

2024年春

程序设计实习：C++程序设计

第八讲 输入输出和文件操作

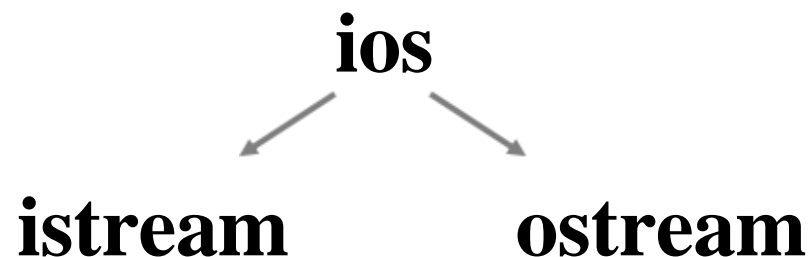
贾川民
北京大学



主要内容

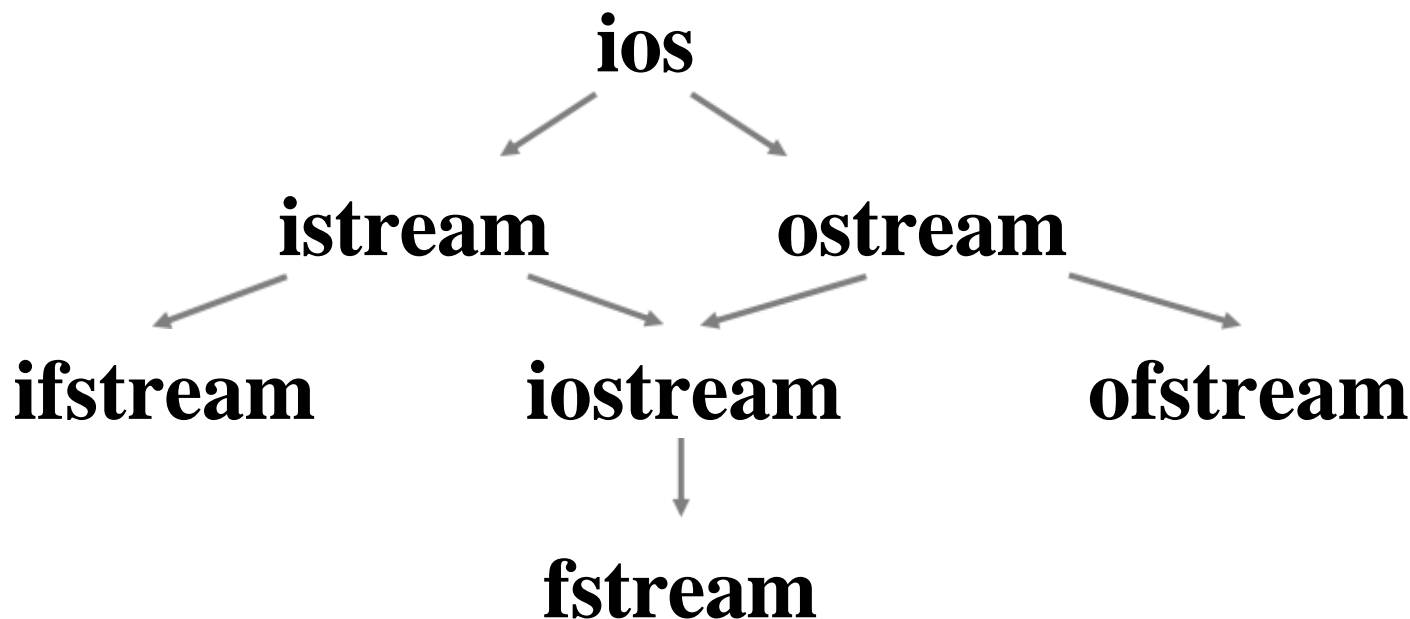
- 输入输出相关的类
- 流操纵算子
- 文件读写

与输入输出流操作相关的类



- **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**
- **iostream** 是既能用于输入, 又能用于输出的类

与输入输出流操作相关的类



- **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**
- **iostream** 是既能用于输入, 又能用于输出的类
- **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
- **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

