



问题求解与实践 ——输入输出与文件

主讲教师： 陈雨亭、沈艳艳



C++语言



输入输出、
错误处理



数据结构



现代C++

STL/容器
的设计



Boost



FLTK



OneAPI

搜索

贪心算法
遗传算法
动态规划

AI

深度学习
神经网络

例子：计算器、
数值计算、树
图同构



例子：用FLTK改
装计算器、用
OneAPI异构计算



例子：多项式插值、
傅里叶变换、马踏
棋盘、计划安排等

问题

- 什么叫“流”？什么叫“流对象”？
- 如何利用流，支持程序输入输出？

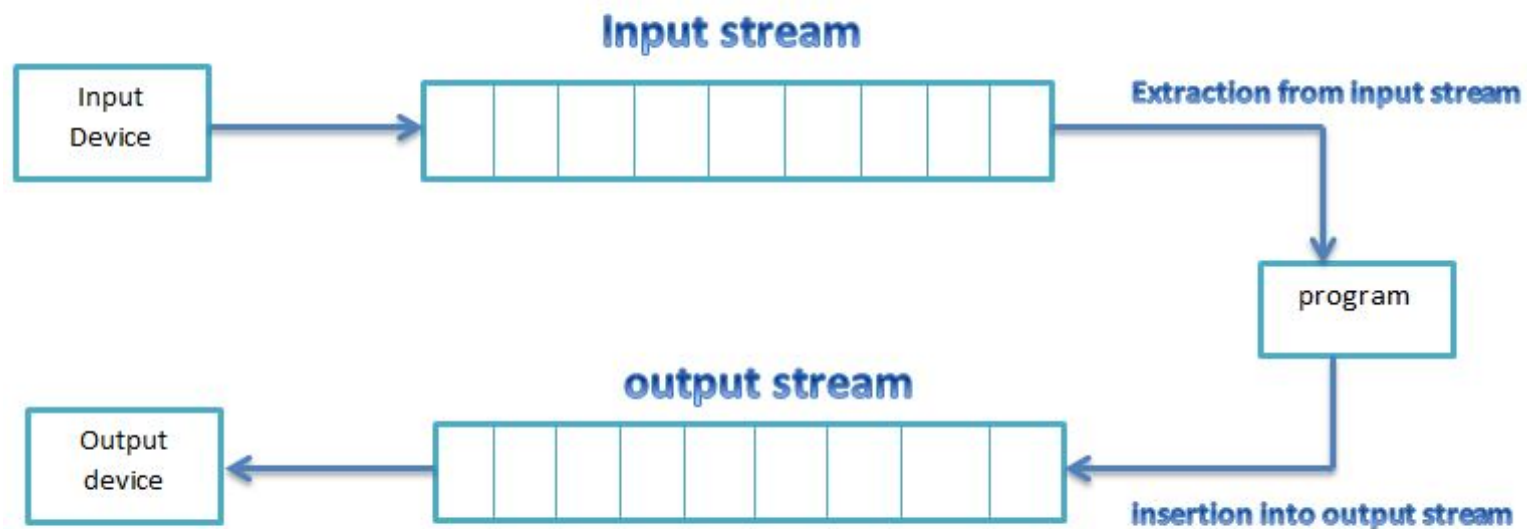


Fig: data streams

输入输出与文件



- 输入输出是指程序与外部设备交换信息
- C++把输入输出看成是一个数据流
 - 输入流：外围设备流向内存的数据
 - 输出流：内存流向外围设备的数据
- 在C++中，输入输出不是语言所定义的部分，而是由标准库提供。
- C++的输入输出分为：
 - 基于控制台的I/O
 - 基于字符串的I/O
 - 基于文件的I/O

```
cin >>
```

```
cout <<
```

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- 基于字符串的I/O
- 基于文件的I/O

流的概念及用途



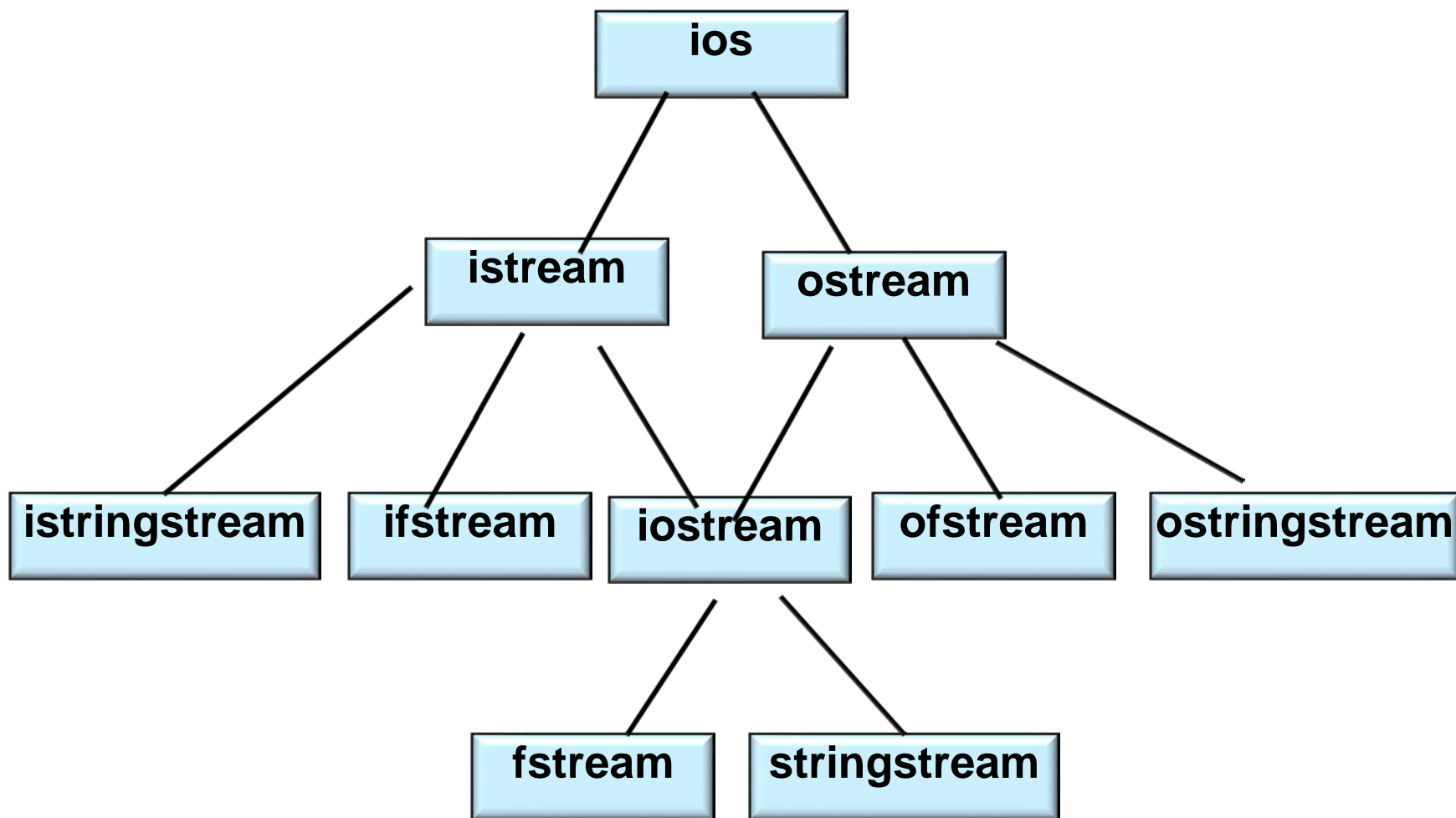
- I/O操作是以对数据类型敏感的方式执行的。C++的I/O操作是以字节流的形式实现的。流实际上就是字节序列。
- C++提供了低级和高级I/O功能
 - 低级I/O功能通常只在设备和内存之间传输一些字节。
 - 高级I/O功能把若干个字节组合成有意义的单位，如整数、浮点数、字符、字符串以及用户自定义类型的数据。
- C++提供了无格式I/O和格式化I/O两种操作
 - 无格式I/O传输速度快（read函数或write函数），但使用起来较为麻烦
 - 格式化I/O按不同的类型对数据进行处理，但需要增加额外的处理时间，不适于处理大容量的数据传输。

