原 C++中的文件操作

C++中的文件操作

在 C++ 中，有一个 stream 这个类，所有的 I/O 都以这个“流”类为基础的，包括我们要认识的文件 I/O，stream 这个类有两个重要的运算符：

1、插入器(<<)

向流输出数据。比如说系统有一个默认的标准输出流(cout)，一般情况下就是指的显示器，所以，cout<<"Write Stdout"<<'n'; 就表示把字符串"Write Stdout"和换行字符('n')输出到标准输出流。

2、析取器(>>)

从流中输入数据。比如说系统有一个默认的标准输入流(cin)，一般情况下就是指的键盘，所以，cin>>x; 就表示从标准输入流中读取一个指定类型(即变量 x 的类型)的数据。

在 C++ 中，对文件的操作是通过 stream 的子类 fstream(file stream)来实现的，所以，要用这种方式操作文件，就必须加入头文件 fstream.h。下面就把此类的文件操作过程一一道来。

一、打开文件

在 fstream 类中，有一个成员函数 open()，就是用来打开文件的，其原型是：

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

参数：

filename： 要打开的文件名
mode： 要打开文件的方式
access： 打开文件的属性
打开文件的方式在类 ios(是所有流式 I/O 类的基类)中定义，常用的值如下：

ios::app： 以追加的方式打开文件
ios::ate： 文件打开后定位到文件尾，ios:app 就包含有此属性
ios::binary： 以二进制方式打开文件，缺省的方式是文本方式。两种方式的区别见前文
ios::in： 文件以输入方式打开
ios::out： 文件以输出方式打开
ios::nocreate： 不建立文件，所以文件不存在时打开失败
ios::noreplace：不覆盖文件，所以打开文件时如果文件存在失败
ios::trunc： 如果文件存在，把文件长度设为 0

可以用“或”把以上属性连接起来，如 ios::out|ios::binary

打开文件的属性取值是：

0：普通文件，打开访问
1：只读文件

2：隐含文件
4：系统文件
可以用“或”或者“+”把以上属性连接起来 ，如 3 或 1|2 就是以只读和隐含属性打开文件。

例如：以二进制输入方式打开文件 c:config.sys

```
fstream file1;
file1.open("c:config.sys",ios::binary|ios::in,0);
```

如果 open 函数只有文件名一个参数，则是以读/写普通文件打开，即：

```
file1.open("c:config.sys");<=>file1.open("c:config.sys",ios::in|ios::out,0);
```

另外，fstream 还有和 open()一样的构造函数，对于上例，在定义的时侯就可以打开文件了：

```
fstream file1("c:config.sys");
```

特别提出的是，fstream 有两个子类：ifstream(input file stream)和 ofstream(outpu file stream)，ifstream 默认以输入方式打开文件，而 ofstream 默认以输出方式打开文件。

```
ifstream file2("c:pdos.def");//以输入方式打开文件
ofstream file3("c:x.123");//以输出方式打开文件
```

所以，在实际应用中，根据需要的不同，选择不同的类来定义：如果想以输入方式打开，就用 ifstream 来定义；如果想以输出方式打开，就用 ofstream 来定义；如果想以输入/输出方式来打开，就用 fstream 来定义。

二、关闭文件
打开的文件使用完成后一定要关闭，fstream 提供了成员函数 close()来完成此操作，如：file1.close();就把 file1 相连的文件关闭。

三、读写文件
读写文件分为文本文件和二进制文件的读取，对于文本文件的读取比较简单，用插入器和析取器就可以了；而对于二进制的读取就要复杂些，下要就详细的介绍这两种方式

1、文本文件的读写
文本文件的读写很简单：用插入器(<<)向文件输出；用析取器(>>)从文件输入。假设 file1 是以输入方式打开，file2 以输出打开。示例如下：

```
file2<<"I Love You";//向文件写入字符串"I Love You"
int i;
file1>>i;//从文件输入一个整数值。
```

这种方式还有一种简单的格式化能力，比如可以指定输出为 16 进制等等，具体的格式有以下一些

操纵符 功能 输入/输出
dec 格式化为十进制数值数据 输入和输出
endl 输出一个换行符并刷新此流 输出
ends 输出一个空字符 输出
hex 格式化为十六进制数值数据 输入和输出
oct 格式化为八进制数值数据 输入和输出
setpxecision(int p) 设置浮点数的精度位数 输出

比如要把 123 当作十六进制输出：file1<<hex<<123;要把 3.1415926 以 5 位精度输出：file1<<setpxecision(5)<<3.1415926。

2、二进制文件的读写
①put()
put()函数向流写入一个字符，其原型是 ofstream &put(char ch)，使用也比较简单，如 file1.put('c');就是向流写一个字符'c'。

②get()
get()函数比较灵活，有 3 种常用的重载形式：

一种就是和 put()对应的形式：ifstream &get(char &ch);功能是从流中读取一个字符，结果保存在引用 ch 中，如果到文件尾，返回空字符。如 file2.get(x);表示从文件中读取一个字符，并把读取的字符保存在 x 中。

另一种重载形式的原型是： int get();这种形式是从流中返回一个字符，如果到达文件尾，返回 EOF，如 x=file2.get();和上例功能是一样的。

还有一种形式的原型是：ifstream &get(char *buf,int num,char delim='n')；这种形式把字符读入由 buf 指向的数组，直到读入了 num 个字符或遇到了由 delim 指定的字符，如果没使用 delim 这个参数，将使用缺省值换行符'n'。例如：

file2.get(str1,127,'A');//从文件中读取字符到字符串 str1，当遇到字符'A'或读取了 127 个字符时终止。

③读写数据块
要读写二进制数据块，使用成员函数 read()和 write()成员函数，它们原型如下：

```
read(unsigned char *buf,int num);
write(const unsigned char *buf,int num);
```

read()从文件中读取 num 个字符到 buf 指向的缓存中，如果在还未读入 num 个字符时就到了文件尾，可以用成员函数 int gcount();来取得实际读取的字符数；而 write() 从 buf

指向的缓存写 num 个字符到文件中，值得注意的是缓存的类型是 unsigned char *，有时可能需要类型转换。

例：

```
unsigned char str1[]="I Love You";
int n[5];
ifstream in("xxx.xxx");
ofstream out("yyy.yyy");
out.write(str1,strlen(str1));//把字符串 str1 全部写到 yyy.yyy 中
in.read((unsigned char*)n,sizeof(n));//从 xxx.xxx 中读取指定个整数，注意类型转换
in.close();out.close();
```

四、检测 EOF

成员函数 eof()用来检测是否到达文件尾，如果到达文件尾返回非 0 值，否则返回 0。原型是 int eof();

例： if(in.eof())ShowMessage("已经到达文件尾！");

