cout 输出格式控制

如果要在输出流中加入格式控制符则要加载头文件：#include <iomanip>

这里面 iomanip 的作用比较多:

主要是对 cin,cout 之类的一些操纵运算子，比如 setfill,setw,setbase,setprecision 等等。它是 I/O 流控制头文件,就像 C 里面的格式化输出一样.以下是一些常见的控制函数的:

dec 置基数为 10 相当于"%d"

hex 置基数为 16 相当于"%X"

oct 置基数为 8 相当于"%o"　　//作用永久

sample:

cout<<12<<hex<<12<<oct<<12<<12;output 12c1414

setprecision(n) 设显示小数精度为 n 位 //作用永久

sample:

setf(ios:fixed);

cout<<setprecision(2)<<2.345<<endl; ouput 2.34 //注意先用 setf(ios::fixed);否则结果自己测试下

setw(n) 设域宽为 n 个字符 //作用临时

这个控制符的意思是保证输出宽度为 n。如:

cout<<setw(3)<<1<<setw(3)<<10<<setw(3)<<100; 输出结果为

1 10100 （默认是右对齐）当输出长度大于 3 时(<<1000)，setw(3)不起作用。

setfill(c) 设填充字符为 c

setioflags(ios::fixed) 固定的浮点显示

setioflags(ios::scientific) 指数表示

sample cout<<setiosflags(ios::fixed)<<setprecision(2)<<2.345<<endl; output 2.34

setiosflags(ios::left) 左对齐

setiosflags(ios::right) 右对齐

setiosflags(ios::skipws) 忽略前导空白

setiosflags(ios::uppercase) 16 进制数大写输出

setiosflags(ios::lowercase) 16 进制小写输出

setiosflags(ios::showpoint) 强制显示小数点

setiosflags(ios::showpos) 强制显示符号

sample: cout<<setiosflags(ios::uppercase)<<hex<<12<<15<<endl; output CF

cout<<setioflags(ios::showpoint)<<x<<endl;若 float x=1,则 output 1.000000 不使用直接输出 1

cout<<setiosflags(ios::showpos)<<1<<endl;output +1

```
//使用标准 C++编写
#include <iostream>
#include <iomanip>//精度设置必须包括的头文件

using namespace std;

int main()
{
double a=3.5;
int b=10;
//方法一：操作符函数的格式控制
//cout.precision(2),设置精度为 2
//right:设置左对齐；fixed:控制浮点型输出格式；
//setw(5):设置输出位宽为 5

cout<<right<<fixed<<setw(5)<<setfill('0')
<<setprecision(2)<<a<<endl; //输出结果为 03.50

//方法二：IOS 类成员函数的格式控制
cout.precision(4); //setprecision(4),设置精度为 4
cout<<a<<endl; //输出结果为 3.5000

//setfill('0')：设置填充字符为'0'
//static_cast<double>(b):将整型的 b,
//生成一个双精度浮点型副本进行操作，而不改变其值和类型
cout<<fixed<<setfill('0')<<setprecision(2)
<<fixed<<static_cast<double>(b)<<endl;//输出 10.00
return 0;
}
```

方法很多种啦，我们可以这样写：

```
/*一个使用填充，宽度，对齐方式的例子*/
#include <iostream.h>
void main()
{
    cout<<"第一章"<<endl;
```

```
    cout<<"    ";
    cout.setf(ios::left);          //设置对齐方式为 left
    cout.width(7);                 //设置宽度为 7，不足用空格填充
    cout<<"1.1";
    cout<<"什么是 C 语言";
    cout.unsetf(ios::left);        //取消对齐方式，用缺省 right 方式
    cout.fill('.');                //设置填充方式
    cout.width(30);                //设置宽度，只对下条输出有用
    cout<<1<<endl;
    cout<<"    ";
    cout.width(7);                 //设置宽度
    cout.setf(ios::left);          //设置对齐方式为 left
    cout.fill(' ');                //设置填充，缺省为空格
    cout<<"1.11";
    cout<<"C 语言的历史";
    cout.unsetf(ios::left);        //取消对齐方式
    cout.fill('.');
    cout.width(30);
    cout<<58<<endl;
    cout.fill(' ');
    cout<<"第二章"<<endl;
}
```

