

2023

面向对象程序设计

第八讲：文件操作和输入输出流

李际军

lijijun@cs.zju.edu.cn



学习目标 / GOALS



- (1) 了解 C++ 的输入 / 输出的概念。
- (2) 掌握使用 cin 进行输入。
- (3) 掌握 istream 类的方法进行输入。
- (4) 掌握使用 cout 进行输出。
- (5) 掌握格式化输出。
- (6) 掌握 ostream 类的方法进行输出。
- (7) 掌握文件的输入和输出。



01

C++ 的输入 / 输出

02

标准输入流

03

标准输出流

04

文件的输入和输出

05

文件的输入和输出

06

程序实例

01



C++ 的输入 / 输出

- 数据的输入输出是数据运动的过程，如同流水，从一处流到另一处。C++ 形象地将此过程称作**流（stream）**。
- C++ 的输入输出流是指由**若干字节组成的字节序列**，按顺序从一个对象传送到另一个对象。输入时，程序从输入流中**抽取**字节；输出时，程序将字节**插入**到输出流中。
- 对于面向文本的程序，每个字节代表一个字符。
- 输入流中的字节可能来自键盘，硬盘或其他程序。同样，输出流中的字节可以流向显示器、打印机、存储设备或其他程序。
- 读操作在流数据抽象中被称为（从流中）**提取**，写操作被称为（向流中）**插入**。

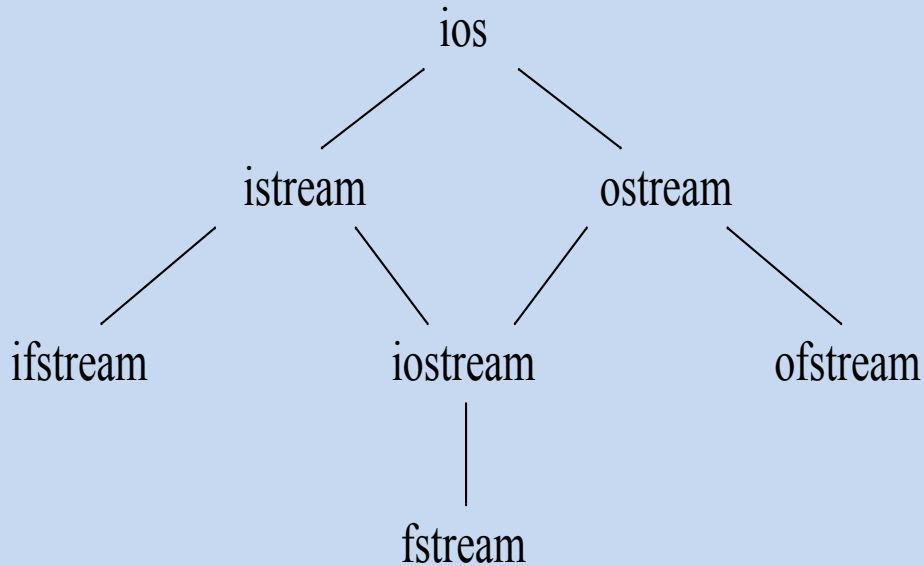
➤ **I/O 流类库**提供对象之间的数据交互服务

输出流：表示数据从内存传送到某个载体或设备中；

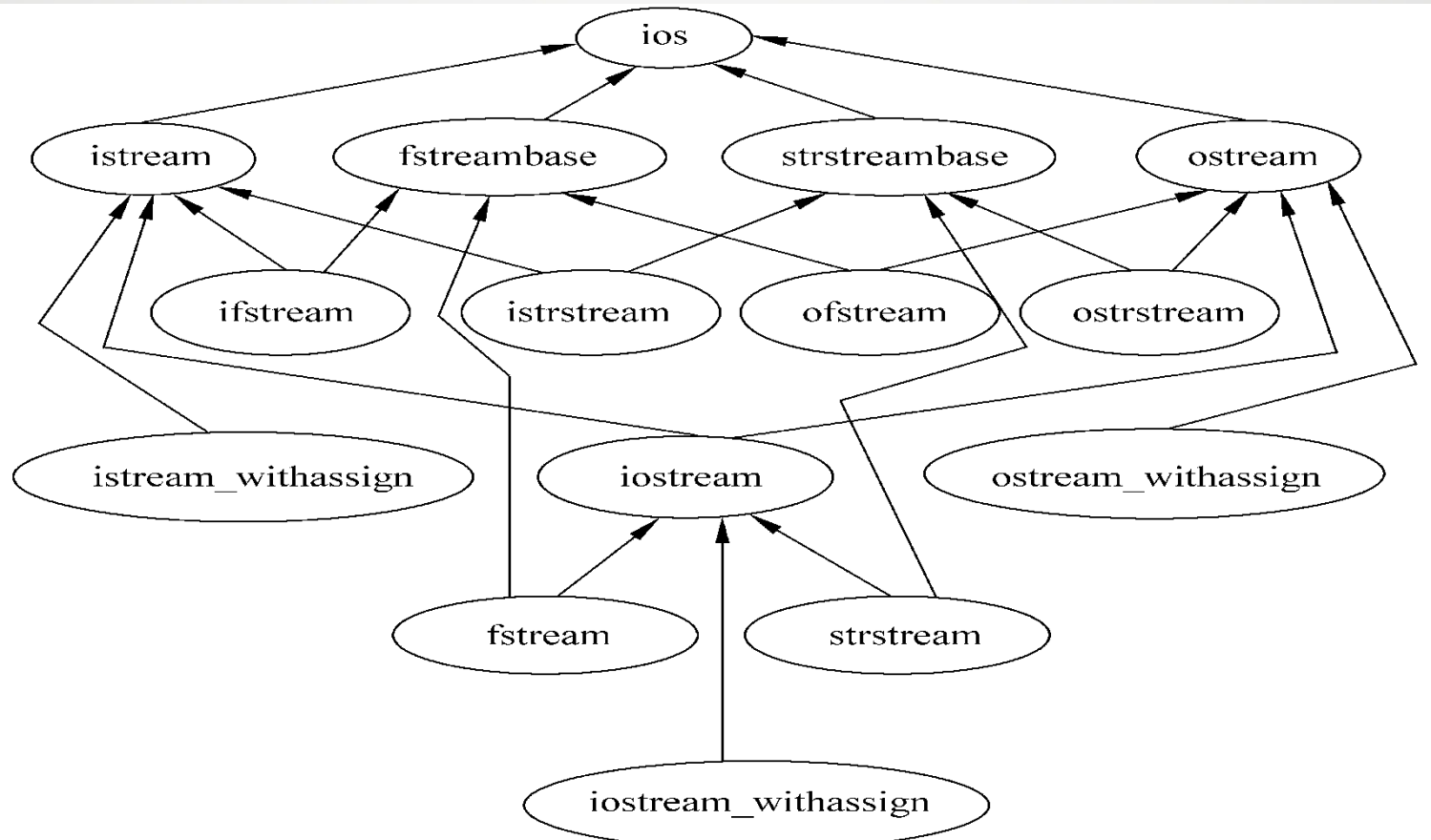
输入流：数据从某个载体或设备传送到内存缓冲区变量中；

- **流类库预定义了一批流对象**，连接常用的外部设备；
- **程序员可以定义所需的 I/O 流对象**，使用流库提供的工作方式实现数据传输；
- **流类对象可以建立和删除**，可以从流中获取数据和向流添加数据。

其中，几个常用的流类继承关系如下图所示。



I/O 类库中还有其他类：



- 我们已经利用 cin/cout 实现了数据的输入 / 输出。在程序声明 iostream 库时，
`#include <iostream>`
- cin 对象管理标准输入流，默认与标准输入设备（通常为键盘）相连； cout 对象管理标准输出流，默认与标准输出设备（通常为显示器）相连。除此之外，还有 cerr、clog、wcin、wcout、wcerr 和 wclog。
- 插入（<<）和提取（>>）运算符是为所有标准 C++ 数据类型定义的，其作用是向流中传送和读取数据。插入和提取运算符与预先定义的操作符一起工作，用来控制输入和输出格式。

- `iostream` 类库中不同的类的声明被放在不同的头文件中，头文件是程序与类库的接口，`iostream` 类库的接口分别由不同的头文件来实现。常用的有：
 - `iostream` 包含了对输入输出流进行操作所需的基本信息。
 - `fstream` 用于用户管理的文件的 I/O 操作。
 - `stringstream` 用于字符串流 I/O 。
 - `stdiostream` 用于混合使用 C 和 C++ 的 I/O 机制时。
 - `iomanip` 在使用格式化 I/O 时应包含此头文件。

➤ `iostream.h`

含有 `cin`、`cout`、`cerr`、`clog` 对象，提供无格式和格式化的 I/O

➤ `iomanip.h`

包含格式化 I/O 操纵算子，用于指定数据输入输出的格式

➤ `fstream.h`

处理文件信息，包括建立文件，读 / 写文件的各种操作接口

每一种 C++ 版本通常还包含其他一些与 I/O 相关的库，提供特定系统的某些功能

(1) 标准流

对系统指定的标准设备的 I/O 操作；

(2) 文件流

以外存中的文件为对象进行输入和输出；

以文件为对象的输入输出，包括从磁盘文件输入数据，
或将数据输出到磁盘文件；

(3) 字符串流

对内存中指定空间进行输入和输出；

通常指定一个字符数组作为存储空间；

02



标准输入流

1. 使用 `cin` 进行输入
2. 使用 `get()` 方法
3. 使用 `getline()` 方法
4. 使用 `read()` 方法

1. 使用 cin 进行输入

- C++ 提供了实用的输入功能，通过键盘产生输入的内容，从而形成字节流。cin 对象可以将输入字节流中的信息存储到相应的内存单元。通常，可以这样使用 cin:

cin >> value_holder

- 其中，>> 是流读取运算符，它重载右移位运算符 >> 来完成。>> 左边的 cin 是 istream 类的对象，右边的操作数是系统定义的任何数据类型的变量。

