

# 第10章 流和文件

- 本章主要教学内容

- C++流的概念和流类库的继承层次结构
- 输入流类istream的功能、用法和常用成员函数
- 输出流类ostream的功能、用法和常用成员函数
- 文件流类fstream的功能、用法，输入输出文件的操作流程和编程方法
- 文件和二进制文件程序设计
- 顺序文件和随机文件程序设计

- 本章教学重点

- C++流类库的功能和继承层次结构
- 输入输出流和文件流的常用成员的功能和程序设计方法
- 文件程序的设计流程和文件数据格式控制
- 二进制文件程序设计
- 用随机文件处理自定义类的文件对象

- 教学难点

- 二进制文件的程序设计
- 随机文件的建立和随机修改和读写文件记录

# 第10章 流和文件

---

- **C++**具有一个功能强大的**I/O**类继承体系结构用于处理数据的输入/输出问题，该体系结构不仅提供了对系统内置数据类型的输入/输出操作，而且允许程序员通过重载扩展其功能以实现自定义数据类型的输入和输出操作。
- 本章是在第**2**章基础上，进一步介绍**C++**输入输入流的继承体系结构，以及数据输入输出及其格式化问题，同时还介绍了**C++**文件的编程技术。

# 10.1 C++ I/O流及流类库

## 1、流的概念

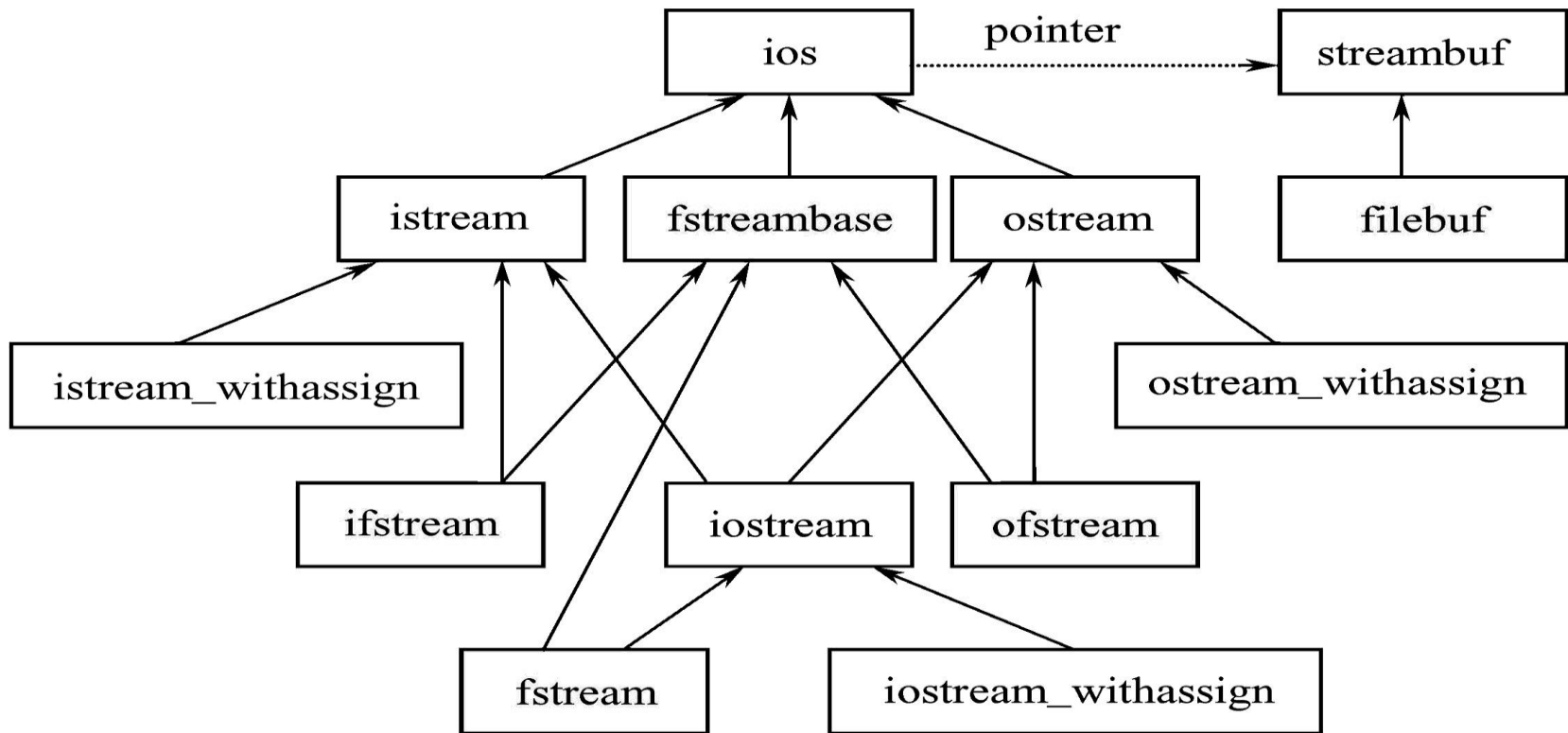
所谓**流**，是指**数据的有向流动**，即数据从一个设备流向另一个设备。

- **流**实际上**是一种对象**，它在使用前被建立，使用后被删除。数据的输入/输出操作就是从流中提取数据或者向流中添加数据。
- **输入流**：从外存将数据输入内存变量中的数据流。通常把从输入流中提取数据给内存变量的操作称为**提取**（也称为析取），即读操作；
- **输出流**：将内存数据输出到外存中的数据流。通常将向输出流中添加数据的操作称为**插入**操作，即写操作。

# 10.1 C++ I/O流及流类库

## 2、C++ I/O流类的继承结构

- C++建立了一个十分庞大的流类库来实现数据的输入/输出操作，其中的每个流类实现不同的功能，这些类通过继承组合在一起。



# 10.1 C++ I/O流及流类库

---

## 3、C++主要的流类简介

- **streambuf**主要作为其他类的支持，定义了对缓冲区的通用操作，如设置缓冲区，从缓冲区中读取数据，或向缓冲区写入数据等操作
- **filebuf**类使用文件来保存缓冲区中的字符序列。它定义了文件读、写、打开、关闭等常用操作。
- **ios**是所有流类的基类，提供对流状态进行设置的主要功能。如文件数据的格式码设置与取消，关联文件缓冲区借以实现数据读写等

# 10.1 C++ I/O流及流类库

---

- **istream**是输入流类，实现数据输入的功能；
- **ostream**是输出流类，实现数据输出的功能；
- **iostream**是**istream**和**ostream**的派生类，它继承了**istream**类和**ostream**类的行为，支持数据输入、输出的双向操作，在程序中常通过它来实现数据的输入与输出功能。