流与标准库



头文件	类型
iostream	istream从流中读取 ostream写到流中去 iostream对流进行读写，从istream和ostream派生
fstream	ifstream从文件中读取，由istream派生而来 ofstream写到文件中去，由ostream派生而来 fstream对流进行读写，由iostream派生而来
sstream	istringstream从string对象中读取，由istream派生而来 ostringstream写到string对象中去，由ostream派生而来 stringstream对string对象进行读写，由iostream派生而来

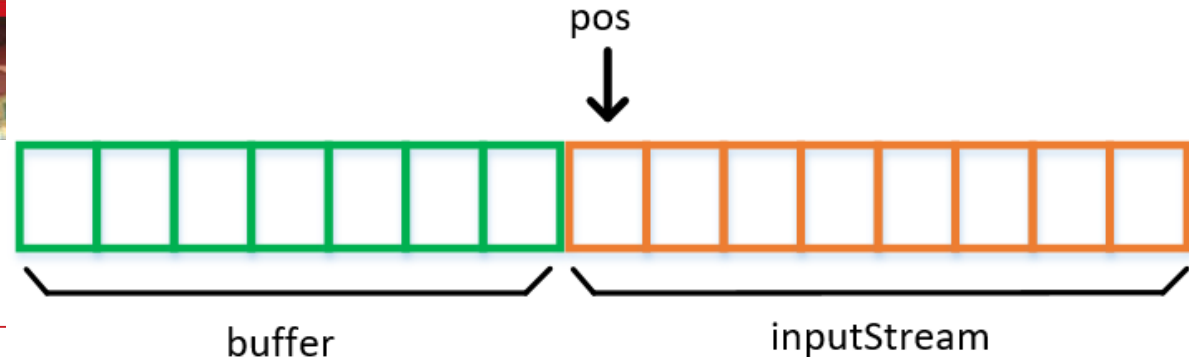
类的继承关系



如何设计一个具体的ios对象？



输入输出缓冲



- C++的输入输出是基于**缓冲**实现的
- 每个I/O对象管理一个缓冲区，用于存储程序读写的数据
 - 当用户在键盘上输入数据时，键盘输入的数据是存储在输入缓冲区中
 - 当执行“>>”操作时，从输入缓冲区中取数据存入变量，如缓冲区中无数据，则等待从外围设备取数据放入缓冲区
 - “<<”是将数据放入输出缓冲区。如有下列语句：

```
os << "please enter the value:";
```

