C/C++的文件操作

C++的文件操作：P.419 - P.433

13.4 对数据文件的操作与文件流

13.4中又用到了13.1-13.3的内容

C语言的文件操作：补充

讲课顺序：13.1-13.4

补充C语言对文件的操作

§13. 输入输出流

13.1. C++的输入与输出

13.1.1. 第3章中有关输入输出的概念

3.4.1. 流的基本概念

流的含义：流是来自设备或传给设备的一个数据流，由一系列字节组成，按顺序排列

★ C/C++的原生标准中没有定义输入/输出的基本语句

★ C语言用printf/scanf等函数来实现输入和输出，通过#include <stdio.h>来调用

★ C++通过cin和cout的流对象来实现，通过#include <iostream>来调用

cout：输出流对象　　<<：流插入运算符

cin：输入流对象　　>>：流提取运算符

3.4.2. 输出流的基本操作

格式：cout << 表达式1 << 表达式2 << ... << 表达式n;

★ 插入的数据存储在缓冲区中，不是立即输出，要等到缓冲区满(不同系统大小不同)或者碰到换行符("\n"/endl)或者强制立即输出(flush)才一齐输出

★ 默认的输出设备是显示器(可更改，称输出重定向)

★ 一个cout语句可写为若干行，或者若干语句

★ 一个cout的输出可以是一行，也可以是多行，多个cout的输出也可以是一行

★ 一个插入运算符只能输出一个值

★ 系统会自动判断输出数据的格式

§13. 输入输出流

13.1. C++的输入与输出

13.1.1. 第3章中有关输入输出的概念

3.4.3. 输入流的基本操作

格式：cin >> 变量1 >>变量2 >> ... >>变量n;

★ 键盘输入的数据存储在缓冲区中，不是立即被提取，要等到缓冲区满(不同系统大小不同)或碰到回车符才进行提取

★ 默认的输入设备是键盘(可更改，称输入重定向)

★ 一行输入内容可分为若干行，或者若干语句

★ 一个提取运算符只能输入一个值

★ 提取运算符后必须跟变量名,不能是常量/表达式等

★ 输入终止条件为回车、空格、非法输入

★ 系统会自动判断输入数据，若超过变量范围则错误

★ 字符型变量只能输入图形字符(33-126)，不能以转义符方式输入(单双引号、转义符全部当作单字符)

★ 浮点数输入时，可以是十进制数或指数形式，只取有效位数(4舍5入)

★ cin不能跟endl，否则编译错

§13. 输入输出流

13.1. C++的输入与输出

13.1.1. 第3章中有关输入输出的概念

3.4.5. 字符的输入和输出

3.4.5.1. 字符输出函数putchar

形式：putchar(字符变量/常量)

功能：输出一个字符

```
char a='A';
putchar(a);
putchar('A') ;
putchar('\x41');
putchar('\101');
```

★ 加#include <cstdio>或#include <stdio.h>

3.4.5.2. 字符输入函数getchar

形式：getchar()

功能：输入一个字符(给指定的变量)

★ 加#include <cstdio>或#include <stdio.h>

★ 返回值是int型，是输入字符的ASCII吗，可赋值给字符型/整型变量

★ 输入时有回显，输入后需按回车结束输入(若直接按回车则得到回车的ASCII码)

★ 可以输入空格，回车等cin无法处理的非图形字符，但仍不能处理转义符

★ cin/getchar 等每次仅从输入缓冲区中取需要的字节，多余的字节仍保留在输入缓冲区中供下次读取

§13. 输入输出流

13.1. C++的输入与输出

13.1.2. 输入输出的基本概念

输入输出的种类：

系统设备：标准输入设备：键盘

(标准I/O) 标准输出设备：显示器

其它设备：鼠标、打印机、扫描仪等

外存文件：从文件中得到输入

(文件I/O) 输出到文件中

内存空间：输入/输出到一个字符数组中

(串I/O)

★ 操作系统将所有系统设备都统一当作文件进行处理

C++输入/输出的特点：

★ 与C兼容，支持printf/scanf

★ 对数据类型进行严格的检查，是类型安全的I/O操作

★ 具有良好的可扩展性，可通过重载操作符的方式输入/输出自定义数据类型

C++的输入/输出流：

★ 采用字符流方式，缓冲区满或遇到endl才输入/输出

★ cin,cout不是C++的语句，也不是函数，是类的对象 >> 和 << 的本质是左移和右移运算符，被重载为输入和输出运算符

§13. 输入输出流

13.1. C++的输入与输出

13.1.2. 输入输出的基本概念

C++中与输入/输出相关的类及对象：(P. 403 - 407)

iostream类库中有关的类

与iostream类库有关的头文件

在iostream头文件中定义的流对象

在iostream头文件中重载运算符

§13. 输入输出流

13.2. 标准输出流

13.2.1. cout，cerr和clog流

cout：向控制台进行输出，缺省是显示器

cerr：向标准出错设备进行输出，缺省是显示器

（直接输出，不必等待缓冲区满或回车）

clog：向标准出错设备进行输出，缺省是显示器

（放在缓冲区中，等待缓冲区满或回车才输出）

★ 三者的使用方法一样

★ 缺省都是显示器，可根据需要进行输出重定向

13.2.2. 格式输出(P.409-412表格及应用举例)

需要掌握的基本格式：

不同数制：dec、hex、oct

设置宽度：setw

左右对齐：setiosflags(ios::left/right)

其余当作手册来查：

P.50　表3.1

P.410 表13.3、13.4　（错误：所有iso=>ios）

★ 输出格式可用控制符控制，也可以流成员函数形式

P.410-411 例13.2

§13. 输入输出流

13.2. 标准输出流

13.2.3. 流成员函数put

形式：cout.put(字符常量/字符变量)

