高级语言C++程序设计
Lecture 12 输入输出流

南开大学 计算机学院

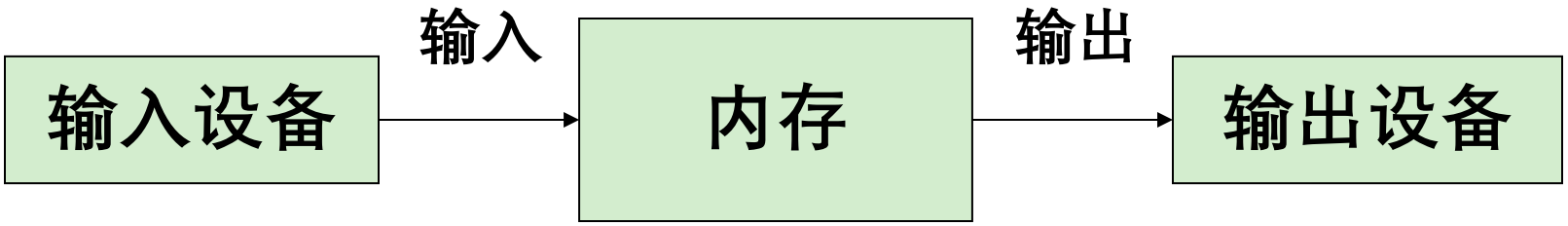
2022

概述

- 计算机所做的任何数据处理工作都是在内存中进行
- 输入是指将数据从输入设备传送到内存的过程，输出是指将数据从内存传送到输出设备的过程

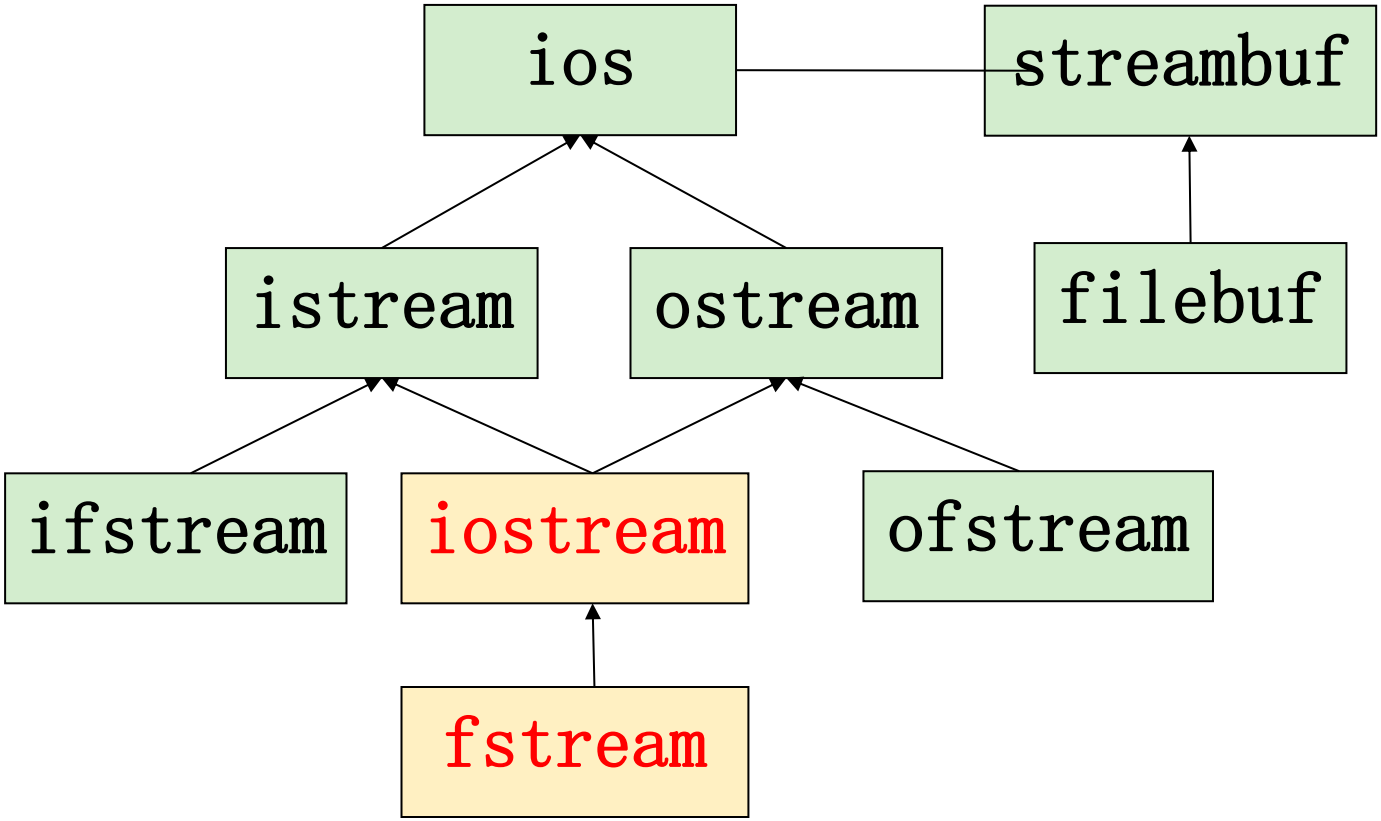
输入举例：从键盘读取数据、从磁盘上的文件读取数据

输出举例：向屏幕输出数据、将数据写入磁盘上的文件



概述

C++定义了一系列类和类对象来帮助用户实现输入和输出，称为**输入输出流**



概述

基本流类

- ios
  - 基本流类的基类
- istream
  - 由ios派生，支持输入(提取“>>”)操作
- ostream
  - 由ios派生，支持输出(插入“<<”)操作
- iostream
  - 由istream与ostream共同派生，支持输入和输出双向操作

概述

用于磁盘操作的文件流类

- ifstream
  - 由istream派生，支持从磁盘文件中输入(读)数据
- ofstream
  - 由ostream派生，支持往磁盘文件中输出(写)数据
- fstream
  - 由iostream派生，支持对磁盘文件进行输入和输出数据的双向操作

标准输入输出

标准输入输出流对象

键盘是标准输入设备，屏幕是标准输出设备

C++预定义了4个标准流对象：cin、cout、cerr、clog，均包含于头文件iostream中

- cin是istream类的对象，用于处理标准输入（即从键盘输入数据到内存中）
- cout是ostream类的对象，用于处理标准输出（即将内存中的数据输出到屏幕上）
- cerr和clog都是ostream类的对象，均用于处理错误信息标准输出

插入操作符

使用“<<”运算符进行标准输出

- 系统提供的ostream类已经对“<<”进行了多次重载，能够实现对不同类型数据的输出操作：

```
ostream& operator<<(int);
// 将内存中的一个int型数据输出到输出设备
ostream& operator<<(double);
// 将内存中的一个double型数据输出到输出设备
ostream& operator<<(char*);
 // 将内存中的一个字符串数据输出到输出设备
```

插入操作符

使用“<<”运算符进行标准输出

```
int x = 10;
double y = 3.5;
char z[] = "C++";

cout<<x;  //cout.operator<<(x);
cout<<y;  //cout.operator<<(y);
cout<<z;  //cout.operator<<(z);
```

重载函数operator<<()以引用方式返回调用该函数时所使用的对象，因此能够连续输出，可以将上面3条语句合为一条

```
cout<<x<<y<<z;
```

提取操作符

使用“>>”运算符进行标准输入

- 系统提供的ostream类已经对“>>”进行了多次重载，能够实现对不同类型数据的输出操作：

```
istream& operator>>(int);
// 提取一个int型数据到内存
istream& operator>>(double);
// 提取一个double型数据到内存
istream& operator(char*);
 // 提取一个字符串到内存
```

提取操作符

使用“>>”运算符进行标准输入

```
int x;
double y;
char z[];

cin>>x;   //cin.operator>>(x);
cin>>y;    //cin.operator>>(y);
cin>>z;    //cin.operator>>(z);
```