系统将字符串常量存储在与流os关联的缓冲区中

输出缓冲区的刷新

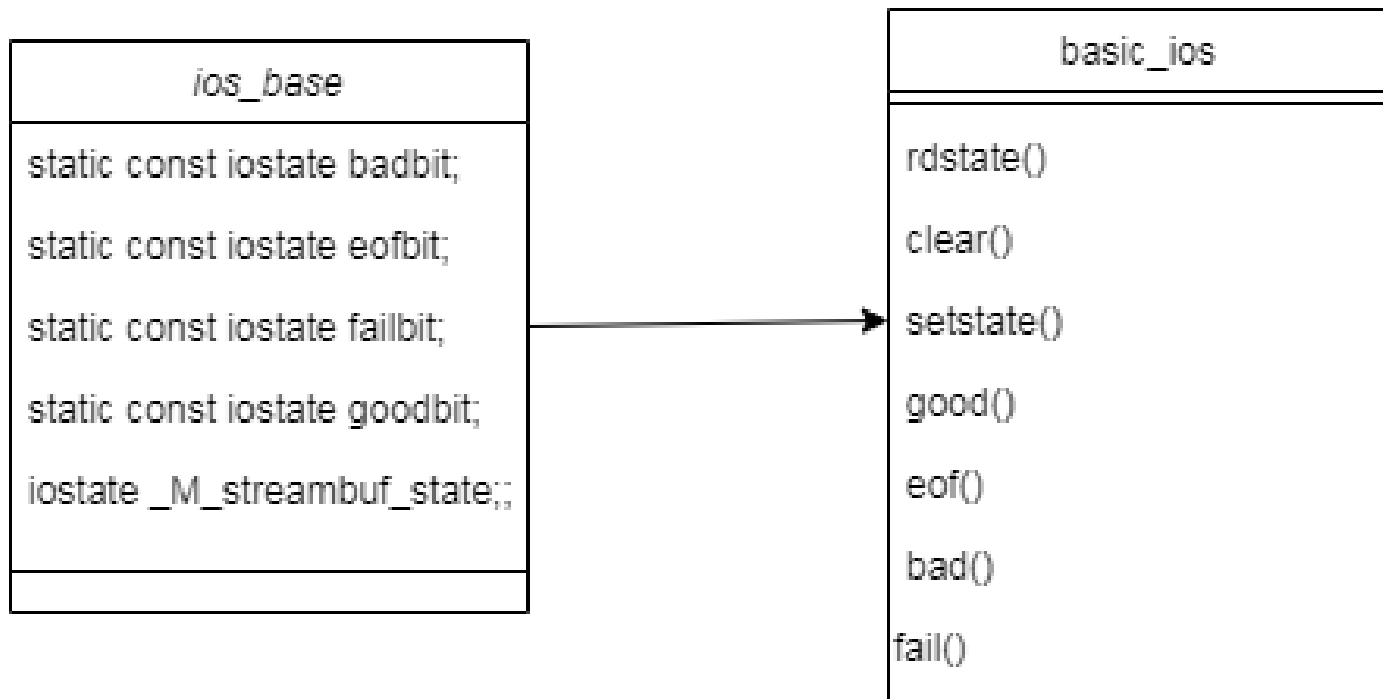


- ① **程序正常结束**。作为main函数返回工作的一部分，将真正输出缓冲区的内容，清空所有的输出缓冲区；
- ② 当**缓冲区已满**时，在写入下一个值之前，会刷新缓冲区；
- ③ **用标准库的操纵符**，如行结束符endl，显式地刷新缓冲区；
- ④ 在每次输出操作执行结束后，**用unitbuf操纵符设置流的内部状态**，从而清空缓冲区；
- ⑤ 可**将输出流与输入流关联起来**。在这种情况下，在读输入流时，将刷新其关联的输出缓冲区。在标准库中，将cout和cin关联在一起，因此每个输入操作都将刷新cout关联的缓冲区。

流状态与操作函数



状态	说明
badbit	在输入输出时遇到了系统级错误，会置为badbit
eofbit	读取文件时读到了文件末尾，就会置为eofbit
failbit	往流缓冲区写入或者读取数据发生错误时，会被置为failbit
goodbit	上面三种都没有时，就是goodbit



```
01. int i = 0;
02. cin >> i;
03. if(!cin){ //只有输入操作失败，才会跳转到这里
04.     if(cin.bad()){ //流发生严重故障，只能退出函数
05.         error("cin is bad!"); //error是自定义函数，它抛出异常，并给出提示信息
06.     }
07.     if(cin.eof()){ //检测是否读取结束
08.         //TODO:
09.     }
10.     if(cin.fail()){ //流遇到了一些意外情况
11.         cin.clear(); //清除/恢复流状态
12.         //TODO:
13.     }
14. }
```

```
//从 ist 中读入整数到 v 中，直到遇到 eof() 或终结符
void fill_vector(istream& ist, vector<int>& v, char terminator){
    for( int i; ist>>i; ) v.push_back(i);
```

//正常情况

```
if(ist.eof()) return; //发现到了文件尾，正确，返回
```

//发生严重错误，只能退出函数

```
if (ist.bad()) error("cin is bad!"); //error是自定义函数，它抛出异常，并给出提示信息
```

//发生意外情况

```
if (ist.fail()) { //最好清除混乱，然后汇报问题
    ist.clear(); //清除流状态
```

//检测下一个字符是否是终结符

```
char c;
```

```
ist>>c; //读入一个符号，希望是终结符
```

```
if(c != terminator) { // 非终结符
```

```
    ist.unget(); //放回该符号
```

```
    ist.clear(ios_base::failbit); //将流状态设置为 fail()
```

```
}
```

```
}}
```

代码来源：<http://c.biancheng.net/view/3755.html>



//从ist中读入整数到v中，直到遇到eof()或终结符

```
void fill_vector(istream& ist, vector<int>& v, char terminator){  
    ist.exceptions(ist.exceptions() | ios_base:: badbit);
```

```
    for (int i; ist>>i; ) v.push_back(i);
```

```
    if (ist.eof()) return; //发现到了文件尾
```

```
    //不是good(), 不是bad(), 不是eof(), ist的状态一定是fail()  
    ist.clear(); //清除流状态
```

```
    char c;
```

```
    ist>>c; //读入一个符号，希望是终结符
```

```
    if (c != terminator) { //不是终结符号，一定是失败了  
        ist.unget(); //也许程序调用者可以使用这个符号  
        ist.clear(ios_base::failbit); //将流状态设置为 fail()
```

```
    }
```

```
}
```

如果 ist 处于 bad() 状态，它会抛出一个标准库异常 ios_base::failure。

输入输出与文件



- 流与标准库
- 基于控制台的I/O
 - 输出流
 - 输入流
 - 格式化输入/输出
- 基于字符串的I/O
- 基于文件的I/O

基于控制台的I/O



■ 标准的输入输出流对象

- cin是类istream的对象，它与标准输入设备(通常指键盘)连在一起。
- cout是类ostream的对象，它与标准输出设备(通常指显示设备)连在一起。
- cerr是类ostream的对象，它与标准错误输出设备连在一起。
- clog是类ostream的对象，它与标准错误输出设备连在一起。

输出流

- C++ 的类ostream提供了格式化输出和无格式输出的功能
- 输出功能包括
 - 用流插入运算符输出标准类型的数据；
 - 用成员函数put输出字符；
 - 成员函数write的无格式化输出；
 - 输出特定形式数值

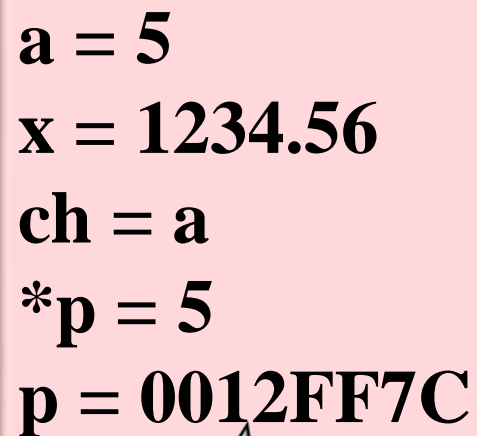
输出标准类型的数据

- 标准类型的数据用流插入运算符<<输出
- 格式：

```
cout << 数据项 ;
```
- C++能自动判别数据类型，并根据数据类型解释内存单元的信息，把它转换成字符显示在显示器上。