例如:

```
int i;  
cin>>i;
```

1. 使用 cin 进行输入

cin

- ✓ istream 类的对象，它从标准输入设备（键盘）获取数据
- ✓ 程序中的变量通过流提取符“>>”从流中提取数据。
- ✓ 在 istream 流类重载 >> 的一组公用成员函数
istream& operator >> （基本类型标识符 &）；
- ✓ 流提取符从流中提取数据时通常跳过输入流中的空格、tab 键、换行符等空白字符。

1. 使用 cin 进行输入

- 输入运算符 >> 也支持**级联输入**。在默认情况下，运算符 >> 跳过空格，读入后面与变量类型相应的值。因此给一组变量输入值时，用空格或换行将输入的数值间隔开。例如：

```
int i;  
float f;  
char w  
cin>>i>>f>>w;
```

- 当从键盘输入： 10 12.34 A

时，数值 10, 12.34 和 A 会分别存储到变量 i , f 和 w 内。

1. 使用 cin 进行输入

- 当输入字符串（char* 类型）时，输入运算符 >> 会跳过空格，读入后面的非空格符，直到遇到另外一个空格结束，并在字符串末尾自动放置字符 ‘\0’ 作为结束标志，例如：

```
char s[20];  
cin>>s;
```

- 当输入：Hello! world!

时，存储在字符串 s 中的值为 “Hello!”，而没有后面的 “world!”。

1. 使用 cin 进行输入

- 数据输入时，不仅检查数据间的空格，还做类型检查、自动匹配，例如：

```
int i;  
float f;  
cin>>i>>f;
```

- 如果输入：12.34 34.56

则存储在 i 、 f 内的数值为 12 和 0.34 ，而不是 12.34 和 34.56 。

例【 8-1 】 通过测试 cin 的真值，判断流对象是否处于正常状态

20

```
#include <iostream>
using namespace std;
int main( )
{float grade;
  cout<<"enter grade:";
  while(cin>>grade)// 能从 cin 流读取数据
  {if(grade>=85) cout<<grade<<"GOOD!"<<endl;
   if(grade<60) cout<<grade<<"fail!"<<endl;
   cout<<"enter grade:";
  }
  cout<<"The end."<<endl;
  return 0;
}
```

运行情况如下：

enter grade: 67✓

enter grade: 89✓

89 GOOD!

enter grade: 56✓

56 fail!

enter grade: 100✓

100 GOOD!

enter grade: ^Z✓// 键入文件结束符

The end.

如果某次输入的数据为

enter grade: 100/2✓

输出“ The end.”。

在不同的 C++ 系统下运行此程序，在最后的处理上有些不同。以上是在 GCC 环境下运行程序的结果，如果在 VC++ 环境下运行此程序，在键入 Ctrl+Z 时，程序运行马上结束，不输出“ The end.”。

istream 类的公有成员函数

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

- `int istream::get()`
 - 如果输入流包括附加的数据，函数取得并返回下一个字符；否则它返回 EOF。
- `istream& istream::get(char &c)`
 - 如果输入流包括附加的数据，函数取得并将下一个字符分配给 `c`；否则就是没有定义对 `c` 的作用。返回一个对 `*this`（调用对象）的引用。
- `istream& istream::get(char s[],int n,char delim='\n')`
 - 从输入流获取字符并将它们分配给 `s` 直到下面的一个条件发生：取得 `n-1` 个字符，没有输入字符了，或者下一个接收的字符的值为 `delim`。
- `istream& istream::getline(char s[],int n,char delim='\n')`
 - 从输入流获取字符并将它们分配给 `s` 直到下面的一个条件发生：取得 `n-1` 个字符，没有输入字符了，或者下一个接收的字符的值为 `delim`。
- `int istream::peek()`
 - 如果输入流包括附加的数据，函数返回下一个接收的字符；否则它返回 EOF。
- `istream& istream::unget(char c)`
 - 字符 `c` 被送到输入流。它将是下一个要接收的字符。函数返回一个对 `*this` 的引用。
- 库 `iostream` 还提供对 `ios` 成员函数的访问，一些程序在输入流中检测文件尾。
- `bool ios::eof()`
 - 如果流中达到文件尾则返回真；否则函数返回假。

2. 其他 istream 类方法

(1) get() 方法

istream 类中的 get() 方法提供不跳过空格的单字符输入功能。使用方式为：

输入流对象.get(字符型变量)

```
int get();
```

```
istream& get( char& rch );
```

```
istream& get( char* pch, int nCount, char delim = '\n' );
```


2. 其他 istream 类方法

(1) get() 方法

例如，如下循环：

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

    cin.get(ch);
    }
```

假如输入了： I can do.

按下回车键后， get(ch) 首先从输入流中读取字符 I，存储在 ch 中，使用 cout 显示它，再将 a 加 1。然后，读取 I 后面的空格字符，存储，显示，让 a 加 1。这样依次循环，直到读取到回车键，终止循环。

2. 其他 istream 类方法

(1) get() 方法

get 方法还有三种重载形式：

- ① 无参数的，
- ② 有两个参数
- ③ 有三个参数的。

① 无参数的

无参数的 get() 方法用于从指定的输入流中提取一个字符（包括空格），函数的返回值为读入的字符。例如：

```
char ch;  
ch=cin.get();
```

2. 其他 istream 类方法

(1) get() 方法

② 有两个参数的

- 有两个参数的 get() 方法，其原型如下：

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

- 其中，第一个参数用于放置字符串的内存单元的地址。第二个参数为读取的最大字符数（额外的一个字符用于存储结尾的空字符，因此只能读取最大字符数 -1 个字符）。例如：

```
char line[50];  
cin.get(line, 50);
```

- cin.get() 函数将在到达第 49 个字符或遇到换行符后停止将输入读取到数组中。
-

2. 其他 istream 类方法

(1) get() 方法

③ 有三个参数的

- 有三个参数的 get() 方法，其原型如下：

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

- 其中，前两个参数与上面的相同，第三个参数指定用作分界符的字符。只有两个参数的 get() 函数将换行符用作分界符。例如：

```
char line[50];
```

```
cin.get(line, 50, ' #' );
```

- 假如输入了 Please give me #3 apples.

由于 get() 函数将字符 ‘#’ 为分界符，所以储存到 line 数组只有 Please give me 。



例【 8-2 】 用 get 函数读入字符

29

```
#include <iostream>

int main( )
{int c;
  cout<<"enter a sentence:"<<endl;
  while((c=cin.get())!=EOF)
    cout.put(c);
  return 0;
}
```

运行情况如下：

enter a sentence:

I study C++ very hard.↵ (输入一行字符)

I study C++ very hard. (输出该行字符)

^Z↵ (程序结束)

C 语言中的 getchar 函数与流成员函数 cin.get() 的功能相同， C++ 保留了 C 的这种用法。

2. 其他 istream 类方法

(2) getline() 方法

- istream 类中的 getline() 方法可以读取整行输入，而不是一个字符。使用方法为：
输入流对象.getline(字符指针, 字符个数)
- 字符指针用来放置输入字符串的内存单元的地址。字符个数用来限制读取的最大字符数。由于存储字符串额外需要存储一个结尾的空字符，读取的最大字符数为字符个数 -1 。

2. 其他 istream 类方法

(2) getline() 方法

- 例如，输入不超过 5 个字符的内容，存储到 ch 数组中，并将 ch 显示。

```
char ch[10];
```

```
cout<<"Please enter less than five characters:";
```

```
cin.getline(ch,5);
```

```
cout<<ch<<endl;
```

- 假如输入了： 123456789 <Enter>

按下回车键后，显示为 1234 。由于 getline(ch,5) 中的第二个参数限制读取的字符数为 4 ，所以只能读取输入流中的前四个字符，存储到 ch 数组中并显示。

2. 其他 istream 类方法

(2) getline() 方法

getline() 重载方法同样也有三个参数的方法，三个参数的作用和上面 get() 方法类似，其原型如下：

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



```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
  cout<<"enter a sentence:"<<endl;
  cin>>ch;
  cout<<"The string read with cin is:"<<ch<<endl;
  cin.getline(ch,20,'/'); // 读 19 个字符或遇'/' 结束
  cout<<"The second part is:"<<ch<<endl;
  cin.getline(ch,20);      // 读 19 个字符或遇'/n' 结束
  cout<<"The third part is:"<<ch<<endl;
  return 0;
}
```



例【 8-3 】 用 getline 函数读入一行字

34

符
程序运行情况如下：

enter a sentence: I like C++./I study C++./I am happy.↵

The string read with cin is:I

The second part is: like C++.

The third part is:I study C++./I am h

2. 其他 istream 类方法

(3) read() 方法

istream 类中的 read() 方法读取指定数目的字节，并将它们存储在指定的位置中。

例如，下面的语句从标准输入流中读取 25 个字符，并将它们存储在数组 a 中：

