

第10讲

输入输出流

主讲人：赵文彬

本章主要内容

- C++的输入与输出
- 标准输出流
- 标准输入流
- 文件操作与文件流
- 字符串流

C++的输入与输出

- 输入输出的含义
 - 标准的输入输出（标准I/O）
 - 从键盘输入数据，输出到显示屏幕（**iostream**）
 - 文件的输入输出（文件I/O）
 - 以外存文件为对象的输入输出（**fstream**）
 - 字符串输入输出（串I/O）
 - 指定一个字符数组作为存储空间进行输入输出（**stringstream**）

C++的输入与输出

➤ C++的输入输出流

- 由若干字节组成的字节序列，这些字节中的数据按顺序从一个对象传送到另一对象。

➤ 输入操作

- 从输入设备流向内存

➤ 输出操作

- 从内存流向输出设备

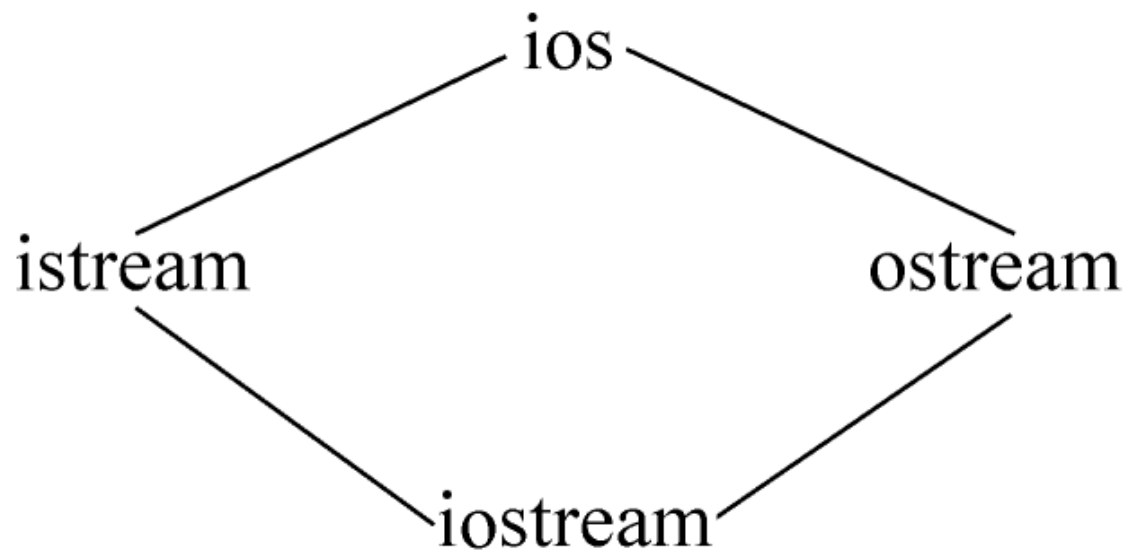
➤ 内容

- ASCII字符、二进制形式的数据、图形图像、数字音频视频或其他形式的信息

注意：输入、输出均是对内存而言的

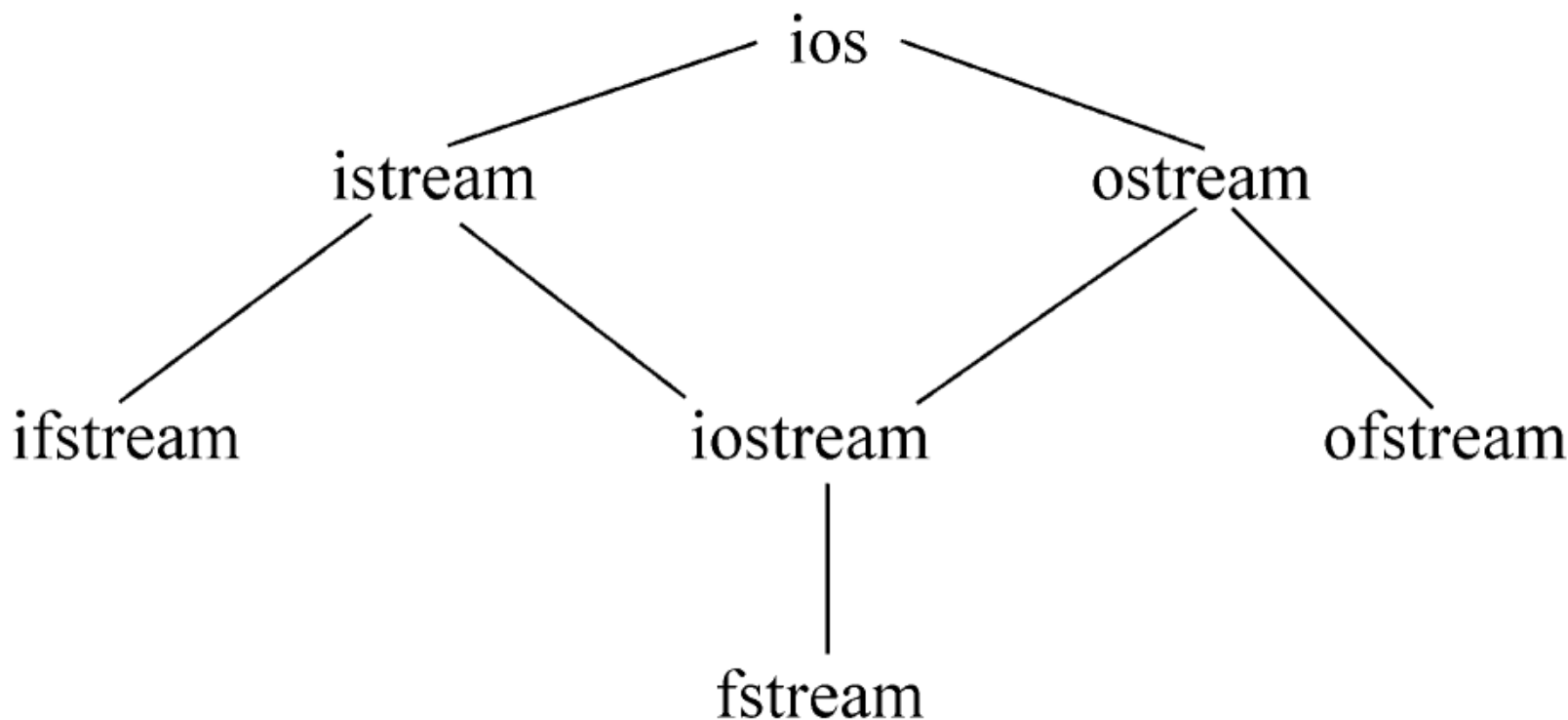
C++的输入与输出

- **iostream类库中有关的类**
 - **i-o-stream**，意为输入输出流



C++的输入与输出

- C++对文件的输入输出需要用ifstream和ofstream类，类fstream继承了类iostream。



C++的输入与输出

- 与iostream类库有关的头文件
 - `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。

C++的输入与输出