我们多次设置了宽度，为的是使我们的间距能一致，也使用了对齐方式，为的是使我们的数据能对齐显示，看起来美观。我们还使用了填充方式。我们下面用操纵算子来实现也是可以的。

```
/*一个使用填充，宽度，对齐方式的例子*/
#include <iomanip.h>
void main()
{
    cout<<"第一章"<<endl;
    cout<<"    ";
    cout<<setiosflags(ios::left)<<setw(7);          //设置宽度为 7，left 对齐方式
    cout<<"1.1";
    cout<<"什么是 C 语言";
    cout<<resetiosflags(ios::left);                 //取消对齐方式
    cout<<setfill('.')<<setw(30)<<1<<endl;          //宽度为 30，填充为'.'输出
    cout<<setfill(' ');                             //恢复填充为空格
    cout<<"    ";
    cout<<setw(7)<<setiosflags(ios::left);          //设置宽度为 7，left 对齐方式
    cout<<"1.11";
    cout<<"C 语言的历史";
    cout<<resetiosflags(ios::left);                 //取消对齐方式
    cout<<setfill('.')<<setw(30)<<58<<endl;         //宽度为 30，填充为'.'输出
    cout<<setfill(' ')<<"第二章"<<endl;
}
```

我们输出了同样的效果，不过依我的性格，我更喜欢用操纵算子来进行格式化输出。最后我们看看浮点数的格式输出，如下例：

```
/*关于浮点数的格式*/
#include <iostream.h>
void main()
{
    float f=2.0/3.0,f1=0.000000001,f2=-9.9;
    cout<<f<<' '<<f1<<' '<<f2<<endl;       //正常输出
    cout.setf(ios::showpos);               //强制在正数前加+号
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout.unsetf(ios::showpos);             //取消正数前加+号
    cout.setf(ios::showpoint);             //强制显示小数点后的无效 0
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout.unsetf(ios::showpoint);           //取消显示小数点后的无效 0
    cout.setf(ios::scientific);            //科学记数法
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout.unsetf(ios::scientific);          //取消科学记数法
    cout.setf(ios::fixed);                 //按点输出显示
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout.unsetf(ios::fixed);               //取消按点输出显示
    cout.precision(18);                    //精度为 18，正常为 6
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout.precision(6);                     //精度恢复为 6
}
```

同样，我们也一样能用操纵算子实现同样的功能：

```
/*关于浮点数的格式*/
#include <iomanip.h>
void main()
{
    float f=2.0/3.0,f1=0.000000001,f2=-9.9;
    cout<<f<<' '<<f1<<' '<<f2<<endl;       //正常输出
    cout<<setiosflags(ios::showpos);       //强制在正数前加+号
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout<<resetiosflags(ios::showpos);     //取消正数前加+号
    cout<<setiosflags(ios::showpoint);     //强制显示小数点后的无效 0
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout<<resetiosflags(ios::showpoint); //取消显示小数点后的无效 0
    cout<<setiosflags(ios::scientific);   //科学记数法
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout<<resetiosflags(ios::scientific); //取消科学记数法
    cout<<setiosflags(ios::fixed);         //按点输出显示
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout<<resetiosflags(ios::fixed);       //取消按点输出显示
    cout<<setprecision(18);                //精度为 18，正常为 6
```

```
    cout<<f<<' '<<f1<<' '<<f2<<endl;
    cout<<setprecision(6);                    //精度恢复为 6
}
```

在 c/c++系统中除了标准的输入输出外，还提供了更多的输入函数。这写函数主要有 getch(),getche(), getchar
(),cin.get(),putch(),putchar(),cout.put(),gets(),cin.getline(),puts()。另外
还有些为了让缓冲区不影响程序的正确操作的缓冲去的操作，如：cin.putback(),fflush(stdin),cout.flush().我们做一下简单的说明。

