17 输入、输出和文件

引言

- 了解如何使用输入和输出是每个学习计算机语言的人面临的首要任务
- C++使用了很多较为高级的语言特性来实现输入和输出，其中包括类、派生类、函数重载、虚函数、模板和多重继承

C++输入和输出概述

1 C++输入和输出概述

- C，C++没有将输入输出建立在语言中
- C语言最初把I/O留给了编译器实现人员。多数实现人员都把I/O建立在最初为UNIX环境开发的库函数的基础之上。 ANSI C正式承认这个I/O软件包时，将其称为标准输入输出库
  - stdio.h
- C++依赖自己的I/O解决方案
  - iostream, fstream
  - ANSI/ISO C++委员会决定把这个类正式作为一个标准类库，并添加其他一些标准类

输入输出流示例

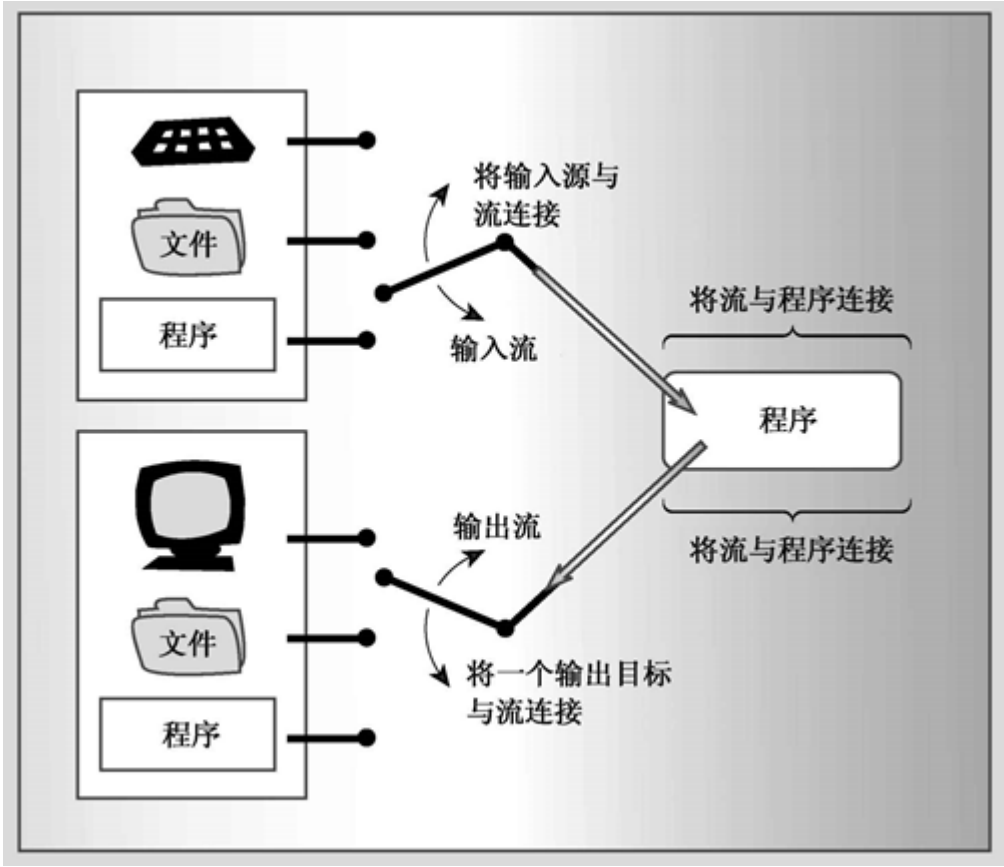
```
#include <iostream>
#include <fstream>
#include <string>
int main(){
    using namespace std;
    string filename;
    cin >> filename;

    ofstream fout(filename.c_str());
    // write to file
    fout << "For your eyes only!\n";
    // write to screen
    cout << "Enter your secret number: ";
    float secret;
    cin >> secret;
    fout << "Your " << secret << endl;
    fout.close(); // close file
// create input stream object for new file and call
   it fin

    ifstream fin(filename.c_str());
    cout << "Here " << filename << ":\n";
    char ch;
    while (fin.get(ch)) // read character
        cout << ch; // write it to screen
    cout << "Done\n";
    fin.close();
    // std::cin.get();
    // std::cin.get();
    return 0;
}
```

