程序设计与算法（三）

C++面向对象的程序设计

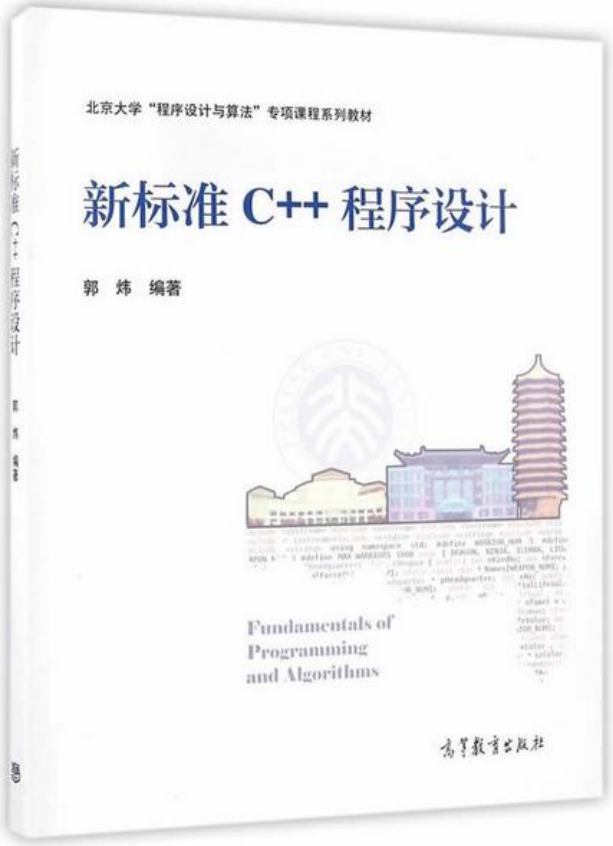
郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431

