CS 1502H 程序设计思想与方法 (C++)
Chapter 14. 输入/输出与文件

陈东尧
上海交通大学
2025年秋季

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O

流与标准库

输入输出是指程序与外部设备交换信息

C++把输入输出作为一个**数据流（data stream）**

- 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**

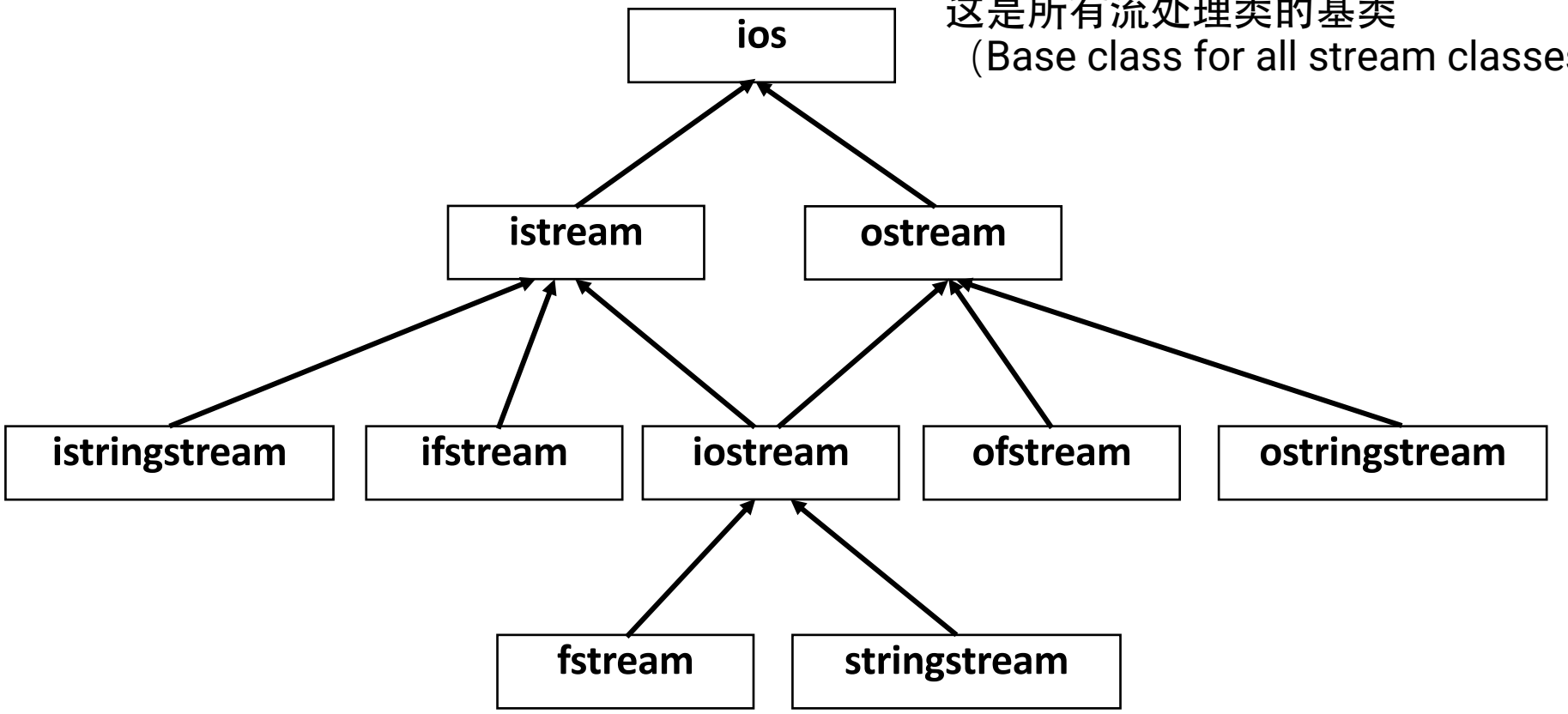
C++的输入输出分为

- 基于控制台的I/O
- 基于文件的I/O
- 基于字符串的I/O

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

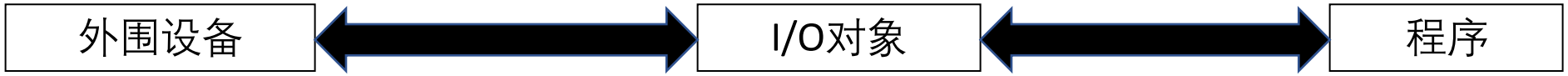

ios
这是所有流处理类的基类
（Base class for all stream classes）
istream
ostream
istringstream
ifstream
iostream
ofstream
ostringstream
fstream
stringstream