```
char a[50];
```

```
cin.read(a,25);
```

2. 其他 istream 类方法

(3) read() 方法

与 `getline()` 和 `get()` 不同的是，`read()` 不会在输入后加上空值字符，因此不能将输入转换为字符串。该方法的返回类型为 `istream &`，因此可以像下面拼接起来：

```
char a[50];  
char b[100]  
cin.read(a, 50).read(b, 100);
```

03



标准输出流

1. 使用 `cout` 进行输出
2. 使用 `cout` 进行格式化输出
3. 使用 `put()` 和 `putchar()` 方法
4. 使用 `write()` 方法

1. 使用 cout 进行输出

出

- cout 是输出流类 ostream 的对象，输出结果流向标准的输出设备显示器。在 C++ 中，流输出使用插入运算符 <<(重载左移位运算符) 完成输出，使之能够识别 C++ 中所有的基本类型。插入运算符 << 左边的操作数是 ostream 类的一个对象 (如 cout)，右边可以是 C++ 的合法表达式。

1. 使用 cout 进行输出

- cout

- ✓ ostream 类的对象（console ou
通常连向显示器，可以重定向（大软盘文件）

流插入运算符

- ✓ 在输出流类中重载 << 的一组公用成员函数

ostream& operator << （类型标识符）；

- ✓ cout 流在内存中对应开辟了一个缓冲区。

1. 使用 cout 进行输

```

#include <stdafx.h>
#include <iostream>
using namespace std;
void main()
{
    int a=22;
    char b='B';
    float c=1.25;
    double d=3.1415926;
    cout<<"a="<<a<<"    b="<<b<<endl;
    cout<<"c="<<c<<"    d="<<d<<endl;
}

```

```

a=22    b=B
c=1.25   d=3.14159

```

1. 使用 `cout` 进行输

出

C++ 用指向字符串存储位置的指针来表示字符串。指针的形式可以是 `char` 数组名、显式的 `char` 指针或用引号括起的字符串。C++ 还允许输出项为显式对象的地址。默认情况下，地址以十六进制的形式显示。但对于其他类型的指针，C++ 可以使用 `void *` 来强制转换输出。

1. 使用 cout 进行输

```
#include "stdafx.h"
#include "iostream"
using namespace std;
void main()
{
    int a=12;
    char b[20]="Hello world!";
    char *c= b;
    cout<<"Hi. \n";
    cout<<b<<endl;
    cout<<c<<endl;
    cout<<&b<<endl;
    cout<<(void *)c<<endl;
    cout<<&c<<endl;
}
```

A screenshot of a terminal window showing the output of the C++ program. The output consists of seven lines: "Hi.", "Hello world!", "Hello world!", "0021FBCC", "0021FBCC", "0021FBC0", and "0021FBC0". The text is displayed in a white, monospaced font on a black background.

```
Hi.
Hello world!
Hello world!
0021FBCC
0021FBCC
0021FBC0
0021FBC0
```

1. 使用 cout 进行输出

(1) 上面的 cout 代码可以合并成一行来执行:

```
cout<<"Hi. \n"<<b<<endl<<c<<endl<<&b<<endl<<(void *)c<<endl<<&c<<endl;
```

- 这种级联的形式在 C++ 中是允许，因为重载的 << 运算符返回对它左边操作数对象的引用（即 cout），当执行完 cout<<" Hi. \n" 后，输出 Hi.，并返回 cout 对象，则该语句变为：

```
cout<<b<<endl<<c<<endl<<&b<<endl<<(void *)c<<endl<<&c<<endl;
```

- 这样，依次显示，并返回 cout，直到执行完毕

1. 使用 cout 进行输

出 注意:

(2) 利用 << 输出的时候需要注意优先级的问题, 例如求两者中的最大值问题:

```
int i=10, j=20;  
cout<<"the max is "  
cout<<(i>j)?i:j;
```

- 程序的输出结果为: the max is 0 .

2. 其他 ostream 类方法

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

- ostream& ostream::put(char c)
将字符 c 插入输入流中。函数返回一个对 *this 的引用。
- ostream& ostream::write(const char s[],int n)
将 s 中的 n 个字符插入到输入流中。空字符也是有效的。
函数返回一个对 *this 的引用。
- ostream& ostream::flush()
强制任何没有完成的操作符插入完成。函数返回一个对 *this 的引用。

2. 其他 ostream 类方法

(1) put() 方法

- ostream 类中 put() 方法用于输出一个字符，其原型如下：

```
ostream & put(char);
```

- 可以用类方法表示法来调用它：

```
cout.put('A');
```

- 其中，cout 是调用方法的对象，put() 是类成员函数。和 << 运算符函数一样，该函数也返回一个指向调用对象的引用，因此实现拼接输出：

```
cout.put('O').put('K');
```



```
#include <iostream>
using namespace std;
int main( )
{char *a="BASIC";// 字符指针指向' B'
  for(int i=4;i>=0;i--)
    cout.put(*(a+i));      // 从最后一个字符开始输出
  cout.put('\n');
  return 0;
}
```

运行时在屏幕上输出：

CISAB

还可以用 putchar 函数输出一个字符。putchar 函数是 C 语言中使用的，在 stdio.h 头文件中定义。C++ 保留了这个函数，在 iostream 头文件中定义。

例也可以改用 putchar 函数实现。

#include <iostream> // 也可以用 #include <stdio.h> , 同时不要下一行

using namespace std;

int main()

{char *a="BASIC";

for(int i=4;i>=0;i--)

putchar(*(a+i));

putchar('\n');

}

运行结果与前相同。

成员函数 put 不仅可以用 cout 流对象来调用, 而且也可以用 ostream 类的其他流对象调用。

2. 其他 ostream 类方法

(2) write() 方法

- ostream 类中 write() 方法用于显示字符串，其原型：
ostream & write (const char* s , streamsize n);
- write() 的第一个参数是指向 char 型的指针，第二个参数指出显示字符的数量。write() 方法不会在遇到空字符时自动停止输出字符，而会按照指定数量输出字符，即使超出了字符串的边界。如果超出了字符串的边界，程序会将字符串在内存中存储位置后面数据输出。

```
#include <iostream>
using namespace std;
int main( )
{char *a="BASIC";// 字符指针指向' B'
  for(int i=4;i>=0;i--)
    cout.write(*(a+i));      // 从最后一个字符开始输出
  cout. write('\n');
  return 0;
}
```

运行时在屏幕上输出：

CISAB

04



格式化控制

- 插入（<<）和提取（>>）运算符是为所有标准 C++ 数据类型定义的，其作用是向流中传送和读取数据。插入和提取运算符与预先定义的操作符一起工作，用来控制输入和输出格式。

1. 输出宽度

- 为了调整输出时的显示宽度，可以通过调用 `width` 成员函数为每个项（item）指定输出宽度或在流中放入 `setw` 操纵符。

2. 对齐方式

- 输出流的默认对齐方式为文本右对齐，程序中可以用 `setiosflags` 和 `resetiosflags` 操作符重设对齐方式。

3. 精度

- 使用 `setprecision` 操作符改变精度，该操作符有两个标志，`ios::fixed` 和 `ios::scientific`。

4. 进制

- 可以用 `dec`、`oct` 和 `hex` 操纵符设置输入和输出的默认进制。

- 格式化标志是类定义的**枚举集合**，指定输入输出格式化和操作的不同选择。
- 该枚举类型定义：

enum

{skipws, left, right, internal, dec, oct, hex, showbase, showpoint, uppercase, showpos, scientific, fixed, unitbuf, stdio };

- 引用格式化标志：

ios:: 格式化标志



1. 数据流的格式控制：设置标志字

57

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

ios 控制格式的函数

函数	功能
long flags(long <i>lFlags</i>); long flags() const;	用参数 <i>lFlags</i> 更新标志字 返回标志字
long setf(long <i>lFlags</i>);	设置 <i>lFlags</i> 指定的标志位
long unsetf(long <i>lFlags</i>);	将参数 <i>lMask</i> 指定的标志位清 0
int width(int <i>nw</i>);	设置下一个输出项的显示宽度为 <i>nw</i>
char fill(char <i>cFill</i>);	空白位置以字符参数 <i>cFill</i> 填充
int4_t ... (...)	

