C++期末不挂科

CH9 - 文件读写流

本章内容

- 文件输入输出流
- 流操作算子

理解以下名词：

- I/O
- 输入输出流对象
- 文本文件与二进制文件
- 流格式控制符

熟悉以下文件流对象函数：

- open 与 close
- is_open
- << 与 >>
- write 与 read

回答以下问题：

- 文件在C++程序中以什么方式存在以及处理
- 文本文件和二进制文本的区别是什么

记忆以下文件打开方式效果：

- ios::in
- ios::out
- ios::app
- ios::binary

熟悉以下代码：

- 文件的打开 - 判断 - 读写 - 关闭流程
- 通过 << 和 >> 进行文本文件的输入与输出
- 通过 write 和 read 进行二进制文件的输入与输出

文件输入输出

流操作算子

文件IO：IN & OUT

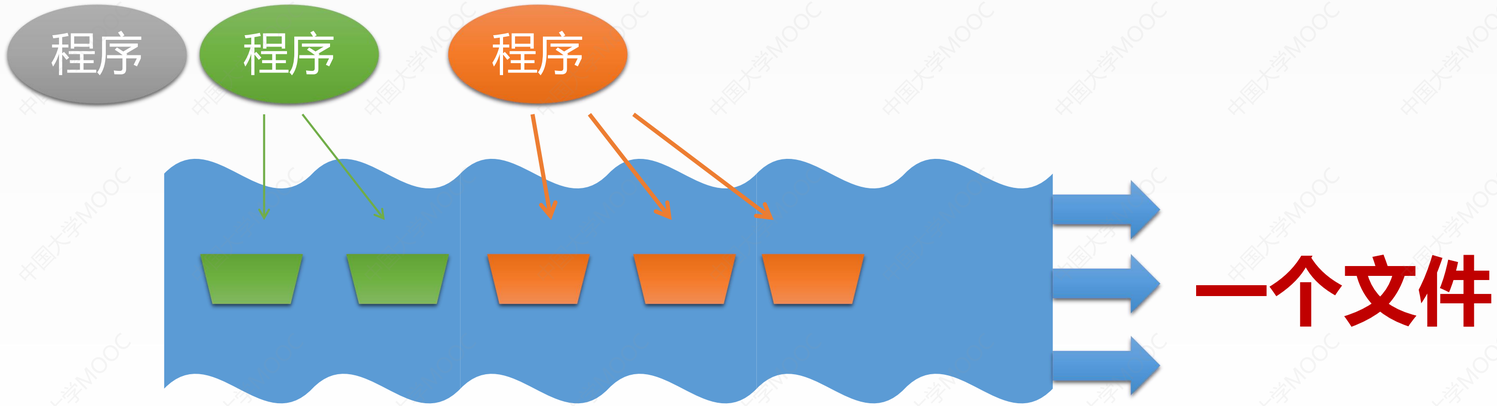
烤面包
牛肉
芝士
黄瓜
火腿
...

好吃的汉堡ლ(´ڡ`ლ)

- 程序 = 数据 + 指令
- 数据需要输入input和输出output

文件输入输出流

- 回顾：cin 和 cout 叫做标准输入输出流对象，其机制类似一条河流
- 事实上，文件IO和使用cin/cout没有本质区别，标准IO只是文件IO的一个特例
- **文件输入输出流**就是这条河流的来源/终点是一个文件



从标准IO到文件IO

- cout << “hello”；
- 现在我们知道cout是一个对象而<< 是运算符重载，含义实际上是 cout.插入(“hello”);
- cout 是标准输出流对象因此输出到控制台，如果换成**文件输出流对象**，就实现了往文件输出，输入流同理

标准输入输出流：

```
#include <iostream>

cin >> nVar;
cout << “hello”；
```

文件输入输出流：

```
#include <fstream>
std::ifstream fin;
std::ofstream fout;
// 文件打开，检测…
fin >> nVar;
fout << “hello”；
// 文件关闭…
```

file stream的意思

通过fin和fout读写文本文件
就相当于把其内容原封不动
放到控制台中用cin和cout读写

文件IO：打开 - 读/写 - 关闭

打开文件输出流并检测

```
#include <fstream>
// 省略...
std::ofstream fout;
fout.open("myFile.txt", std::ios::out);
if (!fout.is_open()) {
        return 0;
}
```

文件打开模式

模式标记	作用
ios::in	以输入模式打开已存在文件，不存在会报错
ios::out	以输出模式打开文件，不存在则新建，存在则清空
ios::app	以追加模式打开文件，不存在则新建，存在从尾部追写
ios::binary	以二进制模式打开文件，不用此标记则默认以文本模式打开

输出内容到文件

```
fout << “hello”;
```

关闭文件

```
fout.close();
```

文件IO：fstream

- 读文件要用ifstream对象，写文件要用ofstream对象，能不能就用一个流对象，又能读又能写？
- std::fstream

```
std::fstream fs;
fs.open("myFile.txt", std::ios::in | std::ios::out);
```

