



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 1. 流的基本概念

流的含义：流是来自设备或传给设备的一个数据流，由一系列字节组成，按顺序排列（字节流）

★ C/C++的原生标准中没有定义输入/输出的基本语句

★ C语言用printf/scanf等函数来实现输入和输出，通过#include <stdio.h>来调用

★ C++通过cin和cout的流对象来实现，通过#include <iostream>来调用

cout：输出流对象    <<：流插入运算符

cin：输入流对象    >>：流提取运算符



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 2. 输出流的基本操作

格式: `cout << 表达式1 << 表达式2 << ... << 表达式n;`

- ★ 插入的数据存储在缓冲区中, 不是立即输出, 要等到缓冲区满 (不同系统大小不同) 或者碰到换行符 ("`\n`" / `endl`) 或者强制立即输出 (`flush`) 才一齐输出
- ★ 默认的输出设备是显示器 (可更改, 称输出重定向)
- ★ 一个 `cout` 语句可写为若干行, 或者若干语句
- ★ 一个 `cout` 的输出可以是一行, 也可以是多行, 多个 `cout` 的输出也可以是一行
- ★ 一个插入运算符只能输出一个值
- ★ 系统会自动判断输出数据的格式



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 3. 输入流的基本操作

格式: `cin >> 变量1 >>变量2 >> ... >>变量n;`

- ★ 键盘输入的数据存储在缓冲区中, 不是立即被提取, 要等到缓冲区满(不同系统大小不同)或碰到回车符才进行提取
- ★ 默认的输入设备是键盘(可更改, 称输入重定向)
- ★ 一行输入内容可分为若干行, 或者若干语句
- ★ 一个提取运算符只能输入一个值
- ★ 提取运算符后必须跟变量名, 不能是常量/表达式等
- ★ 输入终止条件为回车、空格、非法输入
- ★ 系统会自动判断输入数据, 若超过变量范围则错误
- ★ 字符型变量只能输入图形字符(33-126), 不能以转义符方式输入(单双引号、转义符全部当作单字符)
- ★ 浮点数输入时, 可以是十进制数或指数形式, 只取有效位数(4舍5入)
- ★ `cin`不能跟`endl`, 否则编译错



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

#### 3. 4. C++的输入与输出

##### 3. 4. 4. 在输入输出流中使用格式化控制符

C++缺省输入/输出格式是默认格式，为满足一些特殊要求，需要对数据进行格式化



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 5. 字符的输入和输出

##### 3. 4. 5. 1. 字符输出函数putchar

形式: putchar(字符变量/常量)

功能: 输出一个字符

```
char a='A';  
putchar(a);  
putchar('A');  
putchar('\x41');  
putchar('\101');
```

★ 加#include <cstdio>或#include <stdio.h>

★ 返回值是int型, 是输出字符的ASCII码, 可赋值给字符型/整型变量



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 5. 字符的输入和输出

##### 3. 4. 5. 2. 字符输入函数getchar

形式: `getchar()`

功能: 输入一个字符 (给指定的变量)

★ 加`#include <cstdio>`或`#include <stdio.h>`

★ 返回值是`int`型, 是输入字符的ASCII码, 可赋值给字符型/整型变量

★ 输入时有回显, 输入后需按回车结束输入 (若直接按回车则得到回车的ASCII码)

★ 可以输入空格, 回车等`cin`无法处理的非图形字符, 但仍不能处理转义符

★ `cin/getchar` 等每次仅从输入缓冲区中取需要的字节, 多余的字节仍保留在输入缓冲区中供下次读取



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 5. 字符的输入和输出

##### 3. 4. 5. 3. 字符输入函数\_getch与\_getche

##### ★ 几个字符输入函数的差别

getchar : 有回显, 不立即生效, 需要回车键

\_getche : 有回显, 不需要回车键

\_getch : 无回显, 不需要回车键

##### ★ 在Dev C++中

getch() ⇔ \_getch()

getche() ⇔ \_getche()



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 1. 有关输入输出基本概念的回顾

#### § 3. 结构化程序设计基础

##### 3. 4. C++的输入与输出

##### 3. 4. 6. C语言的格式化输入与输出函数

★ 格式化输出: `printf`

★ 格式化输入: `scanf`

下发相关资料并参考其它书籍，结合作业进行自学

#### 要求:

- 1、熟练使用C++的`cin/cout`
- 2、能看懂C的`printf/scanf`
- 3、除非明确规定，C++作业不允许`printf/scanf`





## § 17. 输入输出流

### 1. C++的输入与输出

#### 1. 2. 输入输出的基本概念

输入输出的种类:

系统设备: 标准输入设备: 键盘

(标准I/O) 标准输出设备: 显示器

其它设备: 鼠标、打印机、扫描仪等

外存文件: 从文件中得到输入

(文件I/O) 输出到文件中

内存空间: 输入/输出到一个字符数组/string中

(串I/O)

★ 操作系统将所有系统设备都统一当作文件进行处理

C++输入/输出的特点:

★ 与C兼容, 支持printf/scanf

★ 对数据类型进行严格的检查, 是类型安全的I/O操作

★ 具有良好的可扩展性, 可通过重载操作符的方式输入/输出自定义数据类型

C++的输入/输出流:

