第13章 输入输出流

13.1 C++的输入和输出

13.1.1 输入输出的含义

以前所用到的输入和输出，都是以终端为对象的，即从键盘输入数据，运行结果输出到显示器屏幕上。从操作系统的角度看，每一个与主机相连的输入输出设备都被看作一个文件。除了以终端为对象进行输入和输出外，还经常用磁盘(光盘)作为输入输出对象，磁盘文件既可以作为输入文件，也可以作为输出文件。

程序的输入指的是从输入文件将数据传送给程序，程序的输出指的是从程序将数据传送给输出文件。C++的输入与输出包括以下3方面的内容：

（1） 对系统指定的标准设备的输入和输出。即从键盘输入数据，输出到显示器屏幕。这种输入输出称为标准的输入输出，简称标准**I/O**。

（2） 以外存磁盘文件为对象进行输入和输出，即从磁盘文件输入数据，数据输出到磁盘文件。以外存文件为对象的输入输出称为文件的输入输出，简称文件**I/O**。

（3） 对内存中指定的空间进行输入和输出。通常指定一个字符数组作为存储空间(实际上可以利用该空间存储任何信息)。这种输入和输出称为字符串输入输出，简称串**I/O**。

C++采取不同的方法来实现以上**3**种输入输出。

为了实现数据的有效流动，**C++**系统提供了庞大的**I/O**类库，调用不同的类去实现不同的功能。

13.1.2 C++的I/O对C的发展——类型安全和可扩展性

在C语言中，用printf和scanf进行输入输出，往往不能保证所输入输出的数据是可靠的、安全的。

在C++的输入输出中，编译系统对数据类型进行严格的检查，凡是类型不正确的数据都不可能通过编译。因此C++的I/O操作是类型安全(type safe)的。

C++的I/O操作是可扩展的，不仅可以用来输入输出标准类型的数据，也可以用于用户自定义类型的数据。C++对标准类型的数据和对用户声明类型数据的输入输出，采用同样的方法处理。

C++通过I/O类库来实现丰富的I/O功能。C++的输入输出优于C语言中的printf和scanf，但是比较复杂，要掌握许多细节。

13.1.3 C++的输入输出流

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，在输出操作时，字节流从内存流向输出设备(如屏幕、打印机、磁盘等)。流中的内容可以是ASCII字符、二进制形式的数据、图形图像、数字音频视频或其他形式的信息。

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。流是与内存缓冲区相对应的，或者说，缓冲区中的数据就是流。

在**C++**中，输入输出流被定义为类。**C++**的**I/O**库中的类称为流类**(stream class)**。用流类定义的对象称为流对象。

cout和**cin**并不是**C++**语言中提供的语句，它们是**iostream**类的对象，在未学习类和对象时，在不致引起误解的前提下，为叙述方便，把它们称为**cout**语句和**cin**语句。

在学习了类和对象后，我们对**C++**的输入输出应当有更深刻的认识。

1. iostream类库中有关的类

C++编译系统提供了用于输入输出的iostream类库。iostream这个单词是由3个部分组成的，即i-o-stream，意为输入输出流。在iostream类库中包含许多用于输入输出的类。常用的见书中表13.1。

ios是抽象基类，由它派生出istream类和ostream类，两个类名中第1个字母i和o分别代表输入(input)和输出(output)。istream类支持输入操作，ostream类支持输出操作，iostream类支持输入输出操作。iostream类是从istream类和ostream类通过多重继承而派生的类。其继承层次见图13.1表示。

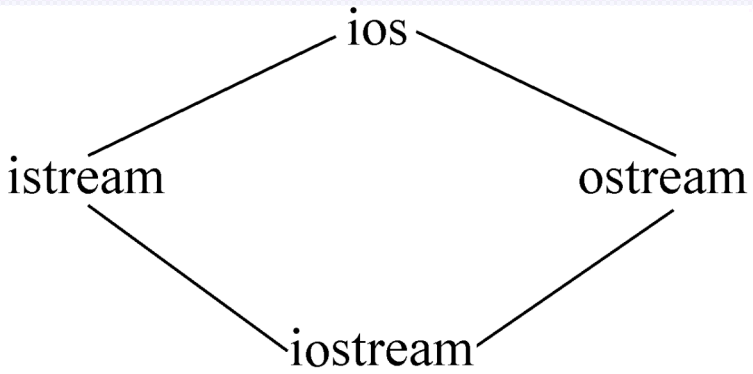

图13.1

C++对文件的输入输出需要用**ifstream**和**ofstream**类，两个类名中第**1**个字母**i**和**o**分别代表输入和输出，第**2**个字母**f**代表文件**(file)**。**ifstream**支持对文件的输入操作，**ofstream**支持对文件的输出操作。类**ifstream**继承了类**istream**，类**ofstream**继承了类**ostream**，类**fstream**继承了类**iostream**。见图**13.2**。