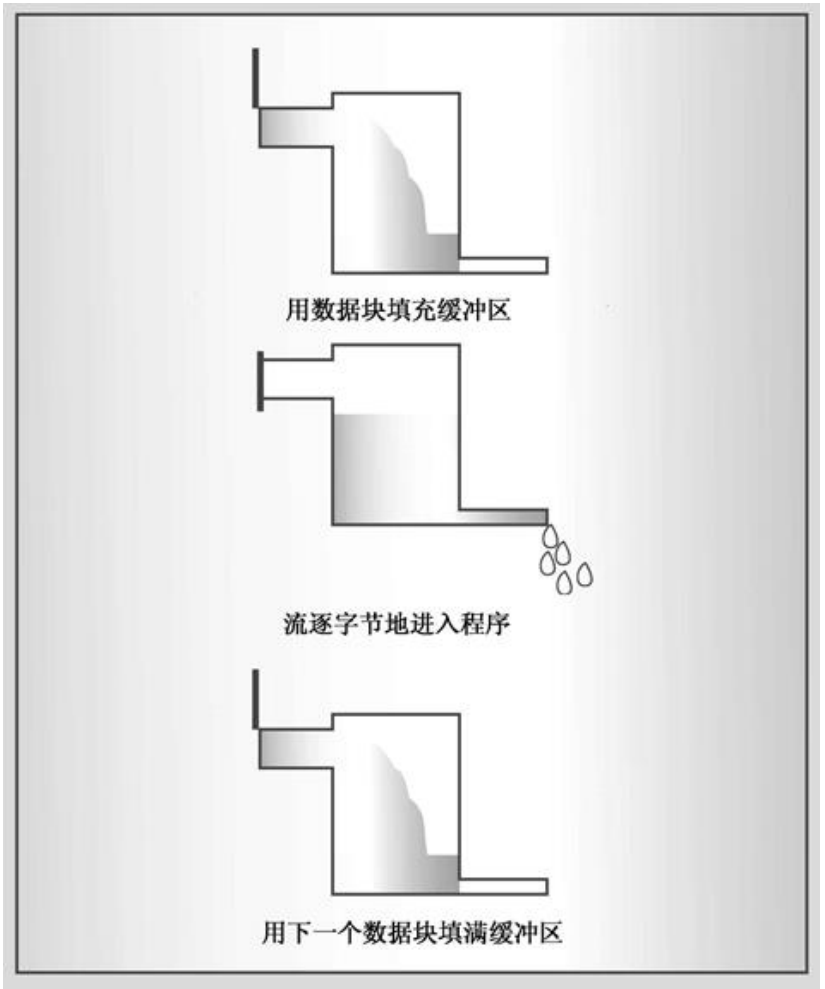
1.1 流和缓冲区

- C++把输入和输出看做字节流
  - 输入时，程序从输入流中抽取字节
  - 输出时，程序将字节插入到输出流中
  - 流充当了程序和流源或流目标之间的桥梁
    - 使得 C++程序可以以相同的方式，对待来自键盘的输入和来自文件的输入
- 通过使用流，C++程序处理输出的方式独立于其去向。管理输入包含两步：
  - 将流与输入去向的程序关联起来
  - 将流与文件连接起来
  - 换句话说，输入流需要两个连接，每端各一个