重载函数operator>>()以引用方式返回调用该函数时所使用的对象，因此能够连续提取，可以将上面3条语句合为一条

```
cin>>x>>y>>z;
```

使用<<和>>操作用户自定义类型

- 插入与提取运算符（<<和>>）无法直接实现用户自定义的类对象进行输入输出操作
  - 需要在类complex的定义中，对运算符<<进行重载，使其实现输出复数的功能
  - 提取运算符>>同样需要进行重载实现输入功能

```
complex x;
cout<<x; //无法实现complex类的输出
```

使用<<和>>操作用户自定义类型

重载插入与提取运算符，实现复数的输入和输出

```
#include<fstream.h>
class complex {
   double r;
   double i;
public:
   complex(double r0=0, double i0=0) {
       r=r0;i=i0;
   }
   complex operator +(complex c2);
   complex operator *(complex c2);
   friend istream& operator>>(istream& in, complex& com);
   friend ostream& operator<<(ostream& out,complex com);
};
```

插入和提取操作符

```
complex complex::operator +(complex c2) {
   complex c;
   c.r=r+c2.r;
   c.i=i+c2.i;
   return c;
}
complex complex::operator * (complex c2) {
   complex temp;
   temp.r=(r*c2.r)-(i*c2.i);
   temp.i=(r*c2.i)+(i*c2.r);
   return temp;
}
istream& operator >> (istream& in, complex& com) {
   in>>com.r>>com.i;
   return in;  //不可缺少，因为函数返回类型为“istream&”
}
```

插入和提取操作符

```
ostream& operator << (ostream& out, complex com) {
   out<<"("<<com.r<<", "<<com.i<<")"<<endl;
   return out; //不可缺少，因为函数返回类型为“ostream&”
}

void main() {
   complex c1(1,1), c2(2,3), c3, res;
   cout<<"c1="<<c1; //等价于operator<<(cout, c1);
   res = c1+c2;
   cout<<"c1+c2="<<res;
   cout<<"c1*c2="<<c1*c2;
   cout<<"Input c3:";
   cin>>c3; //等价于operator>>(cin, c3);
   cout<<"c3+c3="<<c3+c3;
}
```

程序结果：

```
c1=(1, 1)
c1+c2=(3, 4)
c1*c2=(-1, 5)
Input c3:3 -5
c3+c3=(6, -10)
```

输入输出流函数：put()

输出一个字符（字符型参数），格式为：

```
ostream& put( char ch )
```

```
char ch='a';
cout.put(ch);
//在屏幕上输出变量ch中存储的字符'a'
cout.put('b').put('c')<<endl;
//因为返回引用，可以连续调用
```

输入输出流函数：get()

提取一个字符（存放在字符型参数），格式为：

```
istream& get( char& rch );
```

```
char ch, ch1;
cin.get(ch);
//从键盘上读入字符，并存放在字符变量ch中
cin.get(ch).get(ch1)<<endl;
//连续从键盘上读入两个字符，放到ch和ch1中
```

get()函数和运算符“>>”的区别在于：

get()函数能够读取空格、回车符和制表符等空白字符，而“>>”只能读取非空白字符

输入输出流函数：getline()

从输入设备读入一行，格式为：

```
istream& getline(char* pch, int n,
char delim = '\n');
```

- 从输入设备读取n-1个字符，并在其后加上字符串结束符'\0'，存入第1个参数所指向的内存空间中
- 若在读取够n-1个字符前遇到由第3个参数指定的终止符，则提前结束读取，终止符的默认值是`\n`
- 若读取成功，函数返回值为真（非0）值，若读取失败（遇到文件结束符EOF），函数返回值为假（0）值

getline()能够读取空格、回车符和制表符等空白字符

输入输出流函数：getline()

```
int main() {
      int len, maxlen=0;
      char s[80], t[80];
      while (cin.getline(s,80)) {
      //Ctrl+Z相当于文件结束符EOF，可以让getline返回0
            len=(int)strlen(s);
            if (len>maxlen){
                  maxlen=len;
                  strcpy(t, s);
            }
      }
      cout<<"长度最大的字符串是:"<<t<<endl;
      cout<<"其长度为:"<<maxlen<<endl;
      return 0;
}
```

输入输出流函数：getline()

另一种写法：

```
istream& getline(istream &is, string
&str, char delim='\n');
```

- 从输入设备读取字符串，放在str中，直到遇到结尾符delim；或者读到文件末尾
- 若读取成功，函数返回值为真（非0）值，若读取失败（遇到文件结束符EOF），函数返回值为假（0）值

输入输出流函数：getline()

```
int main(){
    string name;
    string city;
    cout << "Please enter your name: ";
    getline(cin, name);
    cout << "Enter the city: ";
    getline(cin, city);
    cout << "Hello, " << name << endl;
    cout << "You live " << city << endl;
    return 0;
}
```

输入输出格式

格式控制标志字

ios类中定义了一批公有的格式控制标志来控制格式

状态标志	值	含义
skipws	**0X0001**	跳过输入中的空白
left	**0X0002**	左对齐输出
right	**0X0004**	右对齐输出
internal	**0X0008**	在符号位和基指示符后填入字符
dec	**0X0010**	转换基制为十进制
oct	**0X0020**	转换基制为八进制
hex	**0X0040**	转换基制为十六进制
showbase	**0X0080**	在输出中显示基指示符
showpoint	**0X0100**	输出时显示小数点
uppercase	**0X0200**	十六进制输出时一律用大写字母
showpos	**0X0400**	正整数前加“+”号

格式控制标志字

ios类中定义了一批公有的格式控制标志来控制格式

状态标志	值	含义
scientific	**0X0800**	科学示数法显示浮点数
fixed	**0X1000**	定点形式显示浮点数
unitbuf	**0X2000**	输出操作后立即刷新流
stdio	**0X4000**	输出操作后刷新**stdout** 和**stdree**

- 每个格式标志用一个long型常数表示，且只有一个bit是1，其余bit都是0
- 多个标志位可以组合，例如，ios::fixed | ios::left 表示以定点形式左对齐输出浮点数

通过ios类的公有函数控制I/O格式

控制I/O格式的第一种方式是使用ios类的公有函数

函数	功能
long flags(long lFlags); long flags() const;	用参数Flags更新标志字 返回标志字
long setf(long lFlags); long setf(long lFlgas, long lMask);	设置lFlags指定的标志位 将lMask指定的位清0，然后设置lFlags制定位
long unsetf(long lMask);	将参数lMask制定的标志位清0
int width(int nw);	设置下一个输出项的显示宽度为nw
char fill(char cFill);	空白位置以字符参数cFill填充
int precision(int np);	用参数np设置数据显示精度

通过ios类的公有函数控制I/O格式

```
int main() {
    char *s = "Hello";
    cout.fill('*');  // 置填充符
    cout.width(10);  // 置输出宽度
    cout.setf(ios :: left); // 左对齐
    cout << s << endl;
    cout.width(15); // 置输出宽度
    cout.setf(ios :: right, ios ::
left);
    // 清除左对齐标志位，置右对齐
    cout << s << endl;
    return 0;
}
```

```
Hello*****
**********Hello
```

通过ios类的公有函数控制I/O格式

```
int main() {
    float a = 123.4567;
    cout.flags(ios::right | ios::fixed);
    // 置右对齐、定点输出，默认保留6位小数
    cout<< a << endl;
    cout.flags(ios::left |
ios::scientific);
    // 置左对齐、科学计数法
    cout.precision(2); //保留2位小数
    cout << a << endl;
    return 0;
}
```

```
123.456700
1.23e+002
```

通过ios类的公有函数控制I/O格式

```
int main() {
    int a , b , c ;
    cin.setf(ios::dec, ios::basefield) ;
    cin>>a ; //改为十进制输入，之前的进制清除
    cin.setf(ios::hex, ios::basefield) ;
    cin>>b ; //十六进制输入
    cin.setf(ios::oct, ios::basefield) ;
    cin>>c ; //八进制输入
    cout.setf(ios::dec, ios::basefield);
    cout<<a<<" "<<b<<" "<<c<<endl; //十进制输出
    cout.setf(ios::hex, ios::basefield);
    cout<<a<<" "<<b<<" "<<c<<endl; //十六进制输出
    cout.setf(ios::oct, ios::basefield);
    cout<<a<<" "<<b<<" "<<c<<endl; //八进制输出
    return 0;
}
```