五、文件定位

和 C 的文件操作方式不同的是，C++ I/O 系统管理两个与一个文件相联系的指针。一个是读指针，它说明输入操作在文件中的位置；另一个是写指针，它下次写操作的位置。每次执行输入或输出时，相应的指针自动变化。所以，C++ 的文件定位分为读位置和写位置的定位，对应的成员函数是 seekg()和 seekp()，seekg()是设置读位置，seekp 是设置写位置。它们最通用的形式如下：

```
istream &seekg(streamoff offset,seek_dir origin);
ostream &seekp(streamoff offset,seek_dir origin);
```

streamoff 定义于 iostream.h 中，定义有偏移量 offset 所能取得的最大值，seek_dir 表示移动的基准位置，是一个有以下值的枚举：

ios::beg： 文件开头
ios::cur： 文件当前位置
ios::end： 文件结尾

这两个函数一般用于二进制文件，因为文本文件会因为系统对字符的解释而可能与预想的值不同。

例：

```
file1.seekg(1234,ios::cur);//把文件的读指针从当前位置向后移 1234 个字节
file2.seekp(1234,ios::beg);//把文件的写指针从文件开头向后移 1234 个字节
```

有了这些知识，我们就可以完成对文件的操作了，当然，还有好多的成员函数我没介绍，但有这些我们已经能完成大多数的需要了，这种文件操作方式是我比较喜欢的一种方法，比 C 的方法灵活，又比 BCB 函数和 WINAPI 函数具有通用性。

原 C++中的文件输入/输出（3）：掌握输入/输出流

C++中的文件输入/输出(3)

原作：Ilia Yordanov， loobian@cpp-home.com

掌握输入/输出流

在这一章里，我会提及一些有用的函数。我将为你演示如何打开一个可以同时进行读、写操作的文件；此外，我还将为你介绍其它打开文件的方法，以及如何判断打开操作是否成功。因此，请接着往下读！

到目前为止，我已为你所展示的只是单一的打开文件的途径：要么为读取而打开，要么为写入而打开。但文件还可以以其它方式打开。迄今，你应当已经认识了下面的方法：

```
ifstream OpenFile("cpp-home.txt");
```

噢，这可不是唯一的方法！正如以前所提到的，以上的代码创建一个类 ifstream 的对象，并将文件的名字传递给它的构造函数。但实际上，还存在有不少的重载的构造函数，它们可以接受不止一个的参数。同时，还有一个 open() 函数可以做同样的事情。下面是一个以上代码的示例，但它使用了 open()函数：

```
ifstream OpenFile;
OpenFile.open("cpp-home.txt");
```

你会问：它们之间有什么区别吗？哦，我曾做了不少测试，结论是没有区别！只不过如果你要创建一个文件句柄但不想立刻给它指定一个文件名，那么你可以使用 open()函数过后进行指定。顺便再给出一个要使用 open()函数的例子：如果你打开一个文件，然后关闭了它，又打算用同一个文件句柄打开另一个文件，这样一来，你将需要使用 open()函数。

考虑以下的代码示例：

```
#include <fstream.h>

void read(ifstream &T) //pass the file stream to the function
{
      //the method to read a file, that I showed you before
    char ch;

    while(!T.eof())
    {
        T.get(ch);
        cout << ch;
    }

    cout << endl << "--------" << endl;
}

void main()
{
    ifstream T("file1.txt");
    read(T);
    T.close();

    T.open("file2.txt");
    read(T);
    T.close();
}
```

据此，只要 file1.txt 和 file2.txt 并存储了文本内容，你将看到这些内容。

现在，该向你演示的是，文件名并不是你唯一可以向 open() 函数或者构造函数（其实都一样）传递的参数。下面是一个函数原型：

```
ifstream OpenFile(char *filename, int open_mode);
```

你应当知道 filename 表示文件的名称（一个字符串），而新出现的则是 open_mode（打开模式）。open_mode 的值用来定义以怎样的方式打开文件。下面是打开模式的列表：

名称	描述
ios::in	打开一个可读取文件
ios::out	打开一个可写入文件
ios::app	你写入的所有数据将被追加到文件的末尾，此方式使用 ios::out
ios::ate	你写入的所有数据将被追加到文件的末尾，此方式不使用 ios::out
ios::trunk	删除文件原来已存在的内容（清空文件）

ios::nocreate	如果要打开的文件并不存在，那么以此参数调用 open()函数将无法进行。
ios::noreplace	如果要打开的文件已存在，试图用 open()函数打开时将返回一个错误。
ios::binary	以二进制的形式打开一个文件。

实际上，以上的值都属于一个枚举类型的 int 常量。但为了让你的编程生涯不至于太痛苦，你可以像上表所见的那样使用那些名称。

下面是一个关于如何使用打开模式的例子：

```
#include <fstream.h>

void main()
{
    ofstream SaveFile("file1.txt", ios::ate);

    SaveFile << "That's new!\n";

    SaveFile.close();
}
```

正如你在表中所看到的：使用 ios::ate 将会从文件的末尾开始执行写入。如果我没有使用它，原来的文件内容将会被重新写入的内容覆盖掉。不过既然我已经使用了它，那么我只会在原文件的末尾进行添加。所以，如果 file1.txt 原有的内容是这样：

Hi! This is test from www.cpp-home.com!

那么执行上面的代码后，程序将会为它添上“That’s new!”，因此它看起来将变成这样：

Hi! This is test from www.cpp-home.com!That's new!

假如你打算设置不止一个的打开模式标志，只须使用 **OR** 操作符或者是 | ，像这样：

```
ios::ate | ios::binary
```

我希望现在你已经明白“打开模式”是什么意思了！

现在，是时候向你展示一些真正有用的东西了！我敢打赌你现在还不知道应当怎样打开一个可以同时进行读取和写入操作的文件！下面就是实现的方法：

```
fstream File("cpp-home.txt",ios::in | ios::out);
```

实际上，这只是一个声明语句。我将在下面数行之后给你一个代码示例。但此时我首先想提及一些你应当知道的内容。

上面的代码创建了一个名为“File”的流式文件的句柄。如你所知，它是 fstream 类的一个对象。当使用 fstream 时，你应当指定 ios::in 和 ios::out

作为文件的打开模式。这样，你就可以同时对文件进行读、写，而无须创建新的文件句柄。噢，当然，你也可以只进行读或者写的操作。那样的话，相应地你应当只使用 ios::in 或者只使用 ios::out —— 要思考的问题是：如果你打算这么做，为什么你不分别用 ifstream 及 ofstream 来实现呢？

下面就先给出示例代码：

```
#include <fstream.h>

void main()
{
    fstream File("test.txt",ios::in | ios::out);

    File << "Hi!"; //将"Hi!"写入文件
    static char str[10]; //当使用 static 时，数组会自动被初始化
                //即是被清空为零



    File.seekg(ios::beg); // 回到文件首部
                  // 此函数将在后面解释
    File >> str;
    cout << str << endl;

    File.close();
}
```

OK，这儿又有一些新东西，所以我将逐行进行解释：

`fstream File("test.txt", ios::in | ios::out);` —— 此行创建一个 fstream 对象，执行时将会以读/写方式打开 test.txt 文件。这意味着你可以同时读取文件并写入数据。

`File << "Hi!";` —— 我打赌你已经知道它的意思了。

`static char str[10];` —— 这将创建一个容量为 10 的字符数组。我猜 static 对你而言或者有些陌生，如果这样就忽略它。这只不过会在创建数组的同时对其进行初始化。

