

C++流与输入/输出

I/O流的概念

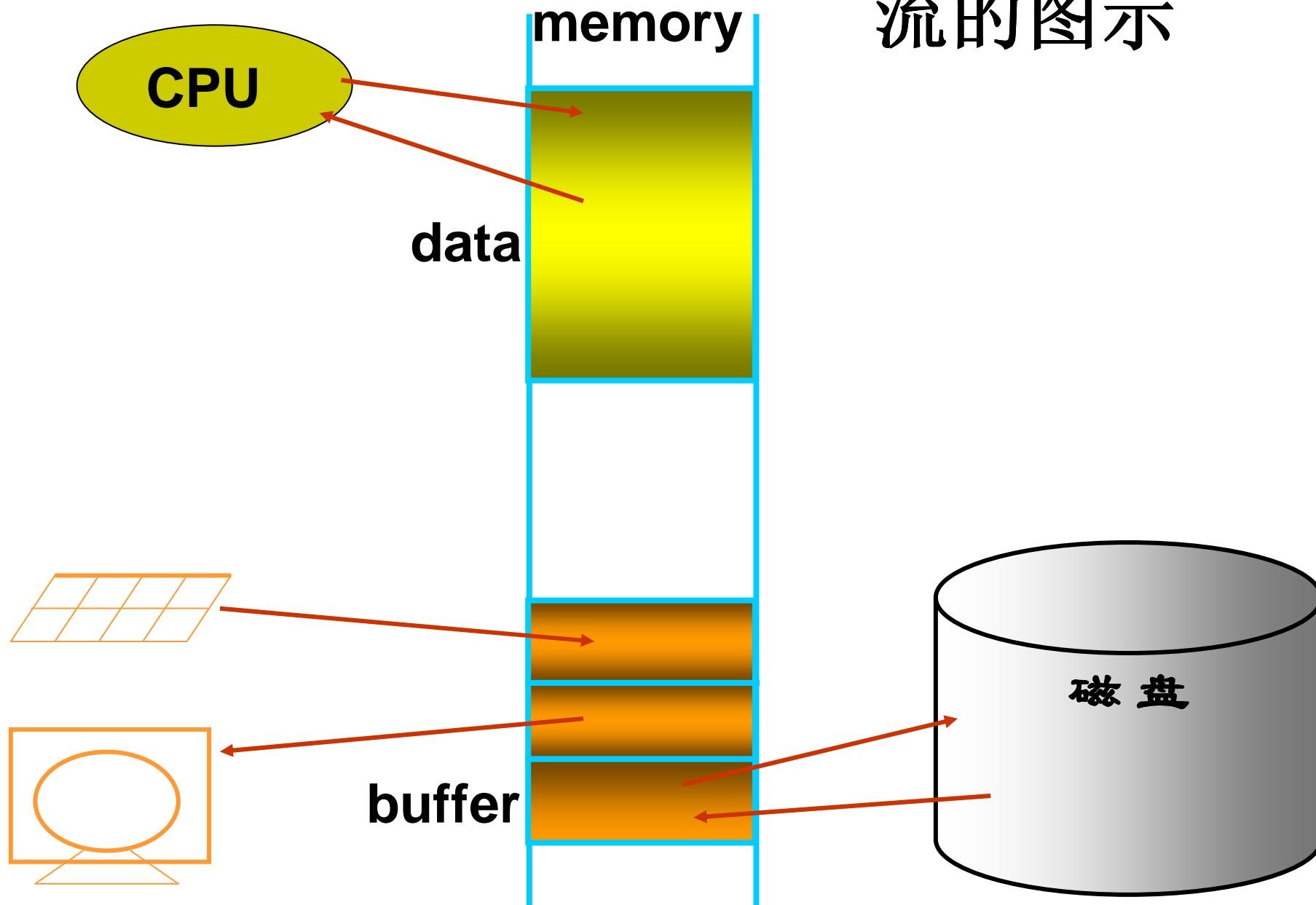
输出流

输入流

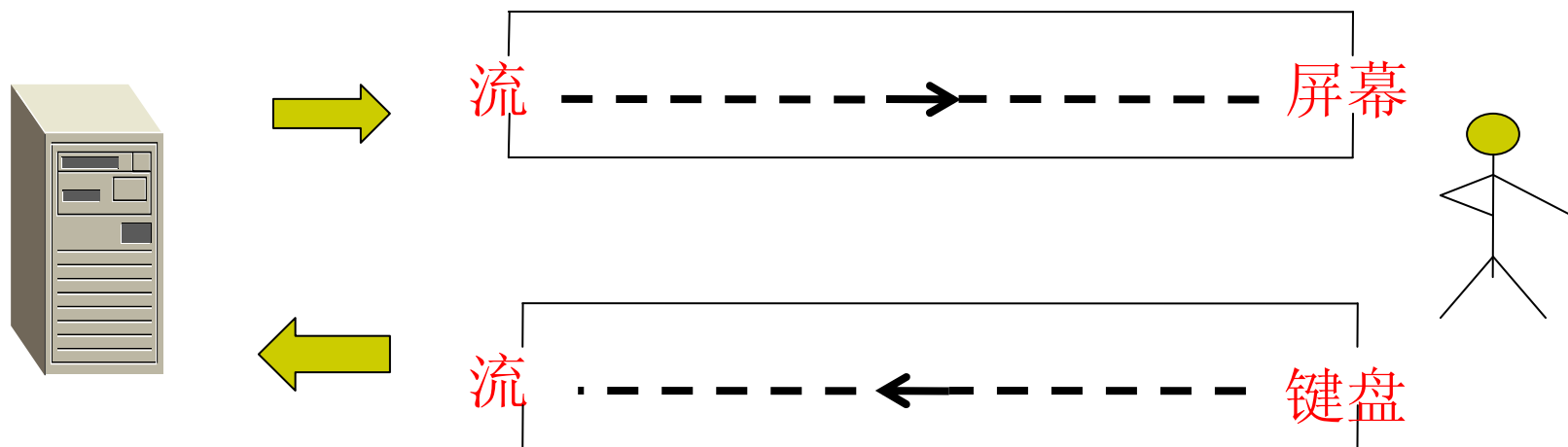
I/O流的概念

- 我要将一封信送给远在北京的朋友，只要将信写好，封上，投入邮筒即可。于是信便开始了它的旅行：进入邮局、分拣、登上火车（或飞机）、再分拣、投递到收信人手中。这段过程我是全然不知的，只是在信发出后几天，朋友就收到了。
- 如果将数据比作信，邮政系统则是“流”，程序对数据的“收发”则是对流的I/O操作。
- 其实“流”是司空见惯的，马路、铁道、航线、传送带、自动扶梯、电梯...都是流，流就是个通道。

流的图示



C++语言把设备之间的信息交换称作“流”是非常形象的。外部设备到计算机的输入信息和计算机到外部设备的输出信息就像是一条条的水流。



流的工作原理

- “流”是个字节搬运系统，它只管把内存或外设上的一个个字节的東西(无论是字符的文本，还是二进制的數據)搬來搬去，根本不理会搬的是些什么，统统视为川流不息的字节。流就是这个搬运系统的总称。至于搬运的“源”和“目的地”，流也是不加区分的。于是标准操作、文件操作、串操作都统一起来了。
- 至于如何理解这些字节，就是程序的事了。外设(比如硬盘)也不关心放在里面的究竟是些什么，反正要用还得程序取走。
- 程序依靠的是两件法宝：一是数据类型；二是流的工作方式(文本方式或二进制方式)。

两种工作方式

- **文本方式**：“流”在搬运字节给外设时，把内存中的数据按照人的习惯对照**ASCII**表，一个个的转换成字符，一个字节装一个字符，写到外设上；从外设搬运字节给内存时，再把外设上的一个个的字符还原成数据原本的模样。比如整型的**65535**，按**ASCII**码转换就变成了**54、53、53、51、53**
- **二进制方式**：“流”在搬运字节时(不管是输入还是输出)，都会原封不动的搬运，不对字节做任何加工。
还是整数**65535**，用二进制表示占两个字节：**11111111
11111111**

