第十一章 输入输出流

11.1 流类和流对象

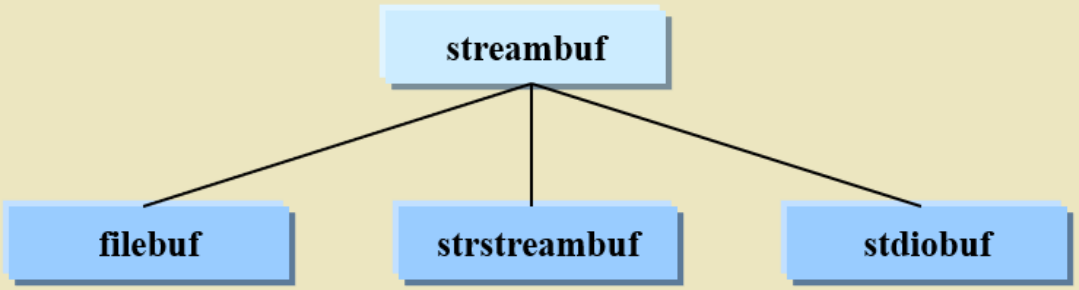
- **程序中，对数据的输入/输出是以字节流实现的**
- **应用程序对字节序列作出各种数据解释**
- **I/O系统的任务就是在内存和外部设备之间稳定可靠地传输数据和解释数据。**
- **流类对象可以建立和删除，可以从流中获取数据和向流添加数据**

流类库

- 流库（stream library）是用继承方法建立的输入输出类库
- 流库具有两个平行的基类：streambuf 和 ios 类，所有流类均以两者之一作为基类
- streambuf 类提供对缓冲区的低级操作

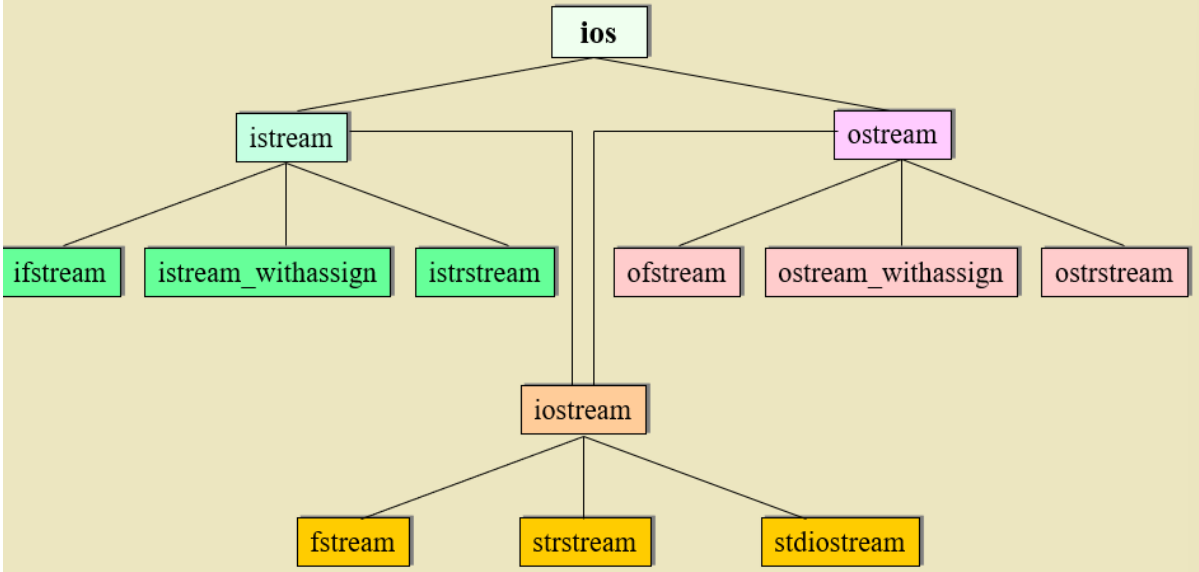
  设置缓冲区　　对缓冲区指针操作　　向缓冲区存/取字符
- ios 类及其派生类提供用户使用流类的接口，支持对streambuf 的缓冲区输入/输出的格式化或非格式化转换

1. streambuf 对缓冲区的低级操作



> filebuf:文件缓冲区管理
>
> strstreambuf:字符串缓冲区管理
>
> stdiobuf:标准I/O缓冲区管理

2. ios　　提供用户接口



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
- **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**
- **每种C++版本还包含其他一些与I/O相关的库，提供特定系统的某些功能**

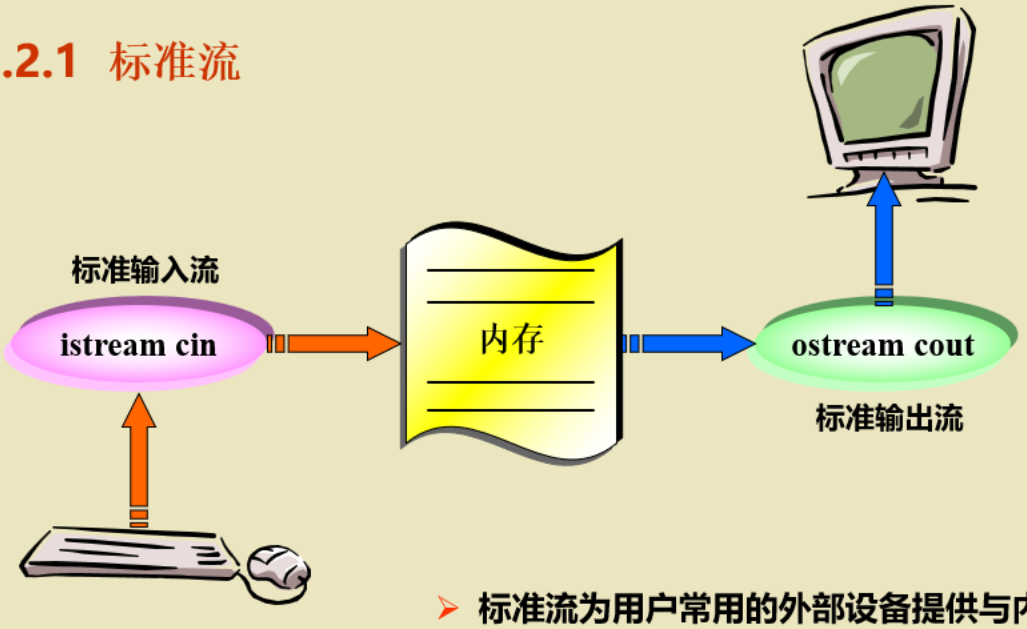
11.2 标准流和对象流

标准流是c++预定义的对象，提供内存与外部设备进行数据交互功能。

流的操作是流类公有的成员函数。

标准流

11.2.1 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

了解:

(1)	cin	istream 类的对象，通常连向键盘，可以重定向
(2)	cout	ostream 类的对象，通常连向显示器，可以重定向
(3)	cerr	ostream 类的对象，连向显示器。不能重定向
(4)	clog	ostream 类的对象，连向打印机。不能重定向