1、getch()和 getche(),非缓冲式输入，从键盘读入一个字符。getch()读入字符不显示。有 conio.h 支持。

2、cin.get(),getchar()，缓冲式输入，从键盘读入一个字符，并显示。getchar()由 stdio.h 支持，cin.get()由 iostream.h 支持。

3、putch()和 putchar()，非缓冲式输出，输出一个字符到显示器。putch()由 conio.h 支持，putchar()由 stdio.h 支持。

4、cout.put()，缓冲式输出，输出一个字符到显示器。由 iostream.h 支持。

5、gets()和 cin.geline()，缓冲式输入，读入一字符串（包括空格，不包括最后的回车），gets()由 stdio.h 支持，cin.getline()由 iostream.h 支持。

6、puts()，非缓冲输出，输出一个字符串，由 stdio.h 支持。

7、cin.putback()，把一个字符送回输入缓冲区。

8、fflush(stdin)，清除输入缓冲区操作。无法清除 cin.get()等带来的操作。

9、cout.flush()，清楚输出缓冲区。

在这里我们稍微说一下输入/输出缓冲区，这是为了减少程序访问 io 带来中断而设的一段空间。当程序满足某个刷新条件时，那就将清理缓冲区。具体条件为：

1、输入缓冲区

a，程序要求输入时，按下了回车键。

b，遇到程序结束。

c，遇到下一条输入语句。

d，遇到清除缓冲区操作

e，缓冲区溢出

2、输出缓冲区

a，输出缓冲区溢出

b，遇到下一条输入语句

c，使用刷新缓冲区迫使清除

d，程序结束。

缓冲区操作有时会带来程序的不正确的输入，如前面说的 scanf()，在连续输入的时候，会把一个回车赋给下一个字符变量。我们操作的时候一定要注意。

使用标记进行格式设置的 setf()函数

格式化的各个方面，

要使用各种格式只需把该位设置为 1 即可，setf 函数就是用来设置标记的成员函数，使用该函数需要一个或两个在 c++

中定义的位的常量值作为参数，这些位的常量值在下面介绍。

1、c++在 ios 类中定义了位的常量值，这些值如下：

a、ios::boolalpha 使布尔值用字符表示

b、iso::showbase 显示进制的基数表示即 16 进位以 0x 开始，8 进制以 0 开始，

c、ios::showpoint 显示末尾的小数点，
d、、ios::uppercase 使 16 进制的 a~f 用大写表示，同时 0x 还有科学计数的字母 e 都用大写表示，
e、ios::showpos 在十进制的正数前面显示符号+

注意这些常量都是 ios 类中定义的，所以使用他们时需要加上 ios::限定符

比如：cout.setf(ios::showbase);注意使用
setf 函数时需要用对象来调用，因为它是流类的成员函数。

2、使用带两个参数的 setf()函数：setf()函数不但可以带有一个参数，还可以带有两个参数。带两个参数的 setf()函数的
第一个参数为要设置的位的值，第二个参数为要清除的位的值。

参数说明如图所示

setf()函数的第一和第二 参数

第一个参数	第二个参数	说明
ios::dec	ios::basefield	比如：cout.setf(ios::hex, ios::basefield);表示把 16 进制位设为 1，而 dec 和 oct 位设为 0，最后以 16 进制输出
ios::oct		
ios::hex		
ios::fixed	ios::floatfield	比如：cout.setf(ios::scientific,ios::floatfield);表示把科学计数表示浮点数的位设为 1，而把以浮点数表示的设为 0。最后以科学计数法表示浮点数
ios::scientific		
ios::left	ios::adjustfield	比如：cout.setf(ios::left,iosadjustfield);表示把左对齐的位设为 1，把右对齐和居中设为 0，最后输出以左对齐
ios::right		
ios::internal		

