
# 第7章：输入/输出流库



计算机系 江志英 jiangzy@mail.buct.edu.cn

# 本章内容

1. 流的基本概念
2. 输出流
3. 输入流
4. 输入/输出格式控制
5. 文件I/O流





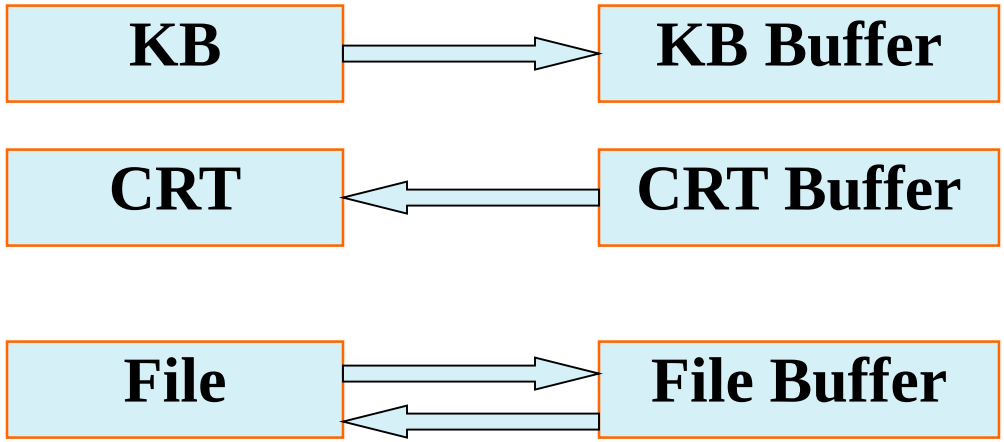
计算机系 江志英 jiangzy@mail.buct.edu.cn

# 一、流的概念

- 流（stream）：流操作，简称流
- 在计算机内存中，数据从内存的一个地址移动到另一个地址称为数据流动——流操作。
- 流操作是通过缓冲区（buffer）机制实现的。
- 缓冲区（buffer）：内存的一块区域——用作文件与内存交换数据。
- 将数据从文件中读出：


- 将数据写入文件：

- 在C++中，把输入设备（如键盘KB）、输出设备（如显示器CRT）看成一种文件——即输入输出设备均引入缓冲区机制——称设备文件。
- 流操作：

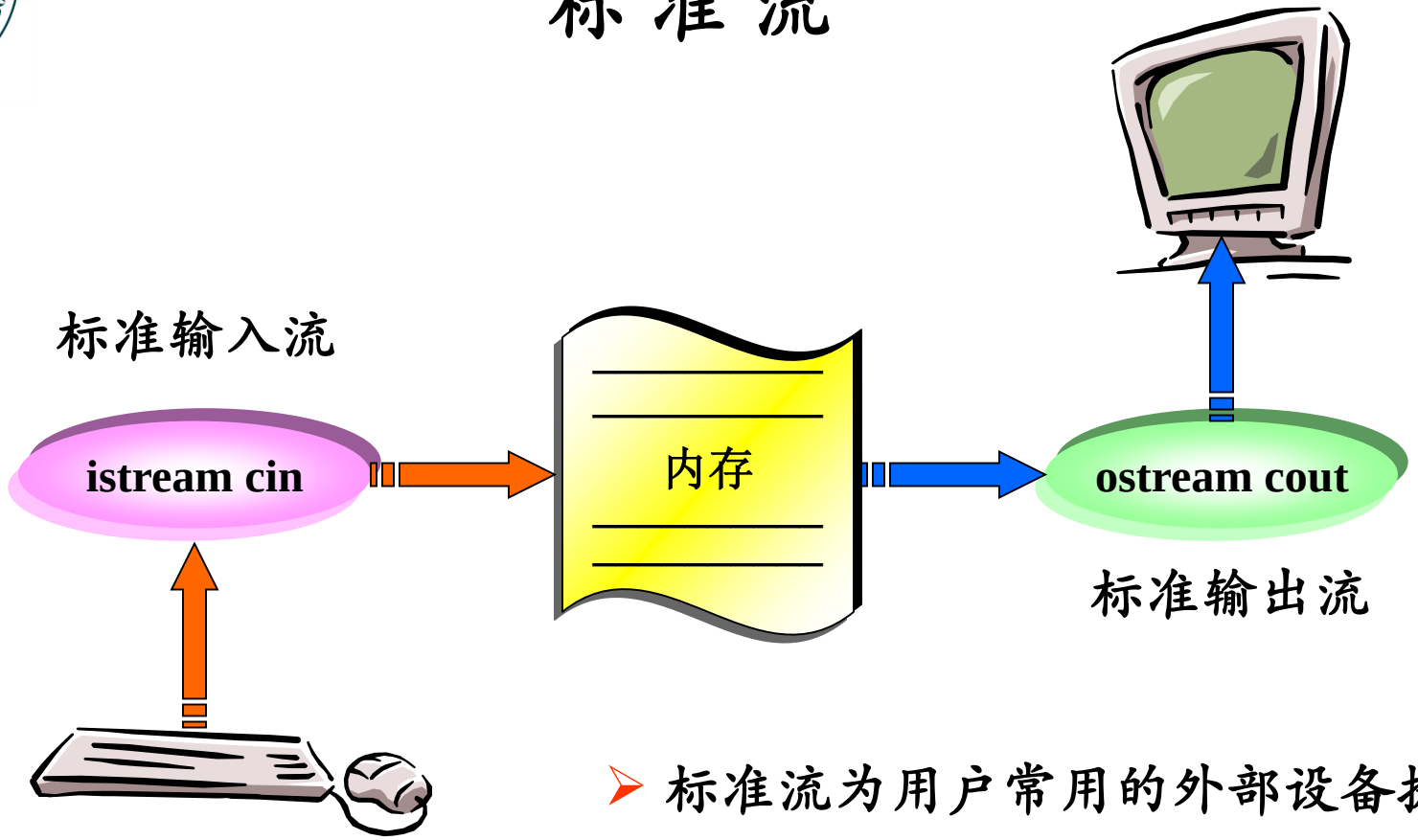



- C++中输入输出操作通过调用标准流库实现：
  - ☆KB：标准输入流——用标准输入流对象cin表示
  - ☆CRT：标准输出流——用标准输出流对象cout表示

# 头文件

- iostream.h 包含操作所有输入/输出流所需的基本信息
  含有cin、cout、cerr、clog对象，提供无格式和格式化的I/O
  - cin与标准输入设备（即键盘）相关联
  - cout 与标准输出设备（即显示器）相关联
  - cerr 与标准错误输出设备（默认为显示器）相关联（非缓冲方式）
  - clog与标准错误输出设备（默认为显示器）相关联（缓冲方式）
- iomanip.h 包含格式化I/O操纵算子，用于指定数据输入输出的格式
- fstream.h 处理文件信息，包括建立文件，读/写文件的各种操作接口
- 每一种C++版本通常还包含其他一些与I/O相关的库，提供特定系统的某些功能

# 标 准 流




- 标准流为用户常用的外部设备提供与内存之间的通信通道，对数据进行解释和传输，提供必要数据缓冲

# 二、输出流

- 在C++中，将“<<”（即左移运算符）重载为输出运算符；
- 输出运算符“<<”有二个运算分量，左边（左分量）为输出流ostream对象（cout），右边（右分量）为一个基本类型数据
- 可以重载“<<”输出结构变量或类对象。
- 下例重载“<<”运算符，输出point对象。

```
#include<iostream.h>
class point
{
 private:
     float x,y,z;
 public :
    point(float a=0,float b=0,float c=0)
      { x=a;y=b;z=c;}
    friend ostream &operator<<(ostream &os,point p)
      //重载输出运算符“<<”
    {
          os<<"("<<p.x<<","<<p.y<<","<<p.z<<")\n";
          return os;
          //返回输出流类ostream的一个对象引用os：即cout
     }
};

void main(void)
{
    point p1(1,2,3),p2(4,5,6),p3(7,8,9);
    cout<<p1<<p2<<p3<<endl;
}
```

# 输出运算符重载

1、重载“<<”运算符格式:

```
ostream &operator<<(ostream &os,point p)
{
   相对于该类对象的输出操作
    return os;
}
```

2、说明:

- 运算符“<<”必须重载为友元;
- 重载运算符“<<”中，第一个形参必须是输出流ostream的一个对象引用；第二个形参为输出对象或对象引用
- 重载运算符返回值应是一个输出流对象引用——能连续输出多个对象。