`File.seekg(ios::beg);` —— OK，我要让你明白它究竟会做些什么，因此我将以一些有点儿离题、但挺重要的内容开始我的解释。

还记得它么：

```
while(!OpenFile.eof())
{
    OpenFile.get(ch);
    cout << ch;
}
```

你是不是曾经很想知道那背后真正执行了什么操作？不管是或不是，我都将为你解释。这是一个 while 型循环，它会一直反复，直至程序的操作到达文件的尾端。但这个循环如何知道是否已经到了文件末尾？嗯，当你读文件的时候，会有一个类似于“内置指针（inside-pointer）”的东西，它表明你读取（写入也一样）已经到了文件的哪个位置，就像记事本中的光标。而每当你调用 OpenFile.get(ch)的时候，它会返回当前位置的字符，存储在 ch 变量中，并将这一内置指针向前移动一个字符。因此下次该函数再被调用时，它将会返回下一个字符。而这一过程将不断反复，直到读取到达文件尾。所以，让我们回到那行代码：函数 seekg()将把内置指针定位到指定的位置（依你决定）。你可以使用：

ios::beg —— 可将它移动到文件首端

ios::end —— 可将它移动到文件末端

或者，你可以设定向前或向后跳转的字符数。例如，如果你要向定位到当前位置的 5 个字符以前，你应当写：

```
File.seekg(-5);
```

如果你想向后跳过 40 个字符，则应当写：

```
File.seekg(40);
```

同时，我必须指出，函数 seekg()是被重载的，它也可以带两个参数。另一个版本是这样子的：

```
File.seekg(-5,ios::end);
```

在这个例子中，你将能够读到文件文本的最后 4 个字符，因为：

1）你先到达了末尾（ios::end）

2）你接着到达了末尾的前五个字符的位置（-5）

为什么你会读到 4 个字符而不是 5 个？噢，只须把最后一个看成是“丢掉了”，因为文件最末端的“东西”既不是字符也不是空白符，那只是一个位置（译注：或许 ios::end 所“指”的根本已经超出了文件本身的范围，确切的说它是指向文件最后一个字符的下一个位置，有点类似 STL 中的各个容器的 end 迭代点是指向最后一个元素的下一位置。这样设计可能是便于在循环中实现遍历）。

你现在可能想知道为什么我要使用到这个函数。呃，当我把“Hi”写进文件之后，内置指针将被设为指向其后面……也就是文件的末尾。因此我必须将内置指针设回文件起始处。这就是这个函数在此处的确切用途。

File >> str; —— 这也是新鲜的玩意儿！噢，我确信这行代码让你想起了 cin >> .实际上，它们之间有着相当的关联。此行会从文件中读取一个单词，然后将它存入指定的数组变量中。

例如，如果文件中有这样的文本片断：

Hi! Do you know me?

使用 File >> str，则只会将“Hi!”输出到 str 数组中。你应当已经注意到了，它实际上是将空格作为单词的分隔符进行读取的。

由于我存入文件中的只是单独一个“Hi!”，我不需要写一个 while 循环，那会花费更多的时间来写代码。这就是我使用此方法的原因。顺便说一下，到目前为止，我所使用的读取文件的 while 循环中，程序读文件的方式是一个字符一个字符进行读取的。然而你也可以一个单词一个单词地进行读取，像这样：

```
char str[30]; // 每个单词的长度不能超过 30 个字符
while(!OpenFile.eof())
{
    OpenFile >> str;
    cout << str;
}
```

你也可以一行一行地进行读取，像这样：

```
char line[100]; // 每个整行将会陆续被存储在这里
while(!OpenFile.eof())
{
      OpenFile.getline(line,100); // 100 是数组的大小
      cout << line << endl;
}
```

你现在可能想知道应当使用哪种方法。嗯，我建议你使用逐行读取的方式，或者是最初我提及的逐字符读取的方式。而逐词读取的方式并非一个好的方案，因为它不会读出新起一行这样的信息，所以如果你的文件中新起一行时，它将不会将那些内容新起一行进行显示，而是加在已经打印的文本后面。而使用 getline()或者 get()都将会向你展现出文件的本来面目！

现在，我将向你介绍如何检测文件打开操作是否成功。实现上，好的方法少之又少，我将都会涉及它们。需要注意的是，出现“X”的时候，它实际可以以“o”、 “i”来代替，或者也可以什么都不是（那将是一个 fstream 对象）。

例 1：最通常的作法

```
Xfstream File(“cpp-home.txt”);
if (!File)
{
```

```
        cout << "Error opening the file! Aborting…\n";
        exit(1);
}
```

例 2：如果文件已经被创建，返回一个错误

```
ofstream File("unexisting.txt", ios::nocreate);

if(!File)
{
        cout << "Error opening the file! Aborting…\n";
        exit(1);
}
```

例 3：使用 fail()函数

```
ofstream File("filer.txt", ios::nocreate);

if(File.fail())
{
        cout << "Error opening the file! Aborting…\n";
        exit(1);
}
```

例 3 中的新出现的东西，是 fail()函数。如果有任何输入/输出错误（不是在文件末尾）发生，它将返回非零值。

我也要讲一些我认为非常重要的内容！例如，如果你已经创建一个流文件对象，但你没有进行打开文件操作，像这样：

```
ifstream File; //也可以是一个 ofstream
```

这样，我们就拥有一个文件句柄，但我们仍然没有打开文件。如果你打算迟些打开它，那么可以用 open()函数来实现，我已经在本教程中将它介绍了。但如果在你的程序的某处，你可能需要知道当前的句柄是否关联了一个已经打开的文件，那么你可以用 is_open()来进行检测。如果文件没有打开，它将返回 0 (false)；如果文件已经打开，它将返回 1 (true)。例如：

```
ofstream File1;
File1.open("file1.txt");
cout << File1.is_open() << endl;
```

上面的代码将会返回 1（译注：指 File1.is_open()函数，下句同），因为我们已经打开了一个文件（在第二行）。而下面的代码则会返回 0，这是由于我们没有打开文件，而只是创建了一个流文件句柄：

```
ofstream File1;
cout << File1.is_open() << endl;
```

好啦，这一章讲得够多啦。

 # doodoo 发表于 2004-08-13 11:49:00 IP: 211.137.101.*

写的好啊，应该多发一些这样的基础的文章。

 # 翔龙 发表于 2004-09-01 20:36:00 IP: 219.154.94.*

这些我在书本上早看明白了
我是想知道它能不能当数据库用
我自己的实践是用指针来输入输出数据比较方便 用 get()put()一次只能输入或输出一个字符
用 read() write()要指定写的字符数量 我比较喜欢用重载运算符<< 和 >>
如：

```
#include<iostream.h>
#include<fstream.h>
void main()
{
fstream iofile;
iofile.open("a.txt",ios::in|ios::out);//输入输出类型
char*p=[10000000];//只要你的机器内存够大 你可以把数组写的更大一次就可以写
//入很多字符
cin>>p;//写入字符
iofile<<p;//将字符输出到文件流
iofile>>p;//将输入的字符出入到内存给 p
cout<<pl;// 将输入的字符在屏幕上输出查看
iofile.close();
}
```