在 C++中有一个受保护的数据成员，其中的各位分别控制着
3、用 setf()函数设置的格式需要使用对应的 unsetf()函数来恢复以前的设置，比如 setf(ios::boolalpha)将使布尔值以字
符的形式使用，如再使用 unsetf(ios::boolapha)则又将使布尔值以数字的形式使用。
带两个参数的 setf 函数可以使用第 1 个参数为 0 的 unsetf 函数来恢复其默认值，比如 unsetf(0,ios::basefield)或直接使用 unsetf(ios::basefield)，注
意在 visual C++中不能使用第一个参数为 0 的 unsetf 函数，只能使用第二种形式。

格式输出
在输出数据时，为简便起见，往往不指定输出的格式，由系统根据数据的类型采取默认的格式，但有时希望数据按指定的格式输出，如要求以下六进制或八进制形式输出一个整数，对输出的小数只保留两位小数等；有两种方法可以达到此目的。一种是使用控制符；另一种是使用流对象的有关成员函数。分别叙述如下：

1、 用控制符控制输出格式

表 7.3　输入输出流的控制符

控 制 符	作　　用
dec	设置整数的基数为 10
hex	设置整数的基数为 16
oct	设置整数的基数为 8
setbase(n)	设置整数的基数为 n(n 只能是 8,10,16 三者之一)
setfill(c)	设置填充字符 c,c 可以是字符常量或字符变量
setprecision(n)	设置实数的精度为 n 位。在以一般十进制小数形式输出时 n 代表有效数字。在以 fixed(固定小数位数)形式和 scientific(指数)形式输出时 n 为小数位数
setw(n)	设置字段宽度为 n 位
setiosflags(ios::fixed)	设置浮点数以固定的小数位数显示
setiosflags(ios::scientific)	设置浮点数以科学记数法(即指数形式)显示
setiosflags(ios::left)	输出数据左对齐
setiosflags(ios::right)	输出数据右对齐
setiosflags(ios::skipws)	忽略前导的空格
setiosflags(ios::uppercase)	在以科学记数法输出 E 和以十六进制输出字母 X 时以大写表示
setiosflags(ios::showpos)	输出正数时给出“+”号
resetioflags( )	终止已设置的输出格式状态,在括号中应指定内容

应当注意:

这些控制符是在头文件 iomanip 中定义的，因而程序中应当包含头文件 iomanip。通过下面的例子可以了解使用它们的方法，

[indent]例 2　用控制符控制输出格式

，

```
#include <iostream>
#include <iomanip> //不要忘记包含此头文件


using namespace std;
int main()
{
int a;
cout<<"input a:";
cin>>a;
cout<<"dec:"<<dec<<a<<endl; //以上进制形式输出整数
cout<<"hex:"<<hex<<a<<endl; //以十六进制形式输出整数 a
cout<<"oct:"<<setbase(8)<<a<<endl;//以八进制形式输出整数 a
char *pt="China";
//pt 指向字符串”China”
cout<<setw(10)<<pt<<endl; //指定域宽为 10，输出字符串
cout<<setfill('*')<<setw(10)<<pt<<endl;//指定域宽 10，输出字符
串，空白处以“*”填充
double pi=22.0/7.0; //计算 pi 值
cout<<setiosflags(ios::scientific)<<setprecision(8);//按指数
形式输出，8 位小数
cout<<"pi="<<pi<<endl; //输出 pi 值
```

```
cout<<"pi="<<setprecision(4)<<pi<<endl;//改为 4 位小数
cout<<"pi="<<setiosflags(ios::fixed)<<pi<<endl;//改为小数形式输出
return 0; }
```

运行结果如下

:

inputa：34（输入 a 的值）

dec：34（十进制形式）

hex：22（十六进制形）

oct：42（八进制形式）

China（域宽为 10）

***** China（域宽为 10，空白处以'*'填充）

pi=3.14285714e+00（指数形式输出，8 位小数）

pi=3.1429e+00）（指数形式输小，4 位小数）

pi=3.143（小数形式输出，梢度仍为 4）

2. 用流对象的成员函数控制输出格式

除了可以用控制符来控制输出格式外，还可以通过调用流对象 COUt 中用于控制输出格式的成员函数来控制输出格式。用于控制输出格式的常用的成员函数见表 4。

表 7.4　用于控制输出格式的流成员函数

流成员函数	与之作用相同的控制符	作　　用
precision(n)	setprecision(n)	设置实数的精度为 n 位
width(n)	setw(n)	设置字段宽度为 n 位
fill(c)	setfill(c)	设置填充字符 c
setf( )	setiosflags( )	设置输出格式状态，括号中应给出格式状态，内容与控制符 setiosflags 括号中的内容相同，如表 7.5 所示
unsetf( )	resetioflags( )	终止已设置的输出格式状态，在括号中应指定内容