配套教材：

高等教育出版社

《新标准C++程序设计》

郭炜 编著

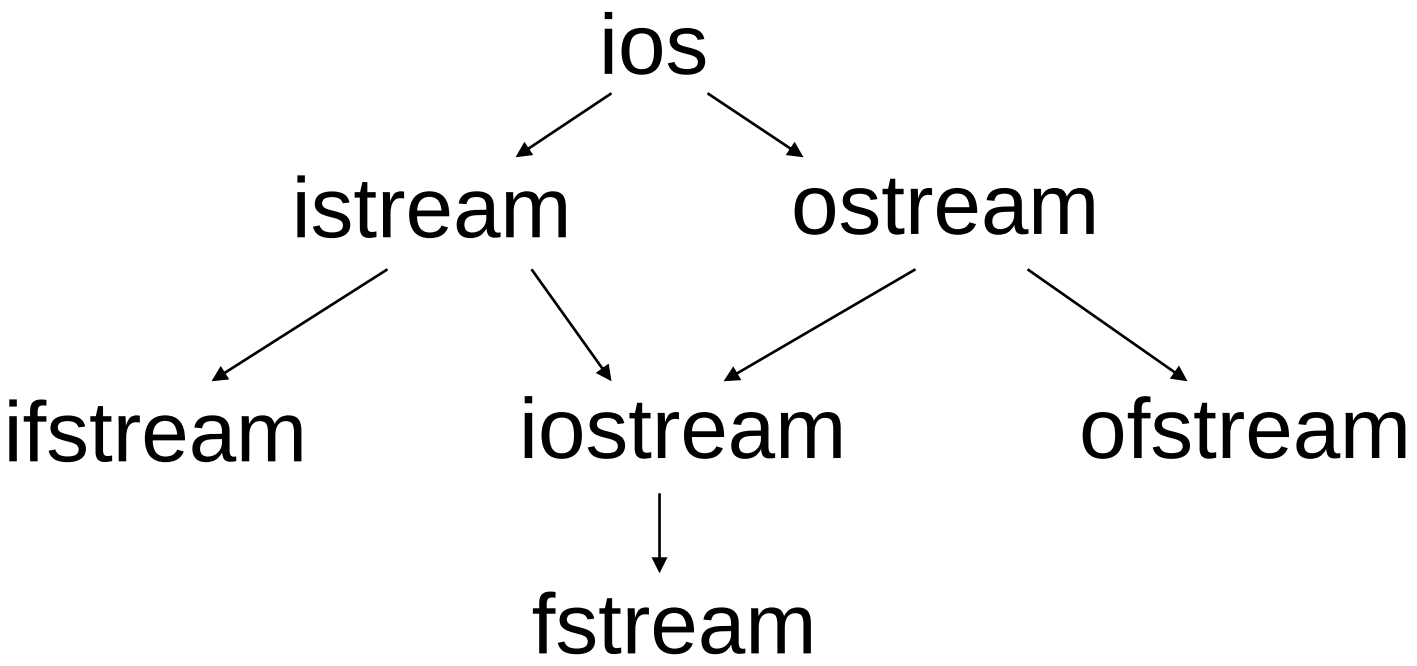


输入和输出

输入输出相关的类

冰岛塞尔福斯瀑布

与输入输出流操作相关的类



与输入输出流操作相关的类

istream是用于输入的流类，cin就是该类的对象。

ostream是用于输出的流类，cout就是该类的对象。

ifstream是用于从文件读取数据的类。

ofstream是用于向文件写入数据的类。

iostream是既能用于输入，又能用于输出的类。

fstream 是既能从文件读取数据，又能向文件写入数据的类。

标准流对象

- 输入流对象：cin　　与标准输入设备相连
- 输出流对象：cout　　与标准输出设备相连
  cerr　　与标准错误输出设备相连
  clog　　与标准错误输出设备相连

缺省情况下

```
cerr << "Hello,world" << endl;
clog << "Hello,world" << endl;
```

和

```
cout << “Hello,world” << endl;
```

一样

标准流对象

- cin对应于标准输入流，用于从键盘读取数据，也可以被重定向为从文件中读取数据。
- cout对应于标准输出流，用于向屏幕输出数据，也可以被重定向为向文件写入数据。
- cerr对应于标准错误输出流，用于向屏幕输出出错信息，
- clog对应于标准错误输出流，用于向屏幕输出出错信息，
- cerr和clog的区别在于cerr不使用缓冲区，直接向显示器输出信息；而输出到clog中的信息先会被存放在缓冲区，缓冲区满或者刷新时才输出到屏幕。

判断输入流结束

可以用如下方法判输入流结束：

```
int x;
while(cin>>x){
    ……
}
return 0;
```

```
istream &operator >>(int & a)
    {
        .......
        return *this ;
    }
```

- 如果是从文件输入，比如前面有

  freopen(“some.txt”,”r”,stdin);

  那么，读到文件尾部，输入流就算结束

- 如果从键盘输入，则在单独一行输入Ctrl+Z代表输入流结束

istream类的成员函数

istream & getline(char * buf, int bufSize);
从输入流中读取bufSize-1个字符到缓冲区buf，或读到碰到‘\n’为止（哪个先到算哪个）。
istream & getline(char * buf, int bufSize,char delim);
从输入流中读取bufSize-1个字符到缓冲区buf，或读到碰到delim字符为止（哪个先到算哪个）。

两个函数都会自动在buf中读入数据的结尾添加\0’。，‘\n’或delim都不会被读入buf，但会被从输入流中取走。如果输入流中‘\n’或delim之前的字符个数达到或超过了bufSize个，就导致读入出错，其结果就是：虽然本次读入已经完成，但是之后的读入就都会失败了。

可以用 if(!cin.getline(…)) 判断输入是否结束

istream类的成员函数

bool eof(); 判断输入流是否结束

int peek(); 返回下一个字符,但不从流中去掉.

istream & putback(char c); 将字符ch放回输入流

istream & ignore(int nCount = 1, int delim = EOF);

从流中删掉最多nCount个字符，遇到EOF时结束。

istream类的成员函数

```
#include <iostream >
using namespace std;
int main()  {
        int x;
        char buf[100];
        cin >> x;
        cin.getline(buf,90);
        cout << buf << endl;
        return 0;
}
```

输入：
12 abcd↙
输出：
 abcd   (空格+abcd)