计算机系 江志英 jiangzy@mail.buct.edu.cn

# 三、输入流

- 在C++中，将“>>”（即右移运算符）重载为输入运算符；
- 输入运算符“>>”有二个运算分量，左边（左分量）为输入流istream对象（cin），右边（右分量）为一个基本类型数据
- 可以重载“>>”输入结构变量或类对象。
- 下例重载“>>”运算符，输入point对象

```
#include<iostream.h>
class point {
  private:
     float x,y,z;
  public :
    point(float a=0,float b=0,float c=0) { x=a;y=b;z=c;}
    friend ostream &operator<<(ostream &os,point p){
          os<<"("<<p.x<<","<<p.y<<","<<p.z<<")\n";
          return os;
    }
    friend istream &operator>>(istream &is,point &p) {
          //重载输入运算符“>>”
          cout<<"x=";is>>p.x;cout<<"y=";is>>p.y;
          cout<<"z=";is>>p.z;
          return is;
    }
};
```

```
void main(void){
          point p1(4,5,6),p2;
          cin>>p2;
          cout<<p1<<p2<<endl;
}
```

# 输入运算符重载

1、重载“>>”运算符格式：

```
istream &operator<<(istream &is,point &p)
{
   相对于该类对象的输出操作
    return is;
}
```

2、说明：

- 运算符“>>”必须重载为友元；
- 重载运算符“>>”中，第一个形参必须是输出流istream的一个对象引用；第二个形参为输入对象引用
- 重载运算符返回值应是一个输入流对象引用——能连续输入多个对象。

# 四、输入输出格式控制

## 用于输入输出格式控制的成员函数

| 函数原型 | 功　能 |
|---|---|
| long ios_base::setf(long flags); | 设置状态标志flags |
| long ios _base ::unsetf(long flags); | 清除状态标志并返回清除前的标志 |
| long ios _base ::flags( ); | 测试状态标志 |
| long ios _base :: flags(long flags); | 设置标志flags并返回设置前标起 |
| int ios _base ::width(); | 返回当前宽度设置值 |
| int ios _base ::width(int w); | 设置域宽w并返回以前的设置 |
| int ios _base ::precision(int p); | 设置小数位数，并返回以前的小数位数 |
| char ios::fill( ) | 返回当前的填充字符 |
| char ios::fill( char ch) | 设置填充字符ch，返回当前的填充字符 |

# 使用操作符进行输入输出格式控制

使用ios类中的成员函数进行输入输出格式控制时，每个函数的调用都需要写一条语句，不能将其嵌入到输入输出语句中。

- C++提供了另一种I/O格式控制的方法——操作符。

㈠ C++预定义的操作符-iostream中的控制符

- dec 以十进制输入输出整型数，用于输入输出
- hex 以十六进制输入输出整型数，用于输入输出
- oct 以八进制输入输出整型数，用于输入输出
- ws 输入时跳过开头的空白符，用于输入
- endl 插入一个换行符并刷新输出流，用于输出
- ends 插入一个空字符，用以结束一个字符串，用于输出


计算机系 江志英 jiangzy@mail.buct.edu.cn


# （二） C++预定义的操作符-iomanip中的控制符

- flush 刷新一个输入流，用于输入
- setbase(int n) 把转换基数设置为n(n的取值为0,8,10,16)，n的缺省值为0（以十进进制形式输出）。
- resetiosflags(long f) 关闭由参数f指定的格式标志，用于输入输出
- setiosflags(long f) 设置由参数f指定的格式标志，用于输入输出
- setfill(char c) 设置填充字符
- setprecision(int n) 设置数据小数位数，缺省时为6，用于输入输出
- setw(int n) 设置域宽，用于输入输出
- 其中:setiosflags(long f)、resetiosflags(long f) 中的long f为格式标志——同上。

## 控制输出宽度

- 为了调整输出，可以通过在流中放入setw操纵符或调用width成员函数为每个项指定输出宽度。

使用width控制输出宽度

```
#include <iostream>
void main()
{ double values[] = {1.23,35.36,653.7,4358.24};
  for(int i=0;i<4;i++)
  { cout.width(10);
    cout << values[i] <<'\n';
  }
}
```

```
      1.23
     35.36
     653.7
   4358.24
Press any key to continue
```


计算机系 江志英 jiangzy@mail.buct.edu.cn


## 使用*填充

```
#include <iostream>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  for(int i=0; i<4; i++)
  {  cout.width(10);
     cout.fill('*');
     cout<<values[i]<<'\n';
  }
}
```

```
*******1.23
******35.36
******653.7
****4358.24
Press any key to continue
```

## 使用setw指定宽度

```
#include <iostream.h>
#include <iomanip.h>
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  for(int i=0;i<4;i++)
   cout<<setw(6)<<names[i]
       <<setw(10)<<values[i]
       <<endl;
}
```

```
  Zoot      1.23
 Jimmy     35.36
    Al     653.7
  Stan   4358.24
Press any key to continue
```


# 设置对齐方式

```
#include <iostream>
#include <iomanip>
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  for(int i=0;i<4;i++)
   cout<<setiosflags(ios::left)
       <<setw(6)<<names[i]
       <<resetiosflags(ios::left)
       <<setw(10)<<values[i]
       <<endl;
}
```

```
Zoot          1.23
Jimmy        35.36
Al           653.7
Stan       4358.24
Press any key to continue
```

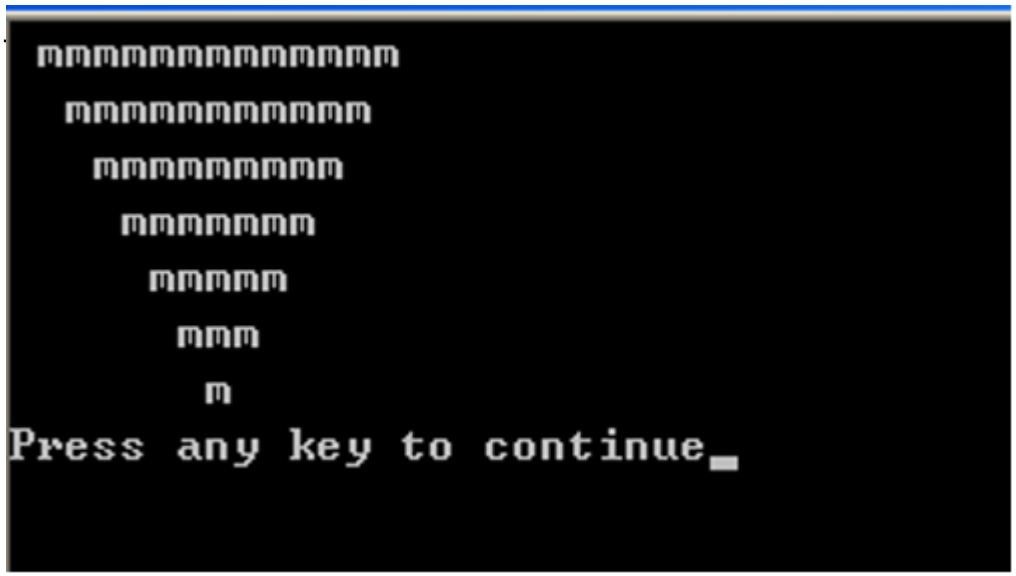
# 控制输出精度

```
Zoot     1.2e+000
Jimmy    3.5e+001
Al       6.5e+002
Stan     4.4e+003
Press any key to continue
```

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{ double values[]={1.23,35.36,653.7,4358.24};
  char *names[]={"Zoot","Jimmy","Al","Stan"};
  cout<<setiosflags(ios::scientific);
  for(int i=0;i<4;i++)
   cout<<setiosflags(ios::left)
       <<setw(6)<<names[i]
       <<resetiosflags(ios::left)
       <<setw(10)<<setprecision(1)
       << values[i]<<endl;
}
```


计算机系 江志英 jiangzy@mail.buct.edu.cn

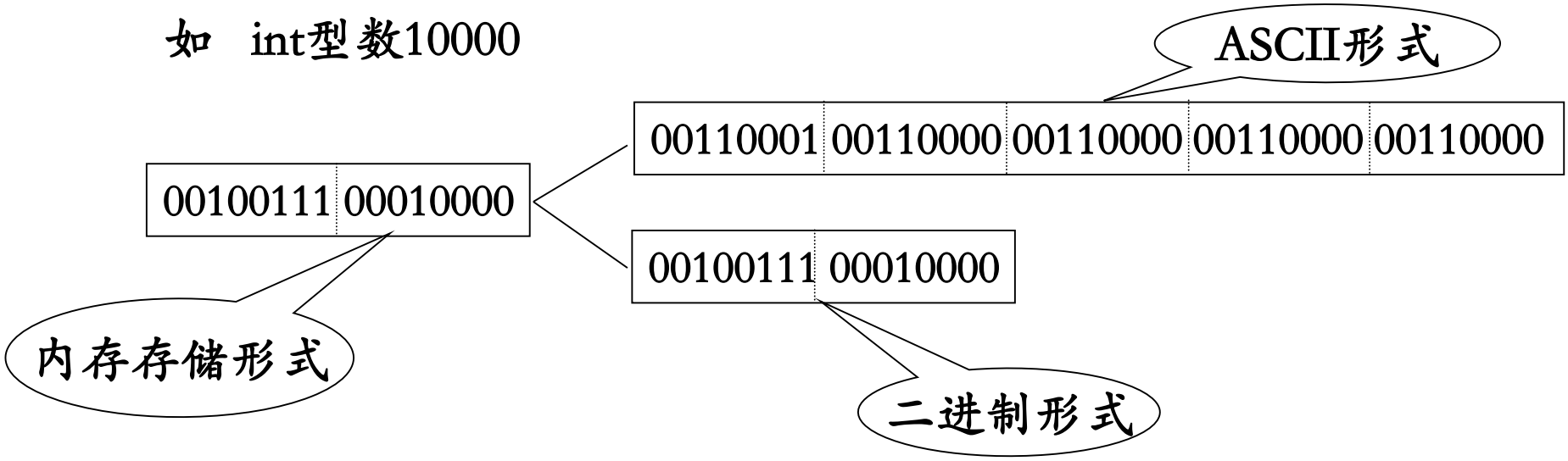
```
#include <iostream>
using namespace std;
#include <iomanip>
```




