Lecture 9.
Stream Input & Output

SMIE-121 Software Design II

zhangzizhen@gmail.com

School of Mobile Information Engineering, Sun Yat-sen University

Outline

Introduction

Design of C++ IOStreams

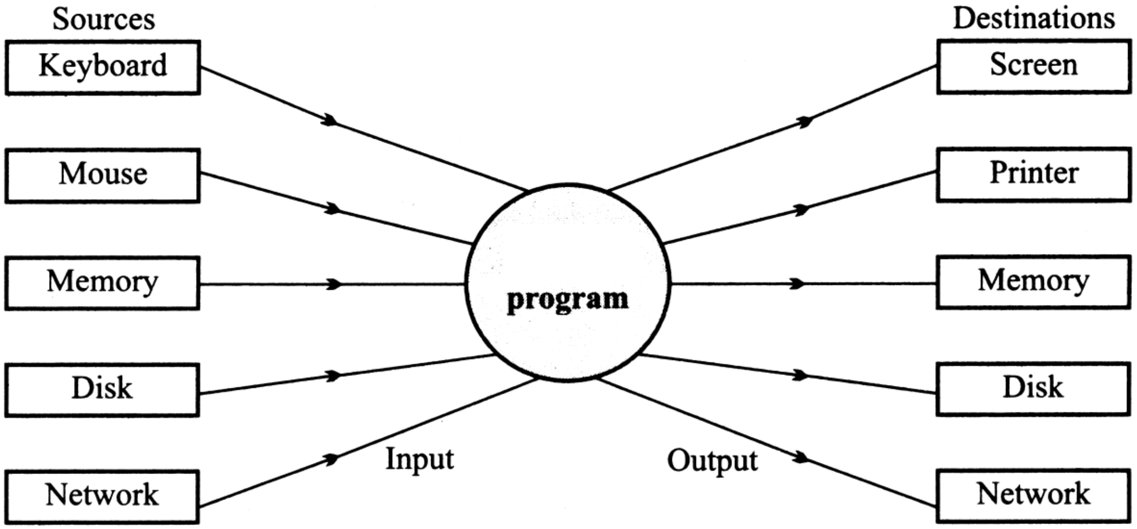
Usage of IOStreams

Introduction

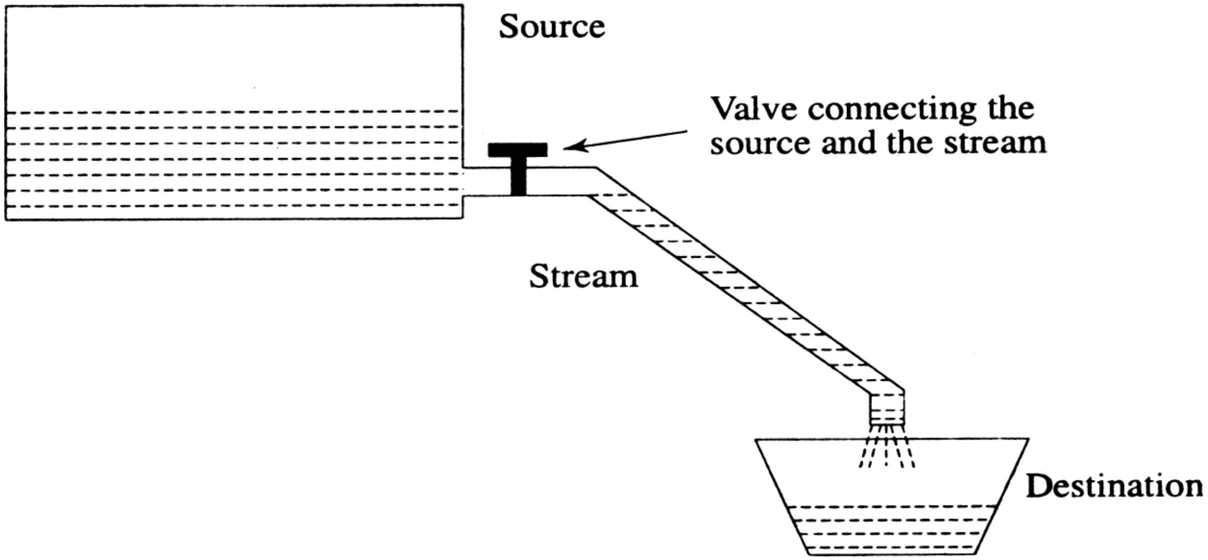
- C语言的输入/输出(input/output)由标准库提供
- 标准库定义了一族类型，支持对文件和控制窗口等设备的读写(IO)
- C++支持C语言中所有的输入输出操作，并对其做了大量的优化

I/O and Data Movement

- The flow of data into a program (input) may come from different devices such as keyboard, mouse, memory, disk, network, or another program.
- The flow of data out of a program (output) may go to the screen, printer, memory, disk, network, another program.
- Both input and output share a certain common property such as unidirectional movement of data – a sequence of bytes and characters and support to the sequential access to the data.