我是专学语言 对平台操作不会的 呵呵
我的电子邮件 bianqiwei@yahoo.com.cn 希望高手多多指教

 # ares 发表于 2004-12-13 23:20:00 IP: 222.28.215.*

```
void main(){
char c;
while (c!='q'&&c!='Q'){
cout<<"ok"<<endl;
cin>>c;
}
}
```

这段代码，如果你输入 a b c d e，它会自动执行 5 次，
怎么让它执行一次之后就接受键盘输入？

weiwenjiangatm@163.com
等待高手指教

 # turkeycock 发表于 2006-05-14 12:26:00 IP: 222.90.13.*

表格中对 ios::ate 的解释不对,使用 ate 模式会覆盖原来的数据,使用 app 模式才会在最后追加字符,我使用的 visual studio 2003 环境编译执行的

Trackback: http://tb.blog.csdn.net/TrackBack.aspx?PostId=7379

原 C++中的文件输入/输出（5）：二进制文件的处理

C++中的文件输入/输出(5)

原作：Ilia Yordanov， loobian@cpp-home.com

二进制文件的处理

虽然有规则格式（formatted）的文本（到目前为止我所讨论的所有文件形式）非常有用，但有时候你需要用到无格式（unformatted）的文件——二进制文件。它们和你的可执行程序看起来一样，而与使用<<及>>操作符创建的文件则大不相同。get()函数与 put()函数则赋予你读/写无规则格式文件的能力：要读取一个字节，你可以使用 get()函数；要写入一个字节，则使用 put()函数。你应当回想起 get()——我曾经使用过它。你可能会疑惑为什么当时我们使用它时，输出到屏幕的文件内容看起来是文本格式的？嗯，我猜这是因为我此前使用了<<及>>操作符。

译注：作者的所谓“规则格式文本（formatted text）”即我们平时所说的文本格式，而与之相对的“无格式文件（unformatted files）”即以存储各类数据或可执行代码的非文本格式文件。通常后者需要读入内存，在二进制层次进行解析，而前者则可以直接由预定好的<<及>>操作符进行读入/写出（当然，对后者也可以通过恰当地重载<<及>>操作符实现同样的功能，但这已经不是本系列的讨论范围了）。

get()函数与都各带一个参数：一个 char 型变量（译注：指get()函数）或一个字符（译注：指 put()函数，当然此字符也可以以 char 型变量提供）。

假如你要读/写一整块的数据，那么你可以使用 read()和 write()函数。它们的原型如下：

```
istream &read(char *buf, streamsize num);
ostream &write(const char *buf, streamsize num);
```

对于 read()函数，buf 应当是一个字符数组，由文件读出的数据将被保存在这儿。对于 write()函数，buf 是一个字符数组，它用以存放你要写入文件的数据。对于这两个函数，num 是一个数字，它指定你要从文件中读取/写入的字节数。

假如在读取数据时，在你读取“num”个字节之前就已经到达了文件的末尾，那么你可以通过调用 gcount()函数来了解实际所读出的字节数。此函数会返回最后一次进行的对无格式文件的读入操作所实际读取的字节数。

在给出示例代码之前，我要补充的是，如果你要以二进制方式对文件进行读/写，那么你应当将 ios::binary 作为打开模式加入到文件打开的参数表中。

现在就让我向你展示示例代码，你会看到它是如何运作的。

示例 1：使用 get( )和 put( )

```
#include <fstream.h>

void main()
{
    fstream File("test_file.txt",ios::out | ios::in | ios::binary);

    char ch;
    ch='o';

    File.put(ch); // 将 ch 的内容写入文件

    File.seekg(ios::beg); // 定位至文件首部

    File.get(ch); // 读出一个字符

    cout << ch << endl; // 将其显示在屏幕上

    File.close();
}
```

示例 2：使用 read( )和 write( )

```
#include <fstream.h>
#include <string.h>

void main()
{
    fstream File("test_file.txt",ios::out | ios::in | ios::binary);

    char arr[13];
    strcpy(arr,"Hello World!"); //将 Hello World!存入数组
```

```
    File.write(arr,5); // 将前 5 个字符——"Hello"写入文件

    File.seekg(ios::beg); // 定位至文件首部

    static char read_array[10]; // 在此我将打算读出些数据

    File.read(read_array,3); // 读出前三个字符——"Hel"

    cout << read_array << endl; // 将它们输出

    File.close();
}
```

Trackback: http://tb.blog.csdn.net/TrackBack.aspx?PostId=7382

原 用 C++实现简单的文件 I/O 操作

文件 I/O 在 C++中比烤蛋糕简单多了。 在这篇文章里，我会详细解释 ASCII 和二进制文件的输入输出的每个细节，值得注意的是，所有这些都是用 C++完成的。

一、ASCII 输出

为了使用下面的方法，你必须包含头文件<fstream.h>(译者注：在标准 C++中，已经使用<fstream>取代< fstream.h>，所有的 C++标准头文件都是无后缀的。)。这是<iostream.h>的一个扩展集，提供有缓冲的文件输入输出操作．事实上，<iostream.h> 已经被<fstream.h>包含了，所以你不必包含所有这两个文件，如果你想显式包含他们，那随便你。我们从文件操作类的设计开始，我会讲解如何进行 ASCII I/O 操作。如果你猜是"fstream,"恭喜你答对了！ 但这篇文章介绍的方法,我们分别使用"ifstream"? 和 "ofstream" 来作输入输出。

如果你用过标准控制台流"cin"?和 "cout," 那现在的事情对你来说很简单。 我们现在开始讲输出部分，首先声明一个类对象。 ofstream fout;

这就可以了，不过你要打开一个文件的话，必须像这样调用 ofstream::open()。

```
fout.open("output.txt");
```

你也可以把文件名作为构造参数来打开一个文件．

```
ofstream fout("output.txt");
```

这是我们使用的方法，因为这样创建和打开一个文件看起来更简单．顺便说一句，如果你

要打开的文件不存在，它会为你创建一个，所以不用担心文件创建的问题．现在就输出到文件，看起来和"cout"的操作很像。 对不了解控制台输出"cout"的人，这里有个例子。

```
int num = 150;
char name[] = "John Doe";
fout << "Here is a number: " << num << "\n";
fout << "Now here is a string: " << name << "\n";
```

现在保存文件，你必须关闭文件，或者回写文件缓冲．文件关闭之后就不能再操作了，所以只有在你不再操作这个文件的时候才调用它，它会自动保存文件。 回写缓冲区会在保持文件打开的情况下保存文件，所以只要有必要就使用它。回写看起来像另一次输出，然后调用方法关闭。像这样：

```
fout << flush; fout.close();
```

现在你用文本编辑器打开文件，内容看起来是这样：

```
Here is a number: 150 Now here is a string: John Doe
```

很简单吧！现在继续文件输入，需要一点技巧，所以先确认你已经明白了流操作，对 "<<" 和">>" 比较熟悉了，因为你接下来还要用到他们。继续...