```
endl;
```

# 五、文件I/O流

- 文件的概念
  - C++中的文件是一个字符流或二进制流——流式文件；
  - 文件的分类：按文件的组织形式分为字符文件(ASCII文件)、二进制文件；
  - 文件的操作：打开文件、文件的读/写、文件的关闭



# 文件处理的三个流类

- ifstream—文件输入流类，用于文件的输入（读文件）
- ofstream—文件输出流类，用于文件的输出（写文件）
- fstream—文件输入/输出流类，用于文件的输入/输出（读/写文件）
- 当程序中进行文件操作时，应加上头文件“fstream.h”
- 若要打开文件进行读写操作，必须定义相应的流对象。如：

```
ifstream in;    //文件输入流对象
ofstream out;   //文件输出流对象
fstream both;   //文件输入/输出流对象
```

# 文件的打开与关闭

## 1、文件打开用成员函数open()

- 原型：

  void open( const char *s,ios_base::openmode ，mode=ios_base::out|ios_base::trunc)

- 其中，第一个参数表示打开的文件，第二个参数表示文件打开方式，第三个参数表示访问方式。

| 标志 | 涵义 |
|---|---|
| ios::ate | 以添加方式打开 |
| ios::in | 作为输入文件(ifstream默认) |
| ios::out | 作为输出文件(ofstream默认) |
| ios::trunc | 若文件存在，清除文件内容(默认) |
| ios::nocreate | 若文件不存在，返回错误 |
| ios::noreplace | 若文件存在，返回错误 |
| ios::binary | 以二进制方式打开 |


计算机系 江志英 jiangzy@mail.buct.edu.cn

## 2、文件打开的方法：

- 方法一：先定义一个文件流对象，再用文件流对象调用成员函数open()打开一个文件。如：

```
ifstream  f1;  //定义文件输入流对象f1
f1.open( “d:\\vcprg\\7-3.cpp” ); //打开D盘vcprg文件夹（目录）下的7-3.cpp文件，可进行文件读操作
```

- 方法二：在定义文件流对象时打开文件。如：

```
ifstream f1( “d:\\vcprg\\7-3.cpp” )
```

- 用文件流对象打开文件后，该对象就代表了被打开的文件。

//打开D盘vcprg文件夹（目录）下的test.cpp文件，
并显示文件中的内容。

```
#include<iostream>
#include<fstream>
using namespace std;
void main(void)
{
    ifstream f1;        //定义文件输入流对象
    f1.open("d:\\vcprg\\test.cpp"); //打开文件
    char c;
    while(f1)          //判断文件未结束则循环
    {
        f1.get(c);
        cout<<c;
    }
    cout<<endl;
    f1.close();//关闭文件
}
```


## 3、文件访问方式

- 当用ifstream定义流对象并打开一个文件时，默认为ios_base::in方式；
- 当用ofstream定义流对象并打开一个文件时，默认为ios_base::out方式；
- 当用fstream定义流对象并打开一个文件时，应给出打开方式（可用“位或”运算以多种方式打开文件）。

如：

```
fstream f;
f.open(“file.cpp”,ios_base::in|ios_base::out)
```

例：将D盘vcprg文件夹（目录）下的test.cpp文件，复制为text.txt文件。

```
#include<iostream>
#include<fstream>
using namespace std;
void main(void)
{
   ifstream f1;        //定义文件输入流对象
   ofstream f2;
   f1.open("d:\\vcprg\\test.cpp"); //以读方式打开文件
   f2.open("d:\\vcprg\\text.txt"); //以写方式打开文件
   char c;
   while(f1) { //判断文件未结束则循环
      f1.get(c);
      f2<<c;
   }
   cout<<"文件复制成功"<<endl;
   f1.close();//关闭文件
   f2.close();
}
```


## 4、文件异常处理

- 当打开一个文件可能失败，在程序中通常应加上异常处理程序。

(1) 用条件语句判断文件流标志位failbit是否为true。

```
if(!文件流对象){<异常处理程序>}
if(文件流对象){<正常处理程序>}
```

(2) 用成员fail()函数

```
if(文件流对象.fail()){<异常处理程序>}
if(!文件流对象.fail()){<正常处理程序>}
```


## 5、文件的关闭

格式：文件流对象.close();

关闭文件操作包括把缓冲区数据完整地写入文件，添加文件结束标志，切断流对象和外部文件的连接

若流对象的生存期没有结束，可以重用

当一个流对象的生存期结束，系统也会自动关闭文件

*例如：*

```
ofstream  ofile ;   // 创建输出文件流
ofile . open ( "myfile1" ) ; // ofile流与文件“myfile1”相关联
……      // 访问文件“myfile1”
ofile . close ( ) ;   // 关闭文件“myfile1”
ofile . open ( "myfile2" );  // 重用ofile流
```