通过格式控制符控制I/O格式

控制I/O格式的第二种方式是使用格式控制符

控制符作为重载插入运算符<<或提取运算符>>的右操作数控制I/O格式

iostream的控制符	功能
endl	输出一个新行符，并清空流
ends	输出一个空格符，并清空流
flush	清空流缓冲区
dec	十进制输入或输出
hex	十六进制输入或输出
oct	八进制输入或输出
ws	提取空白字符

通过格式控制符控制I/O格式

控制I/O格式的第二种方式是使用格式控制符

控制符作为重载插入运算符<<或提取运算符>>的右操作数控制I/O格式

iomanip的控制符	功能
resetiosflags(ios::lFlags)	清除lFlags指定的标志位
setiosflags(ios::lFlags)	设置lFlags指定的标志位
setbase(int base)	设置基数,base=8, 10, 16
setfill(char c)	设置填充符c
setprecision(int n)	设置浮点数输出精度
setw(int n)	设置输出宽度

通过格式控制符控制I/O格式

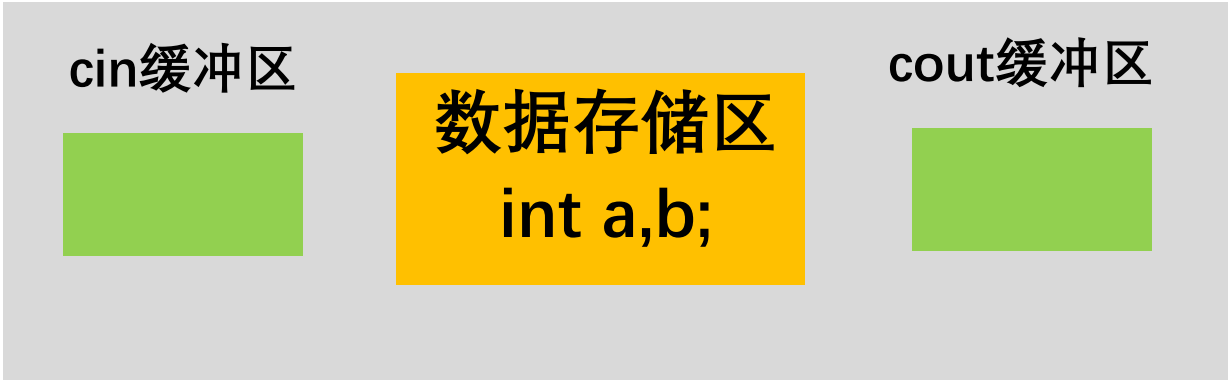
```
#include <iomanip>
void main() {
    cout.width(6); //只管随后一个数的域宽，默认右对齐
    cout<<4785<<27.4272<<endl; //□□478527.4272
    cout<<setw(6)<<4785<<setw(8)<<27.4272<<endl;
    //□□4785□27.4272
    cout.width(6);
    cout.precision(3);
    /*当格式为ios::scientific或ios::fixed时，浮点数精度np指
    小数点后的位数，否则指有效数字；此例中没有fixed或
    者scientific，所以指的是有效数字为3*/
    cout<<4785<<setw(8)<<27.4272<<endl;
    //□□4785□□□□27.4
```

通过格式控制符控制I/O格式

```
    cout<<setw(6)<<4785<<setw(8)<<setprecision(2);
    cout<<27.4272<<endl<<endl;
    //□□4785□□□□□□27
    //setprecision(2)设置浮点数的有效数字
    cout.setf(ios::fixed, ios::floatfield);
    //今后以定点格式显示浮点数(无指数部分)
    cout.width(6);
    cout.precision(3);
    //当格式为ios::fixed时，设置小数点后的位数
    cout<<4785<<setw(8)<<27.4272<<endl;
    //□□4785□□27.427
}
```