★ 采用字符流方式, 缓冲区满或遇到endl才输入/输出

★ cin, cout不是C++的语句, 也不是函数, 是类的对象 >> 和 << 的本质是左移和右移运算符, 被重载为输入和输出运算符



## § 17. 输入输出流

### 1. C++的输入与输出

#### 1.2. 输入输出的基本概念

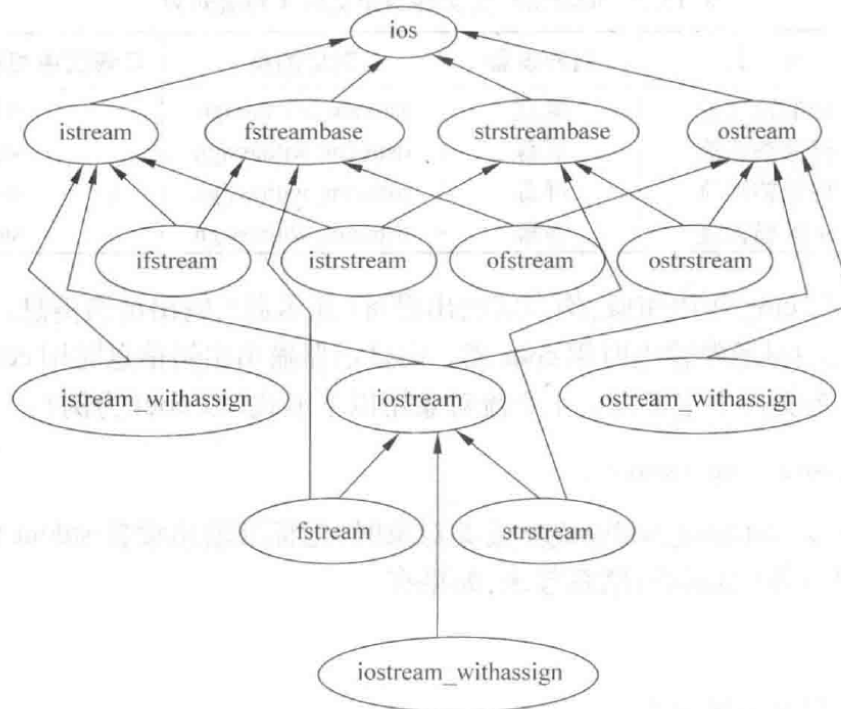
C++中与输入/输出相关的类及对象：

iostream类库中有关的类

与iostream类库有关的头文件

在iostream头文件中定义的流对象

在iostream头文件中重载运算符



C++输入输出流的整体设计结构

建议读此部分源码



## § 17. 输入输出流

### 2. 标准输出流

#### 2.1. cout, cerr和clog流

cout: 向控制台进行输出, 缺省是显示器

cerr: 向标准出错设备进行输出, 缺省是显示器(直接输出, 不必等待缓冲区满或回车)

clog: 向标准出错设备进行输出, 缺省是显示器(放在缓冲区中, 等待缓冲区满或回车才输出)

★ 三者的使用方法一样

★ 缺省都是显示器, 可根据需要进行输出重定向(上学期补充内容)

#### 13.2.2. 格式输出

需要掌握的基本格式:

不同数制: dec、hex、oct

设置宽度: setw

左右对齐: setiosflags(ios::left/right)

其余当作手册来查:

setfill、setprecision等

★ 输出格式可用控制符控制, 也可以流成员函数形式

cout << setw(20)            ⇔ cout.width(20)

cout << setprecision(9) ⇔ cout.precision(9)

...



## § 17. 输入输出流

### 2. 标准输出流

#### 2.3. 流成员函数put

形式: `cout.put(字符常量/字符变量)`

★ 功能与`putchar`相同, 输出一个字符

```
char a='A';  
cout.put(a);      //变量  
cout.put('A');    //常量  
cout.put('\x41'); //十六进制转义符  
cout.put('\101'); //八进制转义符  
cout.put(65);     //整数当作ASCII码  
cout.put(0x41);   //整数当作ASCII码(十六)  
cout.put(0101);   //整数当作ASCII码(八)
```