可以同时指定多个打开方式，用按位或｜连接

- 打开已存在的文件，既可读，也可写
- 若文件不存在则出错。
- 文件刚打开时，原有内容保持不变，随着写入可能覆盖原有内容

二进制文件

- 思考：通过文本文件存储整数123456789需要多少空间?
- 在文本文件中，将被存放成字符串“123456789”，共占9个字节
- 但它本来是个int，只占4字节呀！直接把这4字节写入文件就好了
- 二进制文件：将数据在内存中的实际存储内容直接写入到文件中，读取时**需要知道存储和理解的格式**
- 文本文件：将数据作为一串字符逐一输出到文件中，读取时再按照ASCII码逐字符解读为文本

可以尝试用记事本打开一个.mp3文件，或者.jpg文件，.ppt文件，.mp4文件...等

```
ofstream fout;
fout.open("data.bin", ios::out|ios::binary);
if (!fout.is_open()) { return 0; }
int a = 999999; char b = ‘a’ ; float c = 12.34f;
fout.write((char*)&a, sizeof(int));
fout.write((char*)&b, sizeof(char));
fout.write((char*)&c, sizeof(float));
```

```
ifstream fin;
fin.open("data.bin", ios::in|ios::binary);
if (!fin.is_open()) { return 0; }
int a; char b; float c;
fin.read((char*)&a, sizeof(int));
fin.read((char*)&b, sizeof(char));
fin.read((char*)&c, sizeof(float));
```

文件输入输出

流操作算子

输出格式控制

- 在C语言中，通过printf() / fprintf()函数的格式控制符控制输出的格式
- 而在C++中，文件被封装成了流对象，控制流对象的格式，就能控制输出的格式
- 通过往流对象中插入**流格式控制符**控制流对象的输出格式
- 如希望控制输出小数时保留小数点后两位：

```
#include <iostream>
#include <iomanip>
// 省略...
double pi = 3.1415926;
std::cout << std::setiosflags(std::ios::fixed) << std::setprecision(2) << pi;
```

manipulate: 控制，操纵

设置浮点数以固定的小数位数显示

设置精度(此时是小数点位数)为2

常用的流格式控制符

- 控制输出数据的宽度：**setw**(n)

```
std::cout << std::setw(8) << "C++" << std::setw(6)<<"101";
```

输出：
*<5个空格>*C++*<3个空格>*101

- 控制输出浮点数的精度：**setprecision**(n)

```
std::cout << std::setprecision(4) << 12.34567;
```

输出：12.35

```
std::cout << std::setiosflags(std::ios::fixed)
          << std::setprecision(4) << 12.34567;
```

输出：12.3457

- 控制输出整数的进制：**setbase**(n)，n∈{8, 10, 16}才有效

```
std::cout << std::setbase(16) << 15;
```

输出：f

- 控制左对齐右对齐：setiosflags(**ios::left**) / setiosflags(**ios::right**)

⭐ Key 1：文件在C++中以**流对象**的形式存在，通过流对象读写文件

⭐ Key 2：文件输入流：ifstream　文件输出流：ofstream　文件读写流: fstream

⭐ Key 3：流对象可以构造同时打开文件，也可以先定义，再调用open打开

1.要通过文件输入流对象myFile打开同目录下的xxk.dat文件用于文本输入时，正确的语句为：C

A. ofstream myFile(“xxk.dat”);

B. ifstream myFile; myFile.open(“xxk.dat”, ios::out);

C. ifstream myFile(“xxk.dat”, ios::in);

D. ifstream myFile; myFile.open(“xxk.dat”, ios::binary);

append: 追加

2.语句ofstream f(“SALARY.DAT”,ios::app); 的功能是建立流对象f并关联文件SALARY.DAT，并且：A

A.若文件存在，将文件写指针定位于文件尾；若文件不存在，建立一个新文件

B.若文件存在，将其置为空文件；若文件不存在，打开失败

C.若文件存在，将文件写指针定位于文件首；若文件不存在，建立一个新文件

D.若文件存在，打开失败；若文件不存在，建立一个新文件

⭐ Key 4：打开文件时要注意使用正确的相对路径或绝对路径

⭐ Key 5：操作流对象之前，要通过其 is_open 方法判断是否打开成功

1. 编译好的可执行文件为D:/bin/a.exe，若需要打开文件D:/file/data.txt进行**追写**，正确的语句是：D

A. ofstream fout(“data.txt”, ios::app);

B. ofstream fout; fout.open(D:/file/data.txt, ios::app);

C. ofstream fout(“file/data.txt”, ios::app);

D. ofstream fout; fout.open(“../../file/data.txt”, ios::app);

2. 若无法确定可执行程序是否有创建文件的权限，请补全下列代码使其鲁棒：

```
ofstream fout(“info.dat”, ios::out);
if (    fout.is_open()    ) {
    fout << “Information”;
}
```

