



第五章 文件的操作

模块5.2: C++方式的文件操作

主讲教师: 同济大学电子与信息工程学院 陈宇飞
同济大学电子与信息工程学院 龚晓亮



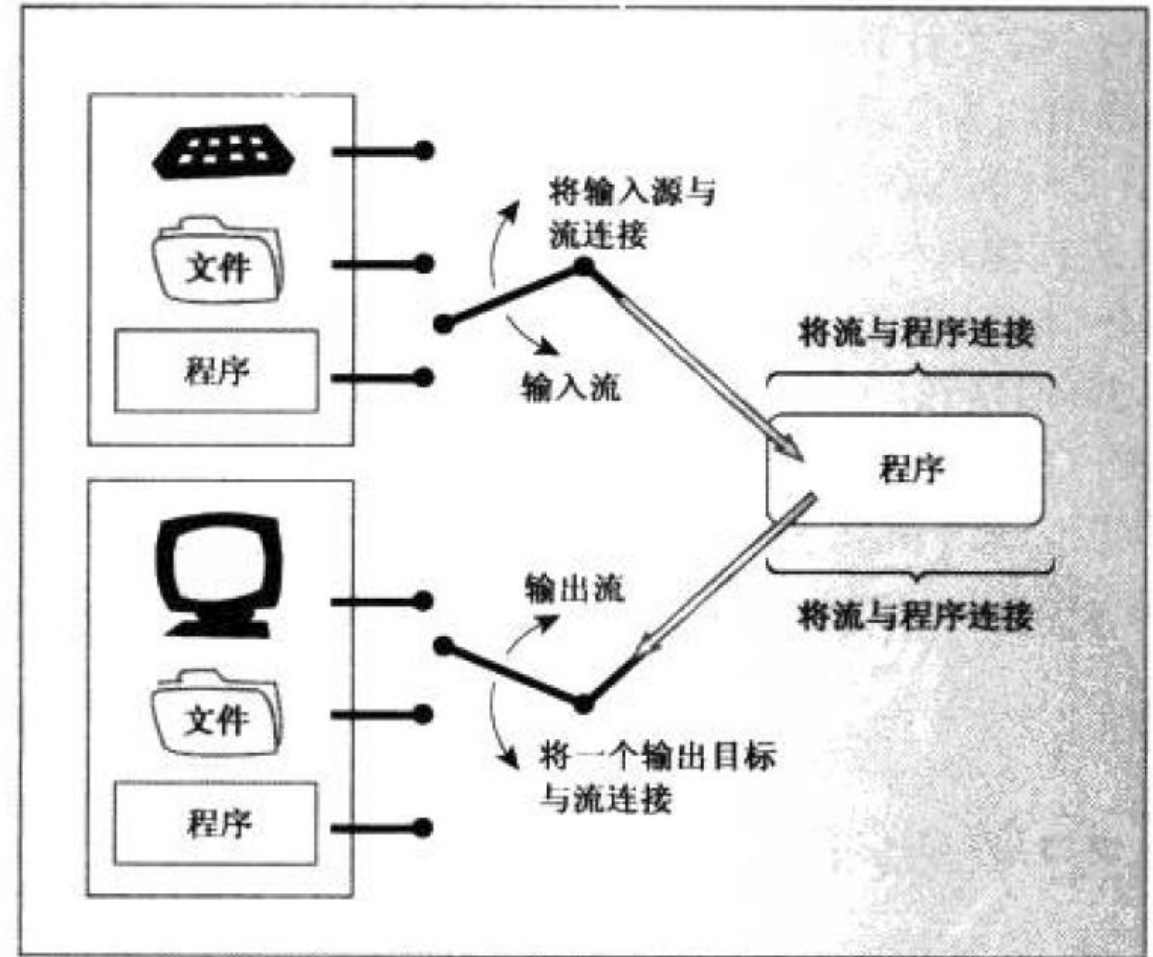
目录

- 标准输入流
- C++方式的文件操作与文件流
- C++方式的字符串流
- C方式类似字符串流的操作

5.2.1 标准输入流

1. 流和缓冲区

- C++将输入和输出看作字节流。**输入**时，程序从输入流中**抽取**字节；**输出**时，程序将字节**插入**输出流。
- 输入流中的字节可能来自键盘以及存储设备或其他程序
- 输出流中的字节可以流向屏幕、打印机、存储设备或其他程序
- 管理输入包括：1) 将流与输入去向的程序关联起来；2) 将流与文件端连接起来。

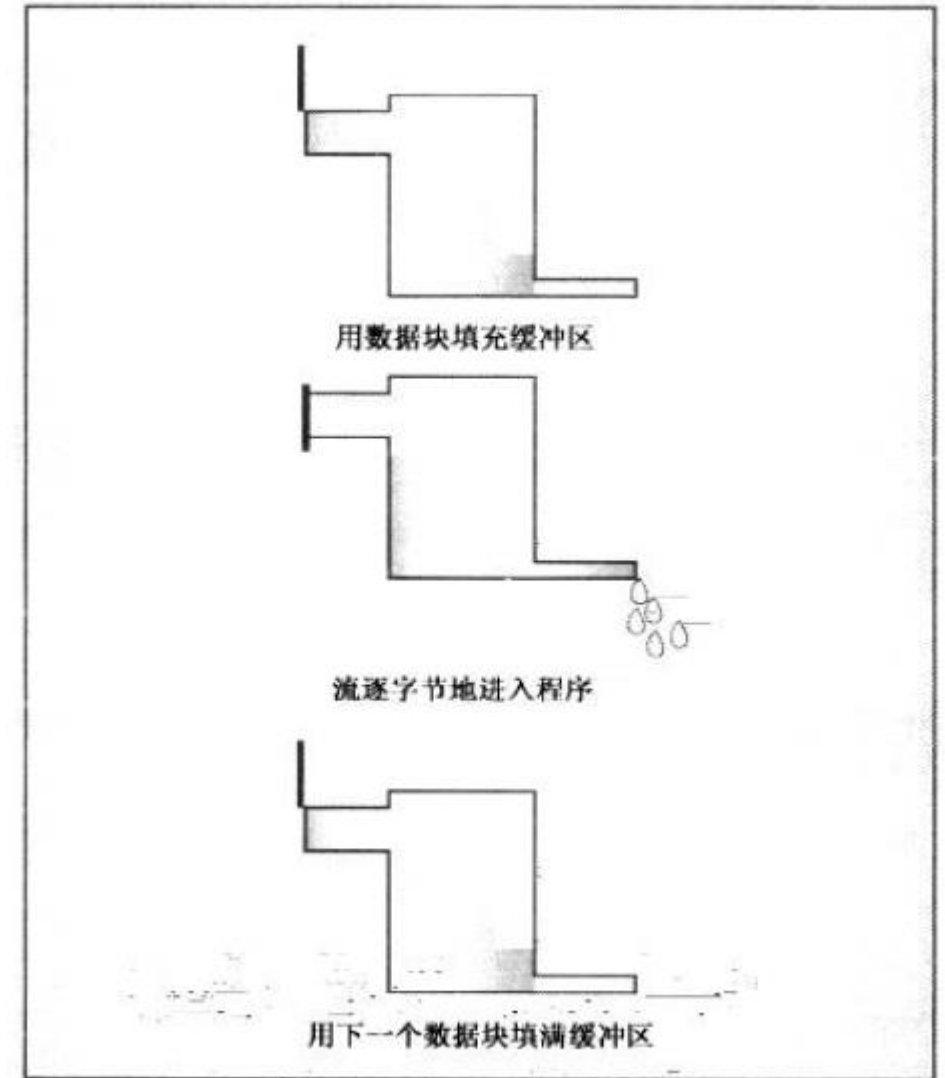


5.2.1 标准输入流

1. 流和缓冲区

- 流充当了程序和流源或流目标之间的桥梁：水滴就是字节，水滴汇成的水流就是流，从水流中取水（从流中提取输出的内容），往水流中加水（插入输入的内容）。
- **缓冲区**：用作中介的内存块，将信息从设备传输到程序或从程序传输到设备的临时存储工具，可以提高数据的读取速率。
- 输出时，程序首先填满缓冲区，然后把整块的数据传输给硬盘，并清空缓冲区，这被称为刷新缓冲区。

—缓冲区满；换行符；flush...