- 当程序与外界环境进行信息交换时，要同时动用两个对象，一个是程序中的对象，另一个是文件对象。文件是物理概念，流是逻辑概念。
- 流是文件的抽象，表示了数据的流淌。它负责在数据的生产者（源头）和数据的消费者（目的地）之间建立联系，并管理数据的流动。
- 生产者和消费者可以是变量，常量、数组、对象、引用、指针所指、还可以是键盘、显示器、文件等设备。
- 键盘是字符设备，显示器是块设备。

- **C++把C的FILE类型的结构体改造成了流对象，把对流设备的操作收编为流类的成员函数，把文件视为流对象所控制把持的另一个对象。则流与内存(缓冲区)、与文件对象就形成了类的聚集关系。IO操作就完全变成了流的自家事了。**
- 程序建立流对象，并指定这个流对象与某文件对象建立连接。程序操作流对象，流对象通过文件系统对所连接的文件对象产生作用。流对象还要把持通道资源。

- 读操作在流数据抽象中被称为（从流中）提取，写操作被称为（向流中）插入。
- 广义地，计算机将输入输出设备都视为文件。
- 若将流对象视为旅行社（提供旅游目的地和交通工具），则字节就是游客，文件对象就是出发或目的地，提取和插入就是行驶方向，缓冲区就是候车厅，流就是某某几日游这样一次旅游活动的总称。

流的分类

- 标准流

是内存与标准外设（键盘、显示器）间的数据操作。

操作方式：文本字符，顺序

头文件： **iostream**

缓冲类： **stdiobuf**

- 串流

是内存（数据区）与内存（模拟外设）间的数据操作。

操作方式：文本字符，顺序

头文件： **sstream**

缓冲类： **strstreambuf**

- 文件流

是内存与外存（磁盘）间的数据操作。

操作方式：文本字符或二进制字节数据，顺序或随机

头文件： **fstream**

缓冲类： **filebuf**

输出流

- 三个重要的输出流：
 - ostream
 - ofstream
 - ostream
- 这是三个输出流类。

用流类*iostream*替代*stdio*

- 用流类*iostream*替代*stdio*的理由：
 - stdio*的函数类型不安全；
 - stdio*的函数的不可扩充性；
- 次要的原因是：将输入输出变量与控制输入输出的格式分开书写的规矩。

输出流对象

输出流

- 系统定义的输出流对象：
 - **cout** 标准输出，有缓冲。
 - **cerr** 标准错误输出，没有缓冲，发送给它的内容立即被输出，不可重定向到它处。
 - **clog** 类似于**cerr**，但是有缓冲，缓冲区满时被输出。

输出流对象

- 输出流
- ofstream类支持磁盘文件输出
 - 如果在构造函数中指定一个文件名，当构造这个文件时该文件是自动打开的
 - **ofstream fout("filename", iosmode);**
 - 可以在调用默认构造函数之后使用open成员函数打开文件
 - **ofstream fout; //声明一个输出文件流对象**
 - **fout.open("filename", iosmode);**
//打开文件，使流对象与文件建立联系
 - **ofstream* pfout = new ofstream;**
//建立一个输出文件流对象
 - **pfout->open("filename", iosmode);**
//打开文件，使流对象与文件建立联系

插入运算符<<

- 输出流**
- 插入(<<)运算符是输出流对象的成员函数，是针对所有C++基本数据类型预先设计的，它不支持用户自定义类型的对象。
 - 用于按字节传送给输出流对象。

重载插入运算符<<

插入(<<)运算符是ostream流类的重载了的友元函数:

```
ostream & operator <<(ostream& s , 类名& r)
```

```
{
```

```
    // ....
```

```
    return s;
```

```
}
```