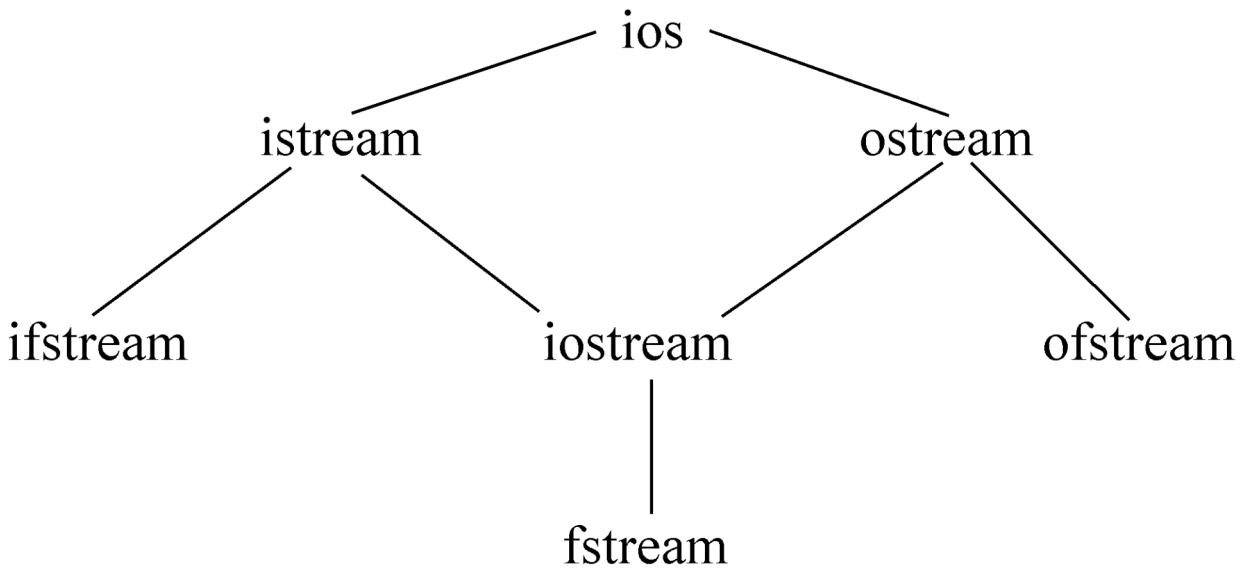

图13.2

I/O类库中还有其他类，见图13.3。

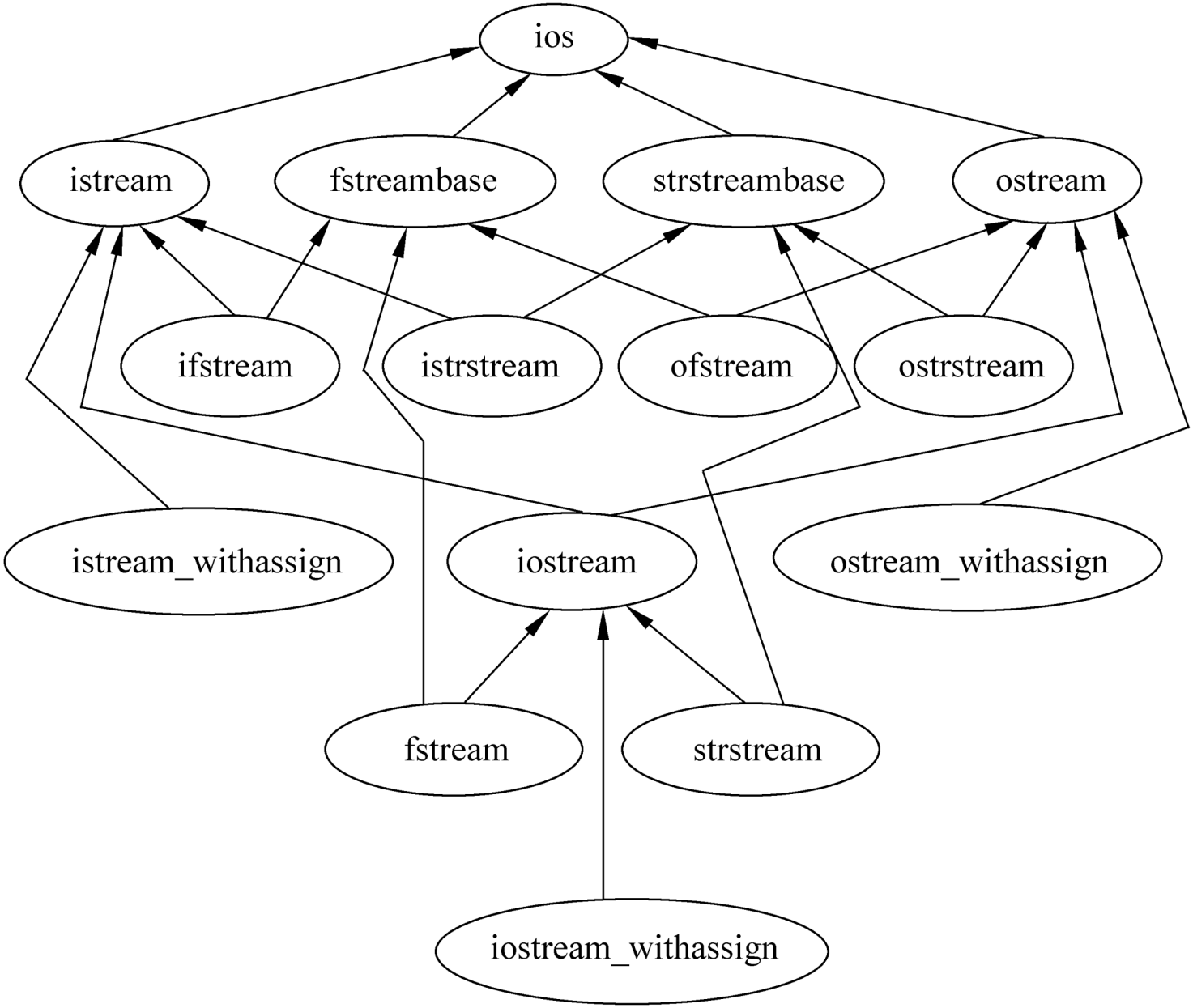
ios
istream
fstreambase
strstreambase
ostream
ifstream
istrstream
ofstream
ostrstream
istream_withassign
iostream
ostream_withassign
fstream
strstream
iostream_withassign

图13.3

2. 与iostream类库有关的头文件

iostream类库中不同的类的声明被放在不同的头文件中，用户在自己的程序中用**#include**命令包含了有关的头文件就相当于在本程序中声明了所需要用到的类。可以换一种说法: 头文件是程序与类库的接口，**iostream**类库的接口分别由不同的头文件来实现。常用的有

- **iostream**包含了对输入输出流进行操作所需的基本信息。
- **fstream**用于用户管理的文件的**I/O**操作。
- **strstream**用于字符串流**I/O**。
- **stdiostream**用于混合使用**C**和**C++**的**I/O**机制时。
- **iomanip**在使用格式化**I/O**时应包含此头文件。

3. 在iostream头文件中定义的流对象

在**iostream**头文件中定义的类有**ios,istream,ostream,iostream,istream _withassign, ostream_withassign,iostream_withassign**等。

iostream.h包含了对输入输出流进行操作所需的基本信息。因此大多数**C++**程序都包括**iostream.h**。在**iostream.h**头文件中不仅定义了有关的类，还定义了**4**种流对象，见书中表**13.2**。

cin是**istream**的派生类**istream_withassign**的对象,它是从标准输入设备(键盘)输入到内存的数据流，称为**cin**流或标准输入流。**cout**是**ostream**的派生类**ostream_withassign**的对象，它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**作用相似，均为向输出设备(显示器) 输出出错信息。因此用键盘输入时用**cin**流，向显示器输出时用**cout**流。向显示器输出出错信息时用**cerr**和**clog**流。

在**iostream**头文件中定义以上**4**个流对象用以下的形式(以**cout**为例)**:**

```
ostream cout (stdout);
```

在定义**cout**为**ostream**流类对象时，把标准输出设备**stdout**作为参数，这样它就与标准输出设备(显示器)联系起来，如果有

```
cout<<3;
```

就会在显示器的屏幕上输出**3**。

4. 在iostream头文件中重载运算符

“<<”和“>>”本来在C++中是被定义为左位移运算符和右位移运算符的，由于在iostream头文件中对它们进行了重载，使它们能用作标准类型数据的输入和输出运算符。所以，在用它们的程序中必须用#include命令把iostream包含到程序中。

```
#include <iostream>
```

在istream和ostream类(这两个类都是在iostream中声明的)中分别有一组成员函数对位移运算符“<<”和“>>”进行重载，以便能用它输入或输出各种标准数据类型的数据。对于不同的标准数据类型要分别进行重载，如

```
ostream operator << (int );//用于向输出流插入一个int数据
ostream operator << (float ); //用于向输出流插入一个float数据
```

ostream operator << (char); //用于向输出流插入一个**char**数据
ostream operator << (char *); //用于向输出流插入一个字符串数据等。

如果在程序中有下面的表达式:

cout<<"C++";

根据第**11**章所介绍的知识，上面的表达式相当于

cout.operator<<("C++")

"C++"的值是其首字节地址，是字符型指针**(char*)**类型，因此选择调用上面最后一个运算符重载函数，通过重载函数的函数体，将字符串插入到**cout**流中，函数返回流对象**cout**。

在**istream**类中已将运算符“**>>**”重载为对以下标准类型的提取运算符**: char,signed char,unsigned char,short,unsigned short,int,unsigned int,**

long,unsigned long,float, double,long double, char*,signed char*,unsigned char*等。

在**ostream**类中将“**<<**”重载为插入运算符，其适用类型除了以上的标准类型外，还增加了一个**void***类型。

如果想将“**<<**”和“**>>**”用于自己声明的类型的数据，就不能简单地采用包含**iostream**头文件来解决，必须自己用第**11**章的方法对“**<<**”和“**>>**”进行重载。

怎样理解运算符“**<<**”和“**>>**”的作用呢？它们指出了数据移动的方向，例如

>>a

箭头方向表示把数据放入**a**中。而

<<a

13.2 标准输出流

标准输出流是流向标准输出设备(显示器)的数据。

13.2.1 cout，cerr和clog流

ostream类定义了**3**个输出流对象，即**cout,cerr,clog**。分述如下。

1. cout流对象

cout是**console output**的缩写，意为在控制台(终端显示器)的输出。

(1) cout不是**C++**预定义的关键字，它是**ostream**流类的对象，在**iostream**中定义。

(2) 用**“cout<<”**输出基本类型的数据时，可以不必考虑数据是什么类型，系统会判断数据的类型，并根据其类型选择调用与之匹配的运算符重载函数。

(3) cout流在内存中对应开辟了一个缓冲区，用来存放流中的数据，当向**cout**流插入一个**endl**时，不论缓冲区是否已满，都立即输出流中所有数据，然后插入一个换行符，并刷新流(清空缓冲区)。

(4) 在**iostream**中只对“**<<**”和“**>>**”运算符用于标准类型数据的输入输出进行了重载，但未对用户声明的类型数据的输入输出进行重载。

2. cerr流对象

cerr流对象是标准错误流。**cerr**流已被指定为与显示器关联。**cerr**的作用是向标准错误设备**(standard error device)**输出有关出错信息。**cerr**与标准输出流**cout**的作用和用法差不多。但有一点不同**: cout**流通常是传送到显示器输出，但也可以被重定向输出到磁盘文件，而**cerr**流中的信息只能在显示器输出。当调试程序时，往往不希望程序运行时的出错信息被送到其他文件，而要求在显示器上及时输出，这时应该用**cerr**。**cerr**流中的信息是用户根据需要指定的。