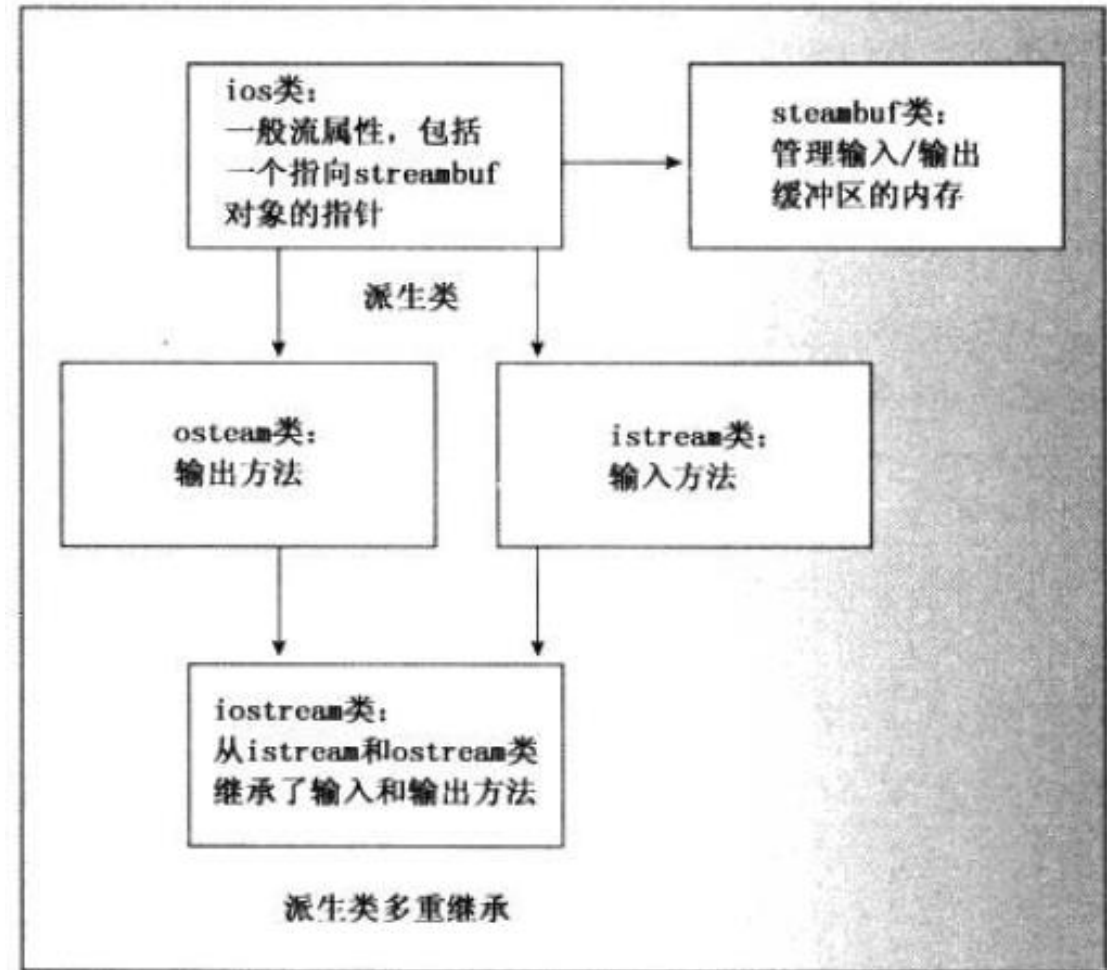


5.2.1 标准输入流

1. 流和缓冲区

- 为了管理流，C++提供了一系列头文件：
 - streambuf类为缓冲区提供了内存，并提供了填充缓冲区、访问缓冲区、刷新缓冲区和缓冲区内内存的类方法；
 - ios_base类表示流的一般特征，如是否可读取，是二进制流还是文本流等；
 - ios类基于ios_base；
 - ostream类提供了输出方法；
 - istream类提供了输入方法；
 - iostream类提供了输入和输出方法。

//后续课程讲解





5.2.1 标准输入流

1. 流和缓冲区

- C++为了能够处理需要16位国际字符集或更宽的字符类型
传统的8位char（“窄”）类型 新添加wchar_t（“宽”）类型
有专门的输入输出对象用于处理宽字符如wcin/wcout（了解即可）。
- iostream头文件会自动生成八个流对象（四个是宽类型的），重点掌握窄类型。
 - cin标准输入流：
 - cout标准输出流：
 - cerr标准错误流：
 - clog标准错误流：



5.2.1 标准输入流

1. 流和缓冲区

➤cin标准输入流:

- 输入各种数据类型，也可以通过重载使其输入用户自定义的类;

➤cout标准输出流:

- 输出各种数据类型，也可以通过重载使其输出用户自定义的类;

➤cerr标准错误流:

- 用于显式错误信息，默认情况下，这个流被关联到标准输出设备（显示器），这个流没有被缓冲，这意味着信息将直接发送给屏幕;

➤clog标准错误流:

- 用于显式错误信息，默认情况下，这个流被关联到标准输出设备（显示器），与cerr不同的是，clog有缓冲区。



5.2.1 标准输入流

2. 使用cin进行输入

- cin提取数据后，会根据数据类型是否符合要求而返回逻辑值

```
#include <iostream>
using namespace std;
int main()
{
    int a=-9;
    cin >> a;
    cout << a << " " << (cin ? 1 : 0) << endl;
    return 0;
}
```

输入	结果
10	10 1
ab	0 0
12ab	12 1
很大的数字	2147483647 0



5.2.1 标准输入流

2. 使用cin进行输入

```
#include <iostream>
using namespace std;
int main()
{
    float grade;
    cout << "enter grade:";
    while( cin >> grade ) {
        if ( grade >= 85 && grade <= 100)
            cout << "Good!" << endl;
        if ( grade < 60 )
            cout << "fail!" << endl;
    }
    return 0;
}
```

//循环一直执行，直到输入为非数字格式

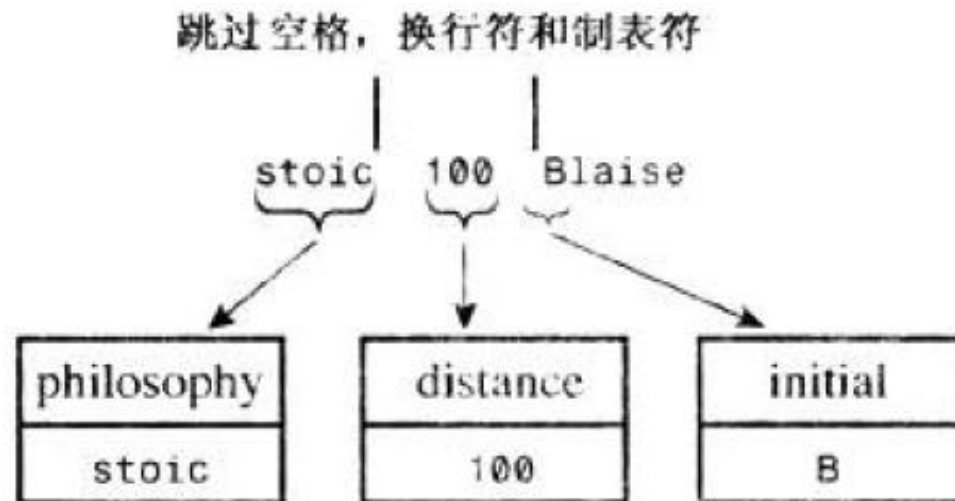


5.2.1 标准输入流

2. 使用cin进行输入

- cin>>如何检查输入：**跳过空白**（空格、换行符和制表符），直到遇到非空白字

```
char philosophy[20];  
int distance;  
char initial;  
  
cin >> philosophy >> distance >> initial;
```



//注意此例:

```
int elevation;  
cin >> elevation;
```

假设键盘输入: -123Z

则: elevation = -123

Z留在输入流中，下一个cin语句从这里开始读取。



```
//check_it.cpp -- checking for valid input
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{  
    cout << "Enter numbers:";  
    int sum = 0;  
    int input;  
    while ( cin >> input )  
    {  
        sum += input;  
    }  
    cout << "Last value entered = " << input << endl;  
    cout << "sum = " << sum << endl;  
    return 0;  
}
```

```
Enter numbers:200  
10 -50 -123Z 60  
Last value entered = 0  
sum = 37
```