➤ 在iostream头文件中定义的流对象

对象	含义	对应设备
cin	标准输入流	键盘
cout	标准输出流	屏幕
cerr	标准错误流	屏幕
clog	标准错误流	屏幕

标准输出流

- **cout、cerr和clog流**
 - **cout**意为在控制台（终端显示器）的输出。
 - **cerr**的作用是向标准错误设备输出有关错误信息。
 - **clog**与**cerr**作用相同，二者之间一个微小的区别在于：**cerr**是不经过缓冲区，直接向显示器上输出有关信息，而**clog**中的信息存放在缓冲区中，缓冲区满后或遇**endl**时向显示器输出。

标准输出流

例10.1 有一元二次方程 $ax^2+bx+c=0$ ，其一般解为

$x_{1,2}=(-b\pm\sqrt{b^2-4ac})/2a$ ，但若 $a=0$ 或 $b^2-4ac<0$ 时，用此公式出错。编程，从键盘输入 a,b,c 的值，求 x_1 和 x_2 。如果 $a=0$ 或 $b^2-4ac<0$ ，输出出错信息。

```
#include <iostream>
```

```
#include <cmath>
```

```
using namespace std;
```

```
void main( )
```

```
{float a,b,c,disc;
```

```
  cout<<"please input a,b,c:";
```

```
  cin>>a>>b>>c;
```

```
  if (fabs(a)<1e-6) cerr<<"a is equal to zero,error!"<<endl;
```