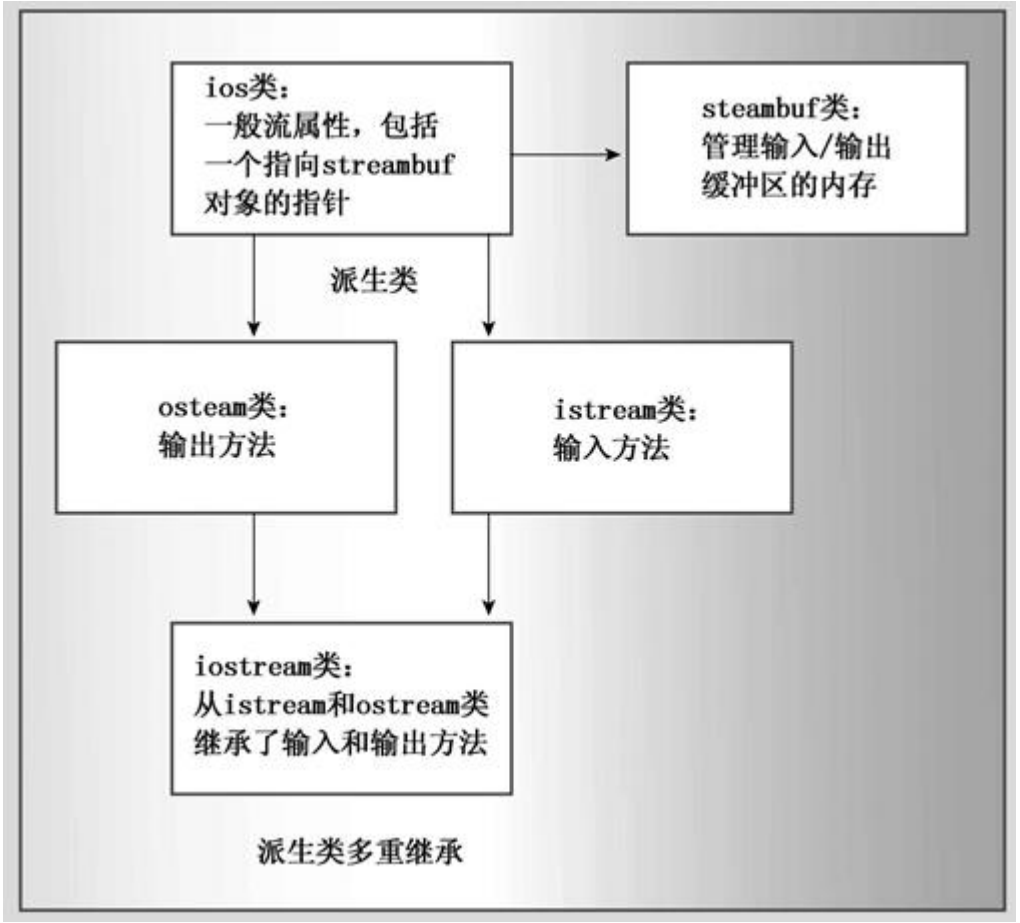
1.1 流和缓冲区

- 通过使用缓冲区可以更高效地处理输入和输出。缓冲区是用作中介的内存块
  - 它是将信息从设备传输到程序或从程序传输给设备的临时存储工具。



1.2 流、缓冲区和iostream文件

- iostream文件中包含用来实现管理流和缓冲区的类
  - streambuf类为缓冲区提供了内存等的类方法
  - ios_base类表示流的一般特征，如是否可读取、是二进制流还是文本流等
  - ios类基于ios_base，其中包括了一个指向streambuf对象的指针成员
  - ostream类从ios类派生而来的，提供了输出方法
  - istream类从ios类派生而来的，提供了输入方法
  - iostream类基于istream和ostream类，继承了输入方法和输出方法
- cin，cout，cerr，clog，对象代表流



1.3 重定向

- 标准输入和输出流通常连接键盘和屏幕，但也可以重定向
  - 使得能够改变标准输入和标准输出
- 示例：通过重定向可以从文件中读取，向文件输入

```
C>counter
Hello
and goodbye!
Control-Z << simulated end-of-file
Input contained 19 characters.
C>

cow_cnt file:
C>counter <oklahoma >cow_cnt
C>
```

使用cout进行输出

2 使用cout进行输出

- ostream类将数据内部表示转换为由字符字节组成的输出流
  - 要在屏幕上显示数字-2.34，需要将5个字符(-2.3和4)，而不是64位内部浮点表示发送到屏幕上