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O

输入输出缓冲

- C++程序不能直接与输入输出设备交换信息
- 输入输出是通过一个对象实现的。对象是输入输出设备在程序中的代理
- 每个I/O对象管理一个缓冲区，用于存储程序读写的数据
- 外围设备与对象批量交换数据



以控制台输入输出为例

输入

- 当用户在键盘上输入数据，在输入回车后，输入数据被移到输入缓冲区
- 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

输出

- “<<”是将数据放入输出缓冲区
- 如有下列语句：

```
cout << “please enter the value:”;
```

将字符串常量存储在cout的缓冲区中

那么何时刷新输出缓冲区呢?

输出缓冲区的刷新

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；

缓冲区满

在写入下一个值之前，会刷新缓冲区；

强制刷新

用标准库的操纵符，如行结束符endl，显式地刷新缓冲区；

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；

关联输出流与输入流

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。

输出缓冲区的刷新

程序正常结束

缓冲区满

强制刷新

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；

关联输出流与输入流

```
#include <iostream>
#include <sstream>

using namespace std;

int main()
{

   // Initializing the character
   char a, b, c;

   // Stream input with leading whitespaces
   istringstream iss(”KFC");

   // Using unitbuf()
   // buffer flushed 3 times
   iss >> unitbuf >> a >> b >> c;
   cout << "unitbuf flag: "
        << a << endl
        << b << endl
        << c << endl;

   return 0;
}
```

```
unitbuf flag:
K
F
C
```

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O

基于控制台的I/O

- **通过4个标准的输入输出流对象完成**

cin对象

类istream的对象，与标准输入设备(通常指键盘)连在一起

cout对象

类ostream的对象，与标准输出设备(通常指显示设备)连在一起

cerr对象

类osteam的对象，它与标准错误输出设备(通常指显示设备）连在一起，没有缓冲

clog对象

类ostream的对象，与标准错误输出设备(通常指显示设备）连在一起，有缓冲

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

设置其他数制，可以用流操纵符

十六进制	八进制	十进制
hex setbase(16)	oct setbase(8)	dec setbase(10)

注意

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 an octal number: ";
    cin >> oct >> n;

    cout << "octal " << oct << n   << " in hexdecimal is:" << hex << n << '\n' ;
    cout << "hexdecimal " << 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 hexdecimal is: 18
Hexdecimal 18 in decimal is: 24
Octal 30 in octal is: 30
```

设置浮点数精度

- **浮点数的精度指实型数的有效位数**

- **设置方法**
- 流操纵符：setprecision（位数）
- 成员函数：precision （位数）

设置了精度后，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

设置浮点数精度

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double x = 123.456789, y = 9876.54321;

    for (int i = 9; i > 0; --i)      {
        cout.precision(i);
        cout << x << '\t' << y << endl;
    }

     return 0;
}
```

```
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
```

设置小数点后的位数

将两个流操纵符fixed和setprecision结合使用。fixed表示以**定点小数形式**输出，此时setprecision的参数表示小数点后的位数

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double x = 123.456789, y = 9876.54321;

    cout << fixed << setprecision(2) << x << '\t' << y << endl;
    cout << fixed << setprecision(3) << x << '\t' << y << endl;

    return 0;
}
```

```
123.46      9876.54
123.457     9876.543
```

设置域宽

域宽是指数据所占的字符个数

设置方法

成员函数：width（宽度）

流操纵符：setw（宽度）

注意

可用于输入，也可用于输出

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

一旦设置了域宽，下一个输出必须占满域宽

如果输出值的宽度比域宽小，则左边插入填充字符填充。默认的填充字符是空格

当没有设置输出宽度时，C++按实际长度输出

设置域宽

如整型变量a=123，b=456，则输出

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

将输出 123456

- 如语句

```
cout << setw(5) << a <<  setw(5) << b << 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

- 设置域宽也可用于输入
- 当输入是字符串时，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“efgh”。

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

- **返回值是ostream或istream引用的函数，可以直接插入在输入输出流中**
- **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
}
```

如何定义带参数的流操纵符?

用户自定义的流操纵算子

