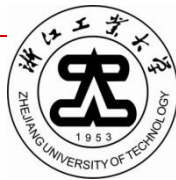


C++程序设计(II)



浙江工业大学计算机学院



第12讲 其他：输入输出流和容错机制

❖ 参考书目

- C++程序设计 谭浩强
第13章， 第14章
- 面向对象程序设计基础 李师贤 李文军 周晓聪
- C++程序设计教程（第二版） 钱能

第12讲 其他：输入输出流和容错机制



1

C++的输入输出流

2

文件操作和文件流

3

异常处理

4

命名空间



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流

- C++的输入与输出包括以下3方面的内容：
 - (1) 对系统指定的标准设备的输入和输出。即从键盘输入数据，输出到显示器屏幕，简称**标准I/O**。
 - (2) 以外存磁盘文件为对象进行输入和输出。即从磁盘文件输入数据，数据输出到磁盘文件，简称**文件I/O**。
 - (3) 对内存中指定的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(实际上可以利用该空间存储任何信息)，简称**串I/O**。
- C++采取不同的方法来实现以上3种输入输出。为了实现数据的有效流动，C++系统提供了庞大的**I/O类库**，调用不同的类去实现不同的功能。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流

- **C++的输入输出流**：**C++**的输入输出流是指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。

流表示了信息从源到目的端的流动。（1）在输入操作时，字节流从输入设备(如键盘、磁盘)流向内存；
（2）在输出操作时，字节流从内存流向输出设备(如屏幕、打印机、磁盘等)。

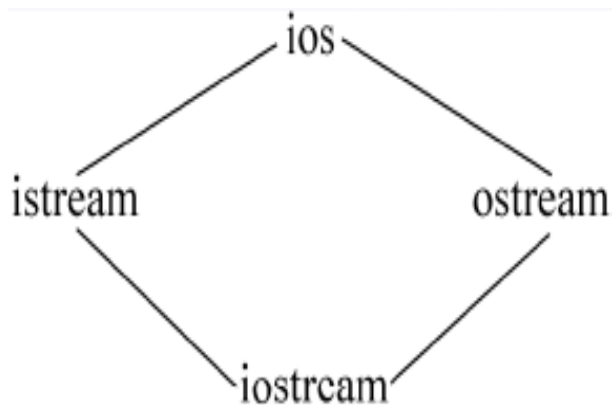
流中的内容可以是**ASCII**字符、二进制形式的数据、图形图像、数字音频视频或其他形式的信息。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流

- 在C++中，输入输出流被定义为类。C++的I/O库中的类称为**流类(stream class)**。用流类定义的对象称为**流对象**。
- **cout和cin**并不是C++语言中提供的语句，是iostream类的对象。
- **iostream**类库中有关的类-1



▲ios基类;

▲istream输入流类，支持输入操作;

▲ostream输出流类，支持输出操作;