2.1 重载的<<运算符

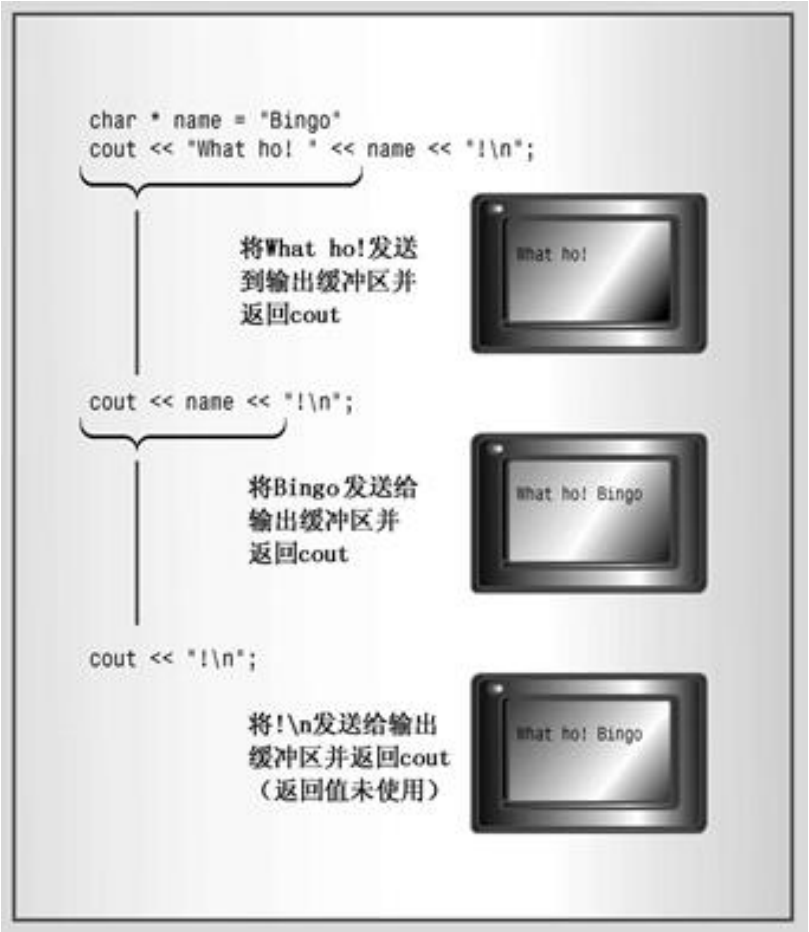
- ostream类重新定义<<运算符，<<称为插入运算符
  - 重载<<被重载，使之能识别C++中所有基本类型，包括指针类型
  - cout << value;

- 拼接输出
  - <<所有化身返回类型都是ostream&，该特性支持连续输出
  - `cout << "We have" << cout << "unhatched chickens.\n";`

2.2 其他ostream方法

P17.1 write.cpp

- ostream 类还提供
  - put()，显示字符
  - write()，显示字符串
    - write()方法并不会在遇到空字符时自动停止打印字符，而只是打印指定数目的字符，即使超出了字符串的边界
    - 如果传递的是普通类型数据的地址，则输出地址



2.3 刷新输出缓冲区

- ostream类对cout对象处理的输出进行缓冲
  - 输出不会立即发送到目标地址，而是被存储在缓冲区中，直到缓冲区填满
  - 输出到文件，缓冲节省大量时间，因为文件读写效率低
  - 输出到屏幕，响应可能不及时
- 如果实现不能在所希望时刷新输出，可以使用两个控制符中的一个来强行进行刷新
  - cin会引发缓冲区的刷新
  - flush
  - endl

2.4 用cout进行格式化

- ostream 插入运算符将值转换为文本格式
  - 修改显示时的计数系统dec，hex，oct：即以什么进制显示
  - 调整字段宽度width()：将长度不同的数字放到宽度相同的字段中，
  - 填充字符fill()：用什么符号填充字段中未被使用的部分
  - 设置浮点数的显示精度precision()
    - 在默认模式下，它指的是显示的总位数。在定点模式和科学模式下(稍后将讨论)，精度指的是小数点后面的位数。
  - 打印末尾的0和小数点setf()
  - 标准控制符
  - 头文件iomanip
    - setprecision(), setfill(), setw()

P17.2 defaults.cpp P17.3 manip.cpp P17.4 width.cpp P17.5 fill.cpp P17.6 precise.cpp

