第 12 章　C++的 I/O 流类库

本章摘要

输入输出是程序的一个重要组成部分，程序运行所需要的数据往往要从外设（如键盘、磁盘等）得到，程序的运行结果通常也要输出到外设（如显示器、打印机、磁盘等）中去。C++系统提供了一个用于输入输出操作的类体系，这个类体系提供了对预定义类型进行输入输出操作的能力，程序员也可以利用这个类体系进行自定义类型的输入输出操作。

12．1　C++的流及流类库

12．1．1　C++的流

输入和输出是数据传送的过程，数据如流水从一处流向另一处。C++形象地将此过程称为流（stream）。C++的输入输出流是指由若干字节组成的字节序列，这些字节中的数据按顺序从一个对象传送到另一个对象。流表示了信息从源到目的端的流动。在输入操作时，字节流从输入设备（如键盘、磁盘）流向内存，称为输入流；在输出操作时，字节流从内存流向输出设备（如显示器、打印机、磁盘等），称为输出流。流中的内容可以是 ASCII 字符、二进制形式的数据、图形图像、数字音频视频或其它形式的信息。

实际上，在内存中为每一个数据流开辟一个内存缓冲区，用来存放流中的数据。当用 cout 和插入运算符“<<”向显示器输出数据时，先将这些数据送到程序中的输出缓冲区保存，直到缓冲区满了或遇到 endl，就将缓冲区中的全部数据送到显示器显示出来。在输入时，从键盘输入的数据先放在键盘的缓冲区中，当按回车键时，键盘缓冲区中的数据输入到程序中的输入缓冲区，形成 cin 流，然后用提取运算符“>>”从输入缓冲区中提取数据送给程序中的有关变量。

在 C++中，输入输出流被定义为类。C++的 IO 库中的类称为流类（stream class）。用流类定义的对象称为流对象。cin 和 cout 就是 iostream_withassign 类的对象。

12．2．2　流类库

C++的 I / O 流类库是用继承方法建立起来的一个输入输出类库，它具有两个平行类：即 streambuf 类和 ios 类，这两个类是基本类，所有的流类都可以由它们派生出来。

1．streambuf 类

streambuf 类提供物理设备的接口，它提供缓冲或处理流的通用方法，几乎不需要任何格式。缓冲区由一个字符序列和两个指针组成，这两个指针是输入缓冲区指针和输出缓冲区指针，它们分别指向字符要被插入或取出的位置。

streambuf 类提供对缓冲区的低级操作，如设置缓冲区、对缓冲区指针进行操作、从缓

冲区取字符、向缓冲区存储字符。它有三个派生类，即 filebuf、strstreambuf 和 conbuf，其成员函数大多采用内置函数方式定义，以提高效率。

filebuf 类使用文件来保存缓冲区中的字符序列。当读文件时，实际是将指定文件中的内容读到缓冲区中来；当写文件时，实际是将缓冲区的字符写到指定的文件中。

strstreambuf 类扩展了 streambuf 类的功能，它提供了在内存中进行提取和插入操作的缓冲区管理。

conbuf 类扩展了 streambuf 类的功能，用于处理输出。它提供了控制光标、设置颜色、定义活动窗口、清屏等功能，为输出操作提供缓冲区管理。

2．ios 类

ios 类是一个虚基类，它主要定义了用于格式化输入输出及出错处理的成员函数。在 ios 类和它的各级派生类中，均含有一个指向流缓冲类 streambuf 的对象的指针。ios 类（及其派生类）使用 streambuf 类以及从它派生的文件缓冲类 filebuf 和字符串缓冲类 strstreambuf 进行输入输出。

ios 作为流类库中的一个基类，可以派生出许多类。常用 ios 流类的简要说明和类声明所在的头文件名如表 12-1 所示。

表 12-1 ios 流类列表

类名	说明	包含头文件
抽象流基类		
ios	流基类	iostream.h
输入流类		
istream	普通输入流类和用于其它输入流的基类	iostream.h
ifstream	输入文件流类	fstream.h
istream_withassign	cin 的输入流类	iostream.h
istrstream	输入字符串流类	strstream.h
输出流类		
ostream	普通输出流类和用于其它输出流类的基类	iostream.h
ofstream	输出文件流类	fstream.h
ostream_withassign	用于 cout、cerr 和 clog 的流类	iostream.h
ostrstream.	输出字符串流类	strstream.h
输入/输出流类		
iostream	普通输入/输出流类和用于其它输入/输出流的基类	iostream.h
iostream_withassign	iostream	iostream.h
fstream	输入/输出文件流类	fstream.h
strstream	输入/输出字符串流类	strstream.h

从表中可以看出，C++流类库中，streambuf、ios、istream、ostream、iostream、istream_withassign 和 ostream_withassign 这些基本 I/O 流类和预定义的 cin、cout、cerr 和 clog 在 iostream.h 文件中说明。ifstream、ofstream 和 fstream 在 fstream.h 中说明。istrstream、

ostrstream 和 strstream 在 strstream.h 中说明。

由于 fstream.h 和 strstream.h 中都包含了 iostream.h，所以如果使用标准输入/输出（控制台 I/O），只要包含 iostream.h 头文件即可，如果使用 fstream 或者 strstream，只要包含相应的 fstream.h 和 strstream.h 即可。