```
#include <fstream.h>
void main ( )
{
  ofstream  ost ;     // 创建输出流对象
  ost . open ( “d:\\my1.dat ") ;  // 建立文件关联，缺省为文本模式
  ost << 20 << endl ;  // 向流插入数据
  ost << 30.5 << endl ;
  ost . close ( ) ;   // 关闭文件
  ifstream  ist ( “d:\\my1.dat" ) ;  // 创建输入流对象并建立关联
  int  n ;
  double  d ;
  ist >> n >> d ;   // 从流提取数据
  cout << n << endl << d << endl ;  // 向预定义流插入数据
}
```

# 文件的读写

1、文本文件的读写

读文件格式：文件输入流对象.get(char);

写文件格式：文件输出流对象.put(char ch);

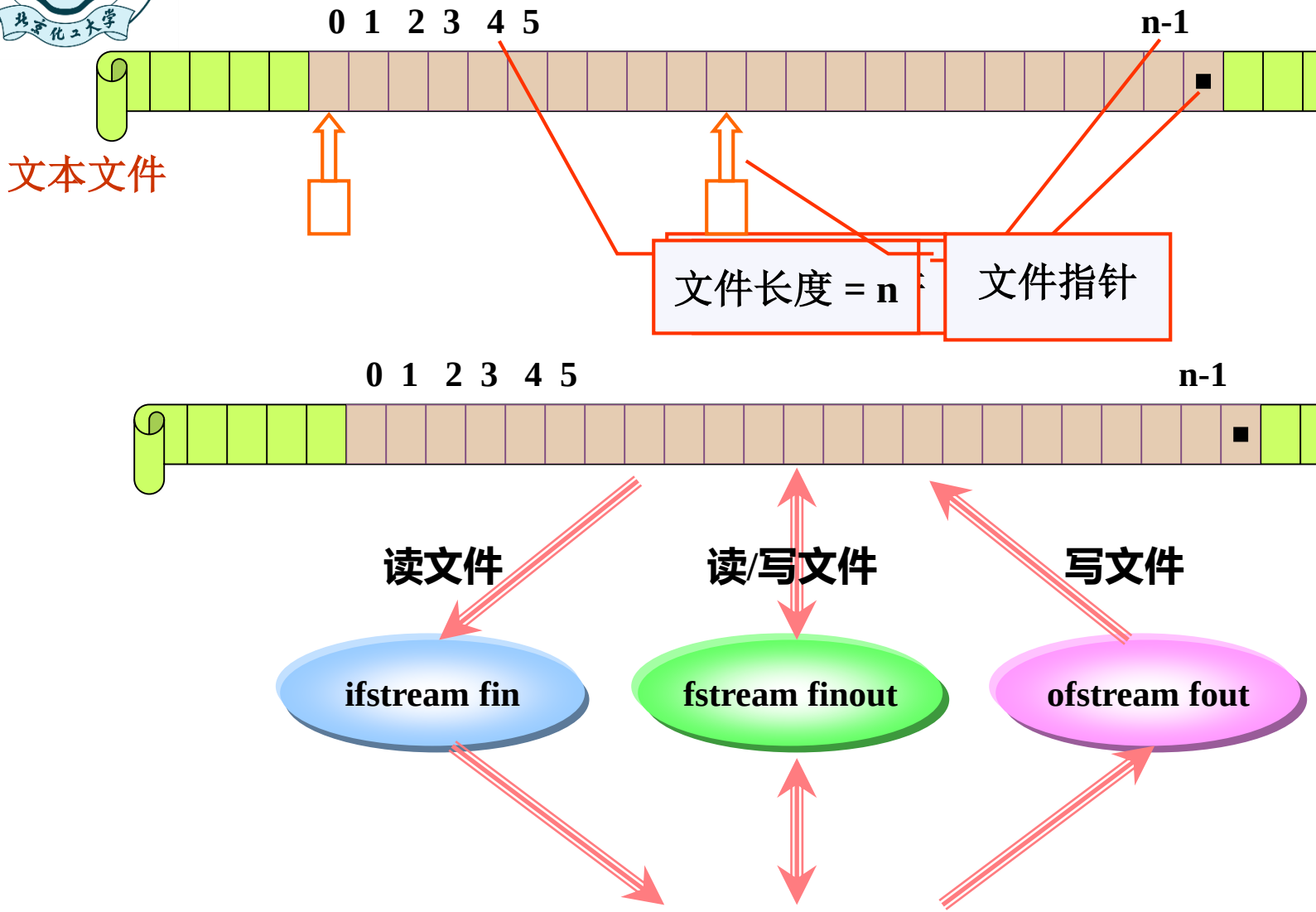
2、二进制文件的读写

- 二进制文件的读/写分别用成员函数read()、write()实现。
- 写二制文件的格式：

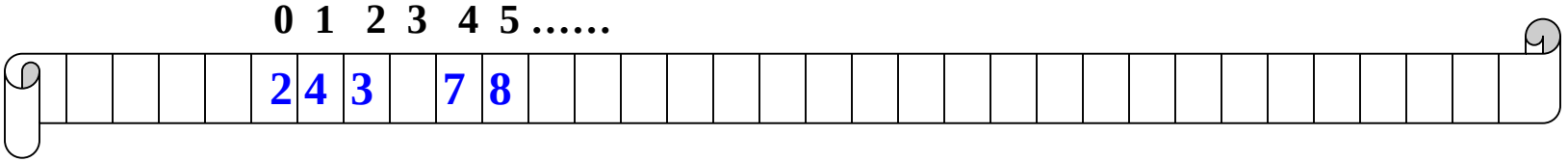
  输出文件流对象.write((char*)&对象或&对象数组名[下标], sizeof(对象名或所属类名));

- 读二进制文件的格式

  输入文件流对象.read((char*)&对象或&对象数组名[下标], sizeof(对象名或所属类名));


0 1 2 3 4 5
n-1
文本文件
文件长度 = n
文件指针
0 1 2 3 4 5
n-1
读文件
读/写文件
写文件
ifstream fin
fstream finout
ofstream fout

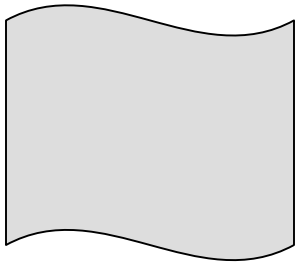
## 文本文件



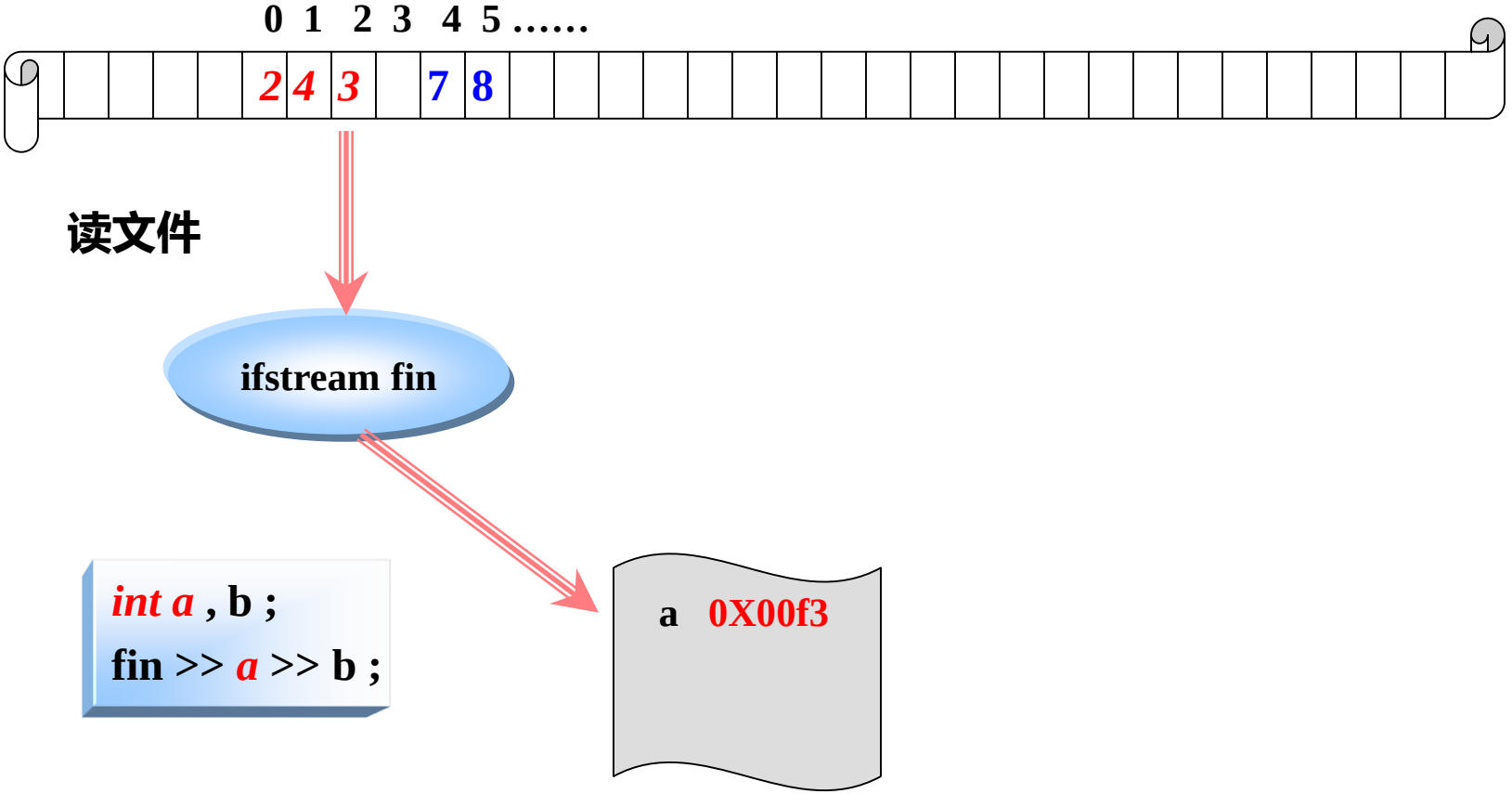

**读文件**

**ifstream fin**

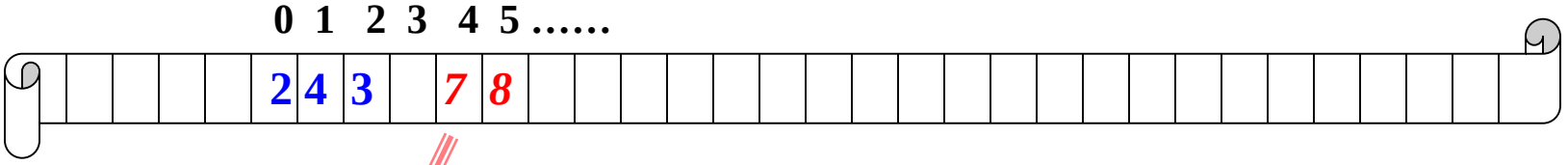
```
int a , b ;
fin >> a >> b ;
```



文本文件



## 文本文件



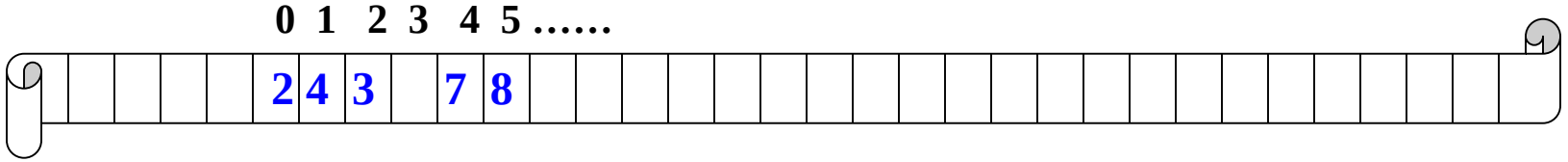

**读文件**

**ifstream fin**