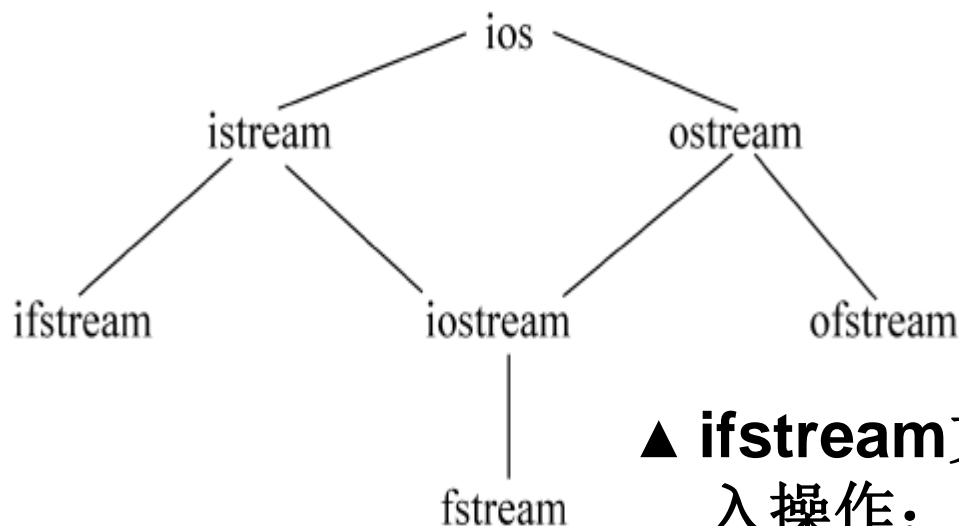
▲iostream输入输出流类，支持输入输出操作。--多重继承得到

第12讲 其他：输入输出流和容错机制



❖ 一、C++的输入输出流

■ **iostream**类库中有关的类-2



▲ **ifstream** 文件输入流类，支持文件输入操作；

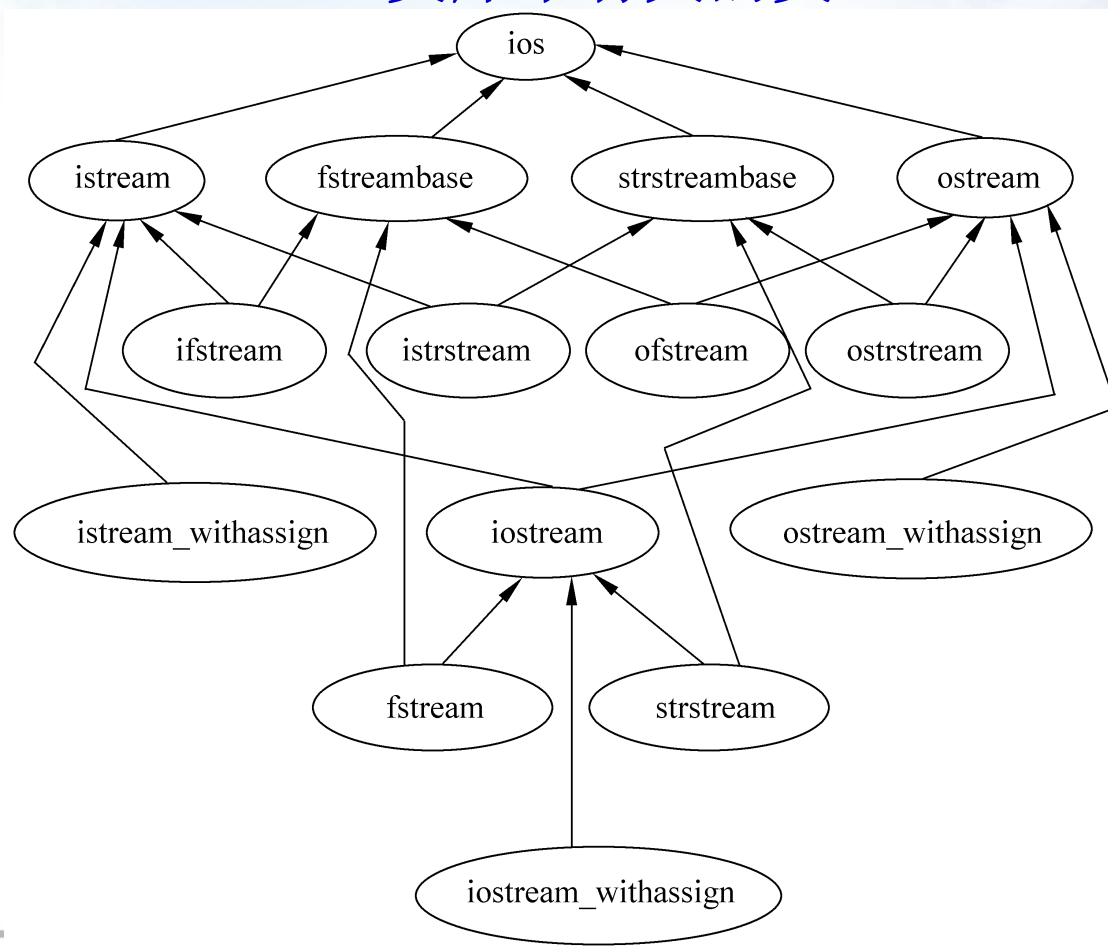
▲ **ofstream** 文件输出流类，支持文件输出操作；

▲ **fstream** 文件输入输出流类，支持文件输入输出操作。

第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流

■ iostream类库中有关的类-3





第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流

■ 与iostream类库有关的头文件（常用）

- **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。
- **omanip**: 在使用格式化I/O时应包含此头文件。



第12讲 其他：输入输出流和容错机制

- 在**`iostream`**头文件中定义的流对象
- **`iostream`**头文件中声明的类有**`ios`**, **`istream`**, **`ostream`**, **`iostream`**, **`istream_withassign`**, **`ostream_withassign`**, **`iostream_withassign`**。
- **`iostream`**头文件中定义了四个流对象：**`cin`**, **`cout`**, **`cerr`**, **`clog`**
- 在**`iostream`**头文件中定义以上4个流对象用以下的形式(以**`cout`**为例):