输出流

```
#include <iostream>
using namespace std;
int main()
{int a = 5, *p = &a;
 double x = 1234.56;
 char ch = 'a';
 cout << "a = " << a << endl;
 cout << "x = " << x << endl;
 cout << "ch = " << ch << endl;
 cout << "*p = " << *p << endl;
 cout << "p = " << p << endl;
 return 0;
}
```




a = 5
x = 1234.56
ch = a
***p = 5**
p = 0012FF7C



**地址用十六进制
输出**

指针输出的特例

- 如果输出的指针变量是一个指向字符的指针时，C++并不输出该指针中保存的地址，而是输出该指针指向的字符串。
- 如果确实想输出这个指向字符的指针变量中保存的地址值，可以用强制类型转换，将它转换成`void*`类型



```
#include <iostream>
using namespace std;
int main()
{char *ptr = "abcdef";
  cout << "ptr指向的内容为：" << ptr << endl;
  cout << "ptr中保存的地址为：" << (void*)ptr
  << endl;
  return 0;
}
```

ptr指向的内容为：abcdef
ptr中保存的地址为：0046C04C

用成员函数put输出字符

- `cout.put('A');` 将字符A显示在屏幕上，并返回当前对象。
- `cout.put(65);` 用ASCII码值表达式调用put函数，语句也输出字符A。
- 连续调用put函数：
`cout.put('A').put('\n');` 该语句在输出字符A后输出一个换行符。圆点运算符(.)从左向右结合。

write的无格式输出

- 调用成员函数write可实现无格式输出。它有两个参数。第一个参数是一个指向字符的指针，第二个参数是一个整型值。这个函数把一定量的字节从字符数组中输出。这些字节都是未经任何格式化的，仅仅是以原始数据形式输出。
例如：

```
char buffer[] = "HAPPY BIRTHDAY";  
cout.write(buffer, 10 );
```

输出buffer的10个字节
- 函数调用：

```
cout.write("ABCDEFGHIJKLMNOPQRSTUVWXYZ", 10);
```

显示了字母表中的前10个字母。

输入输出与文件



- 流与标准库
- 基于控制台的I/O
 - 输出流
 - 输入流
 - 格式化输入/输出
- 基于字符串的I/O
- 基于文件的I/O

输入流

- 流读取运算符 >>
- Get函数
- Getline函数
- 其他函数

流读取运算符>>

- 输入流最常用的操作是流读取运算符。
- 流读取运算符通常会跳过输入流中的空格、tab键、换行符等空白字符。
- 当遇到输入流中的文件结束符时，流读取运算符返回0(false);否则，流读取运算符返回对调用该运算符的对象的引用。
- 流读取运算符在读入EOF时返回0的特性使得它经常被用作为循环的判别条件，以避免选择特定的表示输入结束的值
- EOF在各个系统中有不同的表示。在windows中是Ctrl+z

实例



- 统计某次考试的最高成绩。假定事先不知道有多少个考试成绩，在输入结束时用户会输入表示成绩输入完毕的文件结束符。当用户输入文件结束符时，while循环结构中的条件(`cin >> grade`)将变为0(即false)。

```
#include <iostream>
using namespace std;
int main()
{int grade, highestGrade = -1;
  cout << "Enter grade (enter end-of-file to end): ";
  while ( cin >> grade) {
    if ( grade > highestGrade) highestGrade = grade;
    cout << "Enter grade (enter end-of-file to end): ";
  }
  cout << "\n\nHighest grade is: " << highestGrade << endl;
  return 0;
}
```

输出结果：

```
Enter grade (enter end-of-file to end): 67
Enter grade (enter end-of-file to end): 87
Enter grade (enter end of file to end): 73
Enter grade (enter end-of-file to end): 95
Enter grade (enter end-of-file to end): 34
Enter grade (enter end-of-file to end): 99
Enter grade (enter end-of-file to end): ^ z
Highest grade is: 99
```

成员函数get



- Get函数用于读入字符或字符串
- get函数有三种格式：
 - 不带参数
 - 带一个参数
 - 带三个参数

不带参数的get函数

- 不带参数的get函数从当前对象读入一个字符，包括空白字符以及表示文件结束的EOF，并将读入值作为函数的返回值返回。如下列语句

```
while((ch = cin.get()) != EOF) cout<< ch;
```

将输入的字符回显在显示器上，直到输入EOF。

```
#include <iostream>
Using namespace std;
int main()
{ char c;
  while ( ( c = cin.get() ) != EOF )
    cout.put( c );
  cout << "\nEOF in this system is: " << c;
  return 0;
}
```

输出结果：

```
Enter a sentence followed by end-of-file:
Testing the get and put member functions^z
Testing the get and put member functions
EOF in this system is: -1
```

带一个参数的get函数

- 带一个字符类型的引用参数，它将输入流中的下一字符（包括空白字符）存储在参数中，它的返回值是当前对象的引用。

例如，下面的循环语句将输入一个字符串，存入字符数组ch，直到输入回车。

```
cin.get(ch[0]);
for (i = 0; ch[i] != '\n'; ++i) cin.get(ch[i+1]);
ch[i] = '\0';
```

带有三个参数的get成员函数

▪ 参数

- 接收字符的字符数组
- 字符数组的大小
- 分隔符(默认值为'\n')。

▪ 函数或者在读取比指定的最大字符数少一个字符后结束，或者在遇到分隔符时结束。

- 为使字符数组(被程序用作缓冲区)中的输入字符串能够结束，空字符会被插入到字符数组中。
- 函数不把分隔符放到字符数组中，但是分隔符仍然会保留在输入流中。

- 要输入一行字符，可用下列语句：

```
cin.get(ch, 80, '\n'); 或 cin.get(ch, 80);
```

- 要输入一个以句号结尾的句子，可用：

```
cin.get(ch, 80, '.');
```

- 当遇到输入结束符时，程序插入一个'\0'作为输入字符串的结束标记，输入结束符没有放在字符数组中，而是保留在输入流中，下一个和输入相关的语句会读入这个输入结束符。如对应于语句

```
cin.get(ch, 80, '.');
```