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

* 重定向: 将输入的源或输出的目的地改变

标准流对象

- 输出流对象

- **cout** 与标准输出设备相连

cout 对应于标准输出流, 用于向屏幕输出数据, 也可以被重定向为从文件中写入数据

- **cerr** 与标准错误输出设备相连

- **clog** 与标准错误输出设备相连

cerr和clog的区别: 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

标准流对象

- 输出流对象

- **cout** 与标准输出设备相连

cout 对应于标准输出流, 用于向屏幕输出数据, 也可以被重定向为从文件中写入数据

- **cerr** 与标准错误输出设备相连

- **clog** 与标准错误输出设备相连

cerr和clog的区别: 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

判断输入流结束

- 可以用如下方法判输入流结束:

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

- 如果从键盘输入, 则在单独一行输入Ctrl+Z代表输入流结束
- 如果从文件输入, 例如前面有

```
freopen("some.txt", "r", stdin); //将一个指定的文件打开一个  
//预定义的流: 标准输入
```

读到文件尾部, 输入流就算结束

判断输入流结束

- 原理

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 ) { ... } // 类型不匹配, 为什么可以?
```

在 `istream` 或其基类里重载了 `operator bool`

```
#include <iostream>
using namespace std;
class MyCin{
    ...
};

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

补齐MyCin类, 要求输入100, 则程序结束

```
#include <iostream>
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;
    }
};
```

istream类的成员函数

- istream类的成员函数

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

从输入流中读取 (bufSize-1) 个字符到缓冲区, 或读到 '\n' 为止 (哪个先到算哪个)

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

从输入流中读取 (bufSize-1) 个字符到缓冲区, 或读到delim为止

istream类的成员函数

- **istream类的成员函数**

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

从输入流中读取 (bufSize-1) 个字符到缓冲区, 或读到 '\n' 为止 (哪个先到算哪个)

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

从输入流中读取 (bufSize-1) 个字符到缓冲区, 或读到delim为止

- 两个函数都会自动在缓冲区中读入数据的结尾添加 '\0'
- '\n' 或 delim 都不会被读入缓冲区, 但会被从输入流中取走
- 如果输入流 '\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: 本次读入已经完成, 但之后的读入就都会失败

□ 可以用 **if(!cin.getline (...))** 判断输入是否结束

istream类的成员函数

- `bool eof();` //判断输入流是否结束
- `int peek();` //返回下一个字符, 但不从流中去掉
- `istream & putback(char c);` //将字符c 放回输入流
- `istream & ignore(int nCount = 1, int delim = EOF);`
//从流中删掉最多nCount 个字符, 遇到EOF 时结束

istream类的成员函数

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int x;
5      char buf[100];
6      cin >> x;
7      cin.getline(buf, 90);
8      cout << buf << endl;
9      return 0;
10 }
```

- 输入:
12 abcd
- 输出
abcd (空格+abcd)
- 输入
12
 - 程序立即结束, 无输出:

因为getline读到留在流中的'\n'就会返回

输出重定向

```
1  #include <iostream>
2  using namespace std;
3  int main()  {
4      int x,y;
5      cin >> x >> y;
6      freopen("test.txt", "w", stdout); //将标准输出重定向到 test.txt 文件
7      if( y == 0 ) //除数为 0 则在屏幕上输出错误信息
8          cerr << "error." << endl;
9      else
10         cout << x / y ; //输出结果到 test.txt
11     return 0;
12 }
```


输出重定向

```
1  #include <iostream >
2  using namespace std;
3  int main() {
4      double f;
5      int n;
6      freopen("t.txt", "r", stdin); //cin 被改为从 t.txt 中读取数据
7      cin >> f >> n;
8      cout << f << "," << n << endl;
9      return 0;
10 }
```

t.txt:

3.14 123

输出:

3.14,123

流操纵算子

- 整数流的基数: dec, oct, hex, setbase
- 浮点数的精度(precision, setprecision)
- 设置域宽(setw, width)
- 用户自定义的流操纵算子

使用流操纵算子需要 `#include <iomanip>`

流操纵算子

- 整数流的基数：流操纵算子 `dec`, `oct`, `hex`

