@阿西拜-南昌

IO库

- C++ Primer第五版
- 第8章

IO类

文件输入输出

string流

小结

```
#include<iostream>
using namespace std;

int main(){
 int a;
  cout<<"before a bad input operation:"
   <<"\ncin.eof():"<<cin.eof()
  <<"\ncin.fail():"<<cin.fail()
   <<"\ncin.bad():"<<cin.bad()
   <<"\ncin.good():"<<cin.good()<<endl;
  cin>>a;

  cout<<"after a bad input operation:"
   <<"\ncin.eof():"<<cin.eof()
  <<"\ncin.fail():"<<cin.fail()
   <<"\ncin.bad():"<<cin.bad()
   <<"\ncin.good():"<<cin.good()<<endl;
 cin.clear();

 cout<<"\ncin.eof():"<<cin.eof()
  <<"\ncin.fail():"<<cin.fail()
   <<"\ncin.bad():"<<cin.bad()
   <<"\ncin.good():"<<cin.good()<<endl;

  system("pause");
  return 0;
}
```

分别读入 a or 1 就可以理解了

IO库类型和头文件	
头文件	类型
iostream	istream, wistream从流中读取 ostream,wostream写到流中去 iostream,wiostream对流进行读写；
fstream	ifstream,wifstream从文件中读取； ofstream,wofstream写到文件中去； fstream,wfstream读写文件；
sstream	istringstream,wistringstream从string对象中读取； ostringstream,wostringstream写到string对象中去； stringstream,wstringstream对string对象进行读写；

w表示wchar_t

```
ofstream out1, out2;
out1 = out2; //错误：不能对流对象赋值
ofstream print(ofstream); //错误：不能将形参或是返回类型设为流类型
out2 = print(out2); //错误：out2不是可修改的左值
```

IO对象不可以拷贝或赋值

读写一个IO对象会改变其状态，因此传递和返回的引用不能是const的

IO类所定义的一些函数和标志，可以帮助我们访问和操纵流的条件状态

IO库条件状态	
strm::iostate	strm是一种IO类型。iostate是一种机器相关的类型，提供了表达条件状态的完整功能
strm::badbit	用来指出流已崩溃
strm::failbit	用来指出一个IO操作失败了
strm::eofbit	用来指出流到达了文件结束
strm::goodbit	用来指出流未处于错误状态。此值保证为零
s.eof()	若流s的eofbit置位，则返回true
s.fail()	若流s的failbit或badbit置位，则返回true
s.bad()	若流s的badbit值位，则返回true
s.good()	若流s处于有效状态，则返回true
s.clear()	将流s中所有条件状态位复位，将流的状态设置为有效。返回void
s.clear(flags)	根据给定的flags标志位，将流s中对应条件状态位复位。flags的类型为strm::iostate。返回void
s.setstate(flags)	根据给定的flags标志位，将流s中对应条件状态位置位。flags的类型为strm::iostate。返回void
s.rdstate()	返回流s的当前条件状态，返回值类型为strm::iostate

```
goodbit = 0x0;   000
eofbit = 0x1;    001
failbit = 0x2;   010
badbit = 0x4;    100
```

看第一页的代码

^Z 为false 且前面不能有其他字符

000

全部变为0

0 -> 1

有时候我们需要知道流为什么失败
iostate类型提供了表达流状态的完整功能，这个类型应作为一个位集合来使用

```
auto old_state = cin.rdstate(); //记住cin的当前状态
cin.clear(); //使cin有效
process_input(cin); //使用cin
cin.setstate(old_state); //将cin置为原有状态

//复位failbit和badbit，保持其他标志位不变
cin.clear(cin.rdstate() & ~cin.failbit & ~cin.badbit);
```

100 010

如果rdstate是 101

~failbit 011
~bad 101
计算后 cin是 001
就是保留了最后一位
剩下的改为0

每个输出流都管理一个缓冲区
有了缓冲机制，操作系统就可以在需要时将程序的多个输出操作合并，提升性能

```
//刷新输出缓冲区
cout<< "hi!" <<endl; //输出hi和一个换行，然后刷新缓冲区
cout<< "hi!" <<flush; //输出hi，然后刷新缓冲区，不附加任何额外字符
cout<< "hi!" <<ends; //输出hi和一个空字符，然后刷新缓冲区

//通过设置unitbuf操纵符来控制是否立即刷新
cout<<unitbuf; //所有输出操作后都会立即刷新缓冲区
cout<<nounitbuf; //回到正常的缓冲方式
```

