

程序设计实习

C++ 面向对象程序设计

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

2025 年 4 月 9 日

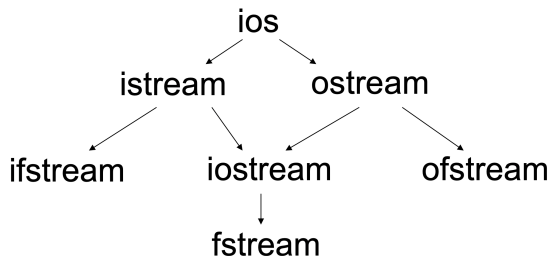
大纲

① 输入输出相关的类

② 流操纵算子

③ 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。
- `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
`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。
- 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，也可以被重定向为向文件写入数据。
 - `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。

判断输入流结束

```
1  int x;  
2  while (cin>>x) {  
3      //  
4  }
```

原理?

判断输入流结束

```
1  int x;  
2  while (cin>>x) {  
3      //  
4  }
```

原理?

```
1  istream& operator>>(int &a) {  
2      //  
3      return *this;  
4  }
```

判断输入流结束

```
1  int x;  
2  while (cin>>x) {  
3      //  
4  }
```

原理?

```
1  istream& operator>>(int &a) {  
2      //  
3      return *this;  
4  }
```

```
1  operator bool() {  
2      //  
3  }
```

istream 类的成员函数

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

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）。

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

从输入流中读取bufSize - 1个字符到缓冲区buf，或读到碰到delim字符为止（哪个先到算哪个）。

两个函数都会自动在 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之前的字符个数达到或超过了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

istream 类的成员函数

```
1  bool eof();    //判断输入流是否结束
2  int peek();    //返回下一个字符，但不从流中去掉.
3  istream &putback(char ch); //将字符 ch 放回输入流
4  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 <istream>
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 1

12 abcd

输出 1

abcd

输入 2

12

输出 2

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 }
```

输入重定向

```
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
```

输入重定向

```
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：流操纵算子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

```
1  int n = 10;  
2  cout << n << endl;  
3  cout << hex << n << endl  
4      << dec << n << endl  
5      << oct << n << endl;
```

输出结果：

```
10  
a  
10  
12
```

控制浮点数精度的流操纵算子

precision, setprecision

- precision是成员函数，其调用方式为：

```
1 cout.precision(5);
```

- setprecision 是流操作算子，其调用方式为：

```
1 cout << setprecision(5); // 可以连续输出
```

它们的功能相同。

- 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

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int main() {
5      double x = 1234567.89, y = 12.34567;
6      int n = 1234567;
7      int m = 12;
8      cout << setprecision(6) << x << endl
9           << y << endl
10          << n << endl
11          << m << endl;
12 }
```

输出:

```
1.23457e+06
12.3457
1234567
12
```

控制浮点数精度的流操纵算子

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int main() {
5      double x = 1234567.89, y = 12.34567;
6      int n = 1234567;
7      int m = 12;
8      cout << setiosflags(ios::fixed)
9           << setprecision(6) << x << endl
10          << y << endl
11          << n << endl
12          << m << endl;
13 }
```

输出:

```
1234567.890000
12.345670
1234567
12
```

控制浮点数精度的流操纵算子

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4  int main() {
5      double x = 1234567.89;
6      cout << setiosflags(ios::fixed)
7           << setprecision(6) << x << endl
8           << resetiosflags(ios::fixed) << x << endl;
9  }
```

输出:

```
1234567.890000
1.23457e+06
```

`resetiosflags(ios::fixed)`取消以小数点位置固定的方式输出

设置域宽的流操纵算子

设置域宽 (setw, width) 两者功能相同，一个是成员函数，另一个是流操作算子，调用方式不同：

```
1  cin >> setw(4); //  cin.width(4);  
2  cout << setw(4); //  cout.width(4);
```

设置域宽的流操纵算子

设置域宽 (setw, width) 两者功能相同，一个是成员函数，另一个是流操作算子，调用方式不同：

```
1 cin >> setw(4); // cin.width(4);
2 cout << setw(4); // cout.width(4);
```

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int w = 4;
6     char string[10];
7     cin.width(5);
8     while (cin >> string) {
9         cout.width(w++);
10        cout << string << endl;
11        cin.width(5);
12    }
13 }
```

输入：

1234567890

输出：

1234
5678
90

设置域宽的流操纵算子

设置域宽 (setw, width) 两者功能相同，一个是成员函数，另一个是流操作算子，调用方式不同：

```
1  cin >> setw(4); //  cin.width(4);
2  cout << setw(4); //  cout.width(4);
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int w = 4;
6      char string[10];
7      cin.width(5);
8      while (cin >> string) {
9          cout.width(w++);
10         cout << string << endl;
11         cin.width(5);
12     }
13 }
```

输入：

1234567890

输出：

1234
5678
90

宽度设置有效性是一次性的，在每次读入和输出之前都要设置宽度。

流操纵算子