其中 r 是类名的引用, 是被输出的目标;

s 是ostream类的引用,其对象通常是 cout。

后面的提取运算符(>>)也类似, 不另。

使用运算符重载函数

```
#include <iostream>
```

```
class coord
```

```
{
```

```
private:
```

```
    int x,y;                //私有数据
```

```
public:                    //外部接口
```

```
    coord(int i = 0, int j = 0) : x(i), y(j) { }
```

```
    //声明为友元函数
```

```
    friend ostream& operator << (ostream & os, const coord& obj);
```

```
};
```

```
std::ostream & operator <<(std::ostream& os, const coord& obj)
{
    os << obj.x << ", " << obj.y;
    return os;
}
```

```
int main()
{
    coord a(56,78), b(99,88); //生成类的对象
    cout << a << "    " << b << '\n';
}
```

输出文件流成员函数

输出流

- 输出流成员函数有三种：
 - 与操纵符等价的成员函数。
 - 执行非格式化写操作的成员函数。
 - 其它修改流状态且不同于操纵符或插入运算符的成员函数。

输出文件流成员函数

输出流

- **open**函数
把流与一个特定的磁盘文件关联起来。
需要指定打开模式。
- **put**函数
把一个字符写到输出流中，只一个字符。
- **write**函数
把内存中的一块内容写到输出文件流中
- **seekp**和**tellp**函数
操作文件流的内部指针
- **close**函数
关闭与一个输出文件流关联的磁盘文件
- **错误处理函数**
在写一个流时进行错误处理

open函数的函数原型

程序对文件的访问，前提是建立文件与流的关联，即所谓“打开”。文件打开成功，对文件的访问就变成对流的访问。

```
void open(  
    char const *,    //可带盘符路径的文件名（串）  
    int fmode=out/in,    //打开方式  
    int access=filebuf::openprot //保护方式: 0    普通  
);
```

{	1	只读
	2	隐含
	4	系统
	8	备份

输出文件流打开方式

0x08	ios_base:: app	追加到文件末尾	出
0x04	ios_base:: ate	打开现存文件，到末尾	出
0x01	ios_base:: in	打开输入文件，不删除	入/出
0x02	ios_base:: out	打开输出文件	出
0x10	ios_base:: trunc	打开输出文件若存在则删除	出
0x20	ios_base:: nocreat		出
0x40	ios_base:: noreplace		出
0x80	ios_base:: binary		入/出

“文件操作”的步骤

1. 创建输入输出流对象；
2. 设置流对象和文件对象的联系；
3. 进行读写操作；
4. 关闭流对象。

输出到文件

输出流

```
#include <fstream>
struct Date
{ int mo, da, yr; };
int main()
{
    Date dt = { 6, 10, 92 };
    std::ofstream tfile("date.dat",ios::binary);
    tfile.write((char *) &dt, sizeof(dt));
    tfile.close();
}
```

// 对于有虚函数的类，不要用sizeof计算对象大小作为写入文件的
// 字节数，如果把虚指针写入到文件，可能引起文件内容混乱。

文件打开后测试

输出 ● 检验一个文件打开是否成功，要用`fail()`它是输出流成员函数，当它的返回值为1时表示失败。

● 用法：

```
int main()
```

出 {

```
    Date dt = {6,10,92};
```

流

```
    ofstream tfile("date.dat",ios::binary);
```

```
    if(tfile.fail())
```

```
    {   std::cerr<<"error opening file"<<"\n"; return; }
```

```
    tfile.write((char *)&dt, sizeof (dt));
```

```
    tfile.close();
```

```
}
```

二进制输出文件的由来

输出流

- 流的原本(默认) 工作方式是文本方式。
- 文本方式是将字节内容转换成文本。
- 转换是为了易读，在不需要读的场所转换是多余的。
- 有时还会弄巧成拙：
 - 将换行符(`\n`)自动扩充为回车(13)换行(10);
 - 会对所有的控制符(0~31)都进行转换;
 - 转换会消耗时间，影响效率。
- 二进制方式则是原样直接传输不转换。

二进制输出文件的方法

输

出

流

- 以通常方式构造一个流，然后使用**setmode**成员函数，在文件打开后改变模式：

```
ofstream ofs("test.dat");  
ofs.setmode(filebuf::binary);
```

- 使用**ofstream**构造函数中的模式参量指定二进制输出模式，或使用 **open()** 函数；
- 使用二进制操作符代替**setmode**成员函数：

```
ofs << binary;
```