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态:

//primer书P756 表17.4

成员	描述
eofbit	如果到达文件尾, 则设置为1
badbit	如果流被破坏, 则设置为1
failbit	如果输入操作未能读取预期的字符或输出操作没有写入预期的字符, 则设置为1
goodbit	另一种表示0的方法
good()	如果流可以使用 (所有的位都被清除), 则返回true
eof()	如果eofbit被设置, 则返回true
bad()	如果badbit被设置, 将返回true
fail()	如果badbit或failbit被设置, 则返回true
rdstate()	返回流状态
exceptions()	返回一个位掩码, 指出哪些标记导致异常被引发
exceptions(iostate ex)	设置哪些状态将导致clear()引发异常
clear(iostate s)	将流状态设置为s; s的默认值为0; 如果(rdstate() & exceptions()) != 0, 则引发异常 basic_ios::failure
setstate(iostate s)	调用clear(rdstate() s)。这将设置与s中设置的位对应的流状态位, 其他流状态位保持不变

一位
0或1

判断

设置



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态—设置状态

- `clear()` 方式将状态设置为它的参数:

`clear();` //默认参数0, 清除全部3个状态位 (`eofbit`, `badbit`, `failbit`)

`clear(eofbit);` //将状态设置为`eofbit`, 另外两个状态位清除

- `setstate()` 方法只影响其参数中已设置的位, 而**不会影响**其他位:

`setstate(eofbit);` //只设置`eofbit`, 不影响其他位



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态—I/O和异常

- `exceptions()` 方法用来控制异常如何被处理
- `exceptions()` 方法返回一个位字段，它包含3位，分别对应于`eofbit`、`failbit`和`badbit`，指出哪些标记导致异常被引发
- 修改流状态后，`clear()` 方法将当前的流状态与`exceptions()` 返回的值进行比较。如果在返回值中某一位被设置，而当前状态中的对应位也被设置，则`clear()` 将引发`ios_base::failure`异常，该异常包含`what()` 方法
- 如果`exceptions()` 返回`goodbit`，则不会引发任何异常



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态—I/O和异常

- exceptions() 的默认设置为goodbit, 不会引发异常

- 重载的exceptions(iostate) 函数可以控制其行为:

- `cin.exceptions(badbit);` //setting badbit causes exception

- 位运算符| 使得能够指定多位:

- `cin.exceptions(badbit | eofbit);`



```
// cinexcp.cpp -- having cin throw an exception
```

```
#include <iostream>
#include <exception>
using namespace std;
int main()
{
```

```
Enter numbers: 20 30 40 pi 6
ios_base::failbit set: istream stream error
0! the horror!
Last value entered = 40
Sum = 90
```

```
    // have failbit cause an exception to be thrown
```

```
    cin.exceptions(ios_base::failbit);
```

```
    cout << "Enter numbers: ";
```

```
    int sum = 0;
```

```
    int input;
```

```
    try {
        while (cin >> input)
        {
            sum += input;
        }
    }
```

```
    catch (ios_base::failure& bf)
```

```
    {
        cout << bf.what() << endl;
        cout << "0! the horror!\n";
    }
    cout << "Last value entered = "
        << input << endl;
    cout << "Sum = " << sum << endl;
    return 0;
}
```



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态的影响

```
...  
while ( cin >> input )  
{  
    sum += input;  
}  
...  
cout << "Now enter a new number:";  
cin >> input; //won't work
```

//流状态位被设置

方法1: 一直读取字符, 直到到达空白

```
...  
while ( cin >> input )  
{  
    sum += input;  
}  
...  
cout << "Now enter a new number:";  
cin.clear(); //reset stream state  
while (!isspace(cin.get()))  
    continue; //get rid of bad input  
cin >> input; //will work now
```

//将流状态重置, 但不匹配输入仍在
输入队列, 必须跳过



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态的影响

```
...  
while ( cin >> input )  
{  
    sum += input;  
}  
...  
cout << "Now enter a new number:";  
cin >> input; //won't work
```

//流状态位被设置

方法2: 丢弃行中的剩余部分

```
...  
while ( cin >> input )  
{  
    sum += input;  
}  
...  
cout << "Now enter a new number:";  
cin.clear(); //reset stream state  
while (cin.get() != '\n')  
    continue; //get rid rest of line  
cin >> input; //will work now
```

//将流状态重置，但不匹配输入仍在输入队列，必须跳过



5.2.1 标准输入流

2. 使用cin进行输入

- 流状态的影响

循环结束的条件:

- 1) 不恰当输入而终止; //已讨论
- 2) 到达文件尾或者由于硬件故障。

```
...
while ( cin >> input )
{ sum += input;}
...
if (cin.fail() && !cin.eof())
{ //failed because of mismatched input
  cin.clear(); //reset stream state
  while (!isspace(cin.get()))
    continue; //get rid of bad input
}
else //else fail out
{
  cout << "I cannot go on !\n";
  exit(1);
}
cout << "Now enter a new number:";
cin >> input; //will work now
```



5.2.1 标准输入流

3. cin的成员函数

- 单字符输入

(1) 成员函数 `get(char &)`

(2) 成员函数 `get(void)`

特征	<code>cin.get(ch)</code>	<code>ch=cin.get()</code>
传输输入字符的方法	赋给参数ch	将函数返回值赋给ch
字符输入时函数的返回值	指向istream对象的引用	字符编码 (int值)
达到文件尾时函数的返回值	转换为false	EOF



```
int ct = 0;
char ch;
cin >> ch;
while (ch != '\n') //Fails
{
    cout << ch;
    ct++;
    cin >> ch;
}

cout << ct << endl;
//输入 I love C++
//输出 IloveC++ (跳过空格)
//循环不终止!!!
(提取运算符跳过了换行符)
```

```
int ct = 0;
char ch;
cin.get(ch);
while (ch != '\n')
{
    cout << ch;
    ct++;
    cin.get(ch);
}

cout << ct << endl;
//输入 I love C++
//输出 I love C++10
//程序运行成功!
```

```
char c1, c2, c3;
cin.get(c1).get(c2) >> c3;
返回调用对象cin

    cin.get(c2) >> c3;
    返回调用对象cin

        cin >> c3;
        非空白字符赋给c3
```

```
char ch;
while (cin.get(ch))
{
    //process input
}
```

特征	<u>cin.get(ch)</u>
传输输入字符的方法	赋给参数ch
字符输入时函数的返回值	指向istream对象的引用
达到文件尾时函数的返回值	转换为false

```
int ct = 0;
char ch;
cin.get(ch);
while (ch != '\n')
{
    cout << ch;
    ct++;
    cin.get(ch);
}

cout << ct << endl;
//输入 I love C++
//输出 I love C++10
//程序运行成功!
```

```
int ct = 0;
char ch;
ch = cin.get();
while (ch != '\n')
{
    cout << ch;
    ct++;
    ch = cin.get();
}

cout << ct << endl;
//输入 I love C++
//输出 I love C++10
//程序运行成功!
```

```
char c1, c2, c3;
cin.get().get() >> c3;
返回int: not valid

char c1;
cin.get(c1).get(); //valid

int ch;
while ((ch=cin.get())!=EOF)
{
    //process input
}
```

特征	<u>cin.get(ch)</u>	<u>ch=cin.get()</u>
传输输入字符的方法	赋给参数ch	将函数返回值赋给ch
字符输入时函数的返回值	指向istream对象的引用	字符编码 (int值)
达到文件尾时函数的返回值	转换为false	EOF



5.2.1 标准输入流

3. cin的成员函数

- 字符串输入

(1) `cin.get(...)` //读取一行输入，直到到达换行符，将换行符**保留**在输入序列

(2) `cin.getline(...)` //读取一行输入，直到到达换行符，并**丢弃**换行符

`istream & get(char *, int, char);`

//第三个参数是分界符，遇到分界符输入停止，并**保留**在输入序列

`istream & get(char *, int);`

`istream & getline(char *, int, char);`

//第三个参数是分界符，遇到分界符输入停止，并**丢弃**分界符

`istream & getline(char *, int);`



5.2.1 标准输入流

3. cin的成员函数

- 意外字符串输入

方法	行为
<code>getline(char *, int)</code>	如果没有读取任何字符(换行符视为读取一个字符), 则设置failbit 如果读取了最大数目的字符, 且行中还有其他字符, 则设置failbit
<code>get(char *, int)</code>	如果没有读取任何字符, 则设置failbit

```
char temp[80];
```

```
while (cin.get(temp, 80)) //terminates on empty line
```

```
while (cin.getline(temp, 80) && temp[0] != '\0')
```

```
//terminates on empty line, getline仍将抽取换行符, 虽然不存储它
```



