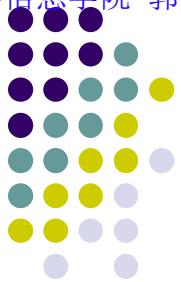
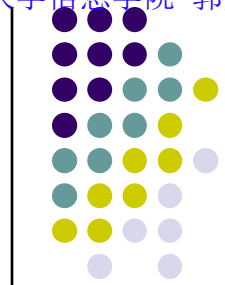


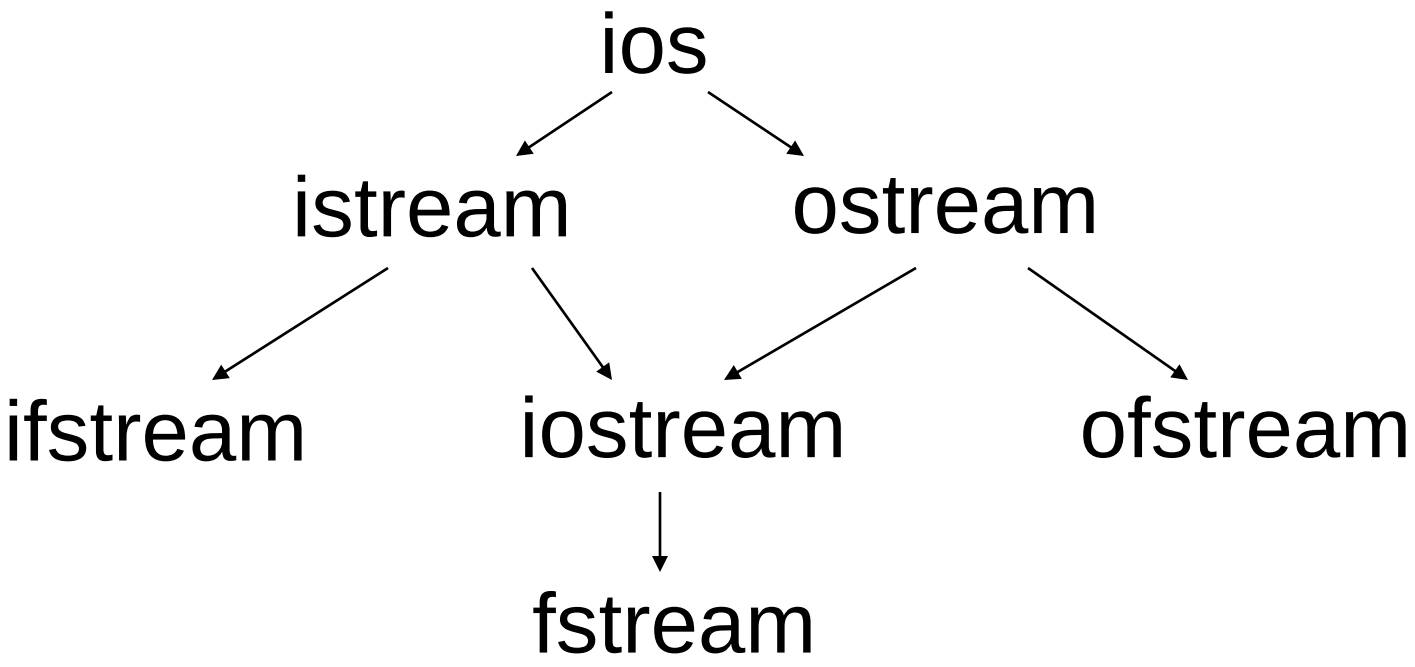
新标准C++程序设计

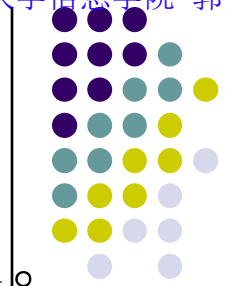
北京大学信息学院　郭 炜

GWPL@PKU.EDU.CN



输入输出流和文件操作


ios
istream
ostream
ifstream
iostream
ofstream
fstream



istream是用于输入的流类，cin就是该类的对象。

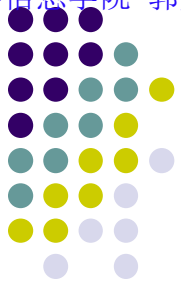
ostream是用于输出的流类，cout就是该类的对象。

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;
```

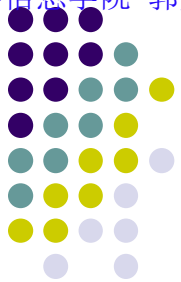
一样

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，不能被重定向。
- clog对应于标准错误输出流，用于向屏幕输出出错信息，不能被重定向。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。

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则输出错误信息
                cerr << "error." << endl;
    else
            cout << x /y ;
       return 0;
}
```



- 流插入运算符 <<

```
cout << “Good morning!\n”;   不刷新缓冲区
cout << “Good”;              不刷新缓冲区
cout << “morning!”;          不刷新缓冲区
cout << endl;                刷新缓冲区
cout << flush;               刷新缓冲区
```

可以用如下方法判输入流结束：

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

如果从键盘输入，则在单独一行输入Ctrl+Z代表输入流结束

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

输入流

- 读取运算的返回值

重载>>运算符的定义：

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

可以用如下方法判输入结束：

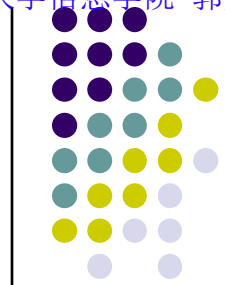
```
int x;
while(cin>>x){

} // 明显类型不匹配，为什么可以？
```

```
#include <iostream>
#include <stdio.h>
using namespace std;
class MyCin
{
    bool bStop;
    public:
        MyCin():bStop(false) { }
        operator bool( ) { //重载类型强制转换运算符 bool