标准输出流

```
Else if ((disc=b*b-4*a*c)<0)
cerr<<"disc=b*b-4*a*c<0"<<endl;
else
{cout<<"x1="<<(-b+sqrt(disc))/(2*a)<<endl;
  cout<<"x2="<<(-b-sqrt(disc))/(2*a)<<endl;
}
}
```

运行情况如下:

① please input a,b,c: 0 2 3✓

a is equal to zero,error!

② please input a,b,c: 5 2 3✓

disc=b*b-4*a*c<0

③ please input a,b,c: 1 2.5 1.5✓

x1=-1

x2=-1.5

标准输出流



```
#include <iomanip>
```

➤ 格式输出

➤ 使用控制符控制输出格式

- 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16, 10, 8之一)
- setfill(c) 设置填充字符c, c可以是字符常量或字符变量

标准输出流



```
#include <iomanip>
```

➤ 格式输出

➤ 使用控制符控制输出格式

- `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)` 设置浮点数以科学计数法(即指数形式)显示。

标准输出流

```
#include <iomanip>
```

➤ 格式输出

➤ 使用控制符控制输出格式

- `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)` 在以科学计数法输出E和十六进制输出字母X时，以大写表示。
- `setiosflags(ios::showpos)` 输出正数时，给出“+”号。
- `resetiosflags` 终止已设置的输出格式状态，在括号中应指定内容。

标准输出流

例10.2 用控制符控制输出格式。

```
#include <iostream>
#include <iomanip>
using namespace std;
void main( )
{int a;
  cout<<"input a:";
  cin>>a;
  cout<<"dec:"<<dec<<a<<endl;
  cout<<"hex:"<<hex<<a<<endl;
  cout<<"oct:"<<setbase(8)<<a<<endl;
  char *pt="China";
  cout<<setw(10)<<pt<<endl;
```

标准输出流

```
cout<<setfill('*')<<setw(10)<<pt<<endl;
double pi=22.0/7.0;
cout<<setiosflags(ios::scientific)<<setprecision(8);
  cout<<"pi="<<pi<<endl;
cout<<"pi="<<setprecision(4)<<pi<<endl;
cout<<"pi="<<resetiosflags(ios::scientific)<<setiosflag
s(ios::fixed)<<pi<<endl;
}
```


标准输出流

- 用流对象的成员函数控制输出格式
 - `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` 括号中内容相同。
 - `unsetf()` (`resetiosflags()`) 终止已设置的输出格式状态。

标准输出流

- 设置格式状态的格式标志
 - `ios::left` 输出数据在本域宽范围内左对齐
 - `ios::right` 输出数据在本域宽范围内右对齐
 - `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(八进制以0打头，十六进制以0x打头)

标准输出流

- 设置格式状态的格式标志
 - `ios::showpoint` 强制输出浮点数的小点和尾数0
 - `ios::uppercase` 在以科学计数法输出E和十六进制输出字母X时，以大写表示
 - `ios::showpos` 输出正数时，给出“+”号。
 - `ios::scientific` 设置浮点数以科学计数法(即指数形式)显示
 - `ios::fixed` 设置浮点数以固定的小数位数显示
 - `ios::unitbuf` 每次输出后刷新所有流
 - `ios::stdio` 每次输出后清除 `stdout`, `stderr`

标准输出流

例10.3 用流控制成员函数输出数据。

```
#include <iostream>
using namespace std;
void main( )
{int a=21;
  cout.setf(ios::showbase);
  cout<<"dec:"<<a<<endl;
  cout.unsetf(ios::dec);
  cout.setf(ios::hex);
  cout<<"hex:"<<a<<endl;
  cout.unsetf(ios::hex);
  cout.setf(ios::oct);
```