二、ASCII 输入

输入和"cin" 流很像．和刚刚讨论的输出流很像，但你要考虑几件事情。在我们开始复杂的内容之前，先看一个文本：

```
12 GameDev 15.45 L This is really awesome!
```

为了打开这个文件，你必须创建一个 in-stream 对象,?像这样。

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

现在读入前四行．你还记得怎么用"<<" 操作符往流里插入变量和符号吧？好,?在 "<<" (插入)?操作符之后，是">>" (提取) 操作符．使用方法是一样的．看这个代码片段．

```
int number;
float real;
char letter, word[8];
fin >> number; fin >> word; fin >> real; fin >> letter;
```

也可以把这四行读取文件的代码写为更简单的一行。

```
fin >> number >> word >> real >> letter;
```

它是如何运作的呢？文件的每个空白之后，">>" 操作符会停止读取内容，直到遇到另一个>> 操作符．因为我们读取的每一行都被换行符分割开(是空白字符)，">>" 操作符只把这一行的内容读入变量。这就是这个代码也能正常工作的原因。但是，可别忘了文件的最后一行。

This is really awesome!

如果你想把整行读入一个 char 数组，我们没办法用">>"?操作符，因为每个单词之间的空格（空白字符）会中止文件的读取。为了验证：

```
char sentence[101]; fin >> sentence;
```

我们想包含整个句子，"This is really awesome!" 但是因为空白，现在它只包含了 "This". 很明显，肯定有读取整行的方法，它就是 getline()。这就是我们要做的。

```
fin.getline(sentence, 100);
```

这是函数参数．第一个参数显然是用来接受的 char 数组．第二个参数是在遇到换行符之前，数组允许接受的最大元素数量．现在我们得到了想要的结果："This is really awesome!"。

你应该已经知道如何读取和写入 ASCII 文件了。但我们还不能罢休，因为二进制文件还在等着我们。

三、二进制 输入输出

二进制文件会复杂一点，但还是很简单的。首先你要注意我们不再使用插入和提取操作符(译者注：<< 和 >> 操作符)．你可以这么做，但它不会用二进制方式读写。你必须使用 read() 和 write() 方法读取和写入二进制文件．创建一个二进制文件，看下一行。

```
ofstream fout("file.dat", ios::binary);
```

这会以二进制方式打开文件，而不是默认的 ASCII 模式。首先从写入文件开始。函数 write() 有两个参数。 第一个是指向对象的 char 类型的指针，第二个是对象的大小（译者注：字节数）。 为了说明，看例子。

```
int number = 30; fout.write((char *)(&number), sizeof(number));
```

第一个参数写做"(char *)(&number)". 这是把一个整型变量转为 char *指针。如果你不理解，可以立刻翻阅 C++ 的书籍，如果有必要的话。第二个参数写作 "sizeof(number)". sizeof() 返回对象大小的字节数．就是这样!

二进制文件最好的地方是可以在一行把一个结构写入文件。 如果说，你的结构有 12 个不同的成员。用 ASCII?文件，你不得不每次一条的写入所有成员。但二进制文件替你做好了。看这个。

```
struct OBJECT { int number; char letter; } obj;
obj.number = 15;
obj.letter = 'M';
fout.write((char *)(&obj), sizeof(obj));
```

这样就写入了整个结构！接下来是输入．输入也很简单，因为 read()?函数的参数和 write()是完全一样的，使用方法也相同。

```
ifstream fin("file.dat", ios::binary); fin.read((char *)(&obj), sizeof(obj));
```

我不多解释用法，因为它和 write()是完全相同的。二进制文件比 ASCII 文件简单，但有个缺点是无法用文本编辑器编辑。 接着，我解释一下 ifstream 和 ofstream 对象的其他一些方法作为结束．

四、更多方法

我已经解释了 ASCII 文件和二进制文件，这里是一些没有提及的底层方法。

检查文件

你已经学会了 open() 和 close() 方法，不过这里还有其它你可能用到的方法。

方法 good() 返回一个布尔值，表示文件打开是否正确。

类似的，bad() 返回一个布尔值表示文件打开是否错误。 如果出错，就不要继续进一步的操作了。

最后一个检查的方法是 fail()，和 bad()有点相似，但没那么严重。

读文件

方法 get() 每次返回一个字符。

方法 ignore(int,char) 跳过一定数量的某个字符，但你必须传给它两个参数。第一个是需要跳过的字符数。 第二个是一个字符，当遇到的时候就会停止。 例子，

```
fin.ignore(100, '\n');
```

会跳过 100 个字符，或者不足 100 的时候，跳过所有之前的字符，包括 '\n'。

方法 peek() 返回文件中的下一个字符，但并不实际读取它。所以如果你用 peek() 查看下一个字符，用 get() 在 peek()之后读取，会得到同一个字符，然后移动文件计数器。

方法 putback(char) 输入字符，一次一个，到流中。我没有见到过它的使用，但这个函数确实存在。

写文件

只有一个你可能会关注的方法.?那就是 put(char)，它每次向输出流中写入一个字符。

打开文件

当我们用这样的语法打开二进制文件：

```
ofstream fout("file.dat", ios::binary);
```

"ios::binary"是你提供的打开选项的额外标志．默认的，文件以 ASCII 方式打开，不存在则创建，存在就覆盖．这里有些额外的标志用来改变选项。

ios::app 添加到文件尾
ios::ate 把文件标志放在末尾而非起始。
ios::trunc 默认．截断并覆写文件。
ios::nocreate 文件不存在也不创建。
ios::noreplace 文件存在则失败。

文件状态

我用过的唯一一个状态函数是 eof()，它返回是否标志已经到了文件末尾。 我主要用在循环中。 例如，这个代码断统计小写‘e’在文件中出现的次数。

```
ifstream fin("file.txt");
char ch; int counter;
while (!fin.eof()) {
ch = fin.get();
if (ch == ‘e’) counter++;
}
fin.close();
```

我从未用过这里没有提到的其他方法。 还有很多方法，但是他们很少被使用。参考 C++ 书籍或者文件流的帮助文档来了解其他的方法。

结论

你应该已经掌握了如何使用 ASCII 文件和二进制文件。有很多方法可以帮你实现输入输出，尽管很少有人使用他们。我知道很多人不熟悉文件 I/O 操作，我希望这篇文章对你有所帮助。 每个人都应该知道．文件 I/O 还有很多显而易见的方法,?例如包含文件 <stdio.h>．我更喜欢用流是因为他们更简单。 祝所有读了这篇文章的人好运，也许以后我还会为你们写些东西。

原 各类文件管理函数

各类文件管理函数

1.2　文件的输入输出函数

键盘、显示器、打印机、磁盘驱动器等逻辑设备，其输入输出都可以通过文件管理的方法来完成。而在编程时使用最多的要算是磁盘文件，因此本节主要以磁盘文件为主，详细介绍 Turbo C2.0 提供的文件操作函数，当然这些对文件的操作函数也适合于非磁盘文件的情况。

另外，Turbo C2.0 提供了两类关于文件的函数。一类称做标准文件函数也称缓冲型文件函数，这是 ANSI 标准定义的函数；另一类叫非标准文件函数，也称非缓冲型文件函数。这类函数最早公用于 UNIX 操作系统，但现在 MS-DOS3.0 以上版本的操作系统也可以使用。下面分别进行介绍。