- **fstreambase**从**ios**派生，提供了文件操作的许多功能，作为其它文件操作类的公共基类。
- **ifstream**类用来实现文件读取操作，
- **ofstream**类用来实现文件写入操作。
- **fstream**继承了**fstreambase**和**iostream**类的功能，实现了文件读/写的双向操作。

# 10.1 C++ I/O流及流类库

## 4、C++预定义的输入/输出流对象

- 为了便于程序数据的输入/输出，C++预定义了几个标准输入/输出流对象。在程序中可以直接引用它们来输入/输出数据。如下表所示

对象定义	说 明
<code>ostream cout;</code>	<code>cout</code> 与标准输出设备相关联
<code>ostream cerr;</code>	<code>cerr</code> 与标准错误输出设备相关联（非缓冲方式）
<code>ostream clog;</code>	<code>clog</code> 与标准错误输出设备相关联（缓冲方式）
<code>istream cin;</code>	与标准输入设备相关联

# 10.2 I/O流类的成员函数

## 10.2.1 istream流中的常用成员函数

1、**istream**类定义了许多用于从流中提取数据和操作文件的成员函数，如下所示

```
class istream : virtual public ios {  
public:  
    istream& operator>>(double &);  
    //具有许多operator>>重载成员函数  
    .....  
    int get();  
    istream& get(char *,int,char ='\n');  
    istream& get(char &);  
    istream& getline( char *,int,char ='\n');  
    istream& read(char *,int);  
    istream& ignore(int =1,int =EOF);  
    int peek();  
    istream& putback(char);  
    .....  
};
```



**istream 或  
istream.h头  
文件中找到  
此class的声  
明！**



## 10.2.1 istream流的常用成员函数

---

### 2、几个常用的输入流成员函数

(1) int **get**( )

(2) istream& **get**( char \* c, int n,char ='\n')

(3) istream& **read**(char \*c, int n);

(4) istream& **ignore**(int =1,int =EOF)