流成员函数 setf 和控制符 setiosflags 括号中的参数表示格式状态，它是通过格式标志来指定的。格式标志在类 ios 中被定义为枚举值。因此在引用这些格式标志时要在前面加上类名 ios 和域运算符“::”。格式标志见表 5。

表 7.5　设置格式状态的格式标志

格式标志	作　　用
ios :: left	输出数据在本域宽范围内向左对齐
ios :: right	输出数据在本域宽范围内向右对齐
ios :: internal	数值的符号位在域宽内左对齐，数值右对齐，中间由填充字符填充
ios :: dec	设置整数的基数为 10
ios :: oct	设置整数的基数为 8
ios :: hex	设置整数的基数为 16
ios :: showbase	强制输出整数的基数（八进制数以 0 打头，十六进制数以 0x 打头）
ios :: showpoint	强制输出浮点数的小点和尾数 0
ios :: uppercase	在以科学记数法格式 E 和以十六进制输出字母时以大写表示
ios :: showpos	对正数显示“ +”号
ios :: scientific	浮点数以科学记数法格式输出
ios :: fixed	浮点数以定点格式（小数形式）输出
ios :: unitbuf	每次输出之后刷新所有的流
ios :: stdio	每次输出之后清除 stdout，stderr

[/indent][indent]例 3 用流控制成员函数输出数据。

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

{
int a=21;
```

```
cout.setf(ios::showbase); //设置输出时的基数符号
cout<<"dec:"<<a<<endl; //默认以十进制形式输出 a
cout.unsetf(ios::dec); //终止十进制的格式设置
cout.setf(ios::hex); //设置以十六进制输出的状态
cout<<"hex:"<<a<<endl; //以十六进制形式输出 a
cout.unsetf(ios::hex); //终止十六进制的格式设置
cout.setf(ios::oct); //设置以八进制输出的状态
cout<<"oct:"<<a<<endl; //以八进制形式输出 a
cout.unsetf(ios::oct); //终止以八进制的输出格式设置
char *pt="China"; //pt 指向字符串” china”
cout.width(10); //指定域宽为 10
cout<<pt<<endl; //输出字符串
cout.width(10); //指定域宽为 10
cout.fill('*'); //指定空白处以'*'填充
cout<<pt<<endl; //输出字符串
double pi=22.0/7.0; //计算 pi 值
cout.setf(ios::scientific);//指定用科学记数法输出
cout<<"pi="; //输出"pi="
cout.width(14); //指定域宽为 14
cout<<pi<<endl; //输出"pi 值
cout.unsetf(ios::scientific); //终止科学记数法状态
cout.setf(ios::fixed); //指定用定点形式输出
```

```
cout.width(12); //指定域宽为 12
cout.setf(ios::showpos); //在输出正数时显示“+”号
cout.setf(ios::internal); //数符出现在左侧
cout.precision(6); //保留 6 位小数
cout<<pi<<endl; //输出 pi，注意数符“+”的位置
return 0;}
```

运行情况如下：

