CHAPTER 3

第3章 数据的输入和输出

一个通用的程序往往需要包含对数据的输入和输出的处理过程，数据的输入和输出部分是程序和用户之间的界面部分。在C++语言程序中也可以直接使用C语言中的函数来进行标准输入输出和文件处理，如scanf( )函数、printf( )函数等，只要在源文件的首部用#include指令包含进stdio.h头文件，就可以使用这些函数来进行输入和输出，有关这些函数的具体用法请参阅相关的C语言程序设计教程。除此之外，在C++语言中还可以利用系统提供的流类来进行标准输入输出和文件处理。

“流”的类型有两种：文本流和二进制流。

文本流：文本流是一个字符序列。在文本流中，有时可能需要对字符进行转换，如将换行符转换成回车换行符等。

二进制流：二进制流是一组字节序列。二进制流与外部设备上的输入输出序列存在一一对应的关系，即不对字符进行任何转换。

“文件”是一个逻辑的概念，它可用于从磁盘文件到终端的任何东西，通过打开操作，便可将一个“流”同一个特定的文件联系起来，文件打开以后，程序就可以同文件交换数据了，而通过关闭操作，则可将文件与特定的“流”之间的联系断开。在本章中，只介绍C++语言中特有的通过流类来进行标准输入输出以及文件的处理过程。

3.1 标准输入和输出

C++语言中的标准输入输出是由以下三个流类构成的。

istream：输入用流类

ostream：输出用流类

iostream：输入输出用流类

上面三个流类都是在iostream.h头文件中定义的，因此，在用上述三个流类进行输入输出时，需要在程序的首部包含进iostream.h头文件。

3.1.1 基于运算符<<和>>的输入输出

1. <<和>>运算符的基本功能

当在程序的首部利用#include <iostream.h>命令包含进iostream.h头文件时,系统将自动产生如下4个流类对象。

cin:标准输入流

cout:标准输出流

cerr:非缓冲型的标准出错流

clog:缓冲型的标准出错流

在上面4个流类对象中,用的最多的是cin和cout,可以通过这2个流类对象及插入运算符<<和提取运算符>>来进行标准输入输出。

【例3.1】 编一程序,利用cin和cout对象进行数据的输入输出操作。

```
#include <iostream.h>
void main( )
{
      int dt;
      char ss[80];
      cout << "Input data: ";
      cin >> dt;
      cout << " Data is [" << dt << " ]\n";
      cout << " Input String: ";
      cin >> ss;
      cout << " String is [" << ss << " ]\n";
}
```

程序的运行结果如下:

```
Input data: 123
Data is [123]
Input String: abcde
String is [abcde]
```

说明:

(1) 要想利用cin和cout对象来进行数据的输入和输出操作,就必须要在程序的首部利用#include命令包含进iostream.h头文件,这就同在C语言中使用scanf( )和printf( )函数时需要包含进stdio.h头文件一样。

(2) 由于cin和cout能够自动识别变量的数据类型,因此,在进行输入输出时,不需要指定数据类型(printf( )和scanf( )函数在输入输出时需指定数据类型)。其中,cin是用于数据输入的,其写法为:cin >> ss;(从键盘向ss变量输入数据)。cout是用于数据输出的,其写法为:cout << ss;(将ss变量中的数据输出到屏幕上)。

在程序中可以连续利用>>和<<来进行数据的输入和输出,下面的例子就说明了这一用法。

【例 3.2】 编一程序，连续利用>>和<<进行数据的输入和输出。

```
#include < iostream.h >
int main(void)
{
      int a;
      long b;
      double c;
      char s[32];
      cout << " Input String,int,long,double: \n";
      cin >> s >> a >> b >> c;
      cout <<"String = " << s << " A = " << a << " B = " << b <<"C = " << c << '\n ';
      return(0);
}
```

程序的执行结果如下：

```
Input String,int,long,double:
Language
10
1000
3.14
String = Language A = 10 B = 1000 C = 3.14
```