标准输出流

```
cout<<"oct:"<<a<<endl;
cout.unsetf(ios::oct);
char *pt="China";
cout.width(10);
cout<<pt<<endl;
cout.width(10);
cout.fill('*');
cout<<pt<<endl;
double pi=22.0/7.0;
cout.setf(ios::scientific);
cout<<"pi=";
cout.width(14);
```

标准输出流

```
cout<<pi<<endl;  
cout.unsetf(ios::scientific);  
cout.setf(ios::fixed);  
cout.width(12);  
cout.setf(ios::showpos);  
cout.setf(ios::internal);  
cout.precision(6);  
cout<<pi<<endl;  
}
```

- 用流成员函数put输出字符

- cout.put('a');
- cout.put(65+32);
- cout.put(71).put(79).put(79).put(71).put('\n');

}

标准输入流

➤ cin流

- cin是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。
- 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。
- **注意：**只有在输入完数据再按回车键后，该行数据才被送入键盘缓冲区，形成输入流，提取运算符“>>”才能从中提取数据。需要注意保证从流中读取数据能正常进行

标准输入流

例10.5 通过测试cin的真值，判断流对象是否处于正常状态。

```
#include <iostream>
using namespace std;
void 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;
}
```

标准输入流

- 用于字符输入的流成员函数
 - 用get函数读入一个字符
 - 不带参数的get函数

`cin.get()`

用来从指定的输入流中**提取一个字符**，函数的**返回值**就是**读入的字符**。若遇到输入流中的**文件结束符**，则函数值返回文件结束标志**EOF(End Of File)**。

标准输入流

例10.6 用get函数读入字符。

```
#include <iostream>
using namespace std;
int main( )
{int c;
  cout<<"enter a sentence:"<<endl;
  while((c=cin.get())!='\n')
  {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)

标准输入流

➤ 有一个参数的get函数

`cin.get(ch)`

```
#include <iostream>
using namespace std;
void main( )
{char c;
  cout<<"enter a sentence:"<<endl;
  while(cin.get(c)) cout.put(c);
  cout<<"End!"<<endl;
}
```

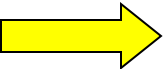
其作用是从输入流中读取一个字符，赋给字符变量**ch**。如果读取成功则函数返回非0值(真)，如失败(遇文件结束符) 则函数返回0值(假)。

标准输入流

➤ 有3个参数的get函数

`cin.get(字符数组(或字符指针), 字符个数n, 终止字符)`

作用是从输入流中读取n-1个字符，赋给指定的字符数组(或字符指针指向的数组)，如果在读取n-1个字符之前遇到指定的终止字符，则提前结束读取。如果读取成功则函数返回非0值(真)，如失败(遇文件结束符) 则函数返回0值(假)。

```
#include <iostream>
using namespace std;
void main( )
{char ch[20]; cout<<"enter a sentence:"<<endl;
  cin.get(ch,10,'\n');  cin.get(ch,10);
  cin.get(ch,10,'X');
  cout<<ch<<endl;}
```

标准输入流

例10.7 用getline函数读入一行字符。

```
#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;
}
```

标准输入流

■ **例10.8** 逐个读入一行字符，将其中的非空格字符输出。

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

```
int main( )
```

```
{char c;
```

```
while(!cin.eof( ))
```

//eof()为假表示未遇到文件结束符

```
if((c=cin.get( ))!=' ')
```

//检查读入的字符是否为空格字符

```
cout.put(c);
```

```
return 0;
```

```
}
```

标准输入流

➤ peek函数

- peek是“观察”的意思，peek函数的作用是观测当前所指字符。其调用形式为

`c=cin.peek();`

函数的返回值是指针指向的当前字符，但它只是观测，指针仍停留在当前位置，并不后移。

➤ putback函数

- 其调用形式为：`cin.putback(ch);`
- 其作用是将前面用get或getline函数从输入流中读取的字符ch返回到输入流，插入到当前指针位置，以供后面读取

标准输入流

例10.9 peek