cin/cout缓冲区

缓冲区：内存空间的一部分，用来缓冲输入或者输入的数据

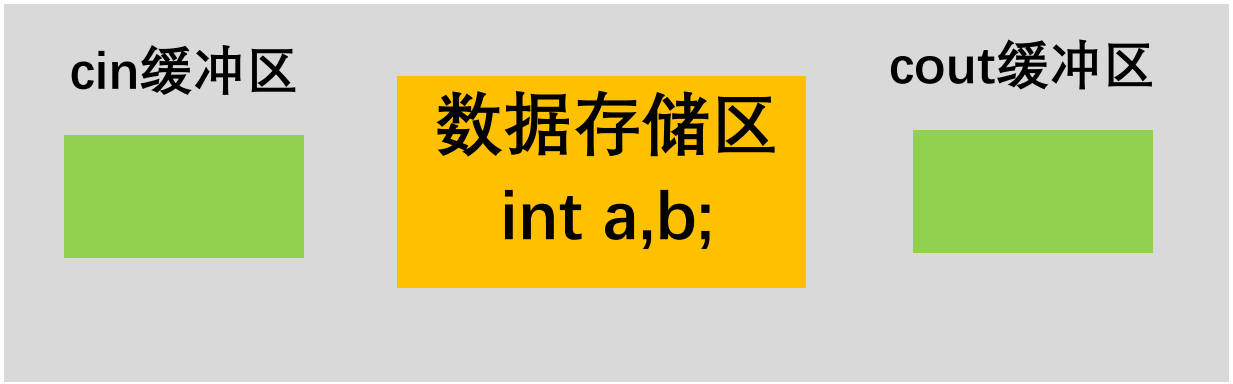


键盘

显示器

cin>>a>>b；如何执行？

cin/cout缓冲区



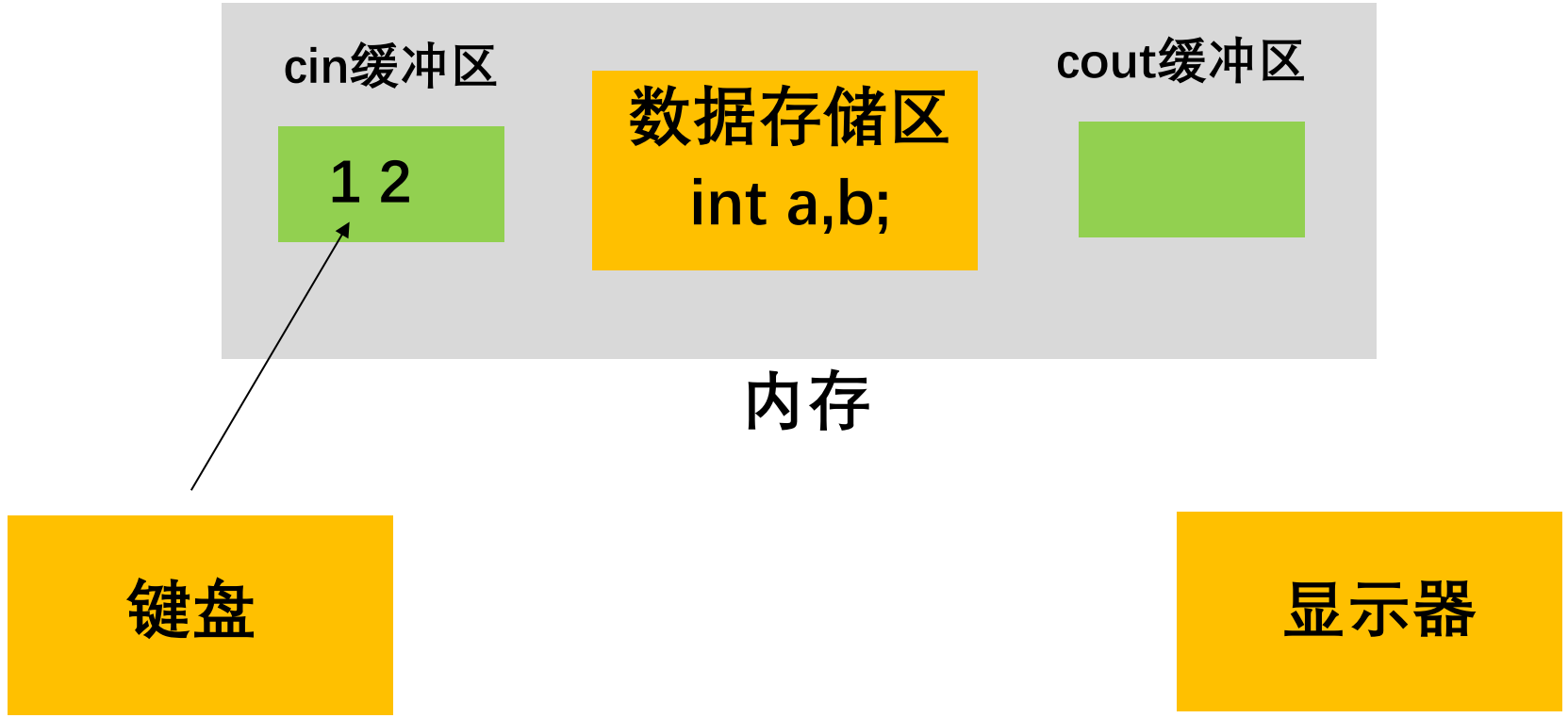
内存

键盘
1 2

显示器

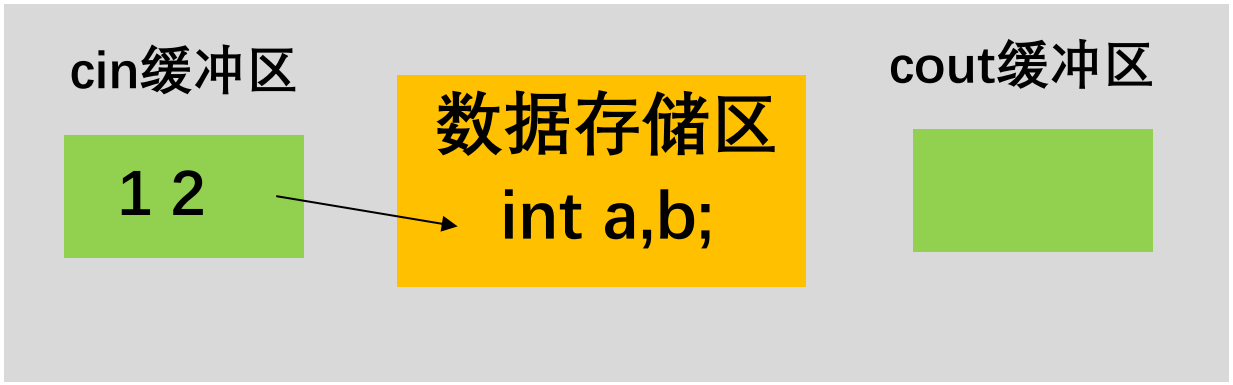
步骤一：键盘输入1 2两个整数，中间空格隔开

cin/cout缓冲区



步骤二：输入回车，1 2被送入缓冲区

cin/cout缓冲区



内存

键盘

显示器

步骤三：cin从缓冲区读取数据，赋值给a和b

(1)读取整数1（以空格/回车/Tab作为分隔符），赋值给a

(2)读取整数2，赋值给b

cin/cout缓冲区

```
int main(){
    int a;
    int i = 1;
    while (cin >> a) {
        cout << a <<endl;
        i++;
    if (i == 5)
        break;
    }
    char ch;
    cin >> ch;
    cout << "ch: " << ch;
    return 0;
}
```

当输入1 2 3 4 5并回车时，程序输出？

(1)缓冲区中1 2 3 4 5

(2)while循环只读取了前4个数字

(3)cin>>ch并没有等待用户输入，直接从缓冲区读取了第5个数字

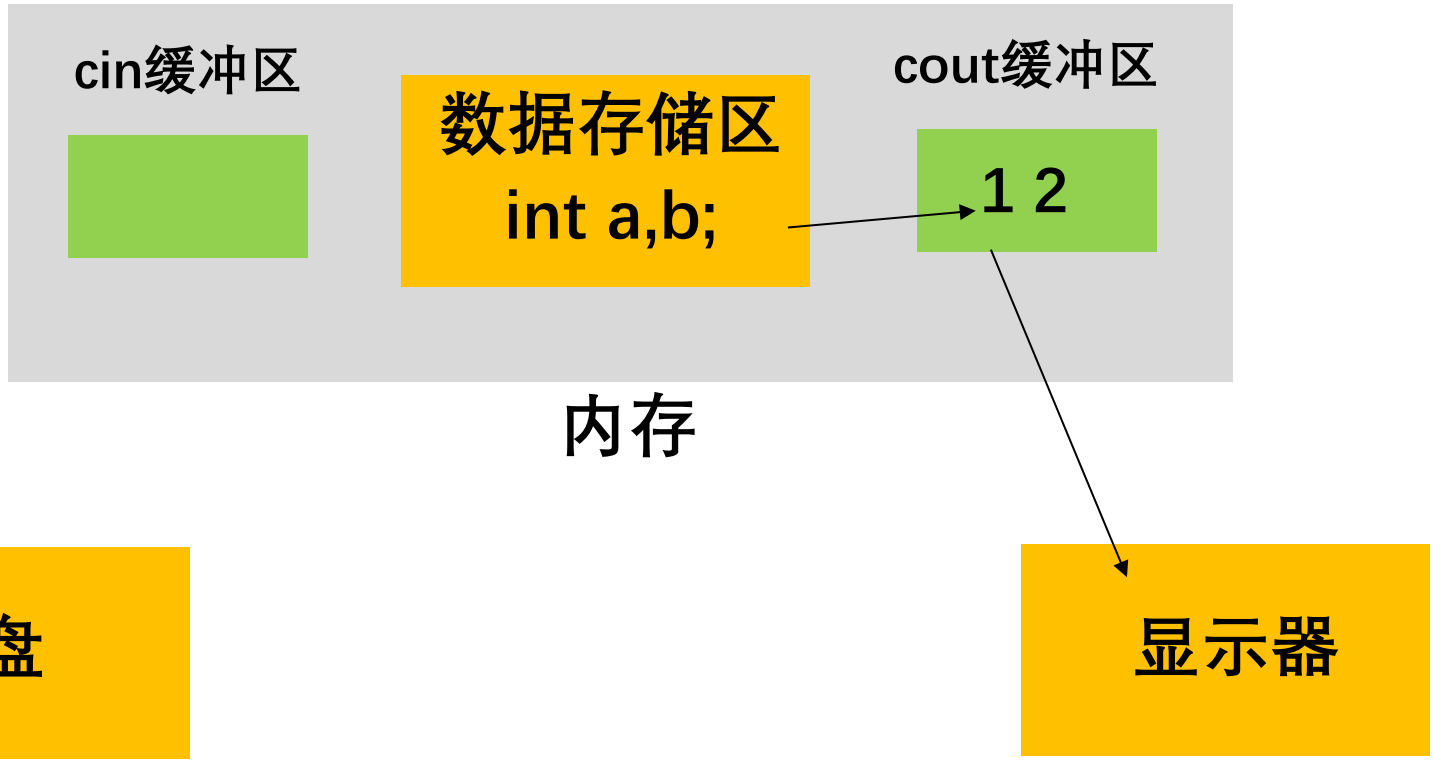
cin/cout缓冲区

```
int main(){
    int a, i = 1;
    while (cin >> a) {
        cout << a <<endl;
        i++;
        if (i == 5) break;
    }
    cin.clear();
    cin.ignore(numeric_limits<std::
streamsize>::max(), '\n');
    char ch;
    cin >> ch;
    cout << "ch: " << ch;
    return 0;
}
```

解决方案

下次cin之前清空缓冲区

cin/cout缓冲区



cout<<a<<b<<endl;