说明：

(1) 在利用提取运算符>>来进行连续输入时，只要根据所需要输入的数据类型(由变量的类型来判断)，由键盘连续输入即可，每输入完一个数据需要按一下回车键。

(2) 在利用插入运算符<<来连续显示数据时，系统会自动根据相应变量的数据类型来显示数据。

2. 格式化输出

在实际进行数据的输出处理时，往往需要指定相应的输出格式，如域宽、精度等。

1) 域宽的指定 width( )

在利用插入运算符<<进行输出时，往往是按照最低的域宽对数值或字符串等进行输出处理，如整数 123 输出是三位，57 是以两位形式输出。

如果要设置输出的域宽，就需要利用 cout 对象的 width( )成员函数。但是，利用 width( )函数设置域宽时，每一次输出完成后都将被清除，这也就是说，如果需要，在每一次输出的前面都必须要重新设置域宽。例如：

```
cout.width(10);
cout << " ABCDE" << 123 << '\n ';
cout.width(10);
cout << " ABCDE";
cout.width(10);
cout << 123;
```

输出结果如下：

```
uuuuuABCDE123
```

```
uuuuuABCDEuuuuuuu123  (这里的u表示空格位置)
```

为了明确地清除所设置的域宽,可以利用下面的语句:

```
cout.width(0);
```

此函数执行了以后,输出时的宽度就采用默认值,也就是按照实际所需要的宽度来进行显示。

2) 填充文字的指定 fill( )

在利用 width( )成员函数指定了域宽以后,如果所显示的数据宽度小于所指定的域宽,则前面空的位置将由空格来填充,如果不想用空格来填充,就可以使用 fill( )成员函数来设置要填充的文字。例如:

```
cout.width(10);
cout.fill('$');
cout << "abcde"
cout << '\n';
cout.width(5);
cout.fill('#');
cout << 123;
```

输出结果如下:

```
$$$$$abcde
##123
```

需要注意的是,如果把上述程序段改为如下这样:

```
cout.width(10);
cout.fill('$');
cout << "abcde\n"
cout.width(5);
cout.fill('#');
cout << 123;
```

则其输出结果如下:

```
$$$$abcde
##123
```

这是由于'\n'字符也占用一个字符位置的原因。

与 width( )函数不同,由 fill( )函数所设置的填充字符将一直有效,直到设置了新的填充字符为止。

3) 精度的指定 precision( )

在输出浮点数时,可以利用 cout 对象中的 precision( )成员函数来指定所要输出的浮点数的位数(包括整数部分的位数和小数点部分的位数,但不包含小数点)。例如:

```
cout.width(12);
cout.precision(7);
cout << 1234.56789 <<'\n';
cout.width(12);
```

```
cout.precision(5);
cout << 1234.56789 << '\n';
```

输出结果：

```
uuuu1234.568
uuuuuu1234.6
```

显然，最后一位数字是进行四舍五入运算之后显示出来的。

4）输出的刷新 flush( )

flush( )成员函数用于刷新输出缓冲区。请看下面的程序段：

```
#include < iostream.h >
int main(void)
{
    char ch;
    do {
            …
            cout << "继续(Y/N): ";
            cin >> ch;
    }while(ch == 'Y' || ch == 'y');
    return(0);
}
```

此程序段的意思是在进行了某一操作后，询问是否继续这一操作。由于每个流（输入/输出）都有自己的缓冲区，因此，如果在执行 cin >> ch;操作之前，cout 缓冲区没有被刷新，此时，字符串“继续(Y/N):”就有可能显示不出来，为了保证 cout 缓冲区中的数据能够显示出来，就需要调用 cout 对象的 flush( )成员函数。例如：

```
cout <<"继续(Y/N): ";
cout.flush( );
cin >> ch;
```

为了避免上述操作的烦琐性，可以在输入输出开始前，调用 cin 对象的成员函数 tie( )，以保证在对 cin 进行输入操作时，cout 一定被刷新。例如：

```
cin.tie(&cout);
```

3. 控制符处理

在数据的输入输出过程中，如果需要指定数据的格式，可以使用 C++语言中提供的控制符。下面的例子说明了控制符的使用方法。

【例 3.3】 编一程序，用于说明控制符的使用方法。

```
#include < iostream.h >
void main( )
{
    int dt = 255;
    cout << dec << dt << '\n';
    cout << hex << dt <<'\n';
}
```