```
#include <
using nam
```

```
void main
```

```
{char c[20
```

```
int ch;
```

```
cout<<"please enter a sentence:"<<endl;
```

```
cin.getline(c,15,'/');
```

```
cout<<"The first part is:"<<c<<endl;
```

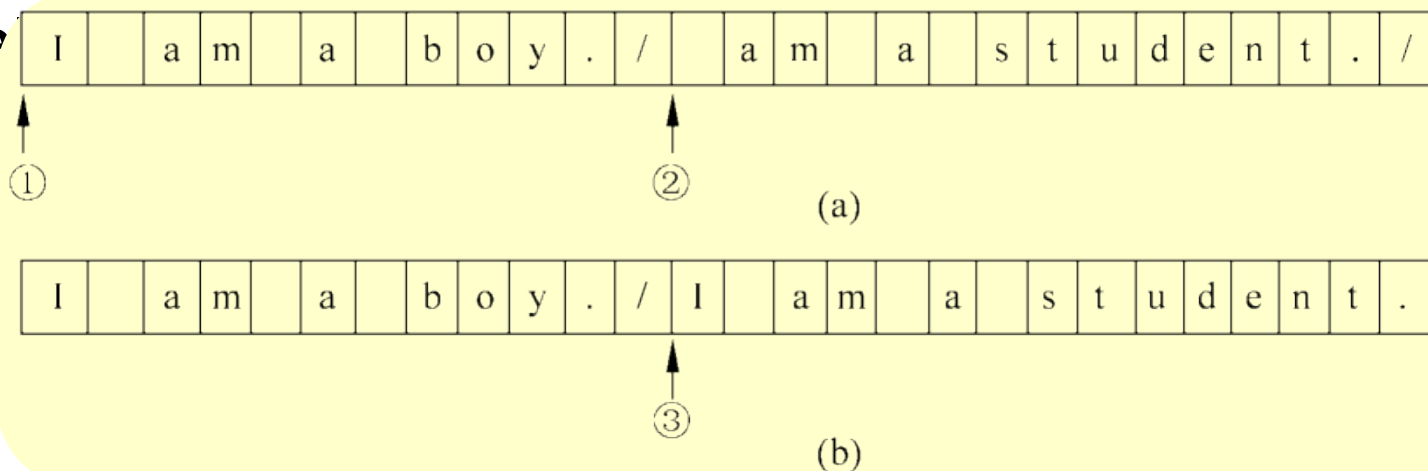
```
ch=cin.peek( );//观看当前字符
```