```
dec：21（十进制形式）
hex：0xl5（十六进制形式，以 0x 开头）
oct：025（八进制形式，以 0 开头）
China（域宽为 10）
*****china（域宽为 10，空白处以'*'填充）
pi=**3.142857e+00（指数形式输出，域宽 14，默认 6 位小数）
****3.142857（小数形式输㈠，精度为 6，最左侧输出数符“+”）
```

说明：

1、成员函数width(n)和控制符setw(n)只对其后的第一个输出项有效。如果要求在输出数据时都按指定的同一域宽n输出，不能只调用一次width(n)，而必须在输出每一项前都调用一次width(n)。

2、在表5中的输出格式状态分为5组，每一组中同时只能选用一种(例如，dec，hex和oct中只能选一，它们是互相排斥的)，在用成员函数setf和控制符setiosflags设置输出格式状态后,如果想改设置为同组的另一状态，应当调用成员函数unsetf(对应于成员函数setf)或resetiosflags(对应于控制符setiosflags)，先终止原来设置的状态。然后再设置其他状态。

同理，程序倒数第8行的unsetf函数的调用也是不可缺少的。读者不妨上机试一试。

3、用setf函数设置格式状态时，可以包含两个或多个格式标志，由于这些格式标志在iOS类中被定义为枚举值，每一个格式标志以一个二进位代表，因此可以用“位或”运算符“I”组合多个格式标志

4、可以看到：对输出格式的控制，既可以用控制符(如例 2)，也可以用 cout 流的有关成员函数(如例 3)，二者的作用是相同的。控制符是在头文件 iomanip 中定义的，因此用控制符时，必须包含 iomanip 头文件。cout 流的成员函数是在头文件 iostream 中定义的，因此只需包含头文件 iostream，不必包含 iomanip。许多程序人员感到使用控制符方便简单，可以在一个 cout 输出语句中连续使用多种控制符。

5、关于输山格式的控制，在使用中还会遇到一些细节问题，不可能在这里全部涉及。在遇到问题时，请查阅专门手册或上机试验一下即可解决。

用于 char 类型的字符数组的输入流类的成员函数
使用输入流类的成员函数进行输入：
注意：以下这些成员函数都只能用于 char 类型的字符数组，而不能用于 string 类型的对象。

1、使用 get()函数输入单个字符：输入单个字符的 get()函数都是类 istream 中的成员函数，调用他们需要使用类的对象
来调用，该函数有两个原型，即 get(char &ch)和 get(void)。
a、get(char &ch)函数：该函数返回调用对象的引用，这里要注意该函数的参数类型必须要是 char 类型的，不能是 int
型变量，比如 cin.get(a)其中参数 a 只能是 char 类型，不能是 int 型，如果是 int 型则会出现错误。该函数可以连
续输入，即 cin.get(a).get(a);
b、get(void)函数:该函数反回 int 型的值,调用该 get 函数时不需要使用参数。该函数不能连续输入，比如 cin.get().get()
就是错误的。
c、两个 get 函数都接收输入的单个字符，且不跳过空白符号和回车换行符。如

果输入了 2 个以上的字符，则以后的
字符保存在输入流中，留给下一次输入。比如有 char a; cin.get(a);cout<<a; cin.get(a); cout<<a;如果输入 a 并按回车
的话，则第二次的 get 调用将不会再提示输入字符，而是接受了第一次输入的单个字符 a 后面的回车换行了，所
以最后只输入一次，并输出 a 再换行。同样如果连续输入两个字符比如 ad 则第一个字符 a 赋给第一个变量，第
二个字符 d 赋给第二个变量，同样不会出现提示两次输入的情况，这不是我们所预期希望的效果。同样 get()函数
有同样的笑果，即 char a; a=cin.get(); cout<<a; a=cin.get(); cout<<a;如果输入 a 并按回车，有和 get(char &ch)同样的
笑果。注意使用>>操作符输入时将忽略掉空格和回车换行符等符号，而 get 函数则不会。解决上述问题的方法是
在第二次输入前使用 ignore()函数读取并丢弃剩下的字符，这样就会提示两次输入。

2、使用 get 和 getline 函数输入字符串：
a、字符串输入的 get 和 getline 函数原型如下：get(char *, int ,char); get(char* ,int); getline(char*,int ,char); getline(char*,int);