程序的输出结果如下：

```
255
ff
```

说明：

dec 和 hex 都是用于在输入输出过程中进行数据格式转换的控制符。其中，dec 用于将数据转换为十进制的形式，而 hex 则用于将数据转换为十六进制的形式。

可以使用的一些主要的控制符如表 3.1 所示。

表 3.1　控制符

控制符	功　能	用　途
oct	八进制数处理	输入输出
dec	十进制数处理	输入输出
hex	十六进制数处理	输入输出
ws	跳过空格	输入
flush	刷新缓冲区(强制输出)	输出
endl	输出时追加'\n'，并刷新缓冲区	输出
ends	输出时追加'\0'，并刷新缓冲区	输出

对控制符使用的说明如下。

(1) 表 3.1 中的控制符是在 iostream.h 头文件中定义的，因此，要想使用这些控制符，就必须在程序的首部包含进 iostream.h 头文件。

(2) cin >> hex >>x;表示所输入的是十六进制数。例如：

```
int x;
cin >> hex >> x;                    //输入 12(十六进制的 12)
cout << x;                          //显示 18(十进制的 18)
```

(3) 控制符 endl 的作用同'\n'是相同的。例如：

```
int a = 10, b = 20;
cout << "a = " << a << endl << "b = " << b << endl;
```

其输出结果如下：

```
a = 10
b = 20
```

3.1.2　字符的输入 get( )和输出 put( )

在 C++语言中，可以不使用<<和>>运算符，通过使用其成员函数也可以进行数据的输入输出。

这里，get( )函数是 cin 对象的成员函数，用于输入操作；put( )函数是 cout 对象的成员函数，用于输出操作。其用法如下面的例子所示。

【例 3.4】 编一程序，利用 get( )函数和 put( )函数来进行数据的输入和输出。

```
#include < iostream.h >
```

```
void main( )
{
    char ch;
    while(cin.get(ch))
    {
        cout.put(ch);
    }
}
```

说明：

(1) get( )函数出错时返回 null，正常时返回 this 指针。

(2) 每输入一个字符按一下回车键，最后结束时按 Ctrl+Z 键即可。

3.1.3　字符串的输入 get( )和 getline( )

使用 cin 对象中的 get( )和 getline( )函数可以进行字符串输入，其函数说明如下所示：

```
istream& get(char * ss, int count, char delim = '\n');
istream& getline(char * ss, int count, char delim = '\n');
```

其中：

ss——缓冲区指针；

count——最多向 ss 存入 count 个字符；

delim——如果遇到此字符，输入结束。

对于字符串输入来讲，get( )和 getline( )函数的功能基本相同，但 get( )函数并不读入由 delim 标识的区分字符，而 getline( )函数读入由 delim 标识的区分字符，这一点在实际应用中需要引起注意。

【例 3.5】 编一程序，利用 get( )和 getline( )成员函数来输入数据，并比较其差异。

```
#include <iostream.h>
void main( )
{
   char ss[80];
   cout <<"getline test\n";
   cout <<"First string: ";
   cin.getline(ss,80);
   cout <<"First string is["<< ss <<"]\n";
   cout <<"Second string: ";
   cin.getline(ss,80);
   cout <<"Second string is["<< ss <<"]\n";
   cout <<"get test\n";
   cout <<"First string: ";
   cin.get(ss,80);
   cout <<"First string is["<< ss <<" ]\n";
   cout <<"Second string: ";
   cin.get(ss,80);
   cout <<"Second string is["<< ss <<"]\n";
}
```

程序的执行结果如下：

```
getline test
First string:ABC
First string is[ABC]
Second string:DEF
Second string is[DEF]
get test
First string:GHI
First string is[GHI]
Second string:Second string is[ ]
```

说明：

由最后一行的显示结果可知，由于 get( )函数没有读出上一个字符串中的'\n'字符，所以在第二次处理时，没等输入字符就显示了空串，这样若在 while 语句中处理时，将会陷入死循环中，所以，在实际使用时应引起注意。

3.2 文件

在 C++语言中，与磁盘文件的输入输出有关的流类有如下几个。

ifstream：文件输入用流类

ofstream：文件输出用流类

fstream：文件输入输出用流类

由于这些类的定义是在 fstream.h 中进行的，因此，在使用这些类进行输入输出操作时，必须要在程序的首部利用 #include 指令包含进 fstream.h 头文件。