```
cout<<"The next character(ASCII code) is:"<<ch<<endl;
```

```
cin.putback(c[0]); //将'I'插入到指针所指处
```

```
cin.getline(c,15,'/');
```

```
cout<<"The second part is:"<<c<<endl;}
```



标准输入流

➤ ignore函数

- 其调用形式为 `cin.ignore(n, 终止字符)`
- 函数作用是跳过输入流中n个字符，或在遇到指定的终止字符时提前结束(此时跳过包括终止字符在内的若干字符)。
- 例如：`ignore(5, 'A')`//跳过输入流中5个字符，遇'A'后就不再跳了
- 也可以不带参数或只带一个参数。如
`ignore()`(n默认值为1，终止字符默认为EOF)，
相当于`ignore(1,EOF)`

标准输入流小结

➤ 需掌握函数

- `cin.get()` //读入一个字符并返回它的值
- `cin.get(char ch)` //读入一个字符并把它存储在ch
- `cin.get(char*,int,char)` //可以读字符串
- `cin.getline(char*,int,char)` //同上
- `cin.ignore()` //忽略字符
- `cin.peek()` //检查下一个输入的字符，不会把字符从流中移除
- `cin.putback()` //返回一个字符给一个流

标准输入流小结

➤ 课堂练习

- 1、从流中取一个字符，然后再放进去；
- 2、判断流中的第一个字符是不是放进去的那个字符；
- 3、从流中读取10个字符；
- 4、从流中忽略5个字符，再读取10个字符；
- 5、最后读取剩下的字符，最后输出读到的所有字符

文件操作与文件流

➤ 文件流类与文件流对象

- **文件流**是以外存文件为输入输出对象的数据流。**输出文件流**是从内存流向外存文件的数据，**输入文件流**是从外存文件流向内存的数据。**每一个文件流都有一个内存缓冲区与之对应。**
- **文件流本身不是文件**，而只是以文件为输入输出对象的流。若要对磁盘文件输入输出，就必须通过文件流来实现。

文件操作与文件流

➤ 文件流类与文件流对象

- **ifstream**类，它是从**istream**类派生的。用来支持从磁盘文件的输入。
- **ofstream**类，它是从**ostream**类派生的。用来支持向磁盘文件的输出。
- **fstream**类，它是从**iostream**类派生的。用来支持对磁盘文件的输入输出。

文件操作与文件流

调用文件流的成员函数open。如
ofstream outfile;//定义ofstream类(要的准备工
输出文件流类)对象outfile

outfile.open() 在定义文件流对象时指定参数
文件流类ofstream在声明文件流类时定义了带参数的构
造函数，其中包含了打开磁盘文件的
文件流对象.open()功能。因此，可以在定义文件流对象
输入输出方式)时指定参数，调用文件流类的构造函数
磁盘文件名可以参数来实现打开文件的功能。如
"c:\\new\\f1.dat" ostream outfile("f1.dat",ios::out);

径，则默认为工程目录下的文
件。

文件操作与文件流

方 式	说 明
<code>ios::in</code>	以输入方式打开文件
<code>ios::out</code>	以输出方式打开文件, 如果文件已存在, 则将其原有内容全部清除
<code>ios::app</code>	以输出方式打开文件, 写入的数据附加在文件末尾
<code>ios::ate</code>	打开一个已有的文件, 文件指针指向文件末尾
<code>ios::trunc</code>	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 <code>ios::out</code> 方式, 而未指定 <code>ios::app</code> , <code>ios::ate</code> , <code>ios::in</code> , 则同时默认此方式
<code>ios::binary</code>	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
<code>ios::nocreate</code>	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
<code>ios::noreplace</code>	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
<code>ios::in ios::out</code>	以输入和输出方式打开文件, 文件可读可写
<code>ios::out ios::binary</code>	以二进制方式打开一个输出文件
<code>ios::in ios::binary</code>	以二进制方式打开一个输入文件

以 `ios::in` 的方式打开要求该文件已经存在

文件操作与文件流

➤ 说明

- 每一个打开的文件都有一个文件**指针**。
- 可以用“位或”运算符“|”对输入输出方式进行组合。
- 如果打开操作失败，open函数的返回值为0(假)，如果是用调用构造函数的方式打开文件的，则流对象的值为0。

文件操作与文件流

➤ 关闭磁盘文件

- 在对已打开的磁盘文件的读写操作完成后，应关闭该文件。关闭文件用成员函数close。如

`outfile.close();`

- 所谓**关闭**，实际上是解除该磁盘文件与文件流的关联，原来设置的工作方式也失效，这样，就不能再通过文件流对该文件进行输入或输出。此时可以将文件流与其他磁盘文件建立关联，通过文件流对新的文件进行输入或输出。

文件操作与文件流

➤ 对ASCII文件的操作

- 如果文件的每一个字节中均以ASCII代码形式存放数据，即一个字节存放一个字符，这个文件就是ASCII文件(或称字符文件)。程序可以从ASCII文件中读入若干个字符，也可以向它输出一些字符。
- 对ASCII文件的读写操作可以用以下两种方法：
 - 用流插入运算符“<<”和流提取运算符“>>”输入输出标准类型的数据。
 - 用文件流的`put`,`get`,`getline`等成员函数进行字符的输入输出。

例10.10 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;
```

```
}
```