ios 流类和它的派生类的结构关系如图 12-1 所示。

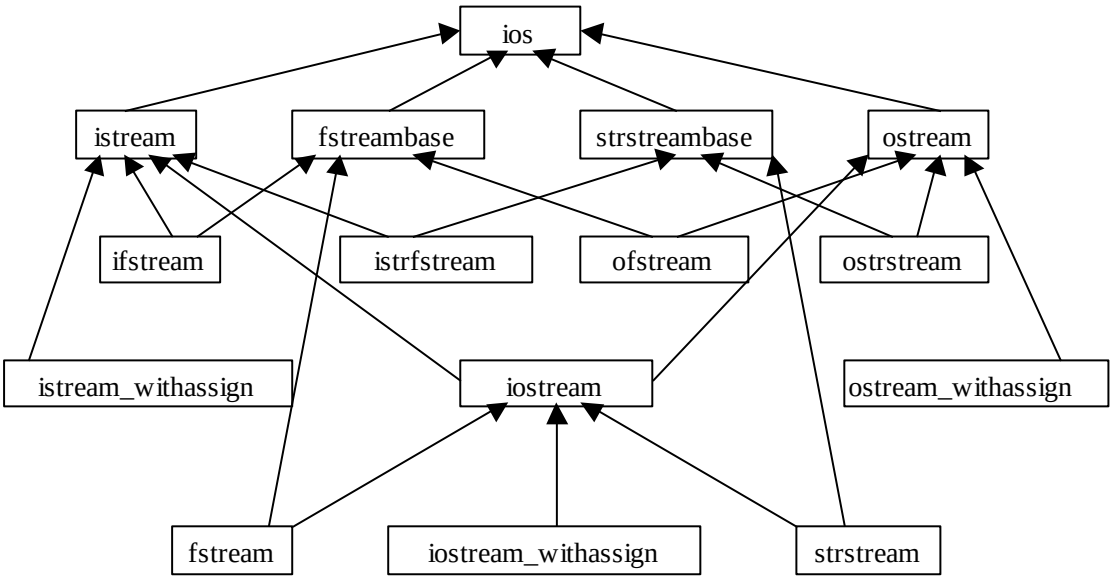

图 12-1　ios 流类层次图

从图 12-1 中可以看出 ios 类有四个直接派生类，即输入流类、输出流类、文件流类和字符串流类，这四种流作为流库中的基本流类。

从该图 12-1 中还可以看出各个类的继承关系。例如，输入文件流类 ifstream 是由输入流类 istream 和文件流类 fstreambase 派生而来，输入字符串流类 istrstream 是由输入流类 istream 和字符串流类 strstreambase 派生而来的，输入输出流类 iostream 是由输入流类 istream 和输出流类 ostream 派生而来的。例如：

class ios；

class istream：virtual public ios；

class ostream：virtual public ios；

class iostream：public istream，public ostream；

3 个_withassgn 类分别是输入流类 istream、输出流类 ostream 和输入输出流类 iostream 的派生类：

class istream_withassign：public istream；

class ostream_withassign：public ostream；

class iostream_withassign：public iostream；

在头文件 iostream．h 中，提供了 4 个预定义的标准流对象：

extern istream_withassign cin；　// 预定义代表标准输入设备键盘

extern ostream_withassign cout； // 预定义代表标准输出设备显示器

extern istream_withassign cerr； // 预定义代表标准出错输出设备显示器

extern istream_withassign clog； // 预定义代表标准出错输出设备显示器

这四个标准流对象是程序中进行标准输入输出要用到的。cin 对象是从标准输入设备（键盘）输入到内存的数据流，称为 cin 流或标准输入流。cout 对象是从内存输入到标准输出设备（显示器）的数据流，称为 cout 流或标准输出流。cerr 和 clog 均为向输出设备（显示器）输出出错信息。

12．2 格式化 I/O

在很多情况下，程序员需要控制输入输出格式。C++提供了两种格式控制方法：一种方法是使用 ios 类中有关格式控制的成员函数；另一种方法是使用称为控制符的特殊类型的函数。

12．2．1 用 ios 类的成员函数进行格式控制

ios 类中有几个成员函数可以用来对输入输出进行格式控制。进行格式控制主要是通过对格式状态字、域宽、填充字符以及输出精度的操作来完成的。

1．格式状态字

C++可以给每个流对象的输入输出进行格式控制，以满足程序员对输入输出格式的需求。输入输出格式由一个 long int 类型的格式状态字确定，状态字的各位都控制一定的输入输出特征，例如格式状态字的最右一位为 1，则表示在输入时跳过空白符号。在 ios 类的 public 部分定义了一个枚举，它的每个成员可以分别定义格式状态字的一个位，每一个位称为一个状态标志位。这个枚举的定义如下：

```
enum {
        skipws      = 0x0001,
        left        = 0x0002,
        right       = 0x0004,
        internal    = 0x0008,
        dec         = 0x0010,
        oct         = 0x0020,
        hex         = 0x0040,
        showbase    = 0x0080,
        showpoint   = 0x0100,
        uppercase   = 0x0200,
        showpos     = 0x0400,
        scientific  = 0x0800,
        fixed       = 0x1000,
        unitbuf     = 0x2000,
```

```
        stdio          = 0x4000
    };
```

枚举类型中定义的值用于设置或清除格式化输入输出中的控制信息的标志，其作用如下：

（1）skipws

设置该标志后，当从一个流中输入时，将跳过开头的空白字符（包括空格、制表符及换行符）；否则不忽略空白字符。缺省为设置。

（2）left、right、internal

在一次输出中，这三个标志只能设置其中的一个。如果设置 1eft，则输出左对齐，域宽的其余部分用填充符填充；如果设置 right，则输出右对齐；如果设置 internal，则数值的符号左对齐，而数值本身右对齐，符号和数值之间为填充符。

(3)dec、oct 、hex

这三个标志也只能设置其中的一个。如果设置 dec 标志，则数值以十进制形式输入输出；如果设置 oct 标志，则以八进制形式输入输出，如果设置 hex，则以十六进制形式输入输出。缺省为 dec。

（4）showbase

设置该标志后，数值数据的前面要显示“基指示符”。例如，假定设置了标志 hex 和 showbase，则在输出数值数据时，前面都要显示前缀“0x”。缺省为不设置。

（5）showpoint

设置该标志后，强制显示 float 和 double 型数据的小数点及小数点后的无效 0。缺省为不设置。

（6）uppercase

如果设置该标志，则数值中的字母（如指数符号“e”、十六进制数字等）均以大写字母显示。缺省为不设置。

（7）showpos

设置该标志后，当显示正数时，前面加上一个正号“＋”。缺省为不设置。

（8）scientific、fixed

如果设置了 scientific，则浮点数用科学记数法输出；如果设置了 fixed，则浮点数按常规显示。如果二者都不设置，则编译程序根据情况选用一种合适的方式。

（9）unitbuf

设置该标志后，在输出操作后立即刷新所有流，刷新一个流意味着把输出写到与流相连的物理设备上。

（10）stdio

如果设置该标志，则每次输出后，刷新 stdout 和 stderr。

2．ios 类中用于控制输入输出格式的成员函数

在 ios 类的说明中，定义了几个用于格式控制的公有成员函数，列举如下：

```
long    ios::flags();           // 返回当前格式状态字
long    ios::flags(long):       // 设置格式状态字并返回
```

```
long    ios::setf(long);        // 设置指定的标志位
long    ios::unsetf(long);      // 清除指定的标志位
int     ios::width();           // 返回当前显示数据的域宽
int     ios::width(int);        // 设置当前显示数据域宽并返回原域宽
char    ios::fill();            // 返回当前填充字符
char    ios::fill(char);        // 设置填充字符并返回原填充字符
int     ios::precision();       // 返回当前浮点数的精度
int     ios::precision(int);    // 设置浮点数精度并返回原精度
```