5.2.1 标准输入流

3. cin的成员函数

- istream类的其它成员函数

`cin.eof()` //文件结束

`cin.peek()` //观测下一个字符

`cin.putback(ch)` //将字符ch返回到输入流

`cin.ignore(...)` //跳过字符



```
//get_fun.cpp—using get() and getline()
#include <iostream>
using namespace std;
const int Limit = 255;
int main()
{
    char input[Limit];
    cout << "Enter a string for getline() processing:\n";
    cin.getline(input, Limit, '#'); //读取直到#, 并丢弃#
    cout << "Here is your input:\n";
    cout << input << "\nDone with phase 1\n";
}
```

//接下页

```
Enter a string for getline() processing:
Please pass
me a #3 melon!
Here is your input:
Please pass
me a
Done with phase 1
```



```
char ch;
cin.get(ch); //保留换行符在输入序列
cout << "The next input character is " << ch << endl;
if (ch != '\n')
    cin.ignore(Limit, '\n'); // discard rest of line
cout << "Enter a string for get() processing:\n";
cin.get(input, Limit, '#'); //读取直到#, 并保留#在输入序列
cout << "Here is your input:\n";
cout << input << "\nDone with phase 2\n";
cin.get(ch); //读取#
cout << "The next input character is " << ch << endl;
return 0;
```

```
}
```

```
The next input character is 3
Enter a string for get() processing:
I still
want my #3 melon!
Here is your input:
I still
want my
Done with phase 2
The next input character is #
```



```
//peeker.cpp—some istream methods
```

```
...
```

```
while (cin.get(ch)) // terminates on EOF
```

```
{    if (ch != '#') cout << ch;
```

```
    else
```

```
    {    cin.putback(ch);    // reinsert character
```

```
        break;
```

```
    }
```

```
}
```

```
if (!cin.eof())
```

```
{    cin.get(ch);
```

```
    cout << endl << ch << " is next input character.\n";
```

```
}
```

```
else
```

```
{    cout << "End of file reached.\n";
```

```
    exit(0);
```

```
}
```

```
//未完接下页
```

```
I used a #3 pencil when I should have used a #2.  
I used a  
# is next input character.
```



//接上页

```
while (cin.peek() != '#')    // look ahead
{
    cin.get(ch);
    cout << ch;
}
if (!cin.eof())
{
    cin.get(ch);
    cout << endl << ch << " is next input character.\n";
}
else
    cout << "End of file reached.\n";
...
```

```
I used a #3 pencil when I should have used a #2.
I used a
# is next input character.
3 pencil when I should have used a
# is next input character.
```




目录

- 标准输入流
- C++方式的文件操作与文件流
- C++方式的字符串流
- C方式类似字符串流的操作



5.2.2 文件操作与文件流

- C++对文件的访问:

低级I/O: 字符流方式输入/输出 (以字节为单位)

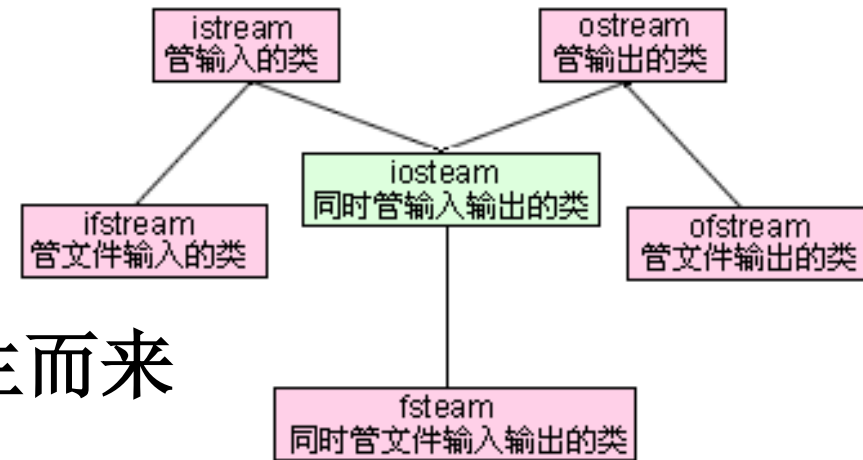
高级I/O: 转换为数据指定形式的输入/输出 (若干个字节, 有意义)

- 与磁盘文件有关的流类:

输入: ifstream类, 从istream类派生而来

输出: ofstream类, 从ostream类派生而来

输入/输出: fstream类, 从iostream类派生而来





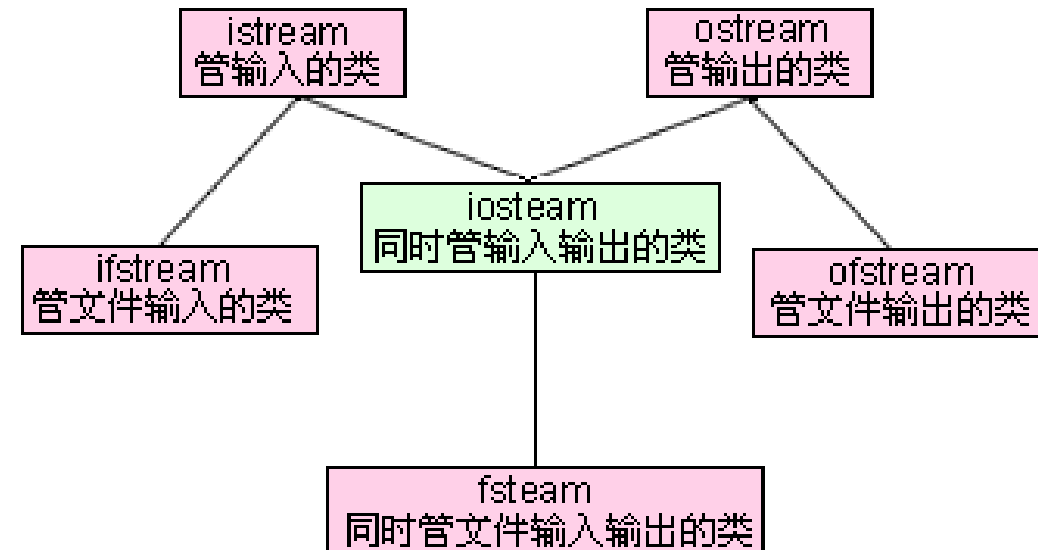
5.2.2 文件操作与文件流

- 流对象的建立:

ifstream 流对象名: 用于输入文件的操作

ofstream 流对象名: 用于输出文件的操作

fstream 流对象名: 用于输入/输出文件的操作





5.2.2 文件操作与文件流

- 文件的打开：文件流对象名.open(文件名, 打开方式);
- 加#include <fstream>

- 打开方式

常量	含义
ios::in	打开文件，以便读取
ios::out	打开文件，以便写入。若文件存在，内容全部清除
ios::ate	打开文件，并移到文件尾
ios::app	追加到文件尾
ios::trunc	如果文件存在，则截短文件；如不存在，建立新文件
ios::binary	二进制方式打开文件
ios::_Nocreate	打开已有文件，不建立新文件，如文件不存在，打开失败
ios::_Noreplace	如文件不存在则建立新文件，如文件存在则操作失败



5.2.2 文件操作与文件流

- 各个打开方式可用“位或运算符|”进行组合，但不能组合互相排斥的方式

```
ios::in|ios::out           //以输入和输出的方式打开文件，文件可读可写
ios::in|ios::binary        //以二进制方式打开一个输入文件
ios::_Nocreate|ios::_Noreplace //不允许
```

- 文件名允许带全路径，若不带路径，则表示与可执行文件同目录

```
ofstream out;
out.open("aa.dat", ios::out);
out.open("../\\C++\\aa.dat", ios::out);  .. 表示父目录
out.open(".\\C++\\aa.dat", ios::out);    . 表示当前目录
out.open("\\C++\\aa.dat", ios::out| ios::app);
out.open("c:\\C++\\aa.dat", ios::out);
```