★ 允许连续调用

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout.put(72).put(0x65).put('l').put(0154).put('a'+14);  
    return 0;  
}
```

Hello



## § 17. 输入输出流

### 3. 标准输入流

#### 3.1. cin流

★ 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

<pre>#include &lt;iostream&gt; using namespace std; int main() {     int a=-9;     cin &gt;&gt; a;     cout &lt;&lt; a &lt;&lt; " " &lt;&lt; (cin ? 1 : 0) &lt;&lt; endl;     return 0; } //不同编译器，cin为0时，a值可能不同</pre>	输入	cout的结果	
	10	10	1
	ab	0	0
	12ab	12	1
	很大的数字	2147483647	0

- 当cin返回为1/true时，读入的值才可信  
=>正确的处理逻辑：cin读入后，先判断cin，为1再取值
- 不同编译器，cin为0时，a的值可能不同(不可信)

★ 允许进行输入重定向(上学期内容)



## § 17. 输入输出流

### 3. 标准输入流

#### 3.2. 文件结束符与文件结束标记

文件结束符：表示文件结束的特殊标记

- ★ 设备也当作文件处理
- ★ 一般用CTRL+Z表示键盘输入文件结束符

文件结束标记：判断文件是否结束的标记，用宏定义EOF来表示

- ★ 不同系统EOF的值可能不同，不必关心
- ★ 一般用于字符流输入的判断，对其它类型一般不用

#### 3.3. 用于字符输入的流成员函数

- ★ `cin.get()`
- ★ `cin.get(字符变量)`
- ★ `cin.get(字符数组, 字符个数n, 中止字符)`
- ★ `cin.getline(字符数组, 字符个数n, 中止字符)`
- ★ `cin.eof()`
- ★ `cin.peek()`
- ★ `cin.putback(字符变量/字符常量)`
- ★ `cin.ignore(字符个数n, 中止字符)`

上学期作业



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.1. 文件的基本概念

文件及文件名：

文件：存储在外存储器上的数据的集合

文件名：操作系统用于访问文件的依据

文件的分类：

#### ★ 按设备分

输入文件：键盘等输入设备

输出文件：显示器、打印机等输出设备

磁盘文件：存放在磁盘(光盘、U盘)上的文件

#### ★ 按文件的类型分：

程序文件：执行程序所对应的文件(.exe/.dll等)

数据文件：存放对应数据的文件(.cpp/.doc等)

#### ★ 按数据的组织形式

ASCII码文件(文本文件)：按数据的ASCII代码形式存放的文件

二进制文件：按数据的内存存放形式存放的文件



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.1. 文件的基本概念

文件的分类:

#### ★ 按数据的组织形式

ASCII码文件(文本文件): 按数据的ASCII代码形式存放的文件

二进制文件: 按数据的内存存放形式存放的文件

例: int型整数100000: ASCII文件为6个字节

二进制文件为4个字节

\x31	\x30	\x30	\x30	\x30	\x30
------	------	------	------	------	------

\x00	\x01	\x86	\xA0	100000=0x186A0
------	------	------	------	----------------

双精度数123.45: ASCII文件为6个字节

二进制文件为8个字节

\x31	\x32	\x33	\x2E	\x34	\x35
------	------	------	------	------	------

IEEE754 作业

d	o	u	b	l	e	格	式
---	---	---	---	---	---	---	---

字符串"China": ASCII文件为6个字节

\x43	\x68	\x69	\x6E	\x61	\x0
------	------	------	------	------	-----

\x43	\x68	\x69	\x6E	\x61	\x0
------	------	------	------	------	-----





## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.1. 文件的基本概念

C++对文件的访问:

低级I/O: 字符流方式输入/输出

高级I/O: 转换为数据指定形式的输入/输出

#### 4.2. 文件流类及文件流对象

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

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



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.3. 文件的打开与关闭

文件的打开:

文件流对象名.open(文件名, 打开方式);

★ 加#include <fstream>

★ 有多种打开方式

方 式	作 用
ios :: in	以输入方式打开文件
ios :: out	以输出方式打开文件(这是默认方式), 如果已有此名字的文件, 则将其原有内容全部清除
ios :: app	以输出方式打开文件, 写入的数据添加在文件末尾
ios :: ate	打开一个已有的文件, 文件指针指向文件末尾
ios :: trunc	打开一个文件, 如果文件已存在, 则删除其中全部数据; 如文件不存在, 则建立新文件。如已指定了 ios :: out 方式, 而未指定 ios :: app, ios :: ate, ios :: in, 则同时默认此方式
ios :: binary	以二进制方式打开一个文件, 如不指定此方式则默认为 ASCII 方式
ios :: nocreate	打开一个已有的文件, 如文件不存在, 则打开失败。nocreat 的意思是不建立新文件
ios :: noreplace	如果文件不存在则建立新文件, 如果文件已存在则操作失败, noreplace 的意思是不更新原有文件
ios :: in   ios :: out	以输入和输出方式打开文件, 文件可读可写
ios :: out   ios :: binary	以二进制方式打开一个输出文件
ios :: in   ios :: binary	以二进制方式打开一个输入文件

★ 各个打开方式可用“位或运算符|”进行组合

ios::nocreate  
ios::noreplace

DevC++/Linux不支持

在VS2019下是

ios::\_Nocreate  
ios::\_Noreplace



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.3. 文件的打开与关闭

文件的打开:

文件流对象名.open(文件名, 打开方式);

★ 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);
```

```
out.open(".\\C++\\aa.dat", ios::out);
```

```
out.open("\\C++\\aa.dat", ios::out | ios::app);
```

```
out.open("c:\\C++\\aa.dat", ios::out);
```

- VS2019等编译器, 如在集成环境内运行, 则当前目录是指**源程序文件(\*.cpp)所在的目录**, 如果离开集成环境(例如用cmd命令行运行), 则当前目录是指**可执行文件(\*.exe)所在目录**

★ 用"\"表示目录间分隔符的方式在Linux下无效, 用"/"方式则在Windows/Linux下均有效

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

```
out.open("../C++/aa.dat", ios::out);
```

```
out.open("./C++/aa.dat", ios::out);
```

```
out.open("/C++/aa.dat", ios::out | ios::app);
```

```
out.open("c:/C++/aa.dat", ios::out);
```

- 1、路径有绝对路径和相对路径两种
- 2、..表示父目录, .表示当前目录
- 3、为什么要\\而不能\?
- 4、假设程序在D:\test下运行, 这5个open, 分别打开的是下面的哪个aa.dat文件?

```
C:\
|--test      (文件夹)
|  |--aa.dat
|  |--C++    (文件夹)
|     |--aa.dat
|--C++       (文件夹)
|  |--aa.dat
```

```
D:\
|--test      (文件夹)
|  |--aa.dat
|  |--C++    (文件夹)
|     |--aa.dat
|--C++       (文件夹)
|  |--aa.dat
```



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.3. 文件的打开与关闭

文件的打开:

文件流对象名.open(文件名, 打开方式);

★ 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, 缺省仍为 ios::in
```

★ 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)

★ 执行open操作后, 要判断文件是否打开成功

<pre>if (!outfile) if (outfile.is_open()==0) if (!outfile.is_open())</pre>	<p>两种方法均可以 1、判断流对象自身 2、is_open函数</p>
--	--

<pre>if (outfile.open("f1.dat", ios::app)==0) //open返回void if (!outfile.open("f1.dat", ios::app)) if (outfile==NULL) //outfile不是指针</pre>	<p>错</p>
--	----------

文件的关闭:

文件流对象名.close();



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.3. 文件的打开与关闭

文件的打开:

文件流对象名.open(文件名, 打开方式);

文件的关闭:

文件流对象名.close();

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

不存在时: 失败  
存在时: 成功

```
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;
}
```

若换成ios::out, 则打开不受影响, 但后续读写会有问题

ifstream

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

不存在时: 成功(创建)  
存在时: 成功(覆盖)  
存在并只读: 失败

```
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;
}
```

ofstream

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

不存在时: 失败  
存在时: 成功(覆盖)  
存在并只读: 失败

```
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();
}
```

ofstream





## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.4. 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，同时前面介绍的关于cin/cout的 get/getline/put/eof/peek/putback/ignore等成员函数也可以被文件流对象所使用

★ >>和<<使用时的注意事项与cin、cout时相同

cin >> 变量 => infile >> 变量

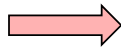
cout << 变量 => outfile << 变量

★ 成员函数的使用方法与前面相同

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

★ 流对象与打开方式、流插入/流提取运算符之间要求匹配

● 错误的例子：ifstream打开的写文件用流插入运算符