(5) istream& **getline**(char \*c,int n,char ='\n');

**【例10-1】** 用函数get和getline读取数据。用get和getline输入数据时遇到的问题及解决方法可参见1.4.8。

```
#include <iostream>
using namespace std;
void main(){
    char c,a[50],s1[100];
    cout<<"use get() input char: ";
    while((c=cin.get())!='\n') //L1
        cout<<c;
    cout<<endl;
    cout<<"use get(a,10) input char: ";
    cin.get(a,10); //L2
    cout<<a<<endl;
    cin.ignore(1); //L3
    cout<<"use getline(s1,10) input char: ";
    cin.getline(s1,10); //L4
    cout<<s1<<endl;
}
```

# 分析例10-1的程序执行情况

下面是程序执行时的一组输入数据和输出结果：

```
use get() input char: abcd
```

```
abcd
```

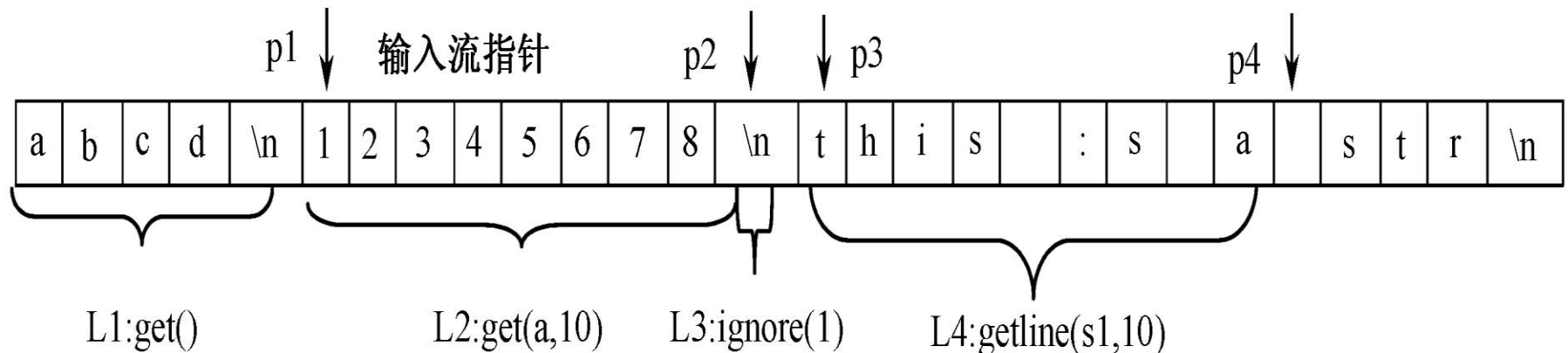
```
use get(a,10) input char: 12345678
```

```
12345678
```

```
use getline(s1,10) input char: this is a str
```

```
this is a
```

上面的输入数据将建立如下输入流：



## 10.2.2 ostream流中的常用成员函数

- 1、ostream类提供了许多用于数据输出的成员函数，并通过流的输出运算符<<重载，实现了对内置数据类型的输出功能。
  - 在ostream类中的声明原型如下：

```
class _CRTIMP ostream : virtual public ios {
public:
    ostream& operator<<(...);
    ostream& flush();
    ostream& operator<<(long);
    ostream& put(char);
    ostream& write(const char *,int);
    ostream& seekp(streampos);
    .....
};
```

## 10.2.2 ostream流中的常用成员函数

---

### 2、几个常用输出流类的成员函数

`ostream& put(char c);`

`ostream& write(const char *c,int n);`

`flush();`

## 【例10-2】 用get读取数据，用put和write输出数据。

```
//Eg10-3.cpp
```

```
#include<iostream>
```

```
using namespace std;
```

```
void main(){
```

```
    char c;
```

```
    char a[50]="this is a string...";
```

```
    cout<<"use get() input char: ";
```

```
    while((c=cin.get())!='\n') //L1 用get读取字符，遇回车键结束
```

```
        cout.put(c); //L2 将c中的字符输出
```

```
    cout.put('\n'); //L3 输出一个回车换行符
```

```
    cout.put('t').put('h').put('i').put('s').put('\n'); //L4 输出this
```

```
    cout.write(a,sizeof(a)-1).put('\n');//L5 write一次输出多个字符
```

```
    cout<<"look"<<"\t here! "<<endl;
```

```
}
```

运行程序，其输入与输出结果如下：  
use get() input char: how are you!  
how are you  
this  
this is a string...  
look here!