例13.1 有一元二次方程$ax^2+bx+c=0$，其一般解为$x_{1,2}=(-b\pm b^2-4ac)/2a$,但若$a=0$，或$b^2-4ac<0$时，用此公式出错。

编程序，从键盘输入a,b,c的值，求x_1和x_2。如果$a=0$或$b^2-4ac<0$，输出出错信息。

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{float a,b,c,disc;
 cout<<"please input a,b,c:";
 cin>>a>>b>>c;
 if (a==0)
cerr<<"a is equal to zero,error!"<<endl;
//将有关出错信息插入cerr流，在屏幕输出
 else
```

```
if ((disc=b*b-4*a*c)<0)
cerr<<"disc=b*b-4*a*c<0"<<endl;  //将有关出错信息插入cerr流，在屏幕输出
else
{cout<<"x1="<<(-b+sqrt(disc))/(2*a)<<endl;
   cout<<"x2="<<(-b-sqrt(disc))/(2*a)<<endl;
}
 return 0;
}
```

运行情况如下：

①

please input a,b,c: 0 2 3↙

a is equal to zero,error!

② please input a,b,c: 5 2 3↙

sc=b*b-4*a*c<0

③please input a,b,c: 1 2.5 1.5↙

x1=-1

x2=-1.5

3. clog流对象

clog流对象也是标准错误流，它是**console log**的缩写。它的作用和**cerr**相同，都是在终端显示器上显示出错信息。区别**: cerr**是不经过缓冲区，直接向显示器上输出有关信息，而**clog**中的信息存放在缓冲区中，缓冲区满后或遇**endl**时向显示器输出。

13.2.2 格式输出

在输出数据时，有时希望数据按指定的格式输出。有两种方法可以达到此目的。一种是第3章已介绍过的使用控制符的方法；第2种是使用流对象的有关成员函数。

1. 使用控制符控制输出格式

输出数据的控制符见书中表13.3。

应当注意，这些控制符是在头文件iomanip中定义的，因而程序中应当包含iomanip。

例13.2 用控制符控制输出格式。

```
#include <iostream>
#include <iomanip>//不要忘记包含此头文件
using namespace std;
int main()
{int a;
 cout<<"input a:";
 cin>>a;
 cout<<"dec:"<<dec<<a<<endl;            //以十进制形式输出整数
 cout<<"hex:"<<hex<<a<<endl;            //以十六进制形式输出整数a
 cout<<"oct:"<<setbase(8)<<a<<endl;     //以八进制形式输出整数a
 char *pt="China";                      //pt指向字符串"China"
 cout<<setw(10)<<pt<<endl;              //指定域宽为10，输出字符串
cout<<setfill('*')<<setw(10)<<pt<<endl; //指定域宽10，输出字符串，空白处以'*'
填充
 double pi=22.0/7.0;                    //计算pi值
 cout<<setiosflags(ios::scientific)<<setprecision(8);//按指数形式输出，8位小数
 cout<<"pi="<<pi<<endl;                 /输出pi值
 cout<<"pi="<<setprecision(4)<<pi<<endl;          //改为4位小数
 cout<<"pi="<<setiosflags(ios::fixed)<<pi<<endl;//改为小数形式输出
 return 0;
}
```

运行结果如下：

input a:34↙(输入a的值)
dec:34 (十进制形式)
hex:22 (十六进制形式)
oct:42 (八进制形式)
China (域宽为10)
*******China (域宽为10，空白处以'*'填充)**
pi=3.14285714e+00 (指数形式输出，8位小数)
pi=3.1429e+00 (指数形式输出，4位小数)
pi=3.143 (小数形式输出，精度仍为4)

2. 用流对象的成员函数控制输出格式

除了可以用控制符来控制输出格式外，还可以通过调用流对象**cout**中用于控制输出格式的成员函数来控制输出格式。用于控制输出格式的常用的成员函数见书中表**13.4**。

流成员函数**setf**和控制符**setiosflags**括号中的参数表示格式状态，它是通过格式标志来指定的。格式标志在类**ios**中被定义为枚举值。因此在引用这些格式标志时要在前面加上类名**ios**和域运算符**“::”**。格式标志见书中表**13.5**。

例13.3 用流控制成员函数输出数据。

```
#include <iostream>
using namespace std;
int main( )
{int a=21
 cout.setf(ios::showbase);//显示基数符号(0x或0)
 cout<<"dec:"<<a<<endl;        //默认以十进制形式输出a
 cout.unsetf(ios::dec);        //终止十进制的格式设置
 cout.setf(ios::hex);          //设置以十六进制输出的状态
 cout<<"hex:"<<a<<endl;        //以十六进制形式输出a
cout.unsetf(ios::hex);         //终止十六进制的格式设置
 cout.setf(ios::oct);          //设置以八进制输出的状态
 cout<<"oct:"<<a<<endl;        //以八进制形式输出a
cout.unseft(ios::oct);
 char *pt="China";             //pt指向字符串"China"
 cout.width(10);               //指定域宽为10
 cout<<pt<<endl;               //输出字符串
 cout.width(10);               //指定域宽为10
```

```
 cout.fill('*');              //指定空白处以'*'填充
 cout<<pt<<endl;              //输出字符串
 double pi=22.0/7.0;          //输出pi值
cout.setf(ios::scientific);   //指定用科学记数法输出
cout<<"pi=";                  //输出"pi="
cout.width(14);               //指定域宽为14
cout<<pi<<endl;               //输出pi值
cout.unsetf(ios::scientific); //终止科学记数法状态
cout.setf(ios::fixed);        //指定用定点形式输出
cout.width(12);               //指定域宽为12
cout.setf(ios::showpos);      //正数输出“+”号
cout.setf(ios::internal);     //数符出现在左侧
cout.precision(6);            //保留6位小数
cout<<pi<<endl;               //输出pi，注意数符“+”的位置
return 0;
}
```

运行情况如下：

dec:21(十进制形式)
hex:0x15 (十六进制形式，以0x开头)
oct:025 (八进制形式，以0开头)
China (域宽为10)
*******China (域宽为10，空白处以'*'填充)**
pi=3.142857e+00 (指数形式输出，域宽14，默认6位小数)**
+*3.142857 (小数形式输出，精度为6，最左侧输出数符“+”)**

13.2.3 用流成员函数put输出字符

ostream类除了提供上面介绍过的用于格式控制的成员函数外，还提供了专用于输出单个字符的成员函数**put**。如

```
cout.put('a');
```

调用该函数的结果是在屏幕上显示一个字符**a**。**put**函数的参数可以是字符或字符的**ASCII**代码(也可以是一个整型表达式)。如

```
cout.put(65+32);
```

也显示字符**a**，因为**97**是字符**a**的**ASCII**代码。

可以在一个语句中连续调用**put**函数。如

```
cout.put(71).put(79).pu(79).put(68).put('\\n');
```

在屏幕上显示**GOOD**。

例13.4 有一个字符串"BASIC"，要求把它们按相反的顺序输出。

```
#include <iostream>
using namespace std;
int main( )
{char *a="BASIC";//字符指针指向'B'
 for(int i=4;i>=0;i--)
  cout.put(*(a+i));                    //从最后一个字符开始输出
 cout.put('\\n');
 return 0;
}
```

运行时在屏幕上输出:

```
CISAB
```

还可以用putchar函数输出一个字符。putchar函数是C语言中使用的，在stdio.h头文件中定义。C++保留了这个函数，在iostream头文件中定义。

例**13.4**也可以改用**putchar**函数实现。