输入流操作

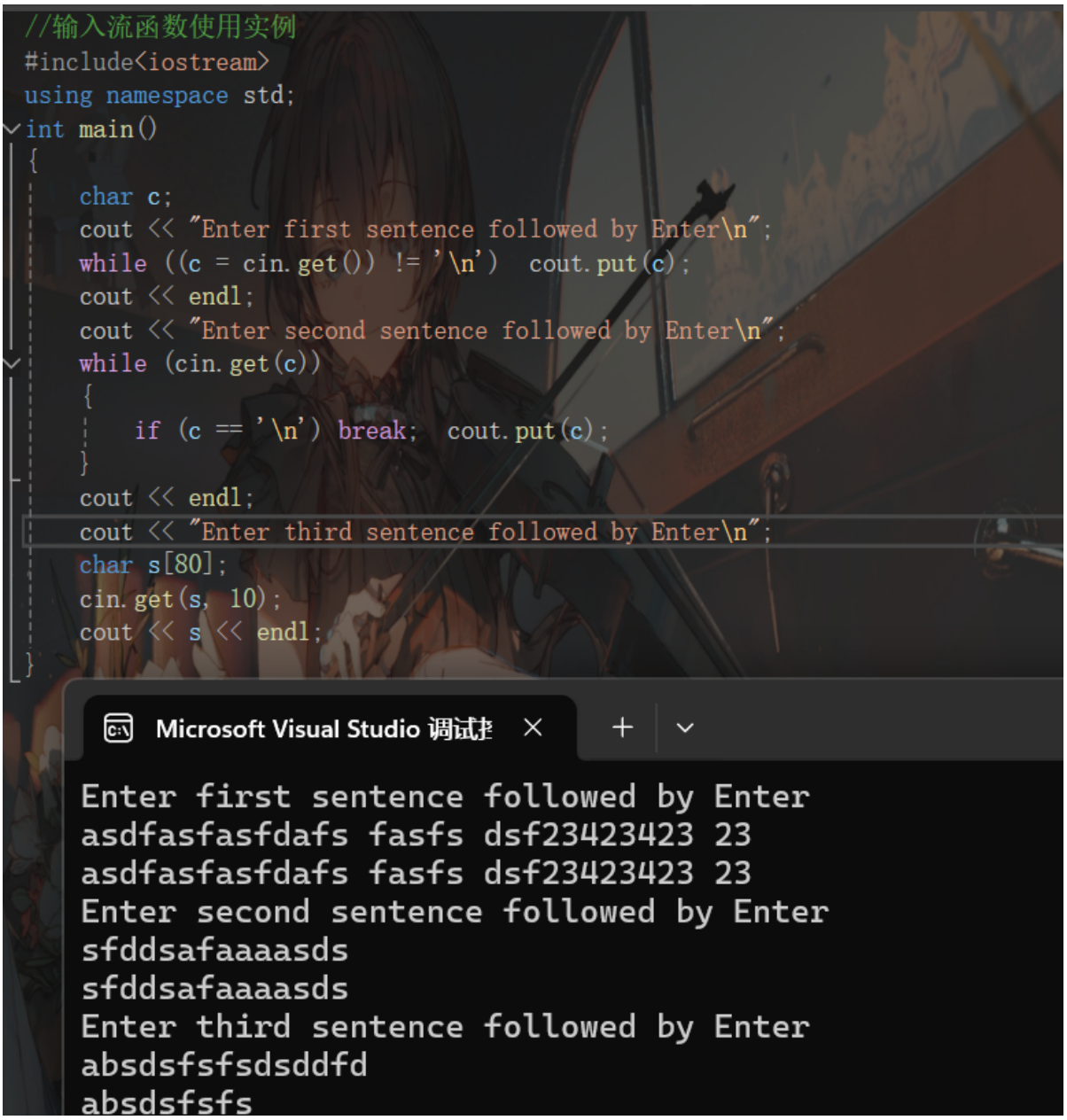
cin类中的函数

istream类的公有成员函数

函数	功能
read	无格式输入指定字节数
get	从流中提取字符，包括空格
getline	从流中提取一行字符
ignore	提取并丢弃流中指定字符
peek	返回流中下一个字符，但不从流中删除
gcount	统计最后输入的字符个数
eatwhite	忽略前导空格
seekg	移动输入流指针
tellg	返回输入流中指定位置的指针值
operstor>>	提取运算符

每个具都有多个重载版本

```
//输入流函数使用实例
#include<iostream>
using namespace std;
int main()
{ char c;
  cout << "Enter first sentence followed by Enter\n" ;
  while ( (c = cin.get()) != '\n' )  cout.put(c);
  cout << endl ;
  cout << "Enter second sentence followed by Enter\n" ;
  while ( cin.get(c) )
      { if ( c=='\n' ) break;  cout.put(c); }
  cout << endl ;
  cout << "Enter third sentence followed by Enter\n" ;
  char s[ 80 ] ;
  cin.get ( s, 10 ) ;
  cout << s << endl ;
}

```



均是通过输入流函数控制输入满足某个值时读取

输出流操作

ostream类的公有成员函数

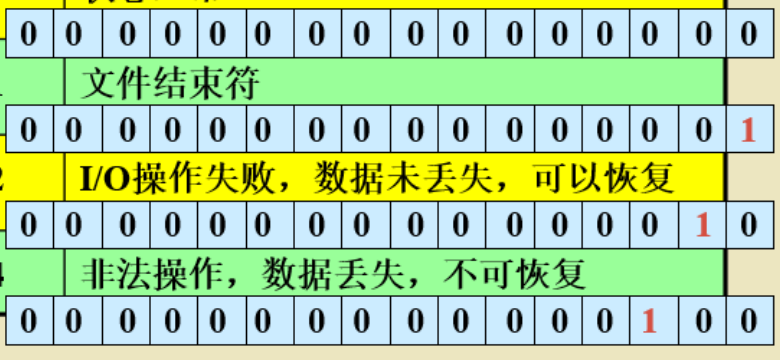
函数	功能
put	无格式, 插入一个字节
write	从无格式, 插入一字节序列
flush	刷新输出流
seekp	移动输出流指针
tellp	返回输出流中指定位置的指针值
operstor<<	插入运算符

每一个函数都有多种重载版本

流错误状态

错误状态字描述

标识常量	值	意义
goodbit	0x00	状态正常
eofbit	0x01	文件结束符
failbit	0x02	I/O操作失败，数据未丢失，可以恢复
badbit	0x04	非法操作，数据丢失，不可恢复



ios 处理流错误状态的公有成员函数

函数	功能
int eof() const;	返回eofbit状态值。文件结束符时返回1，否则返回0
int fail() const;	返回failbit状态值
int good() const; int operator void *() ;	eofbit、failbit和badbit都没有被设置，则返回1
int bad() const; int operator !() ;	eofbit、failbit或badbit其中一个被设置，则返回1
int rdstate() const;	返回状态字
void clear(int *nState* = 0);	恢复或设置状态字