```
//例：打开d:\test\data.txt文件，并写入"Hello"
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream out;
    char ch;
    out.open("d:/test/data.txt", ios::out); //ios::out无效
    if (out.is_open()==0) {
        cout << "文件打开失败" << endl;
        return -1;
    }
    out << "Hello" << endl; //编译错!!!
    out.close();
    return 0;
}
```

error C2676: 二进制 "<<" : "std::ifstream" 未定义该运算符或到预定义运算符可接收的类型的转换



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.4. 对ASCII文件的操作

```
//例：键盘输入10个int，输出到文件中
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    int a[10];
    ofstream outfile("f1.dat", ios::out);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;
        exit(1); //结束程序运行，向操作系统返回1
    }
    cout << "enter 10 integer numbers:" << endl;
    for(int i=0; i<10; i++) {
        cin >> a[i]; //键盘输入
        outfile << a[i] << " "; //int型输出到文件
    }
    outfile.close();
    return 0;
}
```

运行两次，观察结果

```
//例：键盘输入10个int，输出到文件中（变化，加ios::app）
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    int a[10];
    ofstream outfile("f1.dat", ios::out | ios::app);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;
        exit(1); //结束程序运行，向操作系统返回1
    }
    cout << "enter 10 integer numbers:" << endl;
    for(int i=0; i<10; i++) {
        cin >> a[i]; //键盘输入
        outfile << a[i] << " "; //int型输出到文件
    }
    outfile.close();
    return 0;
}
```

运行两次，观察结果



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4. 4. 对ASCII文件的操作

```
//例：从文件中读入10个int，输出到屏幕上
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    int a[10];
    ifstream infile("f1.dat", ios::in);
    if (!infile.is_open()) {
        cerr << "open error!" << endl;
        exit(1); //结束程序运行，向操作系统返回1
    }
    for(int 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
    }

    infile.close();
    return 0;
}
```

利用上例生成的f1.dat  
自己编辑完全正确的f1.dat  
自己编辑含错误的f1.dat





## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.4. 对ASCII文件的操作

//例：打开d:\test\data.txt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;
    "d:/test/data.txt"
    in.open("d:\\test\\data.txt", ios::in); //双斜杠
    if (in.is_open() == 0) {                //!in.is_open()
        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);

EOF是系统定义的文件结束标记

//例：将 d:\test\data.txt 文件复制为 d:\demo\data2.txt

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

int main()
{
    ifstream in;
    ofstream out;
    char ch;
    "d:/test/data.txt"
    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(); //记得关掉
        return -1;
    }
}
```

```
while(in.get(ch))
    out.put(ch);
```

```
while(!in.eof()) {
    ch = in.get();
    out.put(ch);
}
```

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

```
}
```

问：1、左右两种复制方式，哪种方式复制后的字节大小与原文件不同？为什么？  
2、在保持读源文件的函数不变的情况下如何改正？  
3、左侧的输出到屏幕有同样问题吗？



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 对二进制文件的操作

##### ★ 用ASCII文件的字符方式进行操作

- 仅能按字节读写
- 如果文件中有0x1A则无法继续读取 (文本文件不可能有此字符)

##### ★ 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

- read/write均为纯字节, 无尾零等任何附加信息
- read/write一般仅用于二进制读写, 如果用于十进制读写, 则仅受长度的限制, 不考虑格式化 (是否有尾零/数据是否合理等, 相当于二进制)



# § 17. 输入输出流

- 4. 文件操作与文件流
- 4. 5. 对二进制文件的操作
- ★ 用read/write进行操作

```
//例：将内存以ASCII方式写入文件中（二进制转十进制）
#include <iostream>
#include <fstream>
#include <cstdlib> //exit用
using namespace std;

struct student {
    char name[20];
    int num;
    int age;
    char sex;
}; //含填充共32字节
int main()
{
    student stud[3]={“Li”,1001,18,‘f’, “Fan”,1002,19,‘m’,“Wang”,1004,17,‘f’}; //每个数组的存储为机内二进制形式
    ofstream outfile(“stud.dat”, ios::out);
    if (!outfile.is_open()) {
        cerr << “open error!” << endl;
        exit(-1); //强制结束程序
    }

    for(int i=0; i<3; i++)
        outfile << stud[i].name << stud[i].num << stud[i].age << stud[i].sex << endl; //通过 << 转换为十进制形式

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

以ASCII文件方式写入

生成的stud. dat文件共36字节:

Li100118f  
Fan100219m  
Wang100417f

	stud.dat
文件类型:	DAT 文件 (.dat)
打开方式:	 UltraEdit Professional
位置:	D:\WorkSpace\VS2019-De
大小:	36 字节 (36 字节)
占用空间:	0 字节



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 对二进制文件的操作

#### ★ 用read/write进行操作

<pre>//用read方式读ASCII文件（stud.dat由上例生成） #include &lt;iostream&gt; #include &lt;fstream&gt; using namespace std;  int main() {     ifstream infile("stud.dat", ios::in);     if (!infile.is_open()) {         cerr &lt;&lt; "open error!" &lt;&lt; endl;         exit(-1);     }     char name[20];     infile.read(name, 20);     cout &lt;&lt; '*' &lt;&lt; name &lt;&lt; '*' &lt;&lt; endl;     infile.close(); }</pre>	<pre>//用流对象方式读ASCII文件（stud.dat由上例生成） #include &lt;iostream&gt; #include &lt;fstream&gt; using namespace std;  int main() {     ifstream infile("stud.dat", ios::in);     if (!infile.is_open()) {         cerr &lt;&lt; "open error!" &lt;&lt; endl;         exit(-1);     }     char name[20];     infile &gt;&gt; name;     cout &lt;&lt; '*' &lt;&lt; name &lt;&lt; '*' &lt;&lt; endl;     infile.close(); }</pre>
<ol style="list-style-type: none"><li>1、观察read后name的输出</li><li>2、将read后的20改为2、200，再观察</li><li>3、如果read的长度超过文件长度，如何处理？</li><li>4、改成ios::in ios::binary，为什么少读了一个字符？</li></ol>	<ol style="list-style-type: none"><li>1、观察read后name的输出</li><li>2、用编辑软件将stud.dat文件的第3个字符改为空格，再观察</li></ol>



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 对二进制文件的操作

#### ★ 用read/write进行操作

//例：将内存以原始二进制方式写入文件中