```
#include <iostream>//也可以用#include <stdio.h>，同时不要下一行
using namespace std;
int main( )
{char *a="BASIC";
 for(int i=4;i>=0;i--)
  putchar(*(a+i));
 putchar('\\n');
}
```

运行结果与前相同。

成员函数**put**不仅可以用**cout**流对象来调用，而且也可以用**ostream**类的其他流对象调用。

13.3 标准输入流

标准输入流是从标准输入设备(键盘)流向程序的数据。

13.3.1 cin流

在上一节中已知，在头文件**iostream.h**中定义了**cin,cout,cerr,clog 4**个流对象，**cin**是输入流，**cout,cerr,clog**是输出流。

cin是**istream**类的对象，它从标准输入设备(键盘)获取数据，程序中的变量通过流提取符“**>>**”从流中提取数据。流提取符“**>>**”从流中提取数据时通常跳过输入流中的空格、**tab**键、换行符等空白字符。 注意**:** 只有在输入完数据再按回车键后，该行数据才被送入键盘缓冲区，形成输入流，提取运算符“**>>**”才能从中提取数据。需要注意保证从流中读取数据能正常进行。

例**13.5** 通过测试**cin**的真值，判断流对象是否处于正常状态。

```
#include <iostream>
using namespace std;
int main( )
{float grade;
 cout<<"enter grade:";
 while(cin>>grade)//能从cin流读取数据
{if(grade>=85) cout<<grade<<"GOOD!"<<endl;
if(grade<60) cout<<grade<<"fail!"<<endl;
cout<<"enter grade:";
}
 cout<<"The end."<<endl;
 return 0;
}
```

运行情况如下：

enter grade: 67↙
enter grade: 89↙
89 GOOD!
enter grade: 56↙
56 fail!
enter grade: 100↙
100 GOOD!
enter grade: ^Z↙ //键入文件结束符
The end.

如果某次输入的数据为
enter grade: 100/2↙
输出**"The end."**。

在不同的**C++**系统下运行此程序，在最后的处理上有些不同。以上是在**GCC**环境下运行程序的结果，如果在**VC++**环境下运行此程序，在键入**Ctrl+Z**时，程序运行马上结束，不输出**"The end."**。

13.3.2 用于字符输入的流成员函数

除了可以用**cin**输入标准类型的数据外，还可以用**istream**类流对象的一些成员函数，实现字符的输入。

1. 用get函数读入一个字符

流成员函数**get**有**3**种形式**:** 无参数的，有一个参数的，有**3**个参数的。

(1) 不带参数的**get**函数

其调用形式为

cin.get()

用来从指定的输入流中提取一个字符，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)** 。

例13.6 用get函数读入字符。

```
#include <iostream>
int main( )
{int c;
 cout<<"enter a sentence:"<<endl;
 while((c=cin.get())!=EOF)
  cout.put(c);
 return 0;
}
```

运行情况如下：

```
enter a sentence:
I study C++ very hard.↙(输入一行字符)
I study C++ very hard.          (输出该行字符)
^Z↙(程序结束)
```

C语言中的getchar函数与流成员函数cin.get( )的功能相同，C++保留了C的这种用法。

(2) 有一个参数的**get**函数

其调用形式为

cin.get(ch)

其作用是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**)**，如失败**(**遇文件结束符**)** 则函数返回**0**值**(**假**)**。例**13.6**可以改写如下**:**

```
#include <iostream>
int main( )
{char c;
 cout<<"enter a sentence:"<<endl;
 while(cin.get(c))  //读取一个字符赋给字符变量c，如果读取成功，cin.get(c)为真
{cout.put(c);}
 cout<<"end"<<endl;
 return 0;
}
```

(3) 有**3**个参数的**get**函数

其调用形式为

cin.get(字符数组，字符个数**n**，终止字符**)**

或

cin.get(字符指针，字符个数**n**，终止字符**)**

其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**)**，如失败**(**遇文件结束符**)** 则函数返回**0**值**(**假**)**。再将例**13.6**改写如下**:**

```
#include <iostream>
using namespace std;
int main( )
```

```
{char ch[20];
 cout<<"enter a sentence:"<<endl;
 cin.get(ch,10,'\\n');//指定换行符为终止字符
 cout<<ch<<endl;
 return 0;
}
```

运行情况如下：

```
enter a sentence:
I study C++ very hard.↙
I study
```

get函数中第3个参数可以省写，此时默认为'\\n'。

下面两行等价：

```
cin.get(ch,10,'\\n');
cin.get(ch,10);
```

终止字符也可以用其他字符。如

```
cin.get(ch,10,'x');
```

2. 用成员函数getline函数读入一行字符

getline函数的作用是从输入流中读取一行字符，其用法与带**3**个参数的**get**函数类似。即

cin.getline(字符数组(或字符指针)，字符个数**n**，终止标志字符**)**

例**13.7** 用**getline**函数读入一行字符。

```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
 cout<<"enter a sentence:"<<endl;
 cin>>ch;
 cout<<"The string read with cin is:"<<ch<<endl;
 cin.getline(ch,20,'/');//读19个字符或遇'/'结束
 cout<<"The second part is:"<<ch<<endl;
 cin.getline(ch,20);                    //读19个字符或遇'/n'结束
 cout<<"The third part is:"<<ch<<endl;
 return 0;
}
```

程序运行情况如下：

enter a sentence: I like C++./I study C++./I am happy.↙

The string read with cin is:I

The second part is: like C++.

The third part is:I study C++./I am h

13.3.3 istream类的其他成员函数

除了以上介绍的用于读取数据的成员函数外，**istream**类还有其他在输入数据时用得着的一些成员函数。常用的有以下几种：

1. eof 函数

eof是**end of file**的缩写，表示“文件结束”。从输入流读取数据，如果到达文件末尾(遇文件结束符)，**eof**函数值为非零值(表示真)，否则为**0**(假)。

例**13.8** 逐个读入一行字符，将其中的非空格字符输出。

```
#include <iostream>
using namespace std;
int main( )
{char c;
 while(!cin.eof( ))//eof( )为假表示未遇到文件结束符
if((c=cin.get( ))!=' ')          //检查读入的字符是否为空格字符
    cout.put(c);
  return 0;
}
```

运行情况如下：

```
C++ is very interesting.↙
C++isveryinteresting.
^Z(结束)
```

2. peek函数

peek是“观察”的意思，**peek**函数的作用是观测下一个字符。其调用形式为

c=cin.peek( );cin.peek

函数的返回值是指针指向的当前字符，但它只是观测，指针仍停留在当前位置，并不后移。如果要访问的字符是文件结束符，则函数值是**EOF(-1)**。

3. putback函数

其调用形式为

cin.putback(ch);

其作用是将前面用**get**或**getline**函数从输入流中读取的字符**ch**返回到输入流，插入到当前指针位置，以供后面读取。

例**13.9 peek**函数和**putback**函数的用法。

```
#include <iostream>
using namespace std;
int main( )
{char c[20];
 int ch;
 cout<<"please enter a sentence:"<<endl;
 cin.getline(c,15,'/');
 cout<<"The first part is:"<<c<<endl;
```

```
 ch=cin.peek( );//观看当前字符
 cout<<"The next character(ASCII code) is:"<<ch<<endl;
 cin.putback(c[0]);            //将'I'插入到指针所指处
 cin.getline(c,15,'/');
 cout<<"The second part is:"<<c<<endl;
 return 0;
}
```

运行情况如下：

```
please enter a sentence:
I am a boy./ am a student./↙
The first part is:I am a boy.
The next character(ASCII code) is:32(下一个字符是空格)
The second part is:I am a student
```

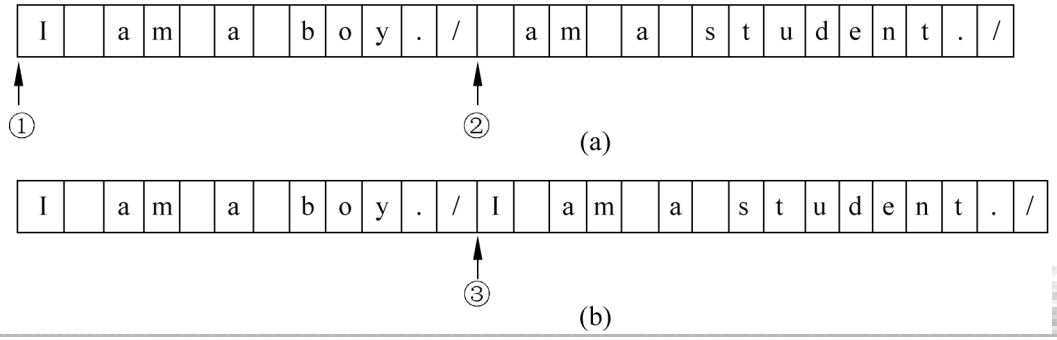

图13.4

4. ignore函数

其调用形式为

cin,ignore(n, 终止字符**)**

函数作用是跳过输入流中**n**个字符，或在遇到指定的终止字符时提前结束(此时跳过包括终止字符在内的若干字符)。如

ighore(5, 'A')//跳过输入流中**5**个字符，遇**'A'**后就不再跳了

也可以不带参数或只带一个参数。如

ignore( )(n默认值为**1**，终止字符默认为**EOF)**

相当于

ignore(1,EOF)

例13.10用ignore函数跳过输入流中的字符。

先看不用ignore函数的情况：