```
#include <iostream.h>

void main()
{
    char *s = "Hello";
    cout.fill( '*' );           // 置填充符
    cout.width( 10 );           // 置输出宽度
    cout.setf( ios :: left );   // 左对齐
    cout << s << endl ;

    cout.width( 15 );           // 置输出宽度
    cout.setf( ios :: right, ios :: left ); // 清除左对齐标志位，置右对齐
    cout << s << endl ;
}
```

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    double x = 22.0/7 ;
```

```
    int i ;
```

```
    cout << "output in fixed :\n" ;
```

```
    cout.setf( ios::fixed | ios::showpos ) ;           // 定点输出, 显示 +
```

```
    for( i=1; i<=5; i++ )
```

```
    { cout.precision( i ) ; cout << x << endl ; }
```

```
    cout << "output in scientific :\n" ;
```

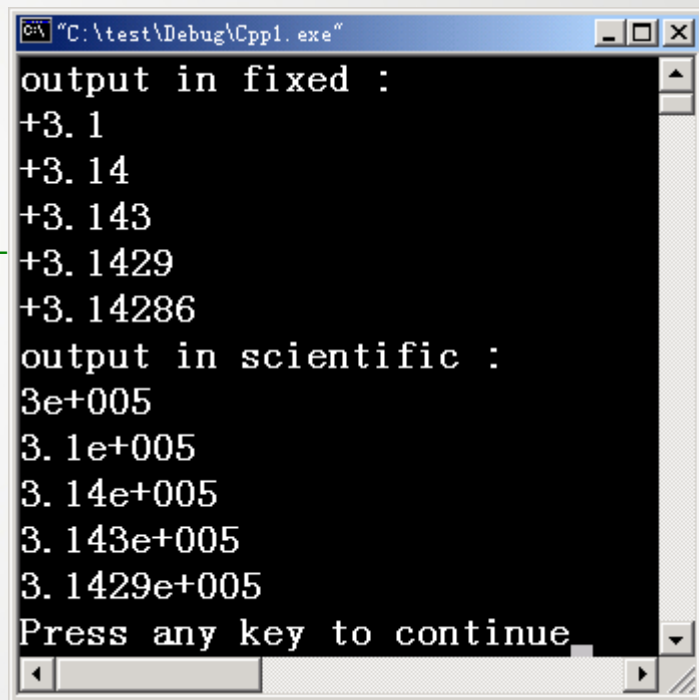
```
    // 清除原有设置, 科学示数法输出
```

```
    cout.setf(ios::scientific, ios::fixed|ios::showpos ) ;
```

```
    for( i=1; i<=5; i++ )
```

```
    { cout.precision(i) ; cout << x*1e5 << endl ; }
```

```
}
```



```

C:\test\Debug\Cppl.exe
output in fixed :
+3.1
+3.14
+3.143
+3.1429
+3.14286
output in scientific :
3e+005
3.1e+005
3.14e+005
3.143e+005
3.1429e+005
Press any key to continue
  
```

➤ 控制符是 istream 和 ostream 类定义了一批函数，作为重载插入运算符 << 或提取运算符 >> 的右操作数 控制 I/O 格式； **iostream 几个常用的控制符：**

控制符	功能	输入 / 输出
endl	输出一个新行符，并清空流	O
ends	输出一个空格符，并清空流	O
flush	清空流缓冲区	O
dec	用十进制表示法输入或输出数值	I/O
hex	用十六进制表示法输入或输出数值	I/O
oct	用八进制表示法输入或输出数值	I/O

```
#include <iostream.h>
```

```
void main()
```

```
{ int a , b , c ; cout << "please input a in decimal: " ;
```

```
    cin >> dec >> a ;
```

// 置十进制输入

```
    cout << "please input b in hexadecimal: " ;
```

```
    cin >> hex >> b ;
```

// 置十六进制输入

```
    cout << "please input c in octal: " ;
```

```
    cin >> oct >> c ;
```

// 置八进制输入

```
    cout << "Output in decimal :\n" ;
```

```
    cout <<"a = "<<a<<" b = "<<b<<" c = "<<c<<endl ;
```

// 置十进制输出

```
    cout.setf( ios :: hex , ios :: basefield ) ;
```

```
    cout << "Output in hexadecimal :\n" ;
```

```
    cout <<hex<<"a = "<<a<<" b = "<<b<<" c = "<<c<<endl ;
```

// 置十六进制输出

```
    cout.setf( ios :: oct , ios :: basefield ) ;
```

```
    cout << "Output in octal :\n" ;
```

```
    cout <<oct<<"a = "<<a<<" b = "<<b<<" c = "<<c<<endl ;
```

// 置八进制输出

```
}
```

iomanip 的控制符

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

控制符	成员函数	作用
setfill(c)	fill(c)	设置填充字符为字符常量或字符变量 c
setprecision(n)	precision(n)	设置显示小数的精度为 n 位
setw(n)	width(n)	设置域宽为 n 个字符
setbase(n)	setf ()	设置整数的基数为 n(n=8,10,16)
setiosflags()	setf ()	设置输出格式的状态
resetiosflags()	unsetf()	终止已设置输出格式的状态

// 整数的格式化输出

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
void main()
```

```
{
```

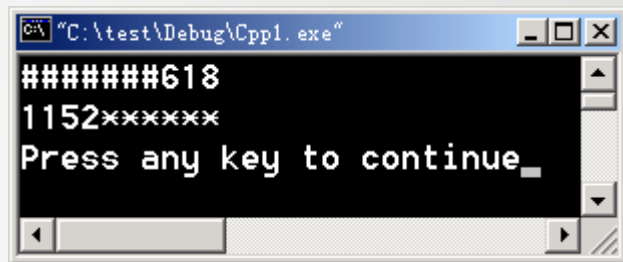
```
    const int k = 618;
```

```
    cout << setw(10) << setfill('#') << setiosflags(ios::right) << k << endl;
```

```
    cout << setw(10) << setbase(8) << setfill('*')
```

```
        << resetiosflags(ios::right) << setiosflags(ios::left) << k << endl;
```

```
}
```



```
#include <iostream.h>
```

```
void main()
```

```
{ double x = 22.0/7 ;
```

```
int i ;
```

```
cout << "output in fixed :\n" ;
```

```
cout << setiosflags( ios::fixed | ios::showpos ) ;
```

// 定点输出 ,

```
for( i=1; i<=5; i++ )
```

```
{ cout << setprecision(i) <<  
x<<endl ; }
```

```
cout << "output in scientific :\n" ;
```

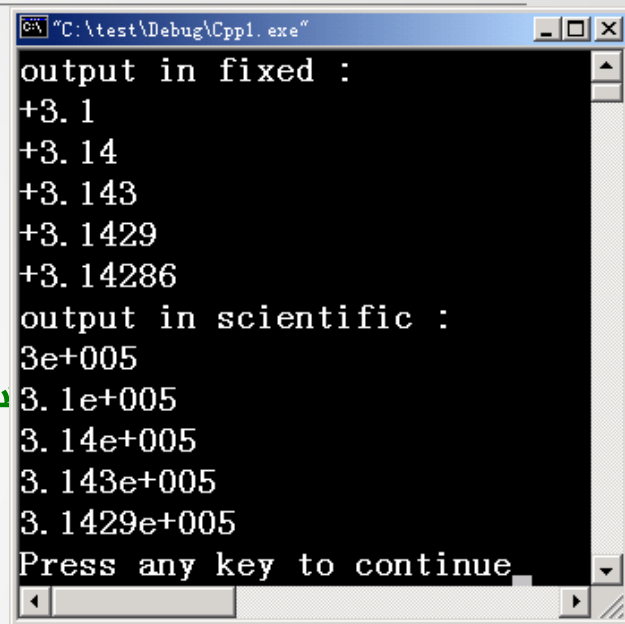
// 清除原有设置 , 科学示数法输出

```
cout << resetiosflags( ios::fixed | ios::showpos )<< setiosflags( ios::scientific ) ;
```

```
for( i=1; i<=5; i++ )
```

```
cout<<setprecision(i)<<x*1e5<<endl ;
```

```
}
```



```
C:\test\Debug\Cppl.exe
output in fixed :
+3.1
+3.14
+3.143
+3.1429
+3.14286
output in scientific :
3e+005
3.1e+005
3.14e+005
3.143e+005
3.1429e+005
Press any key to continue
```

```
#include <iostream>
using namespace std;
void main()
{
    double values[] = {1.23,35.36,653.7,4358.24};
    for(int i=0;i<4;i++)
    {
        cout.width(10);
        cout << values[i] <<'\n';
    }
}
```

输出结果：

1.23

35.36

653.7

4358.24

```
#include <iostream>
using namespace std;
void main()
{
    double values[]={1.23,35.36,653.7,4358.24};
    for(int i=0; i<4; i++)
    {
        cout.width(10);
        cout.fill('*');
        cout<<values[i]<<"\n";
    }
}
```

输出结果：

*****1.23

*****35.36

*****653.7

***4358.24

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    double values[]={1.23,35.36,653.7,4358.24};
    char *names[]{"Zoot","Jimmy","Al","Stan"};
    for(int i=0;i<4;i++)
        cout<<setw(6)<<names[i]
            <<setw(10)<<values[i]<<endl;
}
```

输出结果：

Zoot 1.23

Jimmy 35.36

Al 653.7

Stan 4358.24

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    double values[]={1.23,35.36,653.7,4358.24};
    char *names[]{"Zoot","Jimmy","Al","Stan"};
    for(int i=0;i<4;i++)
        cout<<setiosflags(ios::left)
            <<setw(6)<<names[i]
            <<resetiosflags(ios::left)
            <<setw(10)<<values[i]<<endl;
}
```

输出结果：

Zoot	1.23
Jimmy	35.36
Al	653.7
Stan	4358.24

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    double values[]={1.23,35.36,653.7,4358.24};
    char *names[]{"Zoot","Jimmy","Al","Stan"};
    cout<<setiosflags(ios::scientific);
    for(int i=0;i<4;i++)
        cout<<setiosflags(ios::left)
            <<setw(6)<<names[i]
            <<resetiosflags(ios::left)
            <<setw(10)<<setprecision(1)
            << values[i]<<endl;
}
```

输出结果：

Zoot	1
Jimmy	4e+001
Al	7e+002
Stan	4e+003

```
#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;           // 以十进制形式输出整数
  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";
  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;
  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：

input a:34↵(输入 a 的值)

dec:34 (十进制形式)

hex:22 (十六进制形式)

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 (指数形式输出, 4 位小数)

pi=3.143 (小数形式输出, 精度仍为 4)