```
int a , b ;
fin >> a >> b ;
```

a 0X00f3
b 0X004e

## 文本文件



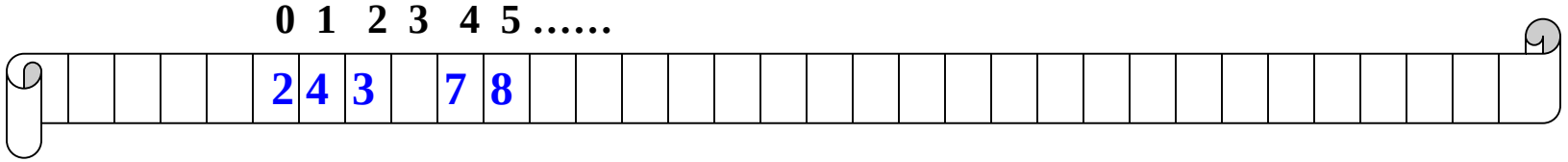

ofstream fout

```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
```

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

## 文本文件



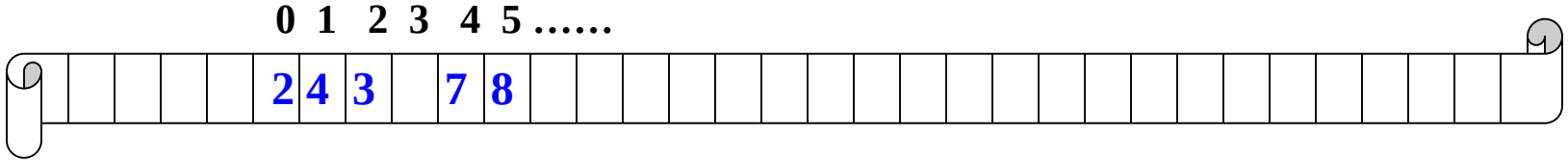

ofstream fout

```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

文本文件




**写文件**

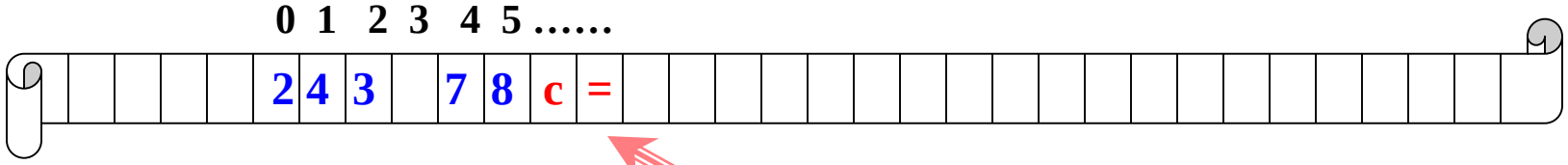
**ofstream fout**

```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

## 文本文件




**写文件**

**ofstream fout**

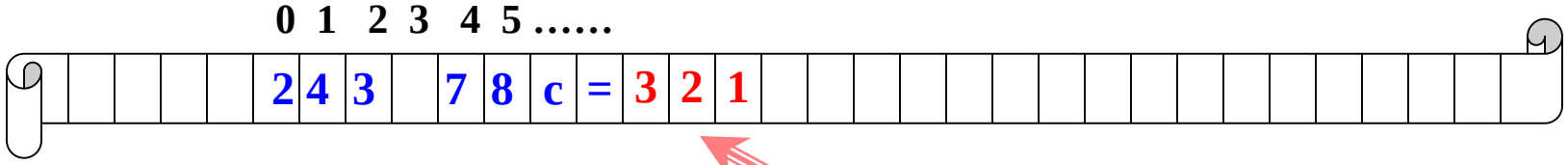
```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

*"c= "*

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

## 文本文件




写文件

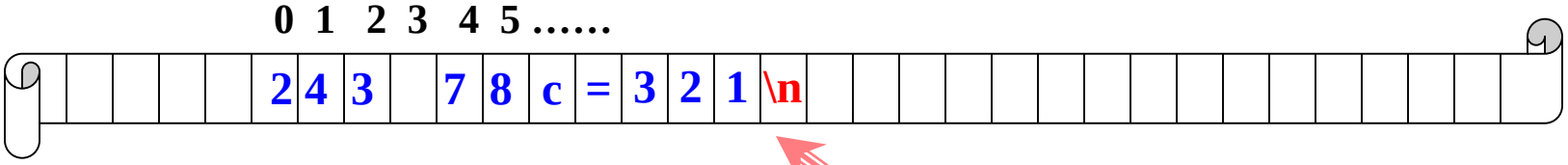
ofstream fout

```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

文本文件




写文件

ofstream fout

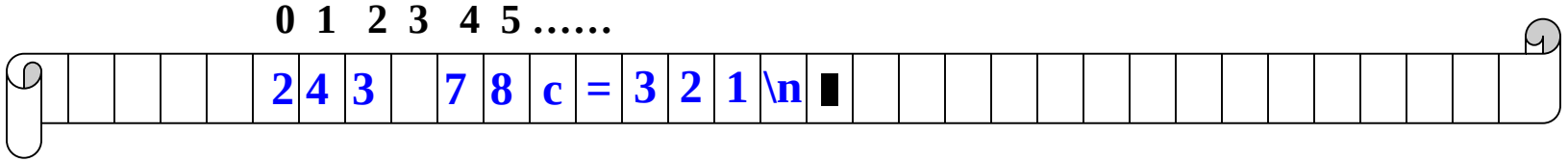
```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

*endl*

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

## 文本文件




```
int a , b ;
fin >> a >> b ;
```

```
a  0X00f3
b  0X004e
c  0X0141
```

```
int c ;
c = a + b ;
fout << "c= " <<c<<endl ;
```

# 使用格式控制建立的文本文件

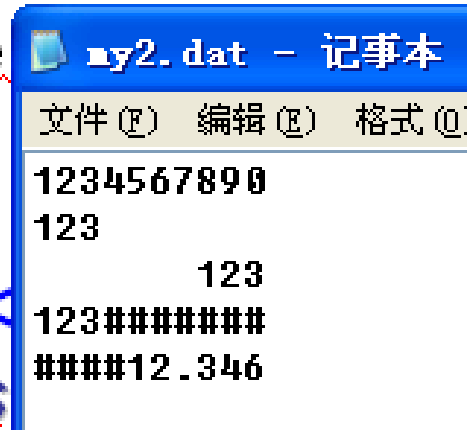
```
#include <fstream.h>
#include <iomanip.h>
void main ( )
{ ofstream  ost ;
   ost.open ( "d:\\my2.
   ost << "1234567890"
   int  a = 123 ;
   ost << a << endl ;
   ost << setw ( 10 ) << a << endl ;
   ost << resetiosflags ( ios :: right ) << setiosflags
       << setfill ( '#' ) << setw ( 10 ) << a << endl
   ost << resetiosflags ( ios :: left ) << setiosflags (
       << setprecision ( 5 ) << setw ( 10 ) << 12.34
    ;
   ost . close ( ) ;
}
```