用户输入

abcdef. ✓

则ch中保存的是字符串“abcdef”，而“.”仍保留在输入缓冲区中。如果继续调用

```
cin.get(ch1); 或 cin >> ch1;
```

则字符变量ch1中保存的是“.”。


```
#include <iostream>
Using namespace std;
int main()
{ const int SIZE = 80;
  char buffer1[ SIZE ], buffer2[ SIZE ] ;
  cout << "Enter a sentence:\n";
  cin >> buffer1;
  cout << "\nThe string read with cin was:\n"
        << buffer1 << "n\n";
  cin.get( buffer2, SIZE );
  cout << "The string read with cin.get was:\n"
        << buffer2 << endl;
  return 0;
}
```

输出结果：

Enter a sentence:

Contrasting string input with cin and cin.get

The string read with cin was:

Contrasting

The string read with cin.get was:

string input with cin and cin.get

成员函数getline



- 与带三个参数的get函数类似，它读取一行信息到字符数组中，然后插入一个空字符。所不同的是，getline要去除输入流中的分隔符(即读取字符并删除它)，但是不把它存放在字符数组中。

```
#include <iostream>
Using namespace std;
int main()
{ const SIZE = 80;
  char buffe[ SIZE ];
  cout << "Enter a sentence:\n";
  cin.getline( buffer, SIZE );
  cout << "\nThe sentence entered is:\n" << buffer
        << endl;
  return 0;
}
```

输出结果：

Enter a sentence:

Using the getline member function

The sentence entered is:

Using the getline member function

用Read函数输入

- 调用成员函数read可实现无格式输入。它有两个参数。第一个参数是一个指向字符的指针，第二个参数是一个整型值。这个函数把一定量的字节从输入缓冲区读入字符数组，不管这些字节包含的是什么内容。

例如：
`char buffer[80] ;`
`cin.read(buffer, 10) ;`

读入10个字节，放入buffer

- 如果还没有读到指定的字符数，遇到了EOF，则读操作结束。此时可以用成员函数gcount统计输入的字符个数

```
#include <iostream>
using namespace std;
int main()
{char buffer[ 80 ];
  cout << "Enter a sentence:\n";
  cin.read( buffer, 20 );
  cout << "\nThe sentence entered was:\n";
  cout.write( buffer, cin.gcount() );
  cout << endl;
  cout << "一共输入了" << cin.gcount() << "个字符\n";
  return 0;
}
```

输出结果：

Enter a sentence:

Using the read, write, and gcount member functions

The sentence entered was:

Using the read,write

一共输入了 20个字符

输入输出与文件



- 流与标准库
- 基于控制台的I/O
 - 输出流
 - 输入流
 - 格式化输入/输出
- 基于字符串的I/O
- 基于文件的I/O

格式化输入/输出



- C++提供了大量用于执行格式化输入/输出的流操纵算子和成员函数。

- 功能:

整数流的基数：dec、oct、hex和setbase

设置浮点数精度:precision、setprecision

设置域宽:setw、width

设置域填充字符:fill、setfill

设置整型数的基数



- 输入输出流中的整型数默认为十进制表示。
 - 可以插入hex操纵符将基数设为十六进制，插入oct操纵符将基数设为八进制，也可以插入dec操纵符将基数重新设为十进制
 - 也可以通过流操纵符setbase来改变流的基数。该操纵符有一个整型参数，它的值可以是16，10或8，表示将整型数的基数设为十六进制，十进制或八进制
- 使用任何带参数的流操纵符，都必须包含头文件iomanip
- 流的基数值只有被显式更改时才会变化，否则一直沿用原有的基数。

hex、oct、dec和setbase



```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{int n;
  cout << "Enter a octal number: ";
  cin >> oct >> n;
  cout << "octal " << oct << n
       << " in hexadecimal is:" << hex << n << '\n' ;
  cout << "hexadecimal " << n
       << " in decimal is:" << dec << n << '\n' ;
  cout << setbase(8) << "octal " << n
       << " in octal is:" << n << endl;
  return 0;
}
```

Enter a octal number: 30
Octal 30 in hexadecimal is: 18
Hexadecimal 18 in decimal is: 24
Octal 30 in octal is: 30

设置浮点数精度

- 设置浮点数的精度（即，实型数的有效位数）可以用流操纵符setprecision或基类ios的成员函数precision来实现。
- 一旦调用了这两者之中的某一个，将影响所有输出的浮点数的精度，直到下一个设置精度的操作为止。
- 这个操纵符和成员函数都有一个参数，表示有效位数的长度。

```
double x = 123.456789, y = 9876.54321;
```

```
for (int i = 9; i > 0; --i)
```

```
{cout.precision(i); cout << x << '\t' << y << endl;}
```

```
//for (int i = 9; i > 0; --i)
```

```
// cout << setprecision(i) << x << '\t' << y << endl;
```

```
123.456789  9876.54321
```

```
123.45679   9876.5432
```

```
123.4568    9876.543
```

```
123.457     9876.54
```

```
123.46      9876.5
```

```
123.5       9877
```

```
123         9.88e+003
```

```
1.2e+002    9.9e+003
```

```
1e+002      1e+004
```

设置域宽



- 域宽是指数据所占的字符个数。
- 设置域宽可以用基类的成员函数 `width`，也可以用流操纵符 (`setw`)。`width`和`setw`都包含一个整型的参数，表示域宽。
- 设置域宽可用于输入，也可用于输出。设置宽度是适合于下一次输入或输出，之后的操作的宽度将被设置为默认值。
- 当没有设置输出宽度时，C++按实际长度输出。如整型变量 `a=123`，`b=456`，则输出

```
cout << a << b;
```


将输出123456。

- 一旦设置了域宽，该输出必须占满域宽。如果输出值的宽度比域宽小，则插入填充字符填充。默认的填充字符是空格。如果实际宽度大于指定的域宽，则按实际宽度输出。如语句 `cout << setw(5) << x << setw(5) << y << endl;`
的输出为 `123 456`
每个数值占5个位置，前面用空格填充。
- 设置域宽也可用于输入。当输入是字符串时，如果输入的字符个数大于设置的域宽时，C++只读入域宽指定的字符个数。如有定义 `char a[9]`，`b[9]`；
执行语句 `cin >> setw(5) >> a >> setw(5) >> b;`
用户在键盘上的响应为 `abcdefghijklm`
则字符串 `a` 的值为“abcd”，字符串 `b` 的值为“defg”。