★ 功能与putchar相同，输出一个字符

```
char a='A';
cout.put(a);        //变量
cout.put('A') ;     //常量
cout.put('\x41'); //十六进制转义符
cout.put('\101'); //八进制转义符
cout.put(65) ;      //整数当作ASCII码
cout.put(0x41) ;   //整数当作ASCII码(十六)
cout.put(0101) ;   //整数当作ASCII码(八)
```

★ 允许连续调用

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

int main()
{
    cout.put(72).put(0x65).put('l').put(0154).put('a'+14);
    return 0;
}
```

Hello

```
//P.413 例13.3
#include <iostream>
using namespace std;
int main()
{   char *p="BASIC";
    for(int i=4; i>=0; i--)
       cout.put(*(p+i));
    cout.put('\n');
    return 0;;
}
```

CISAB

§13. 输入输出流

13.3. 标准输入流

13.3.1. cin流

★ cin提取数据后，会根据数据类型是否符合要求而返回逻辑值

```
#include <iostream>
using namespace std;
int main()
{
    int a=-9;
    cin >> a;
    cout << a << " " << (cin ? 1 : 0) << endl;
    return 0;
} //不同编译器，cin为0时，a值可能不同
```

输入	cout的结果	
10	10	1
ab	-9	0
12ab	12	1
很大的数字	-9	0

● 上例为VS2017下的运行结果

输入	cout的结果	
10	10	1
ab	0	0
12ab	12	1
很大的数字	2147483647	0

● 上例为CodeBlocks运行结果

循环一直执行，
直到输入为非数字格式

```
#include <iostream>
using namespace std;
int main()
{
    float grade;
    cout << "enter grade:";
    while(cin>>grade) {
        if (grade>=85 && grade<=100)
            cout << "Good!" << endl;
        if (grade<60)
            cout << "fail!" << endl;
        }
     return 0;
}
```

★ 允许进行输入重定向

§13. 输入输出流

13.3. 标准输入流

13.3.1. cin流

13.3.2. 文件结束符与文件结束标记

13.3.3. 用于字符输入的流成员函数

13.3.4. 与字符输入有关的其它成员函数

通过作业理解并记忆

§13.输入输出流

13.4.文件操作与文件流

13.4.1.文件的基本概念

文件及文件名：

文件：存储在外存储器上的数据的集合

文件名：操作系统用于访问文件的依据

文件的分类：

★ 按设备分

输入文件：键盘等输入设备

输出文件：显示器、打印机等输出设备

磁盘文件：存放在磁盘(光盘、U盘)上的文件

★ 按文件的类型分：

程序文件：执行程序所对应的文件(.exe/.dll等)

数据文件：存放对应数据的文件(.cpp/.doc等)

★ 按数据的组织形式

ASCII码文件(文本文件)：按数据的ASCII代码形式存放的文件

二进制文件：按数据的内存存放形式存放的文件

§13. 输入输出流

13.4. 文件操作与文件流

13.4.1. 文件的基本概念

文件的分类：

★ 按数据的组织形式

ASCII码文件(文本文件)：按数据的ASCII代码形式存放的文件

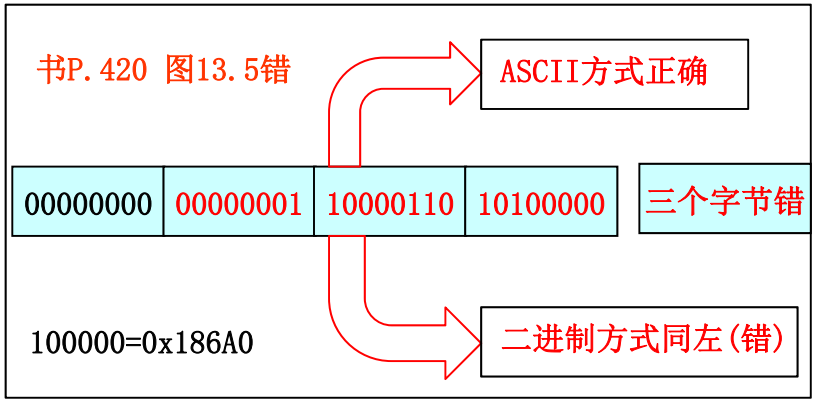
二进制文件：按数据的内存存放形式存放的文件

例：int型整数100000：ASCII文件为6个字节

二进制文件为4个字节

\x31	\x30	\x30	\x30	\x30	\x30

\x00	\x01	\x86	\xA0



双精度数123.45：ASCII文件为6个字节

二进制文件为8个字节

\x31	\x32	\x33	\x2E	\x34	\x35

d	o	u	b	l	e	格	式

字符串"China"：ASCII文件为6个字节

二进制文件为6个字节

\x43	\x68	\x69	\x6E	\x61	\x0

\x43	\x68	\x69	\x6E	\x61	\x0

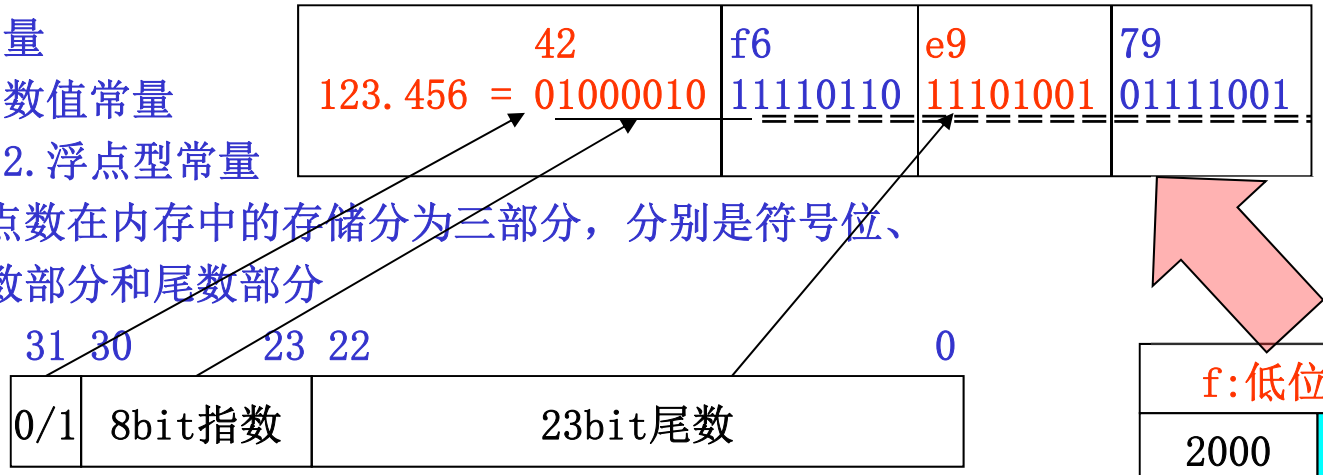
§2. 数据类型和表达式

2.2. 常量

2.2.2. 数值常量

2.2.2.2. 浮点型常量

★ 浮点数在内存中的存储分为三部分，分别是符号位、指数部分和尾数部分



浮点数存储遵守 IEEE 754 规范(具体不做要求)

P.22 图2.3仅是一个分段示范，不准确

f:低位在前存放	
2000	0111 1001
2001	1110 1001
2002	1111 0110
2003	0100 0010