默认方式

插入字符串

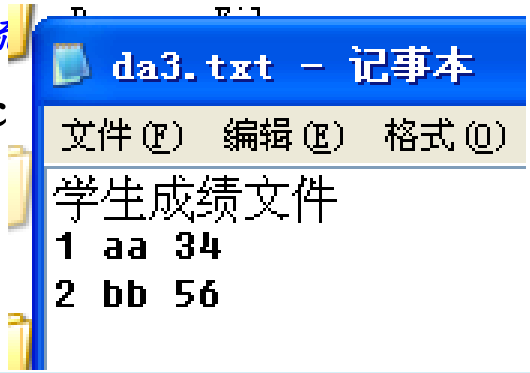
把整型

以指定格式
插入数据



*建立一个包含学生学号、姓名、成绩的文本文件*

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
void main()
{ char fileName[30] , name[30] ;     int number , score ;
  ofstream outstuf ;  // 建立输出文件流对象
  cout << "Please input the name of students file :\n" ;
  cin >> fileName ;  // 输入文件名
  outstuf.open( fileName, ios::out ) ;  // 连接文件，指定打开方式
  if ( !outstuf ) // 调用重载运算符函数测试流
    { cerr << "File could not be open." << endl ;    abort(); }
  outstuf << "学生成绩文件\n" ; // 写入一行标题
  cout << "Input the number, name, and score : (Enter Ctrl-Z to end
    input)\n? " ;
  while( cin >> number >> name >> score ) // 向流
   { outstuf << number << ' ' << name << ' ' << sc
     cout << "? " ;
   }
  outstuf.close() ; // 关闭文件
}
```



# 二进制文件

二进制文件以基本类型数据在内存的二进制表示形式存放数据，不对写入或读出的数据做格式转换

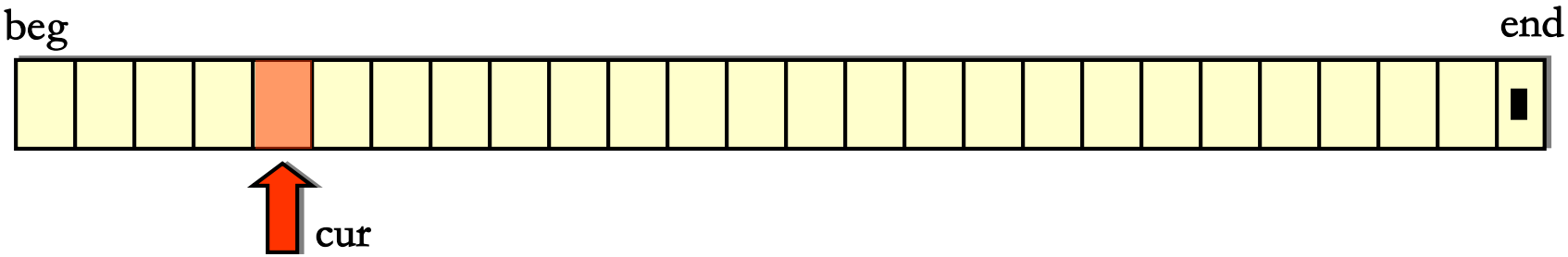
二进制文件的读写方式由程序控制

打开二进制文件用binary方式

二进制文件是随机存取文件

*随机访问流*

流的状态表示：流的内容、长度和操作位置




# *istream 类操作流读指针的成员函数*

- istream & istream :: seekg ( long *pos*) ;

*作用*　读指针从流的起始位置向后移动由*pos*指定字节

- istream & istream :: seekg ( long *off*, ios::*seek_dir*) ;

*作用*　读指针从流的*seek_dir*位置移动 *off* 指定字节

ios::seek_dir 值：

| | |
|---|---|
| cur | 相对于当前读指针所指定的当前位置 |
| beg | 相对于流的开始位置 |
| end | 相对于流的结尾处 |

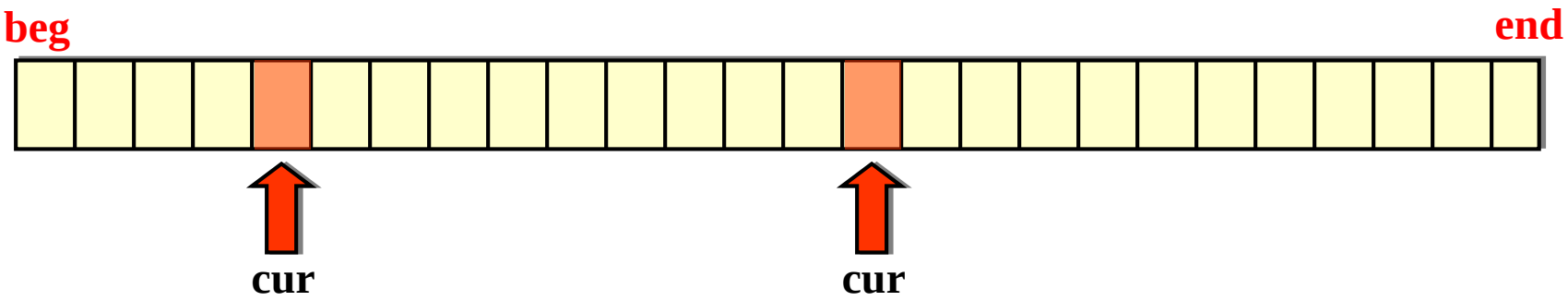
enum ios::seek_dir { beg = 0 ; cur = 1 , end = 2 } ;

- istream & istream :: teelg () ;

*作用*　返回读指针当前所指位置值

*例1：*

```
istream  input ;
    ……
input . seekg ( - 10 , ios :: cur ) ;
    // 读指针以当前位置为基准，向前移动10 个字节
```



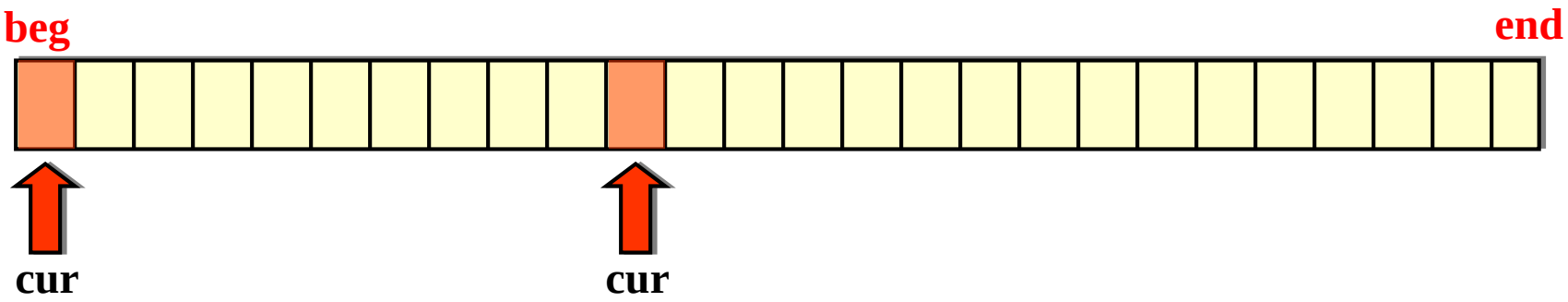
*例1：*

```
istream  input ;

……

input . seekg ( 10 , ios :: beg ) ;
```

函数　　seekg ( n ) ;

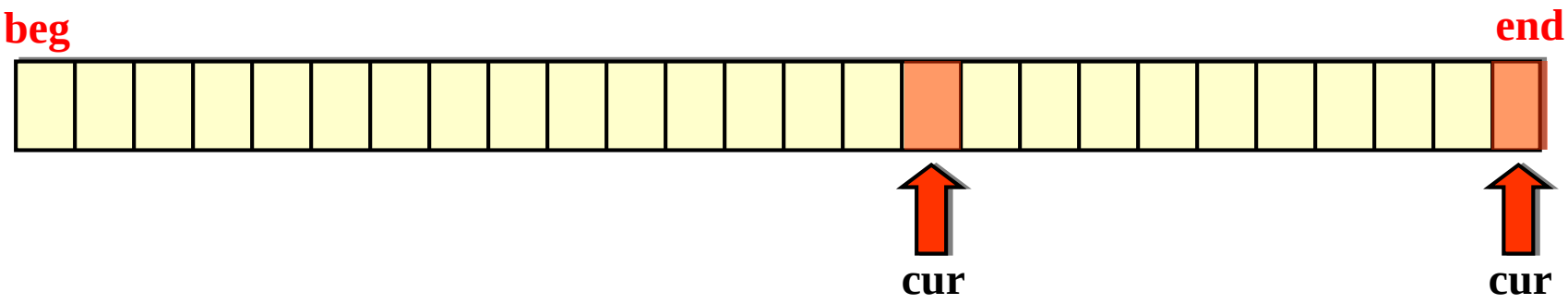
等价于　seekg ( n , ios::beg ) ;

*// 读指针从流的开始位置，向后移动10 个字节*



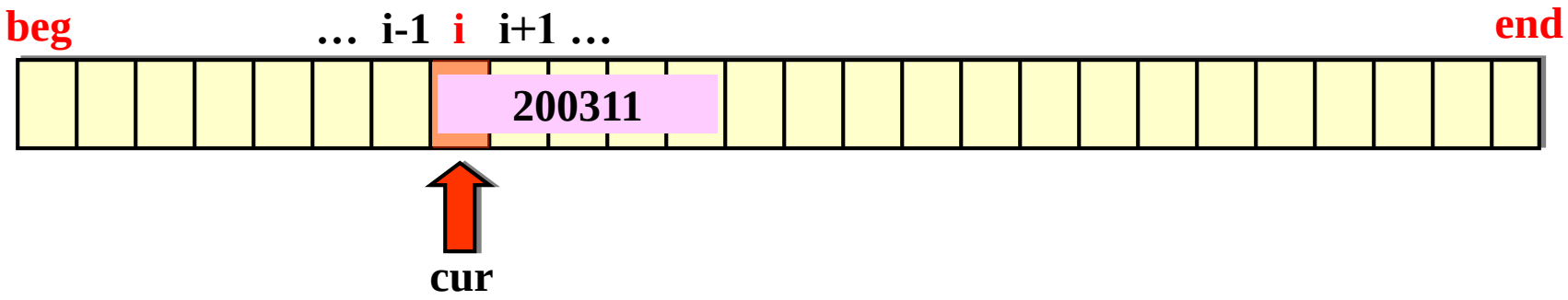
*例1：*