```
5  int main() {
6      int n = 141;
7      //1) 分别以十六进制、十进制、八进制先后输出 n
8      cout << "1) " << hex << n << " " << dec << n << " " << oct << n << endl;
9      double x = 1234567.89, y = 12.34567;
10     //2) 保留 5 位有效数字
11     cout << "2) " << setprecision(5) << x << " " << y << " " << endl;
12     //3) 保留小数点后面 5 位
13     cout << "3) " << fixed << setprecision(5) << x << " " << y << endl ;
14     //4) 科学计数法输出, 且保留小数点后面 5 位
15     cout << "4) " << scientific << setprecision(5) << x << " " << y << endl ;
16     //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
17     cout << "5) " << showpos << fixed << setw(12) << setfill('*') << 12.1 << endl;
18     //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
19     cout << "6) " << noshowpos << setw(12) << left << 12.1 << endl;
20     //7) 输出宽度为 12 字符, 宽度不足则左边用填充字符填充
21     cout << "7) " << setw(12) << right << 12.1 << endl;
22     //8) 宽度不足时, 负号和数值分列左右, 中间用填充字符填充
23     cout << "8) " << setw(12) << internal << -12.1 << endl;
24     cout << "9) " << 12.1 << endl;
25     return 0;
26 }
```

输出

```
1) 8d 141 215
2) 1.2346e+06 12.346
3) 1234567.89000 12.34567
4) 1.23457e+06 1.23457e+01
5) ***+12.10000
6) 12.10000****
7) ****12.10000
8) -***12.10000
9) 12.10000
```


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
```

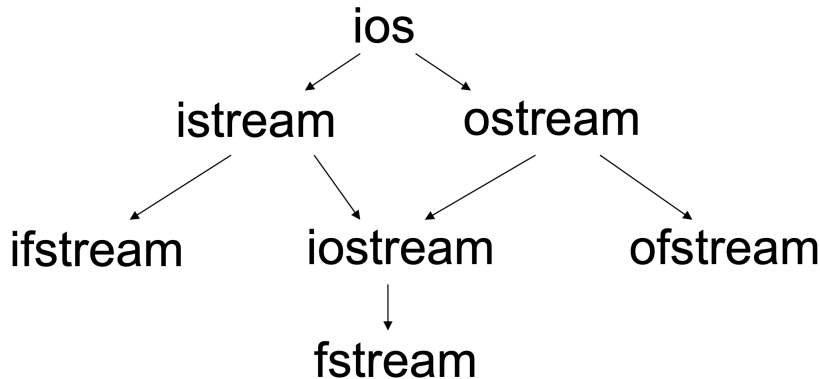
为什么可以？ 因为 `iostream` 里对 `<<` 进行了重载 (成员函数)

```
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

文件和流

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：



创建文件

```
1 #include <fstream> // 包含头文件
2
3 ofstream outFile("clients.dat", ios::out | ios::binary); //创建文件
```

clients.dat 要创建的文件的名称，文件名可以给出绝对路径，也可以给相对路径
ios::out 文件打开方式

- 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 (二进制跟普通的文本格式最大差别在于对换行符的处理方式不同)

创建文件

也可以先创建ofstream对象，再用 open函数打开

```
1  ofstream fout;  
2  fout.open("test.out", ios::out | ios::binary);
```

判断打开是否成功：

```
1  if (!fout) {  
2      cout << "File open error!" << endl;  
3  }
```

文件名的绝对路径和相对路径

- 绝对路径:

- windows:

```
"c:\\tmp\\mydir\\some.txt"
```

- linux\ macos:

```
"/tmp/same.txt"
```

- 相对路径:

- windows:

```
"\\tmp\\mydir\\some.txt" //当前盘符的根目录下的 tmp\\dir\\some.txt  
"tmp\\mydir\\some.txt" //当前文件夹的 tmp 子文件夹里面的.....  
"..\\tmp\\mydir\\some.txt" //当前文件夹的父文件夹下面的 tmp 子文件夹里面的
```

- linux\ macos:

```
"tmp/same.txt"  
"..tmp/mydir/some.txt"
```

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。

```
1  ofstream fout("a1.out", ios::app); //以添加方式打开
2  long location = fout.tellp();      //取得写指针的位置
3  location = 10;
4  fout.seekp(location);              // 将写指针移动到第 10 个字节处
5  fout.seekp(location, ios::beg);    //从头数 location
6  fout.seekp(location, ios::cur);    //从当前位置数 location
7  fout.seekp(location, ios::end);    //从尾部数 location
```

`location` 可以为负值

显式关闭文件

```
1 ifstream fin("test.dat", ios::in);  
2 fin.close();  
3  
4 ofstream fout("test.dat", ios::out);  
5 fout.close();
```


字符文件读写

因为文件流也是流，所以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
```