```
#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
```

```
int main()
{
    student stud[3]={"Li", 1001, 18, 'f', "Fun", 1002, 19, 'm', "Wang", 1004, 17, 'f'}; //每个数组的存储为机内二进制形式
    ofstream outfile("stud.dat", ios::binary);
    if (!outfile.is_open()) {
        cerr << "open error!" << endl;
        exit(-1); //强制结束程序
    }
```

```
    for(int i=0; i<3; i++) //一次写一个数组元素 (32字节)
        outfile.write((char *)&stud[i], sizeof(stud[i]));
```

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

以二进制文件方式写入

stud.dat文件共96字节，前32字节为：

4C	69	00	??	??	??	??	??
??	??	??	??	??	??	??	??
??	??	??	??	E9	03	00	00
12	00	00	00	66	??	??	??

4C6900	=>	"Li" (含尾零, 多余17个)
E9030000	=>	0x000003E9 => 1001
12000000	=>	0x00000012 => 18
66	=>	'f' (后3个是填充字节)

outfile.write((char \*)stud, sizeof(stud)); //整个数组96字节)



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 对二进制文件的操作

#### ★ 用read/write进行操作

#### 以二进制文件方式读取

```
//用read方式读二进制文件（stud.dat由上例生成）
#include <iostream> //书上缺，要补上
#include <fstream>
using namespace std;
struct student {
    char name[20]; //不能是string name。必须是char name[20]，[19]或[21]都不行，必须要保证与文件的32字节一致
    int num;
    int age;
    char sex;
}; //含填充共32字节
int main()
{
    student stud[3];
    int i;
    ifstream infile("stud.dat", ios::binary); //stud.dat的内容是由上例生成的二进制文件
    if (!infile.is_open()) {
        cerr << "open error!" << endl;
        exit(-1); //退出
    }
    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;
}
```



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 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  
`tellg()` : 返回输入文件的当前指针  
`seekg(位移量, 位移方式)`: 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, 位移方式)`: 移动输出文件指针

位移方式：

`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

- `ifstream`无`tellp`, `ofstream`无`tellg`
- `fstream`的`tellg/tellp`是同步移动的



## § 17. 输入输出流

### 4. 文件操作与文件流

#### 4.5. 对二进制文件的操作

#### ★ 与文件指针有关的流成员函数

#### ★ 随机访问二进制数据文件

#### ★ 关于二进制访问的几个注意事项

- `read/write`虽然是内存首地址，实际编程中用字符数组，但注意不是字符串，不处理`\0`
- `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多用`good()/fail()/eof()/clear()`等函数，具体自行体会

#### ★ 结合作业加深理解





# C语言的文件操作

## 1. 文件指针

FILE \*文件指针变量

- ★ FILE是系统定义的结构体
- ★ C语言中文件操作的基本依据，所有针对文件的操作均要依据该指针
- ★ #include <stdio.h> (VS2019可以不需要)
- ★ VS2019以为不安全，需要加 #define \_CRT\_SECURE\_NO\_WARNINGS
- ★ 文件读写后，文件指针会自动后移



# C语言的文件操作

## 2. 文件的打开与关闭

假设: `FILE *fp`定义一个文件指针

### 2.1. 文件的打开

`FILE *fopen(文件名, 打开方式)`

`fp = fopen("test.dat", "r");`

`fp = fopen("c:\\demo\\test.dat", "w");` //也可表示为: `"c:/demo/test.dat"`

★ 打开的基本方式如下:

r: 只读方式

w: 只写方式

a: 追加方式

+: 可读可写

b: 二进制

t: 文本方式(缺省)

★ 打开的基本方式及组合见右表

★ 若带路径, 则\必须用\\表示

★ 若打开不成功, 则返回NUL

打开方式	意义
r/rt	只读方式打开文本文件(不存在则失败)
w/wt	只写方式打开或建立文本文件(存在则清零)
a/at	追加写方式打开或建立文本文件(头读尾写)
rb	只读方式打开二进制文件(不存在则失败)
wb	只写打开或建立二进制文件(存在则清零)
ab	追加写方式打开或建立二进制文件(头读尾写)
r+/rt+	读写方式打开文本文件(不存在则失败)
w+/wt+	读写方式创建文本文件(存在则清零)
a+/at+	读+追加写方式打开或建立文本文件(头读尾写)
rb+	读写方式打开二进制文件(不存在则失败)
wb+	读写方式创建二进制文件(存在则清零)
ab+	读+追加写方式打开二进制文件(头读尾写)

### 2.2. 文件的关闭

`fclose(文件指针)`

`fclose(fp);`



# C语言的文件操作

## 3. 文本文件的读写

### 3.1. 按字符读写文件

读: `int fgetc(文件指针)`

- 返回读到字符的ASCII码 (返回值同`getchar`)

写: `int fputc(字符常量/变量, 文件指针)`

- 返回写入字符的ASCII码 (返回值同`putchar`)

★ 必须保证文件的打开方式符合要求

```
char ch1;
ch1=fgetc(fp);

char ch2 = 'A';
fputc(ch2, fp);
```

### 3.2. 判断文件是否到达尾部

`int feof(文件指针)`

★ 若到达尾部, 返回1, 否则为0

### 3.3. 按格式读写文件

读: `int fscanf(文件指针, 格式串, 输入表列)`

- 返回读取正确的数量 (返回值同`scanf`)

写: `int fprintf(文件指针, 格式串, 输出表列)`

- 返回输出字符的个数 (返回值同`printf`)

★ 格式串、输入/输出表列的使用同`scanf/printf`