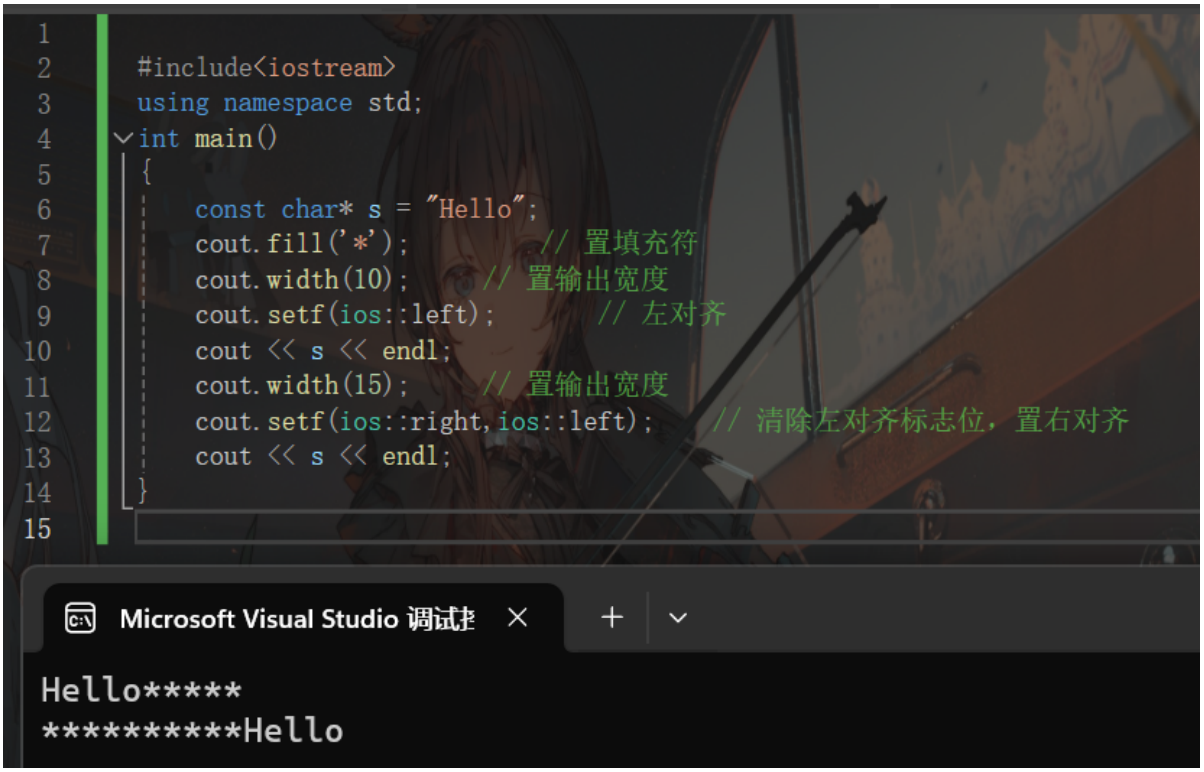
11.3 格式控制

设置标志字

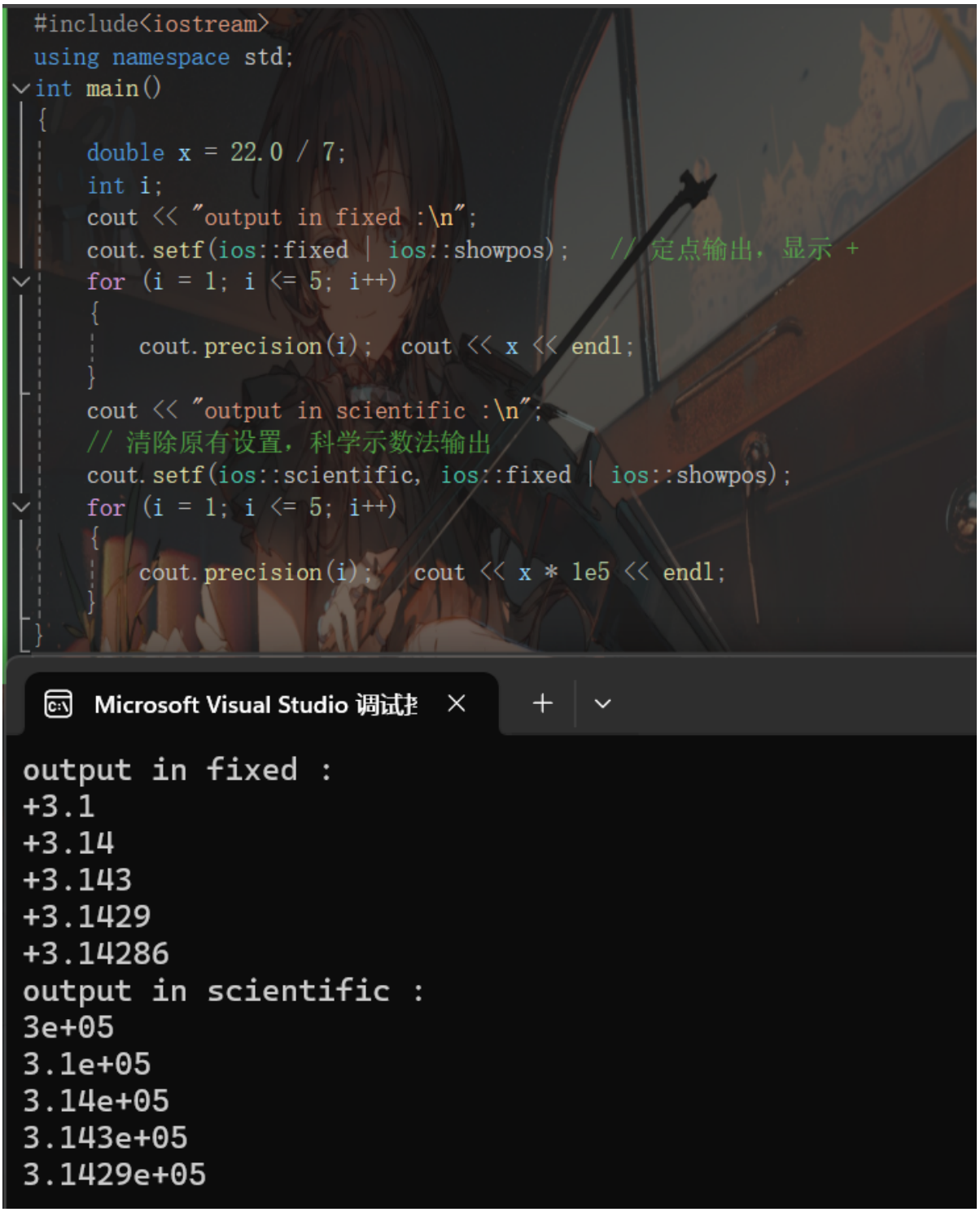
IOS的格式控制通过标志字

状态标志	*值*	*含义*	*输入/输出*
skipws	0X0001	跳过输入中的空白	I
left	0X0002	左对齐输出	O
right	0X0004	右对齐输出	O
internal	0X0008	在符号位和基指示符后填入字符	O
dec	0X0010	转换基制为十进制	I/O
oct	0X0020	转换基制为八进制	I/O
hex	0X0040	转换基制为十六进制	I/O
showbase	0X0080	在输出中显示基指示符	O
showpoint	0X0100	输出时显示小数点	O
uppercase	0X0200	十六进制输出时一律用大写字母	O
showpos	0X0400	正整数前加“+”号	O
scientific	0X0800	科学示数法显示浮点数	O
fixed	0X1000	定点形式显示浮点数	O
unitbuf	0X2000	输出操作后立即刷新流	O
stdio	0X4000	输出操作后刷新stdout 和 stdree	O