下面用几个例子来说明以上几个成员函数的使用。

例 12-1　格式状态字的设置和清除

```
#include <iostream>
using namespace std;
int main()
{
    long f;
    int i;
    double x;
    i=200;
    x=53.6;
    f=cout.flags();                          // 取当前格式状态字
    cout<<"The flags is "<<f<<endl;
    cout<<"i="<<i<<",x="<<x<<endl;
    cout.unsetf(ios::dec);                   // 终止十进制的格式设置
    cout.setf(ios::hex|ios::showbase);       // 设置以十六进制输出并显示基数符号
    cout.setf(ios::scientific);              // 指定科学记数法输出
    cout<<"i="<<i<<",x="<<x<<endl;
    cout.setf(ios::uppercase);               // 输出字母时以大写显示
    cout<<"i="<<i<<",x="<<x<<endl;
    return 0;
}
```

程序运行结果为：

```
The flags is 513
i=200,x=53.6
i=0xc8,x=5.360000e+001
i=0XC8,x=5.360000E+001
```

说明：

（1）在格式状态字中，dec、oct、hex 属于一组，它们是互相排斥的，只能选一。系统默认指定为 dec。如果想改设置为同一组的另一状态，应该先调用成员函数 unsetf（），先终止原来的设置，然后再调用成员函数 setf() 设置其它状态。如果删除程序中的

“cout.unsetf(ios::dec);”语句，则在其后虽然设置了 hex 格式，由于未终止 dec 格式，因此 hex 的设置不起作用，系统依然以十进制输出。

（2）用 setf（）函数设置格式状态时，可以包含两个或多个格式标志，由于这些格式标志在 ios 类中被定义为枚举值，每一个格式标志以一个二进制位代表，因此可以用位运算符“|”组合多个格式标志。例如程序中的“cout.setf(ios::hex|ios::showbase);”。

例 12-2　域宽和填充字符的设置

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    double x,y;
    i=100;
    x=23.14159;
    y=23.575;
    cout<<"Default width is "<<cout.width()<<endl;
    cout<<"Default fill is "<<cout.fill()<<endl;
    cout<<"Default precision is "<<cout.precision()<<endl;
    cout<<i<<" "<<x<<" "<<y<<endl;
    cout.width(10);
    cout<<i<<" "<<x<<" "<<y<<endl;
    cout.precision(4);
    cout<<i<<" "<<x<<" "<<y<<endl;
    cout.fill('*');
    cout.width(10);
    cout<<i<<" "<<x<<" "<<y<<endl;
    return 0;
}
```

程序运行结果为：

```
Default width is 0
Default fill is
Default precision is 6
100 23.1416 23.575
         100 23.1416 23.575
100 23.14 23.57
*******100 23.14 23.57
```

分析上面的程序和运行结果，可以看出：

（1）在缺省情况下，width()的取值为“0”，表示无域宽，数据按自身的宽度打印；fill()取值为空格；在 VC++中，precision()取值为 6，即输出时保留 6 位有效数字，例如 23.14159

输出时结果为 23.1416，保留 6 位有效数字。

（2）当用 width()函数设置了域宽后，只对紧跟着它的第一个输出有影响，当第一个输出完成后，width 立即自动置为 0。而调用 fill()函数和 precision()函数，设置了 fill 和 precision 后，在程序中一直有效，除非它们被重新设置。

12．2．2　使用 I/O 操纵符进行格式控制

使用 ios 类的成员函数进行输入输出格式控制时，每个函数的调用需要写一条语句，而且不能将它们直接嵌入到输入输出语句中，使人感到不够简洁、方便。C++提供另一种进行输入输出格式控制的方法，这一方法使用一种称为操纵符的特殊函数。在很多情况下，使用操纵符进行格式化控制比用 ios 格式标志和成员函数要方便。

C++预定义的操纵符分为带参数的操纵符和不带参数的操纵符。通常，不带参数的操纵符在 iostream 文件中定义，而带参数的操纵符在 iomanip 文件中定义。

在进行输入输出时，操纵符被嵌入到输入或输出链中，用来控制输入输出格式，而不执行输入或输出操作。程序中如果使用带参数的操纵符，还必须使用预编译命令：

＃ include < iomanip>

将需要的头文件包含进来。

定义在 iostream 头文件中的操纵符有：

endl　输出时插入换行符并刷新流

ends　输出时在字符串后插入空字符（NULL）作为尾符

flush　刷新，把流从缓冲区输出到目标设备

ws　输入时略去空白字符

dec　以十进制形式输入或输出整型数

hex　以十六进制形式输入或输出整型数

oct　以八进制形式输入或输出整型数

定义在 iomanip 头文件中的操纵符有：

setbase(int n)　设置转换基数为 n（n 取值为 0、8、10 或 16），缺省时为 0，表示采用十进制形式输出

resetiosflags(long f)　清除由参数 f 指定的标志位，用于输入输出

setiosflags(long f)　设置参数 f 指定的标志位

setfill(char c)　设置填充字符

setprecision(int n)　设置浮点数精度（缺省为 6）

setw(int n)　设置数据宽度

上面这些操纵符大致可以代替 ios 的格式成员函数的功能，且使用比较方便。例如，为了把整数 456 按十六进制输入，可有两种方式：

```
int i= 456;
cout.unsetf(ios::dec);
cout.setf(ios::hex);
cout<< i <<endl;
```

或者:

```
int i＝456；
cout<< hex << i<<endl;
```

下面通过一个例子来说明操纵符的使用。

例 12-3　操纵符的使用

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int i;
    cout<<"Input i :";
    cin>>i;
    cout<<"Dec: "<<i <<endl;            // 以十进制形式输出
    cout<<"Oct: "<<oct<<i<<endl;     // 以八进制形式输出
    cout<<"Hex: "<<setbase(16)<<i<<endl; // 以十六进制输出
    cout<<dec<<setw(8)<<123<<456<<endl;   // ①
    cout<<123.4567<<"    "<<setiosflags(ios::scientific)<<123.456<<endl;   // ②
    cout<<setfill('*')<<setw(10)<<123<<endl;        // ③
    cout<<setiosflags(ios::left)<<setfill('#')<<setw(10)<<123<<endl;
    cout<<resetiosflags(ios::left)<<setfill('$')<<setw(10)<<123<<endl;
    return 0;
}
```

程序的运行情况如下：

```
Input i :100
Dec: 100
Oct: 144
Hex: 64
      123456
123.457    1.234560e+002
*******123
123#######
$$$$$$$123
```

分析上面的程序和运行结果，可以看出：

（1）程序中语句①设置域宽为 8，之后输出 123 和 456，123 和 456 被连到了一起，得到结果 123456，这表明操纵符 setw 只对最靠近它的输出起作用，也就是说，它的作用是“一次性”的。在 setw(8)之前使用了操纵符 dec，如果去掉 dec，则输出结果为”7b1c8”，这是由于在上一语句使用了操纵符 setbase(16)，其作用仍然保持，所以将 123 和 456 以十六进制形式输出。

