C++文件操作详解（ifstream、ofstream、fstream）

C++ 通过以下几个类支持文件的输入输出：

- ofstream: 写操作（输出）的文件类 (由 ostream 引申而来)
- ifstream: 读操作（输入）的文件类(由 istream 引申而来)
- fstream: 可同时读写操作的文件类 (由 iostream 引申而来)

打开文件(Open a file)

对这些类的一个对象所做的第一个操作通常就是将它和一个真正的文件联系起来，也就是说打开一个文件。被打开的文件在程序中由一个流对象(stream object)来表示 (这些类的一个实例) ，而对这个流对象所做的任何输入输出操作实际就是对该文件所做的操作。

要通过一个流对象打开一个文件，我们使用它的成员函数 open()：

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

这里 filename 是一个字符串，代表要打开的文件名，mode 是以下标志符的一个组合：

ios::in	为输入(读)而打开文件
ios::out	为输出(写)而打开文件
ios::ate	初始位置：文件尾
ios::app	所有输出附加在文件末尾
ios::trunc	如果文件已存在则先删除该文件
ios::binary	二进制方式

这些标识符可以被组合使用，中间以”或”操作符(|)间隔。例如，如果我们想要以二进制方式打开文件"example.bin" 来写入一些数据，我们可以通过以下方式调用成员函数 open（）来实现：

```
ofstream file;
file.open ("example.bin", ios::out | ios::app | ios::binary);
```

ofstream, ifstream 和 fstream 所有这些类的成员函数 open 都包含了一个默认打开文件的方式，这三个类的默认方式各不相同：

类	参数的默认方式
ofstream	ios::out | ios::trunc
ifstream	ios::in
fstream	ios::in | ios::out

只有当函数被调用时没有声明方式参数的情况下，默认值才会被采用。如果函数被调用时声明了任何参数，默认值将被完全改写，而不会与调用参数组合。

由于对类 ofstream, ifstream 和 fstream 的对象所进行的第一个操作通常都是打开文件，这些类都有一个构造函数可以直接调用 open 函数，并拥有同样的参数。这样，我们就可以通过以下方式进行与上面同样的定义对象和打开文件的操作：

```
ofstream file ("example.bin", ios::out | ios::app |
ios::binary);
```

两种打开文件的方式都是正确的。

你可以通过调用成员函数 **is_open()**来检查一个文件是否已经被顺利的打开了：

```
bool is_open();
```

它返回一个布尔**(bool)**值，为真（**true**）代表文件已经被顺利打开，假**(false)**则相反。

关闭文件**(Closing a file)**

当文件读写操作完成之后，我们必须将文件关闭以使文件重新变为可访问的。关闭文件需要调用成员函数 **close()**，它负责将缓存中的数据排放出来并关闭文件。它的格式很简单：

```
void close ();
```

这个函数一旦被调用，原先的流对象**(stream object)**就可以被用来打开其它的文件了，这个文件也就可以重新被其它的进程**(process)**所有访问了。

为防止流对象被销毁时还联系着打开的文件，析构函数**(destructor)**将会自动调用关闭函数 **close**。

文本文件**(Text mode files)**

类 **ofstream, ifstream** 和 **fstream** 是分别从 **ostream, istream** 和 **iostream** 中引申而来的。这就是为什么 **fstream** 的对象可以使用其父类的成员来访问数据。

一般来说，我们将使用这些类与同控制台(console)交互同样的成员函数(**cin** 和 **cout**)来进行输入输出。如下面的例题所示，我们使用重载的插入操作符<<：

```
// writing on a text file
#include <fiostream.h>

int main () {
    ofstream examplefile
("example.txt");
    if
(examplefile.is_open()) {
        examplefile << "This
is a line.\n";
        examplefile << "This
is another line.\n";
        examplefile.close();
    }
    return 0;
```

file example.txt

This is a line.

This is another line.

```
}
```

从文件中读入数据也可以用与 **cin** 的使用同样的方法：

```
// reading a text file
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

int main () {
    char buffer[256];
    ifstream examplefile
("example.txt");
    if (!
examplefile.is_open())
    { cout << "Error opening
file"; exit (1); }
    while (!
examplefile.eof() ) {
        examplefile.getline
(buffer,100);
        cout << buffer <<
endl;
```

This is a line.

This is another line.

```
    }
    return 0;
}
```

上面的例子读入一个文本文件的内容，然后将它打印到屏幕上。注意我们使用了一个新的成员函数叫做 **eof** ，它是 **ifstream** 从类 **ios** 中继承过来的，当到达文件末尾时返回 **true** 。

状态标志符的验证(Verification of state flags)

除了 **eof()**以外，还有一些验证流的状态的成员函数（所有都返回 **bool** 型返回值）：

- **bad()**

  如果在读写过程中出错，返回 **true** 。例如：当我们要对一个不是打开为写状态的文件进行写入时，或者我们要写入的设备没有剩余空间的时候。

- **fail()**

  除了与 **bad()** 同样的情况下会返回 **true** 以外，加上格式错误时也返回 **true** ，例如当想要读入一个整数，而获得了一个字母的时候。

- **eof()**

  如果读文件到达文件末尾，返回 **true**。

- good()