在输出之前可以预设输出格式




```cpp
//格式化输出
#include<iostream>
using namespace std;
int main()
{ double x = 22.0/7 ;
   int i ;
   cout << "output in fixed :\n" ;
   cout.setf( ios::fixed | ios::showpos ) ; // 定点输出，显示 +
   for( i=1; i<=5; i++ )
      { cout.precision( i ) ;  cout << x << endl ; }
   cout << "output in scientific :\n" ;
   // 清除原有设置，科学示数法输出
   cout.setf(ios::scientific, ios::fixed|ios::showpos ) ;
   for( i=1; i<=5; i++ )
      { cout.precision(i) ;   cout << x*1e5 << endl ; }
}

```



在取消fixed格式后precision(i)代表保留i为有效数字，而不是保留i位小数

设置格式控制符

控制符是istream和ostream类定义了一批函数，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。（推荐使用格式控制符）

iostream几个常用的控制符

控制符	功能	输入/输出
endl	输出一个新行符，并清空流	O
ends	输出串结束符，并清空流	O
flush	清空流缓冲区	O
dec	用十进制表示法输入或输出数值	I/O
hex	用十六进制表示法输入或输出数值	I/O
oct	用八进制表示法输入或输出数值	I/O
ws	提取空白字符	I

在iomanip头文件中

iomanip的控制符

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

```
//整数的格式化输出
#include <iostream>
#include <iomanip>
using namespace std ;
int main()
{ const int k = 618 ;
   cout << setw(10) << setfill('#') << setiosflags(ios::right) << k <<endl ;
   cout << setw(10) << setbase(8) << setfill('*')
           << resetiosflags(ios::right) << setiosflags(ios::left) << k <<
endl ;
}

```

```cpp
//浮点数的格式化输出
#include<iostream>
using namespace std;
#include <iomanip>
int main()
{ double x = 22.0/7 ;
   int i ;
   cout << "output in fixed :\n" ;
   cout.setf( ios::fixed | ios::showpos ) ;                    // 定点输出，显示
+
   for( i=1; i<=5; i++ )
      { cout.precision( i ) ;  cout << x << endl ; }
   cout << "output in scientific :\n" ;
   // 清除原有设置，科学示数法输出
   cout.setf(ios::scientific, ios::fixed|ios::showpos ) ;
   for( i=1; i<=5; i++ )
      { cout.precision(i) ;   cout << x*1e5 << endl ; }
}

```

11.4 串流

- 串流类是 ios 中的派生类
- C++的串流对象可以连接string对象或字符串
- 串流提取数据时对字符串按变量类型解释；插入数据时把类型数据转换成字符串
- 串流I/O具有格式化功能

输入串流提取对象

```cpp
#include<iostream>
#include<sstream>
using namespace std;
int main()
{
    string testStr("Input test 256 * 0.5");
    string s1, s2, s3;
    double x, y;
    istringstream input(testStr);
    input >> s1 >> s2 >> x >> s3 >> y;
    cout << s1 << " " << s2 << " " << x << s3 << y << "=" << x * y << endl;
}
```

```
Microsoft Visual Studio 调试控
Input test 256*0.5=128
```

这个例子演示了从string对象中提取数据，类似的可以从string指针中提取

input是 istringstream 类的一个对象

输出串流中插入数据

```cpp
#include<iostream>
#include<sstream>
using namespace std;
int main()
{
    ostringstream Output;
    double x, y;
    cout << "Input x : ";
    cin >> x;
    cout << "Input y : ";
    cin >> y;
    Output << x << " * " << y << " = " << x * y << endl;
    cout << Output.str();
}
```

```
Microsoft Visual Studio 调试控
Input x : 1
Input y : 2
1 * 2 = 2
```

这个例子演示了从键盘输入两个数然后通过输出串流插入输出乘法算式，其中Output.str()函数是以字符串格式输出输出流中的元素

Output是ostringstream类的一个对象

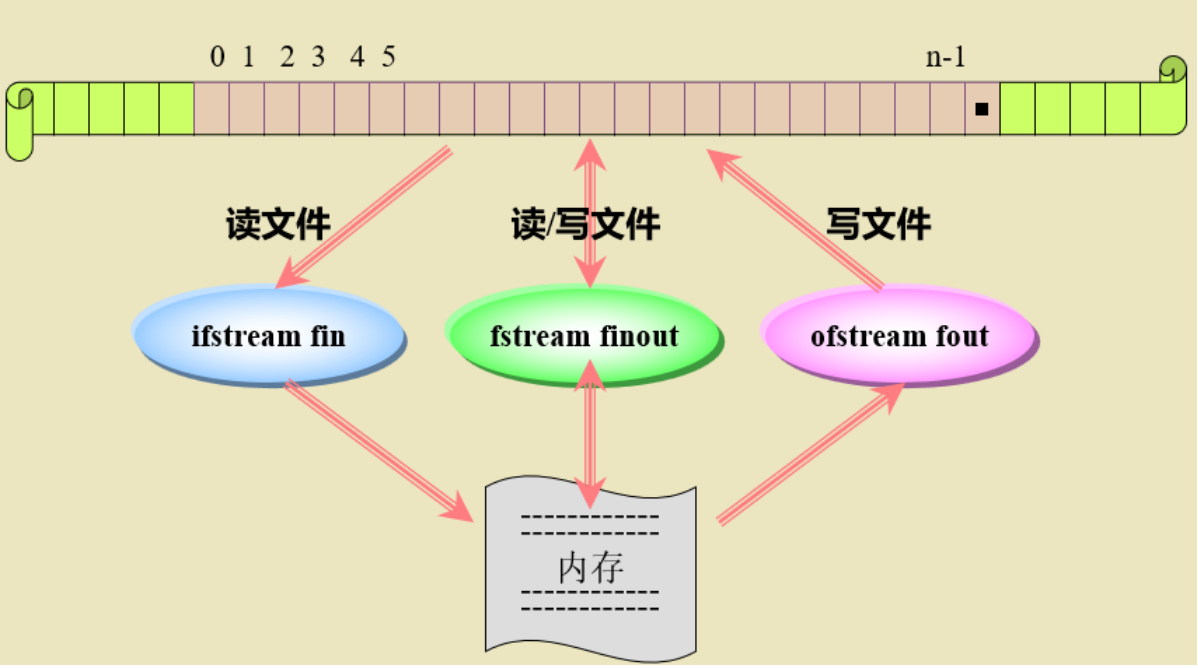
11.5 文件处理

- 各种计算机应用系统通常把一些相关信息组织起来保存在外存储器中，称为文件，并用一个名字（称为文件名）加以标识
- C++把文件看成无结构的字节流，
  编码方式： 文本文件　二进制文件
  存取方式： 顺序文件　随机文件
- ifstream、ofstream 和 fstream 类用于内存与文件之间的数据传输