```
#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
```

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O
  - 流式文件
  - 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

数据项（字段）

记录

文件

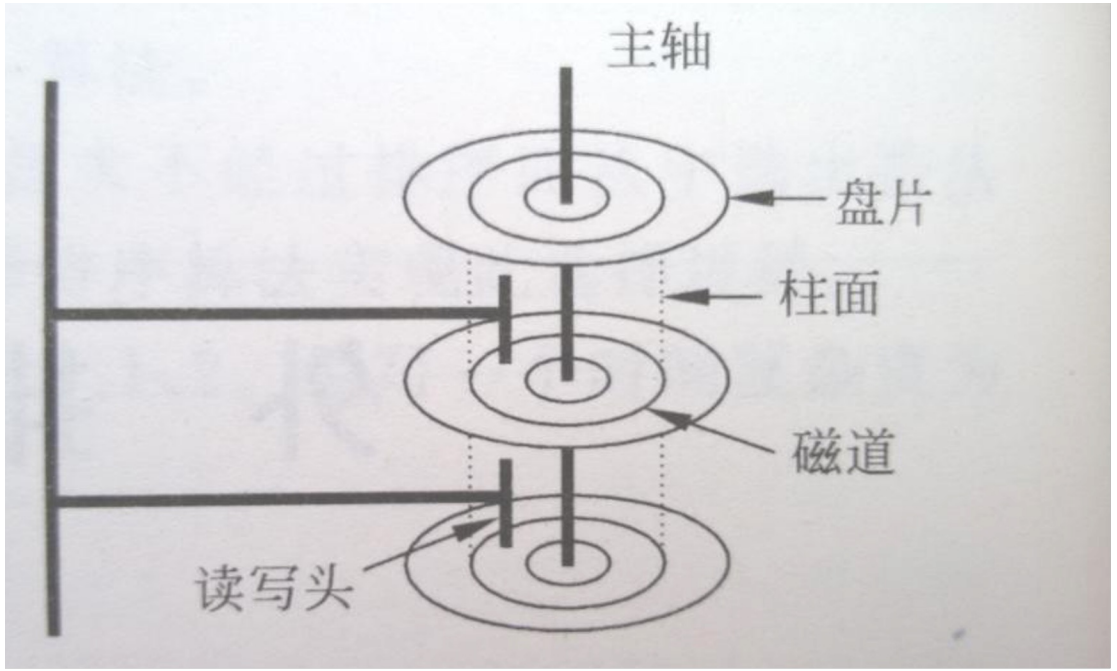
数据库

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。

外存与内存

程序可以直接访问内存中的数据，但不能直接访问外存

外存访问原理



磁道

存储信息的场所

柱面

不同盘片的同一磁道

扇区

磁道划分成扇区，存储一个数据块

访问速度

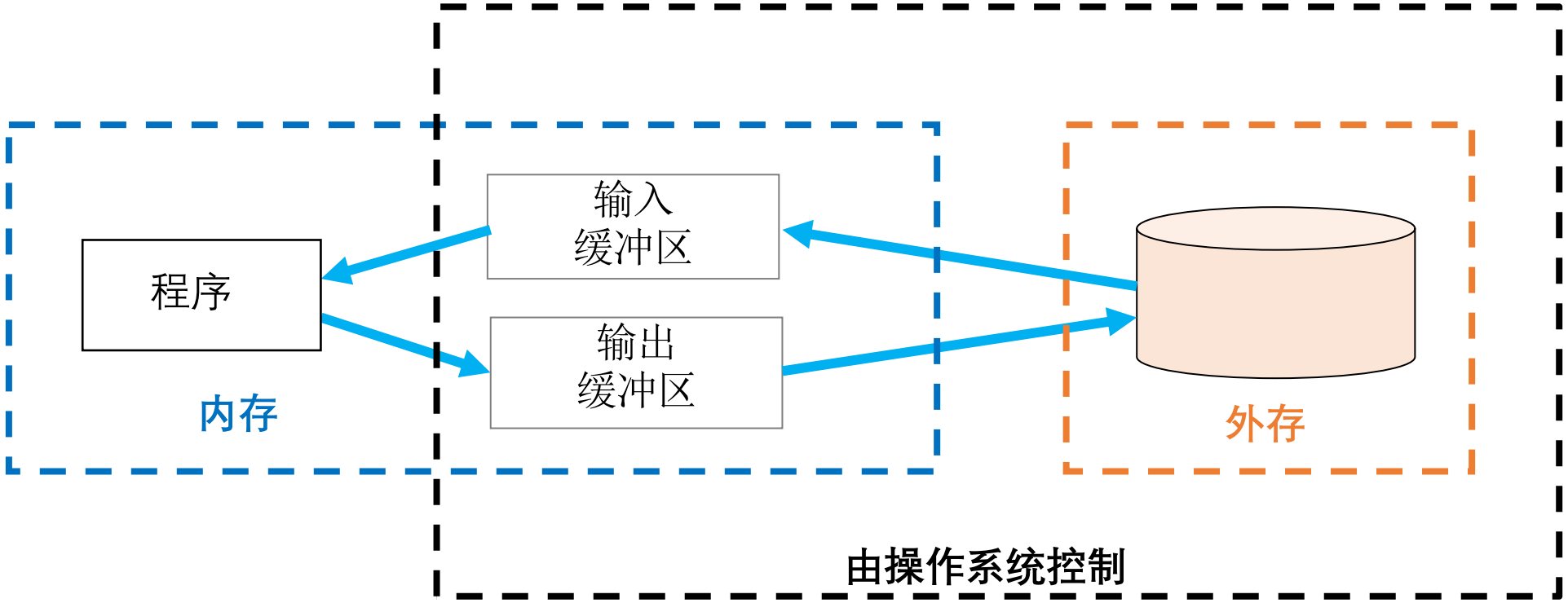
5400转：50-90MB/s

7200转：90-190MB/s

固态硬盘：500MB/s

外存储器的访问

- 外存储器中的信息以文件为单位
- 每个文件对应一个对象
- 每个对象在内存有一个缓冲区存放正在处理的文件中的数据块

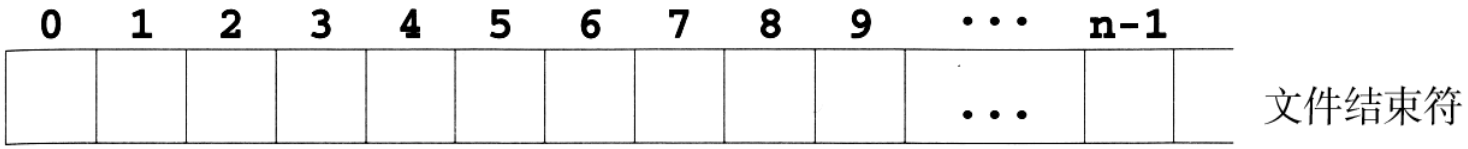


本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O
  - 流式文件
  - 文件的顺序访问
  - 文件的随机访问
  - 访问有记录概念的文件

流式文件

- **C++语言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



ASCII文件和二进制文件

程序如何解释字节序列

ASCII文件

也被称为文本文件

将文件中的每个字节解释成一个字符的ASCII值

二进制文件

将文件中的每个字节仅看成是一个二进制比特串

由程序解释比特串的含义

如果二进制文件有4个字节，值为0000 0000 0000 0000 1111 1111 1111 1111

想解释成一个整型数，可以将这4个字节读入一个整型变量

想解释成float类型，可以将这4个字节读入一个float类型的变量

ASCII文件可以直接显示在显示器上

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

两种打开方式

- 成员函数open
- 构造函数

函数参数

- 打开的文件名：一个字符串
- 文件打开模式

检查文件打开是否成功

- 打开失败，文件流对象值为0

打开模式	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

默认打开模式

ifstream对象：以in模式打开

ofstream对象：以out模式打开

fstream对象：以in和out方式打开

打开文件示例

打开输入文件

用open函数： ifstream infile;
infile.open（“file1”）; 或 infile.open（“file1”, ifstream::in);
用构造函数： ifstream infile（“file1”）; 或 ifstream infile（“file1” , ifstream::in);

打开输出文件

用open函数： ofstream outfile;
outfile.open（“file2”）; 或 outfile.open（“file2”, ofstream::out);
用构造函数： ofstream outfile（“file2”）; 或 ofstream outfile（“file2” , ofstream::out);