输入
12↙
程序立即结束，无输出：

因为getline读到留在流中的'\n'就会返回

输出重定向

```
#include <iostream>
using namespace std;
int main()   {
    int x,y;
    cin >> x >> y;
    freopen("test.txt","w",stdout);  //将标准输出重定向到 test.txt文件
    if( y == 0 )  //除数为0则在屏幕上输出错误信息
            cerr << "error." << endl;
    else
             cout << x /y ;  //输出结果到test.txt
    return 0;
}
```

输入重定向

```
#include <iostream >
using namespace std;
int main()  {
        double f;          int n;
        freopen(“t.txt”,“r”,stdin);  //cin被改为从 t.txt中读取数据
        cin >> f >> n;
        cout << f << "," <<n << endl;
        return 0;
}
```

t.txt:
3.14 123

输出:
3.14,123

流操纵算子

冰岛众神瀑布

流操纵算子

- 整数流的基数：流操纵算子**dec,oct,hex,setbase**
- 浮点数的精度（ precision,setprecision ）
- 设置域宽(setw,width)
- 用户自定义的流操纵算子

使用流操纵算子需要 #include <iomanip>

流操纵算子

- 整数流的基数：流操纵算子dec, oct, hex

```
int n = 10;
cout << n << endl;
cout << hex << n << "\n"
     << dec << n << "\n"
     << oct << n << endl;
```

输出结果：

```
10
a
10
12
```

控制浮点数精度的流操纵算子

precision, setprecision

- precision是成员函数，其调用方式为：

```
cout.precision(5);
```

- setprecision 是流操作算子，其调用方式为：

```
cout << setprecision(5); // 可以连续输出
```

它们的功能相同。

指定输出浮点数的有效位数（非定点方式输出时）

指定输出浮点数的小数点后的有效位数（定点方式输出时）

定点方式：小数点必须出现在个位数后面

控制浮点数精度的流操纵算子

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
      double x = 1234567.89,y = 12.34567;
      int n = 1234567;
      int m = 12;
      cout << setprecision(6) << x << endl
           << y << endl << n << endl << m;
}
```

浮点数输出最多6位有效数字

输出：

```
1.23457e+006
12.3457
1234567
12
```

控制浮点数精度的流操纵算子

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        double x = 1234567.89,y = 12.34567;
        int n = 1234567;
        int m = 12;
        cout << setiosflags(ios::fixed) <<
                setprecision(6) << x << endl
                << y << endl << n << endl << m;
}
```

保留小数点后几位

以小数点位置固定的方式输出

输出:
1234567.890000
12.345670
1234567
12

控制浮点数精度的流操纵算子

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
      double x = 1234567.89;
      cout << setiosflags(ios::fixed) <<
            setprecision(6) << x << endl <<
            resetiosflags(ios::fixed)  << x ;
}
```

取消以小数点位置固定的方式输出

输出：
1234567.890000
1.23457e+006

设置域宽的流操纵算子

- 设置域宽(setw,width)

两者功能相同，一个是成员函数，另一个是流操作算子，调用方式不同：

cin >> setw(4); 或者 cin.width(5);

cout << setw(4); 或者 cout.width(5);

设置域宽的流操纵算子

- 设置域宽(setw,width)

```
int w = 4;
char string[10];
 cin.width(5);
 while(cin >> string){
     cout.width(w++);
     cout << string << endl;
     cin.width(5);
 }
```

宽度设置有效性是一次性的，在每次读入和输出之前都要设置宽度。

输入:
1234567890
输出:

```
1234
 5678
     90
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
   int n = 141;
   //1) 分别以十六进制、十进制、八进制先后输出 n
   cout << "1) " << hex << n << " " << dec << n << " " << oct << n << endl;
   double x = 1234567.89,y = 12.34567;
   //2) 保留5位有效数字
   cout << "2) " << setprecision(5) << x << " " << y << " " << endl;
   //3) 保留小数点后面5位
   cout << "3) " << fixed << setprecision(5) <<  x << " " << y << endl ;
   //4) 科学计数法输出，且保留小数点后面5位
   cout << "4) " << scientific << setprecision(5) <<x << " " << y << endl ;
```

1) 8d 141 215
2) 1.2346e+006 12.346
3) 1234567.89000 12.34567
4) 1.23457e+006 1.23457e+001