文本文件本质上还是二进制文件，只不过文本文件一般是通过ascii或者unicode编码，每条数据是定长的

可以通过这篇文章详细了解二进制文件与文本文件的异同

文件和流

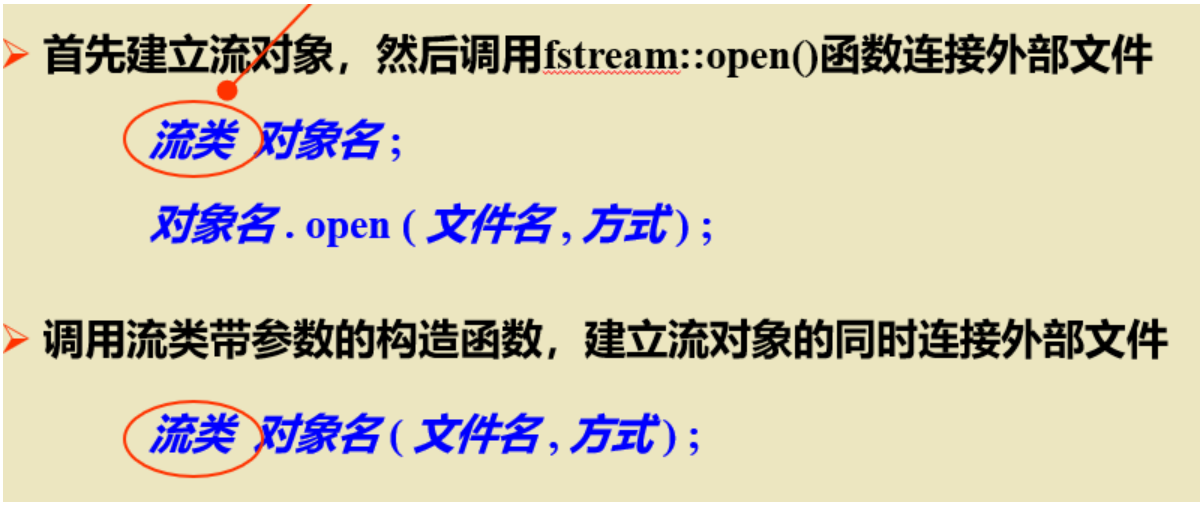


打开和关闭文件

文件操作的基本方式：打开文件->读/写文件->关闭文件

打开文件

两种方式：



open函数用于打开文件

open 函数原型

```
void open ( const char * , int mode , int = filebuf :: openprot ) ;
```

一个参数表示文件名，第二个参数表示打开方式，第三个参数表示文件的保护方式（具有默认值filebuf::openprot)

mode参数包含：

标识常量	值	意义
ios::in	0x0001	读方式打开文件
ios::out	0x0002	写方式打开文件
ios::ate	0x0004	打开文件时，指针指向文件尾
ios::app	0x0008	追加方式
ios::trunc	0x0010	删除文件现有内容
ios::nocreate	0x0020	如果文件不存在，则打开操作失败
ios::noreplace	0x0040	如果文件存在，则打开操作失败
ios::binary	0x0080	二进制方式打开，默认为文本方式

文件保护参数包含：

filebuf::openprot	适应MS-DOS 模式
filebuf::sh_compat	适应MS-DOS 模式
filebuf::sh_none	无模式
filebuf::sh_read	读模式
filebuf::sh_write	写模式

两种打开方式比较

- **打开一个已有文件datafile.dat，准备读：**

```
ifstream infile ;     // 建立输入文件流对象
infile.open( "datafile.dat" , ios::in ) ; // 连接文件，指定打开方式
```

- **打开（创建）一个文件newfile.dat，准备写：**

```
ofstream outfile ;  // 建立输出文件流对象
outfile.open( "d:\\newfile.dat" , ios::out ) ;    // 连接文件，指定打开方式
```

第二种方法 调用文件流类带参数构造函数：

例1

```
ifstream infile ( "datafile.dat" , ios::in ) ;
ofstream outfile ( "d:\\newfile.dat" , ios::out );
fstream rwfile ( "myfile.dat" , ios::in | ios::out ) ;
```

用或运算符 “|” 连接
两个表示打开方式的标识常量

如果不直接指定地址，会默认在cpp所在的地址下。

创建一个文件写的时候可以指定任意的文件格式，即.txt .doc甚至自定义.myfile都可以，区别只是系统能不能打开

关闭文件

- 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
```

```cpp
//open() close()函数应用实例
#include<iostream>
#include<fstream>
using namespace std;
int main ( )
{ ofstream  ost ;
   ost . open ( "c:\\my1.txt ") ;// 要注意这里\\实际上有一个\是转义，因此实际上是C:\my.txt
   ost << 20 << endl ;
   ost << 30.5 << endl ;
   ost . close ( ) ;
   ifstream  ist ( "c:\\my1.txt" ) ;
   int  n ;
   double  d ;
   ist >> n >> d ;
   cout << n << endl << d << endl ;
}
//运行结束可以打开电脑C盘看看是否存在my1.txt文件
```

文本文件

文本文件用默认方式打开

文本文件用文本文件流进行读/写操作

文本文件是顺序存取文件

描述一个对象的信息称为一个记录

文本文件本身没有记录逻辑结构

通常一个逻辑记录用换行符分隔；数据项之间可以用空白符、换行符、制表符等分隔

对文本文件简单操作

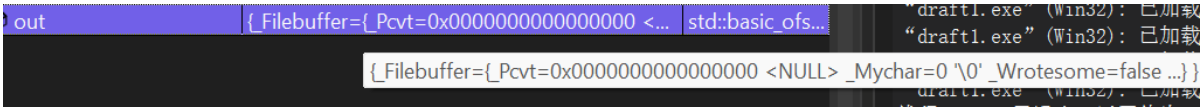
```
//文本文件格式控制
#include<fstream>
#include <iomanip>
using namespace std;
int main ( )
{ ofstream  ost ;
   ost.open ( "d:\\my2.dat" ) ;
   ost << "1234567890" << endl ;
   int  a = 123 ;
   ost << a << endl ;
   ost << setw ( 10 ) << a << endl ;
   ost << resetiosflags ( ios :: right ) << setiosflags ( ios :: left )
        << setfill ( '#' ) << setw ( 10 ) << a << endl ;
   ost << resetiosflags ( ios :: left ) << setiosflags ( ios :: right )
        << setprecision ( 5 ) << setw ( 10 ) << 12.34567890 << endl
;//setfill ( '#' )设定后不需要多次重写，之后的ost输出流默认填充#
   ost . close ( ) ;
}

```