分析：无法确定是否有创建文件的权限意味着有可能打开文件失败，因此需要对文件打开状态进行检查

end of file

⭐ Key 6：读文件时，通过流对象的 eof() 方法判断是否已经读到文件尾了

1. 文件test.txt的内容如下，则程序的输出为：D

A.
```
hello
eof world
end eof
```

B.
```
hello
```

C.
```
hello
eof
```

D.
```
hello
eof
world
end
eof
```

test.txt:

```
hello
eof world
end eof
```

```
std::ifstream fin("test.txt", ios::in);
if (!fin.is_open()) { return 0; }
string line;
while (!fin.eof()) {
        fin >> line;
        cout << line << endl;
}
fin.close();
```

⭐ Key 7：通过头文件<iomanip>中的流操作算子控制输出格式

1. 请写出下列程序的输出（如有空格，用'_'表示）：

```
int main() {
    double fVar = 12.3456789;
    cout << fVar << endl;
    cout << setprecision(3) << fVar << endl;
    cout << setiosflags(ios::fixed) << setprecision(2) << fVar << endl;
    cout << setfill('#') << setw(10) << fVar << endl;
    return 0;
}
```

12.3457
12.3
12.35
#####12.35

2. 填空题

（1）在C++中，控制输出内容所占宽度的流操作算子是__setw__，控制实际长度小于设置长度时用于填充的字符的流操作算子是__setfill__，默认输出的小数有效位数是__6__位。

（2）要对C++的输出格式进行控制，需要使用__<iomanip>__头文件定义的流操作算子。

⭐ Key 8：以二进制模式读写文件将直接拷贝实际存储内容，需要自行编码解码

1. 往文件中写入内容时的代码如下所示，请补全程序使其可以正确读取文件内容

```
ofstream fout;
fout.open("data.bin", ios::out | ios::binary);
if (!fout.is_open()) { return 0; }

char str[] = "Hello, world";
int l = sizeof(str);

fout.write((char*)&l, sizeof(l));
fout.write((char*)&str, sizeof(str));
fout.close();
```

```
ifstream fin;
fin.open("data.bin", ios::in | ios::binary );
if (!fin.is_open()) { return 0; }

int l_in;
fin.read((char*)&l_in, sizeof(l_in) );

char* str_in = new char[l_in];
fin.read(str_in, l_in );

fin.close();
```

sizeof(int)也可

⭐ Key 8：以二进制模式读写文件将直接拷贝实际存储内容，需要自行编码解码

2. 下面是通过二进制模式读写结构体数据的代码段，代码行及其描述对应正确的是：C

A. 行[3]的功能是打开文件student.dat

B. 行[5]中sizeof(s)指的是计算待写入数据的字符串长度

C. 行[b]中的ios::binary指明将通过二进制方式处理打开的文件

D. 行[e]中的&s是获得用于存放待读取数据的变量的引用

分析：A应为检测打开状态，B应为结构体所占空间大小，D应为获得指向s的指针

```
struct Student {
    int age;
    float score;
    char name[20];
};
```

```
[1]ofstream fout;
[2]fout.open("student.dat", ios::out|ios::binary);
[3]if (!fout.is_open()) { return 0; }

[4]Student s = {12, 88.5, "Mike"};
[5]fout.write((char*)&s, sizeof(s));
```

```
[a]ifstream fin;
[b]fin.open("student.dat", ios::in|ios::binary);
[c]if (!fin.is_open()) { return 0; }

[d]Student s;
[e]fin.read((char*)&s, sizeof(s));
```

⭐ Key 8：以二进制模式读写文件将直接拷贝实际存储内容，需要自行编码解码

3. 执行完左侧的代码段后，执行右侧代码段的输出是：B

（注：$1094795585_{(10)} = 41414141_{(16)} = 01000001,01000001,01000001,01000001_{(2)}$）

```
ofstream fout("data.bin", ios::out|ios::binary);
int n = 0x41414141;
fout.write((char*)&n, sizeof(n));
fout.close();
```

```
ifstream fin;
fin.open("data.bin", ios::in);
string s;
fin >> s;
cout << s;
```

中国大学MOOC
搜索：C++不挂科

A. 1094795585

B. AAAA

C. 01000001010000010100000101000001

D. 程序会报错，因为读写文件的方式不一致

分析：变量n实际内存内容就是01000001010000010100000101000001，二进制方式会将这4个字节直接写入文件，即文件实际存放的就是这4字节的内容。而右侧代码是通过文本方式读文件，会把这四个字节理解成一个一个的字符(文本)，即按ASCII码进行解码，得到的结果为字符串“AAAA”，因为字符'A'的ASCII码为$65_{(10)} = 01000001_{(2)}$