```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
 cin.get(ch,20,'/');
 cout<<"The first part is:"<<ch<<endl;
cin.get(ch,20,'/');
 cout<<"The second part is:"<<ch<<endl;
return 0;
}
```

运行结果如下：

```
I like C++./I study C++./I am happy.↙
The first part is:I like C++.
The second part is:(字符数组ch中没有从输入流中读取有效字符)
```

如果希望第二个**cin.get**函数能读取**"I study C++."**，就应该设法跳过输入流中第一个**'/'**，可以用**ignore**函数来实现此目的，将程序改为

```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
 cin.get(ch,20,'/');
 cout<<"The first part is:"<<ch<<endl;
 cin.ignore( );//跳过输入流中一个字符
 cin.get(ch,20,'/');
 cout<<"The second part is:"<<ch<<endl;
 return 0;
}
```

运行结果如下：

```
I like C++./I study C++./I am happy.↙
The first part is:I like C++.
The second part is:I study C++.
```

以上介绍的各个成员函数，不仅可以用**cin**流对象来调用，而且也可以用**istream**类的其他流对象调用。

13.4 文件操作与文件流

13.4.1 文件的概念

迄今为止，我们讨论的输入输出是以系统指定的标准设备(输入设备为键盘，输出设备为显示器)为对象的。在实际应用中，常以磁盘文件作为对象。即从磁盘文件读取数据，将数据输出到磁盘文件。

所谓“文件”，一般指存储在外部介质上数据的集合。一批数据是以文件的形式存放在外部介质上的。操作系统是以文件为单位对数据进行管理的。要向外部介质上存储数据也必须先建立一个文件（以文件名标识），才能向它输出数据。

外存文件包括磁盘文件、光盘文件和U盘文件。目前使用最广泛的是磁盘文件。

对用户来说，常用到的文件有两大类，一类是程序文件(program file)。一类是数据文件(data file)。程序中的输入和输出的对象就是数据文件。

根据文件中数据的组织形式，可分为ASCII文件和二进制文件。

对于字符信息，在内存中是以ASCII代码形式存放的，因此，无论用ASCII文件输出还是用二进制文件输出，其数据形式是一样的。但是对于数值数据，二者是不同的。例如有一个长整数100000，在内存中占4个字节，如果按内部格式直接输出，在磁盘文件中占4个字节，如果将它转换为ASCII码形式输出，则要占6个字节，见图13.5。

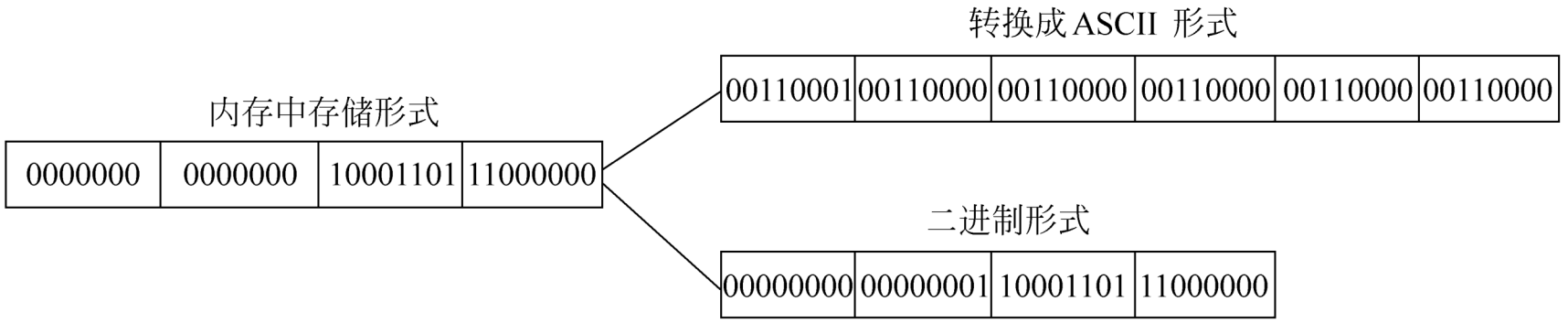

图13.5

C++提供低级的**I/O**功能和高级的**I/O**功能。高级的**I/O**功能是把若干个字节组合为一个有意义的单位，然后以**ASCII**字符形式输入和输出。传输大容量的文件时由于数据格式转换，速度较慢，效率不高。

所谓低级的**I/O**功能是以字节为单位输入和输出的，在输入和输出时不进行数据格式的转换。这种输入输出速度快、效率高，一般大容量的文件传输用无格式转换的**I/O**，但使用时会感到不太方便

13.4.2 文件流类与文件流对象

文件流是以外存文件为输入输出对象的数据流。输出文件流是从内存流向外存文件的数据，输入文件流是从外存文件流向内存的数据。每一个文件流都有一个内存缓冲区与之对应。

请区分文件流与文件的概念。文件流本身不是文件，而只是以文件为输入输出对象的流。若要对磁盘文件输入输出，就必须通过文件流来实现。

在**C++**的**I/O**类库中定义了几种文件类，专门用于对磁盘文件的输入输出操作。在图**13.2**中可以看到除了已介绍过的标准输入输出流类**istream**，**ostream**和**iostream**类外，还有**3**个用于文件操作的文件类：

(1) ifstream类，它是从**istream**类派生的。用来支持从磁盘文件的输入。

(2) ofstream类，它是从**ostream**类派生的。用来支持向磁盘文件的输出。

(3) fstream类，它是从**iostream**类派生的。用来支持对磁盘文件的输入输出。

要以磁盘文件为对象进行输入输出，必须定义一个文件流类的对象，通过文件流对象将数据从内存输出到磁盘文件，或者通过文件流对象从磁盘文件将数据输入到内存。

其实在用标准设备为对象的输入输出中，也是要定义流对象的，如**cin,cout**就是流对象，**C++**是通过流对象进行输入输出的。

由于**cin,cout**已在**iostream.h**中事先定义，所以用户不需自己定义。在用磁盘文件时，由于情况各异，无法事先统一定义，必须由用户自己定义。此外，对磁盘文件的操作是通过文件流对象(而不是**cin**和**cout**)实现的。文件流对象是用文件流类定义的，而不是用**istream**和**ostream**类来定义的。

可以用下面的方法建立一个输出文件流对象：

```
ofstream outfile;
```

现在在程序中定义了**outfile**为**ofstream**类(输出文件流类)的对象。但是有一个问题还未解决：在定义**cout**时已将它和标准输出设备建立关联，而现在虽然建立了一个输出文件流对象，但是还未指定它向哪一个磁盘文件输出，需要在使用时加以指定。

13.4.3 文件的打开与关闭

1. 打开磁盘文件

打开文件是指在文件读写之前做必要的准备工作，包括:

(1) 为文件流对象和指定的磁盘文件建立关联，以便使文件流流向指定的磁盘文件。

(2) 指定文件的工作方式。

以上工作可以通过两种不同的方法实现。

(1) 调用文件流的成员函数**open**。如

```
ofstream outfile;//定义ofstream类(输出文件流类)对象outfile
outfile.open("f1.dat",ios::out);  //使文件流与f1.dat文件建立关联
```

调用成员函数**open**的一般形式为

文件流对象.**open**(磁盘文件名，输入输出方式);

磁盘文件名可以包括路径，如**"c:\\new\\f1.dat"**，如缺省路径，则默认为当前目录下的文件。

(2) 在定义文件流对象时指定参数

在声明文件流类时定义了带参数的构造函数，其中包含了打开磁盘文件的功能。因此，可以在定义文件流对象时指定参数，调用文件流类的构造函数来实现打开文件的功能。如