```
//用于看懂float/double内部存储格式的例子
#include <iostream>
using namespace std;
int main()
{   float f = 123.456f;
    char *p = (char *)&f;
    cout << hex << int(*p)     << endl;
    cout << hex << int(*(p+1)) << endl;
    cout << hex << int(*(p+2)) << endl;
    cout << hex << int(*(p+3)) << endl;
    return 0;
}
```

```
79
ffffffe9
ffffff6
42
```

问：想知道float/double型数据的二进制形式到底是怎样的，应该怎么做？

§13.输入输出流

13.4.文件操作与文件流

13.4.1.文件的基本概念

C++对文件的访问：

低级I/O：字符流方式输入/输出

高级I/O：转换为数据指定形式的输入/输出

13.4.2.文件流类及文件流对象

与磁盘文件有关的流类：

输入　　：ifstream类，从istream类派生而来

输出　　：ofstream类，从ostream类派生而来

输入/输出：fstream类，从iostream类派生而来

流对象的建立：

ifstream 流对象名：用于输入文件的操作

ofstream 流对象名：用于输出文件的操作

fstream　流对象名：用于输入/输出文件的操作

§13. 输入输出流

13.4. 文件操作与文件流

13.4.3. 文件的打开与关闭

文件的打开：

文件流对象名.open(文件名，打开方式);

★ 加#include <fstream>

★ 打开方式见 P.422 表13.5

ios::nocreate
ios::noreplace

DevC++/CB/Linux不支持

在VS2017下是

ios::_Nocreate
ios::_Noreplace

★ 各个打开方式可用"位或运算符|"进行组合

1、..表示父目录
.表示当前目录
2、为什么要\\ ？

★ 文件名允许带全路径，若不带路径，则表示与可执行文件同目录

```
ofstream out;
out.open("aa.dat", ios::out);
out.open("..\\C++\\aa.dat", ios::out);
out.open(".\\C++\\aa.dat", ios::out);
out.open("\\C++\\aa.dat", ios::out| ios::app);
out.open("c:\\C++\\aa.dat", ios::out);
```

下面5个open，分别打开的是哪个文件？

```
C:\
   |--test      (文件夹)
   |  |--aa.dat
   |  |--C++    (文件夹)
   |     |--aa.dat
   |--C++       (文件夹)
      |--aa.dat
```

```
D:\
   |--test      (文件夹)
   |  |--aa.dat
   |  |--C++    (文件夹)
   |     |--aa.dat
   |--C++       (文件夹)
      |--aa.dat
```

● VS2017等编译器，如在集成环境内运行，则当前目录是指源程序文件(*.cpp)所在的目录，如果离开集成环境(例如用cmd命令行运行)，则当前目录是指可执行文件(*.exe)所在目录

§13. 输入输出流

13.4. 文件操作与文件流

13.4.3. 文件的打开与关闭

文件的打开：

文件流对象名.open(文件名，打开方式);

★ 可在声明文件流对象时直接打开

```
ofstream out("aa.dat", ios::out);
```

P.422 第5行错 ostream => ofstream

★ 打开方式与文件流对象之间要兼容,否则无意义

```
ifstream in;
in.open("aa.dat", ios::out);
//in对象用out打开，无意义
```

★ 每个文件被打开后，都有一个文件指针，初始指向开始/末尾的位置(根据打开方式决定)

★ 判断文件是否成功打开的方法，不同编译器有差别(P.423 说明(4) 错误)

if (outfile.open("f1.dat",ios::app)==0) if (!outfile.open("f1.dat",ios::app))	错
if (outfile==NULL) if (!outfile)	两种方法均可以 1、判断流对象 2、is_open函数
if (outfile.is_open()==0) if (!outfile.is_open())	

文件的关闭：

文件流对象名.close();

§13. 输入输出流

13.4. 文件操作与文件流

13.4.3. 文件的打开与关闭

文件的打开：

文件流对象名.open(文件名，打开方式);

文件的关闭：

文件流对象名.close();

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

int main()
{   ifstream in;
    in.open("bb.dat", ios::in);
    if (!in.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    in.close();

    return 0;
}
```