```
#include <iostream>
using namespace std;
int main( )
{int a=21
cout.setf(ios::showbase);// 显示基数符号 (0x 或 0)
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);
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;       // 输出 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);   // 正数输出“+”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:0x15 (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 (小数形式输出, 精度为 6 , 最左侧输出数符“+”)

04



标准错误输出流



. 标准错误输出流

- cerr (无缓冲标准错误输出流)

cerr 与 cout 的差别在于:

- 1) cerr 不能重定向, 只能输出到显示器;
- 2) cerr 不能被缓冲, 直接输出到显示器

```
cerr << "Error" << "\n";
```

- clog (有缓冲标准错误输出流)

clog 与 cerr 区别:

clog 能被缓冲, 缓冲区满时输出。



流错误状态

所有流都把流的状态存储在状态字中，不同标志位中存储不同的错误状态位；
包含在类 ios 的 enum 成员中

标识常量	值	意义
ios:: goodbit	0x00	状态正常
ios:: eofbit	0x01	文件结束符，当文件结尾时设置该标志
ios:: failbit	0x02	I/O 操作失败，数据未丢失，可以恢复
ios:: badbit	0x04	数据丢失，无法恢复



ios 处理流错误状态的公有成员函数

函数	功能
int eof() const;	返回 eofbit 状态值。文件结束符时返回 1，否则返回 0
int fail() const;	返回 failbit 状态值
int bad() const;	返回 badbit 状态
int good() const;	eofbit、failbit 和 badbit 都没有被设置，则返回 1
int rdstate() const;	返回状态字
void clear(int <i>n</i> = 0);	恢复或设置状态字



流 `cerr` 和 `clog` 把显示输出到默认错误日志上，该日志通常是控制台显示器。

- 对流 `cerr` 的每个插入请求应该立即送出显示。下面给出几个错误消息的例子：
 - `cerr<<" 系统将在 10 秒后重起! \n";`
 - `cerr<<" 输入值无效，请重新输入! \n";`



- 如果出于效率的缘故，则有必要缓冲错误和系统状态消息，这样就应该使用带缓冲的错误流 `clog`。像 `cerr` 一样，`clog` 通常是定向到显示器。

下面给出几个样例：

- `clog<<UserName<<" 成功登陆! \n";`
- `clog<<" 有新邮件! \n";`

04



文件的输入和输出

- C++ 在进行文件操作时，必须首先建立一个文件流，并把这个流与实际的文件相关联，然后就可以按照要求进行读写操作。C++ 的文件流实际上就是以外存文件为输入输出对象的数据流。输入文件流是指从外存文件流向内存的过程，输出文件流是指从内存流向外存的过程。
- C++ 将文件流分为 3 类：
 - (1) ifstream 流类，是从 istream 类派生的，用于文件的输入操作；
 - (2) ofstream 流类，是从 ostream 类派生的，用于文件的输出操作；
 - (3) fstream 流类，是从 iostream 类派生的，用于文件的输入和输出操作。



- C++ 的文件 I/O 模式分为两种，一种为格式化文字模式，另一种为二进制模式，默认的文件 I/O 模式为文字模式。
- 当使用格式化文字模式时，输出至文件的内容将被储存为字符，因此，格式化文字模式适合储存字符或字符串。
- 如果以二进制的方式处理数字，不论是储存的方式，还是占有文件空间的方式都与其储存在内存中的方式相同，因此，在储存数字时使用二进制模式是比较合适的。



- 文件的最小单位为**字符**，由字符组成一个**字段**，好几个字段组成一个**记录**，而一个文件则由好几笔记录组成。
- 将数据输入输出到文件中，除了可以使用 **write** 和 **read** 函数外，还可以使用“<<”和“>>”运算符，而所有前面提及的格式控制方法，均可以在文件 I/O 中使用。
- 在二进制模式下，文件的 I/O 将利用 **write** 和 **read** 函数将所有数据以字节形式储存至文件里。
- 对于原本就是字符的数据，并不需要转型，即可读取 / 写入到文件里；但对于数字型的数据，则必须转为字符指针，然后传入 **write** 和 **read** 函数。



◆ 文件处理函数

函 数	功 能 说 明
<code>open(filename,mode)</code>	以 mode 模式打开名为 filename 的文件
<code>close()</code>	关闭文件
<code>is_open()</code>	检查文件是否为打开状态，是则返回真，否则返回假
<code>write(str,size)</code>	将 str 数组中 size 个字符写入到文件中
<code>read(str,size)</code>	从文件中读取数据至文件结尾，并设定给 str 数组，但至多不超过 size 个字符



1. 文件的打开与关闭

(1) 打开文件

`fstream` 类可以用于将数据写入文件，或读取文件的数据。要使用 `fstream` 类执行文件的 I/O 时，首先必须先定义一个 `fstream` 类的对象。例如：

```
fstream file; // 定义一个 fstream 对象
```

文件在进行读写操作前，应先打开，其目的是为文件流对象和特定的外存文件建立关联，并指定文件的操作方式。打开文件的方式有以下两种。

- ① 使用 `open()` 函数。
- ② 使用构造函数。



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

`open` 函数是 `ifstream`、`ofstream` 和 `fstream` 类的成员函数。文件打开方式的一般格式为：

文件流对象名 . `open`(" 文件名 " , 打开模式) ;



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

◆ 表 文件流的文件打开模式

模式参数	功 能
<code>ios::in</code>	文件打开为读取（输入）状态，此为 <code>istream</code> 类的默认模式参数
<code>ios::out</code>	文件打开为写入（输出）状态，此为 <code>ostream</code> 类的默认模式参数
<code>ios::ate</code>	打开一个现存文件，从文件结尾处读取（输入）或写入（输出）
<code>ios::app</code>	打开一个输出文件从文件结尾写入（输出）数据
<code>ios::trunc</code>	打开一个文件，如果它已经存在，就删除其中原有的内容
<code>ios::nocreate</code>	如果一个文件存在则打开它，否则该操作失败
<code>ios::noreplace</code>	如果一个文件不存在则作为新文件打开它；如果文件已存在，则该操作失败
<code>ios::binary</code>	以二进制模式打开一个文件，默认是文本模式



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

- a) 每个被打开的文件都有一个文件指针，该指针的初始位置由打开方式指定。文件每次读写都从文件指针的当前位置开始。当读出或写入一个字符，指针自动后移一个字节。当文件指针指向文件尾时，将遇到文件结束符 EOF（文件结束符占一个字节，其值为 -1）。此时，流对象的成员函数 `eof()` 的值为非 0 值（一般为 1），表示文件结束
- 文件指针有两种，一个是写入指针，另一个是读取指针。当文件被打开时，这两个指针都将被设定指向文件的起始处。
 - 在 `ios` 类里，定义了三个特定的文件指针。通过指针函数与特定的指针的配合使用，可供在处理文件 I/O 时移动文件指针之用。这三个位移指针分别是 `ios::beg`（文件开头）、`ios::end`（文件结尾）、`ios::cur`（当前的指针位置）。



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

说明：

- b) 用“`ios::in`”方式打开文件只能用于从文件向计算机输入，而不能用于向该文件输出数据，而且该文件必须已经存在。如果用“`ios::in`”打开一个不存在的文件，将会出错。如果用类 `ifstream` 产生的流，将隐含为输入流，默认为“`ios::in`”，可以不必显式地声明打开方式。

例如：

```
ifstream fin;  
fin.open("abc.txt");
```



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

说明：

- c) 用“`ios::out`”方式打开文件，表示计算机向该文件输出数据。如果用类 `ofstream` 产生的流，将隐含为输出流，默认为“`ios::out|ios::trunc`”，可以不必显式地声明打开方式。以这种方式打开文件进行输出时，如果没有这样的文件，将创建一个新文件；如果有这样的文件，则打开文件并清空文件，输出将进入一个空文件中。

例如：

```
ofstream fout;  
fout.open("abc.txt");
```



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

说明：

- d) `fstream` 类不提供默认的模式值，所以使用 `fstream` 类创建对象时，必须显式地提供模式。
- e) 如果希望不删除文件原来数据，向文件末尾添加新数据，则应当用 “`ios::app`” 方式打开文件。使用 “`ios::app`” 方式，文件必须存在，而且只能用于输出。
- f) 用 “`ios::ate`” 方式打开一个已存在的文件，文件指针自动位于原有文件的尾部。



1. 文件的打开与关闭

(1) 打开文件

① 使用 `open()` 函数：

说明：

- g) 在默认情况下，打开的文件均以文本方式打开文件。在用文本文件向计算机输入时，把回车和换行两个字符转换为一个换行符，而在输出时把换行符转换为回车和换行两个字符。若需要以二进制方式打开文件，则需要将打开方式设置为“`ios::binary`”。用二进制方式时，在内存中的数据形式与输出到外部文件中的数据形式完全一致。
- h) 打开方式可以用位运算符“`|`”将两个或多个位合并成一个组合。例如，“`ios::in|ios::binary`”表示打开的文件可以进行二进制的读入。



1. 文件的打开与关闭

(1) 打开文件

② 使用构造函数:

使用构造函数同样也可以打开文件，与 `open()` 函数实现的功能一样。由于不同的输入输出类，其使用的格式分别为：

`ifstream` 对象名 (" 文件名 ", " 打开方式 ");

`ofstream` 对象名 (" 文件名 ", " 打开方式 ");

`fstream` 对象名 (" 文件名 ", " 打开方式 ");