5.2.2 文件操作与文件流

- 文件的打开与关闭

- 可在声明文件流对象时直接打开

```
ofstream out("aa.dat", ios::out);
```

- 打开方式与文件流对象之间要兼容, 否则无意义

```
ifstream in;
```

```
in.open("aa.dat", ios::out); //in对象用out打开, 无意义
```

- 每个文件被打开后, 都有一个文件指针, 初始指向开始/末尾的位置

(根据打开方式决定)



5.2.2 文件操作与文件流

- 文件的打开与关闭

- 判断文件是否成功打开的方法 //注意不同编译器有差别

<pre>if (outfile.open("f1.dat", ios::app)==0)</pre>	
<pre>if (!outfile.open("f1.dat", ios::app))</pre>	open函数返回值
<pre>if (outfile==NULL)</pre>	
<pre>if (!outfile)</pre>	流对象
<pre>if (outfile.is_open()==0)</pre>	打开后判断
<pre>if (!outfile.is_open())</pre>	is_open函数返回值

- 文件的关闭：文件流对象名.close();



```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream out;
    out.open("aa.dat", ios::out);
    if (out.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    out.close();
    return 0;
}
```

不存在时：成功(创建)

存在时：成功(覆盖)

存在并只读：失败



```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in;
    in.open("bb.dat", ios::in);
    if (!in.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    in.close();
    return 0;
}
```

不存在时：失败
存在时：成功



```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream out;
    out.open("bb.dat", ios::out|ios::_Nocreate);
    if (out.is_open()==0)
        cout << "open failed." << endl;
    else
        cout << "open success." << endl;
    out.close();
    return 0;
}
```

不存在时：失败

存在时：成功(覆盖)

存在并只读：失败



5.2.2 文件操作与文件流

- 对ASCII文件的操作

基本方法：将文件流对象名当作cin/cout对象，用>>和<<进行格式化的输入和输出

>>和<<使用时的注意事项与cin、cout时相同

cin >> 变量 => infile >> 变量

cout << 变量 => outfile << 变量

成员函数的使用方法与前面相同

cout.put('A') => outfile.put('A')



```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main()
{   int a[10];
    ofstream outfile("f1.dat", ios::out);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl; exit(1);
    }
    cout << "enter 10 integer numbers:" << endl;
    for(int i=0; i<10; i++) {
        cin >> a[i];
        outfile << a[i] << " "; //int型输出到文件
    }
    outfile.close(); return 0;
}
```

打开方式若修改方式为:
`ios::out` | `ios::app`

//运行两次（观察文件内容）:

1 3 5 2 4 6 10 8 7 9

1 3 5 2 4 6 10 8 7 9 1 3 5 2 4 6 10 8 7 9

//运行两次（观察文件内容）:

1 3 5 2 4 6 10 8 7 9

1 3 5 2 4 6 10 8 7 9



```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main()
{
    int a[10], max, i, order;
    ifstream infile("f1.dat", ios::in);
    if (!infile.is_open()) {
        cerr << "open error!" << endl; exit(1);
    }
    for(i=0; i<10; i++) {
        infile >> a[i]; //从文件中读10个int放入a数组
        cout << a[i] << " "; //int型输出到屏幕
    }
    //找最大值...
    infile.close(); return 0;
}
```



//打开文件并将内容输出到屏幕上

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main()
```

```
{ ifstream in;
```

```
  char ch;
```

```
  in.open("d:\\test\\data.txt", ios::in);
```

```
  if (in.is_open()==0) {
```

```
    cout << "文件打开失败\n";           //cerr
```

```
    return -1;                           //exit(1)
```

```
  }
```

```
  while(!in.eof()) {
```

```
    ch = in.get();           //in.get(ch);
```

```
    putchar(ch);           //cout.put(ch);
```

```
  }
```

```
  in.close();return 0;
```

```
}
```

```
while((ch=in.get())!=EOF)
    cout.put(ch);
```



//复制文件内容至另一个文件

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main()
```

```
{    ifstream in;    ofstream out;    char ch;
```

```
    in.open("d:\\test\\data.txt", ios::in);
```

```
    if (!in.is_open()) {
```

```
        cout << "无法打开源文件" << endl;    return -1;
```

```
}
```

```
    out.open("d:\\demo\\data2.txt", ios::out);
```

```
    if (!out.is_open()) {
```

```
        cout << "无法打开目标文件" << endl;    in.close();    return -1;
```

```
}
```

```
    while(in.get(ch))    //从输入文件中一次读取
```

```
        out.put(ch);    //写入输出文件中
```

```
    in.close();    out.close();    return 0;
```

```
}
```




5.2.2 文件操作与文件流

- 对二进制文件的操作

- 用ASCII文件的字符方式进行操作(按字节读写)

- 用read/write进行操作

- 文件流对象名.read(内存空间首指针, 长度);

- 从文件中读长度个字节, 放入从首指针开始的空间中

- 文件流对象名.write(内存空间首指针, 长度);

- 将从首指针开始的连续长度个字节写入文件中

//ASCII方式

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
struct student {
    char name[20];int num;int age;char sex;
};
int main()
{
    student stud[3]={"Li", 1001, 18, 'f', "Fun", 1002, 19, 'm', "Wang", 1004, 17, 'f'};
    ofstream outfile("stud.dat", ios::out);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;  exit(-1);
    }
    for(int i=0; i<3; i++)
        outfile<<stud[i].name<<stud[i].num<<stud[i].age<<stud[i].sex<<endl;
    outfile.close();
    return 0;
}
```

生成的stud.dat文件共36字节:

Li100118f

Fun100219m

Wang100417f

//二进制文件方式

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
struct student {
    char name[20];int num;int age;char sex;
};
```

```
int main()
```

```
{  student stud[3]={"Li",1001,18,'f', "Fun",1002,19,'m', "Wang",1004,17,'f'};
  ofstream outfile("stud.dat", ios::binary);
```

```
  if (!outfile.is_open()) {
      cerr << "open error!" << endl;
      exit(-1);
  }
```

```
  for(int i=0; i<3; i++)
```

```
      outfile.write((char *)&stud[i], sizeof(stud[i]));
```

```
  outfile.close();  return 0;
```

```
}
```

生成的stud.dat文件共96字节:

4C	69	00	??	??	??	??	??	} 前32 字节
??	??	??	??	??	??	??	??	
??	??	??	??	E9	03	00	00	
12	00	00	00	66	??	??	??	
...								

4C6900	=>	"Li" (含尾零)
E9030000	=>	0x0000003E9 => 1001
12000000	=>	0x000000012 => 18
66	=>	'f' (后3个是填充字节)

//二进制文件方式

```
int main()
{
    student stud[3];
    int i;
    ifstream infile("stud.dat", ios::binary);
    if (!infile.is_open()) {
        cout << "文件打开失败" << endl; return -1;
    }
    for(i=0; i<3; i++) //一次读入一个数组元素（32字节）
        infile.read((char *)&stud[i], sizeof(stud[i]));
    infile.close();
    for(i=0; i<3; i++) {
        cout << "No." << i+1 << endl;
        cout << "name:" << stud[i].name << endl;
        cout << "num:" << stud[i].num << endl;
        cout << "age:" << stud[i].age << endl;
        cout << "sex:" << stud[i].sex << endl << endl; //多空一行
    }
    return 0;
}
```

//stud.dat的内容是前例
生成的二进制文件

4C	69	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	E9	03	00	00	12	00	00	00	66	CC	CC	CC
46	61	6E	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	EA	03	00	00	13	00	00	00	6D	CC	CC	CC
57	61	6E	67	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	EC	03	00	00	11	00	00	00	66	CC	CC	CC

No. 1
name:Li
num:1001
age:18
sex:f

No. 2
name:Fan
num:1002
age:19
sex:m

No. 3
name:Wang
num:1004
age:17
sex:f

```

infile.read((char *)&stud[i], sizeof(stud[i])),
infile.close();
for(i=0; i<3; i++) {
    cout << "No." << i+1 << endl;
    cout << "name:" << stud[i].name << endl;
    cout << "num:" << stud[i].num << endl;
    cout << "age:" << stud[i].age << endl;
    cout << "sex:" << stud[i].sex << endl << endl; //多空
}
return 0;
}