```
int i;
char ch;
fscanf(fp, "%d%c", &i, &ch);

int i=10;
char ch='A';
fprintf(fp, "%d%c", i, ch);
```



# C语言的文件操作

## 3. 文本文件的读写

### 3.4. 用文件方式进行标准输入输出

stdin : 标准输入设备  
stdout : 标准输出设备  
stderr : 错误输出设备

这三个是系统预置的FILE \*, 直接用, 不需要打开关闭

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

int main()
{
    int i;
    char ch;
    ch = fgetc(stdin);           ⇔ getchar();
    putchar(ch);
    fputc('A', stdout);         ⇔ putchar('A');
    fscanf(stdin, "%d", &i);     ⇔ scanf("%d", &i);
    fprintf(stdout, "i=%d\n", i); ⇔ printf("i=%d\n", i);
    fprintf(stderr, "i=%d\n", i); ⇔ cerr<<"i="<<i<< endl;

    return 0;
}
```

//C方式无专用错误输出  
//perror() 功能不同



# C语言的文件操作

## 3. 文本文件的读写

### 3.5. 用freopen重定向标准输入输出

★ FILE \*freopen(文件名, 打开方式, 原FILE \*);

功能: 将已存在的FILE \*映射为另一个新的FILE \*

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

int main()
{
    FILE *fp;
    if ((fp = freopen("out.txt", "w", stdout)) == NULL) {
        printf("freopen failed!\n");
        return -1;
    }
    printf("Hello, world!\n");
    fclose(fp);

    return 0;
}
```

- 1、正常运行, 观察运行结果
- 2、不删除已存在的out.txt, 换成"r", 观察运行结果
- 3、先删除已存在的out.txt, 换成"r", 观察运行结果
- 4、如果在fclose的后面再加printf, 能否正常输出? 如果可以, 输出到哪里了? 如果没有, 为什么?

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

int main()
{
    FILE *fp;
    int a, b;
    if ((fp = freopen("in.txt", "r", stdin)) == NULL) {
        printf("freopen failed!\n");
        return -1;
    }
    scanf("%d %d", &a, &b);
    printf("a=%d b=%d\n", a, b);
    fclose(fp);

    return 0;
}
```

- 1、在当前目录下建立in.txt文件, 写入两个整数, 观察运行结果
- 2、在当前目录下没有/删除in.txt的情况下运行, 观察运行结果
- 3、如果在fclose的后面再加scanf, 能否正常输入? 如果可以, 从哪里读? 如果不行, 为什么?



# C语言的文件操作

## 3. 文本文件的读写

### 3.5. 用freopen重定向标准输入输出

- ★ FILE \*freopen(文件名, 打开方式, 原FILE \*);  
功能: 将已存在的FILE \*映射为另一个新的FILE \*
- ★ 用freopen可以重定向普通文件 (一般不用)

观察运行结果

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

int main()
{
    FILE *fp, *fdup;
    if ((fp = fopen("out.txt", "w")) == NULL) {
        printf("fopen out.txt failed!\n");
        return -1;
    }
    fprintf(fp, "Hello, world!\n");

    if ((fdup = freopen("out_dup.txt", "w", fp)) == NULL) {
        printf("freopen failed!\n");
        fclose(fp);
        return -1;
    }
    fprintf(fp, "I am a student.\n"); //注意: 不是fdup !!!

    fclose(fp);
    fclose(fdup);
    return 0;
}
```



# C语言的文件操作

## 3. 文本文件的读写

### 3.6. 用popen/pclose与系统命令进行交互

★ VS2019下是\_popen与\_pclose

★ Linux下的popen与pclose

★ Dev C++下popen/pclose/\_popen/\_pclose均可

#### ★ Windows示例(分两步操作)

//假设编译为 D:\VS2019-Demo\Debug\demo-cpp.exe

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to Tongji University!" << endl;
    return 0;
}
```

Step1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

int main()

```
{ FILE* fp = _popen("D:\\VS2019-Demo\\Debug\\demo-cpp.exe", "r");
  if (fp == NULL) {
      printf("popen failed!\n");
      return -1;
  }
```

再换为 "dir C:\\Windows"

```
char ch;
while (1) {
    ch = fgetc(fp);
    if (feof(fp))
        break;
    putchar(ch);
}
_pclose(fp);
return 0;
}
```

Step2

#### ★ Linux示例(分两步操作)

//假设编译为 /home/u1234567/test

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to Tongji University!" << endl;
    return 0;
}
```

Step1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

int main()

```
{ FILE* fp;
  if ((fp = popen("/home/u1234567/test", "r")) == NULL) {
      printf("popen failed!\n");
      return -1;
  }
```

再换为 "ls -l /etc"