空格

取消缓冲区

如果程序崩溃，输出缓冲区不会被刷新

每个流同时最多关联到一个流，但多个流可以同时关联到一个ostream

```
cin.tie(&cout); //仅仅是用来展示：标准库内置
//old_tie指向当前关联到cin的流（如果有的话)
ostream *old_tie = cin.tie(nullptr); //cin不再与其他流关联
//将cin与cerr关联；这不是一个好主意，因为cin应该关联到cout
cin.tie(&cerr); //读取cin会刷新cerr而不是cout
cin.tie(old_tie); //重建cin和cout间的正常关联
```

好

cin与cout的绑定，并不是输出输入的内容
就是cin和cout没有绑定的话

```
int ival;
cout << "请输入一个整数：";
cin >> ival;
```

可能就不会输出这段文字

文件操作 i写 o读 f读写
步骤 头文件 》创建流对象 》打开文件x。open（“路径”，打开方式）》写数据》关闭文件

二进制要 writer 写入文件 read 读

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

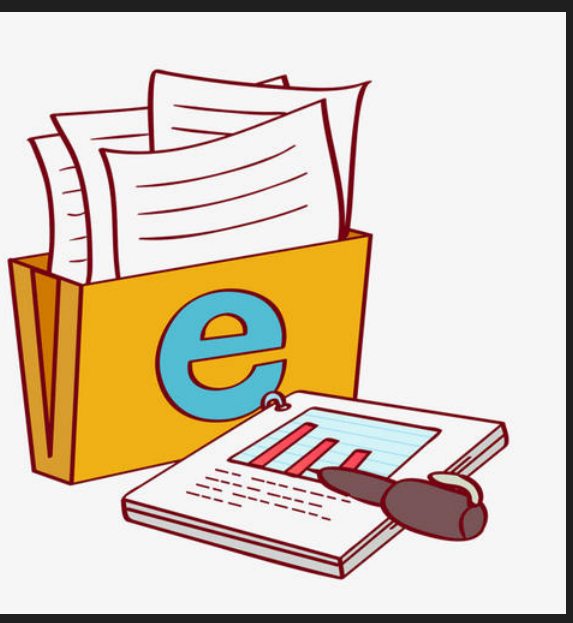
istream& read（char *buf ,int len）；类的len 用siziof 类名

1.cin.clear()
是用来更改cin的状态标示符的。 先clear 再清除
cin.sync()是用来清除缓存区的数据流的。
如果标示符没有改变那么即使清除了数据流也无法输入。所以两个要联合起来使用。
cin.ignore(number,'/n');删除 /n前100个字符

文件输入输出

fstream特有的(除了继承iostream之外)操作	
fstream fstrm;	创建一个未绑定的文件流。***fstream***是头文件fstream中定义的一个类
fstream fstrm(s);	创建一个fstream，并打开名为s的文件。这些构造函数都是explicit的。默认的文件模式mode依赖于***fstream***的类型
fstream fstrm(s,mode);	与前一个构造函数类似，但按指定mode打开文件
fstrm.open(s);	打开名为s的文件，并将文件与fstrm绑定。默认的文件mode依赖于***fstream***的类型，返回为void
fstrm.close(s);	关闭fstrm绑定的文件。返回void
fstrm.is_open(s);	返回一个bool值，指出与fstrm关联的文件是否 成功打开且尚未关闭

```
ifstream in(ifile); //构造一个ifstream并打开指定文件
ofstream out; //输出文件流，未关联到任何文件
out.open(ifile+".copy"); //打开指定文件
if(out) //检查open是否成功
        ...;
in.close(); //关闭文件
in.open(ifile+"2"); //打开另一个文件
```



在要求使用基类对象的地方，
可以用继承类对象来替代

```
ifstream input(argv[1]); //打开销售记录文件
ofstream output(argv[2]); //打开输出文件
Sales_data total; //保存销售总额的变量
if(read(input,total)){
        Sales_data trans; //保存下一条销售记录的变量
        while(read(input, trans)){//读取剩余记录
                if(total.isbn() == trans.isbn()) //检查isbn
                        total.combine(trans);
                else{
                        print(output, total)<<endl; //打印结果
                        totla = trans; //处理下一本书
                }
        }
        print(output,total)<<endl; //打印最后一本书的销售额
}else  //文件中无输入数据
        cerr <<"No data?!"<<endl;
```