```
int n = 10;
```

```
cout << n << endl;
```

```
cout << hex << n << “\n” //十六进制
```

```
    << dec << n << “\n” //十进制
```

```
    << oct << n << endl; //八进制
```

- 输出结果：

10

a

10

12

流操纵算子

- 浮点数的精度 (**precision**, **setprecision**)

precision是成员函数, 其调用方式为:

cin.precision(5);

setprecision 是流操作算子, 其调用方式为:

cin >> setprecision(5); // 可以连续输出

- 功能相同:
 - 指定输出浮点数的有效位数 (非定点方式输出时)
 - 指定输出浮点数的小数点后的有效位数 (定点方式输出时)

*定点方式: 小数点必须出现在个位数后面

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

- 流格式操纵算子 **setiosflags**
 - `setiosflags(ios::fixed)` 用定点方式表示实数
 - `seiosflags(ios::scientific)` 用指数方式表示实数
 - `setiosflags(ios::fixed)` 可以和 `setprecision(n)` 合用
控制小数点右边的数字个数
 - `seiosflags(ios::scientific)` 可以和 `setprecision(n)` 合用
控制指数表示法的小数位
- 在用浮点表示的输出中, `setprecision(n)`表示有效位数
- 在用定点/指数表示的输出中, `setprecision(n)`表示小数位数
- 小数位数截短显示时, 进行四舍五入处理

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 ;
```

```
}
```

取消以小数点位置
固定的方式输出

输出:

1234567.890000

1.23457e+006

流操纵算子

- 设置域宽 (**setw, width**)

两者功能相同，一个是流操作算子，另一个是成员函数，

调用方式不同：

cin >> setw(5); 或者 cin.width(5);

cout << setw(5); 或者 cout.width(5);

- 不含参数的width函数将输出当前域宽

流操纵算子

- 设置域宽 (**setw, width**)

例: **intw = 4;** 输入: **1234567890**

char string[10]; 输出： 1234

```
cin.width(5); 5678
```

```
while(cin >> string){ 90
```

cout.width(w++);

```
cout << string << endl;
```

cin.width(5);

}

输入操作提取字符串的最大宽度**比定义的域宽小1**,

因为在输入的字符串后面必须加上一个空字符

流操纵算子

- 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

流操纵算子

- 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

用户自定义流操纵算子

```
1 ostream &tab(ostream &output){  
2     return output << '\t';  
3 }  
4 cout << "aa" << tab << "bb" << endl;
```

输出: aa bb

为什么可以?

用户自定义流操纵算子

```
1 ostream &tab(ostream &output){  
2     return output << '\t';  
3 }  
4 cout << "aa" << tab << "bb" << endl;
```

输出: aa bb

为什么可以? 因为iostream 里对<<进行了重载(成员函数)

```
1 ostream & operator <<(ostream & ( * p ) ( ostream & )) ;
```

该函数内部会调用 p 所指向的函数, 且以*this 作为参数
hex / dec / oct 都是函数

数据的层次

- 位 bit
- 字节 byte
- 域 / 记录

例：学生记录

intID;

char name[10];

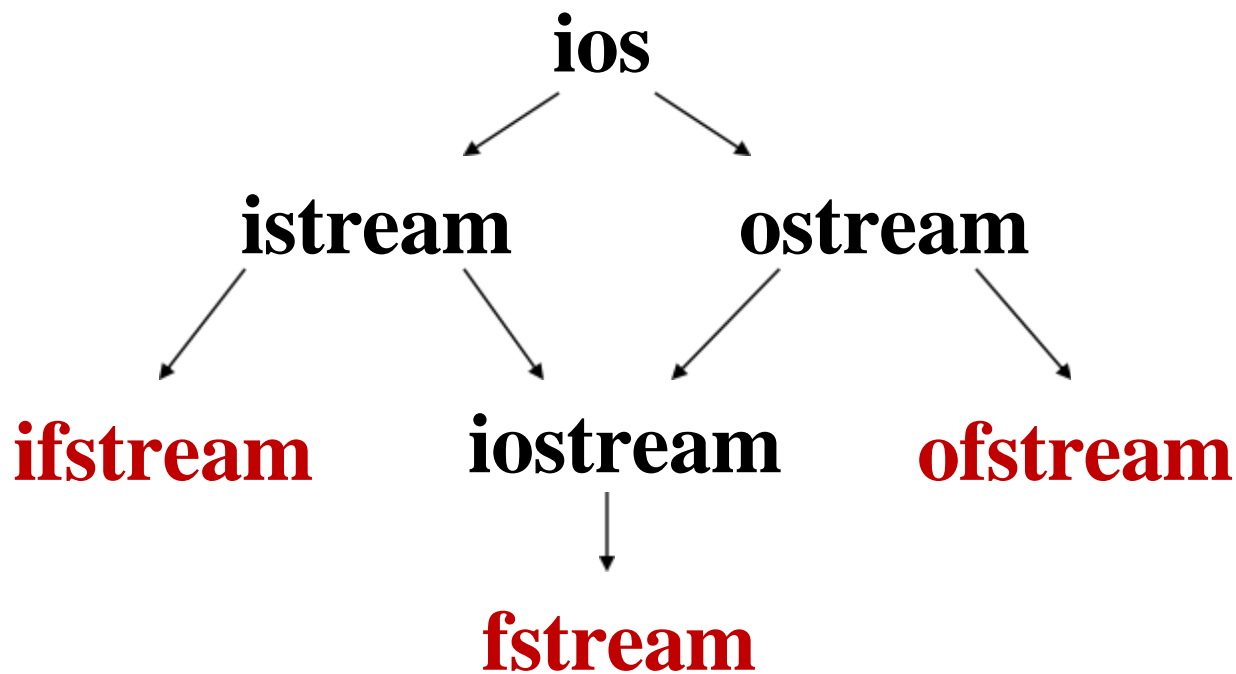
int age;