Streams



Conceptual view of a stream

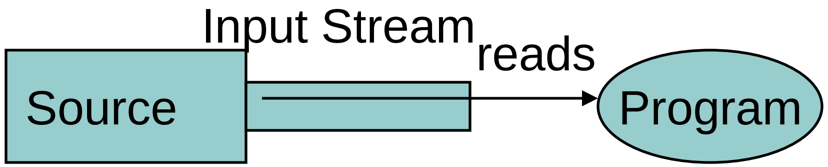
- OO Uses the concept of Streams to represent the ordered sequence of data, a common characteristic shared by all I/O devices.

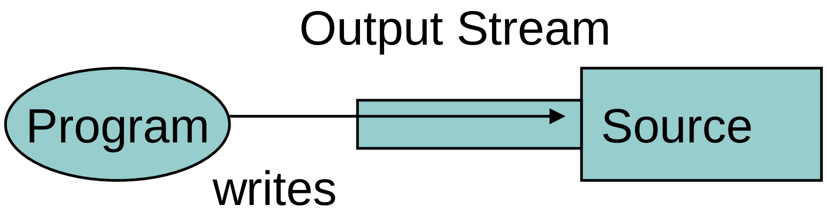
- Streams presents a uniform, easy to use, object oriented interface between the program and I/O devices.

- A stream in OO is a path along which data flows (like a river or pipe along which water flows).

Stream Types

- The concepts of sending data from one stream to another (like a pipe feeding into another pipe) has made streams powerful tool for file processing.
- Connecting streams can also act as filters.
- Streams are classified into two basic types:
  - Input Steam
  - Output Stream





File & In-memory IO

- File I/O involves the transfer of data to and from an external device.
  - The device need not necessarily be a file in the usual sense of the word.
  - It could just as well be a communication channel, or another construct that conforms to the file abstraction.
- In contrast, in-memory I/O involves no external device.
  - Thus code conversion and transport are not necessary; only formatting is performed.
  - The result of such formatting is maintained in memory, and can be retrieved in the form of a character string.

面向对象的标准库

为了管理这种的复杂程度，标准库使用了继承来定义一组面向对象类。IO类型在三个独立的头文件中定义：

- iostream定义读写控制窗口的类型
- fstream定义读写已命名文件的类型
- sstream定义了读写内存中string对象的类型

fstream和sstream都里面定义的每种类型都是从iostream中

IO 标准库类型和头文件

Header	Type
iostream	istream 从流中读取，如cin，这是内置的输入流对象 ostream 写到流中去，如cout，这是内置的输出流对象 iostream 对流进行读写；从 istream 和 ostream 派生而来
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 读写文件；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