            return !bStop;
        }
        MyCin & operator >> (int n)
        {
            cin >> n;
            if( n == 100)
                bStop = true;
            return * this;
        }
};
```

```
int main()
{
    MyCin m;
    int n;
    while( m >> n)
        cout << "number:" << n << endl;
    return 0;
}
```

输入100,则程序结束



输入流

- Istream类的成员函数

int get() 读入一个字符(包括空白字符), 返回该字符，读到末尾则返回 EOF

```
#include <iostream>
using namespace std;
int main()
{
    int c;
    while( ( c = cin.get()) != EOF)
        cout.put(c);
    return 0;
}
```

输入流

istream类的成员函数

```
istream & get(char *buffer, int size, char
  delim= '\n' );
```

读size –1 个字符入buffer,或者遇到delim;

在buffer最后加 '\0',分隔符留在输入流.

cin.getline(char *buffer, int size, char delim='\n')

或者读size –1 个字符入buffer,或者遇到 '\n';

在buffer最后加 '\0', 分隔符从流去掉.

cin.eof() 返回输入是否结束标志.

输入流

istream类的成员函数

```
istream & getline(char * buf, int bufSize);
istream & getline(char * buf, int bufSize,char delim);
```

第一个版本从输入流中读取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）。函数会自动在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

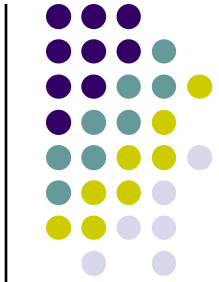
- 整数流的基数：流操纵算子dec, oct, hex

```
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



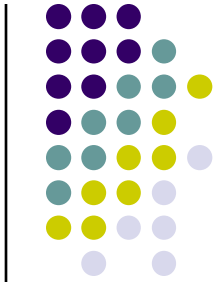
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        double x = 1234567.89,y = 12.34567;
        int n = 1234567;
        int m = 12;
        cout << setprecision(6) << x << endl
                << y << endl << n << endl << m;
}
```