字符文件读写

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <algorithm>
5  using namespace std;
6  int main() {
7      vector<int> v; // v 是一个可变长整型数组
8      ifstream srcFile("in.txt", ios::in);
9      ofstream destFile("out.txt", ios::out);
10     int x;
11     while (srcFile >> x)
12         v.push_back(x); //往 v 末尾添加一个元素
13     sort(v.begin(), v.end()); //排序
14     for (int i = 0; i < v.size(); i++)
15         destFile << v[i] << " ";
16     destFile.close();
17     srcFile.close();
18     return 0;
19 }
```

二进制文件读写

- 二进制读文件：

```
1 istream &read(char *s, long n);
```

将文件读指针指向的地方的 n 个字节内容，读入到内存地址 s ，然后将文件读指针向后移动 n 字节 (以 `ios::in` 方式打开文件时，文件读指针开始指向文件开头)。

二进制文件读写

- 二进制读文件：

```
1 istream &read(char *s, long n);
```

将文件读指针指向的地方的 n 个字节内容，读入到内存地址 s ，然后将文件读指针向后移动 n 字节 (以 `ios::in` 方式打开文件时，文件读指针开始指向文件开头)。

- 二进制写文件：

```
1 ostream &write(const char *s, long n);
```

将内存地址 s 处的 n 个字节内容，写入到文件中写指针指向的位置，然后将文件写指针向后移动 n 字节 (以 `ios::out` 方式打开文件时，文件写指针开始指向文件开头，以 `ios::app` 方式打开文件时，文件写指针开始指向文件尾部)。

在文件中写入和读取一个整数

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6      ofstream fout("some.dat", ios::out | ios::binary);
7      int x = 120;
8      fout.write((const char *)&x, sizeof(int));
9      fout.close();
10     ifstream fin("some.dat", ios::in | ios::binary);
11     int y;
12     fin.read((char *)&y, sizeof(int));
13     fin.close();
14     cout << y << endl;
15     return 0;
16 }
```

二进制文件读写

从键盘输入几个学生的姓名的成绩，并以二进制文件形式保存

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  struct Student {
6      char name[20];
7      int score;
8  };
9  int main() {
10     Student s;
11     ofstream OutFile("students.dat", ios::out | ios::binary);
12     while (cin >> s.name >> s.score)
13         OutFile.write((char *)&s, sizeof(s));
14     OutFile.close();
15     return 0;
16 }
```

二进制文件读写

输入:

```
Tom 60  
Jack 80  
Jane 40
```

则形成的 students.dat 为 72 字节, 用 windows 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 藪 Jane 烫烫烫烫烫烫烫烫?

二进制文件读写

将 students.dat 文件的内容读出并显示

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  struct Student {
5      char name[20];
6      int score;
7  };
8  int main() {
9      Student s;
10     ifstream inFile("students.dat", ios::in | ios::binary);
11     if (!inFile) {
12         cout << "error" << endl;
13         return 0;
14     }
15     while (inFile.read((char *)&s, sizeof(s))) {
16         int readedBytes = inFile.gcount(); //看刚才读了多少字节
17         cout << s.name << " " << s.score << endl;
18     }
19     inFile.close();
20     return 0;
21 }
```


二进制文件读写

将 students.dat 文件的 Jane 的名字改成 Mike

```
5  struct Student {
6      char name[20];
7      int score;
8  };
9  int main() {
10     Student s;
11     fstream iofile("students.dat", ios::in | ios::out | ios::binary);
12     if (!iofile) {
13         cout << "error";
14         return 0;
15     }
16     iofile.seekp(2 * sizeof(s), ios::beg); //定位写指针到第三个记录
17     iofile.write("Mike", strlen("Mike"));
18     iofile.seekg(0, ios::beg); //定位读指针到开头
19     while (iofile.read((char *)&s, sizeof(s)))
20         cout << s.name << " " << s.score << endl;
21     iofile.close();
22     return 0;
23 }
```

文件拷贝程序 mycopy 示例

```
1  /* 用法示例:
2     mycopy src.dat dest.dat
3     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
4  */
5  #include <iostream>
6  #include <fstream>
7  using namespace std;
8  int main(int argc, char *argv[]) {
9      if (argc != 3) {
10         cout << "File name missing!" << endl;
11         return 0;
12     }
13     ifstream inFile(argv[1], ios::binary | ios::in); //打开文件用于读
14     if (!inFile) {
15         cout << "Source file open error." << endl;
16         return 0;
17     }
18     ofstream outFile(argv[2], ios::binary | ios::out); //打开文件用于写
19     if (!outFile) {
20         cout << "New file open error." << endl;
21         inFile.close(); //打开的文件一定要关闭
22         return 0;
23     }
24     char c;
25     while (inFile.get(c)) //每次读取一个字符
26         outFile.put(c);    //每次写入一个字符
27     outFile.close();
28     inFile.close();
29     return 0;
30 }
```