文件操作与文件流

- 对二进制文件的操作
 - 二进制文件不是以ASCII代码存放数据的，它将内存中数据存储形式不加转换地传送到磁盘文件，因此它又称为**内存数据的映像文件**。因为文件中的信息**不是字符数据**，而是字节中的二进制形式的信息，因此它又称为**字节文件**。
 - 对二进制文件的操作也需要先打开文件，用完后要关闭文件。在打开时要用`ios::binary`指定为以二进制形式传送和存储。**二进制文件除了可以作为输入文件或输出文件外，还可以是既能输入又能输出的文件。这是和ASCII文件不同的地方。**

文件操作与文件流

➤ 用成员函数read和write读写二进制文件

- 对二进制文件的读写主要用istream类的成员函数read和write来实现。这两个成员函数的原型为

istream& read(char *buffer,int len);

ostream& write(const char * buffer,int len);

字符指针**buffer**指向内存中一段存储空间。**len**是读写的字节数。

- 调用的方式为

a. **write(p1,50);**

b. **read(p2,30);**

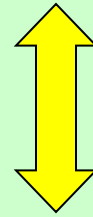
例10.11 将一批数据以二进制形式存放在磁盘文件中。

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

```
int main()
{student stud[3];
stud[0]={ "Li",1001,18,"f", "Fun",1002,19,"m", "wang",1004,17,"f"};
```

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

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



```
ofstream outfile("stud.dat",ios::binary);
if(!outfile)
{cerr<<"open error!"<<endl;
abort( );//退出程序}
```

文件操作与文件流

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。

文件操作与文件流

➤ 与文件指针有关的流成员函数

成员函数	作 用
gcount()	返回最后一次输入所读入的字节数
tellg()	返回输入文件指针的当前位置
seekg(文件中的位置)	将输入文件中指针移到指定的位置
seekg(位移量, 参照位置)	以参照位置为基础移动若干字节(“参照位置”的用法见说明)
tellp()	返回输出文件指针当前的位置
seekp(文件中的位置)	将输出文件中指针移到指定的位置
seekp(位移量, 参照位置)	以参照位置为基础移动若干字节

文件操作与文件流

说明

- 这些函数名的第一个字母或最后一个字母不是g就是p。
- 函数参数中的“文件中的位置”和“位移量”已被指定为long型整数，以字节为单位。“参照位置”可以是下面三者之一：
 - `ios::beg` 文件开头(beg是begin的缩写)，这是默认值。
 - `ios::cur` 指针当前的位置(cur是current的缩写)。
 - `ios::end` 文件末尾。

枚举常量

字符串流

- 文件流是以外存文件为输入输出对象的数据流，字符串流不是以外存文件为输入输出的对象，而以内存中用户定义的字符数组(字符串)为输入输出的对象，即将数据输出到内存中的字符数组，或者从字符数组(字符串)将数据读入。字符串流也称为内存流。
- 字符串流类有`istrstream`, `ostrstream`和`strstream`。

字符串流

- 与文件流类的三点不同
 - 输出时数据不是流向外存文件，而是流向内存中的一个存储空间。输入时从内存中的存储空间读取数据。
 - 字符串流对象关联的不是文件，而是内存中的一个字符数组，因此不需要打开和关闭文件。
 - 每个文件的最后都有一个文件结束符，表示文件的结束。而字符串流所关联的字符数组中没有相应的结束标志，用户要指定一个特殊字符作为结束符，在向字符数组写入全部数据后要写入此字符。

字符串流

- 建立输出字符串流对象
 - `ostream::ostream(char *buffer,int n,int mode=ios::out);`
 - `buffer`是指向字符数组首元素的指针，`n`为指定的流缓冲区的大小(一般选与字符数组的大小相同，也可以不同)，第3个参数是可选的，默认为`ios::out`方式。

字符串流

➤ 建立输入字符串流对象

- `istream::istream(char *buffer);`
- `istream::istream(char *buffer,int n);`
- `buffer`是指向字符数组首元素的指针，用它来初始化流对象(使流对象与字符数组建立关联)

➤ 建立输入输出字符串流对象

- `stringstream::stringstream(char *buffer,int n,int mode);`

例13.12 将一组数据保存在字符数组中。

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

```
struct student
{int num;
char name[20];
float score;
}
```

本章小结

- 标准输入输出流
- 文件流
- 字符串流