int rank[10];

- 将所有记录顺序地写入一个文件, 称为 顺序文件

文件和流

- 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：



建立顺序文件

```
#include <fstream> // 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::ate** 输出到文件, 保留原有内容, 在文件任意位置添加
- ios::binary 以二进制 **文件格式打开文件**

建立顺序文件

- 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

文件的读写指针

- 对于输入文件, 有一个**读指针**
- 对于输出文件, 有一个**写指针**
- 对于输入输出文件, 有一个**读写指针**
- 标识文件操作的当前位置, 该指针在哪里, 读写操作就在哪里进行

文件的读写指针

```
ofstream fout("a1.out", ios::ate);
```

```
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
```

文件的读写指针

```
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: **`in.txt`** 的内容为:

1 234 9 45 6 879

2 4

则执行本程序后, 生成的 **`out.txt`** 的内容为:

1 2 4 6 9 45 234 879

假定待排序的数不超过1000个

参考程序

```
#include "iostream"
#include "fstream"
#include "algorithm"
using namespace std;
int aNum[1000];

int main() {
    ifstream srcFile("in.txt", ios::in);
    ofstream destFile("out.txt", ios::out);
    int x;

    int n = 0;
    while( srcFile >> x )
        aNum[n++] = x;

    sort(aNum, aNum + n); //定义在头文件<algorithm>
    for( int i = 0; i < n; i ++ )
        destFile << aNum[i] << " ";
    destFile.close();
    srcFile.close();
}
```

二进制文件的读写

```
int x=10;  
fout.seekp(20, ios::beg);  
fout.write((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

二进制文件的读写

// 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( strcmp(s.szName, "exit") == 0) //名字为exit则结束
            break;
        OutFile.write( (char * ) & s, sizeof(s) );
    }
    OutFile.close();
    return 0;
}
```

Note -- 文本文件/二进制文件打开文件的区别:

- 在Unix/Linux下，二者一致，没有区别;
- 在Windows下，文本文件是以“\r\n”作为换行符
→ 读出时，系统会将0x0d0a只读入0x0a
→ 写入时，对于0x0a系统会自动写入0x0d

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 烫烫烫烫烫烫烫烫藥 Jane 烫烫烫烫
烫烫烫?

二进制文件的读写

//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("c:\\tmp\\students.dat", ios::in);  
    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.nScore << endl;  
    }  
    InFile.close();  
    return 0;  
}
```

输出：
Tom 60
Jack 80
Jane 40

二进制文件的读写

//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

显式关闭文件

- `ifstream fin("test.dat", ios::in);`
`fin.close();`
- `ofstream fout("test.dat", ios::out);`
`fout.close();`

例: 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;
}
```

二进制文件和文本文件的区别

- Linux / Unix 下的换行符号： '\n' (ASCII码: 0x0a)
- Windows 下的换行符号： '\r\n' (ASCII码: 0x0d0a)

endl 就是 '\n'

- Mac OS 下的换行符号： '\r' (ASCII码: 0x0d)
- 导致 Linux, Mac OS 文本文件在 Windows 记事本中打开时不换行

二进制文件和文本文件的区别

- Unix/Linux下打开文件, 用不用 `ios::binary` 没区别
- Windows下打开文件, 如果不用 `ios::binary`, 则:
 - 读取文件时, 所有的 `'\r\n'` 会被当做一个字符 `'\n'` 处理, 即少读了一个字符 `'\r'`
 - 写入文件时, 写入单独的 `'\n'` 时, 系统自动在前面加一个 `'\r'`, 即多写了一个 `'\r'`

Thanks !