打开输入输出文件

用open函数： fstream outfile;
outfile.open（“file2”）; 或 outfile.open（“file2”, fstream::in | fstream::out);
用构造函数： fstream outfile（“file2”）;
或 fstream outfile（“file2” , fstream::in | fstream::out);

文件关闭

- 用成员函数close
- 对于输出文件，close会将缓冲区中的内容写入文件
- 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

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O
  - 流式文件
  - 文件的顺序访问
  - 文件的随机访问
  - 访问有记录概念的文件

文件的顺序读写

读文件

- 从文件的第一个字节开始依次往下读，读取一定量的数据或读到文件结束

写文件

- 在一个空文件中依次写数据
- 在一个文件后面依次添加数据

ASCII文件的顺序访问

与控制台读写完全一样

- 读文件：>>、get、getline、read
- 写文件：<<、put、write
- 支持格式化读写

判断文件结束

- >>读：可以通过判断输入流对象值是否为0
- get读：判断读入字符是否是EOF，即end of file
- 其他方式读：通过成员函数eof
- 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。

ASCII文件访问实例

要求

- 将数字1到10写入文件file。文件中，数字与数字之间用空格分开
- 然后从file中读取这些数据，把它们显示在屏幕上

设计思想

- 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
- 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
- 再用输入方式打开文件file
- 读出所有数据，并输出到屏幕上

ASCII文件访问实例

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream out("file");
     ifstream in;
     int i;

     if (!out) {   cout << "create file error\n";    }
     for (i = 1; i <= 10; ++i)      out << i << ' ';
     out.close();

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

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

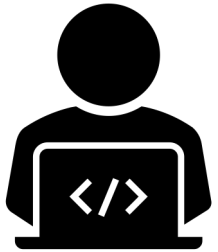
1 2 3 4 5 6 7 8 9 10

包含各种类型数据的ASCII文件操作

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
   ofstream fout("test");

   if (!fout){cout << “cannot open output file\n”；}
   fout<<10<<" "<<123.456<< "\"This is a text file\"\n";
   fout.close();
   return 0;
}
```