## 10.2.3 数据输入/输出的格式控制

### 1. ios类提供的格式控制

ios是C++所有流类的基类，它包含了C++流的主要特性。

#### (1). ios中的格式化标志

ios::skipws	跳过输入流中的空白字符
ios::left	输出数据按左对齐，如[12    ]
ios::right	输出数据按右对齐，如[     12]
ios::dec	按十进制数据输出
ios::oct	按八进制数据输出
ios::hex	按十六进制数据输出
ios::showbase	在输出数据前面显示基数（八进制是0，十六进制是0x）
ios::showpoint	浮点数输出带小数点
ios::uppercase	用大写字母输出十六进制数（即ABCDEF，默认是小写）
ios::showpos	在正数前加“+”
ios::scientific	用科学计数法输出浮点数，如[2.122E2]
ios::fixed	用定点数形式输出浮点数，如[212.2]
ios::unitbuf	完成后立即刷新缓冲区

## 10.2.3 数据输入/输出的格式控制

---

### (2) ios中的格式化成员函数

- `setf(flags)`            设置指定的格式化标志`flags`,
- `unsetf(flags)`        取消指定的格式化标志`flags`,
- `setf(flags,filed)`    先清除、然后设置标志.

– **flags**可以是前面在（1）中列举的格式化标志符



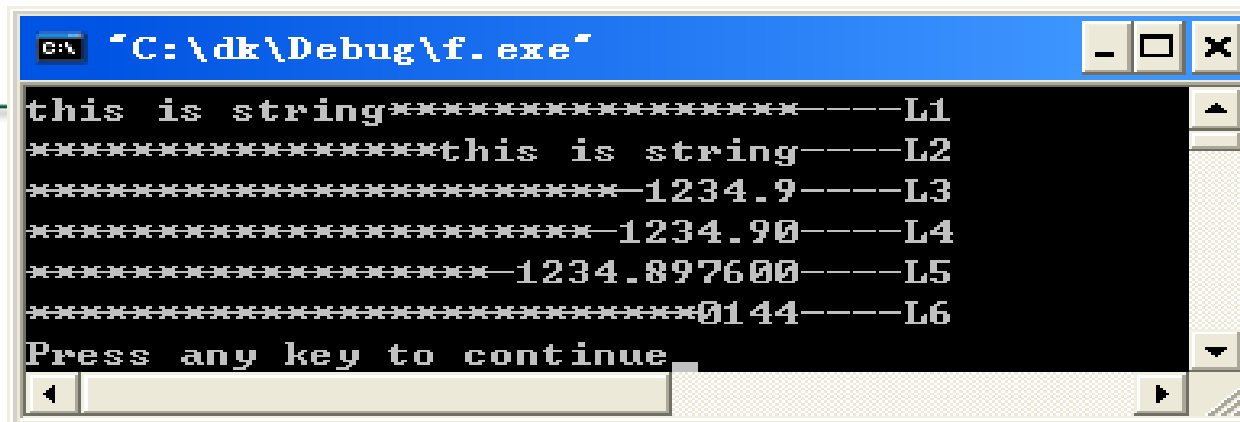
## 10.2.3 数据输入/输出的格式控制

### (3) 设置域宽、精度、填充字符的成员函数

<code>ch=fill()</code>	获取填充字符（默认为空格）， <code>ch</code> 是一个字符变量
<code>fill(ch)</code>	设置填充字符， <code>ch</code> 是要设置为填充的字符
<code>p=precision()</code>	获取当前浮点数的精度， <code>p</code> 是一个整型变量
<code>precision(p)</code>	设置精度， <code>p</code> 是一个整数，代表要设置的数字位数
<code>w=width()</code>	获取当前字段宽度（字符个数）， <code>w</code> 是整型变量
<code>width(w)</code>	设置当前字段宽度， <code>w</code> 是代表设置输出宽度的整数