```
  //5) 非负数要显示正号，输出宽度为12字符，宽度不足则用'*'填补
  cout << "5) " << showpos << fixed << setw(12) << setfill('*') << 12.1
       << endl;
  //6) 非负数不显示正号，输出宽度为12字符，宽度不足则右边用填充字符填充
  cout << "6) " << noshowpos << setw(12) << left << 12.1 << endl;
  //7) 输出宽度为12字符，宽度不足则左边用填充字符填充
  cout << "7) " << setw(12) << right << 12.1 << endl;
  //8) 宽度不足时，负号和数值分列左右，中间用填充字符填充
  cout << "8) " << setw(12) << internal << -12.1 << endl;
  cout << "9) " << 12.1 << endl;
  return 0;
}
```

*5) ***+12.10000*

*6) 12.10000****

*7) ****12.10000*

*8) -***12.10000*

9) 12.10000

用户自定义流操纵算子

```
ostream &tab(ostream &output){
    return output << '\t';
}
cout << “aa” << tab << “bb” << endl;
```

输出：aa　　bb

为什么可以？

用户自定义流操纵算子

因为 iostream 里对 << 进行了重载(成员函数)

```
ostream & operator
  <<(  ostream & ( * p ) ( ostream & ) ) ;
```

该函数内部会调用p所指向的函数，且以 *this 作为参数

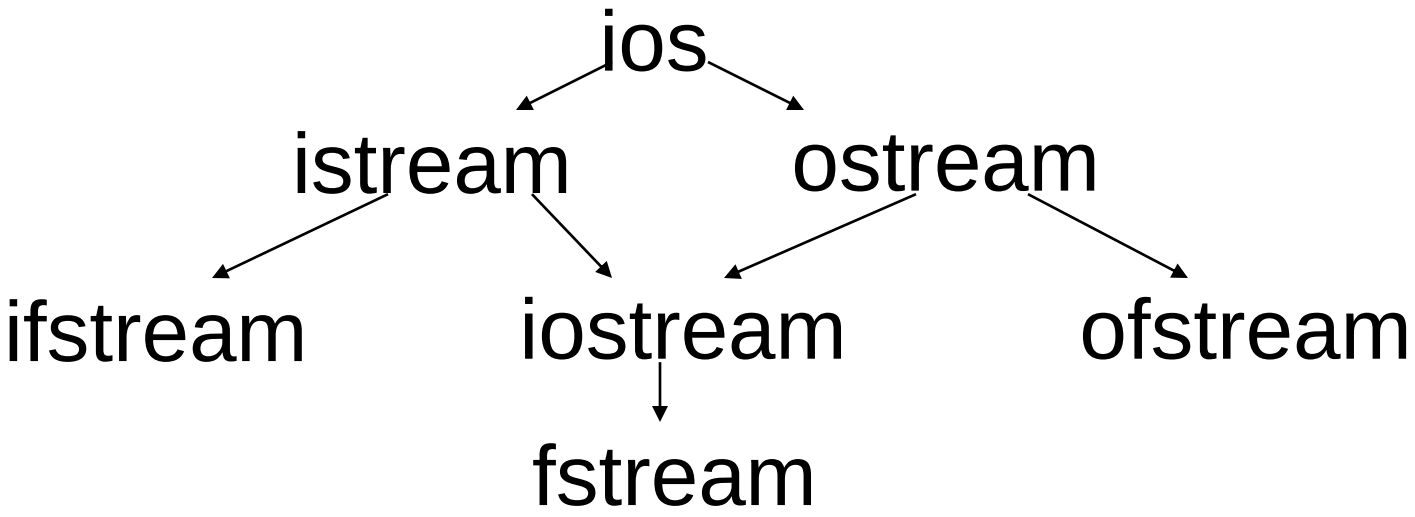
hex 、dec 、oct 都是函数

文件读写

冰岛风光

文件和流

- 可以将顺序文件看作一个有限字符构成的顺序字符流，然后像对 cin, cout 一样的读写。回顾一下输入输出流类的结构层次：



创建文件

- #include <fstream> // 包含头文件

- `ofstream outFile("clients.dat", ios::out|ios::binary);`
  //创建文件

  - clients.dat" 要创建的文件的名字
  - ios::out 文件打开方式
    - ios:out 输出到文件, 删除原有内容
    - ios::app 输出到文件, 保留原有内容，总是在尾部添加
  - ios::binary 以二进制文件格式打开文件

创建文件

- 也可以先创建ofstream对象，再用 open函数打开