1. 文件的打开与关闭

(1) 打开文件

② 使用构造函数：

说明：

- a) 使用 `ifstream` 和 `ofstream` 类的构造函数打开文件，可以省略第二个参数“打开模式”。在默认情况下，`ifstream` 的打开模式为“`ios::in`”，`ofstream` 的打开模式为“`ios::out|ios::trunc`”。



1. 文件的打开与关闭

(1) 打开文件

② 使用构造函数:

说明:

- b) 只有在成功打开文件后，才能对文件进行读写操作。如果由于某些原因打不开文件（即执行函数 `open()` 失败），则流变量的值为 0。为了确保成功打开文件，可以通过下面的方法进行检测。

```
ofstream fout("abc.txt");  
if(!fout)  
{  
    cout<<"Cannot open file!\n"; // 错误处理代码  
}
```



1. 文件的打开与关闭

(2) 关闭文件

当对一个文件的读写操作完成后，为了保证数据安全，切断文件与流的联系，应及时关闭文件。关闭文件的一般格式为：

流对象名.close()

注意：关闭这样的连接并不会删除流，而只是断开流与文件的连接。而流对象还仍然存在，并可以重新连接到同一个文件或另一个文件。



2. 文本文件的读写操作

- ① 用流输入运算符“>>”和流输出运算符“<<”输入输出标准类型的数据
在对文件的操作中，可以通过对文件流对象和运算符“>>”和“<<”实现对文件的读写，如同用 `cin`、`cout` 和 `>>`、`<<` 对标准设备进行读写一样。



2. 文本文件的读写操作

- ① 用流输入运算符“>>”和流输出运算符“<<”输入输出标准类型的数据
在对文件的操作中，可以通过对文件流对象和运算符“>>”和“<<”实现对文件的读写，如同用 `cin`、`cout` 和 `>>`、`<<` 对标准设备进行读写一样。



2. 文本文件的读写操作

② 用 put、get 和 getlline 成员函数进行字符的输入输出

由于 ifstream、ofstream 和 fstream 类继承了 istream、ostream 和 iostream 类的 put、get 和 getlline 成员函数，因此可以使用 put、get 和 getlline 成员函数进行字符的输入输出。



3. 二进制文件的读写操作

- 二进制文件不同于文本文件以 ASCII 代码存放数据，它将内存中数据存储形式不加转换地传送到文件中。对于字符来说，二进制表示与文本表示是一样的，即字符的 ASCII 码的二进制表示。但对于数字来说，由于不需要转换，用二进制格式保存数字速度更快，占用空间更小，并可以大块地存储数据。
- 对二进制文件的读写，主要用 `istream` 类的 `read()` 方法和 `ostream` 类的 `write()` 方法。



4. 使用文件指针成员函数实现随机存取

随机存取指在访问文件中的元素时，不必考虑各个元素的排列次序或位置，根据需要直接访问文件中任一个元素。为了进行随机存取，必须先确定文件指针的位置。



4. 使用文件指针成员函数实现随机存取

文件流提供了常用的文件指针成员函数如下表所示：

文件操作方式	功能
seekg(位置)	将输入位置指针移动到指定位置
seekg(位移量, 参考位置)	以参照位置为基础移动指定的位移量
seekp(位置)	将输出位置指针移动到指定位置
seekp(位移量, 参考位置)	以参照位置为基础移动指定的位移量
tellg()	返回输入文件指针的当前位置
tellp()	返回输出文件指针的当前位置



4. 使用文件指针成员函数实现随机存取

参数中的位置和位移量均为长整型，以字节为单位。“参照位置”可以是：

<code>ios::beg</code>	// 表示文件头，为默认值
<code>ios::cur</code>	// 当前位置
<code>ios::end</code>	// 文件尾



4. 使用文件指针成员函数实现随机存取

例如: fin 是一个 ifstream 的对象:

```
fin.seekg(10); // 把输入位置指针移动到离文件头 10 个字节处
```

```
fin.seekg(10, ios::beg); // 把输入位置指针移动到离文件头 10 个字节处
```

```
fin.seekg(5, ios::cur); // 把输入位置指针移动到当前位置后 5 个字节处
```

```
fin.seekg(-20, ios::end); // 把输入位置指针向前移动到离文件尾 20 个字节处
```





文件的输入和输出 [程序分析]

108

例 有一个整型数组，含 10 个元素，从键盘输入 10 个整数给数组，将此数组送到磁盘文件中存放。

```
#include <fstream>
using namespace std;
int main( )
{int a[10];
  ofstream outfile("f1.dat",ios::out);// 定义文件流对象，打开磁盘文件" f1.dat"
  if(!outfile)           // 如果打开失败， outfile 返回 0 值
  {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]<<" ";      // 向磁盘文件" f1.dat" 输出数据
  }
  outfile.close();          // 关闭磁盘文件" f1.dat"
  return 0;
}
```



析]
运行情况如下：

enter 10 integer numbers:

1 3 5 2 4 6 10 8 7 9 ↵

请注意：在向磁盘文件输出一个数据后，要输出一个（或几个）空格或换行符，以作为数据间的分隔，否则以后从磁盘文件读数据时，10个整数的数字连成一片无法区分。



文件的输入和输出 [程序分

110

析]
例 13. 设置位置指针

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {
    char ch;
    ifstream tfile("payroll",ios::binary|ios::nocreate);
    if(tfile)
    { tfile.seekg(8);
      while(tfile.good())
      { tfile.get(ch);
        if (!ch) break; cout<<ch; }
    }
    else
    { cout<<"ERROR: Cannot open file 'payroll'."<<endl; }
    tfile.close();
}
```



文件的输入和输出 [程序分

111

析]
例 14. 文件读二进制记录

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
void main()
{
    struct
    { double salary;
      char name[23];
    } employee;
    ifstream is("payroll",ios::binary|ios::nocreate);
    if (is)
    { is.read((char *) &employee,sizeof(employee));
      cout<<employee.name<<' '<<employee.salary<<endl;
    }
    else
    { cout<<"ERROR: Cannot open file 'payroll'."<<endl;}
    is.close();
}
```



文件的输入和输出 [程序分

112

析]
例

：从上例建立的数据文件 f1.dat 中读入 10 个整数放在数组中，找出并输出 10 个数中的最大者和它在数组中的序号。

```
#include <fstream>
int main( )
{int a[10],max,i,order;
 ifstream infile("f1.dat",ios::in|ios::nocreate);
// 定义输入文件流对象，以输入方式打开磁盘文件 f1.dat
 if(!infile)
 {cerr<<"open error!"<<endl;
  exit(1);
 }
 for(i=0;i<10;i++)
 {infile>>a[i];// 从磁盘文件读入 10 个整数，顺序存放在 a 数组中
  cout<<a[i]<<" ";    // 在显示器上顺序显示 10 个数
 }
 cout<<endl;
 max=a[0];
```




文件的输入和输出 [程序分 析]

113

```
order=0;
for(i=1;i<10;i++)
    if(a[i]>max)
{max=a[i];          // 将当前最大值放在 max 中
order=i;            // 将当前最大值的元素序号放在 order 中
}
cout<<"max="<<max<<endl<<"order="<<order<<endl;
infile.close();
return 0;
}
```

运行情况如下：

1 3 5 2 4 6 10 8 7 9(在磁盘文件中存放的 10 个数)

max=10 (最大值为 10)

order=6 (最大值是数组中序号为 6 的元素)



文件的输入和输出 [程序分析]

114

例 从键盘读入一行字符，把其中的字母字符依次存放在磁盘文件 f2.dat 中。再把它从磁盘文件读入程序，将其中的小写字母改为大写字母，再存入磁盘文件 f3.dat 。

```
#include <fstream>
using namespace std;
// save_to_file 函数从键盘读入一行字符，并将其中的字母存入磁盘文件
void save_to_file( )
{ofstream outfile("f2.dat");
// 定义输出文件流对象 outfile，以输出方式打开磁盘文件 f2.dat
if(!outfile)
{cerr<<"open f2.dat error!"<<endl;
exit(1);
}
char c[80];
cin.getline(c,80);// 从键盘读入一行字符
for(int i=0;c[i]!=0;i++) // 对字符逐个处理，直到遇' /0' 为止
if(c[i]>=65 && c[i]<=90 || c[i]>=97 && c[i]<=122)// 如果是字母字符
{outfile.put(c[i]); // 将字母字符存入磁盘文件 f2.dat
```



文件的输入和输出 [程序分 析]

115

```
cout<<c[i];}           // 同时送显示器显示
cout<<endl;
outfile.close();       // 关闭 f2.dat
}
```

// 从磁盘文件 f2.dat 读入字母字符，将其中的小写字母改为大写字母，再存入 f3.dat

```
void get_from_file()
{char ch;
 ifstream infile("f2.dat",ios::in|ios::nocreate);
// 定义输入文件流 outfile，以输入方式打开磁盘文件 f2.dat
 if(!infile)
 {cerr<<"open f2.dat error!"<<endl;
  exit(1);
 }
 ofstream outfile("f3.dat");
// 定义输出文件流 outfile，以输出方式打开磁盘文件 f3.dat
 if(!outfile)
 {cerr<<"open f3.dat error!"<<endl;
  exit(1);
 }
```



文件的输入和输出 [程序分析]

