被设置，clear( )将引发异常
clear(iostate s)	将流状态设置为 s；s 的默认值为 0（goodbit）；如果(restate( )& exceptions( ))! = 0，则引发异常 basic_ios::failure
setstate(iostate s)	调用 clear（rdstate( )| s）。这将设置与 s 中设置的位对应的流状态位，其他流状态位保持不变

（3）其他istream类方法：

【1】单字符输入：
cin.get(char& ch)：读取下一个输入字符[即使该字符是空格/制表符/换行符]，将输入字符赋给其参数
cin.get( )：读取下一个输入字符[即使该字符是空格/制表符/换行符]，将输入字符转换成整型(int)

cin.get(char&)：返回值是(istream&)——cin形式

```
int ct = 0;
char ch;
cin.get(ch);
while(ch!='\n')
{
        cout << ch;
        ct++;
        cin.get(ch);
}
cout << ct <<endl;

输入：
I C++ clearly
输出：
I C++ clearly
-----------------------------------------------------------------------------------------------------------------------
如果使用的是cin >> ch;
则：
```

```
（1）输出结果是：IC++clearly
（2）循环不会终止！（cin>>）跳过了换行符，因此代码不会把换行符('\n')赋给ch
```

---

cin.get( )：返回值是int

```
int ct = 0;
char ch;
cin.get();
while(ch!='\n')
{
        cout << ch;
        ct++;
        cin.get();
}
cout << ct <<endl;

输入：
I C++ clearly
输出：
I C++ clearly
```

需要注意的是：

```
char c1,c2,c3;
cin.get(c1).get(c2) >> c3;      // valid!
等价于：
1) cin.get(c1)返回成cin
2) 然后cin.get(c2),返回一个cin
3) 最后 cin >> c3
(c1 与 c2 可以是空格，c3不可以！)
-----------------------------------------------------------------------------------------------------------------------
char c1,c2,c3;
cin.get().get() >> c3;          // not valid!
等价于：
1) cin.get() 返回一个int值
2) int.get()  ? ? ?
```

【2】字符串输入：
（1）get (str_name , num , divid_char)

```
istream& get(char*, int, char)
```

（2）getline (str_name , num , divid_char)

```
 istream& getline(char*, int, char)
```

参数1：用于放置输入字符串的内存单元地址
参数2：要比str的最大字符数大1（用于结尾的空字符）
参数3：指定用作分界符的字符

> ps（二者区别）：
> get()：将换行符留在输入流中，这样接下来的输入操作首先看到的是换行符
> getline()：抽取并丢弃输入流中的换行符