其他流操纵符



流操纵符	描述
skipws	跳过输入流中的空白字符，使用流操纵符noskipws复位该选项
left	输出左对齐，必要时在右边填充字符
right	输出右对齐，必要时在左边填充字符
showbase	指名在数字的前面输出基数，以0开头表示八进制，0x或0X表示十六进制。使用流操纵符noshowbase复位该选择
uppercase	指明当显示十六进制数时使用大写字母，并且在科学计数法输出时使用大写字母E。可以用流操纵符nouppercase复位
showpos	在正数前显示加号（+），可以用流操纵符noshowpos复位
scientific	以科学计数法输出浮点数
fixed	以定点小数形式输出浮点数
setfill	设置填充字符，它有一个字符型的参数

用户自定义的流操纵算子



- 程序员可以定义自己的流操纵符
- 例如，定义输出流操纵符格式如下：

```
ostream &操纵符名 (ostream &os)
```

```
{需要执行的操作}
```

```
#include <iostream>
using namespace std;
ostream &tab(ostream &os)
    {return os << '\t';}

int main()
{int a=5,b=7;
  cout << a << tab << b << endl;
  return 0;
}
```

5

7

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- **基于字符串的I/O**
- 基于文件的I/O

基于字符串的I/O



- iostream标准库支持内存中的输入输出，只要将流与存储在程序内存中的string对象捆绑起来即可
- 标准库定义了三种类型的字符串流：
 - istreamstringstream：由istream派生而来，提供读string的功能。
 - ostreamstringstream：由ostream派生而来，提供写string的功能。
 - stringstream：由iostream派生而来，提供读写string的功能。

字符串流使用实例



```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{ string ch;
  ostringstream os(ch); // 或 ostringstream os;
  for (int i = 0; i<=20; ++i) os << i << ' ';
  cout << os.str();
  cout << endl;
  istringstream is(os.str());
  while (is >> i) cout << i << '\t';
  return 0;
}
```

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- 基于字符串的I/O
- 基于文件的I/O
 - 文件的概念
 - 文件的顺序访问
 - 文件的随机访问
 - 访问有记录概念的文件

文件的概念

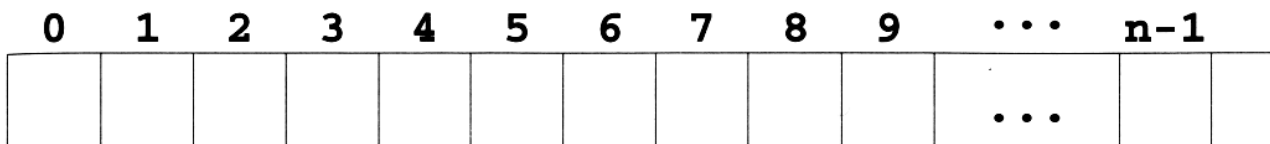


- 文件是驻留在外存储器上、具有标识名的一组信息集合，用来永久保存数据。
- 与文件相关的概念有：
 - 数据项（字段）
 - 记录
 - 文件
 - 数据库
- 如在一个图书管理系统中，有一个数据库。这个数据库由书目文件、读者文件及其它辅助文件组成。书目文件中保存的是图书馆中的所有书目信息，每本书的信息构成一条记录。每本书需要保存的信息有：书名、作者、出版年月、分类号、ISBN号、图书馆的馆藏号以及一些流通信息。其中书名是一个字段，作者也是一个字段。

文件和流



- C++把每一个文件都看成一个有序的字节流（把文件看成 n 个字节）
 - 每一个文件以文件结束符(end-of-file marker)结束
 - 当打开一个文件时，该文件就和某个流关联起来
 - 与这些对象相关联的流提供程序与特定文件或设备之间的通信通道
 - 例如：cin对象(标准输入流对象)使程序能从键盘输入数据，cout对象(标准输出流对象)使程序能向屏幕输出数据。



文件结束符

文件访问过程



- 定义一个流对象
- 打开文件：将流对象与文件关联起来
- 访问文件
- 关闭文件：切断流对象与文件的关联

定义一个流对象



- C++ 有三个文件流类型：
 - ifstream：输入文件流
 - ofstream：输出文件流
 - fstream：输入输出文件流
- 如：ifstream infile;

打开文件

- 用流对象的成员函数open/用流对象的构造函数打开文件
 - 有两个参数：打开的文件名/文件打开模式
 - 如果文件打开失败，返回0
- 打开输入文件：
ifstream infile;
infile.open("f1"); 或 infile.open("f1", ifstream::in);
也可以利用构造函数直接打开：
ifstream infile("f1"); 或 ifstream infile("f1", ifstream::in);
- 打开输出文件
ofstream outfile;
outfile.open("f2"); 或 outfile.open("f2", ofstream::out);
也可以利用构造函数直接打开：
ofstream outfile("f2"); 或 ofstream outfile("f2", ofstream::out);
- 打开输入输出文件
fstream iofile("f3");
fstream iofile("f3", fstream::in | fstream::out);

文件打开模式

文件打开模式名	含义
in	打开文件，做读操作
out	打开文件，做写操作
app	在每次写操作前，找到文件尾
ate	打开文件后，立即将文件定位在文件尾
trunc	打开文件时，清空文件
binary	以二进制模式进行输入输出操作

• 默认打开方式

- ifstream流对象是以in模式打开
- ofstream流关联的文件以out模式打开
- fstream对象以in和out方式打开

文件关闭



- 用成员函数close
- main函数执行结束时，会关闭所有打开的文件
- 良好的程序设计习惯：文件访问结束时，关闭文件

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- 基于字符串的I/O
- **基于文件的I/O**
 - 文件的概念
 - **文件的顺序访问**
 - 文件的随机访问
 - 访问有记录概念的文件