1.2.1　标准文件函数

标准文件函数主要包括文件的打开、关闭、读和写等函数。 不象 BASIC 、FORTRAN 语方有顺序文件和随机文件之分， 在打开时就应按不同的方式确定。Turbo C2.0 并不区分这两种文件，但提供了两组函数，即顺序读写函数和随机读写函数。

一、文件的打开和关闭

任何一个文件在使用之前和使用之后，必须要进行打开和关闭，这是因为操作系统对于同时打开的文件数目是有限制的，DOS 操作系统中， 可以在 DEVICE .SYS 中定义允许同时打开的文件数 n(用 files=n 定义)。其中 n 为可同时打开的文件数，一般 n<=20。因此在使用文件前应打开文件，才可对其中的信息进行存取。用完之后需要关闭，否则将会出现一些意想不到的错误。Turbo C2.0 提供了打开和关闭文件的函数。

1. fopen()函数

fopen 函数用于打开文件，其调用格式为：

FILE *fopen(char *filename, *type);

在介绍这个函数之;前，先了解一下下面的知识。

(1) 流(stream)和文件(file)

流和文件 在 Turbo C2.0 中是有区别的，Turbo C2.0 为编程者和被访问的设备之间提供了一层抽象的东西，称之为"流"，而将具体的实际设备叫做文件 。流是一个逻辑设备，具有相同的行为。因此，用来进行磁盘文件写的函数也同样可以用来进行打印机的写入。在 Turbo C2.0 中有两种性质的流： 文字流(text stream)和二进制(binary stream)。对磁盘来说就是文本文件和二进制文件。本软件为了便于让读者易理解 Turbo C2.0 语言而没有对流和文件作特别区分。

(2) 文件指针 FILE

实际上 FILE 是一个新的数据类型。它是 Turbo C2.0 的基本数据类型的集合，

称之为结构指针。有关结构的概念将在第四节中详细介绍，这里只要将 FILE 理解为一个包括了文件管理有关信息的数据结构，即在打开文件时必须先定义一个文件指针。

(3) 以后介绍的函数调用格式将直接写出形式参数的数据类型和函数返回值的数据类型。例如：上面打开文件的函数，返回一个文件指针，其中形式参数有两个，均为字符型变量(字符串数组或字符串指针)。本软件不再对函数的调用格式作详细说明。

现在再来看打开文件函数的用法。

fopen()函数中第一个形式参数表示文件名，可以包含路径和文件名两部分。如：

"B:TEST.DAT"

"C:\\TC\\TEST.DAT"

如果将路径写成"C:\TC\TEST.DAT"是不正确的，这一点要特别注意。

第二个形式参数表示打开文件的类型。关于文件类型的规定参见下表。

表　文件操作类型

字符	含义
"r"	打开文字文件只读
"w"	创建文字文件只写
"a"	增补，如果文件不存在则创建一个
"r+"	打开一个文字文件读/写
"w+"	创建一个文字文件读/写
"a+"	打开或创建一个文件增补
"b"	二进制文件(可以和上面每一项合用)
"t"	文这文件(默认项)

如果要打开一个 CCDOS 子目录中，文件名为 CLIB 的二进制文件，可写成：

fopen("c:\\ccdos\\clib", "rb");

如果成功的打开一个文件，fopen()函数返回文件指针，　否则返回空指针(NULL)。由此可判断文件打开是否成功。

2. fclose()函数

fclose()函数用来关闭一个由 fopen()函数打开的文件 ，其调用格式为：

int fclose(FILE *stream);

该函数返回一个整型数。当文件关闭成功时，返回 0，　否则返回一个非零值。可以根据函数的返回值判断文件是否关闭成功。

例 10：

```
#iclude
main()
{
    FILE *fp;                    /*定义一个文件指针*/
    int i;
    fp=fopen("CLIB", "rb");   /*打开当前目录名为 CLIB 的文件只读*/
    if(fp==NULL)                 /*判断文件是否打开成功*/
```

```
        puts("File open error");/*提示打开不成功*/
    i=fclose(fp);                /*关闭打开的文件*/
    if(i==0)                     /*判断文件是否关闭成功*/
      printf("O,K");             /*提示关闭成功*/
    else
      puts("File close error");/*提示关闭不成功*/
}
```

二、有关文件操作的函数

本节所讲的文件读写函数均是指顺序读写，即读写了一条信息后，指针自动加 1。下面分别介绍写操作函数和读操作函数。

1. 文件的顺序写函数

fprintf()、fputs()和 fputc()函数

函数 fprintf()、fputs()和 fputc()均为文件的顺序写操作函数， 其调用格式如下:

```
int fprintf(FILE *stream, char *format, );
int fputs(char *string, FILE *steam);
int fputc(int ch, FILE *steam);
```

上述三个函数的返回值均为整型量。fprintf() 函数的返回值为实际写入文件中的字罕个数(字节数)。如果写错误，则返回一个负数，fputs()函数返回 0 时表明将 string 指针所指的字符串写入文件中的操作成功，返回非 0 时， 表明写操作失败。fputc()函数返回一个向文件所写字符的值，此时写操作成功， 否则返回 EOF(文件结束结束其值为-1，在 stdio.h 中定义)表示写操作错误。

fprintf( ) 函数中格式化的规定与 printf( ) 函数相同， 所不同的只是 fprintf()函数是向文件中写入。而 printf()是向屏幕输出。

下面介绍一个例子，运行后产后一个 test.dat 的文件。

例 11:

```
#include
main()
{
    char *s="That's good news");  /*定义字符串指针并初始化*/
    int i=617;                    /*定义整型变量并初始化*/
    FILE *fp;                     /*定义文件指针*/
    fp=fopne("test.dat", "w");    /*建立一个文字文件只写*/
    fputs("Your score of TOEFLis", fp);/*向所建文件写入一串字符*/
    fputc(':', fp);               /*向所建文件写冒号:*/
    fprintf(fp, "%d\n", i);       /*向所建文件写一整型数*/
    fprintf(fp, "%s", s);         /*向所建文件写一字符串*/
    fclose(fp);                   /*关闭文件*/
}
```

用 DOS 的 TYPE 命令显示 TEST.DAT 的内容如下所示:

屏幕显示

```
Your score of TOEFL is: 617
```

That's good news

2. 文件的顺序读操作函数

fscanf()、fgets()和 fgetc()函数

函数 fscanf()、fgets()和 fgetc()均为文件的顺序读操作函数，其调用格式如下:

```
int fscanf(FILE *stream, char *format, );
char fgets(char *string, int n, FILE *steam);
int fgetc(FILE *steam);
```

fscanf()函数的用法与 scanf()函数相似， 只是它是从文件中读到信息。fscanf()函数的返回值为 EOF(即-1)，表明读错误，否则读数据成功。fgets()函数从文件中读取至多 n-1 个字符(n 用来指定字符数)，并把它们放入 string 指向的字符串中，在读入之后自动向字符串末尾加一个空字符，读成功返回 string 指针，失败返回一个空指针。fgetc()函数返回文件当前位置的一个字符， 读错误时返回 EOF。