(1)先将a,b的值放到缓冲区

(2)碰到endl刷新缓冲区，输出到显示器（很多编译器不用endl也可以）

磁盘文件输入输出

文件流类对象

- C++通过文件流类实现对磁盘文件的输入输出
- C++中没有预定义的文件流类的对象，用户需要为每个文件自己定义文件流类对象

自定义文件输出流对象，将字符串写入磁盘文件

```
#include<fstream>   //引入头文件<fstream>
using namespace std;
int main() {
    ofstream outfile("myfile.txt");
    //定义文件输出流类对象，并与"myfile.txt"关联
    outfile<<"Write to file"; //写入文件
    outfile.close(); //关闭文件
    return 0;
}
```

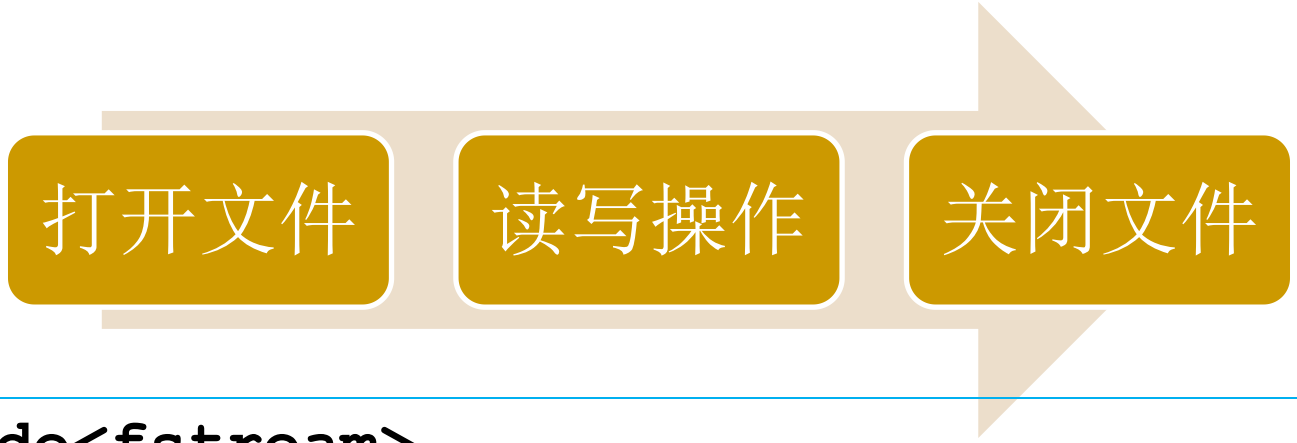
文件流类对象

- C++通过文件流类实现对磁盘文件的输入输出
- C++中没有预定义的文件流类的对象，用户需要为每个文件自己定义文件流类对象

自定义文件输入流对象，实现从磁盘文件读一个整数

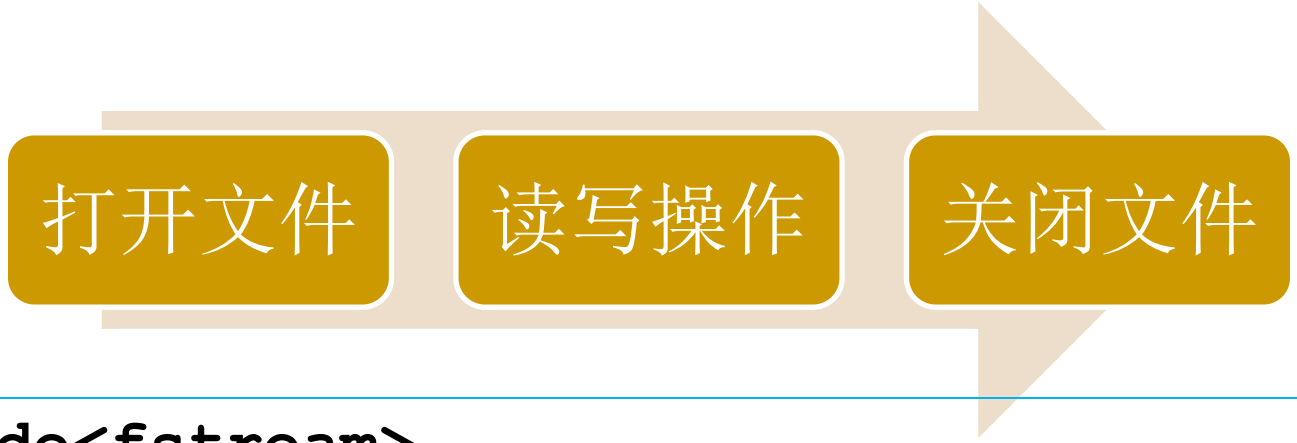
```
#include<fstream>   //引入头文件<fstream>
using namespace std;
int main() {
    int x = 2;
    ifstream infile("myfile.txt"); //定义文件输入流类对象，并与"myfile.txt"关联
    infile>>x; //从文件读入一个整数
    infile.close(); //关闭文件
    return 0;
}
```

读写磁盘文件的一般处理过程




```
#include<fstream>
using namespace std;
int main() {
    int x = 2;
    ifstream infile("myfile.txt");
    //通过构造函数自动打开文件
    infile>>x; //读文件操作
    infile.close(); //关闭文件
    return 0;
}
```

读写磁盘文件的一般处理过程




```
#include<fstream>
using namespace std;
int main() {
    ofstream outfile;
    outfile.open("myfile.txt");
    //通过open函数显示地打开文件
    outfile<<"Write to file"; //写文件操作
    outfile.close();//关闭文件
    return 0;
}
```

文件流类的构造函数

文件输出流 ofstream

szName -- 文件名

```
ofstream(const char* szName, int nMode=ios::out);
```

文件流类的构造函数

文件输出流 ofstream

nMode -- 文件打开方式

```
ofstream(const char* szName, int nMode=ios::out);
```

方式	作用
ios::out	以输出方式打开文件，对文件进行写操作
ios::app	以追加方式打开文件，所有输出附加在文件末尾
ios::ate	打开文件时，文件指针定位在文件尾
ios::binary	以二进制方式打开文件，缺省的方式是文本方式
ios::trunc	如果文件已存在，则先删除文件内容

默认是ios::out，多种mode可以用位或运算符（即“|”）连接，如“ios::out | ios::binary”

文件流类对象

```
int main() {
    ofstream outfile("myfile.txt",ios::out);
    outfile<<"Write to file";//首次打开文件，创
    outfile.close();                建新文件

    ofstream outfile1("myfile.txt",ios::app);
    outfile1<<"Hello"; //打开已有文件，保留原有内
    outfile1.close();   容，以追加方式写文件

    ofstream outfile2("myfile.txt");
    outfile2<<"Hello";//打开已有文件，原有内容清空
    outfile2.close();

    return 0;
}
```

文件流类的构造函数

文件输入流 ifstream

szName -- 文件名

文件流类的构造函数

文件输入流 ifstream

nMode -- 文件打开方式

```
ifstream(const char* szName, int nMode=ios::in);
```

方式	作用
ios::in	以输入方式打开文件，对文件进行读操作，该文件必须存在
ios::binary	以二进制方式打开文件，缺省的方式是文本方式
ios::ate	打开文件时，文件指针定位在文件尾

默认是ios::in，多种mode可以用位或运算符（即“|”）连接，如“ios::in | ios::binary”

文件流类对象

```
int main() {
    int x1 = 0, x2 = 0;
    ofstream outfile("myfile.txt",ios::out);
    outfile<<16; //写入16
    outfile.close();
    //打开文件，默认从文件头开始读
    ifstream infile("myfile.txt",ios::in);
    infile>>x1;
    infile.close();
    //打开文件，从文件尾开始读
    ifstream infile2("myfile.txt",ios::in |
ios::ate);
    infile2>>x2;//已经到文件尾，无法读出16
    infile2.close();
    return 0;
}
```

文件流类的构造函数

文件输入输出流 fstream

szName -- 文件名

```
fstream(const char* szName, int nMode = ios::in |
                            ios::out);
```

fstream 由 ifstream 和 ofstream 派生，即支持文件读又支持文件写！

文件流类的构造函数

文件输入输出流 fstream

nMode -- 文件打开方式

```
fstream(const char* szName, int nMode = ios::in |
                            ios::out);
```