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



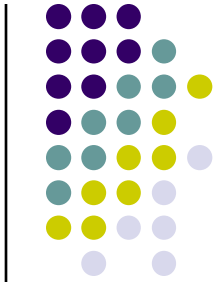
```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        double x = 1234567.89,y = 12.34567;
        int n = 1234567;
        int m = 12;
        cout << setiosflags(ios::fixed) <<
                setprecision(6) << x << endl
                << y << endl << n << endl << m;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

以小数点位置固定的方式输出

输出：

```
1234567.890000
12.345670
1234567
12
```

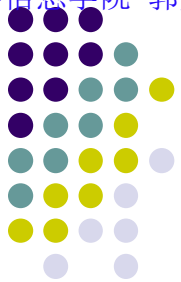
控制浮点数精度的流操纵算子



```
#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 ;
}
```

输出：
1234567.890000
1.23457e+006

取消以小数点位置固定的方式输出



- **设置域宽**(setw,width)

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

输入:1234567890

输出:1234
5678
90

- **设置域宽**(setw,width)
- 需要注意的是在每次读入和输出之前都要设置宽度。例如：

```
char str[10];
cin.width(5);
cin >> string;
cout << string << endl;
cin >> string;
cout << string << endl;
```

输入：1234567890

输出：1234

567890

- **设置域宽**(setw,width)
- 需要注意的是在每次读入和输出之前都要设置宽度。例如：

```
char str[10];
cin.width(5);
cin >> string;
cout << string << endl;
cin.width(5);
cin >> string;
cout << string << endl;
```

输入：1234567890
输出：1234
5678

```
#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 ;
        //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;
}
```

```
1) 8d  141  215
2) 1.2346e+006  12.346
3) 1234567.89000  12.34567
4) 1.23457e+006  1.23457e+001
5) ***+12.10000
6) 12.10000****
7) ****12.10000
8) -***12.10000
```

流操纵算子

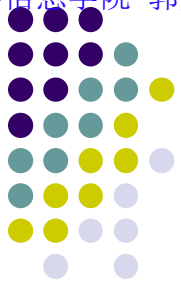
setw, setprecision 是函数，在iomanip中定义

```
struct _Setw { int _M_n; };

inline _Setw   setw(int __n)  {
      _Setw __x;
      __x._M_n = __n;
      return __x;
}
```

自己写一个 mysetw

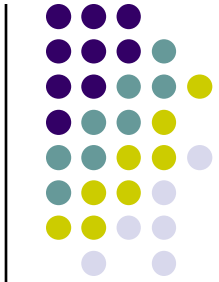
```
#include <iostream>
#include <iomanip>
using namespace std;
_Setw mysetw(int n) {
    _Setw __x;
    __x._M_n = n;
    return __x;
}
int main()
{
    cout << mysetw(10) << 100 << endl;
} //输出: num:       100
```



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

输出： aa　　bb

为什么可以？



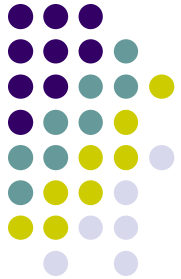
用户自定义流操纵算子

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

流操纵算子



hex, oct 都是函数，在ios_base.h中定义