下面程序读取例 11 产生的 test.dat 文件，并将读出的结果显示在屏幕上。

例 12

```
#include
main()
{
    char *s, m[20];
    int i;
    FILE  *fp;
    fp=fopen("test.dat", "r");      /*打开文字文件只读*/
    fgets(s, 24, fp);               /*从文件中读取 23 个字符*/
    printf("%s", s);                /*输出所读的字符串*/
    fscanf(fp, "%d", &i);           /*读取整型数*/
    printf("%d", i);                /*输出所读整型数*/
    putchar(fgetc(fp));             /*读取一个字符同时输出*/
    fgets(m, 17, fp);               /*读取 16 个字符*/
    puts(m);                        /*输出所读字符串*/
    fclose(fp);                     /*关闭文件*/
    getch();                        /*等待任一键*/
}
```

运行后屏幕显示:

```
Your score of TOEFL is: 617
That's good news
```

如果将上例中 fscanf(fp, "%d", &i)改为 fscanf(fp, "%s", m)， 再将其后的输出语句改为 printf("%s", m)，则可得出同样的结果。由此可见 Turbo C2. 0 中只要是读文字文件，则不论是字符还是数字都将按其 ASCII 值处理。 另外还要说明的一点就是 fscanf()函数读到空白符时，便自动结束，在使用时要特别注意。

3. 文件的随机读写

有时用户想直接读取文件中间某处的信息，若用文件的顺序读写必须从文件

头开始直到要求的文件位置再读，这显然不方便。Turbo C2.0 提供了一组文件的随机读写函数，即可以将文件位置指针定位在所要求读写的地方直接读写。

文件的随机读写函数如下:

```
int fseek (FILE *stream, long offset, int fromwhere);
int fread(void *buf, int size, int count, FILE *stream);
int fwrite(void *buf, int size, int count, FILE *stream);
long ftell(FILE *stream);
```

fseek()函数的作用是将文件的位置指针设置到从 fromwhere 开始的第 offset 字节的位置上，其中 fromwhere 是下列几个宏定义之一:

文件位置指针起始计算位置 fromwhere

符号常数	数值	含义
SEEK_SET	0	从文件开头
SEEK_CUR	1	从文件指针的现行位置
SEEK_END	2	从文件末尾

offset 是指文件位置指针从指定开始位置(fromwhere 指出的位置)跳过的字节数。它是一个长整型量，以支持大于 64K 字节的文件。fseek()函数一般用于对二进制文件进行操作。

当 fseek()函数返回 0 时表明操作成功，返回非 0 表示失败。

下面程序从二进制文件 test_b.dat 中读取第 8 个字节。

例 13:

```
#include
main()
{
      FILE *fp;
      if((fp=fopen("test_b.dat", "rb"))==NULL)
        {
          printf("Can't open file");
          exit(1);
        }
      fseek(fp, 8. 1, SEEK_SET);
      fgetc(fp);
      fclose(fp);
}
```

fread()函数是从文件中读 count 个字段，每个字段长度为 size 个字节，并把它们存放到 buf 指针所指的缓冲器中。

fwrite()函数是把 buf 指针所指的缓冲器中，长度为 size 个字节的 count 个字段写到 stream 指向的文件中去。

随着读和写字节数的增大，文件位置指示器也增大，读多少个字节，文件位置指示器相应也跳过多少个字节。读写完毕函数返回所读和所写的字段个数。

ftell()函数返回文件位置指示器的当前值， 这个值是指示器从文件头开始算起的字节数，返回的数为长整型数，当返回-1 时，表明出现错误。

下面程序把一个浮点数组以二进制方式写入文件 test_b.dat 中。

例 14:

```
#include
main()
{
    float f[6]={3.2, -4.34, 25.04, 0.1, 50.56, 80.5};
                    /*定义浮点数组并初始化*/
    int i;
    FILE *fp;
    fp=fopen("test_b.dat", "wb"); /*创建一个二进制文件只写*/
    fwrite(f, sizeof(float), 6, fp);/*将 6 个浮点数写入文件中*/
    fclose(fp);                    /*关闭文件*/
}
```

下面例子从 test_b.dat 文件中读 100 个整型数，并把它们放到 dat 数组中。

例 15:

```
#include
main()
{
    FILE *fp;
    int dat[100];
    fp=fopen("test_b.dat", "rb");/*打开一个二进制文件只读*/
    if(fread(dat, sizeof(int), 100, fp)!=100)
                                    /*判断是否读了 100 个数*/
      {
        if(feof(fp))
          printf("End of file"); /*不到 100 个数文件结束*/
        else
          printf("Read error");  /*读数错误*/
    fclose(fp);                    /*关闭文件*/
}
```

注意:

当用标准文件函数对文件进行读写操作时，首先将所读写的内容放进缓冲区，即写函数只对输出缓冲区进行操作，读函数只对输入缓冲区进行操作。例如向一个文件写入内容，所写的内容将首先放在输出缓冲区中，直到输出缓冲区存满或使用 fclose()函数关闭文件时，缓冲区的内容才会写入文件中。若无 fclose()函数，则不会向文件中存入所写的内容或写入的文件内容不全。有一个对缓冲区进行刷新的函数，即 fflush()，其调用格式为:

```
int fflush(FILE *stream);
```

该函数将输出缓冲区的内容实际写入文件中，而将输入缓冲区的内容清除掉。

4. feof()和 rewind()函数

这两个函数的调用格式为:

```
int feof(FILE *stream);
int rewind(FILE *stream);
```

feof()函数检测文件位置指示器是否到达了文件结尾， 若是则返回一个非 0 值，否则返回 0。这个函数对二进制文件操作特别有用，因为二进制文件中， 文件结尾标志 EOF 也是一个合法的二进制数， 只简单的检查读入字符的值来判断文件是否结束是不行的。如果那样的话，可能会造成文件未结尾而被认为结尾，所以就必须有 feof()函数。

下面的这条语句是常用的判断文件是否结束的方法。

```
while(!feof(fp))
    fgetc(fp);
```

while 为循环语句，将在下面介绍。

rewind()函数用于把文件位置指示器移到文件的起点处，成功时返回 0， 否则，返回非 0 值。

1.2.2 非标准文件函数

这类函数最早用于 UNIX 操作系统，ANSI 标准未定义， 但有时也经常用到，DOS 3.0 以上版本支持这些函数。它们的头文件为 io.h。

一、文件的打开和关闭

1. open()函数

open()函数的作用是打开文件，其调用格式为：

```
int open(char *filename, int access);
```

该函数表示按 access 的要求打开名为 filename 的文件，返回值为文件描述字，其中 access 有两部分内容：基本模式和修饰符，两者用" "("或")方式连接。 修饰符可以有多个，但基本模式只能有一个。access 的规定如表 3-2。