不存在时：失败
存在时：成功

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

int main()
{   ofstream out;
    out.open("aa.dat", ios::out);
    if (out.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    out.close();

    return 0;
}
```

不存在时：成功(创建)
存在时：成功(覆盖)
存在并只读：失败

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

int main()
{   ofstream out;
    out.open("bb.dat", ios::out|ios::_Nocreate);
    if (out.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    out.close();
}
```

不存在时：失败
存在时：成功(覆盖)
存在并只读：失败

§13. 输入输出流

13.4. 文件操作与文件流

13.4.4. 对ASCII文件的操作

基本方法：将文件流对象名当作cin/cout对象，用>>和<<进行格式化的输入和输出，同时也前面介绍的关于cin/cout的get/getline/put/eof/peek/putback/ignore等成员函数

(作业内容)

★ >>和<<使用时的注意事项与cin、cout时相同 ⇨ cin >> 变量 => infile >> 变量；cout << 变量 => outfile << 变量

★ 成员函数的使用方法与前面相同 ⇨ cout.put('A') => outfile.put('A')

```
#include <iostream>     //P.423-424 例13.8
#include <fstream>
using namespace std;

int main()
{  int a[10];
   ofstream outfile("f1.dat", ios::out);
   if (!outfile.is_open()) {
      cerr << "open error!" << endl;
      exit(1);  //结束程序运行，向操作系统返回1
      }
   cout << "enter 10 integer numbers:" << endl;
   for(int i=0; i<10; i++) {
      cin >> a[i];              //键盘输入
      outfile << a[i] << " ";   //int型输出到文件
      }
   outfile.close();
   return 0;
}
```

运行两次，观察结果

```
#include <iostream>    //P.423-424 例13.8变化
#include <fstream>
using namespace std;

int main()
{  int a[10];
   ofstream outfile("f1.dat", ios::out | ios::app);
   if (!outfile.is_open()) {
      cerr << "open error!" << endl;
      exit(1);  //结束程序运行，向操作系统返回1
      }
   cout << "enter 10 integer numbers:" << endl;
   for(int i=0; i<10; i++) {
      cin >> a[i];              //键盘输入
      outfile << a[i] << " ";   //int型输出到文件
      }
   outfile.close();
   return 0;
}
```

运行两次，观察结果

§13. 输入输出流

13.4. 文件操作与文件流

13.4.4. 对ASCII文件的操作

```
#include <iostream>     //P.425 例13.9
#include <fstream>
using namespace std;
int main()
{
    int a[10], max, i, order;                    ifstream用_Nocreate无意义
    ifstream infile("f1.dat", ios::in | ios::_Nocreate);
    if (!infile.is_open()) {
        cerr << "open error!" << endl;
        exit(1);  //结束程序运行，向操作系统返回1
        }
    for(i=0; i<10; i++) {
        infile >> a[i];          //从文件中读10个int放入a数组
        cout << a[i] << " ";     //int型输出到屏幕
        }

   //找最大值，省略...

   infile.close();
   return 0;
}
```

利用例13.8生成的f1.dat
自己编辑完全正确的f1.dat
自己编辑含错误的f1.dat

§13. 输入输出流

13.4. 文件操作与文件流

13.4.4. 对ASCII文件的操作

例：打开d:\test\data.txt文件,并将内容输出到屏幕上

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

int main()
{
    ifstream in;
    char ch;

    in.open("d:\\test\\data.txt", ios::in); //双斜杠
    if (in.is_open()==0) {            //!in.is_open()
        cout << "文件打开失败\n";  //cerr
        return -1;                    //exit(1)
        }

    while(!in.eof()) {
        ch = in.get();              //in.get(ch);
        putchar(ch);                //cout.put(ch);
        }

    in.close();

    return 0;
}
```

```
while((ch=in.get())!=EOF)
    cout.put(ch);

EOF是系统定义的文件结束标记
```

例：将 d:\test\data.txt 文件复制为 d:\demo\data2.txt

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

int main()
{   ifstream in;
    ofstream out;
    char ch;
    in.open("d:\\test\\data.txt", ios::in);
    if (!in.is_open()) {
        cout << "无法打开源文件" << endl;
        return -1;
        }
    out.open("d:\\demo\\data2.txt", ios::out);
    if (!out.is_open()) {
        cout << "无法打开目标文件" << endl;
        in.close();  //记得关掉
        return -1;
        }

    while(in.get(ch))  //从输入文件中一次读取
        out.put(ch);   //写入输出文件中

    in.close();
    out.close();
    return 0;
}
```

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 用ASCII文件的字符方式进行操作(按字节读写)

★ 用read/write进行操作

文件流对象名.read(内存空间首指针，长度);

从文件中读长度个字节，放入从首指针开始的空间中

文件流对象名.write(内存空间首指针，长度);

将从首指针开始的连续长度个字节写入文件中

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 用read/write进行操作

以ASCII文件方式写入

```
//P.428-429例13.11变化 - ASCII方式写入
#include <iostream>  //书上缺，要补上
#include <fstream>
#include <cstdlib>   //exit用
using namespace std;

struct student {
   char name[20];
   int  num;
   int  age;
   char sex;
};
int main()
{
    student stud[3]={"Li",1001,18,'f', "Fun",1002,19,'m',"Wang",1004,17,'f'};
    ofstream outfile("stud.dat", ios::out);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;
        exit(-1);  //强制结束程序
        }

    for(int i=0; i<3; i++)
        outfile << stud[i].name << stud[i].num  << stud[i].age << stud[i].sex << endl;

    outfile.close();
    return 0;
}
```