```
ostream outfile("f1.dat",ios::out);
```

一般多用此形式，比较方便。作用与**open**函数相同。

输入输出方式是在**ios**类中定义的，它们是枚举常量，有多种选择，见书中表**13.6**。

说明:

① 新版本的**I/O**类库中不提供**ios::nocreate**和**ios::noreplace**。

② 每一个打开的文件都有一个文件指针。

③ 可以用“位或”运算符“|”对输入输出方式进行组合。

④ 如果打开操作失败，**open**函数的返回值为**0**(假)，如果是用调用构造函数的方式打开文件的，则流对象的值为**0**。

2. 关闭磁盘文件

在对已打开的磁盘文件的读写操作完成后，应关闭该文件。关闭文件用成员函数**close**。如

```
outfile.close( );//将输出文件流所关联的磁盘文件关闭
```

所谓关闭，实际上是解除该磁盘文件与文件流的关联，原来设置的工作方式也失效，这样，就不能再通过文件流对该文件进行输入或输出。此时可以将文件流与其他磁盘文件建立关联，通过文件流对新的文件进行输入或输出。如

```
outfile.open("f2.dat",ios::app|ios::nocreate);
```

此时文件流**outfile**与**f2.dat**建立关联，并指定了**f2.dat**的工作方式。

13.4.4 对ASCII文件的操作

如果文件的每一个字节中均以**ASCII**代码形式存放数据，即一个字节存放一个字符，这个文件就是**ASCII**文件(或称字符文件)。程序可以从**ASCII**文件中读入若干个字符，也可以向它输出一些字符。

对**ASCII**文件的读写操作可以用以下两种方法:

(1) 用流插入运算符“**<<**”和流提取运算符“**>>**”输入输出标准类型的数据。

(2) 用本章**13.2.3**节和**13.3.2**节中介绍的文件流的**put,get,geiline**等成员函数进行字符的输入输出。

例13.11 有一个整型数组，含10个元素，从键盘输入10个整数给数组，将此数组送到磁盘文件中存放。

```
#include <fstream>
using namespace std;
int main( )
{int a[10];
 ofstream outfile("f1.dat",ios::out);//定义文件流对象，打开磁盘文件"f1.dat"
 if(!outfile)                    //如果打开失败，outfile返回0值
  {cerr<<"open error!"<<endl;
   exit(1);
  }
 cout<<"enter 10 integer numbers:"<<endl;
 for(int i=0;i<10;i++)
  {cin>>a[i];
   outfile<<a[i]<<" ";}          //向磁盘文件"f1.dat"输出数据
 outfile.close();                //关闭磁盘文件"f1.dat"
 return 0;
}
```

运行情况如下:

enter 10 integer numbers:

1 3 5 2 4 6 10 8 7 9 ↙

请注意: 在向磁盘文件输出一个数据后，要输出一个(或几个)空格或换行符，以作为数据间的分隔，否则以后从磁盘文件读数据时，10个整数的数字连成一片无法区分。

例13.12 从例13.11建立的数据文件f1.dat中读入10个整数放在数组中，找出并输出10个数中的最大者和它在数组中的序号。

```
#include <fstream>
int main( )
{int a[10],max,i,order;
 ifstream infile("f1.dat",ios::in|ios::nocreate);
//定义输入文件流对象，以输入方式打开磁盘文件f1.dat
 if(!infile)
  {cerr<<"open error!"<<endl;
   exit(1);
  }
 for(i=0;i<10;i++)
   {infile>>a[i];//从磁盘文件读入10个整数，顺序存放在a数组中
    cout<<a[i]<<" ";}          //在显示器上顺序显示10个数
 cout<<endl;
 max=a[0];
```

```
 order=0;
 for(i=1;i<10;i++)
   if(a[i]>max)
{max=a[i];              //将当前最大值放在max中
order=i;               //将当前最大值的元素序号放在order中
    }
 cout<<"max="<<max<<endl<<"order="<<order<<endl;
 infile.close();
 return 0;
}
```

运行情况如下：

1 3 5 2 4 6 10 8 7 9(在磁盘文件中存放的10个数)

max=10　　(最大值为10)

order=6　　(最大值是数组中序号为6的元素)

例13.13 从键盘读入一行字符，把其中的字母字符依次存放在磁盘文件f2.dat中。再把它从磁盘文件读入程序，将其中的小写字母改为大写字母，再存入磁盘文件f3.dat。

```
#include <fstream>
using namespace std;
// save_to_file函数从键盘读入一行字符，并将其中的字母存入磁盘文件
void save_to_file( )
{ofstream outfile("f2.dat");
//定义输出文件流对象outfile，以输出方式打开磁盘文件f2.dat
if(!outfile)
  {cerr<<"open f2.dat error!"<<endl;
exit(1);
  }
 char c[80];
 cin.getline(c,80);//从键盘读入一行字符
 for(int i=0;c[i]!=0;i++)            //对字符逐个处理，直到遇'/0'为止
if(c[i]>=65 && c[i]<=90||c[i]>=97 && c[i]<=122)//如果是字母字符
{outfile.put(c[i]);                  //将字母字符存入磁盘文件f2.dat
```

```
cout<<c[i];}                    // 同时送显示器显示
 cout<<endl;
 outfile.close();                // 关闭f2.dat
}

//从磁盘文件f2.dat读入字母字符，将其中的小写字母改为大写字母，再存入f3.dat
void get_from_file()
{char ch;
 ifstream infile("f2.dat",ios::in|ios::nocreate);
//定义输入文件流outfile，以输入方式打开磁盘文件f2.dat
 if(!infile)
  {cerr<<"open f2.dat error!"<<endl;
   exit(1);
  }
ofstream outfile("f3.dat");
 //定义输出文件流outfile，以输出方式打开磁盘文件f3.dat
if(!outfile)
  {cerr<<"open f3.dat error!"<<endl;
   exit(1);
  }
```

```
while(infile.get(ch))//当读取字符成功时执行下面的复合语句
 {if(ch>=97 && ch<=122)        //判断ch是否为小写字母
ch=ch-32;                  //将小写字母变为大写字母
  outfile.put(ch);           //将该大写字母存入磁盘文件f3.dat
  cout<<ch;                  //同时在显示器输出
 }
cout<<endl;
infile.close( );             //关闭磁盘文件f2.dat
outfile.close();             //关闭磁盘文件f3.dat
}
int main( )
{save_to_file( );
    //调用save_to_file( )，从键盘读入一行字符并将其中的字母存入磁盘文件f2.dat
get_from_file( );
  //调用get_from_file()，从f2.dat读入字母字符，改为大写字母，再存入f3.dat
 return 0;
}
```

运行情况如下：

New Beijing, Great Olypic, 2008, China.↙

NewBeijingGreatOlypicChina(将字母写入磁盘文件f2.dat，同时在屏幕显示)

NEWBEIJINGGREATOLYPICCHINA　　(改为大写字母)

磁盘文件**f3.dat**的内容虽然是**ASCII**字符，但人们是不能直接看到的，如果想从显示器上观看磁盘上**ASCII**文件的内容，可以采用以下两个方法**:**

（**1**） 在**DOS**环境下用**TYPE**命令，如

D:\\C++>TYPE f3.dat↙(假设当前目录是**D:\\C++)**

在显示屏上会输出

NEWBEIJINGGREATOLYPICCHINA

如果用**GCC**编译环境，可选择**File**菜单中的**DOS Shell**菜单项，即可进入**DOS**环境。想从**DOS**返回**GCC**主窗口，从键盘输入**exit**即可。

(2) 编一程序将磁盘文件内容读入内存，然后输出到显示器。可以编一个专用函数。

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

```
void display_file(char *filename)
{ifstream infile(filename,ios::in|ios::nocreate);
 if(!infile)
  {cerr<<"open error!"<<endl;
   exit(1);}
 char ch;
 while(infile.get(ch))
cout.put(ch);
 cout<<endl;
 infile.close();
}
```

然后在调用时给出文件名即可:

```
int main( )
{display_file("f3.dat");//将f3.dat的入口地址传给形参filename
 return 0;
}
```

运行时输出**f3.dat**中的字符:

NEWBEIJINGGREATOLYPICCHINA

13.4.5 对二进制文件的操作

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。因为文件中的信息不是字符数据，而是字节中的二进制形式的信息，因此它又称为字节文件。

对二进制文件的操作也需要先打开文件，用完后要关闭文件。在打开时要用**ios::binary**指定为以二进制形式传送和存储。二进制文件除了可以作为输入文件或输出文件外，还可以是既能输入又能输出的文件。这是和**ASCII**文件不同的地方。

1. 用成员函数read和write读写二进制文件

对二进制文件的读写主要用**istream**类的成员函数**read**和**write**来实现。这两个成员函数的原型为

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

字符指针**buffer**指向内存中一段存储空间。**len**是读写的字节数。调用的方式为

```
a. write(p1,50);
b. read(p2,30);
```

例**13.14** 将一批数据以二进制形式存放在磁盘文件中。

```
#include <fstream>
using namespace std;
struct student
{char name[20];
int num;
 int age;
 char sex;
};
int main( )
{student stud[3]={"Li",1001,18,'f',"Fun",1002,19,'m',"Wang",1004,17,'f'};
 ofstream outfile("stud.dat",ios::binary);
   if(!outfile)
    {cerr<<"open error!"<<endl;
     abort( );//退出程序
```

```
    }
   for(int i=0;i<3;i++)
    outfile.write((char*)&stud[i],sizeof(stud[i]));
   outfile.close( );
return 0;
}
```

其实可以一次输出结构体数组的3个元素，将**for**循环的两行改为以下一行：

```
outfile.write((char*)&stud[0],sizeof(stud));
```

执行一次**write**函数即输出了结构体数组的全部数据。

可以看到，用这种方法一次可以输出一批数据，效率较高。在输出的数据之间不必加入空格，在一次输出之后也不必加回车换行符。在以后从该文件读入数据时不是靠空格作为数据的间隔，而是用字节数来控制。

例13.15 将刚才以二进制形式存放在磁盘文件中的数据读入内存并在显示器上显示。

```
#include <fstream>
using namespace std;
struct student
{string name;
 int num;
 int age;
 char sex;
};
int main( )
{student stud[3];
 int i;
 ifstream infile("stud.dat",ios::binary);
 if(!infile)
  {cerr<<"open error!"<<endl;
   abort( );
  }
```

```
  for(i=0;i<3;i++)
infile.read((char*)&stud[i],sizeof(stud[i]));
  infile.close( );
  for(i=0;i<3;i++)
   {cout<<"NO."<<i+1<<endl;
    cout<<"name:"<<stud[i].name<<endl;
    cout<<"num:"<<stud[i].num<<endl;;
    cout<<"age:"<<stud[i].age<<endl;
    cout<<"sex:"<<stud[i].sex<<endl<<endl;
   }
  return 0;
 }
```

运行时在显示器上显示：

```
NO.1
name: Li
num: 1001
age: 18
sex: f
```

NO.2
name: Fun
num: 1001
age: 19
sex: m

NO.3
name: Wang
num: 1004
age: 17
sex: f

请思考：能否一次读入文件中的全部数据，如

infile.read((char*)&stud[0],sizeof(stud));

2. 与文件指针有关的流成员函数

在磁盘文件中有一个文件指针，用来指明当前应进行读写的位置。对于二进制文件，允许对指针进行控制，使它按用户的意图移动到所需的位置，以便在该位置上进行读写。文件流提供一些有关文件指针的成员函数。为了查阅方便，将它们归纳为书中表**13.7**，并作必要的说明。

说明:

(1) 这些函数名的第一个字母或最后一个字母不是**g**就是**p**。

(2) 函数参数中的“文件中的位置”和“位移量”已被指定为**long**型整数，以字节为单位。“参照位置”可以是下面三者之一:

- **ios::beg**文件开头**(beg**是**begin**的缩写)，这是默认值。
- **ios::cur**指针当前的位置**(cur**是**current**的缩写)。
- **ios::end**文件末尾。

它们是在**ios**类中定义的枚举常量。

举例如下：

```
infile.seekg(100);//输入文件中的指针向前移到100字节位置
infile.seekg(-50,ios::cur);   //输入文件中的指针从当前位置后移50字节
outfile.seekp(-75,ios::end);  //输出文件中的指针从文件尾后移50字节
```

3. 随机访问二进制数据文件

一般情况下读写是顺序进行的，即逐个字节进行读写。但是对于二进制数据文件来说，可以利用上面的成员函数移动指针，随机地访问文件中任一位置上的数据，还可以修改文件中的内容。

例13.16 有5个学生的数据，要求：

（1） 把它们存到磁盘文件中；

（2） 将磁盘文件中的第1,3,5个学生数据读入程序，并显示出来；

（3） 将第3个学生的数据修改后存回磁盘文件中的原有位置。

（4） 从磁盘文件读入修改后的5个学生的数据并显示出来。

要实现以上要求，需要解决**3**个问题**:**

(1) 由于同一磁盘文件在程序中需要频繁地进行输入和输出，因此可将文件的工作方式指定为输入输出文件，即**ios::in|ios::out|ios::binary**。

(2) 正确计算好每次访问时指针的定位，即正确使用**seekg**或**seekp**函数。

(3) 正确进行文件中数据的重写**(**更新**)**。

可写出以下程序**:**

```
#include <fstream>
using namespace std;
struct student
{int num;
 char name[20];
 float score;
};
```

```
int main( )
{student stud[5]={1001,"Li",85,1002,"Fun",97.5,1004,"Wang",54,
                  1006,"Tan",76.5,1010,"ling",96};
 fstream iofile("stud.dat",ios::in|ios::out|ios::binary);
//用fstream类定义输入输出二进制文件流对象iofile
 if(!iofile)
  {cerr<<"open error!"<<endl;
   abort( );
  }
 for(int i=0;i<5;i++)//向磁盘文件输出5个学生的数据
   iofile.write((char *)&stud[i],sizeof(stud[i]));
 student stud1[5];                  //用来存放从磁盘文件读入的数据
 for(int i=0;i<5;i=i+2)
   {iofile.seekg(i*sizeof(stud[i]),ios::beg);  //定位于第0,2,4学生数据开头
    iofile.read((char *)&stud1[i/2],sizeof(stud1[0]));
//先后读入3个学生的数据，存放在stud1[0],stud[1]和stud[2]中
    cout<<stud1[i/2].num<<" "<<stud1[i/2].name<<" "<<stud1[i/2].score<<endl;
//输出stud1[0],stud[1]和stud[2]各成员的值
```