文件的顺序访问



- C++文件的读写和控制台读写一样，可以用流提取运算符“>>”从文件读数据，也可以用流插入运算符“<<”将数据写入文件，也可以用文件流的其他成员函数读写文件，如get函数，put函数等。
- 在读文件操作中，经常需要判断文件是否结束（文件中的数据是否被读完）。
 - 可以通过基类ios的成员函数eof来实现。eof函数不需要参数，返回一个整型值。当读操作遇到文件结束时，该函数返回1，否则返回0
 - 另一种判断读结束的方法是用流提取操作的返回值。当“>>”操作成功时，返回true

文件访问实例

- 将数字1到10写入文件file，然后从file中读取这些数据，把它们显示在屏幕上。
- 首先用输出方式打开文件file。如文件file不存在，则自动创建一个，否则打开磁盘上的文件，并清空。用一个循环依次将1到10用流插入符插入文件，并关闭文件。然后，再用输入方式打开文件file，读出所有数据，并输出到屏幕上。

执行该程序后，文件file中的内容为

1 2 3 4 5 6 7 8 9 10

该程序的输出结果是

1 2 3 4 5 6 7 8 9 10

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    ofstream out("file");
    ifstream in;
    int i;
    if (!out) {cerr << "create file error\n"; return 1;}
    for (i = 1; i <= 10; ++i) out << i << ' ';
    out.close();

    in.open("file");
    if (!in) {cerr << "open file error\n"; return 1;}
    while (in >> i) cout << i << ' ';
    in.close();
    return 0;
}
```


包含各种类型数据的文件操作



```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{ofstream fout("test");
  if (!fout){cerr <<"cannot open output file\n";
    return 1;}
  fout<<10<<" "<<123.456<<"\"This is a text
  file\\n";
  fout.close();
  return 0;
}
```

文件中的内容为

10 123.456"This is a text file"

读文件

```
#include <fstream>
#include <iostream>
using namespace std;
int main(){
    ifstream fin("test");
    char s[80];
    int i;    float x;

    if (!fin) {cout << "cannot open input file\n"; return 1;}
    fin >> i >> x >> s;
    cout << i << " " << x << s;
    fin.close();

    return 0;
}
```

10 123.456"This

fin >> i >> x; fin.getline(s, 80, '\n');

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- 基于字符串的I/O
- **基于文件的I/O**
 - 文件的概念
 - 文件的顺序访问
 - **文件的随机访问**
 - 访问有记录概念的文件

文件定位指针



- 文件定位指针：是一个long类型的数据，指出当前读写的位置
- C++文件有两个定位指针：**读指针**和**写指针**
 - 当文件以输入方式打开时，读指针指向文件中的第一个字节。
 - 文件以输出方式打开时，写指针指向文件中的第一个字节。
 - 当文件以添加方式打开时，写指针指向文件尾。

文件的随机访问



- 指定文件定位指针的值，从任意指定位置开始读写
- 获取文件定位指针的当前位置：成员函数tellg和tellp
- 设置文件定位指针的位置：成员函数seekg和seekp

成员函数seekg和seekp

- seekg和seekp都有两个参数：
第一个参数通常为long类型的整数，表示偏移量；第二个参数指定移动的起始位置
- 寻找方向：
 - ios::beg(默认)：相对于流的开头
 - ios::cur：相对于流当前位置
 - ios::end：相对于流结尾

```
// position to the nth byte of fileObject  
// assumes ios::beg  
fileObject.seekg( n );
```

```
// position n bytes forward in fileObject  
fileObject.seekg( n, ios::cur );
```

```
// position y bytes back from end of fileObject  
fileObject.seekg( y, ios::end );
```

```
// position at end of fileObject  
fileObject.seekg( 0, ios::end );
```


随机读写实例

```
fstream in("file");  
int i;  
  
if (!in) {cerr << "open file error\n"; return 1;}  
in.seekp(10);  
in << 20;  
in.seekg(0);  
while (in >> i) cout << i << ' '  
in.close();
```

执行前: 1 2 3 4 5 6 7 8 9 10

执行后: 1 2 3 4 5 20 7 8 9 10

输入输出与文件



- 流与标准库
- 基于控制台的I/O
- 基于字符串的I/O
- **基于文件的I/O**
 - 文件的概念
 - 文件的顺序访问
 - 文件的随机访问
 - **访问有记录概念的文件**

访问要求

- 立即访问到文件甚至大型文件中指定的记录
- 可以在不破坏其他数据的情况下把数据插入到随机访问文件中。
- 也能在不重写整个文件的情况下更新和删除以前存储的数据。

实现考虑



- 要求记录长度是固定的。
- 可以使用istream中read函数和ostream中的write函数
- 如number是整型变量
 - 写入：

```
outFile.write(reinterpret_cast<const char * > (&number), sizeof(number));
```
 - 读出：