```
istream  input ;
    ……
input . seekg ( -10 , ios :: end ) ;
        // 读指针从流的结尾，向前移动 10 个字节
```

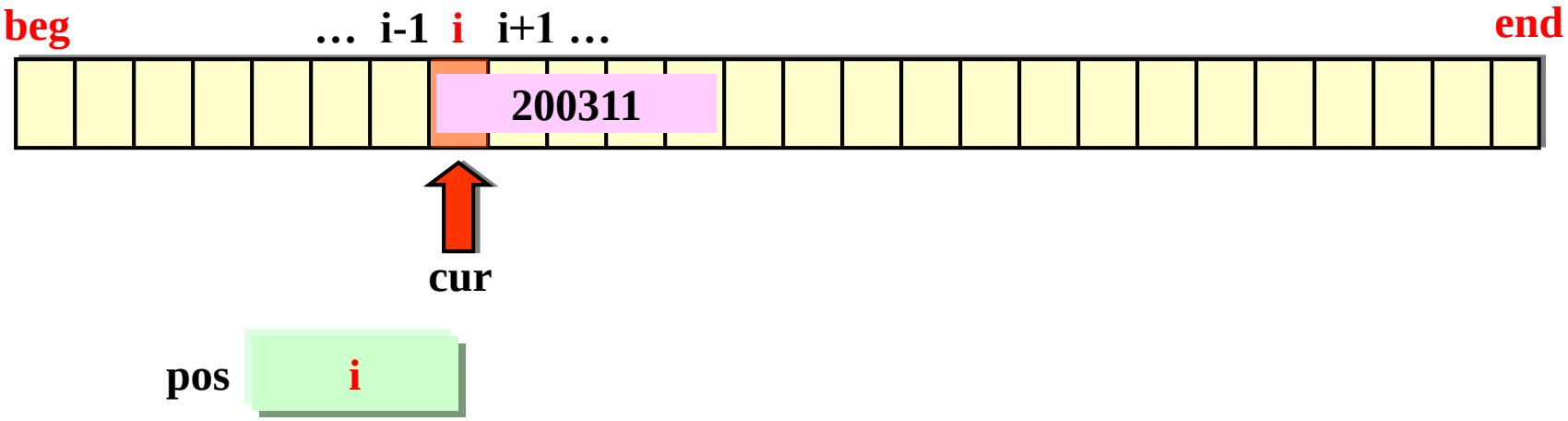


*例2：*

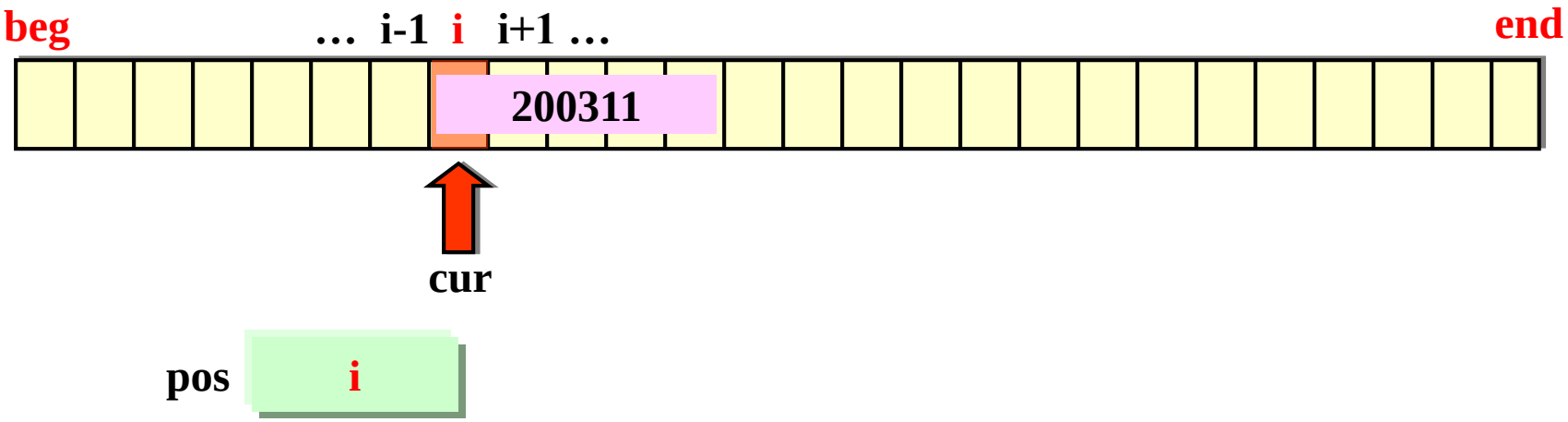
```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



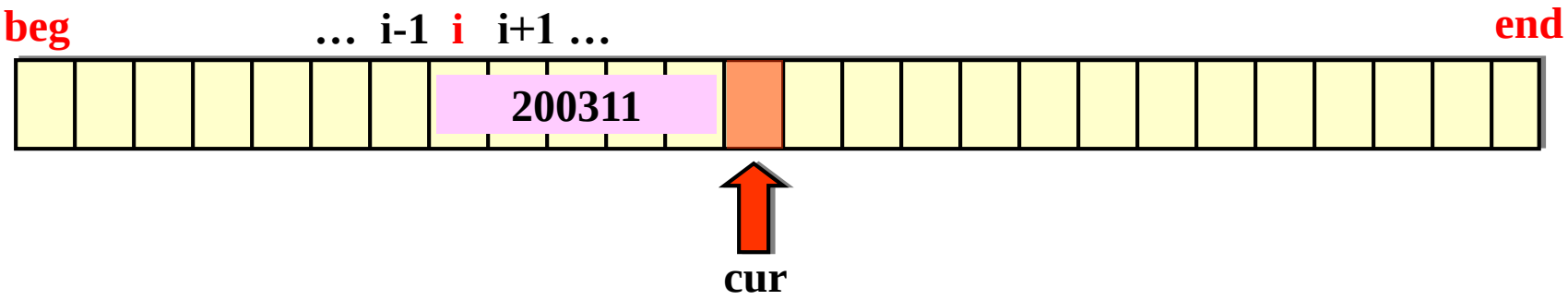
```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



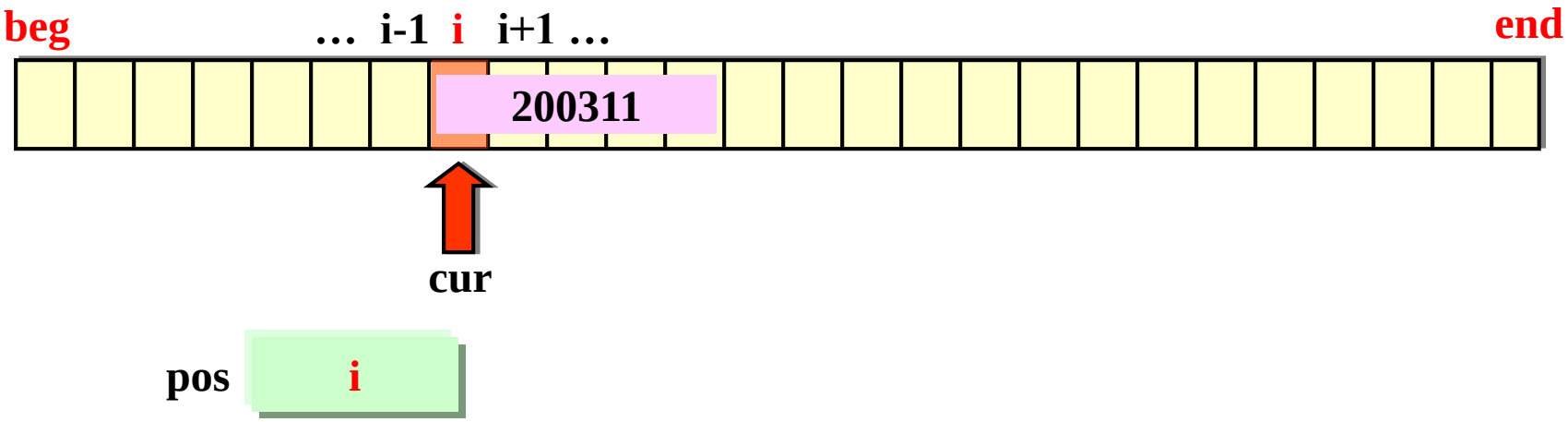
```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