```
//利用if语句判断能否打开文件
#include<iostream>
#include<fstream>
using namespace std;
int main()
{ ofstream out( "d:\\test" ) ;
   if ( !out )
      { cout << "cannot open file. " ;   return 0;  }
   out << 10 << " " << 123.45 << " " ;
   out << "\nThis is a short text file." ;
   out.close () ;
   return 1 ;
```

```
13  }
14
```

如果文件不存在 !out == TRUE。

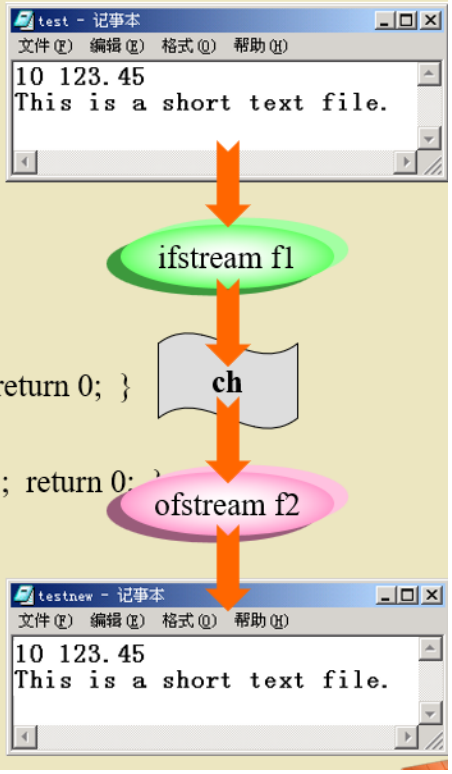
如果文件能打开，则out是一个地址值（指针），如果不能打开，则是一个空指针



一个文本复制的实例

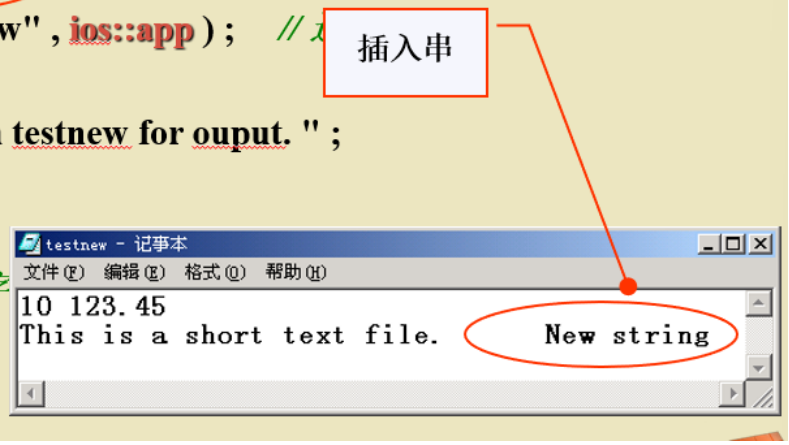
例11-15 把文件 d:\test 复制到文件 d:\testnew 中

```
#include<iostream>
#include <fstream>
using namespace std;
int main ( )
{ char  ch ;
  ifstream  f1 ( "d:\\test" ) ;
  if  ( !f1 )   { cout << "cannot open 'test' for input." ;  return 0;  }
  ofstream  f2 ( "d:\\testnew" ) ;
  if  ( !f2 )   { cout << "cannot open testnew for ouput." ;  return 0;  }
  while ( f1 && f1.get(ch) )   f2.put( ch ) ;
  f1.close () ;
  f2.close () ;
  cout << "It is over !\n" ;
}
```



一个文本追加的实例

```
#include<fstream>
#include<iostream>
using namespace std ;
int main()
{ char str[] = "\tNew string" ;
  ofstream f2 ( "d:\\testnew" , ios::app ) ;    // 
  if ( !f2 )
    { cout << "cannot open testnew for ouput. " ;
      return 0 ;
    }
  f2 << str ;        // 插入字
  f2.close () ;
}
```



要注意：**追加str一定要指定f2的ios::app操作**，如果不写iso::app，str会取代前面的所有内容，因此最好在追加元素之前最好备份文件

实例：读取文本文件，显示每个学生的数据，并计算最大值，最小值，平均分

```
#include<iostream>
#include <fstream>
using namespace std;
int main()
{ char name[30] , s[80] ;
   int number , score ;   int n = 0, max, min, total = 0 ;  double ave;

   ifstream instuf( "d:\\students.txt", ios::in ) ;
   if ( !instuf )
      { cerr << "File could not be open." << endl ;   abort();   }//读取文件

   instuf.getline( s, 80 ) ;//这一行的作用是略去标题行，将指针指向下一行
 while( instuf >> number >> name >> score )
     { cout << number << '\t' << name << '\t' << score << '\n' ;
        if (n==0) { max = min = score; }
      else  { if ( score > max ) max = score ;    if ( score < min ) min = score ;   }
       total+=score;      n++;
     }
   ave = double(total) / n ;
   cout << "maximal is : " << max << endl << "minimal is : " << min << endl
      << "average is : " << ave << endl;
  instuf.close() ;
}

```