方式	作用
ios::in	以输入方式打开文件，对文件进行读操作，该文件必须存在
ios::out	以输出方式打开文件，对文件进行写操作
ios::app	以追加方式打开文件，所有输出附加在文件末尾
ios::ate	打开文件时，文件指针定位在文件尾
ios::binary	以二进制方式打开文件，缺省的方式是文本方式
ios::trunc	如果文件已存在，则先删除文件内容

默认是ios::in | ios::out，多种mode可以用位或运算符（即“|”）连接，如“ios::in | ios::out | ios::ate”

文件流类对象

```
#include<fstream>                          //假设文件原内容为：12 123
#include<iostream>
using namespace std;
int main() {
    int x1 = 0, x2 = 0;
    fstream file1("myfile.txt",ios::out |
ios::in | ios::app); //打开文件，即可以读又可以写

    file1>>x1;   //从文件读入12，放入x1
    file1<<x2;   //在文件尾写入x2
    file1.close();

    return 0;
}
```

文件流类的构造函数

一些规则

ios::app　　以追加的方式打开文件
ios::ate　　打开文件时将文件指针放在文件末尾

ios::app一般只和ios::out配合，用于以追加的方式打开输入流
fstream fs(“aaa.txt”, ios::out | ios::app)

ios::ate一般和ios::in相配合，打开文件时将文件指针定位到文件尾
fstream fs(“aaa.txt”, ios::in | ios::ate);

如果ios::ate和ios::out相结合，将清空原文件
fstream fs(“aaa.txt”, ios::out | ios::ate);

文件流类的构造函数

文件不存在时，是否创建新的文件

fstream fs(“aaa.txt”, ios::in)　　不创建
fstream fs(“aaa.txt”, ios::out)　　创建
fstream fs(“aaa.txt”, ios::trunc)　　不创建, trunc表示清空旧文件
fstream fs(“aaa.txt”);　不创建
fstream fs("aaa.txt", ios::in | ios::out)　不创建
fstream fs("aaa.txt", ios::in | ios::trunc);　不创建
fstream fs("aaa.txt", ios::out | ios::trunc);　创建
fstream fs("aaa.txt", ios::in | ios::out | ios::trunc); 创建

以上总结：
fstream 在文件不存在时创建新文件的条件：
要么单独使用 ios::out
要么同时使用 ios::out | ios::trunc

文件流类的open成员函数

ofstream::open

```
void open(const char* szName,  int nMode =
ios::out);
```

ifstream::open

```
void open( const char* szName,  int nMode =
ios::in);
```

fstream::open

```
void open( const char* szName,  int nMode =
ios::in | ios::out);
```

其中szName和nMode的含义与文件流类构造函数中的参数相同

文件流类的open成员函数

ofstream outfile("myfile.txt")

```
#include<fstream>
using namespace std;
int main() {
    ofstream outfile; //定义文件输出流类对象
    outfile.open("myfile.txt");//打开文件
    if(!outfile.is_open()) { //判断是否打开成功
          cout<<"打开文件失败！"<<endl;
          return 0;
    }

    outfile<<"Write to file"; //写入文件
    outfile.close();//关闭文件
    return 0;
}
```

使用插入运算符<<写文件

```
#include<fstream>
using namespace std;
int main() {
    int x = 1;
    double y = 2.3;
    char z = '3';                    //以写方式打开文件
    ofstream outfile("file.txt", ios::out);
    outfile<<x;  //向文件写入x
    outfile<<endl;
    outfile<<y; //向文件写入y
    outfile<<endl;
    outfile<<z; //向文件写入z
    outfile.close();
    return 0;
}
```

文件流类可以使用插入运算符写文件

使用提取运算符>>读文件

```
#include<fstream>
using namespace std;
int main() {
    int x;
    double y;
    char z;                          //以读方式打开文件
    fstream infile("file.txt", ios::in);
    infile>>x; //从文件中读取整数放入x
    infile>>y; //从文件中读取double型数据放入y
    infile>>z; //从文件中读取char型数据放入y
    infile.close();
    return 0;
}
```

文件流类可以使用提取运算符读文件

使用<<和>>读写文件

文件流类可以使用插入运算符<<和提取运算符>>读写文件

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
      fstream outfile; //使用fstream类对象
      //以写方式打开文本文件digit.txt
      outfile.open("digit.txt", ios::out);
      if(!outfile.is_open()) { //判断是否打开成功
            cout<<"打开digit.txt文件失败！"<<endl;
            return 0;
      }
```

使用<<和>>读写文件

```
char digit='0';
//循环将10个数字字符输出到文件digit.txt中
while (digit<='9'){
    outfile<<digit<<' ';//将digit写入文件
    digit++;
 }
 outfile.close(); //关闭文件

 //以读方式打开文本文件
 fstream infile("digit.txt", ios::in);
 if (!infile.is_open()){ //判断是否打开成功
       cout<<"打开digit.txt文件失败！"<<endl;
       return 0;
 }
```

使用<<和>>读写文件

```
    // 循环从文件中读取数字字符到变量digit中
    while (infile>>digit)
        cout<<digit<<' ';
    cout<<endl;
    infile.close();
    return 0;
}
```

使用<<和>>读写文件

```
#include <fstream.h>
void main() {
      //1) 往文件写数据
      ofstream outfile1("myfile1.txt");
      //以“写”方式打开
      outfile1<<"Hello!...CHINA!
Nankai_University"<<endl;
      outfile1.close();
      //2) 往文件尾部追加数据
      outfile1.open("myfile1.txt", ios::app);
      //以“追加”方式打开
      int x=1212, y=6868;
      outfile1<<x<<" "<<y<<endl;
      //注意，数据间要人为添加分割符空格
      outfile1.close();
```

使用<<和>>读写文件

```
    //3) 从文件读出数据并显示在屏幕上
    char str1[80], str2[80];
    int x2,y2;
    ifstream infile1("myfile1.txt");
    //以读方式打开
    infile1>>str1>>str2;
    //使用“>>”读(遇空格、换行均结束)
    infile1>>x2>>y2;
    infile1.close();
    cout<<"str1="<<str1<<endl;
    cout<<"str2="<<str2<<endl;
    cout<<"x2="<<x2<<endl;
    cout<<"y2="<<y2<<endl;
}
```

使用put和get函数读写文件

文件流类可以使用put和get函数读写文件

```
#include <fstream.h>
#include <iostream>
//键盘输入字符串，通过put将其写到自定义磁盘文件中
void main() {
      char str[80];
      cout<<"Input string:"<<endl;
      gets(str);
      ofstream fout("putgetfile.txt");
      int i=0;
      while (str[i])
            fout.put(str[i++]);
      fout.close();
      cout<<"----------------------"<<endl;
```

使用put和get函数读写文件

```
    //而后使用get从文件中读出该串显示在屏幕上
    char ch;
    ifstream fin("putgetfile.txt");
    fin.get(ch);
    while(!fin.eof()){
    //从头读到文件结束(当前符号非文件结束符时继续)
    //注: 成员函数eof()在读到文件结束时方返回true
          cout<<ch;
          fin.get(ch);
    }
    cout<<endl;
    fin.close();
}
```

使用getline函数读文件

文件流类可以使用getline按行读文件

```
#include <fstream.h>
void main() {
  char line[81];
  ifstream infile("getline_1.cpp");
  infile.getline(line, 80);
  //读出一行(至多80个字符)放入line中
  while(!infile.eof()) {
  //尚未读到文件结束则继续循环(处理)
     cout<<line<<endl;          //显示在屏幕上
     infile.getline(line,80);//再读一行
  }
  infile.close();
}
```

文本文件和二进制文件

文本文件：数据按ASCII码存在文件中

```
#include <fstream.h>
void main() {
  double d = 12.345;
  ofstream outfile("asc.txt"); //ofstream默认按
  outfile<<d;                  文本文件方式打开
  outfile.close();
}
```

12.345以文本方式写入文件时，会自动将每个数字（包括小数点）都转换为ASCII码来存，所以一共要存6个字节，分别是'1', '2', '.', '3', '4', '5'的ASCII码

文本文件和二进制文件

二进制文件：数据按内存的格式存在文件中