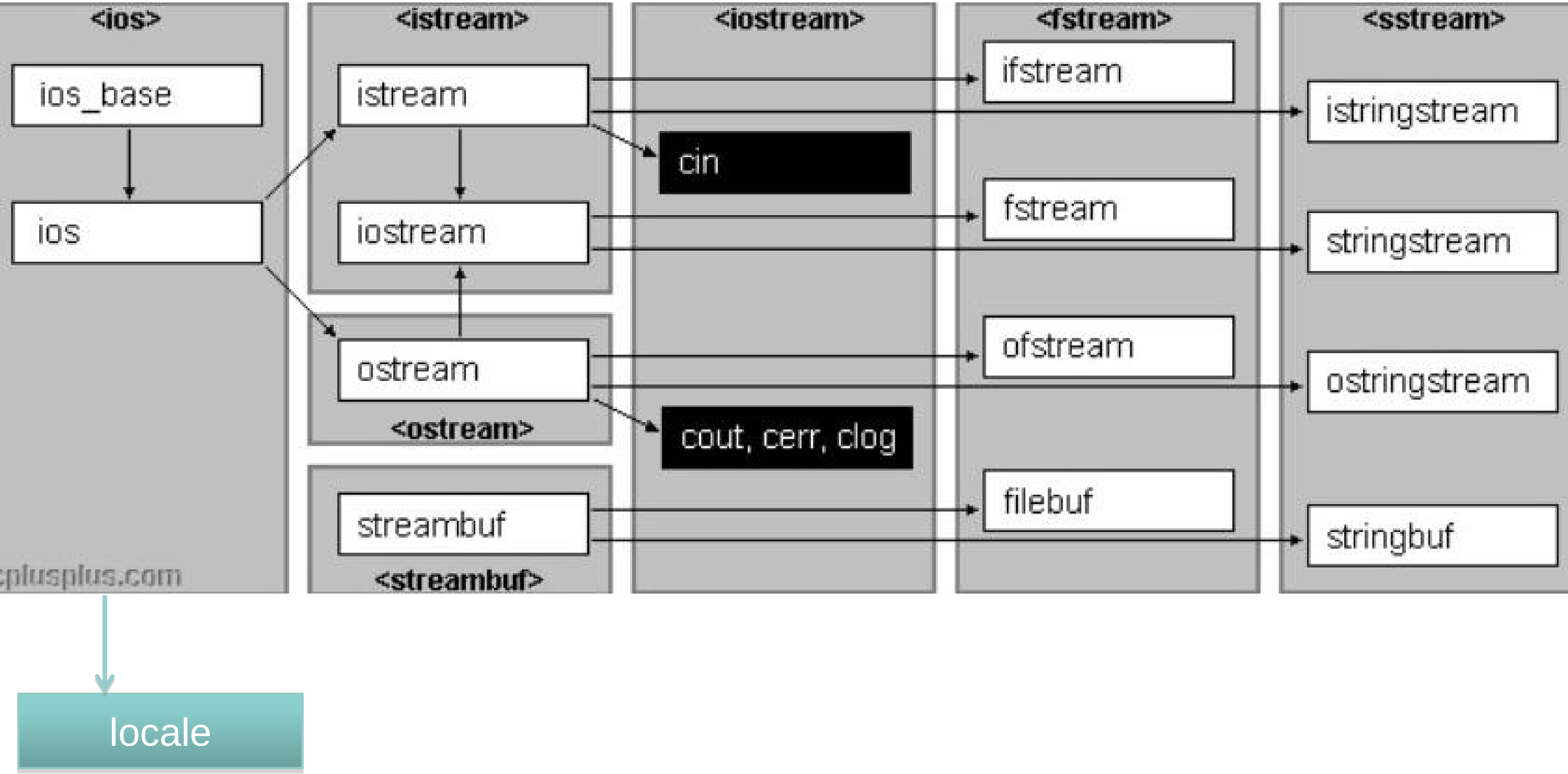
Outline

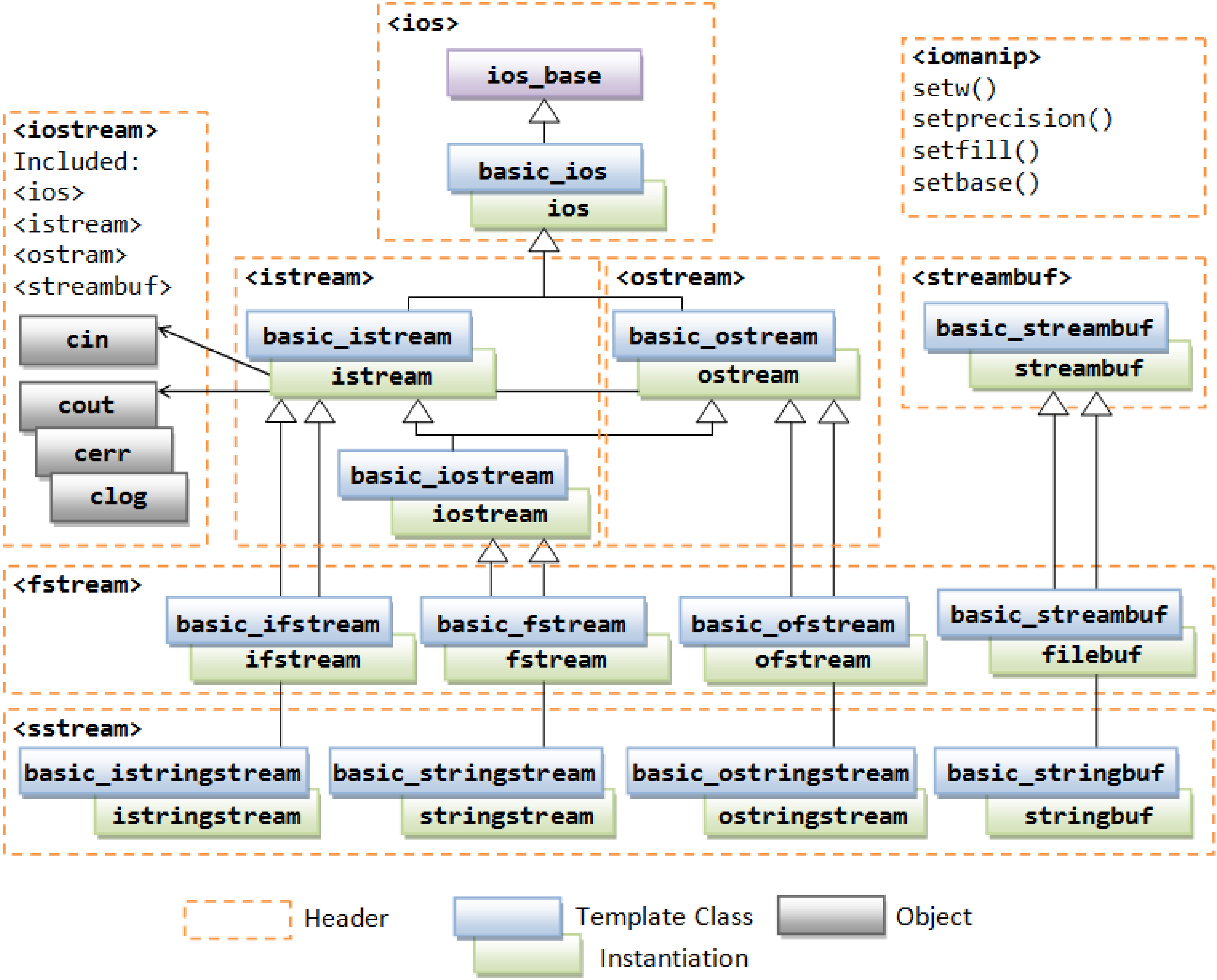
Introduction

Design of C++ IOStreams

Usage of IOStreams

Hierarchy of IOStreams




<ios>
ios_base
basic_ios
ios
<iomanip>
setw()
setprecision()
setfill()
setbase()
<iostream>
Included:
<ios>
<istream>
<ostram>
<streambuf>
cin
cout
cerr
clog
<istream>
basic_istream
istream
<ostream>
basic_ostream
ostream
<streambuf>
basic_streambuf
streambuf
basic_iostream
iostream
<fstream>
basic_ifstream
ifstream
basic_fstream
fstream
basic_ofstream
ofstream
basic_streambuf
filebuf
<sstream>
basic_istringstream
istringstream
basic_stringstream
stringstream
basic_ostringstream
ostringstream
basic_stringbuf
stringbuf
Header
Template Class
Instantiation
Object