表 3-2 access 的规定

基本模式	含义	修饰符	含 义
O_RDONLY	只读	O_APPEND	文件指针指向末尾
O_WRONLY	只写	O_CREAT	文件不存在时创建文件，属性按基本模式属性
O_RDWR	读写	O_TRUNC	若文件存在，将其长度缩为 0，属性不变
		O_BINARY	打开一个二进制文件
		O_TEXT	打开一个文字文件

open()函数打开成功，返回值就是文件描述字的值(非负值)，否则返回-1。

2. close()函数

close()函数的作用是关闭由 open()函数打开的文件，其调用格式为：

```
int close(int handle);
```

该函数关闭文件描述字 handle 相连的文件。

二、读写函数

1. read()函数

read()函数的调用格式为:

int read(int handle, void *buf, int count);

read()函数从handle(文件描述字)相连的文件中，读取count个字节放到buf所指的缓冲区中，返回值为实际所读字节数，返回-1表示出错。返回0 表示文件结束。

2. write()函数

write()函数的调用格式为:

int write(int handle, void *buf, int count);

write()函数把count个字节从buf指向的缓冲区写入与handle相连的文件中，返回值为实际写入的字节数。

三、随机定位函数

1. lseek()函数

lseek()函数的调用格式为:

int lseek(int handle, long offset, int fromwhere);

该函数对与handle相连的文件位置指针进行定位，功能和用法与fseek() 函数相同。

2. tell()函数

tell()函数的调用格式为:

long tell(int handle);

该函数返回与handle相连的文件现生位置指针，功能和用法与ftell()相同。

转 读写文件精华

ASCII 输出：

```
ofstream fout;
fout.open("output.txt");
//ofstream fout("output.txt");
int num = 150;
char name[] = "John Doe";
fout << "Here is a number: " << num << "\n";
fout << "Now here is a string: " << name << "\n";
fout << flush;
fout.close();
//Here is a number: 150 Now here is a string: John Doe
```

ASCII 输入：

```
//12 GameDev 15.45 L This is really awesome!
ifstream fin("input.txt");
int number;
float real;
char letter, word[8];
fin >> number; fin >> word; fin >> real; fin >> letter;
//fin >> number >> word >> real >> letter;
```

文件的每个空白之后，">>" 操作符会停止读取内容，直到遇到另一个>>操作符. 因为我们读取的每一行都被换行符分割开(是空白字符), ">>" 操作符只把这一行的内容读入变量。这就是这个代码也能正常工作的原因。

如果你想把整行读入一个 char 数组，我们没办法用">>"?操作符，因为每个单词之间的空格（空白字符）会中止文件的读取。为了验证：

```
fin.getline(sentence, 100);
```

二进制 输入输出

```
ofstream fout("file.dat", ios::binary);
int number = 30; fout.write((char *)(&number), sizeof(number));
```

二进制文件最好的地方是可以在一行把一个结构写入文件。 如果说，你的结构有 12 个不同的成员。 用 ASCII?文件，你不得不每次一条的写入所有成员。 但二进制文件替你做好了。 看这个。

```
struct OBJECT { int number; char letter; } obj;
obj.number = 15;
obj.letter = 'M';
fout.write((char *)(&obj), sizeof(obj));
```

这样就写入了整个结构! 接下来是输入. 输入也很简单

```
ifstream fin("file.dat", ios::binary); fin.read((char *)(&obj), sizeof(obj));
```

更多方法

检查文件

你已经学会了 open() 和 close() 方法，不过这里还有其它你可能用到的方法。

方法 good() 返回一个布尔值，表示文件打开是否正确。

类似的，bad() 返回一个布尔值表示文件打开是否错误。 如果出错，就不要继续进一步的操作了。

最后一个检查的方法是 fail()，和 bad()有点相似，但没那么严重。

读文件

方法 get() 每次返回一个字符。

方法 ignore(int,char) 跳过一定数量的某个字符，但你必须传给它两个参数。第一个是需要跳过的字符数。 第二个是一个字符，当遇到的时候就会停止。 例子，

fin.ignore(100, '\n');
会跳过 100 个字符，或者不足 100 的时候，跳过所有之前的字符，包括 '\n'。
方法 peek() 返回文件中的下一个字符，但并不实际读取它。所以如果你用 peek() 查看下一个字符，用 get() 在 peek()之后读取，会得到同一个字符，然后移动文件计数器。
方法 putback(char) 输入字符，一次一个，到流中。我没有见到过它的使用，但这个函数确实存在。

写文件
只有一个你可能会关注的方法.?那就是 put(char)，它每次向输出流中写入一个字符。

打开文件
当我们用这样的语法打开二进制文件:

```
ofstream fout("file.dat", ios::binary);
```

"ios::binary"是你提供的打开选项的额外标志．默认的，文件以 ASCII 方式打开，不存在则创建，存在就覆盖．这里有些额外的标志用来改变选项。

ios::app 添加到文件尾
ios::ate 把文件标志放在末尾而非起始。
ios::trunc 默认．截断并覆写文件。
ios::nocreate 文件不存在也不创建。
ios::noreplace 文件存在则失败。

文件状态
我用过的唯一一个状态函数是 eof()，它返回是否标志已经到了文件末尾。 我主要用在循环中。 例如，这个代码断统计小写'e' 在文件中出现的次数。

```
ifstream fin("file.txt");
char ch; int counter;
while (!fin.eof()) {
    ch = fin.get();
    if (ch == 'e') counter++;
}
fin.close();
```

我从未用过这里没有提到的其他方法。还有很多方法，但是他们很少被使用。参考 C++ 书籍或者文件流的帮助文档来了解其他的方法。

结论
你应该已经掌握了如何使用 ASCII 文件和二进制文件。有很多方法可以帮你实现输入输出，尽管很少有人使用他们。 我知道很多人不熟悉文件 I/O 操作，我希望这篇文章对你有所帮助。 每个人都应该知道．文件 I/O 还有很多显而易见的方法,?例如包含文件 <stdio.h>．我更喜欢用流是因为他们更简单。 祝所有读了这篇文章的人好运，也许以后我还会为你们写些东西。

特殊

get()成员函数会在文件读到默尾的时候返回假值，所以我们可以利用它的这个特性作为while 循环的终止条件，

```
 string s;
 getline( cin, s );
cout << "You entered " << s << endl;

//s input to buffer .
in:123 456 789
out:you entered 123 456 789


char c[10];

  cin.getline( &c[0], 5, 'a ' );
  cout << c << endl;

// max is 5, when 'a '(<5) is come , input is over .




getline ();



there is a way to read in the whole line, and it is the method getline(). This is ho
w we would do it.


fin.getline(sentence, 100);


Here are the parameters to the function. The first parameter is obviously the char
 array we want to read in to. The second is the maximum number of characters w
e will read in until we encounter a new line. So now sentence contains "This is rea
lly awesome!" just like we wanted.
```

memset()
///Sets buffers to a specified character.

char buffer[] = "This is a test of the memset function";

```
   printf( "Before: %s\n", buffer );
   memset( buffer, '*', 4 );
   printf( "After:   %s\n", buffer );
```

```
//char buf[30];
//memset( buf, '\0', 30 );    //set '\0' to buffer
//string str = "Trying is the first step towards failure.";
//str.copy( buf, 1 );
//cout << buf << endl;
```