`ostream cout (stdout);`

- 在定义**`cout`**为**`ostream`**流类对象时，把标准输出设备**`stdout`**作为参数，这样它就与标准输出设备(显示器)联系起来，如果有：

`cout<<3;`

就会在显示器的屏幕上输出**3**。



第12讲 其他：输入输出流和容错机制

- 在**iostream**头文件中重载运算符

- “<<”和“>>”在**iostream**头文件中对左移和右移运算符进行了重载，使它们能用作标准类型数据的输入和输出运算符。

- 在**istream**和**ostream**类中分别有一组成员函数对位移运算符“<<”和“>>”进行重载，以便能用它输入或输出各种标准数据类型的数据。

- 对于不同的标准数据类型要分别进行重载，如

- ostream operator << (int);**//用于向输出流插入一个**int**数据

- ostream operator << (float);** //用于向输出流插入一个**float**数据

- ostream operator << (char);** //用于向输出流插入一个**char**数据

- ostream operator << (char *);** //用于向输出流插入一个字符串数据等。



第12讲 其他：输入输出流和容错机制

- 如果在程序中有下面的表达式：

```
cout<<"C++";
```

上面的表达式相当于

```
cout.operator<<("C++")
```

- **"C++"**的值是其首字节地址，是字符型指针(**char***)类型，通过重载函数的函数体，将字符串插入到**cout**流中，函数返回流对象**cout**。

- 在**istream**类中已将运算符**">>"**重载为对标准类型的提取运算符。

- 如果想将**"<<"**和**">>"**用于自己声明的类型的的数据，就不能简单地采用包含**iostream**头文件来解决，必须自己进行运算符重载。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- **标准输出流**：流向标准输出设备（显示器）的数据。
- **ostream**定义了3个输出流对象：**cout, cerr, clog**。
- **cout**: **Console output**的缩写，在控制端输出。

(1)不是C++预定义的关键字，是**ostream**流类的对象，在**iostream**中定义。

(2)不必考虑数据是什么类型，系统会自动判断数据的类型，并根据其类型选择调用与之匹配的运算符重载函数。

(3)**cout**流在内存中对应开辟了一个缓冲区，用来存放流中的数据，当向**cout**流插入一个**endl**时，不论缓冲区是否已满，都立即输出流中所有数据，然后插入一个换行符，并刷新流(清空缓冲区)。

(4)在**iostream**中只对“<<”和“>>”运算符用于标准类型数据的输入输出进行了重载，但未对用户声明的类型数据的输入输出进行重载。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- **标准输出流**：流向标准输出设备（显示器）的数据。
- **ostream**定义了3个输出流对象：**cout**,**cerr**,**clog**。
- **cerr**：标准错误流。**cerr**流已被指定为与显示器关联。

(1)**cerr**的作用是向标准错误设备(**standard error device**)输出有关出错信息。

(2)**cerr**与标准输出流**cout**的作用和用法差不多。但有一点不同：**cout**流通常是传送到显示器输出，但也可以被重定向输出到磁盘文件，而**cerr**流中的信息只能在显示器输出。

当调试程序时，往往不希望程序运行时的出错信息被送到其他文件，而要求在显示器上及时输出，这时应该用**cerr**。**cerr**流中的信息是用户根据需要指定的。

第12讲 其他：C++

❖ 一、C++的输入输出

- 例： $ax^2+bx+c=0$ 的根

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{float a,b,c,disc;
cout<<"please input a,b,c:";
cin>>a>>b>>c;
if (a==0) cerr<<"a is equal to zero,error!"<<endl;
//将有关出错信息插入cerr流，在屏幕输出
else
if ((disc=b*b-4*a*c)<0) //将有关出错信息插入cerr流，在屏幕输出
    cerr<<"disc=b*b-4*a*c<0"<<endl;
else{
    cout<<"x1="<<(-b+sqrt(disc))/(2*a)<<endl;
    cout<<"x2="<<(-b-bsqrt(disc))/(2*a)<<endl; }
return 0;
}
```

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



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- **标准输出流**：流向标准输出设备（显示器）的数据。
- **ostream**定义了3个输出流对象：**cout**,**cerr**,**clog**。
- **clog**：也是标准错误流，它是**console log**的缩写。

(1)作用和**cerr**相同，都是在终端显示器上显示出错信息。

(2)区别：**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。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

■ 输出格式控制

在输出数据时，有时希望数据按指定的格式输出。有两种方法可以达到此目的。一种是使用控制符的方法；第2种是使用流对象的有关成员函数。

1. 使用控制符控制输出格式*iomanip*

输出数据的控制符见书中表13.3。

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*iomanip*。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- 输出格式控制
- 1.使用格式控制符控制格式 **iomanip**