执行结果

文件中的内容为
10 123.456"This is a text file“
显示器无信息显示

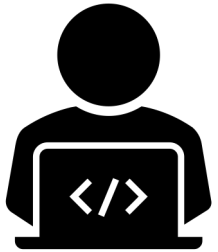


代码演示
ascii-file.cpp

读ASCII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



代码演示
ascii-file-read.cpp

二进制文件的读写

- **读文件：read函数**

  obj.read(char *s, int size);

- **写文件：write函数**

  obj.write(const char *s, int size);

- **判读文件结束：eof函数**

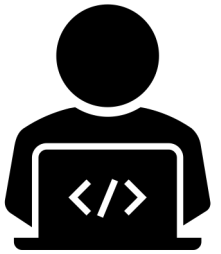
  obj.eof()

二进制文件访问实例

- **要求**
- 将数字1到10写入二进制文件file，然后从file中读取这些数据，把它们显示在屏幕上。

- **设计思想**
- 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write函数入文件，并关闭文件。
- 再用输入方式打开文件file
- 用read读出所有数据，并输出到屏幕，直到eof为真

二进制文件的顺序读写



代码演示

chapter14-file-example.cpp

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

int main()
{
   ofstream out("d:\\file", ofstream::binary);
   ifstream in;
   int i;

  //将1～10写到输出流对象
   if (!out) {  cerr << "create file error\n"; return 1;  }
   for (i = 1; i <= 10; ++i)
        out.write(reinterpret_cast<char *> (&i),  sizeof(int));
   out.close();
```

二进制文件的顺序读写

```
    in.open("d:\\file", ifstream::binary);
    if (!in) {  cerr << "open file error\n"; return 1;  }
    in.read(reinterpret_cast<char *> (&i), sizeof(int));
    while (!in.eof()) {
      cout << i << ' ';
        in.read(reinterpret_cast<char *> (&i), sizeof(int));
    }
  in.close();
  cout << endl;

    return 0;
}
```

执行该程序后，文件file中的内容为

1 2 3 4 5 6 7 8 9 10

文件大小是40字节

该程序的输出结果是

1 2 3 4 5 6 7 8 9 10

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O
  - 流式文件
  - 文件的顺序访问
  - 文件的随机访问
  - 访问有记录概念的文件

文件随机访问

- **随机访问**
- 读写文件中任意位置上的数据

- **实现手段**
- 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。

文件定位指针的操作

- **获取文件定位指针的当前位置**
- tellg：返回读文件指针值
- tellp：返回写文件指针值

- **设置文件定位指针的位置**
- seekg：设置读文件指针值，即get
- seekp：设置写文件指针值，即put

seekg和seekp

- **参数**
- 为long类型的整数，表示偏移量
- 指定移动的起始位置

- **起始位置**
- 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

如