输入流

输入流

- 重要的输入流类：
 - `istream`类最适合用于顺序文本模式输入。
 - `ifstream`类支持磁盘文件输入。
 - `istringstream`类支持串输入。

输入流对象

- 如果在构造函数中指定一个文件名，在构造流对象时该文件便自动打开。

ifstream fin("filename",iosmode);

- 亦可在调用缺省构造函数之后使用open函数来打开文件。

ifstream fin;//建立一个文件流对象

//打开文件"filename"

fin.open("filename",iosmode);

提取运算符>>

输

- 提取运算符(>>)对于所有标准C++数据类型都是预先设计好的，但对于用户自定义类型则是无法操作的，也要做成友元函数来重载。

入

- 是从一个输入流对象(输入缓冲区中)获取字节最容易的方法。

流

- 定义在ios类中的很多操纵符都可以应用于输入流。但是只有少数几个对输入流对象具有实际影响，其中最重要的是进制操纵符dec、oct和hex。

输入流成员函数

输

- **open**函数把该流与一个特定磁盘文件相关联。

- **close**函数关闭与一个输入文件流关联的磁盘文件。

入

- **getline**的功能是从输入流中读取多个字符，并且允许指定输入终止字符，读取完成后，从读取的内容中删除掉终止字符。

流

- **read**成员函数从一个文件读字节到一个指定的内存区域，由长度参数确定要读的字节数。
如果给出长度参数，当遇到文件结束或者在文本模式文件中遇到文件结束标记字符时结束读取。

输入流成员函数

get函数有多种重载的形式:

输

- **get()** 读取输入流的一个字符, 返回它。功能与提取运算符 (>>) 很相像, 主要的不同点是get函数在读入数据时包括空白字符。

入

- **get(char&ch)** 读取输入流中的一个字符(包括空白字符), 将它放到变量ch中, 返回流对象。

流

- **get(char *buf, streamsize num)** 读取输入流的num-1个字符, 或遇到换行符或文件尾时停止, 将串放到buf数组中, 并用null结尾。返回流对象。对于所遇到换行符不读取, 仍留在流中给下次操作作用。
- **get(char *buf, streamsize num, char delim)** 读取输入流的num-1个字符, 或遇到delim所指定的分隔符或文件尾时停止, 将串放到buf数组中, 并用null结尾。返回流对象。对于所遇到分隔符不读取, 仍留在流中给下次操作作用。

输入流成员函数

输

入

流

- **seekg**函数用来设置输入文件流中读取数据位置的指针。
- **tellg**函数返回当前文件读指针的位置。
- **peek**函数用来读出（而非取出，仅看一下数据）输入流缓冲区中的数据。
- **ignore**函数用来扔掉某个/些字符。丢弃的字符或由参数1指定，或直至遇到指定的分隔符为止。
ignore(欲扔的字符数=1，分隔符=EOF);
- **gcount**函数返回上次读取的字符个数。
- **putback(ch)**函数将ch的内容放回到流中，并返回流对象。

```
#include <fstream>
#include <iostream>
```

```
int main()
```

```
{
```

```
    std::ifstream fin("TEST.TXT");
```

```
    if(!fin)
```

```
    {
```

```
        std::cerr<<"Cannot open file."<<"\n";
```

```
        return 1;
```

```
    }
```

```
    in.ignore( 8, ' '); //忽略到读完8个字符或发现空格时为止
```

```
    char ch;
```

```
    while(fin) //然后显示文件的剩余的内容
```

```
    {
```

```
        fin.get(ch);
```

```
        if (fin)
```

```
            std::cout<<ch;
```

```
    }
```