116

```
while(infile.get(ch))// 当读取字符成功时执行下面的复合语句
{if(ch>=97 && ch<=122)      // 判断 ch 是否为小写字母
ch=ch-32;                  // 将小写字母变为大写字母
  outfile.put(ch);          // 将该大写字母存入磁盘文件 f3.dat
  cout<<ch;                // 同时在显示器输出
}
cout<<endl;
infile.close( );           // 关闭磁盘文件 f2.dat
outfile.close();           // 关闭磁盘文件 f3.dat
}

int main( )
{save_to_file( );
  // 调用 save_to_file( ), 从键盘读入一行字符并将其中的字母存入磁盘文件 f2.dat
get_from_file( );
  // 调用 get_from_file(), 从 f2.dat 读入字母字符, 改为大写字母, 再存入 f3.dat
return 0;
}
```

运行情况如下：

New Beijing. Great Olypic. 2008. China.✓

NewBeijingGreatOlypicChina(将字母写入磁盘文件 f2.dat , 同时在屏幕显示)

NEWBEIJINGGREATOLYPICCHINA (改为大写字母)



文件的输入和输出 [程序分析]

117

例：将一批数据以二进制形式存放在磁盘文件中。

```
#include <fstream>
```

```
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)
```

```
{cerr<<"open error!"<<endl;
```

```
abort( );// 退出程序
```



```
    }  
    for(int i=0;i<3;i++)  
        outfile.write((char*)&stud[i],sizeof(stud[i]));  
    outfile.close( );  
    return 0;  
}
```

其实可以一次输出结构体数组的 3 个元素，将 for 循环的两行改为以下一行：

```
outfile.write((char*)&stud[0],sizeof(stud));
```

执行一次 write 函数即输出了结构体数组的全部数据。

可以看到，用这种方法一次可以输出一批数据，效率较高。在输出的数据之间不必加入空格，在一次输出之后也不必加回车换行符。在以后从该文件读入数据时不是靠空格作为数据的间隔，而是用字节数来控制。



文件的输入和输出 [程序分 析]

119

例 13.15 将刚才以二进制形式存放在磁盘文件中的数据读入内存并在显示器上显示。

```
#include <fstream>
using namespace std;
struct student
{string name;
 int num;
 int age;
 char sex;
};
int main( )
{student stud[3];
 int i;
 ifstream infile("stud.dat",ios::binary);
 if(!infile)
 {cerr<<"open error!"<<endl;
 abort( );
 }
```



文件的输入和输出 [程序分析]

120

```
for(i=0;i<3;i++)
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;
}
return 0;
}
```

运行时在显示器上显示：

NO.1

name: Li

num: 1001

age: 18

sex: f



文件的输入和输出 [程序分析]

121

NO.2

name: Fun

num: 1001

age: 19

sex: m

NO.3

name: Wang

num: 1004

age: 17

sex: f

请思考：能否一次读入文件中的全部数据，如

```
infile.read((char*)&stud[0],sizeof(stud));
```

05



程序实例

例 有 5 个学生的数据，要求：

- (1) 把它们存到磁盘文件中；
- (2) 将磁盘文件中的第 1,3,5 个学生数据读入程序，并显示出来；
- (3) 将第 3 个学生的数据修改后存回磁盘文件中的原有位置。
- (4) 从磁盘文件读入修改后的 5 个学生的数据并显示出来。

要实现以上要求，需要解决 3 个问题：

- (1) 由于同一磁盘文件在程序中需要频繁地进行输入和输出，因此可将文件的工作方式指定为输入输出文件，即 `ios::in|ios::out|ios::binary`。
- (2) 正确计算好每次访问时指针的定位，即正确使用 `seekg` 或 `seekp` 函数。
- (3) 正确进行文件中数据的重写（更新）。

可写出以下程序：

```
#include <fstream>
using namespace std;
struct student
{int num;
  char name[20];
  float score;
};
```

```
int main()  
{student stud[5]={1001,"Li",85,1002,"Fun",97.5,1004,"Wang",54,  
                  1006,"Tan",76.5,1010,"ling",96};  
  fstream iofile("stud.dat",ios::in|ios::out|ios::binary);  
  // 用 fstream 类定义输入输出二进制文件流对象 iofile  
  if(!iofile)  
  {cerr<<"open error!"<<endl;  
    abort( );  
  }  
  for(int i=0;i<5;i++)// 向磁盘文件输出 5 个学生的数据  
    iofile.write((char *)&stud[i],sizeof(stud[i]));  
  student stud1[5];      // 用来存放从磁盘文件读入的数据  
  for(int i=0;i<5;i=i+2)  
  {iofile.seekg(i*sizeof(stud[i]),ios::beg); // 定位于第 0,2,4 学生数据开头  
    iofile.read((char *)&stud1[i/2],sizeof(stud1[0]));  
  }  
  // 先后读入 3 个学生的数据, 存放在 stud1[0],stud1[1] 和 stud1[2] 中  
  cout<<stud1[0].num<<" "<<stud1[0].name<<" "<<stud1[0].score<<endl;  
  // 输出 stud1[0],stud1[1] 和 stud1[2] 各成员的值
```

```
}  
cout<<endl;  
stud[2].num=1012;           // 修改第 3 个学生 ( 序号为 2 ) 的数据  
strcpy(stud[2].name,"Wu");  
stud[2].score=60;  
iofile.seekp(2*sizeof(stud[0]),ios::beg); // 定位于第 3 个学生数据的开头  
iofile.write((char *)&stud[2],sizeof(stud[2])); // 更新第 3 个学生数据  
iofile.seekg(0,ios::beg);      // 重新定位于文件开头  
for(int i=0;i<5;i++)  
{iofile.read((char *)&stud[i],sizeof(stud[i])); // 读入 5 个学生的数据  
  cout<<stud[i].num<<" "<<stud[i].name<<" "<<stud[i].score<<endl;  
}  
iofile.close( );  
return 0;  
}
```

运行情况如下：

1001 Li 85(第 1 个学生数据)

1004 Wang 54 (第 3 个学生数据)

1010 ling 96 (第 5 个学生数据)

1001 Li 85 (输出修改后 5 个学生数据)

1002 Fun 97.5

1012 Wu 60 (已修改的第 3 个学生数据)

1006 Tan 76.5

1010 ling 96

本程序将磁盘文件 stud.dat 指定为输入输出型的二进制文件。这样，不仅可以向文件添加新的数据或读入数据，还可以修改（更新）数据。利用这些功能，可以实现比较复杂的输入输出任务。

请注意，不能用 ifstream 或 ofstream 类定义输入输出的二进制文件流对象，而应当用 fstream 类。

```
#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];// 用户定义的字符数组
 ostream strout(c,30); // 建立输出字符串流，与数组 c 建立关联，缓冲区长 30
 for(int i=0;i<3;i++) // 向字符数组 c 写 3 个学生的数据
   strout<<stud[i].num<<stud[i].name<<stud[i].score;
 strout<<ends; //ends 是 C++ 的 I/O 操作符，插入一个'\0'
 cout<<"array c:"<<c<<endl; // 显示字符数组 c 中的字符
}
```


运行时在显示器上的输出如下：

array c:

1001Li781002Wang89.51004Fun90

以上就是字符数组 c 中的字符。可以看到：

(1) 字符数组 c 中的数据全部是以 ASCII 代码形式存放的字符，而不是以二进制形式表示的数据。

(2) 一般都把流缓冲区的大小指定与字符数组的大小相同。

(3) 字符数组 c 中的数据之间没有空格，连成一片，这是由输出的方式决定的。如果以后想将这些数据读回赋给程序中相应的变量，就会出现问题，因为无法分隔两个相邻的数据。为解决此问题，可在输出时人为地加入空格。如 `for(int i=0;i<3;i++)`

```
strout<<" "<<stud[i].num<<" "<<stud[i].name<<" "<<stud[i].score;
```

同时应修改流缓冲区的大小，以便能容纳全部内容，今改为 50 字节。这样，运行时将输出

1001 Li 78 1002 Wang 89.5 1004 Fun 90

再读入时就能清楚地将数据分隔开。

程序实例：将一组数据保存在字符数组中

130

例 在一个字符数组 c 中存放了 10 个整数，以空格相间隔，要求将它们放到整型数组中，再按大小排序，然后再放回字符数组 c 中。

```
#include <sstream>
using namespace std;
int main( )
{char c[50]="12 34 65 -23 -32 33 61 99 321 32";
  int a[10],i,j,t;
  cout<<"array c:"<<c<<endl;// 显示字符数组中的字符串
  istringstream strin(c,sizeof(c)); // 建立输入串流对象 strin 并与字符数组 c 关联
```

```
for(i=0;i<10;i++)
    strin>>a[i];           // 从字符数组 c 读入 10 个整数赋给整型数组 a
cout<<"array a:";
for(i=0;i<10;i++)
    cout<<a[i]<<" ";      // 显示整型数组 a 各元素
cout<<endl;
for(i=0;i<9;i++)          // 用起泡法对数组 a 排序
    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;}
ostream strout(c,sizeof(c)); // 建立输出串流对象 strout 并与字符数组 c 关联
for(i=0;i<10;i++)
    strout<<a[i]<<" ";      // 将 10 个整数存放在字符数组 c
strout<<ends;               // 加入 '\\0'
cout<<"array c:"<<c<<endl;  // 显示字符数组 c
return 0;
}
```

运行结果如下：

array c: 12 34 65 -23 -32 33 61 99 321 32(字符数组 c 原来的内容)

array a: 12 34 65 -23 -32 33 61 99 321 32 (整型数组 a 的内容)

array c: -32 -23 12 32 33 34 61 65 99 321 (字符数组 c 最后的内容)