```
#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; //十进制形式输出
  cout<<"hex:"<<hex<<a<<endl; //十六进制形式输出
  cout<<"oct:"<<setbase(8)<<a<<endl; //八进制形式输出
  .....
```



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- 输出格式控制
- 1.使用格式控制符控制格式 **iomanip**

```
char* pt="China"; //pt指向字符串"China"
cout<<setw(10)<<pt<<endl; //指定域宽为10
cout<<setfill('*')<<setw(10)<<pt<<endl; //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**)



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- 输出格式控制
- 2.用流对象的成员函数控制输出格式

```
#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
```



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- 输出格式控制
- 2.用流对象的成员函数控制输出格式

```
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="
```

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**“+”**)

```
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;
```

```
}
```





❖ 注意

- Width(n)只对其后的第一个输出项有效;
- dec、hex、oct是互相排斥的，只能选其一，转换时必须用unsetf（）；
- Setf设置格式状态时，可包含多个状态;

Cout.setf(ios::internal | ios::showpos)

- 包含iomanip



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输出流

- 输出格式控制
- 3.流成员函数put输出单个字符

```
cout.put('a'); //a
```

```
cout.put(97); //a
```

```
cout.put(71).put(79).put(79).put(68).put( '\\n' );  
//GOOD
```

相当于stdlib.h中的putchar()函数。

```
putchar('a');
```

```
putchar(97);
```



例**13.4** 有一个字符串“**BASIC**”，要求把它们按相反的顺序输出。

```
#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.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

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**13.4**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。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

- **标准输入流**：从标准输入设备（键盘）流向程序的数据。
- **istream**定义了1个输入流对象：**cin**。
- **cin**：是**istream**类的对象，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。
(1)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。
(2)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。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

- **标准输入流**：从标准输入设备（键盘）流向程序的数据。
- **istream**定义了1个输出流对象：**cin**。

```
#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;
} //通过测试cin的真值，判断流对象是否处于正常状态
```



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.”。



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

■ 输入流成员函数

1.get:

(1)不带参数的get函数：调用形式为cin.get()

运行情况如下：

用来从指定的输入流中提取一
读入的字符。若遇到输入流
返回文件结束标志EOF(End

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。

```
#include <iostream>
```

```
int main( )
```

```
{int c;
```

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

```
while((c=cin.get())!=EOF)
```

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

```
return 0; }
```



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

■ 输入流成员函数

1.get:

(2) 一个参数的get函数:调用形式cin.get(ch)

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)。

```
#include <iostream>
```

```
int main( )
```

```
{char c;
```

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

```
while(cin.get(c)) //读取字符,如果读取成功则为真
```

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

```
cout<<"end"<<endl;
```

```
return 0; }
```




第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

■ 输入流成员函数

1.get:

(3)3个参数的get函数:

调用形式为`cin.get(字符数组, 字符个数n, 终止字符)`
或

`cin.get(字符指针, 字符个数n, 终止字符)`

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)



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

■ 输入流成员函数

1.get:

(3)3个参数的get函数:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{char ch[20];
```

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

```
cin.get(ch,10,'\n');//指定换行符为终止字符
```

```
//等价于cin.get(ch,10);
```

```
//或者使用其他终止字符cin.get(ch,10,'E');
```

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

```
return 0;
```

```
}
```

第12讲 其他：输入输出流和容错机制



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

例 12-1

```
#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; }
```



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

- 输入流成员函数
- **3.eof: end of file** 的缩写，表示“文件结束”。从输入流读取数据，如果到达文件末尾(遇文件结束符)，值为非零值(表示真)，否则为0(假)。

```
#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;
}
```

运行情况如下：

```
C++ is very interesting.✓
C++isveryinteresting.
^Z(结束)
```



第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

- 输入流成员函数
- **4.peek:** “查看”下一个字符。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**。如果要访问的字符是文件结束符，则函数值是**EOF(-1)**。

- **5.putback:** 调用形式**`cin.putback();`**

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。

运行情况如下:

please enter a sentence:

I am a boy./ am a student./ ☒

The first part is:I am a boy.