```
#include <fstream.h>
void main() {
  double d = 12.345; //ofstream按二进制方式打开
  ofstream outfile("asc.txt", ios::binary);
  outfile.write((char *)&d, sizeof(d));
  outfile.close();    //以二进制方式写入文件
}
```

12.345在内存中占8个字节，以补码形式存储(0x713D0AD7A3B02840)，用write函数写入文件时，直接将内存中的8个字节拷贝到文件中

read()和write()读写二进制文件

```
istream& read(char* pch, int nCount )
```

功能：从文件中读入nCount个字符放入pch缓冲区中（若读至文件结束尚不足nCount个字符时，也将立即结束本次读取过程）

```
ostream& write(const char* pch
                          , int nCount)
```

功能：将pch缓冲区中的前nCount个字符写出到某个文件中

read()和write()读写二进制文件

先使用write往自定义二进制磁盘文件中写出如下3个“值”：字符串str的长度值Len、字符串str本身、以及一个结构体的数据，而后再使用read读出这些“值”并将它们显示在屏幕上

```
#include <fstream>
#include <string>
using namespace std;
void main() {
  char str[20]="Hello world!";
  struct stu{
      char name[20];
      int age;
      double score;
  } ss={"wu jun", 22, 91.5};
```

read()和write()读写二进制文件

```
ofstream fout("wrt_read_file.bin",
ios::binary);
//打开用于“写”的二进制磁盘文件
int Len=strlen(str);
fout.write((char*)(&Len), sizeof(int));
fout.write(str, Len);
//数据间无需分割符
fout.write((char*)(&ss), sizeof(ss));
fout.close();

char str2[80];
ifstream fin("wrt_read_file.bin",
ios::binary);
fin.read((char*)(&Len), sizeof(int));
fin.read(str2, Len);
```

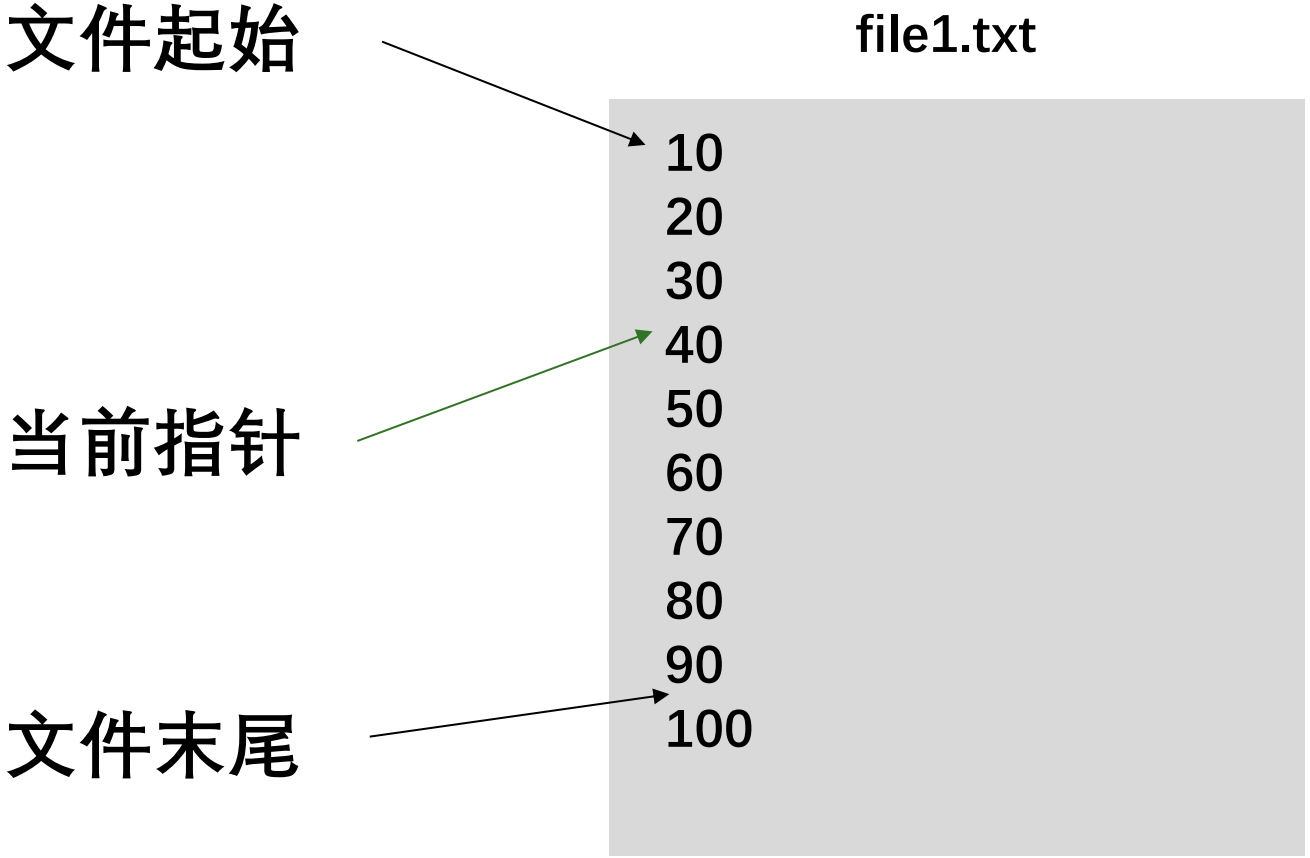
read()和write()读写二进制文件

```
    str2[Len]='\0';
    fin.read((char*)(&ss), sizeof(ss));
    cout<<"Len="<<Len<<endl;
    cout<<"str2="<<str2<<endl;
    cout<<"ss=>"<<ss.name<<","<<ss.age<<","<<ss.s
    core<<endl;
    fin.close();
}
```

设置文件的类型

- 由程序员决定将数据存储为文本文件或者二进制文件
  - 缺省打开方式时，默认为文本文件形式。若欲使用二进制文件形式，要将打开方式设为“ios::binary”
  - 通常将纯文本信息（如字符串）以文本文件形式存储，而将数值信息以二进制文件形式存储
  - 通常使用read()与write()对二进制文件进行读写

对数据文件进行随机访问



文件的随机读写,需要移动文件指针

对数据文件进行随机访问

seekg()和seekp(): 定位输入文件流对象和输出文件流对象的文件指针

```
istream& seekg(long offset, seek_dir origin=ios::beg);
ostream& seekp(long offset, seek_dir origin=ios::beg);
```

功能：将文件指针从参照位置origin移动offset个字节

- offset是长整型，取值为正表示向后移动文件指针，取值为负则表示向前移动文件指针
- origin为参照位置，seek_dir是系统定义的枚举类型，有3个枚举常量

  ios::beg：文件首
  ios::cur：文件指针当前位置
  ios::end：文件尾

对数据文件进行随机访问

假设infile是ifstream类的对象，outfile是ofstream类的对象，则

```
// 将输入文件流对象的文件指针从当前位置向前移5个字节
infile.seekg(-5, ios::cur);
// 将输出文件流对象的文件指针从文件首向后移5个字节
outfile.seekp(5, ios::beg);
```

- 在进行文件的随机读/写时，可使用tellg()和tellp()函数获取文件指针的当前位置
- 对于fstream对象，则既可使用seekg()和tellg()函数，也可使用seekp()和tellp()函数，结果完全一样

对数据文件进行随机访问

将a数组中存储的8个int型数据

- 首先通过运算符“<<”依次写出到text文件ft.txt之中
- 然后通过使用类成员函数write将这相同的8个int型数据依次写出到binary文件fb.bin之中
- 通过使用无参的成员函数“tellp()”来获取当前已写出到各文件的位置信息，以确认每一数据在文件中所占的字节数
  - ✓ostream::tellp()之功能为：获取并返回“输出指针”的当前位置值（从文件首到当前位置的字节数）

对数据文件进行随机访问