## 【例10-3】 用ios成员函数及格式化标志设置输出数据的格式。

```
#include<iostream>
using namespace std;
void main(){
    char c[30]="this is string";
    double d=-1234.8976;
    cout.width(30); cout.fill('*'); cout.setf(ios::left);
    cout<<c<<"----L1"<<endl;
    cout.width(30); cout.setf(ios::right);
    cout<<c<<"----L2"<<endl;
    cout.width(30); cout.setf(ios::internal);
    cout<<d<<"----L3"<<endl;
    cout.setf(ios::dec|ios::showbase|ios::showpoint);
    cout.width(30);
    cout<<d<<"----L4"<<"\n";
    cout.setf(ios::showpoint); cout.precision(10);
    cout.width(30);
    cout<<d<<"----L5"<<"\n";
    cout.width(30); cout.setf(ios::oct,ios::basefield);
    cout<<100<<"----L6"<<"\n";
}
```



```
C:\dk\Debug\f.exe
this is string*****----L1
*****this is string----L2
*****-1234.9----L3
*****-1234.90----L4
*****-1234.897600----L5
*****0144----L6
Press any key to continue
```

## 10.2.3 数据输入/输出的格式控制

---

### 2. 利用操纵符格式化数据

- ① **C++**流类库中的每个流对象都维护着一个**格式状态**，它控制着数据格式化操作的细节。如输出数据的基数（默认为十进制数据）、对齐方式、精度等。
- ② **C++**还提供了一组可以对数据进行格式化的预定义**操纵符**（也称操纵算子）。

## 10.2.3 数据输入/输出的格式控制

### ③ C++流类中的操纵符

<b>showbase</b> ( <b>noshowbase</b> )	显示 (不显示) 数值的基数前缀
<b>showpoint</b> ( <b>noshowpoint</b> )	显示小数点 (存在小数部分时才显示小数点)
<b>showpos</b> ( <b>noshowpos</b> )	在非负数中显示 (不显示) +
<b>skipws</b> ( <b>noskipws</b> )	输入数据时, 跳过 (不跳过) 空白字符
<b>uppercase</b> ( <b>nouppercase</b> )	十六进制显示为 <b>0X</b> ( <b>0x</b> ), 科学计数法显示 <b>E</b> ( <b>e</b> )
<b>dec / oct / hex</b>	十进制/八进制/十六进制
<b>left/right</b>	设置数据输出对齐方式为: 左/右 对齐
<b>fixed</b>	以小数形式显示浮点数
<b>scientific</b>	用科学计数法显示浮点数
<b>flush</b>	刷新输出缓冲区
<b>ends</b>	插入空白字符, 然后刷新 <b>ostream</b> 缓冲区
<b>endl</b>	插入换行字符, 然后刷新 <b>ostream</b> 缓冲区
<b>ws</b>	跳过开始的空白

## 10.2.3 数据输入/输出的格式控制

---

### ④ 头文件<iomanip>或<iomanip.h>中的操纵符函数

- **setfill(ch)**                      设置ch为填充字符
- **setprecision(n)**              设置精度为n位有效数字
- **setw(w)**                      设置数据的输出宽度为w个字符
- **setbase(b)**                  基数设置为b（b=8，10，16）进制

## 10.2.3 数据输入/输出的格式控制

【例10-4】 修改例10-3，用操纵符格式化输出数据，实现同样的功能。

```
//Eg10-4.cpp
#include<iostream>
#include<iomanip>
using namespace std;
void main(){
    char c[30]="this is string";
    double d=-1234.8976;
    cout<<setw(30)<<left<<setfill('*')<<c<<"----L1"<<endl;
    cout<<setw(30)<<right<<setfill('*')<<c<<"----L2"<<endl;
    cout<<dec<<showbase<<showpoint<<setw(30)<<d<<"----L3"<<"\n";
    cout<<setw(30)<<showpoint<<setprecision(10)<<d<<"----L4"<<"\n";
    cout<<setw(30)<<setbase(8)<<100<<"----L5"<<"\n";
}
```

程序运行结果与例10-3相同，但可以看出其代码更简洁

# 10.3 文件操作

---

## 1、文件的概念

文件是存储在存储介质上（如磁盘、磁带、光盘）的数据集合。

## 2、文件的类型

### – 文本文件

- 文本文件在磁盘上存放相关字符的**ASCII**码，所以又称为**ASCII**文件。

### – 二进制文件

- 二进制文件在磁盘上存储相关数据的**二进制编码**，它是把内存中的数据，按其在内存中的存储形式原样写到磁盘上而形成的文件。

# 10.3.1 文件与流

## 1、流与文件

- C++将文件看成是一个个字符在磁盘上的有序集合，用流来实现文件的读写操作。包括文本文件和二进制文件。
- 文本文件在磁盘上存放相关字符的ASCII值，所以又称为ASCII文件。二进制文件在磁盘上存储相关数据的二进制编码，是把内存中的数据，按其在内存中的存储形式原样写到磁盘上而形成的文件。
- 文本文件与二进制文件对待回车换行符的处理方式不同。
  - 在文本方式下，输入流中的回车符和换行符都会被处理成字符'\n'，输出流中的字符'\n'则会被转换成回车符或换行符。
  - 在二进制方式下，不会进行回车符、换行符与'\n'之间的转换。