```
//浏览文件，跳过n行后输出
#include<iostream>
#include<fstream>
using namespace std;
void browseFile( char * , int );
int main()
{ char fileName[80];
  cout << "请输入要打开的文件名：" ;
  cin >> fileName ;
  browseFile(fileName, 1 );
}
void browseFile( char * fileName, int delLine )
{ ifstream inf( fileName, ios::in ) ;
  char s[80];
  for ( int i=1; i <= delLine; i++ )
      inf.getline( s, 80 ) ;
  while( !inf.eof() )
  { inf.getline( s, 80 ) ;
    cout << s << endl ;
  }//inf.eof()返回的是是否读取到文件末尾，这里!inf.eof()和!inf是等效的
 inf.close() ;
}

```

提取文本数据的其他操作

➢ **两个无格式化提取操作成员函数:**

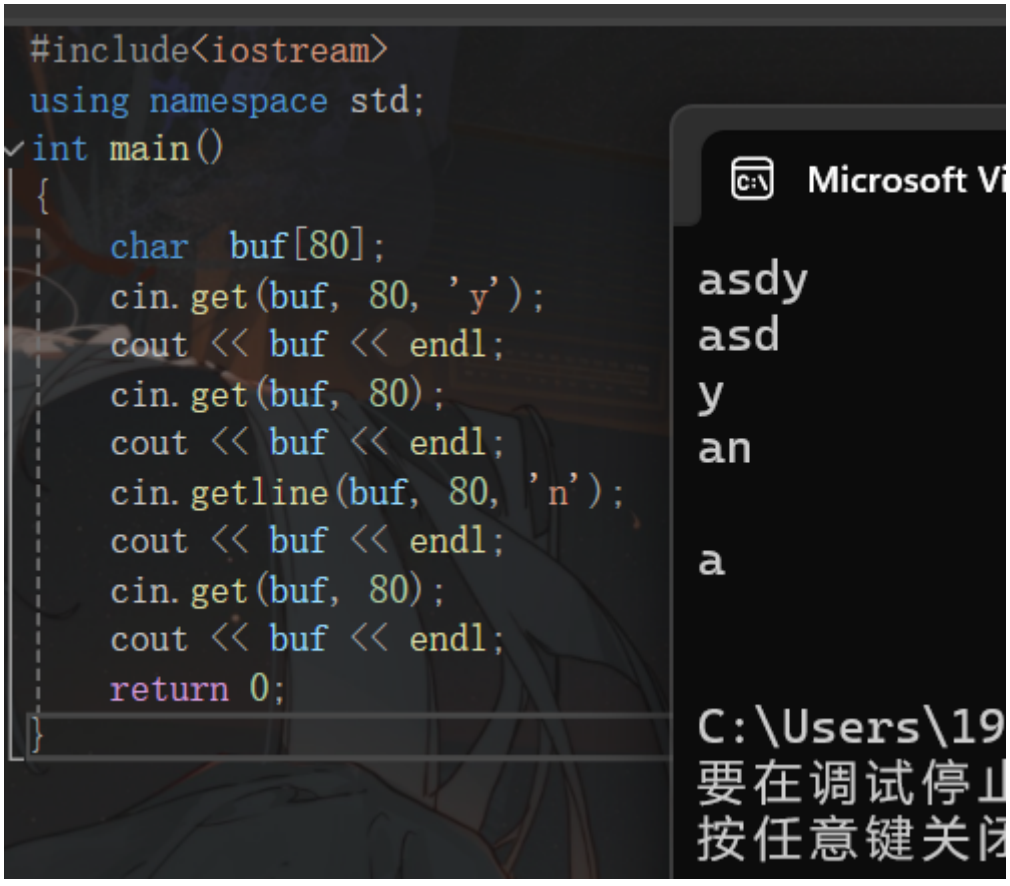
istream & istream :: get (char * , int , char = '\n') ;

istream & istream :: getline (char * , int , char = '\n') ;

***作用*: 从文本中提取指定个数的字符，并在串数组末添加一个空字符**

其中，第一个参数指向接受字符数据的字符数组

```
1  //get()和getline()区别
2  #include<iostream>
3  using namespace std;
4  int main ( )
5  { char  buf [ 80 ] ;
6     cin.get ( buf , 80 , 'y' ) ;
7     cout << buf << endl ;
8     cin.get ( buf , 80 ) ;
9     cout << buf << endl ;
10    cin.getline ( buf , 80 , 'n' ) ;
11    cout << buf << endl ;
12    cin.get ( buf , 80 ) ;
13    cout << buf << endl ;
14    return ;
15 }
```

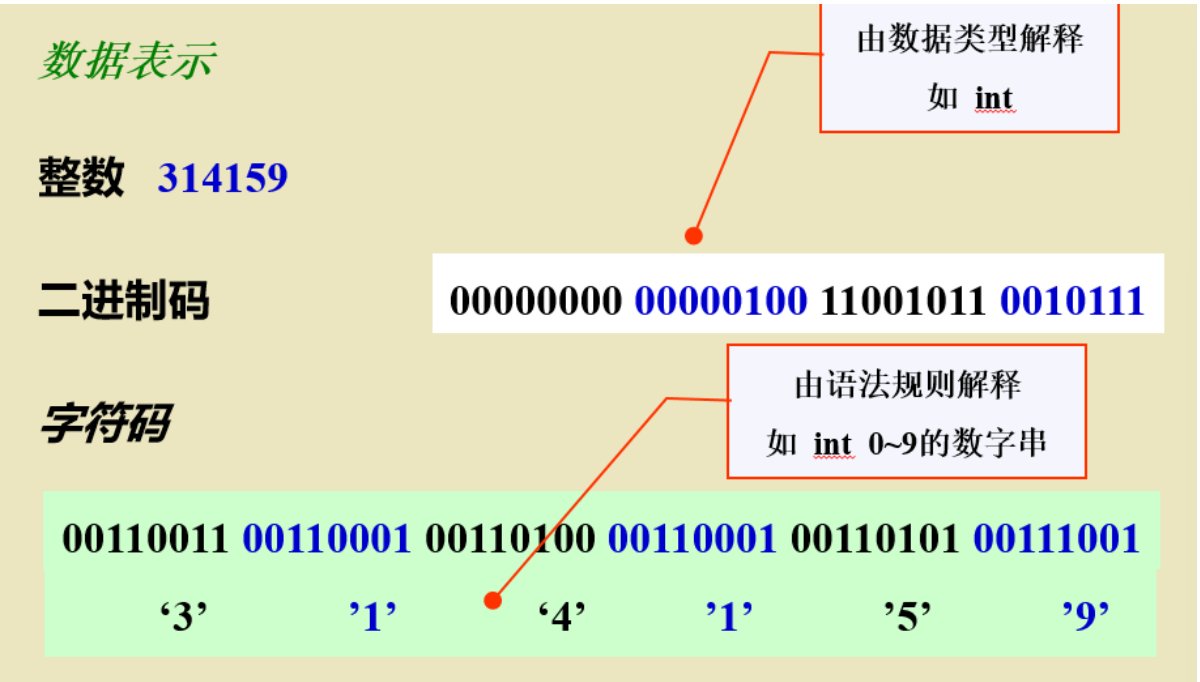


两者都可以通过规定截至的字符规定读取，cin.get()不会将截至的字符读取进存储对象，而cin.getline()会将截止字符读取进，两者默认读取到'\n'截止。要注意：**当读取截止字符标志读取完的时候，还有'\n'在输入流中**

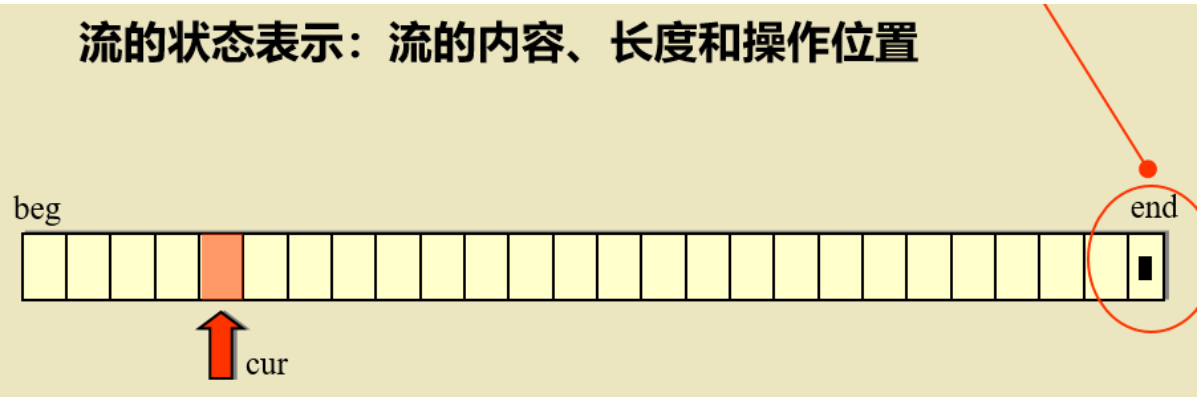
二进制文件

- 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

二进制数据文件



随机访问流



> beg:流起始的指针位置

> cur:流当前指针的位置
>
> end:流末尾的指针位置

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**

- **istream & istream :: tellg () ;**

  作用 **返回读指针当前所指位置值**

一个循环读取某段数据的位置思路