```
#include <fstream>
#include <iostream>
using namespace std;
void main() {
  int a[8]={0,1,-1,1234567890};
  for(int i=4; i<8; i++)
     a[i] = 876543210+i-4;
  //均由9位数字组成，在text文件中所占字节数也为9
  ofstream ft("ft.txt");
  ofstream fb("fb.bin", ios::binary);
```

对数据文件进行随机访问

```
    for(i=0; i<8; i++) {
        ft<<a[i]<<" "; //数据间需要添加分割符
        fb.write((char*)(&a[i]),sizeof(a[i]));
        //数据间不需分割符
        cout<<"ft.tellp()="<<ft.tellp()<<", ";
        //当前ft文件位置
        cout<<"fb.tellp()="<<fb.tellp()<<endl;
        //当前fb文件位置
      }
      ft.close();
      fb.close();
}
```

对数据文件进行随机访问

程序执行后的输出结果如下：

```
ft.tellp()=2,     fb.tellp()=4
ft.tellp()=4,     fb.tellp()=8
ft.tellp()=7,     fb.tellp()=12
ft.tellp()=18,     fb.tellp()=16
ft.tellp()=28,     fb.tellp()=20
ft.tellp()=38,     fb.tellp()=24
ft.tellp()=48,     fb.tellp()=28
ft.tellp()=58,     fb.tellp()=32
```

对数据文件进行随机访问

将3名学生的学号、姓名和成绩写入二进制文件studentinfo.dat中，再将第2名学生的成绩改为627，最后从文件中读取第m（m的值由用户从键盘输入）名学生的信息并将其输出到屏幕上

```
#include<iostream>
#include<fstream>
using namespace std;
struct Student // Student结构体类型定义
{
      char num[10];
      char name[20];
      int score;
};
```

对数据文件进行随机访问

```
int main() {
      Student stu[3] = {
            {"1210101", "张三", 618},
            {"1210102", "李四", 625},
            {"1210103", "王五", 612}
      };
      int i, m;
      fstream outfile("studentinfo.dat",
ios::binary | ios::out);
      for (i=0; i<3; i++)
          outfile.write((char*)&stu[i],
sizeof(Student)); //将学生信息写入文件
      outfile.close();
```

对数据文件进行随机访问

```
        outfile.open("studentinfo.dat",
ios::binary|ios::in|ios::out);
        stu[1].score = 627;
        // 将文件指针定位到第2名学生数据开始的位置
        outfile.seekp(1*sizeof(Student),
ios::beg);
        // 用新数据覆盖原来文件中保存的第2名学生的数据
        outfile.write((char*)&stu[1],
sizeof(Student));
        outfile.close();
        cout<<"请输入待查询的学生序号(1~3):";
        cin>>m;
```

对数据文件进行随机访问

```
    if (m<1 || m>3)        {
        cout<<"学生不存在！"<<endl;
        return 0;
    }
    //以读方式打开二进制文件
    fstream infile("studentinfo.dat",
ios::binary|ios::in);
     //将文件指针定位到第m名学生数据开始的位置
     infile.seekg((m-1)*sizeof(Student),
ios::beg);
     //将指定学生的数据读入到内存中
     infile.read((char*)&stu[m-1],
sizeof(Student));
```

对数据文件进行随机访问

```
        cout<<"第"<<m<<"名学生信息:"<<endl:
        cout<<stu[m-1].num <<","<< stu[m-1].name
<<","<< stu[m-1].score <<endl;
        infile.close();
        return 0;
}
```

字符串流

字符串流类

- 字符串流类对象并不对应于一个具体的物理设备，而是将内存中的字符数组看成是一个逻辑设备，并通过“借用”对文件进行操作的各种运算符和函数，最终完成上述所谓的信息转换工作
- 使用字符串流类时，必须包含头文件strstream

ostrstream类

- ostrstream类将不同类型的信息转换为字符串，并存放在(输出到)用户设定的字符数组中
  - 该类最常用的构造函数的一般格式为：
    ostrstream(char* str, int n, int mode = ios::out);

//其中str是用户指定的字符数组，n用来指定这个数组最多能存放的字符个数, mode给出流的方式

ostrstream类

```
#include <iostream>
#include <strstream>
using namespace std;
int main() {
        int a = 123;
        char *pbuffer = new char[10];
        ostrstream ostr(pbuffer, 10, ios::out);
        ostr<<a; //将123变为字符串，放在pbuffer中
        ostr<<4.5<<ends; //将4.5变为字符串，接在后面
        //ends是增加字符串结尾符
        cout<<pbuffer; //输出1234.5
        delete[] pbuffer;
}
```

ostrstream类

```
#include<fstream>
#include<iostream>
#include<strstream>
using namespace std;
int main() {
    ostrstream os; //未指定字符数组，系统来分配
    string str = "abcef";
    int i = 1000;
    os << str << i; //将str和i连成一个字符串，存放在系统分配的空间中
    cout << os.str()<<endl; //输出"abcef1000"
    cout<<os.pcount(); //ostrstream成员函数，统计当前已经输出了多少个字符
    return 0;
}
```

istrstream类

- istrstream类可将用户字符数组中的字符串取出（读入），而后反向转换为各种变量的内部形式
  - 一参构造函数：istrstream(char* str);
    - 由参数str 指定一个以‘\0’为结束符的字符串（字符数组）
  - 二参构造函数：istrstream(char* str, int n);
    - 由参数str 指定字符数组，由第二参数n 指出仅使用str的前n个字符
    - 二参构造函数时，并不要求str 中必须具有‘\0’结束符号
    - 若n=0，则假定str 为一个以‘\0’为结束符号的字符串（字符数组）

istrstream类

```
#include<fstream>
#include<iostream>
#include<strstream>
using namespace std;
int main() {
    char *str = "1234   100.35 ";
    istrstream inp(str);
    int nNumber;
    float balance;
    inp>>nNumber; //从字符串中拆出整数1234
    inp>>balance; //从字符串中拆出float 100.35
    cout<<nNumber<<' '<<balance;
}
```

stringstream字符串流类

- 由iostream继承
- 支持string类型的字符串流类
  - ostringstream向string写数据
  - istringstream从string读数据
  - stringstream即可从string读数据，也可以像string写数据
- 主要成员：
  - stringstream::str()：返回字符串流保存的string

字符串流类

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main(){
    stringstream ostr("ccc"); //初始字符串为"ccc"
    ostr.put('d'); //向字符串添加'd'，注意从字符串开头增加，即覆盖掉第1个'c'
    ostr.put('e'); //向字符串添加'e'，覆盖掉第2个'c'
    string gstr = ostr.str();
    cout<<gstr<<endl; //输出"dec"
    char a;
    ostr>>a;
    cout<<a;
}
```

字符串流类

stringstream类的对象常用来进行string与各种内置类型数据之间的转换

```
int main() {
    stringstream sstr;
    //--------int转string----------
    int a=100;
    string str;
    sstr<<a; //将100写入sstr
    sstr>>str; //将100转为string
    cout<<str<<endl;
    //--------string转char[]--------
    sstr.clear(); //clear()成员函数，清除缓存
    string name = "colinguan";
    char cname[200];
    sstr<<name;
    sstr>>cname;
    cout<<cname;
}
```

字符串流类

用一个字符串包含浮点型数组的所有元素的文本表示，精度为4位，每行包含5个数，并在宽度为7个字符的输出域内右对齐

```
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

int main() {
    vector<double> values;
    int num;
    cin >> num;
```

字符串流类

```
    for (int i = 0; i < num; ++i){
        double d;
        cin >> d;
        values.push_back(d);
    }
    stringstream ss;
    for (int i = 0; i < num; ++i) {
        //按照格式将double转换为字符串
        ss <<setprecision(4)<<setw(7)<<right<<
values[i];
        if ((i + 1) % 5 == 0)
            ss << endl;
    }
    cout << ss.str() << endl;
}
```

字符串流类

程序结果：蓝色表示输入，黑色表示输出

```
7
1.23456
3.1415
1.4142
-5
17.0183
-25.1283
1000.456
  1.235  3.142  1.414     -5  17.02
 -25.13   1000
```

END