## 2、与文件相关的流

- ifstream
- ofstream
- fstream



## 10.3.1 文件与流

---

### 3、用流操作文件的过程，须经过以下4个步骤

#### (1) 建立文件流

- ifstream iFile;
- ofstream oFile;
- fstream ioFile;

#### (2) 打开文件

`void open(const char *filename,int mode,int access);`

#### (3) 访问文件

- iFile>>.....
- oFile<<.....

#### (4) 关闭文件

- iFile.close();
- oFile.close();

# 10.3.1 文件与流

## 4、打开文件的方式

文件打开方式	说 明
<code>ios::in</code>	以输入方式打开文件，即读文件（ <code>ifstream</code> 类对象默认方式）
<code>ios::out</code>	以输出方式打开文件，即写文件（ <code>ofstream</code> 类对象默认方式）
<code>ios::app</code>	以添加方式打开文件，新增加的内容添加在文件尾
<code>ios::ate</code>	以添加方式打开文件，新增加的内容添加在文件尾，但下次添加时则添加在当前位置
<code>ios::trunc</code>	如文件存在就打开并清除其内容，如不存在就建立新文件
<code>ios::binary</code>	以二进制方式打开文件（默认为文本文件）
<code>ios::nocreate</code>	打开已有文件，若文件不存在，则打开失败
<code>ios::noreplace</code>	若打开的文件已经存在，则打开失败

## 10.3.1 文件与流

---

### 5、文件访问模式

- |                     |            |
|---------------------|------------|
| – filebuf::openport | 共享方式       |
| – filebuf::sh_none  | 独占方式，不允许共享 |
| – filebuf::sh_read  | 允许读共享      |
| – filebuf::sh_write | 允许写共享      |

- 本书2.15节介绍了用文件流ifstream和ofstream处理文本文件的方法，在此不再介绍。

## 10.3.2 二进制文件

### 1、二进制文件与文本文件的区别

- 在读文件时，它们**判定文件结束标志的方法存在区别**：在读文本文件的过程中，当`get`之类的成员函数遇到文件结束符时，它返回常量**EOF**作为文件结束标志；但是，二进制文件不能用**EOF**作为文件结束的判定值。
  - 因为**EOF的值是-1**，若文件中某个字节的值为-1，就会被误认为是文件结束符。
  - **C++**提供了一个成员函数**`eof`**来解决这个问题，它的用法如下：

```
int xx::eof()
```
  - 其中，**xx**代表输入流对象，到达文件末尾时，返回一个非0值，否则返回0。