```
char ch;
while (1) {
    ch = fgetc(fp);
    if (feof(fp))
        break;
    putchar(ch);
}
pclose(fp);
return 0;
}
```

Step2



# C语言的文件操作

## 4. 二进制文件的读写

### 4.1. 按字符读写文件

读: `int fgetc(文件指针)`

- 返回读到字符的ASCII码 (返回值同`getchar`)

写: `int fputc(字符常量/变量, 文件指针)`

- 返回写入字符的ASCII码 (返回值同`putchar`)

★ 必须保证文件的打开方式符合要求

★ 同C++方式, 仅能按字符读写, 且文件中不能有0x1A

### 4.2. 按块读写文件

读: `int fread(缓冲区首址, 块大小, 块数, 文件指针)`

- ★ 返回读满的块数

写: `int fwrite(缓冲区首址, 块大小, 块数, 文件指针)`

- ★ 返回写入成功的块数





# C语言的文件操作

## 4. 文件指针的移动

### 4.1. 指针复位(回到开头)

`rewind(文件指针)`

例: `rewind(fp);`

### 4.2. 任意移动

`fseek(文件指针, 位移量, 位移方式)`

例: `fseek(fp, 123, SEEK_SET):` 从开始移动

`fseek(fp, 78, SEEK_CUR):`  
`fseek(fp, -25, SEEK_CUR):` } 从当前位置移动

`fseek(fp, -57, SEEK_END):` 从最后移动

★ `SEEK_SET`的位移必须为正

`SEEK_CUR`的位移可正可负

`SEEK_END`的位移必须为负

### 4.3. 求文件指针的当前位置

`long ftell(文件指针)`

例: `ftell(fp);`

★ 从开始位置计算



# C++的字符串流 (sstream)

## 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，进行文本和二进制之间的相互转换

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

## 2. 相关流对象的建立

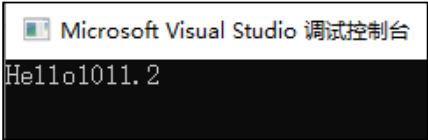
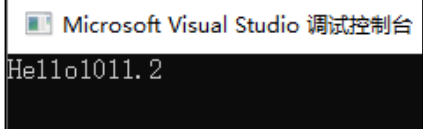
- 字符串输出流对象：
  - ostringstream 对象名
- 字符串输入流对象：
  - istringstream 对象名
- 字符串输入/输出流对象：
  - stringstream 对象名

★ 加 `#include <sstream>`



## C++的字符串流 (sstream)

### 3. 字符串输出流对象的使用

<pre>#include &lt;iostream&gt; #include &lt;sstream&gt; using namespace std;  int main() {     ostringstream out;     out &lt;&lt; "Hello" &lt;&lt; 10 &lt;&lt; 11.2 &lt;&lt; endl;      string s1 = out.str();     cout &lt;&lt; s1 &lt;&lt; endl;      return 0; }</pre> 	<b>例1: 观察cout的输出</b>	<pre>#include &lt;iostream&gt; #include &lt;sstream&gt; using namespace std;  int main() {     ostringstream out;     out &lt;&lt; "Hello" &lt;&lt; 10 &lt;&lt; 11.2 &lt;&lt; endl;      cout &lt;&lt; out.str() &lt;&lt; endl; //等价例1      return 0; }</pre> 	<b>例2: 观察cout的输出</b>
<p>★ 成员函数str()的作用: 将ostringstream的内容转换为string格式</p> <p>★ ostringstream最简单的用法: 将多个格式化内容拼在一起, 集中输出</p>			



# C++的字符串流 (sstream)

## 4. 字符串输入流对象的使用

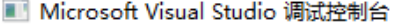
例1: 观察cout的输出

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

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() << endl;
    cout << in.str() << endl;
    return 0;
}
```



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

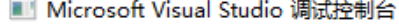
例2: 观察cout的输出

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

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

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

    cout << in.good() << endl;
    cout << in.str() << endl;
    return 0;
}
```



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

- ★ 可用str()打印现有内容
- ★ 读完后, 内容仍在
- ★ 如果现有内容全部读完, goodbit会置0



# C++的字符串流 (sstream)

## 4. 字符串输入流对象的使用

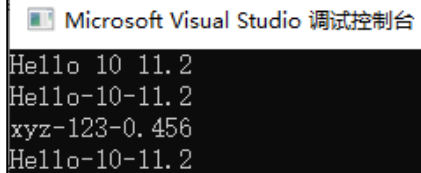
### 例3: 观察cout的输出

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

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;
    cout << s2 << '-' << i2 << '-' << f2 << endl;
    return 0;
}
```



Microsoft Visual Studio 调试控制台

```
Hello 10 11.2
Hello-10-11.2
xyz-123-0.456
Hello-10-11.2
```

★ istringstream的内容可重复读取



# C++的字符串流 (sstream)

## 4. 字符串输入流对象的使用

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

例4: 观察cout的输出

```
int main()
{
    istringstream in("Hello 70000 11.2");

    char s[10];
    short i;
    float f;

    in >> s;

    in >> i;

    in >> f;
    cout << s << '-' << i << '-' << f << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

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

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

例5: 观察cout的输出

```
int main()
{
    istringstream 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 >> f;
    cout << s << '-' << i << '-' << f << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台

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

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

例6: 观察cout的输出

```
int main()
{
    istringstream 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-32767-11.2
```

★ 如果数据超范围, 后续会错, 可clear() 恢复



# C++的字符串流 (sstream)

## 4. 字符串输入流对象的使用

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

### 例7：观察cout的输出

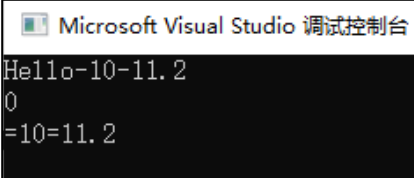
```
int main()
{
    istringstream in("Hello 10 11.2");

    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
```

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

### 例8：观察cout的输出

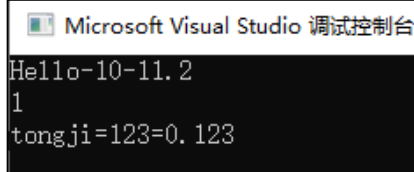
```
int main()
{
    istringstream in("Hello 10 11.212345");

    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



# C++的字符串流 (sstream)

## 5. 字符串输入/输出流对象的使用

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

```
}
```

### 例1: 观察cout的输出

```
Microsoft Visual Studio 调试控制台
Hello-10-11.2
-1
Hello 10 11.2
-1
```

```
#include <iostream>
#include <sstream>
using namespace std;
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.clear();
    ss << "xyz 123 0.456";
    cout << ss.str() << endl;
    cout << ss.tellg() << endl;
    ss.seekg(0, ios::beg);
    ss >> s >> i >> f;
    cout << s << '=' << i << '=' << f << endl;
```