```
ofstream fout;
 fout.open("test.out",ios::out|ios::binary);
```

- 判断打开是否成功：

```
if ( !fout ) {
        cout <<  “File open error!” <<endl;
}
```

- 文件名可以给出绝对路径，也可以给相对路径。没有交代路径信息，就是在当前文件夹下找文件

文件名的绝对路径和相对路径

- 绝对路径:

```
"c:\\tmp\\mydir\\some.txt"
```

- 相对路径：

```
"\\tmp\\mydir\\some.txt"
```

当前盘符的根目录下的tmp\dir\some.txt

```
"tmp\\mydir\\some.txt"
```

当前文件夹的tmp子文件夹里面的.....

```
"..\\tmp\\mydir\\some.txt"
```

当前文件夹的父文件夹下面的tmp子文件夹里面的.....

```
"..\\..\\tmp\\mydir\\some.txt"
```

当前文件夹的父文件夹的父文件夹下面的tmp子文件夹里面的.....

文件的读写指针

- 对于输入文件,有一个读指针;
- 对于输出文件,有一个写指针;
- 对于输入输出文件,有一个读写指针;
- 标识文件操作的当前位置, 该指针在哪里,读写操作就在哪里进行。

文件的读写指针

```
ofstream  fout("a1.out",ios::app); //以添加方式打开
long location = fout.tellp();      //取得写指针的位置
location = 10;
fout.seekp(location);              // 将写指针移动到第10个字节处
fout.seekp(location,ios::beg); //从头数location
fout.seekp(location,ios::cur); //从当前位置数location
fout.seekp(location,ios::end); //从尾部数location
```

- location 可以为负值

文件的读写指针

```
ifstream  fin(“a1.in”,ios::ate);
//打开文件，定位文件指针到文件尾
long location = fin.tellg();    //取得读指针的位置
location = 10L;
fin.seekg(location);   // 将读指针移动到第10个字节处
fin.seekg(location,ios::beg); //从头数location
fin.seekg(location,ios::cur); //从当前位置数location
fin.seekg(location,ios::end); //从尾部数location
```

- location 可以为负值

显式关闭文件

```
ifstream fin(“test.dat”,ios::in);
 fin.close();

 ofstream fout(“test.dat”,ios::out);
 fout.close();
```

字符文件读写

- 因为文件流也是流，所以流的成员函数和流操作算子也同样适用于文件流。

- 写一个程序，将文件 in.txt 里面的整数排序后，输出到 out.txt

例如，若in.txt 的内容为：
1 234 9 45 6 879

则执行本程序后，生成的out.txt的内容为：
1 6 9 45 234 879

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;
int main()  {
  vector<int> v;
  ifstream srcFile("in.txt",ios::in);
  ofstream destFile("out.txt",ios::out);
  int x;
  while( srcFile >> x )
          v.push_back(x);
  sort(v.begin(),v.end());
  for( int i = 0;i < v.size();i ++ )
          destFile << v[i] << " ";
  destFile.close();
  srcFile.close();
  return 0;
}
```

二进制文件读写

- 二进制读文件：

ifstream 和 fstream的成员函数：

```
istream& read (char* s, long n);
```

将文件读指针指向的地方的n个字节内容，读入到内存地址s，然后将文件读指针向后移动n字节 (以ios::in方式打开文件时，文件读指针开始指向文件开头) 。

二进制文件读写

- 二进制写文件：

ofstream 和 fstream的成员函数：

```
istream& write (const char* s, long n);
```

将内存地址s处的n个字节内容，写入到文件中写指针指向的位置，然后将文件写指针向后移动n字节(以ios::out方式打开文件时，文件写指针开始指向文件开头, 以ios::app方式打开文件时，文件写指针开始指向文件尾部）。