输入输出操作符

- 输入(>>)操作符，用于从istream对象中读入输入
- 输出(<<)操作符，用于把输出写到ostream对象中
- C++标准库中对输入/输出操作符进行了重载，使得他们能够输入/输出内置类型。

```
如： cout << 1;          //输出整型
     cout << 1.0;        //输出浮点数
     cout << ‘1’;        //输出字符
     cout << “123”;      //输出字符串
```

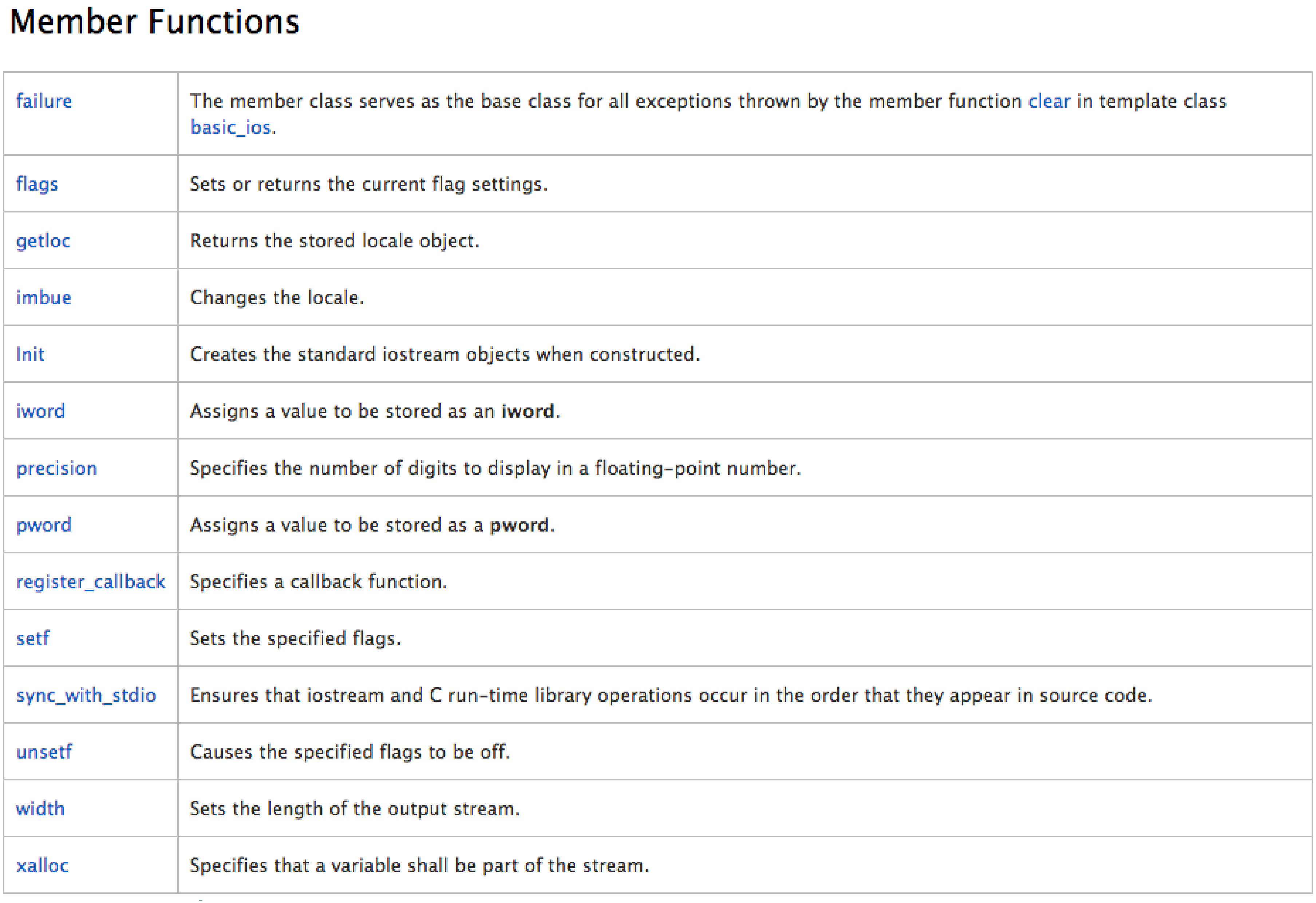
- 用户可对输入输出操作符进行重载，使得它们能够应用于用户自定义类型
- 支持级联(Cascading)：cout << 1 << 2;

<< & >>

- //for input
- istream& operator>>(istream& source, char *pDest);
- istream& operator>>(istream& source, int &dest);
- istream& operator>>(istream& source, char &dest);