（2）程序中语句②输出浮点数，由于系统缺省的精度为 6，输出时保留 6 位有效数字，

小数部分作四舍五入处理，所以输出 123.457。之后采用操纵符 setiosflags(ios::scientific)，设置按科学记数法输出浮点数。

（3）程序中语句③设置域宽 10，并设置填充字符为“*”，从输出结果可以看出系统缺省的对齐方式为右对齐。

12．3　重载 I/O 运算符

C++创建自己的流库的主要目的，就是使用户自定义类型数据的输入输出也能像系统预定义类型数据的输入输出一样简单、方便，这通过重载输入运算符“>>”和输出运算符“<<”来实现的。

12．3．1　重载输出运算符“<<”

通过重载输出运算符“<<”可以实现用户自定义类型的输出。定义输出运算符“<<”重载函数的一般格式如下：

```
ostream    &operate<< (ostream &stream , user_type    obj)
{
    //   操作代码
    return    stream;
}
```

函数中的第一个参数是对 ostream 对象的引用，这意味着 stream 必须是输出流，它可以是其它任何合法的标识符，但必须与 return 后面的标识符相同；第二个参数接收将被输出的对象，其中 user_type 是用户自定义类型名，obj 为该类型的对象名。

例 12-4　输出运算符“<<”的重载

```
#include <iostream>
using namespace std;
class Point
{
public:
      int x,y;
      Point(int i=0,int j=0)
      { x=i;y=j;}
};
ostream &operator<<(ostream &out,Point a)
{
      out<<a.x<<","<<a.y<<endl;
      return out;
}
int main()
```

```
{
    Point a(10,10),b(15,15);
    cout<<a<<b;
    return 0;
}
```

程序运行结果如下：

```
10,10
15,15
```

程序中输出运算符“<<”的重载函数的返回类型为 ostrenm 流对象的引用。此函数有两个参数，第一个参数 out 是对流的引用，定义时出现在“<<”运算符的左边，第二个参数是 Point 类的对象 a，出现在运算符“<<”的右边。函数输出类 Point 对象的两个值。

一般情况下，重载输出运算符“<<”函数及重载输入运算符“>>”函数都不能是类的成员。因为如果一个运算符函数是类的成员，则其左操作数就应当是调用运算符函数的类的对象，而且这一点是无法改变的。但重载输出运算符时，其左边的参数是流，而右边参数是类的对象。因此，重载输出运算符必须是非成员函数。

在例 12-4 中，把变量 x、y 定义为公有成员，因此重载输出运算符函数即使不是类 Point 的成员，仍然可以访问这些变量。但是，把数据定义为公有成员将破坏数据的封装特性，如果重载输出运算符不能作为类的成员函数，而又需要访问类的私有成员，此时应当把输出运算符定义为类的友元，这样就可以访问类的私有成员。

例 12-5　输出运算符重载为类的友元函数

```
#include <iostream.h>
class Point
{
public:
    Point(int i=0,int j=0)
    { x=i;y=j;}
    friend ostream &operator<<(ostream &out,Point &a);
private:
    int x,y;
};
ostream &operator<<(ostream &out,Point &a)
{
    out<<a.x<<","<<a.y<<endl;
    return out;
}
int main()
{
    Point a(10,10),b(15,15);
    cout<<a<<b;
```

```
    return 0;
}
```

上面的程序在 VC++6.0 下能正确运行，如果将第一行改为“#include <iostream>”，并在其下加上一行“using namespace std;”，则在 VC++中编译会出错。这是由于 VC++6.0 编译系统没有完全实现 C++标准，它所提供不带后缀.h 的头文件不支持把成员函数重载为友元函数，但是它所提供的老形式的带后缀.h 的头文件可以支持此项功能，因此程序的第一行为“#include <iostream.h>”。

12．3．2　重载输入运算符“>>”

通过重载输入运算符“>>”可以实现用户自定义类型的输入。定义重载输入运算符函数与重载输出运算符函数的格式基本相同，只是要把 ostream 换成 istream，把“<<”用“>>”代替；此外，定义输入运算符函数时，函数中第二个参数必须是一个引用。完整格式如下：

```
istream &operator>> (istream &stream , user_type    &obj)
{
    //   操作代码
    return    stream;
}
```

与重载输出运算符函数一样，重载输入运算符也不能是所操作的类的成员函数，但可以是该类的友元函数或独立函数。

例 12-6　输入运算符的重载

```
#include <iostream.h>
class Point
{
public:
    Point(int i=0,int j=0)
    { x=i;y=j;}
    friend istream &operator>>(istream &in,Point &a);
    friend ostream &operator<<(ostream &out,Point &a);
private:
    int x,y;
};
istream &operator>>(istream &in,Point &a)
{
    cout<<"Enter the x and y :";
    in>>a.x;
    in>>a.y;
    return in;
}
ostream &operator<<(ostream &out,Point &a)
```

```
{
    out<<a.x<<","<<a.y<<endl;
    return out;
}
int main()
{
    Point a(10,10);
    cout<<a;
    cin>>a;
    cout<<a;
    return 0;
}
```

程序运行情况如下：

```
10,10
Enter the x and y :20 20
20,20
```

在这个程序中，定义了重载输出运算符和输入运算符，它们都是类 Point 的友元函数，分别完成对该类对象的输入和输出操作。在定义重载输入运算符时，参数 a 前面的&一定不能省略，否则将得到错误的结果。

12．4　文件流

文件一般是指放在外部介质上的数据的集合。一批数据以文件的形式存放在外部介质（如磁盘）上。操作系统以文件为单位对数据进行管理，也就是说，如果想查找存放在外部介质上的数据，必须先按文件名找到所指定的文件，然后再从该文件中读取数据。而要把数据存储在外部介质上，也必须先建立一个文件（以文件名标识），才能向它输出数据。

前面介绍的输入输出，都是以终端为对象的，即从终端键盘输入数据，运行结果输出到终端显示器上。从操作系统的角度来说，每一个与主机相联的输入输出设备都可以看作是一个文件。例如，终端键盘是输入文件，显示屏幕和打印机是输出文件。

C++把文件看作是字符（字节）的序列，即由一个一个字符（字节）的数据顺序组成。根据数据的组织形式，可分为 ASCII 文件和二进制文件。ASCII 文件也称文本（text）文件，它的每个字节存放一个 ASCII 代码，代表一个字符。二进制文件则是把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。假定有一个整数 1949，在内存中占两个字节，如果按 ASCII 形式输出到磁盘上，则占 4 个字节；而如果按二进制形式输出，则在磁盘上只占两个字节，如图 12-2 所示。用 ASCII 形式输出时，与字符一一对应，一个字节代表一个字符，因而便于对字符进行逐个处理，也便于输出字符，缺点是占存储空间较多，而且要花费时间进行二进制形式与 ASCII 形式之间的转换。用二进制形式输出数值数据，可以节省外存空间和转换时间，但一个字节不能对应一个字符，不能直接输出字符形式。对于