P17.7 showpt.cpp P17.8 setf.cpp P17.9 setf2.cpp P17.10 iomanip.cpp

使用cin进行输入

3 使用cin进行输入

- cin对象将标准输入表示为字节流
  - 通常情况下，通过键盘来生成这种字符流

```
cin >> value_holder;
istream &operator>>(int &);
```

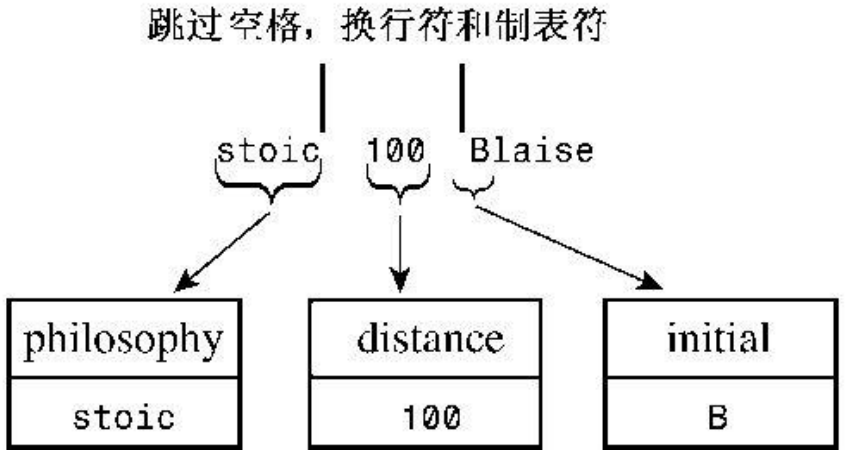
- istream类重载抽取运算符>>
  - 参数和返回值都是引用(直接改变参数变量值，连续输入)

3.1 cin>>如何检查输入

- 抽取运算符查看输入流的方法
  - 跳过空白(空格、换行符和制表符)，直到遇到非空白字符，即使单字符模式
- 其他模式下，＞＞运算符读取一个指定类型的数据。即它读取，从非空白字符开始到与目标类型不匹配的第一个字符之间，的全部内容
  - cin >> elevation;
  - 键入-123Z
  - 运算符将读取字符-12和3，Z留在输入流中，下一个 cin 语句将从这里开始
  - 假设输入是 Zcar。在这种情况下，抽取运算符将不会修改elevation的值，并返回0

```
char philosophy[20];
int distance;
char initial;

cin >> philosophy >> distance >> initial;
```



3.2 流状态

P17.12 cinexcp.cpp

- cin或cout对象包含一个描述流状态的数据成员
  - 流状态由eofbit，badbit，failbit组成
  - 程序可以检查流状态，并使用这种信息来决定下一步做什么
    - 如是否到达文件末尾

流状态

成 员	描 述
eofbit	如果到达文件尾，则设置为1
badbit	如果流被破坏，则设置为1；例如，文件读取错误
failbit	如果输入操作未能读取预期的字符或输出操作没有写入预期的字符，则设置为1
goodbit	另一种表示0的方法
good( )	如果流可以使用(所有的位都被清除)，则返回true
eof( )	如果eofbit被设置，则返回true
bad( )	如果badbit被设置，则返回true
fail( )	如果badbit或failbit被设置，则返回true
rdstate( )	返回流状态
exceptions( )	返回一个位掩码，指出哪些标记导致异常被引发
exceptions(isostate ex)	设置哪些状态将导致clear( )引发异常；例如，如果ex是eofbit，则如果eofbit被设置，clear( )将引发异常
clear(iostate s)	将流状态设置为s；s的默认值为0(goodbit)；如果(restate( )& exceptions( ))! = 0，则引发异常basic_ios::failure
setstate(iostate s)	调用clear(rdstate( )｜ s)。这将设置与s中设置的位对应的流状态位，其他流状态位保持不变