二进制文件读写

- 在文件中写入和读取一个整数

```
#include <iostream>
#include <fstream>
using namespace std;
int main()  {
   ofstream fout("some.dat", ios::out | ios::binary);
   int x=120;
   fout.write( (const char *)(&x), sizeof(int) );
   fout.close();
   ifstream fin("some.dat",ios::in | ios::binary);
   int y;
   fin.read((char * ) & y,sizeof(int));
   fin.close();
   cout << y <<endl;
   return 0;
}
```

二进制文件读写

- 从键盘输入几个学生的姓名的成绩，并以二进制文件形式保存

```
#include <iostream>
#include <fstream>
using namespace std;
struct Student {
        char name[20];
        int score;
};
int main()  {
   Student s;
   ofstream OutFile( "c:\\tmp\\students.dat",ios::out|ios::binary);
   while( cin >> s.name >> s.score )
        OutFile.write( (char * ) & s, sizeof( s) );
   OutFile.close();
   return 0;
}
```

输入：

Tom 60
Jack 80
Jane 40
^Z+回车

则形成的 students.dat 为 72字节，用 记事本打开，呈现：

Tom 烫烫烫烫烫烫烫烫烫< Jack 烫烫烫烫烫烫烫烫蘲 Jane 烫烫烫烫烫烫烫?

二进制文件读写

- 将 students.dat 文件的内容读出并显示

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

二进制文件读写

```
int main()  {
   Student s;
   ifstream inFile("students.dat",ios::in | ios::binary );
   if(!inFile) {
        cout << "error" <<endl;
        return 0;
   }
   while( inFile.read( (char* ) & s, sizeof(s) ) ) {
        int readedBytes = inFile.gcount(); //看刚才读了多少字节
        cout << s.name << " " << s.score << endl;
   }
   inFile.close();
   return 0;
}
```

输出：

Tom 60

Jack 80

Jane 40

二进制文件读写

- 将 students.dat 文件的Jane的名字改成Mike

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

```
int main()
{
   Student s;
   fstream iofile( "c:\\tmp\\students.dat",
   ios::in|ios::out|ios::binary);
   if( !iofile) {
        cout << "error" ;
        return 0;
   }
   iofile.seekp( 2 * sizeof(s),ios::beg);   //定位写指针到第三个记录
   iofile.write("Mike",strlen("Mike")+1);
   iofile.seekg(0,ios::beg); //定位读指针到开头
   while( iofile.read( (char* ) & s, sizeof(s)) )
        cout << s.name << " " << s.score << endl;
   iofile.close();
   return 0;
}
```

输出：
Tom 60
Jack 80
Mike 40

文件拷贝程序mycopy 示例

```
/*用法示例：
  mycopy src.dat dest.dat
  即将 src.dat 拷贝到 dest.dat   如果 dest.dat 原来就有，则原来的文件会被覆
    盖   */
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char * argv[])
{
   if( argc != 3 ) {
        cout << "File name missing!" << endl;
        return 0;
   }
```

```
ifstream inFile(argv[1],ios::binary|ios::in); //打开文件用于读
if( ! inFile ) {
     cout << "Source file open error." << endl;
     return 0;
}
ofstream outFile(argv[2],ios::binary|ios::out); //打开文件用于写
if( !outFile) {
     cout << "New file open error." << endl;
     inFile.close(); //打开的文件一定要关闭
     return 0;
}
```

```
    char c;
    while( inFile.get(c)) //每次读取一个字符
         outFile.put(c);    //每次写入一个字符
    outFile.close();
    inFile.close();
    return 0;
}
```

二进制文件和文本文件的区别

Linux,Unix下的换行符号： ‘\n’ (ASCII码: 0x0a)
Windows 下的换行符号： ‘\r\n’ (ASCII码： 0x0d0a) endl 就是 '\n'
Mac OS下的换行符号： ‘\r’ (ASCII码：0x0d)

导致 Linux, Mac OS 文本文件在Windows 记事本中打开时不换行

二进制文件和文本文件的区别

- Unix/Linux下打开文件，用不用 ios::binary 没区别
- Windows下打开文件，如果不用 ios::binary，则：

  读取文件时，所有的 '\r\n’ 会被当做一个字符'\n'处理，即少读了一个字符'\r'。

  写入文件时，写入单独的'\n'时，系统自动在前面加一个'\r'，即多写了一个'\r'