```
    return 0;
```

```
}
```

### 例2: 观察cout的输出

```
Microsoft Visual Studio 调试控制台
Hello-10-11.2
-1
xyz 123 0.456
13
xyz=123=0.456
```

★ stringstream可读可写，但注意goodbit





# C++的字符串流 (sstream)

## 5. 字符串输入/输出流对象的使用

//先从流对象中输入数据，再把排序后的结果输出到流对象中

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

```
int main()
{
    stringstream ss("12 34 65 -23 -32 33 61 99 321 32");
    int a[10], i, j, t;

    for (i = 0; i < 10; i++)
        ss >> a[i]; //ss中的内容逐个读入int a[10]中

    cout << "array a:";
    for (i = 0; i < 10; i++) //输出int a[10]的内容
        cout << a[i] << " ";
    cout << endl;
```

//进行排序

```
for (i = 0; i < 9; i++)
    for (j = 0; j < 9 - 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，现在覆盖其中的内容)

```
ss.clear();
ss.seekg(0, ios::beg);
for (i = 0; i < 10; i++)
    ss << a[i] << " ";
ss << endl;
cout << "array a after sort:" << ss.str() << endl;

return 0;
}
```

例3：综合应用

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
```



## C++的字符串流 (stringstream)

3. 字符串输出流对象的使用
4. 字符串输入流对象的使用
5. 字符串输入/输出流对象的使用

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()
- ★ C++还有一个stringstream系列, 但是在新标准中已是deprecated
  - 要求: 能读懂别人用stringstream写的代码, 自己不准用!!!



## C++的字符串流 (sstream)

### 6. 用字符串流对象实现不同数据类型的转换

例1: 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;
}
```

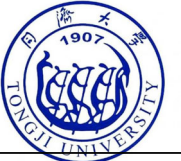
例2: double转字符串

```
#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 << end;

    return 0;
}
```



# C++的字符串流 (sstream)

## 6. 用字符串流对象实现不同数据类型的转换

<pre>#include &lt;iostream&gt; #include &lt;sstream&gt; using namespace std; string tj_to_string(const double d) {     ostringstream out;     out &lt;&lt; d;     return out.str(); } string tj_to_string(const int i) {     ostringstream out;     out &lt;&lt; i;     return out.str(); } string tj_to_string(const char ch) {     ostringstream out;     out &lt;&lt; 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 &lt;&lt; s1 &lt;&lt; endl;     cout &lt;&lt; s2 &lt;&lt; endl;     cout &lt;&lt; s3 &lt;&lt; endl; }</pre>	例3: 多种类型转字符串 (重载方式)	<pre>#include &lt;iostream&gt; #include &lt;sstream&gt; using namespace std;  template &lt;class T&gt; string tj_to_string(const T &amp;value) {     ostringstream out;     out &lt;&lt; value;     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 &lt;&lt; s1 &lt;&lt; endl;     cout &lt;&lt; s2 &lt;&lt; endl;     cout &lt;&lt; s3 &lt;&lt; endl;     return 0; }</pre>	例4: 多种类型转字符串 (模板方式)	<pre>#include &lt;iostream&gt; #include &lt;sstream&gt; using namespace std;  template &lt;class out_type, class in_type&gt; out_type tj_convert(const in_type &amp;value) {     stringstream ss;     ss &lt;&lt; value;      out_type ret;     ss &gt;&gt; ret;     return ret; }  int main() {     string s = tj_convert&lt;string, double&gt;(123.456);     cout &lt;&lt; s &lt;&lt; endl;      double d = tj_convert&lt;double, string&gt;("12.34abc");     cout &lt;&lt; d &lt;&lt; endl;      int k = tj_convert&lt;int, double&gt;(123.456);     cout &lt;&lt; k &lt;&lt; endl;      return 0; }</pre>	例5: 多种类型互转 (模板方式)
---	------------------------	---	------------------------	---	----------------------



# C语言中实现与C++的字符串流相似的功能

## 1. 向字符数组输出格式化的数据

`int sprintf(字符数组, "格式串", 输出表列);`

★ 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

★ 与printf相同, 完成二进制向ASCII的转换

★ VS2019下需加 `#define _CRT_SECURE_NO_WARNINGS`

```
//例: 将不同数据输出到ostringstream中
#include <iostream>
#include <sstream>
using namespace std;

int main()
{
    ostringstream out;

    out << "Hello" << 10 << 11.2 << endl;
    cout << out.str() << endl;

    return 0;
}
```

Microsoft Visual Studio 调试控制台  
Hello1011.2



```
//将不同数据输出到字符数组中
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    char c[80];
    int ret;
    ret = sprintf(c, "%s%d%.1f", "Hello", 10, 11.2);
    printf("%s\n", c);
    printf("ret=%d\n", ret); //搞懂ret的含义

    return 0;
}
```

Microsoft Visual Studio 调试控制台  
Hello1011.2  
ret=11



# C语言中实现与C++的字符串流相似的功能

## 1. 向字符数组输出格式化的数据

int sprintf(字符数组, "格式串", 输出表列);

//例: 接结构体的内容输出到一维字符数组中

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
struct student {
```

```
    int num;
```

```
    char name[20];
```

```
    float score;
```

```
};
```

```
int main()
```

```
{
```

```
    struct 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);
```

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

```
    return 0;
```

```
}
```

Microsoft Visual Studio 调试控制台

array c:1001 Li 78.01002 Wang 89.51004 Fun 90.0

多次向字符数组输出格式化数据  
(注意: 和C++方式的不同)

想明白s+=sprintf(s, "");的用法



## C语言中实现与C++的字符串流相似的功能

### 2. 从字符数组中输入格式化的数据

int sscanf(字符数组, "格式串", 输入表列);

- ★ 返回值是正确读入的输入数据的个数
- ★ 与scanf相同, 完成ASCII向二进制的转换
- ★ VS2019下需加 #define \_CRT\_SECURE\_NO\_WARNINGS

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

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  
Hello1011.2

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

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;
}
```

Microsoft Visual Studio 调试控制台  
Hello1011.2  
ret=3