I/O 和异常

- 只有在流状态良好(所有的位都被清除)的情况下，下面的测试才返回 true:

```
while (cin >> input)
```

3.3 其他istream类方法

P17.13 get_gun.cpp

- 方法get(char&)和get(void)提供不跳过空白的单字符输入功能
  - 即使该字符是空格、制表符或换行符
  - cin»ch；会跳过空格，换行符等
- 函数get(char*, int, char)和getline(char*, int, char)在默认情况下读取整行而不是一个单词
- istream 类的这两组成员函数
  - 单字符输入
    - 是否希望跳过空白
    - 如果希望程序检查每个字符
  - 字符串输入： getline() 、get()和ignore()
    - get()和getline()之间的主要区别在千， get()将换行符留在输入流中，这样接下来的输入操作首先看到是将是换行符，而 getline()抽取并丢弃输入流中的换行符。

3.4 其他istream方法

- 其他istream方法包括read() 、peek() 、gcount()和putback()
- read()常与ostream,write()结合使用
- peek()返回输入中下一个字符
- gcount()返回最后一个非格式化方法读取的字符数
- putback()将一个字符插入到输入字符串

P17.14 peeker.cpp P17.15 truncate.cpp

文件输入和输出

4 文件输入和输出

- 文件输入ifstream
- 文件输出ofstream
- fstream同步文件I/O

4.1 简单的文件I/O

- 写入文件
  - 创建ofstream对象
  - 将对象与文件关联
  - 以cout方式使用对象
- 读取文件
  - 创建ifstream对象
  - 将对象与文件关联
  - 以cin方式使用对象

P17.16 fileio.cpp

4.2 流状态检查和is_open()

- C++文件流从ios_base类继承一个流状态成员以及报告流状态方法
- 检查文件是否被打开：is_open()

```
fin.open(argv[file]);
if (fin.fail()){} // open attempt failed
if (!fin){} // open attempt failed
if (!fin.is_open()){} // open attempt failed, 推荐!
```