```
inFile.read(reinterpret_cast<char * > (&number), sizeof(number));
```

实例：图书馆的书目管理系统



- 如果每本书需要记录的信息有：
 - 馆藏号（整型数）：要求自动生成
 - 书名（最长20个字符的字符串）
 - 借书标记。借书标记中记录的是借书者的借书证号，假设也是整型数。

- 该系统需要实现的功能有：
 - 初始化系统
 - 添加书
 - 借书
 - 还书
 - 显示书库信息

图书目录			
书号	书名	价格	备注
201X-1-1	发现之旅	75	
201X-1-2	大众天文学（上下）	128	
201X-1-3	吃好每天3顿饭	29.8	
201X-1-4	马克思主义哲学	24	
201X-1-5	从一无所知大科学中的事实和臆测	29	
201X-1-6	结束南北大	32	
201X-1-7	礼仪常识全识	29.8	
201X-1-8	不可不知的1000个法律常识	48	
201X-1-9	幸福的方法	28	
201X-1-10	我佛的故事	39.8	
201X-1-11	感动中国——感动力量	35	
201X-1-12	生命的呐喊	32	
201X-1-13	我的安全我做主	30	
201X-1-14	培育与践行社会主义核心价值观	15	
201X-1-15	平凡的世界 一、二、三	79.8	
201X-1-16	十年漫步中国	39	
201X-1-17	苦难辉煌 上下	88	
201X-1-18	中国震撼——一个文明型国家的崛起	30	
201X-1-19	中华经典藏书——唐诗三百首	19	
201X-1-20	论语——中华经典藏书	16	
201X-1-21	周朝中国历史读本	66	
201X-1-22	中国读本	38	
201X-1-23	3D打印——从想象到现实	49	
201X-1-24	日出东方中国共产党创建纪实	48	
201X-1-25	改革是中国最大的红利	39.8	
201X-1-26	习近平关于实现中华民族伟大复兴的中国梦论述摘编	8.6	
201X-1-27	钱学森——故事	56	
201X-1-28	中国共产党历史第二卷 上下册	150	
201X-1-29	经典译林——革命记	21.8	
201X-1-30	解读中国经济	39	
201X-1-31	吃好每天3顿饭2 这样吃饭更健康	29.8	
201X-1-32	世界是平的	58	
201X-1-33	万物简史	36.8	
201X-1-34	之江新语	36	
201X-1-35	好妈妈胜过好老师——一个教育专家的教子手记	28	
201X-1-36	守住中国大地的底线	39.8	
201X-1-37	大数据时代	49.9	
201X-1-38	永不止步	29.8	
201X-1-39	当下的力量	32	
201X-1-40	气功修习术	19.8	
201X-1-41	气功修习术 终极篇	28.8	
201X-1-42	气功修习术 2	20	
201X-1-43	当世界无法改变时改变自己	32.8	

文件设计



- 设计一个文件book，该文件中的每个记录保存一本书的信息。
- 文件中的记录可按馆藏号的次序存放，这样可方便实现添加书和借还书的功能。
添加书时，只要将这本书对应的记录添加到文件尾。借还书时，可以根据馆藏号计算记录的存储位置，修改相应的记录。

book类设计

- 数据成员：
 - 馆藏号、书名、借书标记
 - 为了提供馆藏号自动生成，需要保存系统中最大的馆藏号。这个值可以作为书目类的静态成员。
- 成员函数：
 - 构造函数
 - 借书
 - 还书
 - 显示书的详细信息
 - 静态成员初始化
 - 静态成员值加1

```
#ifndef _book_h
#define _book_h
#include <cstring>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
class book {
    int no;
    char name[20];
    int borrowed;
    static int no_total;

public:
    book(const char *s = "") {no = no_total; borrowed = 0; strcpy(name,s);}
    void borrow(int readerNo) { //借书
        if (borrowed != 0) cerr << "本书已被借，不能重复借\n";
        else borrowed = readerNo; }
    void Return() { //还书
        if (borrowed == 0) cerr << "本书没有被借，不能还\n";
        else borrowed = 0; }

    void display(){ //显示书的信息
        cout << setw(10) << no << setw(20) << name << setw(10) << borrowed << endl;}

    static void resetTotal() {no_total = 0;} //最大馆藏号复位
    static void addTotal() {++no_total;} //馆藏号加1
};

int book::no_total = 0; //静态数据成员的定义
#endif
```

系统分解

- 系统可分解成五大功能，每个功能用一个函数实现。
- Main函数显示菜单，根据用户的选择调用相应的函数

```
#include "book.h"
```

```
void initialize(); //系统初始化
```

```
void addBook(); //添加新书
```

```
void borrowBook(); //借书
```

```
void returnBook(); //还书
```

```
void displayBook(); //显示所有的书目信息
```

```
int main()
```

```
{int selector;
```

```
while (true) {
```

```
    cout << "0 -- 退出\n";
```

```
    cout << "1 -- 初始化文件\n";
```

```
    cout << "2 -- 添加书\n";
```

```
    cout << "3 -- 借书\n";
```

```
    cout << "4 -- 还书\n";
```

```
    cout << "5 -- 显示所有书目信息\n";
```

```
    cout << "请选择 (0-5)  : ";
```

```
    cin >> selector;
```

```
    if (selector == 0) break;
```

```
    switch (selector){
```

```
        case 1: initialize(); break;
```

```
        case 2: addBook(); break;
```

```
        case 3: borrowBook(); break;
```

```
        case 4: returnBook(); break;
```

```
        case 5: displayBook(); break;
```

```
    } }
```

```
    return 0;
```

```
}
```


Initialize的实现

```
void initialize() {  
    ofstream outfile("book");  
    book::resetTotal();  
    outfile.close();  
}
```

addBook的实现

```
void addBook() {  
    char ch[20];  
    book *bp;  
    ofstream outfile("book",ofstream::app);  
  
    book::addTotal();  
    cout << "请输入书名 : ";  
    cin >> ch;  
    bp = new book(ch);  
    outfile.write( reinterpret_cast<const char *>(bp),  
                 sizeof(*bp));  
  
    delete bp;  
    outfile.close();  
}
```

borrowBook

```
void borrowBook()
{int bookNo, readerNo;
 fstream iofile("book");
 book bk;
 cout << "请输入书号和读者号 : ";
 cin >> bookNo >> readerNo;
 iofile.seekg((bookNo - 1) * sizeof(book));
 iofile.read( reinterpret_cast<char *> (&bk),
             sizeof(book) );
 bk.borrow(readerNo);
 iofile.seekp((bookNo - 1) * sizeof(book));
 iofile.write( reinterpret_cast<const char *>(&bk),
              sizeof(book));
 iofile.close();
}
```

returnBook

```
void returnBook()
{int bookNo;
 fstream iofile("book");
 book bk;
 cout << "请输入书号 : ";
 cin >> bookNo ;
 iofile.seekg((bookNo - 1) * sizeof(book));
 iofile.read( reinterpret_cast<char *> (&bk),
             sizeof(book) );
 bk.Return();
 iofile.seekp((bookNo - 1) * sizeof(book));
 iofile.write( reinterpret_cast<const char *>(&bk),
              sizeof(book));
 iofile.close();
}
```

displayBook



```
void displayBook()
{ifstream infile("book");
  book bk;
  infile.read( reinterpret_cast<char *> (&bk), sizeof(book) );
  while (!infile.eof()) {
    bk.display();
    infile.read( reinterpret_cast<char *> (&bk), sizeof(book) );
  }
  infile.close();
}
```

小结



- 输入输出是程序中不可缺少的一部分。在C++中，输入输出功能是以标准库的形式来提供。输入输出操作分为控制台I/O，文件I/O以及字符串I/O。由于文件I/O和字符串I/O类都是从控制台I/O类继承的，因此，这三种I/O的操作方式是相同的。
- 本章介绍了如何利用iostream库进行格式化的输入输出，介绍了如何利用文件永久保存信息，并以图书馆系统为例，介绍了实现文件的随机读写。