```



5.2.2 文件操作与文件流

- 对二进制文件的操作

- 与文件指针有关的流成员函数

- 适用于输入文件的:

- `gcount()`: 返回最后一次读入的字节数

- `tellg()`: 返回输入文件的当前指针

- `seekg(位移量, 参照位置)`: 移动输入文件指针

- 适用于输出文件的:

- `tellp()`: 返回输出文件的当前指针

- `seekp(位移量, 参照位置)`: 移动输出文件指针



5.2.2 文件操作与文件流

- 对二进制文件的操作

- 与文件指针有关的流成员函数

- 参照位置:

- `ios::beg`: 从文件头部移动, 位移量必须为正

- `ios::cur`: 从当前指针处移动, 位移量可正可负

- `ios::end`: 从文件尾部移动, 位移量必须为负

- 随机访问二进制数据文件

在文件的读写过程中, 可前后移动文件指针, 达到按需读写的目的

//随机访问二进制文件

```
int main()
{
    student stud[3]={ "Li", 1001, 18, 'f',
                      "Fan", 1002, 19, 'm', "Wang", 1004, 17, 'f' };
    ofstream outfile("stud.dat", ios::binary);
    if (!outfile.is_open()) {
        cout << "文件打开失败" << endl;
        return -1;
    }
    for(int i=0; i<3; i++) { //一次写入一个数组元素
        outfile.write((char *)&stud[i], sizeof(stud[i])); //从当前位置写32字节
        cout << outfile.tellp() << endl; //写完32字节后，文件指针为32
        outfile.seekp(0, ios::beg); //文件指针移回0
        cout << outfile.tellp() << endl; //文件指针为0
    }
    outfile.close();
    return 0;
}
```

```
#include <iostream>
#include <fstream>
using namespace std;
struct student {
    char name[20];
    int num;
    int age;
    char sex;
}; //含填充共32字节
```

Microsoft Visual Studio 调试控制台

```
32
0
32
0
32
0
```




5.2.2 文件操作与文件流

- 关于二进制访问的几个注意事项
 - read参数中的长度是最大读取长度，不是实际读取长度，因此read后要用gcount()返回真实读到的字节数
 - 如果读写方式打开(`ios::in | ios::out`)，则只有一个文件指针，`seekg()`和`seekp()`是同步的，`tellg()`和`tellp()`也是同步的
 - 在文件的操作超出正常范围后(例：`read()`已到EOF、`seekg()/seekp()`超文件首尾范围等)，再次对文件进行`seekg()/seekp()/tellg()/tellp()`等操作都可能会返回与期望不同的值，建议在文件操作过程中勤用`good()/fail()/eof()/clear()`等函数



目录

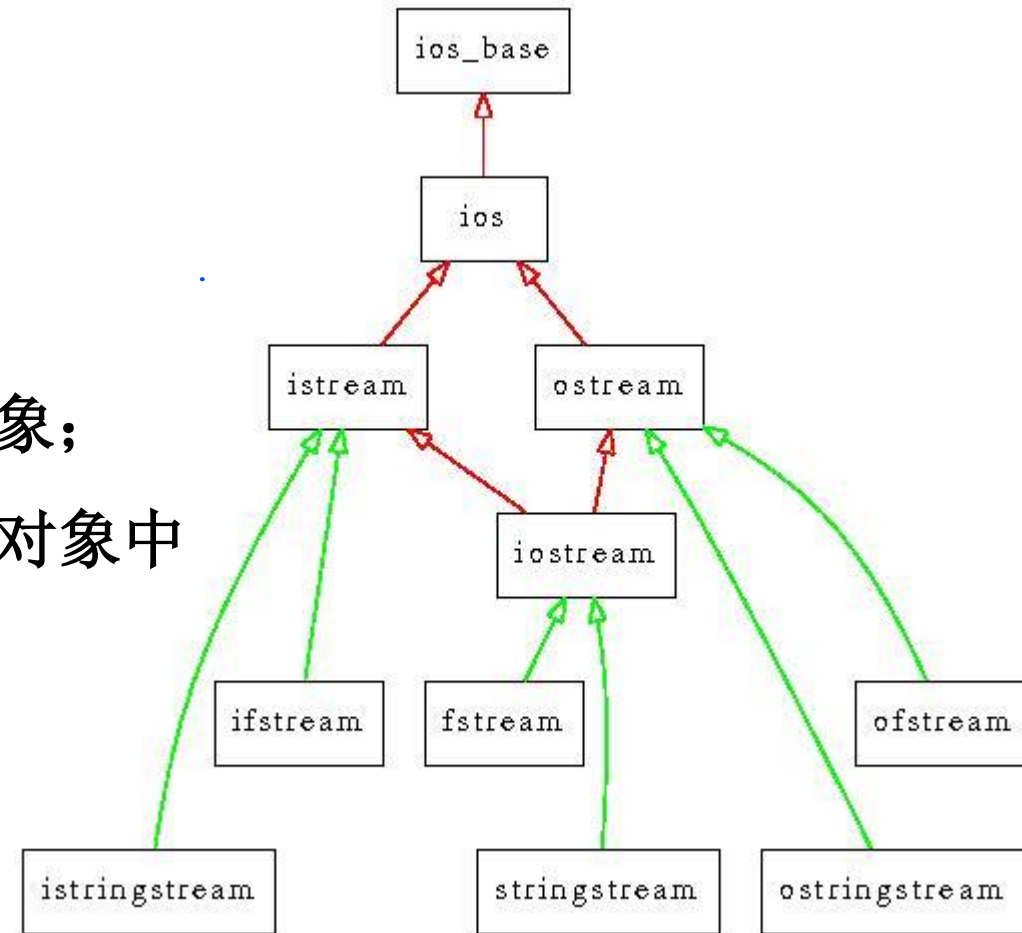
- 标准输入流
- C++方式的文件操作与文件流
- C++方式的字符串流
- C方式类似字符串流的操作

5.2.3 C++方式的字符串流

1. 基本概念

- `iostream`族：程序与终端之间的I/O；
- `fstream`族：程序与文件之间的I/O；
- **`sstream`族**：程序与string对象之间的I/O。
 - `ostream`方法将格式化信息写入到string对象；
 - `istream`方法（如`getline()`）来读取string对象中的信息。

➡ 内核格式化（incore formatting）





5.2.3 C++方式的字符串流

1. 基本概念

- 以内存中的string类型变量为输入/输出对象
 - 可以存放各种类型的数据
 - 与标准输入输出流相同，进行文本和二进制的相互转换

向string存数据 cout: 二进制 => ASCII

从string取数据 cin : ASCII => 二进制

推论：可用于不同数据类型的转换

- 不是文件，不需要打开和关闭



5.2.3 C++方式的字符串流

2. 字符串流对象的使用-写入

- 加 `#include <sstream>`
- 创建 `ostringstream` 类对象 (输出流对象)

```
ostringstream ostr;  
double price = 380.0;  
const char* ps = " for a copy of the ISO/EIC C++ standard!";  
ostr.precision(2);  
ostr << fixed;  
ostr << "Play only CHF" << price << ps << endl;
```

- `ostringstream` 类的成员函数 `str()`，返回一个被初始化为缓冲区内容的字符串对象
`string msg = ostr.str();`

Play only CHF380.00 for a copy of the ISO/EIC C++ standard!