4.3 打开多个文件

- 同时打开多个文件，为每个文件创建一个流
- 依次处理一组文件，打开一个流，依次关联到各个文件

4.3 命令行处理技术

- 文件处理通常使用命令行参数指定文件

```
int main(int argc, char *argv[])
```

  - argc为命令行中的参数个数
  - argv是指向一个指向char的指针的指针

17.17 count.cpp

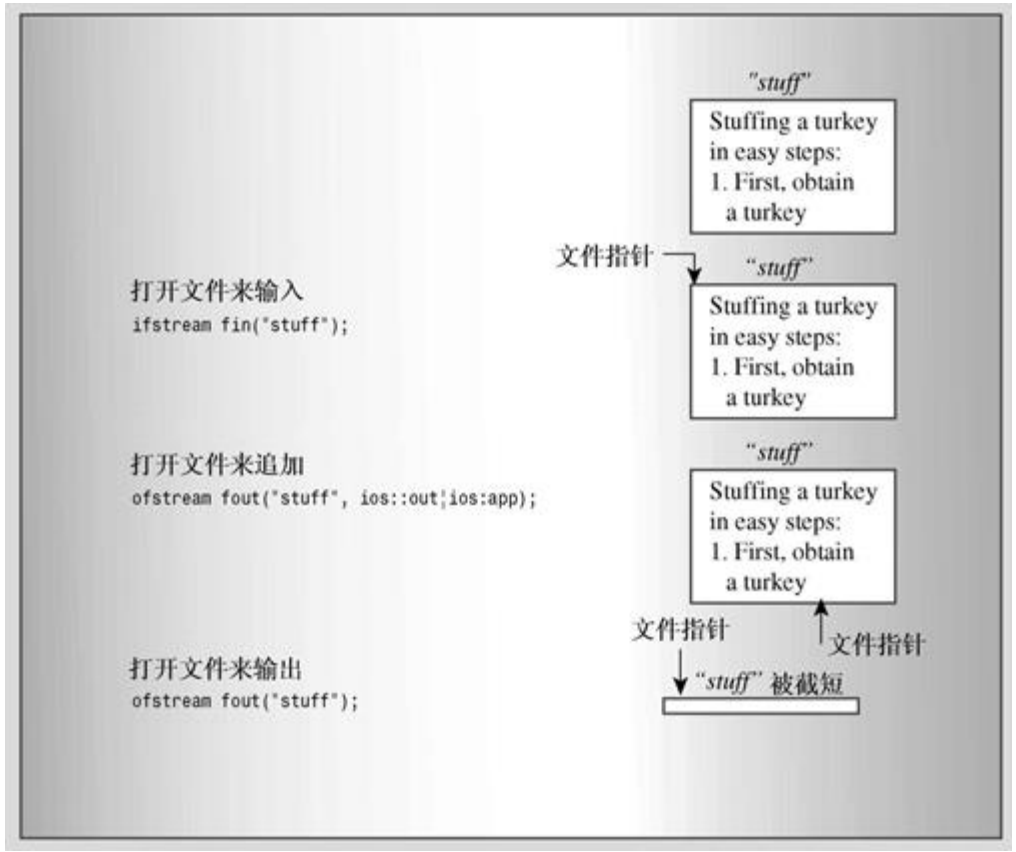
4.5 文件模式

- 文件模式描述文件如何被使用：读，写，追加等

P17.18 append.cpp P17.19 binary.cpp

常 量	含 义
ios_base::in	打开文件，以便读取
ios_base::out	打开文件，以便写入
ios_base::ate	打开文件，并移到文件尾
ios_base::app	追加到文件尾
ios_base::trunc	如果文件存在，则截短文件
ios_base::binary	二进制文件

文件模式



C++模式	C模式	含 义
ios_base :: in	"r"	打开以读取
ios_base :: out	"w"	等价于ios_base :: out | ios_base :: trunc
ios_base :: out | ios_base :: trunc	"w"	打开以写入，如果已经存在，则截短文件
ios_base :: out | ios_base :: app	"a"	打开以写入，只追加
ios_base :: out | ios_base :: out	"r+"	打开以读写，在文件允许的位置写入
ios_base :: out | ios_base :: out | ios_base::trunc	"w+"	打开以读写，如果已经存在，则首先截短文件
c++mode | ios_base :: binary	"cmodeb"	以C++mode(或相应的cmode)和二进制模式打开；例如，ios_base :: in | ios_base :: binary成为“rb”
c++mode | ios_base :: ate	"cmode"	以指定的模式打开，并移到文件尾。C使用一个独立的函数调用，而不是模式编码。例如，ios_base :: in | ios_base :: ate被转换为“r”模式和C函数调用fseek(file, 0, SEEK_END)

4.6 随机存取

- 随机存取指直接移动到文件的任何位置，而不是依次
- seekg()，将输入指针移到指定文件(缓冲区)位置
- seekp()，将输出指针移到指定文件(缓冲区)位置

P17.20 randorn.cpp

内核格式化

5 内核格式化

- sstream提供程序和string对象之间的I/O
  - 可以使用于cout 的 ostream 方法将格式化信息写入到string对象中
  - 用istream::getline()来读取string对象中的信息
- 内核格式化(incore formatting)
  - 读取string对象中的格式化信息，或将格式化信息写入string 对象中
- istringstream, ostringstream

P17.21 strout.cpp　P17.22 strin.cpp

```
1.  #include <iostream>
2.  #include <sstream>
3.  #include <string>
4.  int main(){
5.      using namespace std;
6.      ostringstream outstr; // manages a string stream
7.      string hdisk;
8.      getline(cin, hdisk);
9.      int cap;
10.     cin >> cap;
11.     // write formatted information to string stream
12.     outstr << “H: ” << hdisk << “, C:” << cap;
13.     string result = outstr.str(); // save result
14.     cout << result;               // show contents
15.     return 0;
16. }
```

6 总结

- 流是进出程序的字节流
- I/O类库提供了大量有用的方法
- 使用 ios_base类方法以及文件iostream和iomanip中定义的控制符(可与插入运算符拼接的函数)，可以控制程序如何格式化输出
- fstream文件提供了将iostream方法扩展到文件I/O的类定义
- 文本文件以字符格式存储所有的信息。二进制文件使用计算机内部使用的二进制表示来存储信息
- sstream头文件定义了istringstream和ostringstream类，这些类使得能够使用istream和ostream方法来抽取字符串中的信息，并对要放入到字符串中的信息进行格式化