需要暂时保存在外存上以后又需要输入到内存的中间结果数据，通常用二进制形式保存。

在 C++中，为了进行文件操作，必须首先建立一个文件流，然后把这个流和实际的文件相关联，这称为打开文件；文件打开后，可以按要求进行读写操作，一般来说，在主存与外设的数据传输中，由主存到外设叫做输出或写，而由外设到主存叫做输入或读；完成输入输出后，还必须将已打开的文件关闭，即取消文件和流的关联。

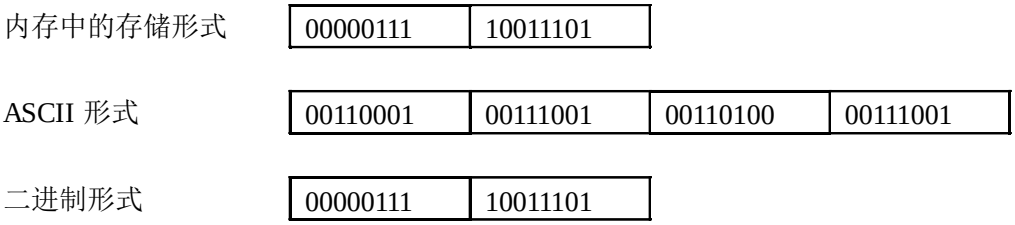

图 12-2　数据的 ASCII 形式和二进制形式

12．4．1　文件的打开和关闭

1．文件的打开

文件使用时，必须先打开，后使用。打开文件时必须指定文件的打开方式。

C++为文件输入输出提供了 3 个类：ifstream、ofstream 和 fstream。与这三个类相对应，C++为文件输入输出提供了三种类型的流，即输入流、输出流以及输入/输出流。也就是说，为了打开一个输入流，必须把它定义为类 ifstream 的对象；要打开一个输出流，必须把它定义为类 ofstream 的对象；而为了打开既有输入、又有输出的流，则必须把它定义为类 fstream 的对象。例如：

```
ifstream    in ;      // 输入
ofstream    out;     // 输出
fstream     both;    // 输入和输出
```

将产生一个输入流(in)，一个输出流(out)和一个既可输入、又可输出的流(both)。

文件的打开有两种方式，一种是用输入输出流的成员函数打开，另一种是用构造函数打开。

（1）用流 ifstream、ofstream 和 fstream 类对象的成员函数方式打开文件。open()函数是这三个流类的成员函数，其原型为：

```
void    open(const unsigned char * , int    mode , int access=filebuf::openprot);
```

其中第一个参数用来传递文件名，第二个参数 mode 的值决定文件的打开方式，第三个参数决定文件的保护方式，用户通常只使用缺省值。

打开方式是已经在 ios 类中定义了的枚举常量。常用的有：

ios::in　　打开一个文件用于输入（读）操作。

ios::out　　打开一个文件用于输出（写）操作

ios::ate　　打开一个已有的文件，文件指针指向文件尾。

ios::app　　以输出方式打开文件，写入的数据添加在文件末尾

ios::trunc　　打开一个文件。如果文件已存在，则删除其中全部数据；如果文件不存

在，则建立新文件。如果已指定了 ios::out，而未指定 ios::app、ios::ate、ios::in，则同时默认此方式。

ios::nocreate　打开一个已有的文件，如果文件不存在，则打开文件失败（即不创建一个新文件）。

ios::noreplace　如果文件不存在则建立新文件，如果文件存在，则打开失败。noreplace是不更新原有文件的意思。

ios::binary　　指定文件以二进制方式打开，而不是缺省说明的文本方式。

下面对文件打开方式作进一步的说明：

① 每一个打开的文件都有一个文件指针，该指针的初始位置由打开方式指定，文件每次读写都从文件指针的当前位置开始。每读入一个字节，指针就后移一个字节。当文件指针移到最后，就会遇到文件结束 EOF（文件结束符也占一个字节，其值为-1），此时流对象的成员函数 eof()的值为非 0 值（一般设为 1），表示文件结束。

② 用“ios::io”方式打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件必须已经存在，不能打开一个并不存在的用于“ios::in”方式的文件，否则会出错。如果用类 ifstream 来产生一个流，则隐含为输入流，不必再说明打开方式。如：

ifstream fin；

fin.open（"文件名"）；

此时打开的是一个输入文件，打开方式可以省略，其缺省值为 ios::in。

③ 用“ios::out”方式打开文件，表示可以向该文件输出数据。如果用类 ofstream 来产生一个流，则隐含为输出流，不必再说明打开方式。如：

ofstream fout；

fout.open（"文件名"）；

此时打开的是一个输出文件，打开方式可以省略，其缺省值为 ios::out。

④ 如果希望向文件末尾添加数据（不删除原有数据），则应当用“ios::app”方式打开文件，但此时文件必须存在。打开时，文件指针移到文件尾部。用这种方式打开的文件只能用于输出。

⑤ 用“ios::ate”方式打开一个已存在的文件时，文件指针自动位于原有文件的尾部。

⑥ 一般情况下，当用 open（）函数打开文件时，如果文件存在，则打开该文件，否则建立该文件。但当用“ios::nocreate”方式打开文件时，表示不建立新文件，在这种情况下，如果要打开的文件不存在，则函数 open（）调用失败。相反，如果使用“ios::noreplace”方式打开文件，则表示不修改原来文件，而是要建立新文件。因此，如果文件已经存在，则函数 open（）调用失败。

⑦ 如果使用“ios::binary”方式，则以二进制方式打开文件。不以“ios::binary”方式打开的文件均以文本方式打开。在用文本文件向计算机输入时，把回车和换行两个字符转换为一个换行符，而在输出时把换行符转换为回车和换行两个字符。对于二进制文件则不进行这种转换，在内存中的数据形式与输出到外部文件中的数据形式完全一致，一一对应。

⑧ 打开方式可以用“|”组合。例如，“ios::in|ios::out|ios::binary”表示打开的文件可以进行二进制的读和写。但不能组合互相排斥的方式，如“ios::nocreate|ios::noreplace”。

（2）用构造函数打开文件。用构造函数打开文件的格式是：

ifstream 对象名("文件名")

ofstream 对象名("文件名")

fstream 对象名("文件名"，打开方式)