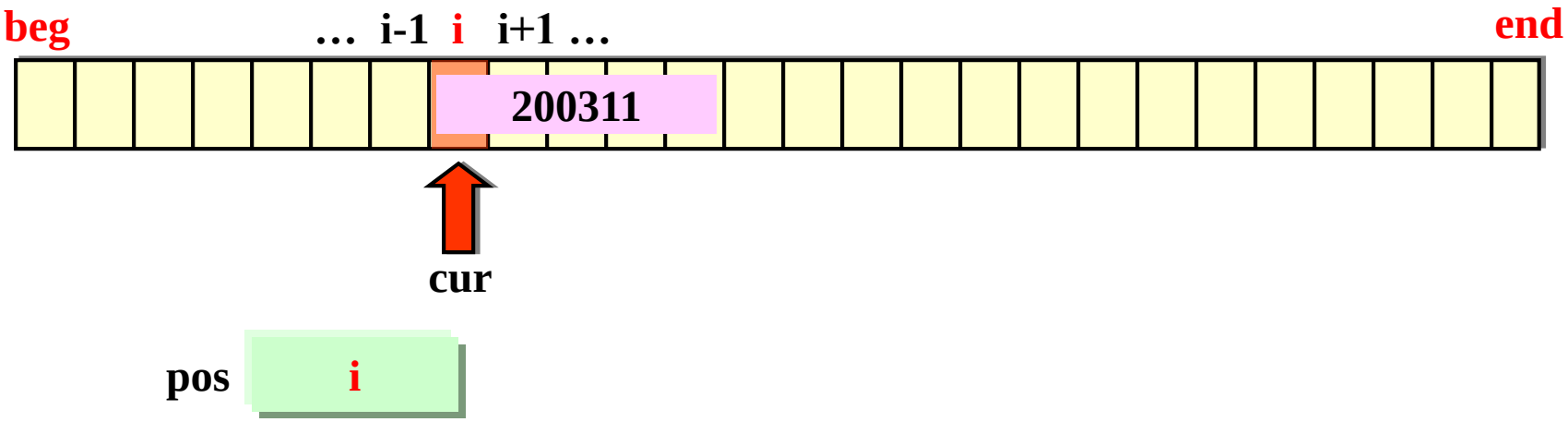
```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



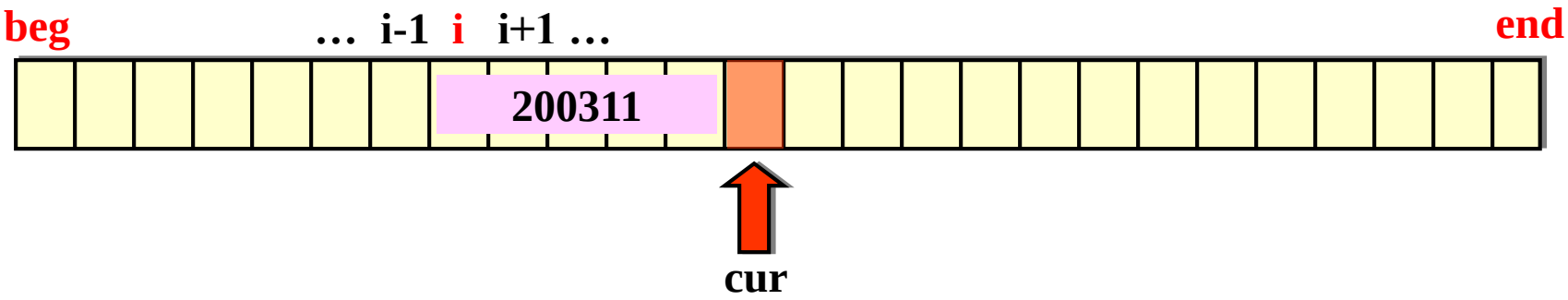
```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



```
istream  input ;
long  pos = input . tellg ( ) ;        // 获取当前位置指针
input >> number ;                      // 读入一个整数，指针后移 4 字节
      ……
input . seekg ( pos ) ;                // 指针返回原来位置
input >> number ;                      // 重读该整数
```



## *ostream 类操作流写指针的成员函数*

- **ostream & ostream :: seekp ( long *pos* ) ;**

  *作用*　写指针从流的起始位置向后移动由参数指定字节

- **ostream & ostream :: seekp ( long *off* , ios::*seek_dir* ) ;**

  *作用*　写指针从流的*seek_dir*位置移动由 *off* 指定字节

- **ostream & ostream :: teelp ( ) ;**

  *作用*　返回写指针当前所指位置值


计算机系 江志英 jiangzy@mail.buct.edu.cn


## 适于二进制流操作的成员函数

这些函数操作对象是单字节数据

它们也可以用于文本流，但必须保证流中存储数据是ASCII码，并且不会跳过空白字符


# 适于二进制流操作的成员函数

- istream 类中三个操作字节数据的成员函数

istream & istream :: get ( char & *c* ) ;

*作用*　从流中提取一个字节数据，更新对象 *c*

int istream :: get ( ) ;

*作用*　函数值返回流中一个字节数据

istream & istream :: read ( char * *buf*, int *n* ) ;

*作用*　从流中提取 n 个字节数据，更新对象 *buf*


# 适于二进制流操作的成员函数

- ostream 类中两个操作字节数据的成员函数

ostream & ostream :: put ( char *c* ) ;

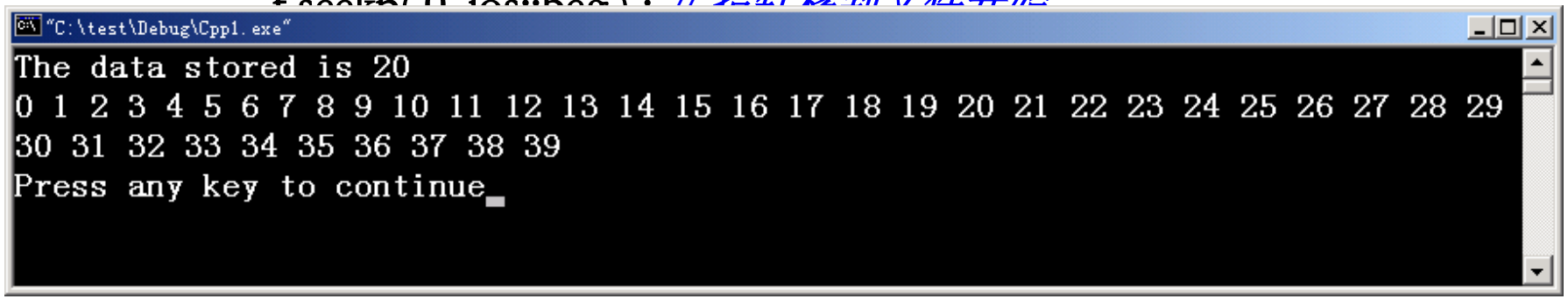
*作用*：向流插入一个字节数据

ostream & ostream :: write ( char * *buf*, int *n* ) ;

*作用*：向流插入 *buf* 对象的由第二个参数指定数目的字节数据

```
#include <fstream.h>
void main (){
    fstream f( "a:DATA.dat" , ios::in | ios::out | ios::binary ) ;
    int i ;
    for( i = 0; i < 20 ; i ++ )  f.write((char *)&i, sizeof(int) ) ;
    long pos = f.tellp( ) ;
    for( i = 20; i<40 ; i ++ ) f.write( (char*)&i, sizeof(int) ) ;
    f.seekg(pos) ;
    f.read((char*)&i, sizeof(int) ) ;
    cout << "The data stored is " << i << endl ;
}
```

// 打开随机文件
// 先写入 20 个整数
// 记录当前写指针位置值
// 再写入 20 个整数
// 将指针移到 pos 所表示的位置
// 读出一个数据
// 指针移到文件开始



# 小　结

C++流库是用继承方法建立起来的一个输入输出类库。流是对字节序列从一个对象移动到另一个对象的抽象。

流对象是内存与文件（或字符串）之间数据传输的信道。

- 标准流与系统预定义的外部设备连接；
- 串流与串对象或字符数组连接；
- 文件流与用户定义的外部文件连接。

一旦流对象和通信对象关联，程序就可以使用流的操作方式传输数据。

标准流与外设的连接是C++预定义的。其他流类对象与通信对象的连接使用流类构造函数实现。

数据流本身没有逻辑格式。数据的解释方式由应用程序的操作决定。流类库提供了格式化和非格式化的I/O功能

文本流I/O提供内存基本类型数据与文本之间的格式转换。包括重载插入和提取运算符，字符输入输出函数，以及各种格式控制。标准流、串流都是文本流。

处理用户定义的文件I/O要用文件流对象。

- 根据代码方式分为文本文件和二进制文件，
- 根据数据存取方式分为顺序存取文件和随机存取文件
- 文本文件是顺序存取文件，二进制文件是随机存取文件。

文件操作的三个主要步骤是：打开文件；读/写文件；关闭文件流。

文件的性质由打开文件的方式决定。

- 文本文件流提供数据格式化I/O;
- 二进制文件流提供字节流无格式I/O。

移动流指针，可以对文件的任意位置进行读/写操作。