- //for output
- ostream& operator<<(ostream& dest, char *pSource);
- ostream& operator<<(ostream& dest, int source);
- ostream& operator<<(ostream& dest, char source);

ios_base



Member Functions

failure	The member class serves as the base class for all exceptions thrown by the member function clear in template class basic_ios.
flags	Sets or returns the current flag settings.
getloc	Returns the stored locale object.
imbue	Changes the locale.
Init	Creates the standard iostream objects when constructed.
iword	Assigns a value to be stored as an **iword**.
precision	Specifies the number of digits to display in a floating-point number.
pword	Assigns a value to be stored as a **pword**.
register_callback	Specifies a callback function.
setf	Sets the specified flags.
sync_with_stdio	Ensures that iostream and C run-time library operations occur in the order that they appear in source code.
unsetf	Causes the specified flags to be off.
width	Sets the length of the output stream.
xalloc	Specifies that a variable shall be part of the stream.

Outline

Introduction

Design of C++ IOStreams

Usage of IOStreams

IO对象不可复制或赋值

- 标准库类型不允许对流对象做复制或赋值操作，这个要求有两层含义：
  - 不能把流对象存储在vector等容器中：因为只有支持复制的元素类型才可以存储在vector等容器中，而流对象不支持复制
  - 形参或返回类型不能为流类型：如果需要传递或返回IO对象，必须传递或返回指向该对象的指针或引用

例子：

```
ostream out1 = cout;        //错误，流对象不能赋值
ostream& out2 = cout;       //正确，可以使用引用
```

- 一般情况下，如果需要传递IO对象对它进行读写，则必须使用非const引用，因为对IO对象的读写会改变它的状态

```cpp
// Fig. 21.11: fig21_11.cpp
// Stream-extraction operator returning false on end-of-file.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

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): ";
   } // end while

   cout << "\n\nHighest grade is: " << highestGrade << endl;
   return 0;
} // end function main
```

```
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
```

ıtput

条件状态

- 实现IO的继承是错误发生的根源，一些错误是可恢复，一些错误则发生在系统底层，位于程序可修正的范围之外。
- IO标准库管理一系列**条件状态**(conition state)成员，用来标记IO所处的状态。

IO标准库的条件状态

strm::iostate	机器相关的整型名，由各个iostream类定义，用于定义条件状态
strm::badbit	strm::iostate类型的值，用于指出被破坏的流
strm::failbit	strm::iostate类型的值，用于指出失败的IO操作
strm::eofbit	strm::iostate类型的值，用于指出流已经到达文件结束符
s.eof()	如果设置了流s的eofbit值，则该函数返回true
s.fail()	如果设置了流s的failbit值，则该函数返回true
s.bad()	如果设置了流s的badbit值，则该函数返回true
s.good()	如果流s处于有效状态，则该函数返回true
s.clear()	将流s中的所有状态值都重设为有效状态
s.clear(flag)	将流s中的某个指定条件状态设置为有效。flag的类型是strm::iostate
s.setstate(flag)	给流s添加指定条件。flag的类型是strm::iostate
s.rdstate()	返回流s的当前条件，返回值类型为strm::iostate

条件状态

考虑下面例子：

```
int ival;
cin >> ival;
```

- 如果输入Borges，则cin在尝试将输入的字符串读为int型数据失败后，会生成一个错误状态
- 如果输入文件结束符（end-of-file），cin 也会进入错误状态
- 如果输入 1024，则成功读取，cin 将处于正确的无错误状态