例如，语句

ofstream fout("mytest.txt")；

它相当于语句：

ofstream fout；

fout.open("mytest.txt")；

只有在打开文件之后，才能对文件进行读写操作。如果由于某些原因打不开文件（即执行函数 open()失败），则流变量的值将为 0。因此，在对文件进行操作前，必须确保文件已打开，否则要进行相应的处理。为了确认打开一个文件是否成功，可以使用类似下面的办法进行检测：

```
ifstream fin("mytest.txt") ； // 打开输入文件
if ( ! fin)
{
    cout<<"Cannot open file! \n" ；
    // 错误处理代码
}
```

为了确保文件操作的顺利进行，在打开文件时通常都要判断是否成功。

2．文件的关闭

当对一个打开的文件操作完毕后，应及时调用文件流的成员函数来关闭相应的文件。具体格式为：

流对象名．close（）；

其中流对象名为待关闭的文件流的对象名。Close()函数不带任何参数，也不返回任何值。

文件流的对象通过文件打开函数，建立起文件名与该对象的联系后，就可对文件进行读、写操作，文件关闭后，就断开了文件流与对象间的联系。如果要对文件再次进行读、写，就要重新打开文件。

12．4．2 文件的读写

当文件打开以后，即文件与流建立了联系后，就可以进行读写操作了。

1．文本文件的读写

对文本文件的读写操作可以用以下两种方法：

（1）用流输出运算符“<<”和流输入运算符“>>”输入输出标准类型的数据。“<<”和“>>”都已在 iostream 中被重载为能用于 ostream 和 istream 类对象的标准类型的输入输出。由于 ifstream 和 ofstream 分别是 istream 和 ostream 类的派生类，因此它们从 ostream 和 istream 类继承了公用的成员函数，所以在对磁盘文件的操作中，可以通过文件流对象和运算符“<<”和“>>”实现对磁盘文件的读写，如同用 cin、cout 和<<、>>对标准设备进行读写一样，只是必须用与文件相连接的文件流代替 cin 和 cout。

例 12-7　生成一个 sqrtable.txt 文件，该文件内容为 100 以内整数的平方根表，结果精确到小数点后 4 位。

```
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
    ofstream table;
    table.open("sqrtable.txt");
    int i,j;
    if(! table)
    {
        cout<<"Cannot open output file!\n";
        return 1;
    }
    table<<"     |";
    for (i=0;i<10;i++)
        table<<setw(5)<<i<<"     ";
    table<<endl;
    table<<"---+";
    for(i=0;i<10;i++)
        table<<"--------";
    table<<fixed<<setprecision(4);
    for (i=0;i<10;i++)
    {
        table<<endl;
        table<<setw(2)<<i<<" |";
        for(j=0;j<10;j++)
            table<<setw(8)<<sqrt(10*i+j);
    }
    table.close();
    return 0;
}
```

程序运行后，文件 sqrtable.txt 中的内容如下（用记事本打开查看）：

```
  |     0         1        2         3         4        5        6        7         8         9
---+--------------------------------------------------------------------------------
 0 |  0.0000   1.0000   1.4142   1.7321   2.0000   2.2361   2.4495   2.6458   2.8284   3.0000
```

```
1 |  3.1623  3.3166  3.4641  3.6056  3.7417  3.8730  4.0000  4.1231  4.2426  4.3589
2 |  4.4721  4.5826  4.6904  4.7958  4.8990  5.0000  5.0990  5.1962  5.2915  5.3852
3 |  5.4772  5.5678  5.6569  5.7446  5.8310  5.9161  6.0000  6.0828  6.1644  6.2450
4 |  6.3246  6.4031  6.4807  6.5574  6.6332  6.7082  6.7823  6.8557  6.9282  7.0000
5 |  7.0711  7.1414  7.2111  7.2801  7.3485  7.4162  7.4833  7.5498  7.6158  7.6811
6 |  7.7460  7.8102  7.8740  7.9373  8.0000  8.0623  8.1240  8.1854  8.2462  8.3066
7 |  8.3666  8.4261  8.4853  8.5440  8.6023  8.6603  8.7178  8.7750  8.8318  8.8882
8 |  8.9443  9.0000  9.0554  9.1104  9.1652  9.2195  9.2736  9.3274  9.3808  9.4340
9 |  9.4868  9.5394  9.5917  9.6437  9.6954  9.7468  9.7980  9.8489  9.8995  9.9499
```

（2）用文件流的 put、get 和 getline 等成员函数进行字符的输入输出。

成员函数 put 用于输出一个字符，格式为：

输出流对象.put(ch);

调用该函数的结果是向输出流中插入（输出）一个字符 ch。

成员函数 get 用于读入一个字符，它有 3 种形式：无参数的，有一个参数的，有三个参数的。

不带参数的调用形式为：

输入流对象.get()；

用来从指定的输入流中提取一个字符（包括空白字符），函数的返回值就是读入的字符。若遇到输入流中的文件结束符，则函数值返回文件结束标志 EOF（End Of File）。

有一个参数的调用形式为：

输入流对象.get(ch);

用来从输入流中读取一个字符，赋给字符变量 CH。如果读取成功则函数返回非 0 值（真），若失败（遇到文件结束符）则函数返回 0 值（假）。

有三个参数的调用形式为：

输入流对象.get(字符数组或字符指针，字符个数 n，终止字符)；

作用是从输入流中读取 n-1 个字符，赋给指定的字符数组（或字符指针指向的数组），如果在读取 n-1 个字符之前遇到指定的终止字符，则提前结束读取。如果读取成功则函数返回非 0 值（真），若失败（遇到文件结束符）则函数返回 0 值（假）。

成员函数 getline 用于读入一行字符，其用法与带 3 个参数的 get 函数类似，即：

输入流对象.getline(字符数组或字符指针，字符个数 n，终止字符)；

例 12-8　将文件 sqrtable.txt 复制为文件 mttable.txt。

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    char c;
    ifstream fin("sqrtable.txt");
    if(! fin)
```

```
    {
        cout<<"Cannot open input file!\n";
        return 1;
    }
    ofstream fout("mytable.txt");
    if(! fout)
    {
        cout<<"Cannot open output file!\n";
        return 1;
    }
    while(fin.get(c))
        fout.put(c);
    return 0;
}
```

程序中，fin 和 fout 分别是两个流类 ifstream 和 ofstream 的对象，通过构造函数分别和当前目录下的两个文件相连接，同时打开了 sqrtable.txt 和 mytable.txt 这两个文件。这两个文件都是文本文件，分别采用缺省的访问模式 ios::in 和 ios::out。通过 while 循环将第一个文件的内容复制到第二个文件中，在程序运行结束时，两个流对象的生命期结束，由系统自动调用析构函数将文件关闭，所以用构造函数打开的文件可以不显式地关闭文件。

2．二进制文件的读写

二进制文件不是以 ASCII 代码存放数据的，它将内存中数据存储形式不加转换地传送到磁盘文件，因此它又称为内存数据的映像文件。

对二进制文件的读写主要用 istream 类的成员函数 read 和 ostream 类的成员函数 write 来实现的。这两个函数的原型如下：