生成的stud.dat文件共36字节：

```
Li100118f
Fun100219m
Wang100417f
```

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 用read/write进行操作

```
//P.428-429例13.11

#include <iostream>  //书上缺，要补上
#include <fstream>
using namespace std;

struct student {
   char name[20];
   int  num;
   int  age;
   char sex;
};
int main()
{
    student stud[3]={"Li",1001,18,'f', "Fun",1002,19,'m',"Wang",1004,17,'f'};
    ofstream outfile("stud.dat", ios::binary);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;
        exit(-1);  //强制结束程序
        }

    for(int i=0; i<3; i++) //一次写一个数组元素（32字节）
        outfile.write((char *)&stud[i], sizeof(stud[i]));

    outfile.close();
    return 0;
}
```

outfile.write((char *)stud, sizeof(stud));//整个数组96字节）

以二进制文件方式写入

stud.dat文件共96字节，前32字节为：

```
4C 69 00 ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? E9 03 00 00
12 00 00 00 66 ?? ?? ??
```

```
4C6900      => "Li"(含尾零,多余17个)
E9030000    => 0x000003E9 => 1001
12000000    => 0x00000012 => 18
66          => 'f'(后3个是填充字节)
```

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 用read/write进行操作

以二进制文件方式读取

```
//P.429-430例13.12
#include <iostream>  //书上缺，要补上
#include <fstream>
using namespace std;

struct student {
   char name[20];  //不能是string name。必须是char name[20]，[19]或[21]都不行，必须要保证与文件的32字节一致
   int  num;
   int  age;
   char sex;
};
int main()
{   student stud[3];
    int i;
    ifstream infile("stud.dat", ios::binary); //stud.dat的内容是由例13.11生成的二进制文件
    if (!infile.is_open()) {
         cerr << "open error!" << endl;
         exit(-1);  //退出
         }
    for(i=0; i<3; i++)  //一次读入一个数组元素（32字节）
         infile.read((char *)&stud[i], sizeof(stud[i]));
    infile.close();
    for(i=0; i<3; i++) {
         cout << "No." << i+1 << endl;
         cout << "name:" << stud[i].name << endl;
         cout << "num:" << stud[i].num << endl;
         cout << "age:" << stud[i].age << endl;
         cout << "sex:" << stud[i].sex << endl << endl;
         }                                    //多空一行
    return 0;
}
```

§13.输入输出流

13.4.文件操作与文件流

13.4.5.对二进制文件的操作

★ 与文件指针有关的流成员函数

适用于输入文件的：

gcount()　　：返回最后一次读入的字节

tellg()　　：返回输入文件的当前指针

seekg(位移量，位移方式)：移动输入文件指针

适用于输出文件的：

tellp()　　：返回输出文件的当前指针

seekp(位移量，位移方式)：移动输出文件指针

位移方式：

ios::beg：从文件头部移动，位移量必须为正

ios::cur：从当前指针处移动，位移量可正可负

ios::end：从文件尾部移动，位移量必须为负

★ 随机访问二进制数据文件

在文件的读写过程中，可前后移动文件指针，达到按需读写的目的

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 与文件指针有关的流成员函数

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

struct student {
   char name[20];
   int  num;
   int  age;
   char sex;
};

int main()          //在P.428-429 例13.11 的基础上改
{
    student stud[3]={"Li",1001,18,'f', "Fun",1002,19,'m', "Wang", 1004, 17, 'f'};
    ofstream outfile("stud.dat", ios::binary);
    if (!outfile.is_open()) {
       cerr << "open error!" << endl;
       exit(-1);  //强制结束程序的运行
       }
    for(int i=0; i<3; i++) { //一次写入一个数组元素
       outfile.write((char *)&stud[i], sizeof(stud[i]));
       cout << outfile.tellp() << endl;
       outfile.seekp(0, ios::beg);
       cout << outfile.tellp() << endl;
       }
    outfile.close();
    return 0;
}
```

```
32
0
32
0
32
0
```

stud.dat文件为32字节：

57	61	6E	67	00	??	??	??
??	??	??	??	??	??	??	??
??	??	??	??	EC	03	00	00
11	00	00	00	66	??	??	??

请观察屏幕输出及stud.dat文件

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 与文件指针有关的流成员函数

```
#include <iostream>
#include <fstream>
using namespace std;
struct student {
   char name[20];
   int  num;
   int  age;
   char sex;
};
int main()          //在P.429-430 例13.12 的基础上改
{   student stud[3];
    int i;
    ifstream infile("stud.dat", ios::binary);
    if (!infile.is_open()) {
       cerr << "open error!" << endl;
       exit(-1);  //强制结束程序的运行
       }
    for(i=0; i<3; i++) { //一次写入一个数组元素
       infile.read((char *)&stud[i], sizeof(stud[i]));
       cout << infile.gcount() << endl;
       cout << infile.tellg() << endl;
       infile.seekg(-32, ios::cur);
       }
    infile.close();
    for(i=0; i<3; i++) {
       cout << "name: " << stud[i].name << endl;
       }
}
```

先运行13.11，保证stud.dat有96字节