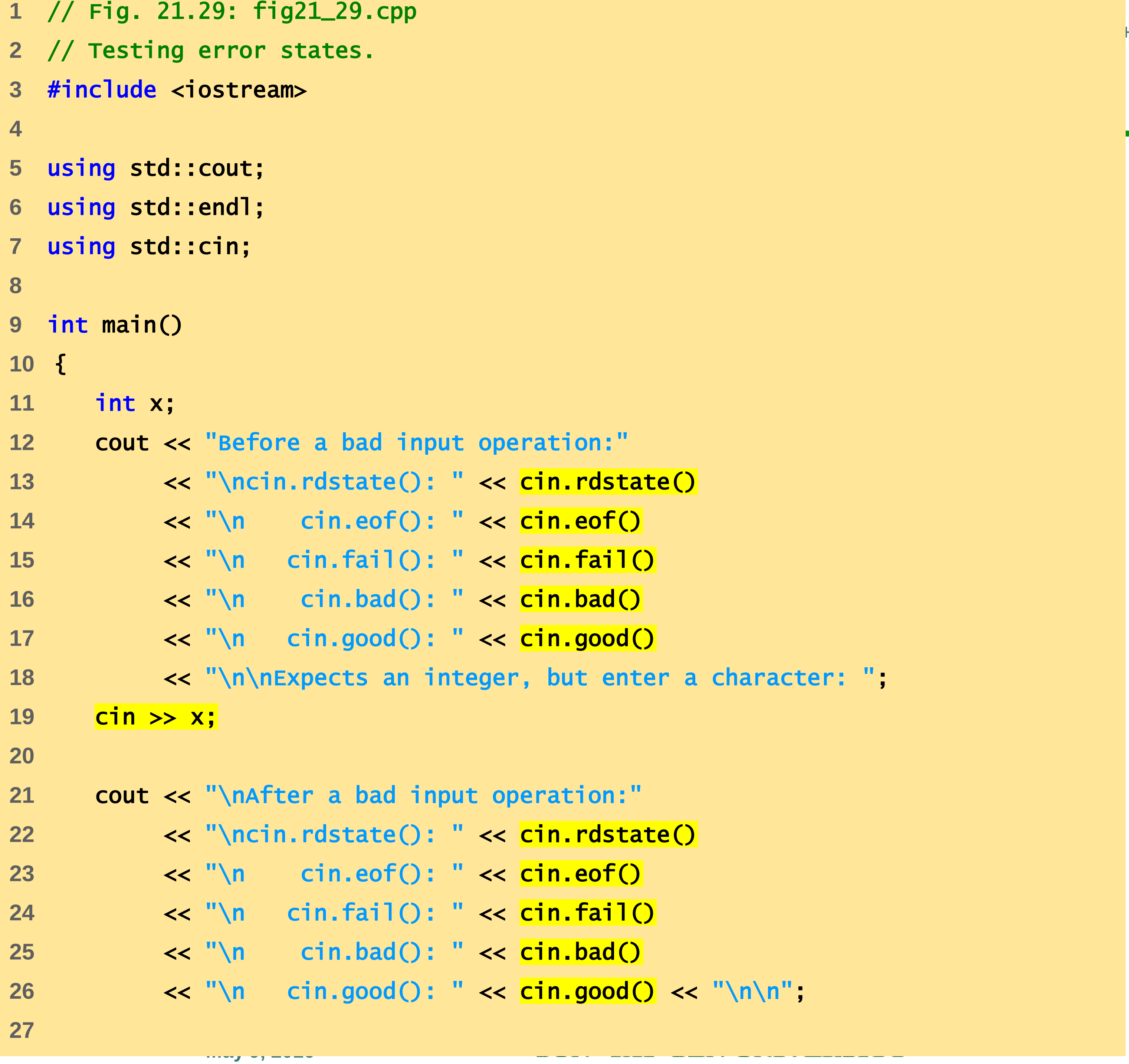
条件状态

流必须处于无错误状态，才能用于输入或输出。检测流是否用的最简单的方法是检查其真值：

```
if (cin)
    // ok to use cin, it is in a valid state
while (cin >> word)
    // ok: read operation successful ...
```

if 语句直接检查流的状态，而 while语句则检测条件表达式返回的流，从而间接地检查了流的状态。如果成功输入，则条件检测为 true。

同学们可以思考一下，为什么IO对象（如上面的cin）可以直接用于条件判断



```cpp
// Fig. 21.29: fig21_29.cpp
// Testing error states.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int x;
   cout << "Before a bad input operation:"
        << "\ncin.rdstate(): " << cin.rdstate()
        << "\n    cin.eof(): " << cin.eof()
        << "\n   cin.fail(): " << cin.fail()
        << "\n    cin.bad(): " << cin.bad()
        << "\n   cin.good(): " << cin.good()
        << "\n\nExpects an integer, but enter a character: ";
   cin >> x;

   cout << "\nAfter a bad input operation:"
        << "\ncin.rdstate(): " << cin.rdstate()
        << "\n    cin.eof(): " << cin.eof()
        << "\n   cin.fail(): " << cin.fail()
        << "\n    cin.bad(): " << cin.bad()
        << "\n   cin.good(): " << cin.good() << "\n\n";

   cin.clear();

   cout << "After cin.clear()"
        << "\ncin.fail(): " << cin.fail()
        << "\ncin.good(): " << cin.good() << endl;
   return 0;
} // end function main
```

```
Before a bad input operation:
cin.rdstate(): 0
    cin.eof(): 0
   cin.fail(): 0
    cin.bad(): 0
   cin.good(): 1

Expects an integer, but enter a character: A

After a bad input operation:
cin.rdstate(): 2
    cin.eof(): 0
   cin.fail(): 1
    cin.bad(): 0
   cin.good(): 0

After cin.clear()
cin.fail(): 0
cin.good(): 1
```

输出缓冲区的管理

- 每个IO对象管理一个输出缓冲区，用于存储程序读写的数据，如语句：

```
cout << “please enter a value”;
```

系统将字符串字面值存储在cout的缓冲区中，并没有输出到设备或者文件中，如上面的语句并没有马上显示在控制窗口中。

- 缓冲区被刷新的时候，缓冲区中的内容会被写入真实的输出设备或者文件中。

输出缓冲区的刷新

下面的几种情况会使得缓冲区被刷新：

- 程序正常结束
- 缓冲区已经满了。在这种情况下，缓冲区将会在写下一个值之前刷新
- 用操纵符显式地刷新缓冲区，例如endl
- 使用unitbuf操作符设置流内部状态
- 将输出流与输入流关联(tie)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

用操纵符刷新缓冲区

C++提供了三个操纵符用于刷新缓冲区：

- endl：用于输出一个换行符并刷新缓冲区
- flush：用于刷新流，但不在输出中添加任何字符
- ends：在缓冲区中插入空字符null，并刷新缓冲区

```
cout << “hi!” << flush;
cout << “hi!” << ends;
cout << “hi!” << endl;
```

unitbuf操纵符

如果需要刷新所有输出，最好使用unitbuf操纵符。这个操纵符在每次执行完后都刷新流：

```
cout << unitbuf << “first” << “second”;
```