```
inline ios_base&  hex(ios_base& __base)
{
   __base.setf(ios_base::hex,
   ios_base::basefield);
   return __base;
}
```

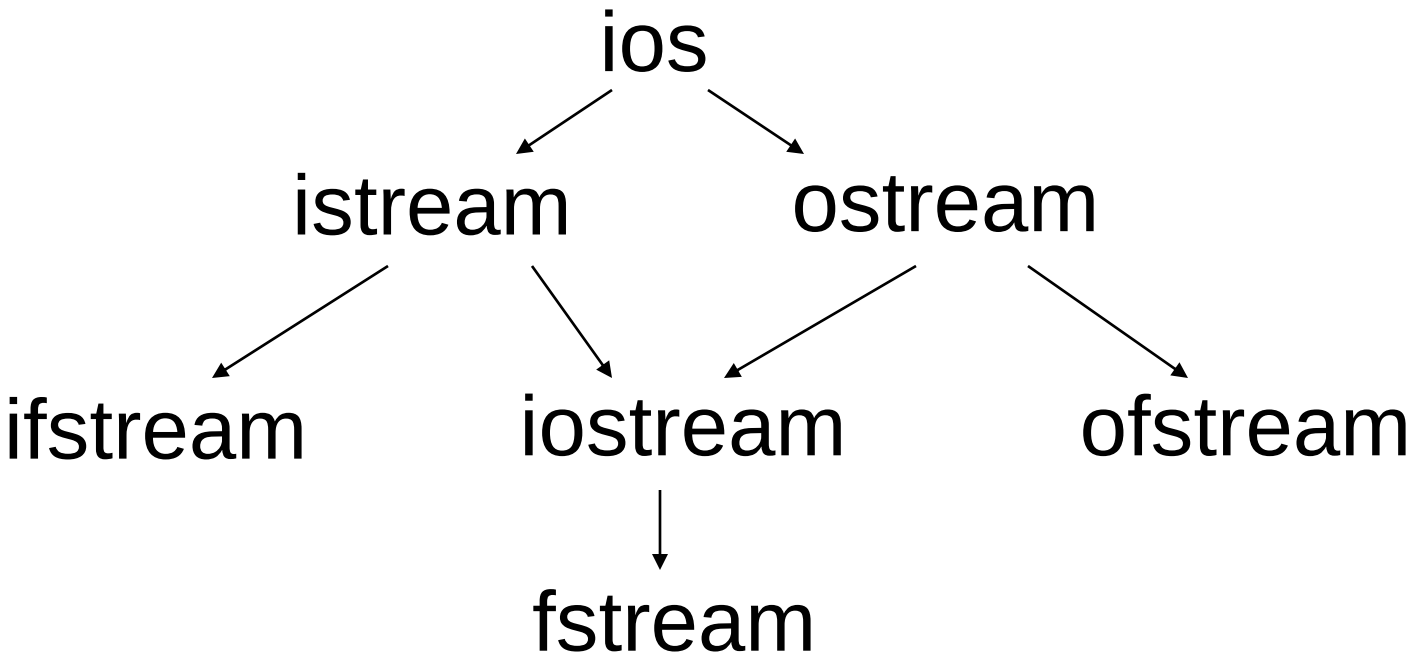
- 位　　bit
- 字节　byte
- 域/记录

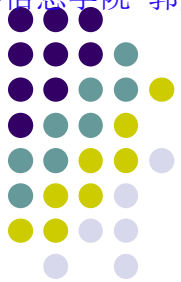
  例如：学生记录

  ```
  int ID;
  char name[10];
  int age;
  int rank[10];
  ```

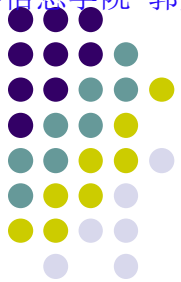
- 我们将所有记录顺序地写入一个文件，称为顺序文件。

- 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：





- #include <fstream.h> // 包含头文件
- ofstream outFile(“clients.dat”, ios::out|ios::binary); //打开文件
  - **ofstream 是 fstream中定义的类**
  - **outFile 是我们定义的ofstream类的对象**
  - **“clients.dat”是将要建立的文件的文件名**
  - **ios::out 是打开并建立文件的选项**
    - **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



- 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）{cerr << “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

```
ofstream  fout("a1.out",ios::app);
long location = fout.tellp();
        //取得写指针的位置
location = 10L;
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

```
ifstream  fin(“a1.in”,ios::ate);
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

字符文件读写

- 因为文件流也是流，所以前面讲过的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：

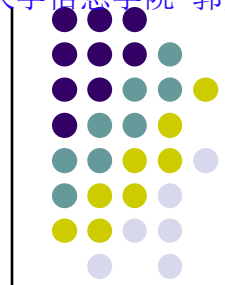
1 234 9 45 6 879

则执行本程序后，生成的out.txt的内容为：

1 6 9 45 234 879

参考程序