例如，若想用 fin 作为输入用对象，fout 作为输出用对象，则可以使用如下定义：

```
ifstream fin;
ofstream fout;
```

定义了上述对象之后，就可以使用它们来打开文件和进行文件操作了。

3.2.1 文件的打开和关闭

打开文件时一般使用成员函数 open( )，关闭文件时使用成员函数 close( )。

输入用的 open 成员函数的格式如下：

```
void open(const char * fname, int openmode = ios::in, int prot = filebut::openprot);
```

输出用的 open 成员函数的格式如下：

```
void open(const char * fname, int openmode = ios::out, int prot = filebut::openprot);
```

其中：

fname——指向文件名的指针；

openmode——打开模式；

prot——打开文件的保护种类(一般采用默认值)。

这里，openmode 对于输入和输出是不同的，下面分别进行简单的介绍。

1. 输入用 openmode 的定义

ios::in——以输入方式打开文件(ifstream 默认的值)，当文件不存在时，创建一个空文件。

ios::nocreate——如果文件不存在，出错，不创建新文件。

ios::binary——以二进制方式打开文件(默认为正文方式)。

2. 输出用 openmode 的定义

ios::app——以追加方式打开文件。

ios::ate——打开文件，并将文件指针移入最后。

ios::in——即使存在同名文件也不删除。

ios::out——以输出方式打开文件(ofstream 的默认值)。

ios::trunc——如果文件存在，则删除其中的内容。

ios::nocreate——如果文件不存在，出错。

ios::noreplace——如果文件存在，出错(防止改写)。

ios::binary——以二进制方式打开文件(默认为正文方式)。

有了上述打开模式之后，就可以采用如下两种方法来打开文件。

方法一：先定义输入用对象，然后利用 open( )成员函数来打开文件。例如：

```
ifstream fin;                    //定义输入用对象
fin.open("datafile.txt");        //打开 datafile.txt 文件
```

这里，open( )函数的后两个参数采用默认值，即打开模式为 ios::in。

方法二：在定义输入用对象的同时打开文件。例如：

```
ifstream fin("datafile.txt");
```

这里，是直接调用构造函数来进行打开文件处理的。

对于 ofstream 类的使用也完全一样。下面的使用方法是加入了打开模式一项。

(1) 利用 ofstream 类来打开输出用文件：

```
ofstream fout("OUTPUT", ios::out);
```

(2) 利用 ifstream 类来打开输入用文件：

```
ifstream fin("INPUT", ios::in);
```

(3) 利用 fstream 类来打开输入输出用文件：

```
fstream file("INOUT", ios::in|ios::out);
```

由此不难看出，打开方式一项可以通过“或”运算符将多项连接起来使用。

当文件读写完成以后，就需要将已打开的文件关闭，close( )成员函数就是用于完成这一功能的。例如：

```
fin.close( );
fout.close( );
```

需要注意的是,这里的 ios::in 和 ios::out 分别表示 ios 类中的枚举常量 in 和 out。

3.2.2 文件的输入和输出

一旦文件被打开以后,就可以利用成员函数或插入运算符<<和提取运算符>>来进行数据的输入和输出操作。

可以用于文件操作的成员函数包括 get( )、put( )和 getline( )等,这些函数的用法与标准输入输出部分基本上是相同的。

例如:

```
ifstream fin;
ofstream fout;
char ch;
    …
fin.open("mydata1.txt");        //打开输入用文件
fout.open("mydata2.txt");       //打开输出用文件
    …
fin.get(ch);                    //从输入文件中读一个字符到 ch 中
fout.put(ch);                   //将 ch 中字符写入输出文件中
    …
fin.close( );                   //关闭输入文件
fout.close( );                  //关闭输出文件
```

在进行文件输入输出时,可以利用 seekp( )或 tellp( )来设置或得到当前输出流的文件指针,也可以利用 seekg( )或 tellg( )来设置或得到当前输入流的文件指针。