等价于

```
cout << “first” << flush << “second” << flush;
```

若要取消unitbuf的作用可以使用nounitbuf操纵符，它将流恢复为使用正常的、由系统管理的缓冲区刷新方式

文件的输入输出

fstream头文件中定义了三种支持文件IO的类型：

- ifstream，由istream派生而来，提供读文件功能
- ofstream，由ostream派生而来，提供写文件功能
- fstream，由iostream派生而来，提供读写同一个文件的功能

这些类型都有相应的iostream类型派生而来，所以iostream上所有的操作适用于fstream中的类型，同样，前面提到的条件状态也同样适合。

文件流对象的使用

- cin、cout、cerr是标准库定义的对象，可直接使用。当需要读写文件时，必须定义自己的对象，并将它绑定在需要的文件上。

```
ifstream infile(“in.txt”);
ofstream outfile(“out.txt”);
```

上述代码定义并打开一对fstream对象。infile是读的流，outfile是写的流。

- 可以使用语句 if(infile) 来判断是否成功打开文件

文件模式

每个fstream类都定义了一组表示不同模式的值，用于指定流打开的不同模式。下表列出了文件模式及其含义：

in	打开文件做读操作
out	打开文件做写操作
app	在每次写之前找到文件尾
ate	打开文件后立即将文件定位在文件尾
trunc	打开文件时清空已存在的文件流
binary	以二进制模式进行 IO 操作

文件模式

- out、trunk和app模式只能用于指定与ofstream或fstream对象关联的文件
- in模式只能用于指定与ifstream或fstream对象关联的文件
- 所有文件都可以用ate或binary模式打开

```
// 使用默认打开方式，即out和trunc模式，会清空文件file1
ofstream out1(“file1”)
// 使用模式out和trunc打开file1
ofstream out2(“file1”, ofstream::out | ofstream::trunc)
// 使用app模式打开file1，保存文件的数据并在最后添加数据
ofstream out3(“file1”, ofstream::app)
```

字符串流

头文件sstream包含三种类型的字符串流：

- istringstream，由istream派生而来，提供读string功能
- ostringstream，由ostream派生而来，提供写string功能
- stringstream，由iostream派生而来，提供读写string的功能

与fstream类型一样，上诉类型由iostream派生而来，所以iostream上所有的操作适用于中的类型

stringstream特定的操作

stringstream strm;	创建自由的stringstream对象
stringstream strm(s);	创建存储s的副本的stringstream对象，其中s是string类型的对象
strm.str();	返回strm中存储的string类型对象
strm.str(s);	将string类型的s复制给strm，返回void

stringstream中类中存有一个string对象，对stringstream的读写操作实际上是对该对象中的string对象进行读写

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

int main ()
{
     // constructs a stringstream object with an empty
     sequence as content.
     std::stringstream ss;

     // write data to the buffer of stringstream object
     ss << 100 << ' ' << 200;

     int foo,bar;
     // read data from the buffer of stringstream object
     ss >> foo >> bar;

     std::cout << "foo: " << foo << '\n';
     std::cout << "bar: " << bar << '\n';

     return 0;
}
```

output

```
foo: 100
bar: 200
```

格式状态

- 除了条件状态外，每个iostream对象还位置一个控制IO格式化细节的格式状态
- 格式状态控制格式化特征，包括：
  - 输出元素的宽度
  - 浮点数的的格式，如精度、记数法等
  - 整型值的基数，如十进制、十六进制等
  - 其他一些格式化特征
- 标准库定义了一组操纵符来修改对象的格式状态

iostream中定义的操纵符(1)

	boolalpha	将真和假显示为字符串
x	noboolalpha	将真和假显示为1，0
	showbase	产生指出数的基数的前缀
x	noshowbase	不产生记数基数前缀
	showpoint	总是显示小数点
x	noshowpoint	有小数部分才显示小数点
	showpos	显示非负数中的 +
x	noshowpos	不显示非负数中的 +
	uppercase	在十六进制中打印 0X，科学记数法中打印 E
x	nouppercase	在十六进制中打印 0x，科学记数法中打印 e
x	dec	用十进制显示
	hex	用十六进制显示
	oct	用八进制显示
	left	在值的右边增加填充字符
	right	在值的左边增加填充字符

注：带x的是默认流状态

iostream中定义的操纵符(2)		
	internal	在符号和值之间增加填充字符
	fixed	用小数形式显示浮点数
	scientific	用科学记数法显示浮点数
	flush	刷新 ostream 缓冲区
	ends	插入空字符，然后刷新 ostream 缓冲区
	endl	插入换行符，然后刷新 ostream 缓冲区
	unitbuf	在每个输出操作之后刷新缓冲区
x	nounitbuf	恢复常规缓冲区刷新
x	skipws	为输入操作符跳过空白
	noskipws	不为输入操作符跳过空白
	ws	“吃掉”空白
注：带x的是默认流状态		

iomanip中定义的操纵符	
setfill(ch)	用ch填充空白
setprecision(n)	将浮点精度置为0
setw(w)	读写w个字符的值
setbase(b)	按基数b输出整数

读写操纵符的时候，不读写数据，相反，会采取某种行动。如前面使用过的一个操纵符endl，我们将它“写至输出流”，就好像它是一个值一样，但endl并不是一个值，相反，它执行一个操作：写换行符并刷新缓冲区

例子：指定输出的基数