可以看到：

- (1) 用字符串流时不需要打开和关闭文件。
- (2) 通过字符串流从字符数组读数据就如同从键盘读数据一样，可以从字符数组读入字符数据，也可以读入整数、浮点数或其他类型数据。
- (3) 程序中先后建立了两个字符串流 `strin` 和 `strout`，与字符数组 `c` 关联。`strin` 从字符数组 `c` 中获取数据，`strout` 将数据传送给字符数组。分别对同一字符数组进行操作。甚至可以对字符数组交叉进行读写。
- (4) 用输出字符串流向字符数组 `c` 写数据时，是从数组的首地址开始的，因此更新了数组的内容。
- (5) 字符串流关联的字符数组并不一定是专为字符串流而定义的数组，它与一般的字符数组无异，可以对该数组进行其他各种操作。



第三部分

综合训练



综合训练 1：文件操作

- 程序功能：
 - 程序中定义 Person 类， Student 类和 MasterStudent 类，它们之间存在着继承关系；
 - 自定义输出流（向文件中输出）。
 - 使用自定义的输出流将学生的信息写到文件中。
- 实现环境：
 - Visual Studio 环境编译通过。
 - 程序代码有两个文件 Student.h 和 Student.cpp



综合训练 1：文件操作

```
#include<iostream.h>    //C++ 流头文件
#include<string.h>       // 字符串操作头文件
class Person            // 定义 Person 类
{
private:
    int age;

public:
    // 带参数的构造函数
    Person(int a)
    {    age=a;    }
    // 向流 out 输出 person 的信息定义为虚函数
    virtual void display(ostream& out)
    {
        // 设定输出格式：左对齐，输出宽度为 10

    out<<setiosflags(ios::left)<<setw(10)<<age;
    }

};
```

头文件： Student.h



综合训练 1：文件操作

//Student 类声明

```
class Student:public Person
```

```
{
```

```
private: // 定义学生类的属性
```

```
    char pName[20];
```

```
    unsigned int uID;
```

```
    float grade;
```

```
public:
```

```
    // 构造函数
```

```
    Student(char *pS, unsigned num, float g, int age);
```

```
    // 重载 Person 类中的虚函数
```

```
    void display(ostream& out);
```

```
    // 自定义输出流 定义为友元函数
```

```
    friend ostream& operator<<(ostream& out, Student& st);
```

```
};
```

头文件: Student.h



综合训练 1：文件操作

//MasterStudent 类声明 公有继承

```
class MasterStudent:public Student
```

```
{
```

```
public:
```

```
    //MasterStudent 的构造函数
```

```
    MasterStudent(char *pS, unsigned num, float g, int  
age, char t)
```

```
        :Student(pS, num, g, age), type(t) { }
```

```
    // 重载虚函数
```

```
    void display(ostream& out);
```

```
protected:
```

```
    char type;
```

```
};
```

头文件: Student.h



综合训练 1：文件操作

```
#include<fstream.h>
#include<iomanip.h>
#include"Student.h"
//Student 类的构造函数
Student::Student(char *pS, unsigned num, float g, int age)
{
    strcpy(pName, pS); // 调用字符串库函数 strcpy 将 pS 所指的字符串赋给 pName
    uID=num;
    grade=g;
}
// 显示 Student 的信息
void Student::display(ostream& out)
{
    Person::display(out); // 调用基类的成员函数
    // 自定义输出格式
    out<<setiosflags(ios::left)<<setw(20)<<pName<<uID<<uID<<", "
    <<setiosflags(ios::right)<<setw(4)<<grade;
}
```

源文件： Student.cpp



综合训练 1：文件操作

// 自定义输出流

```
ostream& operator<<(ostream& out, Student& st)
{
    st.display(out); // 输出数据到流 out
    out<<endl;
    return out;
}
```

// 输出 MasterStudent 的信息

// 重载虚函数 display

```
void MasterStudent::display(ostream& out)
{
    Student::display(out); // 调用基类的成员
    out<<"<type;    // 输出派生类增加的属性值
}
```

源文件: Student.cpp



综合训练 1：文件操作

```
// 主函数
int main()
{
    ofstream out; // 创建流
    // 打开文件
    out.open("c:\\test.txt", ios::out, filebuf::openprot);
    // 建立 Student 和 MasterStudent 对象
    Student s1(" 张三 ", 123456, 98, 24);
    MasterStudent s2(" 李四 ", 234567, 85.0, 21, 'A');
    MasterStudent s3(" 王五 ", 234568, 78.0, 21, 'B');
    // 将对象的值输出到流 out，写到文件
    // 流 out 和文件 test.txt 相连
    out<<s1;
    out<<s2;
    out<<s3;
    return 0;
}
```

源文件： Student.cpp



综合训练 2：文件操作

设计一个管理图书目的简单程序，提供的基本功能包括：可连续将新书存入文件“book.dat”中，新书信息加入到文件的尾部；也可以根据输入的书名进行查找；把文件“book.dat”中同书名的所有书显示出来。为简单起见，描述一本书的信息包括：书号，书名，出版社和作者。

分析：

可以把描述一本书的信息定义为一个 Book 类，它包含必要的成员函数。把加入的新书总是加入到文件尾部，所以，以增补方式打开输出文件。从文件中查找书时，总是从文件开始位置查找，以读方式打开文件。用一个循环语句实现可连续地将新书加入文件或从文件中查找指定的书名。由于是以一个 Book 类的实例进行文件输入输出的，所以，这文件的类型应该是二进制文件。

简化的参考程序如下：



综合训练 2：文件操作

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
class Book
{
    long int num;                // 书号
    char bookname[40];           // 书名
    char publicname[40];         // 出版社
    char name[20];               // 作者
public:
    Book()
    { num=0; bookname[0] =0;publicname[0] =0; name[0] =0;}
    char * Getbookname(void)      { return bookname ;}
    long Getnum(void ) { return num;}
    void Setdata(long , char *,char *,char *);
    void Show(void );
    Book(long , char *,char *,char *);
};
```



综合训练 2：文件操作

```
void Book::Setdata(long nu , char *bn, char *p, char *n)
{
    num = nu; strcpy(bookname, bn);
    strcpy(publicname, p); strcpy(name, n);
}
void Book::Show(void )
{
    cout<<" 书号 : "<<num<<' \t' <<" 书名 : "<<bookname<<' \t' ;
    cout<<" 出版社 : "<<publicname<<' \t' <<" 作者 : "<<name<<' \n' ;
}
Book::Book(long nu, char * bp, char *p, char *n)
{ Setdata(nu , bp, p, n); }
void main(void)
{
    Book b1, b2;
    long nu;
    char bn[40];           // 书名
    char pn[40];           // 出版社
    char na[20];           // 作者
    ifstream file1;
    ofstream file3;
    char flag = 'y';
```



综合训练 2：文件操作

```
while( flag=='y' ||flag=='Y')
{
    // 由 flag 控制循环
    cout<<"\t\t 1: 按书名查找一本书 !\n";
    cout<<"\t\t 2: 加入一本新书 !\n";
    cout<<"\t\t 3: 退出 !\n 输入选择: ";
    int f;
    cin>>f;
    switch(f)
    {
        case 1:
            cout<<" 输入要查找的书名: "; cin>>bn;
            file1.open("book.dat",ios::in | ios::binary); // 按读方式

            while(!file1.eof() )
            {
                int n;
                file1.read((char *)&b1, sizeof(Book));
                n=file1.gcount();
                if(n==sizeof(Book))
                {
                    if(strcmp(b1.Getbookname(), bn)==0) // 显示书

                        b1.Show();
                }
            }
        }
    }
```

打开件

的信息



综合训练 2：文件操作

```
        file1.close();
        break;
    case 2:
        cout<<" 输入书号: "; cin>>nu;
        cout<<" 输入书名: "; cin>>bn;
        cout<<" 输入出版社: "; cin>>pn;
        cout<<" 输入作者: "; cin>>na;
        b1.Setdata(nu, bn, pn, na);
        file3.open("book.dat", ios::app|
ios::binary); // 增补方式打开文件
        file3.write((char*)&b1, sizeof(b1));
        file3.close();
        break;
    default: flag = 'n';
}
}
}
```



综合训练 2：文件操作

```
example98 - Microsoft Visual C++ - [example98.cpp]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
[Globals] [All global members] main
example98 - Microsoft Visual C++ - [example98.cpp *]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
Book [All class members] Book
example98 - Microsoft Visual C++ - [example98.cpp *]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
[Globals] [All global members] main

example98 classes
while(!file1.eof()) {
    int n;
    file1.read((char *) &b1, sizeof(Book));
    n = file1.gcount();
    if (n == sizeof(Book))
    {
        if (strcmp(b1.Getbookname(), bn) == 0) //显示书的信息
            b1.Show();
    }
    file1.close();
    break;
    case 2:
        cout << "输入书号: "; cin >> nu;
        cout << "输入书名: "; cin >> bn;
        cout << "输入出版社: "; cin >> pn;
        cout << "输入作者: "; cin >> na;
        b1.Setdata(nu, bn, pn, na);
        file3.open("book.dat", ios::app | ios::binary); // 增补方式打开文件
        file3.write((char *) &b1, sizeof(b1));
        file3.close();
        break;
    default: flag = 'n';
}
}

example98.exe - 0 error(s), 0 warning(s)
|> 组建 / 调试 在文件F中查找 在文件F中
行 5, 列 1 REC COL 覆盖 读取
```