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

```
}
```

rdbuf的用法

- 为了衔接输入和输出以构成新的流通道，可以由用户指定（更改）流缓冲（**streambuf**）。
- 每个流对象都有自己的一个成员函数**rdbuf()**，该函数返回指向缓冲区的指针，通过该指针就指定了不同于默认的新的缓冲区。
- 函数**rdbuf()**有两个版本，一个无参一个有一个形参。
无参的版本用来取出（返回）流缓冲；有参的版本用来按参数给的流缓冲设定新缓冲——改回原来的缓冲。
- 它们都需要包含**无.h**的头文件。

```
#include<iostream>
#include<fstream> //使用rdbuf函数时，定要使用不带.h的头文件
int main()
{
    // 在当前目录下建立文件rebuf.txt关联到输出文件流
    std::ofstream fout("rebuf.txt");
    // 将该文件输出流缓冲的指针作为实参,交给屏幕输出流重置函数,
    // 由x指针保存该流缓冲. 于是将该文件与标准输出建立了沟通
    streambuf *x = cout.rdbuf(fout.rdbuf());
    std::cout << "text." << '\n'; // 其实将字符串送到了文件
    std::cout << "123456789" << '\n'; // 其实写给了文件
    std::cout.rdbuf(x); // 重置了输出流缓冲,改到用原来的缓冲。
    std::cout << "text2." << '\n'; // 送到显示器去
    std::ifstream fin("rebuf.txt");
    // 输入流与输出流共用同一个缓冲区,即显示读出的文件内容
    std::cout << fin.rdbuf ();
    std::cout << "asdf" << '\n'; // 新字符串会覆盖了原来的输入内容
    std::cout << fin.rdbuf (); // 再显示输入流缓冲,已经没有东西了
}
```

文件的输入输出

输入流

- 可以用**fstream**来创建同时用作文件输入输出的流。
- 此时的文件读写指针用的是同一个。因此，操作时一定要注意调整的合适位置。

```
#include <fstream>
#include <iostream>
int main()
{
    using std::cout;
    char str[20],str1[20],str2[20],str3[20];
    std::fstream finout("TEST.TXT", ios_base::in | ios_base::out);
    finout<<"this is a good day."<<"\n"; //将串写给了文件
    cout<<finout.tellp()<<"\n"; //文件指示器走到了20
    cout<<finout.tellg()<<"\n"; // 20
    finout.seekg(0); //置回到文件头
    cout<<finout.tellp()<<"\n"; // 0
    cout<<finout.tellg()<<"\n"; // 0
    finout>>str>>str1>>str2>>str3;//将文件中的串读入各变量
    cout<<str<<" "<<str1<<" "<<str2<<" "<<str3<<"\n";
    finout.close();
}
```

串 流

- 是内存（数据区）与内存（模拟外设）间的数据操作。
即串流可以将**string**对象或字符串连接到模拟外设上，实现输入 / 输出操作。
- 提取时(输入)对字符串按变量类型解释，插入时(输出)将数据转换成字符串。
- 作用：充当**string**对象或字符串与真正外设间数据交换的媒介。如写盘的数据准备、输入数据验证前的暂存等。
- 操作方式：文本字符，顺序进行。

- 缓冲类： `stringstreambuf`。
- 对C方式(旧版)的字符串，要使用 `stringstream`、`istringstream`、`ostringstream` 类，它们在头文件 `sstream` 中。
- 对C++方式(新版)的 `string` 对象，要使用 `stringstream`、`istringstream`、`ostringstream` 类，它们在头文件 `sstream` 中。
- 两种版本的区别：旧版字符串流的构造函数，要求使用者指明内存的确切大小，以便写入字符；新版字符串流会自动分配字符所需的内存。