```cpp
// Fig. 21.16: fig21_16.cpp
// Using hex, oct, dec and setbase stream manipulators.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <iomanip>

using std::hex;
using std::dec;
using std::oct;
using std::setbase;

int main()
{
   int n;

   cout << "Enter a decimal number: ";
   cin >> n;

   cout << n << " in hexadecimal is: "
        << hex << n << '\n'
        << dec << n << " in octal is: "
        << oct << n << '\n'
        << setbase( 10 ) << n << " in decimal is: "
        << n << endl;

   return 0;
} // end function main
```

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 20
```

例子：指定显示的精度

```cpp
// Fig. 21.17: fig21_17.cpp
// Controlling precision of floating-point values
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <iomanip>

 using std::ios;
 using std::setiosflags;
 using std::setprecision;

 #include <cmath>

 int main()
 {
    double root2 = sqrt( 2.0 );
    int places;

    cout << setiosflags( ios::fixed )
         << "Square root of 2 with precisions 0-9.\n"
         << "Precision set by the "
         << "precision member function:" << endl;

```

```cpp
   for ( places = 0; places <= 9; places++ ) {
      cout.precision( places );
      cout << root2 << '\n';
   } // end for

   cout << "\nPrecision set by the "
        << "setprecision manipulator:\n";

   for ( places = 0; places <= 9; places++ )
      cout << setprecision( places ) << root2 << '\n';

   return 0;
} // end function main
```

```
Square root of 2 with precisions 0-9.
Precision set by the precision member function:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by the setprecision manipulator:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

例子：控制输出的宽度

```cpp
// fig21_18.cpp
// Demonstrating the width member function
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
    int w = 4;
    char string[ 10 ];

    cout << "Enter a sentence:\n";
    cin.width( 5 );

    while ( cin >> string ) {
        cout.width( w++ );
        cout << string << endl;
        cin.width( 5 );
    } // end while

    return 0;
} // end function main
```

```
Enter a sentence:
This is a test of the width member function
This
   is
     a
   test
      of
      the
      widt
          h
        memb
           er
          func
           tion
```

未格式化的输入/输出操作

- 迄今为止，示例程序只使用过格式化的IO操作，输入输出操作符根据处理数据的类型格式化所读写的数据。
- 标准库中还提供了丰富的支持未格式化IO的低级操作，这些操作使我们能够将流作为未解释的字节序列处理，而不是作为数据类型（如char、int、string等）的序列处理

单字节低级IO操作	
is.get(ch)	将 istream is 的下一个字节放入 ch，返回 is
os.put(ch)	将字符 ch 放入 ostream，返回 os
is.get()	返回 is 的下一字节作为一个 int 值
is.putback(ch)	将字符 ch 放回 is，返回 is
is.unget()	将 is 退回一个字节，返回 is
is.peek()	将下一字节作为 int 值返回但不移出它

上述表格中的未格式化的操作一次一个字节地处理流，它们不忽略空白地读

```cpp
// Fig. 21.12: fig21_12.cpp
// Using member functions get, put and eof.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   char c;

   cout << "Before input, cin.eof() is " << cin.eof()
      << "\nEnter a sentence followed by end-of-file:\n";

   while ( ( c = cin.get() ) != EOF )
      cout.put( c );

   cout << "\nEOF in this system is: " << c;
   cout << "\nAfter input, cin.eof() is " << cin.eof() << endl;
   return 0;
} // end function main
```

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input cin.eof() is 1
```

多字节低级IO操作

is.get(sink, size, delim)	从 is 中读 size 个字节并将它们存储到 sink 所指向的字符数组中。读操作直到遇到delim 字符，或已经读入了 size 个字节
is.get(sink, size, delim)	或遇到文件结束符才结束。如果出现了delim，就将它留在输入流上，不读入到 sink 中。
is.getline(sink, size, delim)	与三个实参的 get 行为类似，但读并丢弃 delim
is.read(sink, size)	读 size 个字节到数组 sink。返回 is
is.gcount()	返回最后一个未格式化读操作从流 is 中读到的字节数
os.write(source, size)	将 size 个字从数组 source 写至 os。返回os
is.ignore(size, delim)	读并忽略至多 size 个字符，直到遇到 delim ，但不包括 delim。默认情况下，size 是 1 而 delim 是文件结束符

```cpp
// Fig. 21.13: fig21_13.cpp
// Contrasting input of a string with cin and cin.get.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

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;
} // end function main
```

```
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
```

```cpp
// Fig. 21.14: fig21_14.cpp
// Character input with member function getline.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   const SIZE = 80;
   char buffer[ SIZE ];

   cout << "Enter a sentence:\n";
   cin.getline( buffer, SIZE );

   cout << "\nThe sentence entered is:\n" << buffer << endl;
   return 0;
} // end function main
```

```
Enter a sentence:
Using the getline member function

The sentence entered is:
Using the getline member function
```

Thank you!