```
istream &read(char *buffer,int len);
ostream &write(const char *buffer,int len);
```

read 是流类 istream 中的成员函数，其功能为：从相应的流中读取 len 个字节，并把它们放入指针 buffer 所指向的缓冲区中。该函数有两个参数，其中第一个参数 buffer 是一个指针，它是读入数据的存放地址（起始地址）；第二个参数 len 是一个整数值，它是要读入的数据的字节数。其调用格式为：read（缓冲区首址，读入的字节数），其中，“缓冲区首址”的数据类型为 char *，当输入其他类型数据时，必须进行类型转换。例如：

```
int arrryi[]＝{100，200，300};
read((char *)&arrayi[0]，sizeof(arrayi));
```

该例定义了一个整型数组 arrayi，为了读入它的全部数据，必须在 read 函数中给出它的首地址，并把它转换为 char *类型，要读入的字节数由 sizeof 确定。

write 是流类 ostream 的成员函数，利用该函数，可以从 buffer 所指的缓冲区把 len 个字节写到相应的流上。参数的含义及调用注意事项与 read 函数类似。

例 12-9　将 1～1000 之间的所有素数以二进制形式存放在磁盘文件 Prime.dat 中。

```
#include <fstream>
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    fstream table;
    table.open("Prime.dat",ios::binary|ios::out);
    if(! table)
    {
        cout<<"Cannot open output file!\n";
        return 1;
    }
    int i,k;
    for (i=2;i<=1000;i++)
    {
        for (k=2;k<=sqrt(i);k++)
            if (i%k==0) break;
        if(k>sqrt(i))
            table.write((char *)&i,sizeof(i));
    }
    table.close();
    return 0;
}
```

程序中用成员函数 write 向 Prime.dat 输出数据。需要注意的是，运算符“>>”和“<<”不能用于二进制文件的输入和输出。如果将程序中的“table.write((char *)&i,sizeof(i));”一句改写为“table<<i;”，用运算符“<<”向输出文件流输出，程序仍能运行，但结果是以文本文件的方式保存，而不是以二进制方式保存。

例 12-10　将上例得到的 Prime.dat 中的数据读入内存并在显示器上显示。

```
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    fstream table;
    table.open("Prime.dat",ios::binary|ios::in);
    if(! table)
    {
```

```
        cout<<"Cannot open input file!\n";
        return 1;
    }
    int i;
    while(!table.eof())
    {
        table.read((char *)&i,sizeof(i));
        cout<<setw(5)<<i;
    }
    cout<<endl;
    table.close();
    return 0;
}
```

3．文件的随机读写

上面介绍的文件操作都是按一定顺序进行读写的，因此称为顺序文件。对于顺序文件来说，只能按实际排列的顺序，一个一个地访问文件中的各个元素。也就是说，在访问完第 I 个元素之后，只能访问第 I+1 个元素，既不能访问第 I+2 个元素，也不能访问第 I-1 个元素。在文件中有一个指针，它指向当前读写的位置。当顺序读写一个文件时，每次读写一个字符（字节）。读写完一个字符（字节）后，该位置指针自动移动，指向下一个字节位置。

与顺序文件不同，随机存取文件（又称直接存取文件）在访问文件中的元素时，不必考虑各个元素的排列次序或位置，可以根据需要访问文件中的任一个元素。

为了进行随机存取，必须先确定文件指针的位置。在 C++中，可以用以下几个函数来确定指文件指针位置。

（1）seekg(pos);

seekp(pos);

这两个函数都是从文件头开始，把文件指针向后移动 pos 个字符（字节）。其中 seekg 用于输入文件，而 seekp 用于输出文件。参数 pos 是相对于文件头的位移量，其类型是 streamoff，这种类型在头文件 iostream 中定义如下：

typedef streamoff long；

可以看出，它实际上是 long 型。

（2）seekg(pos,orgin);

seekp(pos，orgin);

这两个函数实际上是前两个函数的重载函数。前两个函数从文件头移动文件指针，而这两个函数则是从指定的位置开始移动文件指针。参数 origin 表示文件指针的起始位置，pos 表示相对于这个起始位置的位移量。origin 的取值有以下三种情况：

ios::beg　　从文件头开始，把文件指针移动由 pos 指定的距离

ios::cur　　从当前位置开始．把文件指针移动由 pos 指定的距离

ios::end　　从文件尾开始，把文件指针移动由 pos 指定的距离

当 origin 为 ios::beg 时，pos 的值为正数；当 origin 为 ios::end 时，pos 的值为正数；而当 origin 为 ios::cur 时，pos 的值可以为正数，也可以为负数。正数时从前向后移动文件指针，负数时从后向前移动文件指针。

seekg 函数用输入文件，而 seekp 函数用于输出文件。

（3）tellg()

tellp()

这两个函数用来返回文件指针的当前位置。其中 tellg 函数用于输入文件，tellp 函数用于输出文件。

对文件进行随机读写关键在于控制文件的位置指针。即为了能进行随机读写，必须把文件指针移到适当的位置。例如：

seekg(100)；　　把输入文件的位置指针移到离文件头 100 个字节处；

seekp(80)；　　把输出文件的位置指针移到离文件头 80 个字节处；

seekg(100，ios::cur)；　　把输入文件指针移到当前位置后 100 个字节处；

seekp(-80，ios::end)；　　把输出文件指针向前移到离文件尾 80 个字节处。

确定了指针位置后，其读写操作与顺序文件相同。

例 12-11　有 5 个联系人的电话号码，编写一个程序，要求：（1）将它们保存到磁盘文件 phonenumber.dat 中；（2）修改文件中指定的第 n 个联系人的信息；（3）将文件中的数据显示出来。

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
struct TelephoneBook
{
    char name[8];
    char code[20];
};

int main()
{
    TelephoneBook entry[5]={"郭靖","02788612788","章一丰","075542618865",
                            "黄蓉","13554332875","李力锋","1790901062778844",
                            "孙宏","02762567802"};
    TelephoneBook entryTemp;
    fstream books;
    int i;
    char name[8],number[20];
    books.open("phonenumber.dat",ios::out|ios::binary);
```

```
        if(! books)
        {
            cout<<"Open error!\n";
            return 1;
        }
        for(i=0;i<5;i++)
            books.write((char *)&entry[i],sizeof(entry[i]));
        books.close();
        books.open("phonenumber.dat",ios::in|ios::out|ios::binary);
        cout<<"请输入待修改的记录号（从 1 开始）：";
        cin>>i;
        if (i>5)
            cout<<"输入的记录号超过了电话本中记录条数！"<<endl;
        else
        {
            books.seekg((i-1)*sizeof(TelephoneBook),ios::beg);   // 定位到待修改记录
            books.read((char *)&entryTemp,sizeof(entryTemp));   //读取待修改记录
            cout<<"姓名："<<entryTemp.name<<"   输入新姓名（不修改直接输入'/'后回
车）：";
            cin.clear();
            cin>>name;
            cout<<"号码："<<entryTemp.code<<"   输入新号码（不修改直接输入'/'后回
车）：";
            cin>>number;
            if(name[0]!='/')
                strcpy(entryTemp.name,name);
            if(number[0]!='/')
                 strcpy(entryTemp.code,number);
            books.seekp((i-1)*sizeof(TelephoneBook),ios::beg);   // 定位到待修改记录
            books.write((char *)&entryTemp,sizeof(TelephoneBook));   // 修改记录
        }
        books.seekg(0,ios::beg);     // 重新定位于文件开头
        for (i=0;i<5;i++)
        {
            books.seekg(i*sizeof(TelephoneBook),ios::beg);
            books.read((char *)&entryTemp,sizeof(entryTemp));
            cout<<entryTemp.name<<" "<<entryTemp.code<<endl;
        }
        books.close();
```

```
    return 0;
}
```
程序运行情况如下：

```
请输入待修改的记录号（从 1 开始）：3
姓名：黄蓉　输入新姓名（不修改直接输入'/'后回车）：/
号码：13554332875　输入新号码（不修改直接输入'/'后回车）：13077186578
郭靖  02788612788
章一丰  075542618865
黄蓉  13077186578
李力锋  1790901062778844
孙宏  02762567802
```