```
#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; }
```

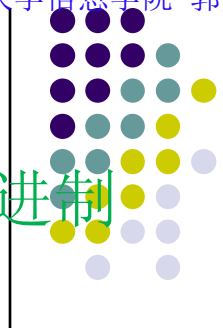


```
int x=10;
fout.seekp(20, ios::beg);
fout.write( (const char *)(&x), sizeof(int) );

fin.seekg(0, ios::beg);
fin.read( (char *)(&x), sizeof(int) );
```

- 二进制文件读写，直接写二进制数据，记事本看未必正确。

二进制文件读写



```
//下面的程序从键盘输入几个学生的姓名的成绩，并以二进制
//文件形式存起来
#include <iostream>
#include <fstream>
using namespace std;
class CStudent
{
    public:
        char szName[20];
        int nScore;
};
```

```
int main()
{
    CStudent s;
    ofstream OutFile( "c:\\tmp\\students.dat",
                                ios::out|ios::binary);
    while( cin >> s.szName >> s.nScore ) {
          if( stricmp(s.szName, "exit ") == 0) //名字为exit则结束
                  break;
          OutFile.write( (char * ) & s, sizeof( s) );
    }
    OutFile.close();
    return 0;
}
```

输入：

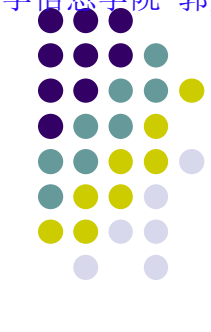
Tom 60
Jack 80
Jane 40
exit 0
则形成的 students.dat 为 72字节，用 记事本打开，呈现：

Tom 烫烫烫烫烫烫烫烫< Jack 烫烫烫烫烫烫烫藁 Jane 烫烫烫烫烫烫烫烫?

二进制文件读写

//下面的程序将 students.dat 文件的内容读出并显示

```
#include <iostream>
#include <fstream>
using namespace std;
class CStudent
{
    public:
        char szName[20];
        int nScore;
};
```



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

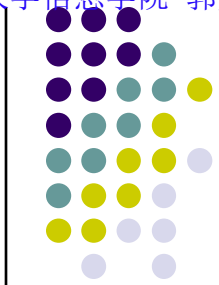
输出：

Tom 60

Jack 80

Jane 40

二进制文件读写



//下面的程序将 students.dat 文件的Jane的名字改成Mike

```
#include <iostream>
#include <fstream>
using namespace std;
class CStudent
{
    public:
        char szName[20];
        int nScore;
};
```

```
int main()
{
    CStudent s;
    fstream iofile( "c:\\tmp\\students.dat", ios::in|ios::out);
    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"));
    iofile.seekg(0,ios::beg); //定位读指针到开头
    while( iofile.read( (char* ) & s, sizeof(s)) )
        cout << s.szName << " " << s.nScore << endl;
    iofile.close();
    return 0;
}
```

输出：

Tom 60

Jack 80

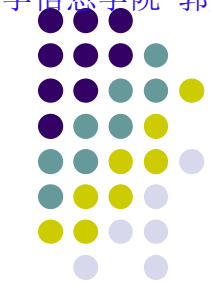
Mike 40

例子：mycopy 程序，文件拷贝

```
/*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;
}
```

在windows操作系统下输出某个路径下的文件



```
#include <windows.h>
using namespace std;
int main()  {
        HANDLE  f1;                                  // 句炳
        WIN32_FIND_DATA  fData;  //存储文件信息
        f1 = FindFirstFile("c://tmp//*.*",&fData);
        do {
                if((fData.dwFileAttributes &
                        FILE_ATTRIBUTE_DIRECTORY)  ==
                      FILE_ATTRIBUTE_DIRECTORY)
                      cout << "directiory: " <<
                              fData.cFileName << endl;
                else
                      cout << "file: " << fData.cFileName <<
                              endl;
        }while(FindNextFile(f1,&fData));
        FindClose( f1);
}
```

windows 操作系统和文件夹操作有关的两个函数：

int _chdir(const char *) 改变当前文件夹

int _mkdir(const char *) 创建文件夹