```
  }
 cout<<endl;
 stud[2].num=1012;                    //修改第3个学生(序号为2)的数据
 strcpy(stud[2].name,"Wu");
 stud[2].score=60;
 iofile.seekp(2*sizeof(stud[0]),ios::beg);   //定位于第3个学生数据的开头
 iofile.write((char *)&stud[2],sizeof(stud[2])); //更新第3个学生数据
 iofile.seekg(0,ios::beg);                   //重新定位于文件开头
 for(int i=0;i<5;i++)
   {iofile.read((char *)&stud[i],sizeof(stud[i]));  //读入5个学生的数据
    cout<<stud[i].num<<" "<<stud[i].name<<" "<<stud[i].score<<endl;
   }
 iofile.close( );
return 0;
}
```

运行情况如下：

1001 Li 85(第1个学生数据)
1004 Wang 54 (第3个学生数据)
1010 ling 96 (第5个学生数据)

1001 Li 85 (输出修改后5个学生数据)
1002 Fun 97.5
1012 Wu 60 (已修改的第3个学生数据)
1006 Tan 76.5
1010 ling 96

本程序将磁盘文件**stud.dat**指定为输入输出型的二进制文件。这样，不仅可以向文件添加新的数据或读入数据，还可以修改(更新)数据。利用这些功能，可以实现比较复杂的输入输出任务。

请注意，不能用**ifstream**或**ofstream**类定义输入输出的二进制文件流对象，而应当用**fstream**类。

13.5 字符串流

文件流是以外存文件为输入输出对象的数据流，字符串流不是以外存文件为输入输出的对象，而以内存中用户定义的字符数组(字符串)为输入输出的对象，即将数据输出到内存中的字符数组，或者从字符数组(字符串)将数据读入。字符串流也称为内存流。

字符串流也有相应的缓冲区，开始时流缓冲区是空的。如果向字符数组存入数据，随着向流插入数据，流缓冲区中的数据不断增加，待缓冲区满了(或遇换行符)，一起存入字符数组。如果是从字符数组读数据，先将字符数组中的数据送到流缓冲区，然后从缓冲区中提取数据赋给有关变量。

在字符数组中可以存放字符，也可以存放整数、浮点数以及其他类型的数据。在向字符数组存入数据之前，要先将数据从二进制形式转换为**ASCII**代码，然后存放在缓冲区，再从缓冲区送到字符数组。从字符数组读数据时，先将字符数组中的数据送到缓冲区，在赋给变量前要先将**ASCII**代码转换为二进制形式。总之，流缓冲区中的数据格式与字符数组相同。

文件流类有**ifstream,ofstream**和**fstream**，而字符串流类有**istrstream,ostrstream**和**strstream**。文件流类和字符串流类都是**ostream,istream**和**iostream**类的派生类，因此对它们的操作方法是基本相同的。向内存中的一个字符数组写数据就如同向文件写数据一样，但有**3**点不同**:**

(1) 输出时数据不是流向外存文件，而是流向内存中的一个存储空间。输入时从内存中的存储空间读取数据。

(2) 字符串流对象关联的不是文件，而是内存中的一个字符数组，因此不需要打开和关闭文件。

(3) 每个文件的最后都有一个文件结束符，表示文件的结束。而字符串流所关联的字符数组中没有相应的结束标志，用户要指定一个特殊字符作为结束符，在向字符数组写入全部数据后要写入此字符。

字符串流类没有**open**成员函数，因此要在建立字符串流对象时通过给定参数来确立字符串流与字符数组的关联。即通过调用构造函数来解决此问题。建立字符串流对象的方法与含义如下：

1. 建立输出字符串流对象

ostrstream类提供的构造函数的原型为

ostrstream::ostrstream(char *buffer,int n,int mode=ios::out);

buffer是指向字符数组首元素的指针，**n**为指定的流缓冲区的大小(一般选与字符数组的大小相同，也可以不同)，第**3**个参数是可选的，默认为**ios::out**方式。可以用以下语句建立输出字符串流对象并与字符数组建立关联:

```
ostrstream strout(ch1,20);
```

作用是建立输出字符串流对象**strout**，并使**strout**与字符数组**ch1**关联(通过字符串流将数据输出到字符数组**ch1**)，流缓冲区大小为**20**。

2. 建立输入字符串流对象

istrstream类提供了两个带参的构造函数，原型为

istrstream::istrstream(char *buffer);

istrstream::istrstream(char *buffer,int n);

buffer是指向字符数组首元素的指针，用它来初始化流对象(使流对象与字符数组建立关联)。可以用以下语句建立输入字符串流对象：

```
istrstream strin(ch2);
```

作用是建立输入字符串流对象**strin**，将字符数组**ch2**中的全部数据作为输入字符串流的内容。

```
istrstream strin(ch2,20);
```

流缓冲区大小为**20**，因此只将字符数组**ch2**中的前**20**个字符作为输入字符串流的内容。

3. 建立输入输出字符串流对象

strstream类提供的构造函数的原型为

strstream::strstream(char *buffer,int n,int mode);

可以用以下语句建立输入输出字符串流对象:

```
strstream strio(ch3,sizeof(ch3),ios::in|ios::out);
```

作用是建立输入输出字符串流对象，以字符数组**ch3**为输入输出对象，流缓冲区大小与数组**ch3**相同。

以上**3**个字符串流类是在头文件**strstream**中定义的，因此程序中在用到**istrstream, ostrstream**和**strstream**类时应包含头文件**strstream**(在**GCC**中，用头文件**strstream**)。

例13.17 将一组数据保存在字符数组中。

```
#include <strstream>
using namespace std;
struct student
{int num;
 char name[20];
 float score;
}
int main( )
{student stud[3]={1001,"Li",78,1002,"Wang",89.5,1004,"Fun",90};
 char c[50];//用户定义的字符数组
 ostrstream strout(c,30);  //建立输出字符串流，与数组c建立关联，缓冲区长30
 for(int i=0;i<3;i++)      //向字符数组c写3个学生的数据
   strout<<stud[i].num<<stud[i].name<<stud[i].score;
 strout<<ends;             //ends是C++的I/O操作符，插入一个'\\0'
 cout<<"array c:"<<c<<endl;   //显示字符数组c中的字符
}
```

运行时在显示器上的输出如下：

array c:

1001Li781002Wang89.51004Fun90

以上就是字符数组**c**中的字符。可以看到：

(1) 字符数组**c**中的数据全部是以**ASCII**代码形式存放的字符，而不是以二进制形式表示的数据。

(2) 一般都把流缓冲区的大小指定与字符数组的大小相同。

(3)字符数组**c**中的数据之间没有空格，连成一片，这是由输出的方式决定的。如果以后想将这些数据读回赋给程序中相应的变量，就会出现问题，因为无法分隔两个相邻的数据。为解决此问题，可在输出时人为地加入空格。如

```
for(int i=0;i<3;i++)
  strout<<" "<<stud[i].num<<" "<<stud[i].name<<" "<<stud[i].score;
```

同时应修改流缓冲区的大小，以便能容纳全部内容，今改为**50**字节。这样，运行时将输出

1001 Li 78 1002 Wang 89.5 1004 Fun 90

再读入时就能清楚地将数据分隔开。

例**13.18** 在一个字符数组**c**中存放了**10**个整数，以空格相间隔，要求将它们放到整型数组中，再按大小排序，然后再存放回字符数组**c**中。

```
#include <strstream>
using namespace std;
int main( )
{char c[50]="12 34 65 -23 -32 33 61 99 321 32";
 int a[10],i,j,t;
 cout<<"array c:"<<c<<endl;//显示字符数组中的字符串
 istrstream strin(c,sizeof(c));    //建立输入串流对象strin并与字符数组c关联
```

```
 for(i=0;i<10;i++)
  strin>>a[i];                    //从字符数组c读入10个整数赋给整型数组a
 cout<<"array a:";
 for(i=0;i<10;i++)
   cout<<a[i]<<" ";               //显示整型数组a各元素
 cout<<endl;
 for(i=0;i<9;i++)                 //用起泡法对数组a排序
for(j=0;j<9-i;j++)
     if(a[j]>a[j+1])
{t=a[j];a[j]=a[j+1];a[j+1]=t;}
 ostrstream strout(c,sizeof(c));    //建立输出串流对象strout并与字符数组c关联
 for(i=0;i<10;i++)
   strout<<a[i]<<" ";              //将10个整数存放在字符数组c
 strout<<ends;                     //加入'\\0'
 cout<<"array c:"<<c<<endl;        //显示字符数组c
 return 0;
}
```

运行结果如下：

array c: 12 34 65 -23 -32 33 61 99 321 32(字符数组**c**原来的内容)
array a: 12 34 65 -23 -32 33 61 99 321 32 (整型数组**a**的内容)
array c: -32 –23 12 32 33 34 61 65 99 321 (字符数组**c**最后的内容)

可以看到：

(1) 用字符串流时不需要打开和关闭文件。

(2) 通过字符串流从字符数组读数据就如同从键盘读数据一样，可以从字符数组读入字符数据，也可以读入整数、浮点数或其他类型数据。

(3) 程序中先后建立了两个字符串流**strin**和**strout**，与字符数组**c**关联。**strin**从字符数组**c**中获取数据，**strout**将数据传送给字符数组。分别对同一字符数组进行操作。甚至可以对字符数组交叉进行读写。

(4) 用输出字符串流向字符数组**c**写数据时，是从数组的首地址开始的，因此更新了数组的内容。

(5) 字符串流关联的字符数组并不一定是专为字符串流而定义的数组，它与一般的字符数组无异，可以对该数组进行其他各种操作。

与字符串流关联的字符数组相当于内存中的临时仓库，可以用来存放各种类型的数据(以**ASCII**形式存放)，在需要时再从中读回来。它的用法相当于标准设备(显示器与键盘)，但标准设备不能保存数据，而字符数组中的内容可以随时用**ASCII**字符输出。它比外存文件使用方便，不必建立文件(不需打开与关闭)，存取速度快。但它的生命周期与其所在的模块(如主函数)相同，该模块的生命周期结束后，字符数组也不存在了。因此只能作为临时存储空间。