//例1: 输出流对象的基本使用: 将多个格式化内容拼在一起, 集中输出

```
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
    ostringstream out;
    out << "Hello" << 10 << 11.2 << endl;
    string s1 = out.str();
    cout << s1 << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台
Hello1011.2

成员函数str()的作用:
将ostringstream的内容转换为string格式

```
ostringstream out;
out << "Hello" << 10 << 11.2 << endl;
cout << out.str() << endl; //跟上例等价
```

Microsoft Visual Studio 调试控制台
Hello1011.2

//例2: 内核格式化 strout.cpp - incore formatting (output)

```
#include <iostream>
```

```
#include <sstream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main()
```

```
{   ostringstream outstr;
```

```
    string hdisk;
```

```
    cout << "What's the name of your hard disk? ";
```

```
    getline(cin, hdisk);           // cin -> hdisk
```

```
    int cap;
```

```
    cout << "What's its capacity in GB? ";
```

```
    cin >> cap;
```

```
    // write formatted information to string stream    // (格式化信息 -> string对象)
```

```
    outstr << "The hard disk " << hdisk << " has a capacity of " << cap << " gigabytes.\n";
```

```
    string result = outstr.str(); //save result
```

```
    cout << result;           //show contents
```

```
    return 0;
```

```
}
```

读取string对象中的格式化信息或将格式化信息写入string对象中被称为内核格式化(incore formatting)

```
What's the name of your hard disk? YChen
```

```
What's its capacity in GB? 20
```

```
The hard disk YChen has a capacity of 20 gigabytes.
```



5.2.3 C++方式的字符串流

3. 字符串流对象的使用-读取

- 加 `#include <sstream>`
- 创建 `istringstream` 类对象 (输入流对象), 并使用 `string` 对象初始化

```
istringstream instr(facts);    //use facts to initialize stream  
  
int n;  
  
int sum = 0;  
while (instr >> n)  
    sum += n;
```




//例1: 内核格式化 strin.cpp - formatted reading from a char array

```
#include <iostream>
```

```
#include <sstream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    string lit = "It was a dark and stormy day, and "  
                " the full moon glowed brilliantly. ";
```

```
    istringstream instr(lit); //use buf for input lit->instr
```

```
    string word;
```

```
    while (instr >> word) //read a word a time
```

```
        cout << word << endl;
```

```
    return 0;
```

```
}
```

It
was
a
dark
and
stormy
day,
and
the
full
moon
glowed
brilliantly.



//例2: 输入流对象-流状态

```
int main()
{
    istringstream in("Hello 10 11.2");
    cout << in.str() << endl;

    char s[10];
    short i;
    float f;
    in >> s >> i >> f;
    cout << s << '-' << i << '-' << f << endl;

    cout << in.good() << ' ' << in.fail() << endl; //good和fail均为0
    //如果现有内容全部读完, goodbit会置0, 此时good和fail均为0, 表示EOF
    cout << in.str() << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
Hello 10 11.2
Hello-10-11.2
0 0
Hello 10 11.2
```

//例3: 输入流对象-流状态

```
int main()
{
    istream in("Hello 10 11.2 xyz");
    cout << in.str() << endl;

    char s[10];
    short i;
    float f;
    in >> s >> i >> f; //读之后内容仍在, 可用str()查看
    cout << s << '-' << i << '-' << f << endl;

    cout << in.good() << ' ' << in.fail() << endl; //good为1, fail为0
    //现有内容尚未读完, goodbit为1, failbit为0
    cout << in.str() << endl; //可用str()打印现有内容
    return 0;
}
```



Microsoft Visual Studio 调试控制台

```
Hello 10 11.2 xyz
Hello-10-11.2
1 0
Hello 10 11.2 xyz
```



//例4: 输入流对象-重复读取

```
int main()
{
    istringstream in("Hello 10 11.2");
    cout << in.str() << endl;

    char s1[10], s2[10]="xyz";
    short i1, i2=123;
    float f1, f2=0.456F;
    in >> s1 >> i1 >> f1;
    cout << s1 << '-' << i1 << '-' << f1 << endl;
    cout << s2 << '-' << i2 << '-' << f2 << endl;

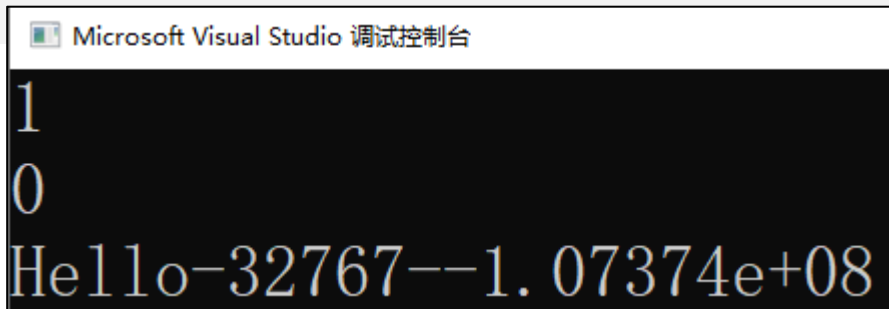
    in.clear();
    in.seekg(0, ios::beg);
    in >> s2 >> i2 >> f2; // istringstream的内容可重复读取
    cout << s2 << '-' << i2 << '-' << f2 << endl;
    return 0;
}
```





//例5: 输入流对象-数据超范围

```
int main()
{
    istream in("Hello 70000 11.2");
    char s[10];
    short i;
    float f;
    in >> s;
    cout << in.good() << endl;
    in >> i;
    cout << in.good() << endl;
    //in.clear();
    in >> f;
    cout << s << '-' << i << '-' << f << endl;
    return 0;
}
```



Microsoft Visual Studio 调试控制台

```
1
0
Hello-32767--1.07374e+08
```

加入in.clear();



Microsoft Visual Studio 调试控制台

```
1
0
Hello-32767-11.2
```

如果数据超范围, 后续会错, 可clear恢复



//例6: 输入流对象-使用带参str()再次赋新内容

```
int main()
{
    istream in("Hello 10 11.2");
    char s[10];
    short i;
    float f;
    in >> s >> i >> f;
    cout << s << '-' << i << '-' << f << endl;
    cout << in.good() << endl;
    in.str("tongji 123 0.123");
    in >> s >> i >> f;
    cout << s << '=' << i << '=' << f << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台
Hello-10-11.2
0
=10=11.2

"=10=11.2"是因为读s出错, s置空, i/f未变



//例7: 输入流对象-使用带参str()再次赋新内容

```
int main()
{
    istream in("Hello 10 11.2 12345");
    char s[10];
    short i;
    float f;
    in >> s >> i >> f;
    cout << s << '-' << i << '-' << f << endl;
    cout << in.good() << endl;
    in.str("tongji 123 0.123");
    in >> s >> i >> f;
    cout << s << '=' << i << '=' << f << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
Hello-10-11.2
1
tongji=123=0.123
```

可用带参str()再次赋新内容, 但注意goodbit



5.2.3 C++方式的字符串流

4. 字符串流对象的使用-读取与写入

- 加 `#include <sstream>`
- 创建stringstream类对象（输入/输出流对象）
- 使用总结：
 - 存储形式为string，不需要用户考虑空间
 - 使用方式同iostream/fstream基本相似（部分细节可能不同）
 - 如果结果与预期不同，多判断good()/fail()



//例1: 字符串流对象

```
int main()
{
    stringstream ss("Hello 10 11.2");
    char s[10];
    short i;
    float f;
    ss >> s >> i >> f;
    cout << s << '-' << i << '-' << f << endl;
    cout << ss.tellg() << endl;

    ss << "xyz 123 0.456";
    cout << ss.str() << endl;
    cout << ss.tellg() << endl;

    return 0;
}
```

stringstream可读可写，但注意goodbit

//左侧相应位置加入下述程序:
ss.clear();

```
ss.seekg(0, ios::beg);
ss >> s >> i >> f;
cout << s << '=' << i << '=' << f << endl;
```

Microsoft Visual Studio 调试控制台

```
Hello-10-11.2
-1
Hello 10 11.2
-1
```

Microsoft Visual Studio 调试控制台

```
Hello-10-11.2
-1
xyz 123 0.456
13
xyz=123=0.456
```

//例2: 从stringstream中读入10个数并排序

```
#include <iostream>
```

```
#include <sstream>
```

```
using namespace std;
```

```
#define N 10
```

```
int main()
```

```
{
```

```
    stringstream ss("12 34 65 -23 -32 33 61 99 321 32");
```

```
    int a[N], i, j, t;
```

```
    for (i = 0; i < N; i++)
```

```
        ss >> a[i]; //ss中的内容逐个读入int a[10]中
```

```
    cout << "array a:";
```

```
    for (i = 0; i < N; i++) //输出int a[10]的内容
```

```
        cout << a[i] << " ";
```

```
    cout << endl;
```

```
int main()
```

```
{
```

```
...
```

```
//冒泡排序
```

```
for (i = 0; i < N-1; i++)
```

```
    for (j = 0; j < N - 1 - i; j++)
```

```
        if (a[j] > a[j + 1]) {
```

```
            t = a[j];
```

```
            a[j] = a[j + 1];
```

```
            a[j + 1] = t;
```

```
        }
```

```
//输出到ss中 (ss刚才用做了输入流)
```

```
ss.clear();
```

```
ss.seekg(0, ios::beg);
```

```
for (i = 0; i < N; i++)
```

```
    ss << a[i] << " ";
```

```
ss << endl;
```

```
cout << "array a after sort:" << ss.str() << endl;
```

```
}
```