```
32
32
32
32
32
32
name: Li
name: Li
name: Li
```

stud.dat文件(96字节)的前32字节：

```
4C 69 00 ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? E9 03 00 00
12 00 00 00 66 ?? ?? ??
```

请观察屏幕输出及stud.dat文件

★ 与文件指针有关的流成员函数

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
using namespace std;
struct student {//结构体与前面不同，28字节
   int num;
   char name[20];
   float score;
};
int main()        //在P.432-433 例13.13
{   student stud[5]={1001,"Li",85, 1002,"Fun",97.5, 1004,"Wang",54, 006,"Tan",76.5, 1010,"Ling",96 };
    fstream iofile("stud.dat", ios::in | ios::out | ios::binary | ios::trunc);  //二进制打开，同时I/O
    if (!iofile) {
       cerr << "open error!" << endl;
       exit(-1); //强制结束程序的运行
       }
    for(int i=0; i<5; i++)  //将数组写入文件中，大小为140字节
       iofile.write((char *)&stud[i], sizeof(stud[i]));

    student stud1[5];
    for(int i=0; i<5; i+=2) { //读stud的[0]/[2]/[4]三个元素，写入stud1的[0]/[1]/[2]中
       iofile.seekg(i*sizeof(stud[i]), ios::beg);
       iofile.read((char *)&stud1[i/2], sizeof(stud1[0]));
       cout << stud1[i/2].num << stud1[i/2].name << endl;
       }
    cout << endl;

    stud[2].num = 1012;             //修改第3个学生的信息
    strcpy(stud[2].name, "Wu");
    stud[2].score = 60;
    iofile.seekp(2*sizeof(stud[0]), ios::beg); //移动文件指针到[2]的位置(Wang)
    iofile.write((char *)&stud[2], sizeof(stud[2]));//覆盖第3个学生(Wang=>Wu)
    iofile.seekg(0, ios::beg); //文件指针移动到开头
    for(int i=0; i<5; i++) { //打印五个学生，其中第三个已被替换
       iofile.read((char *)&stud[i], sizeof(stud[i]));
       cout << stud[i].num << stud[i].name << stud[i].score << endl;
       }
    iofile.close();
    return 0;
}
```

```
1001Li
1004Wang
1010Ling



1001Li85
1002Fun97.5
1012Wu60
1006Tan76.5
1010Ling96
```

§13. 输入输出流

13.4. 文件操作与文件流

13.4.5. 对二进制文件的操作

★ 与文件指针有关的流成员函数

★ 随机访问二进制数据文件

★ 关于二进制访问的几个注意事项

- read参数中的长度是最大读取长度，不是实际读取长度，因此read后要用gcount()返回真实读到的字节数
- 如果读写方式打开(iOS::in | ios::out)，则只有一个文件指针，seekg()和seekp()是同步的，tellg()和tellp()也是同步的
- 在文件的操作超出正常范围后(例：read()已到EOF、seekg()/seekp()超文件首尾范围等)，再次对文件进行seekg()/seekp()/tellg()/tellp()等操作都可能会返回与期望不同的值，建议在文件操作过程中勤用good()/fail()/eof()/clear()等函数，具体自行体会

§13. 输入输出流

13.5. 字符串流

13.5.1. 基本概念

以内存中用户定义的字符数组为输入/输出对象

★ 可以存放各种类型的数据

★ 与标准输入输出流相同，进行ASCII码和二进制之间的相互转换

向字符数组存数据⇔cout：二进制 => ASCII

从字符数组取数据⇔cin ：ASCII => 二进制

★ 不是文件，不需要打开和关闭

13.5.2. 相关流对象的建立

★ 字符串输出流对象：

ostrstream 对象名(字符数组名, 长度, ios::out)

★ 字符串输入流对象：

istrstream 对象名(字符数组名, 长度, ios::in)

★ 字符串输入/输出流对象：

strstream 对象名(字符数组名, 长度, ios::in|ios::out)

★ 加 #include <strstream>

§13. 输入输出流

13.5. 字符串流

13.5.3. 字符串流对象的使用

```
#include <iostream>
#include <strstream>
using namespace std;
int main()
{
    char c[80];
    ostrstream out(c, 80, ios::out);
    out << "Hello" << 10 << 11.2 << endl;
    cout << c << endl;
}
```

1、观察cout的输出
2、将语句替换为下列两种形式，再观察cout的输出
out << "Hello" << 10 << 11.2 << endl << '\0';
out << "Hello" << 10 << 11.2 << endl << ends;

```
#include <iostream>
#include <strstream>
using namespace std;
int main()
{
    char c[80];
    ostrstream out(c, 80, ios::out);
    out << "Hello" << 10 << 11.2 << endl << ends;
    cout << c << endl;
}
```

1、观察cout的输出
2、将语句替换为下列形式，再观察cout的输出
ostrstream out(c, 5);

```
#include <iostream>
#include <strstream>
using namespace std;
int main()
{   char *c = new char[80];
    ostrstream out(c, 80, ios::out);
    out << "Hello" << 10 << 11.2 << ends;
    cout << c << endl;
    delete c;
}
```

★ 不会自动包含'\0'(ends)，需自行加入

★ 定义ostrstream流对象时，第三个参数可省略

★ 向字符串输出流对象的输出内容若超过第二个参数指定的长度，则会出现越界错误（乱码）

★ 字符数组允许动态申请

§13. 输入输出流

13.5. 字符串流

13.5.3. 字符串流对象的使用

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

int main()
{   char c[80] = "Hello 10 11.2";
    istrstream in(c, 80);
    char  s[10];
    int   i;
    float f;
    in >> s >> i >> f;
    cout << s << i << f << endl;
    return 0;
}
```

1、观察cout的输出
2、将语句替换为下列形式，再观察cout的输出
istrstream in(c, 5);

★ 从字符串输入流对象中读取的内容若超过第二个参数指定的长度，则后面无法获取(不确定值)

P.436 解释有错
(2) 如果strout(c,10) 输出会乱码

```
//P.435-436 例13.14
#include <iostream>
#include <strstream>
using namespace std;

struct student {
   int num;
   char name[20];
   float score;
};
int main()
{   student stud[3]={1001,"Li",78, 1002,"Wang",89.5, 1004,"Fun",90};
    char c[50];
    ostrstream strout(c, 30);
    for (int i=0; i<3; i++)
       strout << stud[i].num << stud[i].name << stud[i].score;
    strout << ends;
    cout << "array c:" << c << endl;
    return 0;
}
```

§13. 输入输出流

13.5. 字符串流

13.5.3. 字符串流对象的使用

//P.437 例13.15