## 10.3.2 二进制文件

### 2、二进制文件操作方法

- 用**get/put**按字节方式**读/写**文件
- **get/put**分别定义于istream和ostream类：

**istream &get(char &ch);**

**ostream &put(char ch);**

**【例10-5】** 用二进制方式建立一个磁盘文件，将ASCII 编码为0～90之间的字符写入到文件C:\dk\a.dat中，然后用二进制文件方式读出并在屏幕上显示a.dat的内容。

## 10.3.2 二进制文件

```
//Eg10-5.cpp
```

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
void main(){
```

```
    char ch;
```

```
    ofstream out("C:\\dk\\a.dat",ios::out|ios::binary);    //L1
```

```
    for(int i=0;i<90;i++){
```

```
        if(!(i % 30)) out.put("\\n");    //L2
```

```
        out.put(char(i));    //L3
```

```
        out.put(' ');    //L4
```

```
    }
```

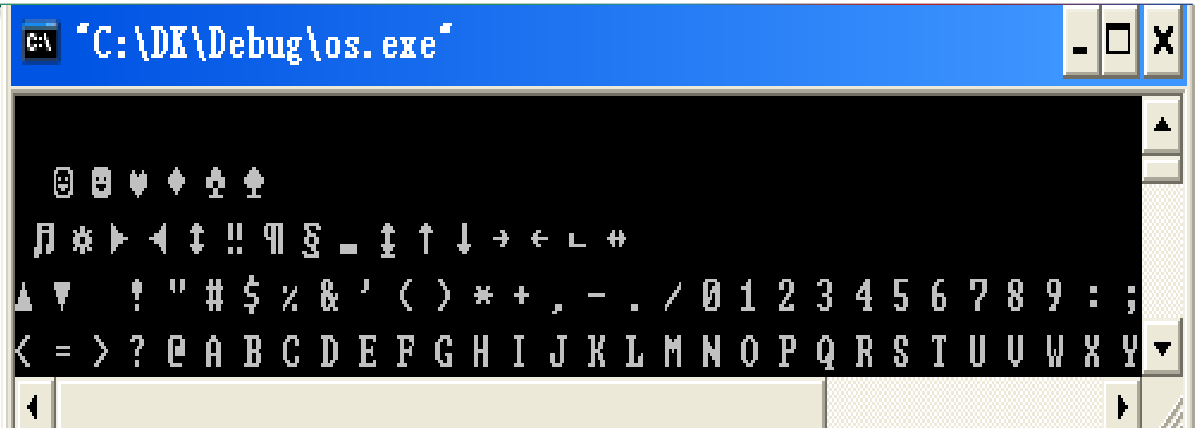
```
    out.close();    //L5
```

```
    ifstream in("C:\\dk\\a.dat",ios::in|ios::binary);    //L6
```

```
    while(in.get(ch)) cout<<ch;    //L7
```

```
    in.close();    //L8
```

```
}
```



## 10.3.2 二进制文件

### 3. 用函数read和write操作二进制文件

- **read/write** 按数据块的方式**读/写**文件数据
- 定义于**istream**和**ostream**类中，原型如下：

**istream& read**(char \*buf, int n);

**ostream& write**(const char \*buf,int n);

- **read**一次从输入流中**读取n字节**的内容放入输入缓冲区域buf中，**write**一次**插入n字节**到输出流中，即一次写入n字节内容到文件中。

**【例10-6】** 设计一个**person**类，从键盘输入每个人的姓名、身份证号、年龄、地址等数据，并将每个人的信息保存在目录**C:\dk**下的二进制文件**person.dat**中，然后将文件中的个人信息读出来，保存在**vector**类型的向量中并显示出来。

```
//Eg10-6.cpp
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;
class Person{
private:
    char name[20];
    char id[18];
    int age;
    char addr[20];
public:
    Person(){}
    Person(char* n,char* PerId,int Age, char* Address){
        strcpy(name,n);
        strcpy(id,PerId);
        strcpy(addr,Address);
        age=Age;
    }
    void display(){
        cout<<name<<"\t"<<id<<"\t"<<age<<"\t"<<addr<<endl;
    }
};
```



```
void main(){
    vector<Person> p;
    vector<Person>::iterator pos;
    char ch;
    ofstream out("C:\\dk\\person.dat",ios::out|ios::app|ios::binary);
    char Name[20],ID[18],Addr[20];
    int Age;
    cout<<"-----输入个人档案-----"<<endl<<endl;
    do{
        cout<<"姓名:    ";        cin>>Name;
        cout<<"身份证号: ";        cin>>ID;
        cout<<"年龄:    ";        cin>>Age;
        cout<<"地址:    ";        cin>>Addr;
        Person s1(Name,ID,Age,Addr);
        out.write((char*)&s1,sizeof(s1));
        cout<<"Enter another person (y/n)?";
        cin>>ch;
    } while(ch=='y');
    out.close();
```

```

ifstream in("C:\\dk\\person.dat",ios::in|ios::binary);
Person s1;
in.read((char*)&s1,sizeof(s1));
while(!in.eof()){
    p.push_back(s1);
    in.read((char*)&s1,sizeof(s1));
}
cout<<"\n-----从文件中读出的数据-----"<<endl<<endl;
pos=p.begin();
for(pos=p.begin();pos!=p.end();pos++)
    (*pos).display();
}

```

```

C:\DK\Debug\os.exe
-----输入个人档案-----
姓名: 王常驻
身份证号: 125
年龄: 32
地址: 重庆渝中区
Enter another person (y/n) ? n