```
istream  input ;    int number ;
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**

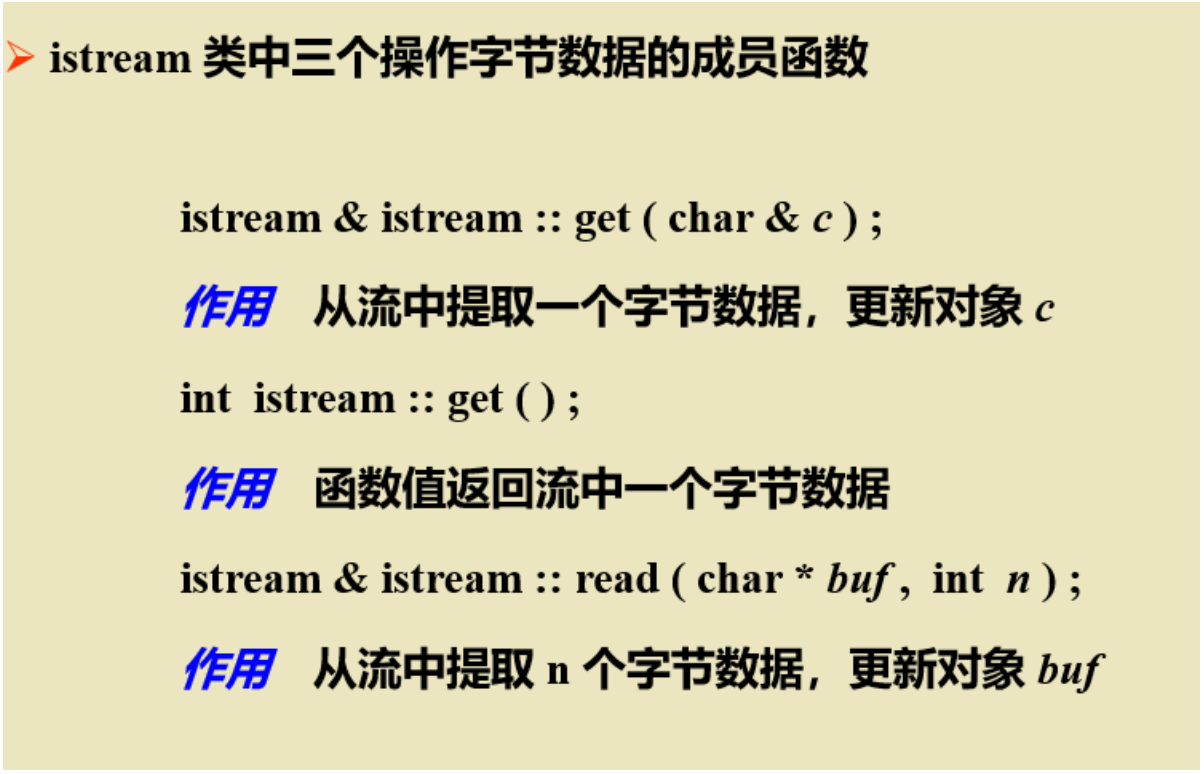
- **ostream & ostream :: tellp ( ) ;**

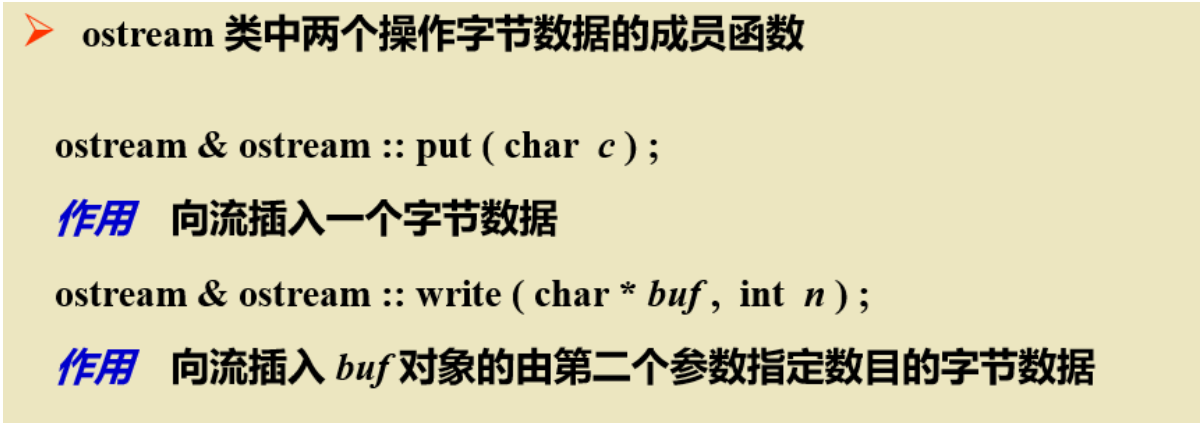
  作用 **返回写指针当前所指位置值**

适于二进制流操作的成员函数

操作对象是流当前指针所指单字节数据

由于二进制文件对于字符的存储和文本文件一样（前提是这些字符都是ascii码编码），这些函数也可以适用于操作存储数据均是ascii码的文本文件，并且不会跳过空白字符





注意：read()函数、write()函数参数一定要强制类型转换为char*

操作实例

```
//一个综合二进制文件操作
#include <fstream>
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    fstream f;
    f.open("C:Data.dat", ios::out | ios::binary);
    for (i = 1; i <= 10; i++)
        f.write((char*)&i, sizeof(int));
    f.close();
```

```
    f.open("C:Data.dat", ios::in | ios::out | ios::binary);
    for (i = 0; i < 10; i++)
    {
        f.seekg(long(sizeof(int) * i));
        f.read((char*)&j, sizeof(int));
        if (j % 2)
        {
            j += 10;
            f.seekp(-long(sizeof(int)), ios::cur);
            f.write((char*)&j, sizeof(int));
        }
    }
    f.seekg(long(sizeof(int) * 10));
    for (i = 91; i <= 95; i++)
        f.write((char*)&i, sizeof(int));
    f.seekg(0, ios::beg);
    for (i = 0; i < 15; i++)
    {
        f.read((char*)&j, sizeof(int));
        cout << j << " ";
    }
    cout << endl;
    f.close();
}

```