其反回类型都为 istream &也就是说这几个函数都可以拼接输出。其中两个参数的函数输入指定长度的字符串，第二
个参数的数目要比将要输入的字符数目大 1，因为最后一个字符将作为字符串结尾的字符。带有三个参数的函数的
第三个参数是分界符字符，也就是说当输入时遇到第三个字符就不会再输入后面的字符了，即使输入的字符串没有
达到指定的长度。
b、get 与 getline 的区别是 get 函数将分界符留在输入流中，这样下次再输入时将输入的是这个分界符。而 getline 则是
读取并丢弃这个分界符。比如 cin.get(a,3,’z’)如果输入 abz 则字符 z 将留在输入流中，等待下一次输入，比如在 get
函数后接着有 cin>>b;则字符 z 将赋给变量 b，而不会再次提示输入。
c、输入的字符数超过了指定数量长度时的处理情况：对于 getline 函数来说，如果输入的字符数大于指定的字符数的长
度，且最后一个字符不是分界符时，将设置 failbit 位。比如 cin.getline(a,3,’z’)如果输入 abdc,则会设置 failbit 位，而
如果输入 abzde 则不会设置 failbit 位，因为虽然输入超过了指定的长度，但是最后一个字符是分界符 z，所以不会设
置 failbit 位，而会将分界字符 z 后面的字符留在输入流中，留给下一次输入。而对于 get 函数当输入的字符数超过了
指定的长度时则不会设置 failbit 位，对于 get 函数可以使用 peek()函数来检查是否程序是正常结束输入。如果使用 get
函数输入超出指定数目的字符时，多于的字符将作为下一次输入的字符，而对于 getline 函数而言则会关闭下一次输

入，因为 getline 函数在输入的字符数超过了指定的数量时将设置状态位 failbit，在状态位被清除前输入会被关闭，
除非被重设。对于这两个函数而言，当达到文件尾时都将设置 eofbit 位，流被破坏时设置 badbit 位。
d、输入是空字符时的处理情况：对于 get 函数而言，如果输入的是一个空字符则会设置 failbit 位，但对于 getline 函数
来说则不会设置该位。比如 cin.get(a,3);这时如果输入时直接按下回车的话将使 get 函数设置 failbit 位，而对于 getline
函数而言则不会设置该位。
e、对于以上的 get 和 getline 函数，不管是输入的是单个字符还是字符串，都需要使用 ignore 函数来读取并丢弃多余的
输入字符，以使后面的输入程序能正常的工作。

3、read()函数：函数原型为 iostream& read(const char* addr,streamsize n)。调用方法为 cin.read(a,144)；表示输入 144 个
字符放到地址从 a 开始的内存中，如果还未读取 144 个字符就到达了文件末尾，就设置 ios::failbit。read 函数与 get
和 getline 不同的是 read 函数不会在输入后加上空值字符，与就是说输入的数目不必比指定的数目少 1，也就是不能
将输入的字符转换为字符串了。read 函数可以拼接。

4、readsome()函数：原型为 iostream& readsome(char* addr,streamsize n)。表示把 n 个字符放到地址从 addr 开始的内存中，
该函数和 read 函数差不多，区别在于，如果没有读取 n 个字符，则设置文件结束状态位 ios::eofbit。

5、write()函数：原型为 iostream& write(const char* addr,streamsize n);表示把从地址 addr 开始的 n 个字符写入到流中。

6、peek()函数：peek 函数反回输入流中的下一个字符，但不抽取输入流中的字符，也就是说他使得能够查看下一个输
入字符。

7、gcount()函数：反回最后一个非格式化抽取方法读取的字符数。非格式化抽取方法即 get 和 getline 这样的函数，>>
这个运算符是格式化抽取方法。

8、strlen()函数：计算数组中的字符数，这种方法比使用 gcount 函数计算字符

数要快。

9、注意：以上的函数都是输入流类的成员函数，使用他们时需要使用输入流类的对象来调用，比如 cin.get()等。

再次提醒:以上的函数只适合于 char 类型的数组,不适用于 string 类型的对象。比如 string a; cin.get(a,3);则将发生错
误，因类 string 类型的对象 a 无法转换为 char 类型的数组。