-----从文件中读出的数据-----
红三    123    21    重庆市南坪
李国民  124    32    重庆沙坪坝
王常驻  125    32    重庆渝中区
Press any key to continue

```

## 10.3.3 随机文件

---

### 1、顺序文件与随机文件

- 顺序访问是指按照从前到后的顺序依次对文件进行读写操作，有些存储设备只支持顺序访问，如磁带。
- 随机访问也称为直接访问，可以按任意次序对文件进行读写操作

### 2、随机文件访问方式

- 对于随机文件，**C++**流类提供了一个操作它的文件指针，该指针可在文件中移动，将它指向要读写的字节位置，然后从该位置读取或写入指定字节数的数据块，这样就实现了文件数据的随机访问。

## 10.3.3 随机文件

### 3、定位输入文件读指针的成员函数

- `istream& seekg(long pos);` ①
- `istream& seekg(long off, dir);` ②
- `long tellg();`

### 4、定位输出文件写指针的成员函数

- `ostream& seekp(long pos);` ①
- `ostream& seekp(long off, dir);` ②
- `long tellp();`

### 5、`seekg`的两种用法，其中的：

- ①绝对定位：直接将文件读指针定位到距离文件开始位置的第`pos`字节处。
- ②相对定位：将文件读指针定位到偏移`dir`位置`off`字节的位置。`dir`可取下面的值：  
`ios::cur`——当前文件指针位置；`ios::beg`——文件开始位置；  
`ios::end`——文件结束位置。

**【例10-7】** 某雇员类Employee有编号、姓名、年龄、工资等数据成员。设计一个随机文件保存各雇员的各项数据。

---

```
//Eg10-8.cpp
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
class Employee{
private:
    int number ,age;
    char name[20];
    double sal;
public:
    Employee(){}
    Employee(int num,char* Name,int Age, double Salary){
        number=num;
        strcpy(name,Name);
        age=Age;
        sal=Salary;
    }
    void display(){
        cout<<number<<"\t"<<name<<"\t"<<age<<"\t"<<sal<<endl;
    }
};
```

## 10.3.3 随机文件

```
void main(){  
    ofstream out("Employee.dat",ios::out|ios::binary); //定义随机输出文件  
    Employee e1(1,"张三",23,2320);  
    Employee e2(2,"李四",32,3210);  
    Employee e3(3,"王五",34,2220);  
    Employee e4(4,"刘六",27,1220);  
    out.write((char*)&e1,sizeof(e1)); //按e1,e2,e3,e4顺序写入文件  
    out.write((char*)&e2,sizeof(e2));  
    out.write((char*)&e3,sizeof(e3));  
    out.write((char*)&e4,sizeof(e4));  
  
    //下面的代码将e3（即王五）的年龄改为40岁  
    Employee e5(3,"王五",40,2220);  
    out.seekp(2*sizeof(e1)); //指针定位到第3（起始为0）个数据块  
    out.write((char*)&e5,sizeof(e5)); //将e5写到第3个数据块位置，覆盖e3  
    out.close(); //关闭文件
```

# 10.3.3 随机文件

//以二进制方式建立输入文件

```
ifstream in("Employee.dat",ios::in|ios::binary);
```

```
Employee s1; //s1: 保存从文件中读出的数据
```

```
cout<<"\n-----从文件中读出第3个人的数据-----\n\n";
```

```
in.seekg(2*(sizeof(s1)),ios::beg) //文件指针定位到第3个数据块
```

```
in.read((char*)&s1,sizeof(s1)); //读出第3个雇员的数据块
```

```
s1.display();
```

```
cout<<"\n-----从文件中读出全部的数据-----\n\n";
```

```
in.seekg(0,ios::beg); //移动文件指针，指向文件开头
```

```
in.read((char*)&s1,sizeof(s1)); //读出第1个数据块
```

```
while(!in.eof()){ //如果没有读完文件，就继续读
```

```
    s1.display(); //显示读出的雇员数据
```

```
    in.read((char*)&s1,sizeof(s1)); //读当前文件指针处的数据
```

```
}
```

```
}
```

程序运行结果如下：

-----从文件中读出第3个人的数据-----

3	王五	40	2220
---	----	----	------

-----从文件中读出全部的数据-----

1	张三	23	2320
---	----	----	------

2	李四	32	3210
---	----	----	------

3	王五	40	2220
---	----	----	------

4	刘六	27	1220
---	----	----	------

//可见王五的年龄已被修改