Microsoft Visual Studio 调试控制台

```
array a:12 34 65 -23 -32 33 61 99 321 32
```

```
array a after sort:-32 -23 12 32 33 34 61 65 99 321
```



5.2.3 C++方式的字符串流

5. 字符串流对象的应用

- 实现不同数据类型的转换
 - 字符串转double
 - double转字符串
 - 多种类型转字符串（重载方式）
- 实现字符串串接



- 应用1：实现不同数据类型的转换

//字符串转double

```
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
    istringstream in("123.456");
    double d;
    in >> d;
    cout << d << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

123.456

//double转字符串

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
    ostringstream out;
    double d = 123.456;
    char str[10];
    out << d;
    strcpy(str, out.str().c_str());
    cout << str << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

123.456



- 应用1：实现不同数据类型的转换

//多种类型转字符串(重载方式)

```
string tj_to_string(const double d)
{
    ostringstream out;
    out << d;
    return out.str();
}

string tj_to_string(const int i)
{
    ostringstream out;
    out << i;
    return out.str();
}

string tj_to_string(const char ch)
{
    ostringstream out;
    out << ch;
    return out.str();
}
```

```
int main()
{
    string s1 = tj_to_string(123.456);
    string s2 = tj_to_string(12345);
    string s3 = tj_to_string('A');
    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
}
```

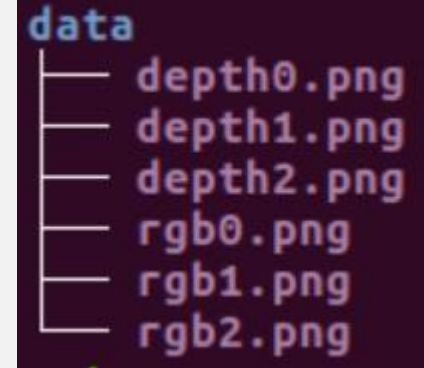


• 应用2：实现字符串串接

```
//stringstream.cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace std;
using namespace cv;
void readFrame(int index);
int main()
{
    for (int i = 0; i < 3; i++)
    {
        readFrame(i);
    }
    return 0;
}
```

```
void readFrame(int index)
{
    string dir = "./data/";
    string rgb = "rgb";
    string dep = "depth";
    string Ext = ".png";

    stringstream ss;
    ss << dir << rgb << index << Ext;    // ./data/rgb1.png
    string filename;
    ss >> filename;
    Mat color = imread(filename, 1);        //rgb
    imshow(filename, color);
    ss.clear();
    ss << dir << dep << index << Ext;    // ./data/depth1.png
    filename.clear();
    ss >> filename;
    Mat depth = imread(filename, 1);        //depth
    imshow(filename, depth);
    waitKey();
}
```



将字符串类型 (rgb) 和整数类型 (index) 拼起来, index不断更新, 从而读取不同下标的文件



目录

- 标准输入流
- C++方式的文件操作与文件流
- C++方式的字符串流
- C方式类似字符串流的操作



5.2.4 C方式类似字符串流的操作

- 向字符串输出格式化的数据

`int sprintf(字符数组, "格式串", 输出表列);`

- ① 返回值是输出字符的个数

指不同类型数据按格式串的要求转换为文本方式后字符的个数

- ② 与printf相同，完成二进制向ASCII的转换

- ③ VS下需加 `#define _CRT_SECURE_NO_WARNINGS`



5.2.4 C方式类似字符串流的操作

- 向字符串输出格式化的数据

int sprintf(字符数组, "格式串", 输出表列);

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{    char c[80];
```

```
    int ret;
```

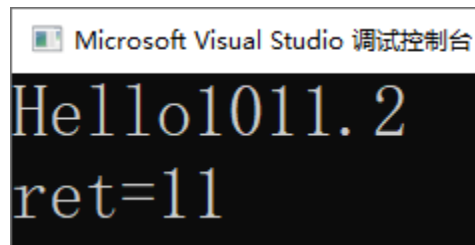
```
    ret = sprintf(c, "Hello%d%.1f", 10, 11.2);
```

```
    printf("%s\n", c);
```

```
    printf("ret=%d\n", ret);
```

```
    return 0;
```

```
}
```



//C++方式

```
int main()
```

```
{    ostream out;
```

```
    out << "Hello" << 10 << 11.2 << endl;
```

```
    cout << out.str() << endl;
```

```
    return 0;
```

```
}
```

```
//多次向字符数组输出
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
using namespace std;
struct student {
    int num; char name[20]; float score;
};
int main()
{
    student stud[3]={1001, "Li", 78, 1002, "Wang", 89.5, 1004, "Fun", 90} ;
    char c[50], *s = c;
    for (int i=0; i<3; i++)
        s+=sprintf(s, "%d %s %.1f", stud[i].num , stud[i].name, stud[i].score);
    cout << "array c:" << c << endl;
    return 0;
}
```

```
array c:1001 Li 78.01002 Wang 89.51004 Fun 90.0
```



5.2.4 C方式类似字符串流的操作

- 从字符串中输入格式化的数据

`int sscanf(字符数组, "格式串", 输入表列);`

- ① 返回值是正确读入的输入数据的个数
- ② 与scanf相同，完成ASCII向二进制的转换
- ③ VS下需加 `#define _CRT_SECURE_NO_WARNINGS`



5.2.4 C方式类似字符串流的操作

- 从字符串中输入格式化的数据

int sscanf(字符数组, "格式串", 输入表列);

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{ char c[80] = "Hello 10 11.2";
```

```
char s[10];
```

```
int i, ret;
```

```
float f;
```

```
ret = sscanf(c, "%s %d %f", s, &i, &f);
```

```
printf("%s-%d-%.1f\n", s, i, f);
```

```
printf("ret=%d\n", ret);
```

```
return 0;
```

```
}
```

```
//C++方式
```

```
int main()
```

```
{ istringstream in("Hello 10 11.2");
```

```
char s[10];
```

```
int i; float f;
```

```
in >> s >> i >> f;
```

```
cout << s << '-' << i << '-' << f << endl;
```

```
return 0;
```

```
}
```

Microsoft Visual Studio 调试控制台

```
Hello-10-11.2
ret=3
```

//例：从字符串中读入10个数并排序

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#define N 10
```

```
int main()
```

```
{    char ss[80] = "12 34 65 -23 -32 33 61 99 321 32", *s = ss;
```

```
    int a[N], i, j, t;
```

```
    //ss中的内容逐个读入int a[10]中
```

```
    for (i = 0; i < N; i++) {
```

```
        sscanf(s, "%d", &a[i]);
```

```
        s = strchr(s, ' '); //在串s中查找给定字符' '的第一个匹配之处
```

```
        s++; //指向空格后的字符
```

```
    }
```

```
    printf("array a:");
```

```
    for (i = 0; i < N; i++) //输出int a[10]的内容
```

```
        printf("%d ", a[i]);
```

```
    printf("\n");
```

```
int main()
```

```
{ ...
```

```
//进行排序
```

```
for (i = 0; i < N - 1; i++)
```

```
    for (j = 0; j < N - 1 - i; j++)
```

```
        if (a[j] > a[j + 1]) {
```

```
            t = a[j];
```

```
            a[j] = a[j + 1];
```

```
            a[j + 1] = t;
```

```
        }
```

```
//输出到ss中 (ss刚才用做了输入流)
```

```
s = ss; //重新指向ss[0]
```

```
for (i = 0; i < N; i++)
```

```
    s+=sprintf(s, "%d ", a[i]);
```

```
s+=sprintf(s, "\n");
```

```
printf("array a after sort:%s\n", ss);
```

```
return 0;
```

```
}
```

Microsoft Visual Studio 调试控制台

```
array a:12 34 65 -23 -32 33 61 99 321 32
```

```
array a after sort:-32 -23 12 32 33 34 61 65 99 321
```



总结

- 标准输入流
- C++方式的文件操作与文件流（熟练）
- C++方式的字符串流（熟练）
- C方式类似字符串流的操作（了解）