自动构造和析构

```
//对每个传递给程序的文件执行循环操作
for(auto p = argv +1; p != argv +argc; ++p){
        ifstream input(*p); //创建输出流并打开文件
        if(input){
                process(input);
        }else
                cerr<<"couldn't open:" + string(*p);
}//每个循环步input都会离开作用域，因此会被消耗
```

当一个fstream对象被消耗时，close会自动被调用

文件输入流

每个流都有一个关联的文件模式（file mode）

文件模式	
in	以读方式打开
out	以写方式打开
app(append)	追加：每次写操作前均定位到文件末尾
ate(at end)	每次打开文件后立即定位到末尾
trunc	截断：如果打开的文件存在，其内容将被丢弃
binary	以二进制方式进行IO

以out模式打开文件会丢弃已有数据

保留ofstream打开的文件中已有数据的唯一方法是显示指定app或in模式

```
//在这几条语句中，file1都被截断
ofstream out("file1"); //隐含以输出模式打开文件并截断文件
ofstream out2("file1", ofstream::out); //隐含地截断文件
ofstream out3("file1", ofstream::out | ofstream::trunc);
//为了保留文件内容，我们必须显示指定app模式
ofstream app("file2", ofstream::app); //隐含为输出模式
ofstream app2("file2", ofstream::out | ofstream::app);

ofstream out; //未指定文件打开模式
out.open("scratchpad"); //模式隐含设置为输出和截断
out.close();
out.open("preious", ofstream::app);//模式为输出和追加
out.close();
```

效果一样

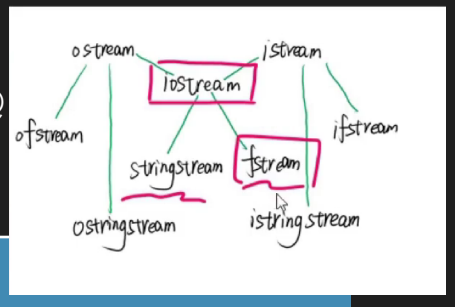
可以用|使用俩种文件方式

out 写和不写都是一样的

string流：

从string读写数据，就像string是一个IO流一样

@



fstream和stringstream都是继承于istream的但是fstream和string无关

stringstream特有的操作	
sstream strm;	strm是一个未绑定的stringstream对象。***sstream***是头文件sstream中定义的一个类型
sstream strm(s);	strm是一个***sstream***对象，保存string s的一个拷贝。此构造函数是explicit的
strm.str()	返回strm所保存的string的拷贝
strm.str(s)	将string s拷贝到strm中。返回void

```
struct PersonInfo{
    string name;
    vector<string> phones;
};
```

```
张三    18170991111
李四    18611118668    0108431123
王二    13022228888    13877776666    07918888888
```

当需要处理行内单词的时候，通常可以使用istringstream

```
string line, word; //分别保存来自输入的一行和单词
vector<PersonInfo> people; //保存来自输入的所有记录
while(getline(cin,line)){
    PersonInfo info;                    //创建一个保存此记录数据的对象
    istringstream record(line);         //绑定刚读入的行
    record >> info.name;                //读取名字
    while( record >> word)              //读物电话号码
        info.phones.push_back(word); //保存他们
    people.push_back(info);             //将此记录追加到people末尾
}
```

string代表的是内存，istringstrem是从内存读取数据，ostringstream是将数据写入内存

```
for(const auto &entry:people){//对people中每一项
    ostringstream formatted,badNums; //每个循环步创建的对象
    for(const auto &nums:entry.phones){//对每个数
        if(!valid(nums)){
            badNums<<" "<<nums;//将数的字符串形式存入badNums
        }else
            formatted<<" "<<format(nums);//将格式化的字符串“入”formatted
    }
    if(badNums.str().empty()) //没有错误的数
        os<<entry.name<<" "<<formatted.str()<<endl;//打印名字和格式化的数
    else//否则，打印名字和错误的数
        cerr<<"input error:"<<entry.name<<"invalid numbers(s) “
            <<badNums.str()<<endl;
}
```