```
#include <iostream>
#include <strstream>
using namespace std;
int main()
{   char c[50] = "12 34 65 -23 -32 33 61 99 321 32";
    int a[10], i, j, t;
    cout << "array c:" << c << endl;
    //c中的内容逐个读入int a[10]中
    istrstream strin(c, sizeof(c));  //注意，若动态申请不能sizeof!!!
    for (i=0; i<10; i++)
       strin >> a[i];
    //输出int a[10]的内容
    cout << "array a:";
    for (i=0; i<10; i++)
       cout << a[i] << " ";
    cout << endl;
    //进行排序
    for (i=0; i<9; i++)
       for(j=0; j<9-i; j++)
          if (a[j] > a[j+1]) {
              t = a[j];
              a[j] = a[j+1];
              a[j+1] = t;
              }
    //输出到c中（c刚才用做了输入流）
    ostrstream strout(c, sizeof(c));
    for (i=0; i<10; i++)
       strout << a[i] << " ";
    strout << ends;
    cout << "array c:" << c << endl;
}
```

P.437-438 解释(1)-(5) 理解

与标准输入/输出流的区别：

★ 使用方法及转换方式相同
★ 字符串流对象可重复使用

C语言的文件操作

1.文件指针

FILE *文件指针变量

★ FILE是系统定义的结构体

★ C语言中文件操作的基本依据，所有针对文件的操作均要依据该指针

★ #include <stdio.h> （VS2017可以不需要）

★ VS2017以为不安全，需要加 #define _CRT_SECURE_NO_WARNINGS

★ 文件读写后，文件指针会自动后移

C语言的文件操作

2.文件的打开与关闭

假设：FILE *fp定义一个文件指针

2.1.文件的打开

FILE *fopen(文件名，打开方式)

```
fp = fopen("test.dat", "r");
fp = fopen("c:\\aaa\\test.dat", "w");
```

★ 打开的基本方式如下：

r：只读方式

w：只写方式

a：追加方式

+：可读可写

b：二进制

t：文本方式(缺省)

★ 打开的基本方式及组合见右表

★ 若带路径，则\必须用\\表示

★ 若打开不成功，则返回NUL

打开方式	意义
r/rt	只读方式打开文本文件(不存在则失败)
w/wt	只写方式打开或建立文本文件(存在则清零)
a/at	追加写方式打开或建立文本文件(头读尾写)
rb	只读方式打开二进制文件(不存在则失败)
wb	只写打开或建立二进制文件(存在则清零)
ab	追加写方式打开或建立二进制文件(头读尾写)
r+/rt+	读写方式打开文本文件(不存在则失败)
w+/wt+	读写方式创建文本文件(存在则清零)
a+/at+	读+追加写方式打开或建立文本文件(头读尾写)
rb+	读写方式打开二进制文件(不存在则失败)
wb+	读写方式创建二进制文件(存在则清零)
ab+	读+追加写方式打开二进制文件(头读尾写)

2.2.文件的关闭

fclose(文件指针)

```
fclose(fp);
```

C语言的文件操作

3. 文本文件的读写

3.1. 按字符读写文件

读：char fgetc(文件指针)

写：fputc(字符常量/变量，文件指针)

★ 必须保证文件的打开方式符合要求

```
char ch1;
ch1=fgetc(fp);

char ch2 = 'A';
fputc(ch2, fp);
```

3.2. 判断文件是否到达尾部

int feof(文件指针)

★ 若到达尾部，返回1，否则为0

```
int i;
char ch;
fscanf(fp, "%d%c", &i, &ch);

int i=10;
char ch='A';
fprintf(fp, "%d%c", i, ch);
```

3.3. 按格式读写文件

读：fscanf(文件指针，格式串，输入表列)

写：fprintf(文件指针，格式串，输出表列)

★ 格式串、输入/输出表列的使用同scanf/printf

3.4. 用文件方式进行标准输入输出

stdin ：标准输入设备

stdout ：标准输出设备

stderr ：错误输出设备

```
int i;
fscanf(stdin, "%d", &i); ⇔ scanf("%d", &i); ⇔ cin >> i;
fprintf(stdout, "i=%d",i);⇔ printf("i=%d",i); ⇔ cout << i;
fprintf(stderr, "i=%d", i); ⇔ cerr << i;
```

C语言的文件操作

3. 文本文件的读写

例：

```
//例：打开d:\test\data.txt文件,并将内容输出到屏幕上
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
   FILE *fp;
   char ch;

   fp = fopen("d:\\test\\data.txt", "r");
   if (fp==NULL) {
       printf("文件打开失败\n");
       return -1;
       }

   while(!feof(fp)) {   while(!feof(fp))
       ch = fgetc(fp);      fputc(fgetc(fps), fpt);
       putchar(ch);
       }

   fclose(fp);

   return 0;
}
```

```
//例：打开d:\test\data.txt,将内容复制到d:\demo\data2.txt
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{   FILE *fps, *fpt;
    char ch;
    fps = fopen("d:\\test\\data.txt", "r");
    if (fps==NULL) {
        printf("源文件打开失败\n");
        return -1;
        }
    fpt = fopen("d:\\demo\\data2.txt", "w");
    if (fpt==NULL) {
        printf("目标文件打开失败\n");
        fclose(fps);  //记得关闭
        return -1;
        }
    while(!feof(fps)) {
        ch=fgetc(fps);  //从源文件里面读
        fputc(ch, fpt); //向目标文件写
        }
    fclose(fps);
    fclose(fpt);
    return 0;
}
```

C语言的文件操作

4. 二进制文件的读写

4.1. 按字符读写文件(与文本文件方式相同)

4.2. 按块读写文件

读：fread(缓冲区首址，块大小，块数，文件指针)

写：fwrite(缓冲区首址，块大小，块数，文件指针)

操作：运行这个程序，复制一个文件(不要是1000的整数倍)，比较复制后的文件字节数和源文件的字节数

问题1：为什么会不同？

问题2：如何做到相同？

(提示：弄懂fread和fwrite的返回值，
弄清楚块数和块大小对返回值的影响)