本章小结

在 C++中，输入输出流被定义为类。C++的 IO 库中的类称为流类（stream class）。用流类定义的对象称为流对象。cin 和 cout 就是 iostream_withassign 类的对象。

C++的 I / O 流类库是用继承方法建立起来的一个输入输出类库，它具有两个平行类：即 streambuf 类和 ios 类，这两个类是基本类，所有的流类都可以由它们派生出来。

为了进行格式化的输入输出，C++提供了两种格式控制方法：一种方法是使用 ios 类中有关格式控制的成员函数；另一种方法是使用称为控制符的特殊类型的函数。

C++创建自己的流库的主要目的，就是使用户自定义类型数据的输入输出也能像系统预定义类型数据的输入输出一样简单、方便，这通过重载输入运算符“>>”和输出运算符“<<”来实现的。

通过重载输出运算符“<<”可以实现用户自定义类型的输出。定义输出运算符“<<”重载函数的一般格式如下：

```
ostream    &operate<< (ostream &stream , user_type    obj)
{
    //    操作代码
    return    stream;
}
```

定义输入运算符“>>”重载函数的一般格式如下：

```
istream &operator>> (istream &stream , user_type    &obj)
{
    //    操作代码
    return    stream;
}
```

在 C++中，为了进行文件操作，必须首先建立一个文件流，然后把这个流和实际的文件相关联，这称为打开文件；文件打开后，可以按要求进行读写操作，一般来说，在主存

与外设的数据传输中，由主存到外设叫做输出或写，而由外设到主存叫做输入或读；完成输入输出后，还必须将已打开的文件关闭，即取消文件和流的关联。

文件的打开有两种方式，一种是用输入输出流的成员函数打开，另一种是用构造函数打开。

当对一个打开的文件操作完毕后，应及时调用文件流的成员函数来关闭相应的文件。具体格式为：

流对象名．close（）；

习　　题

1．C++中有哪四个预定义的标准流？它们分别与什么具体设备相关联？

2．写出下列程序的运行结果

（1）
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
   int i=100;
   double d=123.456789;
   cout<<i<<" "<<d<<endl;
   cout<<hex<<i<<endl;
   cout<<oct<<i<<endl;
   cout<<setfill('*')<<setw(10)<<i<<"Hi"<<endl;
   cout<<dec<<i<<" "<<d<<endl;
   cout<<setw(10)<<d<<endl;
   cout<<setw(8)<<setprecision(10)<<d<<endl;
   return 0;
}
```

（2）
```
#include <iostream>
using namespace std;
int main()
{
   cout<<"x_width="<<cout.width()<<endl;
   cout<<"x_fill="<<cout.fill()<<endl;
   cout<<"x_precision="<<cout.precision()<<endl;
   cout<<123<<" "<<123.45678<<endl;
   cout.width(10);
```

```
    cout.fill('&');
    cout.precision(7);
    cout<<123<<" "<<123.45678<<endl;
    cout.setf(ios::left);
    cout<<123<<" "<<123.45678<<endl;
    cout.width(6);
    cout<<123<<" "<<123.45678<<endl;
    return 0;
}
```

（3）
```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("myfile.txt");
    if(!in)
    {
        cout<<"Can't open myfile."<<endl;
        return 1;
    }
    char ch;
    in.seekg(2,ios::beg);
    if (in.get(ch))
        cout<<ch<<endl;
    in.seekg(11,ios::cur);
    if (in.get(ch))
        cout<<ch<<endl;
    in.seekg(-14,ios::end);
    if (in.get(ch))
        cout<<ch<<endl;
    in.close();
    return 0;
}
```

3．编写程序，重载运算符“<<”和“>>”，使用户能直接输入和输出复数。

4．编写一个程序，定义一个分数类，重载运算符“<<”和“>>”，使得用户能直接输入/输出由分子和分母组成的分数。输入时先输入分子，然后输入空格，最后输入分母，应该在输入过程约去分子与分母的最大公约数。输出时应该考虑分子为零、分母为 1 等特殊情况。

5．编写一个程序，统计文本文件 abc.txt 中的字符个数，并将文件中的所有行加上行号后写到 newabc.txt 文件中。

6．在屏幕上分别以正序和倒序显示任意一个文本文件的内容。

7．函数 Parser 的原型是：

```
const    char * Parser(const char *filename);
```

函数接收一个 C++源程序文件名，检查该文件源程序中的括弧是否匹配，括弧包括（）、[ ]、{　}三种。不同括弧互相嵌套时必须完整嵌套，也就是说如{…[…]…}这样的嵌套是错误的（尽管是配对的）。函数在发现第一个错误或扫描完整个文件时结束，并返回一个反映扫描结果的字符串，例如：“文件…括弧嵌套正确！”或“文件…第…行的…未能正确嵌套！”。

试设计并实现这个函数。

8．编写一个程序：

① 在二进制文件 test.dat 中写入 10 条记录，显示其内容。

② 删除指定的第 n 条记录，并显示删除记录后的文件内容。

③ 按照记录号查询记录并显示结果。