例如:

```
fout.seekp(100L);
fin.seekg(200L);
```

3.2.3 错误处理

在进行文件操作时,往往会出现一些错误,如文件名不存在等,这时就需要在程序中对相应的错误进行判断和处理。

在与文件操作有关的流类中,主要包含如表 3.2 所示的一些用于检查错误的成员函数。

表 3.2 错误检查函数

成员函数	意 义
good( )	状态正常
bad( )	发生重大输入输出错
fail( )	发生可以恢复的输入输出错
eof( )	文件结束
rdstate( )	返回错误标志:
	ios::goodbit 无错
	ios::eofbit 到达文件尾部
	ios::failbit 发生可以恢复的格式错或变换错
	ios::badbit 发生重大错或未知状态

例如，在文件打开操作结束时，可以采用如下两种方法进行判断是否出错。

方法一：
```
if(fin.fail( ))        //如果发生错（可简写为：if(!fin)）
if(!fin.good( ))       //如果出错
```

方法二：
```
if(!fin.fail( ))       //如果正常　（可简写为：if(fin)）
if(fin.good( ))        //如果正常
```

【例 3.6】 编一程序，用于将一个文件中的内容显示到屏幕上。

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char *argv[ ])
{
    char ch;
    if (argc != 2)
    {
        cout << "Usage: PR<filename>\n";
        return 1;
    }
    ifstream in(argv[1]);
    if(in.fail( ))
    {
        cout << "Cannot open file\n";
        return 1;
    }
    while(!in.fail( ))
    {
        in >> ch;              //读入一个字符
        cout << ch;
    }
    in.close( );
    return 0;
}
```

说明：

本程序通过命令行参数来获取输入文件名，而当命令行上无文件名参数时，将给出程序用法提示信息。

【例 3.7】 编一程序，将一个文件中的数据写入另一个文件中，并显示到屏幕上。

```
#include <iostream.h>
#include <fstream.h>
int main( )
{
    ifstream fin;              //输入用对象
    ofstream fout;             //输出用对象
    char ch;
    fin.open("input.txt");     //打开输入文件
    if(fin.fail( ))
    {
        cout << "File open error!\n";
        return 1;
```

```
    }
    fout.open("output.txt");
    if(fout.fail( ))
    {
        cout << "File open error!\n";
        return 1;
    }
    while(fin.get(ch))
    {
        fout.put(ch));          //输出到文件中
        cout.put(ch);           //显示到屏幕上
    }
    fin.close( );
    fout.close( );
    return 0;
}
```

说明：

本程序是一个文件复制程序，其功能是将 input.txt 文件中的数据复制到 output.txt 文件中，同时将其显示到屏幕上。

习题

3.1　编一程序，输入两个整数，并将其和显示出来。

3.2　编一程序，读入一数字字符('0'～'9')，并把其转换为相应的整数后显示出来。例如，输入数字字符'9'，把其转换为十进制整数 9 后显示出来。

3.3　分析下列程序，并运行之，同时比较一下该程序和例 3.6 程序有何区别。

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char *argv[ ])
{
    char ch;
    if (argc != 2)
    {
        cout << "Usage: PR<filename>\n";
        return 1;
    }
    ifstream in(argv[1]);
    if(!in)
    {
        cout << "Cannot open file\n";
        return 1;
    }
    while(in.get(ch))
    {
        cout << ch;
    }
```

```
    in.close( );
    return 0;
}
```

3.4　分析下列程序，并运行之，同时比较一下该程序和例 3.7 程序有何区别。

```
#include < iostream.h >
#include < fstream.h >
int main( )
{
    ifstream fin;                                   //输入用对象
    ofstream fout;                                  //输出用对象
    char ch;
    fin.open("c:\\temp\\input.txt");                //打开输入文件
    if(fin.fail( ))
    {
        cout << "File open error!\n";
        return 1;
    }
    fout.open("c:\\temp\\output.txt");
    if(fout.fail( ))
    {
        cout << "File open error!\n";
        return 1;
    }
    while(!fin.fail( ))
    {
        fin >> ch;
        fout.put(ch));                              //输出到文件中
        cout.put(ch);                               //显示到屏幕上
    }
    fin.close( );
    fout.close( );
    return 0;
}
```

3.5　编一程序，输入一字符串，并将其中的大写字母全部转换为小写字母后显示出来。