```
/* 例：打开d:\test\data.txt
   将内容复制到d:\demo\data2.txt */
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{   FILE *fps, *fpt;
    char buf[1000];
    fps = fopen("d:\\test\\data.txt", "rb");
    if (fps==NULL) {
        printf("源文件打开失败\n");
        return -1;
        }
    fpt = fopen("d:\\demo\\data2.txt", "wb");
    if (fpt==NULL) {
        printf("目标文件打开失败\n");
        fclose(fps);  //记得关闭
        return -1;
        }
    while(!feof(fps)) {
        fread(buf,1000,1,fps);   //从源文件里面读
        fwrite(buf,1000,1,fpt);  //向目标文件写
        }
    fclose(fps);
    fclose(fpt);
    return 0;
}
```

C语言的文件操作

4. 二进制文件的读写

4.1. 按字符读写文件(与文本文件方式相同)

4.2. 按块读写文件

读：fread(缓冲区首址，块大小，块数，文件指针)

写：fwrite(缓冲区首址，块大小，块数，文件指针)

```
/* 例：从键盘输入3个学生的基本情况，
       写入student.txt文件中 */
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstdio>

struct student {
    int  num;
    char name[9];
    char sex;
    int  age;
    float score;
    };
int main()
{   struct student stu;
    FILE *fp;
    int i;
    fp = fopen("student.txt", "wb");
    if (fp==NULL) {
        printf("文件打开失败\n");
        return 0;
        }
    for(i=0;i<3;i++) {
        scanf("%d%s%c%d%f", &stu.num, stu.name,
                &stu.sex, &stu.age, &stu.score);
        fwrite(&stu, sizeof(struct student),1,fp);
        }
    fclose(fp);
    return 0;
}
```

写入的文件按照前面C++部分的课件，分析十六进制的数据

C语言的文件操作

4.文件指针的移动

4.1.指针复位(回到开头)

rewind(文件指针)

例：rewind(fp);

4.2.任意移动

fseek(文件指针，位移量，位移方式)

例：fseek(fp, 123, SEEK_SET)：从开始移动

fseek(fp, 78, SEEK_CUR)：从当前位置移动

fseek(fp, -25, SEEK_CUR)

fseek(fp, -57, SEEK_END)：从最后移动

★ SEEK_SET的位移必须为正

SEEK_CUR的位移可正可负

SEEK_END的位移必须为负

4.3.求文件指针的当前位置

long ftell(文件指针)

例：ftell(fp);

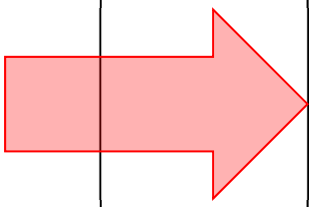
★ 从开始位置计算

C语言中实现与C++的字符串流相似的功能

1. 向字符数组输出格式化的数据

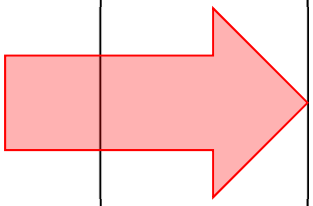
sprintf(字符数组，"格式串"，输出表列)；

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

int main()
{
    char c[80];
    ostrstream out(c, 80, ios::out);
    out << "Hello" << 10 << 11.2 << endl << ends;
    cout << c << endl;

    return 0;
}
```



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

int main()
{
    char c[80];
    sprintf(c, "Hello%d%.1f", 10, 11.2);
    cout << c << endl;

    return 0;
}
```

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

int main()
{
    char *c = new char[80];
    ostrstream out(c, 80, ios::out);
    out << "Hello" << 10 << 11.2 << ends;
    cout << c << endl;
    delete c;

    return 0;
}
```



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

int main()
{
    char *c = new char[80];
    sprintf(c, "Hello%d%.1f", 10, 11.2);
    cout << c << endl;
    delete c;

    return 0;
}
```

C语言中实现与C++的字符串流相似的功能

1.向字符数组输出格式化的数据

sprintf(字符数组，"格式串", 输出表列);

```
//P.435-436 例13.14 改编为C语言向字符串输出格式化的数据
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <strstream>
using namespace std;

struct student {
    int num;
    char name[20];
    float score;
};

int main()
{
    student stud[3]={1001,"Li",78, 1002,"Wang",89.5, 1004,"Fun",90};
    char c[50], *s = c;
    for (int i=0; i<3; i++)
        s+=sprintf(s, "%d %s %.1f", stud[i].num , stud[i].name, stud[i].score);
    cout << "array c:" << c << endl;

    return 0;
}
```

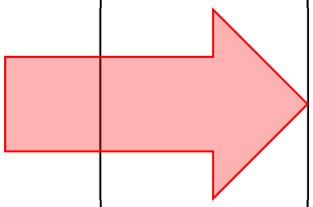
多次向字符数组输出格式化数据
（注意：和C++方式的不同）

C语言中实现与C++的字符串流相似的功能

2. 从字符数组中输入格式化的数据

sscanf(字符数组，"格式串"，输入表列)；

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

int main()
{
    char c[80] = "Hello 10 11.2";
    istrstream in(c, 80);
    char  s[10];
    int   i;
    float f;

    in >> s >> i >> f;

    cout << s << i << f << endl;

    return 0;
}
```



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

int main()
{
    char c[80] = "Hello 10 11.2";
    char  s[10];
    int   i;
    float f;

    sscanf(c, "%s %d %f", s, &i, &f);

    cout << s << i << f << endl;

    return 0;
}
```