```
fileObject.seekg( n );
fileObject.seekg( n, ios::cur );
fileObject.seekg( y, ios::end );
fileObject.seekg( 0, ios::end );
```

ASCII文件的随机读写实例

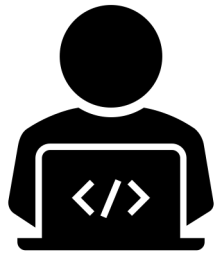
```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream in("file");
    int i;

   if (!in) {cerr << "open file error\n"; return 1;}
   in.seekp(10);
   in << 20; //整数 20 会被写入到文件的第 11 个字节位置
   in.seekg(0); // 设置读取指针：文件开头（重置读取位置）
   while (in >> i)
       cout << i << ‘\n ';
   in.close();

   return 0;
}
```

执行前文件内容：**1 2 3 4 5 6 7 8 9 10**

执行后文件内容：**1 2 3 4 5 207 8 9 10**

显示器上显示

1
2
3
4
5
207
8
9
10



代码演示

chapter14-file-random-example.cpp

二进制文件的随机读写

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

int main()
{
    fstream io(“d:\\file”, fstream::binary)
    int i;

  // 改写6为20
  io.seekp(5 * sizeof(int));
  i = 20;
    io.write(reinterpret_cast<char *> (&i), sizeof(int));
```

```
// 重新读文件
  io.seekg(0);
  io.read(reinterpret_cast<char *> (&i), sizeof(int));
  while (!io.eof()) {
      cout << i << ‘\n';
        io.read(reinterpret_cast<char *> (&i),  sizeof(int));
  }
  io.close();

  return 0;
}
```

显示器上显示
1
2
3
4
5
20
7
8
9
10

本章架构

- 流与标准库
- 输入输出缓冲
- 基于控制台的I/O
- 基于文件的I/O
  - 流式文件
  - 文件的顺序访问
  - 文件的随机访问
  - 访问有记录概念的文件

实例：图书馆的书目管理系统

- **每本书需要记录的信息**
- 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。

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

文件设计

需要长期保存的信息

- 每本书的信息
- 一共有多少本书
- 一般情况下，一个文件保存一类信息。但第二个信息只是一个数字，一个文件保存一个数字有点浪费。于是合并为一个文件。

能否不保存书的数量?

第一本书的信息

第二本书的信息

第三本书的信息

第四本书的信息

……

……

……

……

最后一本书的信息

book类设计

- **数据成员**
- 馆藏号、书名、借书标记

- **成员函数**
- 构造函数
- 借书
- 还书
- 显示书的详细信息

book.h文件

```
#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;
public:
    Book(const char *s = "", int total_no = 0) ;
    void borrow(int readerNo);
    void Return();
    void display() const;
};
#endif
```

book类设计

```
Book::Book(const char *s，int totalNo)
{     no = totalNo;   borrowed = 0;   strcpy(name,s); }

void Book::borrow(int readerNo)
{
    if (borrowed != 0) cerr << "本书已被借，不能重复借\n";
    else borrowed = readerNo;
}

void Book::Return()
{
     if (borrowed == 0) cerr << "本书没有被借，不能还\n";
    else  borrowed = 0;
 }

void Book::display() const
{
   cout << setw(10) << no << setw(20) << name  << setw(10) << borrowed << endl;
}
```

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

Main函数

```
#include "book.h"
void initialize();
void addBook();
void borrowBook();
void returnBook();
void displayBook();

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");

    outfile.close();
}
```

addBook的实现

```
void addBook()
{
      char ch[20];
      Book *bp;
      ofstream outfile("book", ofstream::app | ofstream::ate | ofstream::binary);
            long int no = outfile.tellp() / sizeof(Book) + 1 ;
     cout << "请输入书名：  ";
    cin >> ch;
    bp = new Book(ch, no);

    outfile.write( reinterpret_cast<const char *>(bp), sizeof(*bp));
   delete bp;
   outfile.close();
}
```

borrowBook

```
void borrowBook()
{
   int bookNo, readerNo;
   fstream iofile("book",  fstream::binary);
   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", ofstream::binary );
    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", ofstream::binary);
   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
  - 字符串I/O
- **文件I/O是从控制台I/O类继承的，因此，操作方式是相同的。**