一个使用串流的例子

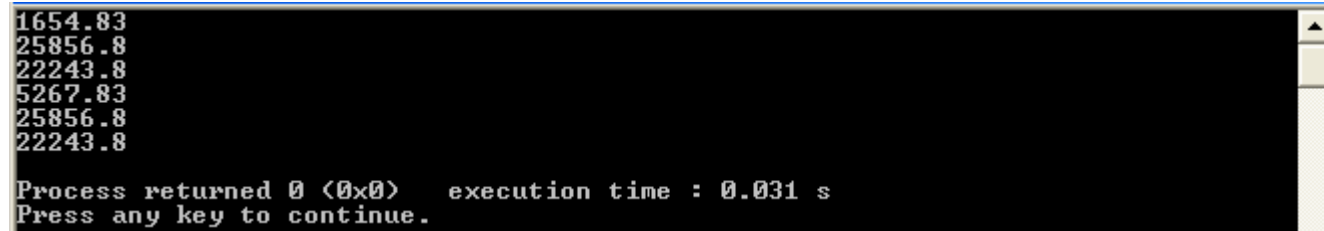
```
#include<iostream>
#include<sstream>
#include<fstream>

int main()
{
    std::ifstream fin( "aaa.txt" );//创建一个文件输入流对象fin
    //将in中的每行字符读入串对象s中
    for(std::string s; std::getline(fin, s); )
    {
        double a, sum = 0.0; // 准备好累加
        // 创建一个以串s为依托的istringstream类的流对象sin
        //然后从串s中读出串值，变成浮点数送给a,再累加
        for(std::istringstream sin(s); sin >> a; sum += a);
        std::cout << sum << '\n' ; //显示本行的累加和
    }
}
```

// 文件aaa.txt的内容:

**3.8 4 89.5 41.288 165.01 358.155 165.65 0.01 3.8 4 89.5 41.288 165.01 358.155 165.65 0.01
16556 4154 0.11 556.02 69.66 55.435 25.1122 1545 112.9 24 15 15.7 2551.0 89.1546 87.6654
3.8 4 89.5 41.288 165.01 358.155 165.65 0.01 16556 4154 0.11 556.02 69.66 55.435 25.1122
1545 112.9 24 15 15.7 2551.0 89.1546 87.6654 3.8 4 89.5 41.288 165.01 358.155 165.65 0.01
16556 4154 0.11 556.02 69.66 55.435 25.1122 1545 112.9 24 15 15.7 2551.0 89.1546 87.6654
3.8 4 89.5 41.288 165.01 358.155 165.65 0.01 16556 4154 0.11 556.02 69.66 55.435 25.1122**

// 运行结果:



```
1654.83
25856.8
22243.8
5267.83
25856.8
22243.8

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

文件读写演示

网络上数据非常丰富，有时我们需要从这些海量数据中筛选出我们感兴趣的一些信息。我们在此把这个复杂的问题简单化，假设我们有**10G**个正整数存放在一个磁盘文件里面，我们需要从这些数中找出**1**百万个(假设仅仅**512MB**内存，但是硬盘容量充足)最大的。

为了模拟以上过程，并节约时间，我们处理数据量小一点。首先用随机数产生器产生**1**千万个正整数，把这些随机数写到硬盘上，放在一个文件里，要求从这些数中找出**1K**个最大的，在**32**位系统上使用额外**4MB**缓存运行速度就基本可以接受。

一种方法：

每次读取1M个数放入内存，建造一个堆，筛选出1K个最大的，这样需要读取10次，然后建堆筛选10次，共产生10K个较大的数，再从这10K个中选出1K个最大的。为了加快存取速度，设置读写缓冲区4MB(sizeof(size_t) * 1M)。

另一种方法：

采用统计的办法，读一遍文件，统计出10, 9, ..., 1位的随机数各有多少个，10位的数中1, 2, ..., 9开头的分别多少个，9位的数中1, 2, ..., 9开头的各有多少个，..., 1位的数中, 0, 1, ..., 9各有多少个，计算出最大1k个数的位数和起始数字范围。再读一遍文件，根据以上范围按照位数和各位数字大小从大到小筛选出1024个写入文件即为所求结果。

第一种方法模拟结果

```
input a file name to create for holding random numbers: a.txt
generating 10000000 numbers and writing them into a.txt used: 4469ms
finding 1024 maximum numbers in a.txt used: 3094ms
input a file name to create for holding maximum numbers: b.txt
testing is in progress... PASSED
Press any key to continue . . .
```

```
input a file name to create for holding random numbers: aa.txt
generating 10000000 numbers and writing them into aa.txt used: 4422ms
finding 1024 maximum numbers in aa.txt used: 3094ms
input a file name to create for holding maximum numbers: bb.txt
testing is in progress... PASSED
Press any key to continue . . . _
```

Thanks!