这是最通用的：如果调用以上任何一个函数返回 **true** 的话，此函数返回 **false** 。

要想重置以上成员函数所检查的状态标志，你可以使用成员函数 **clear()**，没有参数。

获得和设置流指针(get and put stream pointers)

所有输入/输出流对象(i/o streams objects)都有至少一个流指针：

- **ifstream**， 类似 **istream**，有一个被称为 **get pointer** 的指针，指向下一个将被读取的元素。
- **ofstream**，类似 **ostream**，有一个指针 **put pointer** ，指向写入下一个元素的位置。
- **fstream**，类似 **iostream**，同时继承了 **get** 和 **put**

我们可以通过使用以下成员函数来读出或配置这些指向流中读写位置的流指针：

- tellg() 和 tellp()

这两个成员函数不用传入参数，返回 **pos_type** 类型的值(根据 **ANSI-C++** 标准) ，就是一个整数，代表当前 **get** 流指针的位置 (用 **tellg**) 或 **put** 流指针的位置(用 **tellp**).

- seekg() 和 seekp()

这对函数分别用来改变流指针 **get** 和 **put** 的位置。两个函数都被重载为两种不同的原型：

seekg (pos_type position);

seekp (pos_type position);

使用这个原型，流指针被改变为指向从文件开始计算的一个绝对位置。要求传入的参数类型与函数 **tellg** 和 **tellp** 的返回值类型相同。

seekg (off_type offset, seekdir direction);

seekp (off_type offset, seekdir direction);

使用这个原型可以指定由参数 **direction** 决定的一个具体的指针开始计算的一个位移(**offset**)。它可以是：

ios::beg	从流开始位置计算的位移
ios::cur	从流指针当前位置开始计算的位移
ios::end	从流末尾处开始计算的位移

流指针 **get** 和 **put** 的值对文本文件(**text file**)和二进制文件(**binary file**)的计算方法都是不同的，因为文本模式的文件中某些特殊字符可能被修改。由于这个原因，建议对以文本文件模式打开的文件总是使用 **seekg** 和 **seekp** 的第一种原型，而且不要对 **tellg** 或 **tellp** 的返回值进行修改。对二进制文件，你可以任意使用这些函数，应该不会有任何意外的行为产生。

以下例子使用这些函数来获得一个二进制文件的大小：

```
// obtaining file size
```

size of example.txt is 40

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

    const char * filename =
"example.txt";

    int main () {
        long l,m;
        ifstream file (filename,
ios::in|ios::binary);
        l = file.tellg();
        file.seekg (0, ios::end);
        m = file.tellg();
        file.close();
        cout << "size of " <<
filename;
        cout << " is " << (m-l) <<
" bytes.\n";
        return 0;
    }
```

bytes.

二进制文件(Binary files)

在二进制文件中，使用<< 和>>，以及函数（如 getline）来操作符输入和输出数据，没有什么实际意义，虽然它们是符合语法的。

文件流包括两个为顺序读写数据特殊设计的成员函数：write 和 read。第一个函数 (write) 是 ostream 的一个成员函数，都是被 ofstream 所继承。而 read 是 istream 的一个成员函数，被 ifstream 所继承。类 fstream 的对象同时拥有这两个函数。它们的原型是：

write (char * buffer, streamsize size);

read (char * buffer, streamsize size);

这里 buffer 是一块内存的地址，用来存储或读出数据。参数 size 是一个整数值，表示要从缓存（buffer）中读出或写入的字符数。

```
// reading binary file
#include <iostream>
#include <fstream.h>

const char * filename =
"example.txt";

int main () {
    char * buffer;
    long size;
```

The complete file is in a buffer

```
        ifstream file (filename,
ios::in|ios::binary|ios::ate);
        size = file.tellg();
        file.seekg (0, ios::beg);
        buffer = new char [size];
        file.read (buffer, size);
        file.close();

        cout << "the complete file
is in a buffer";

        delete[] buffer;
        return 0;
    }
```

缓存和同步(Buffers and Synchronization)

当我们对文件流进行操作的时候，它们与一个 **streambuf** 类型的缓存(**buffer**)联系在一起。这个缓存（**buffer**）实际是一块内存空间，作为流(**stream**)和物理文件的媒介。例如，对于一个输出流， 每次成员函数 **put** (写一个单个字符)被调用，这个字符不是直接被写入该

输出流所对应的物理文件中的，而是首先被插入到该流的缓存（**buffer**）中。

当缓存被排放出来**(flush)**时，它里面的所有数据或者被写入物理媒质中（如果是一个输出流的话），或者简单的被抹掉(如果是一个输入流的话)。这个过程称为同步**(synchronization)**，它会在以下任一情况下发生：

- 当文件被关闭时：在文件被关闭之前，所有还没有被完全写出或读取的缓存都将被同步。
- 当缓存 **buffer** 满时:缓存 **Buffers** 有一定的空间限制。当缓存满时，它会被自动同步。
- 控制符明确指明:当遇到流中某些特定的控制符时，同步会发生。这些控制符包括：**flush** 和 **endl**。
- 明确调用函数 **sync()**：调用成员函数 **sync()** (无参数)可以引发立即同步。这个函数返回一个 **int** 值，等于**-1** 表示流没有联系的缓存或操作失败。
- 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