The next character(ASCII code) is:32(下一个字符是空格)

The second part is:I am a student

peek和putback例

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{char c[20];
```

```
int ch;
```

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

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

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

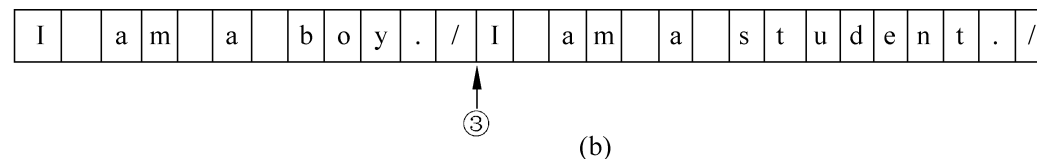
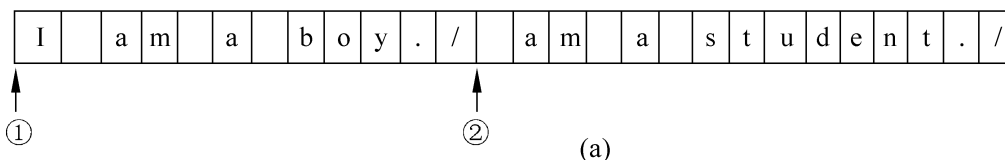
```
ch=cin.peek( );//观看当前字符
```

```
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;
```

```
return 0; }
```





第12讲 其他：输入输出流和容错机制

❖ 一、C++的输入输出流-标准输入流

- 输入流成员函数
- **6.ignore**: 调用形式为`cin.ignore(n,终止字符)`

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)。

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)`。



例**13.10**用**ignore**函数跳过输入流中的字符。

先看不用**ignore**函数的情况：

```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
  cin.get(ch,20,'/');
  cout<<"The first part is:"<<ch<<endl;
  cin.get(ch,20,'/');
  cout<<"The second part is:"<<ch<<endl;
  return 0;
}
```

运行结果如下：

I like C++./I study C++./I am happy.☑

The first part is:I like C++.

The second part is:(字符数组ch中没有从输入流中读取有效字符)

如果希望第二个cin.get函数能读取"I study C++.", 就应该设法跳过输入流中第一个'/', 可以用ignore函数来实现此目的, 将程序改为

```
#include <iostream>
using namespace std;
int main( )
{char ch[20];
  cin.get(ch,20,'/');
  cout<<"The first part is:"<<ch<<endl;
  cin.ignore( );//跳过输入流中一个字符
  cin.get(ch,20,'/');
  cout<<"The second part is:"<<ch<<endl;
  return 0;
}
```

运行结果如下:

I like C++./I study C++./I am happy.☑

The first part is:I like C++.

The second part is:I study C++.



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

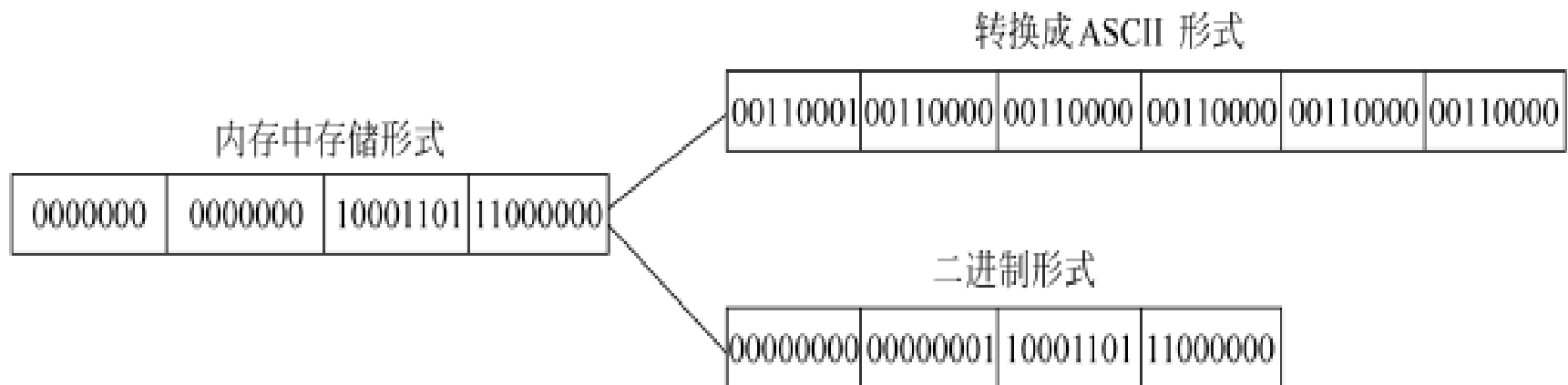
- **文件**：一般指存储在外部介质上数据的集合。一批数据是以文件的形式存放在外部介质上的。操作系统是以文件为单位对数据进行管理的。要向外部介质上存储数据也必须先建立一个文件（以文件名标识），才能向它输出数据。
- **外存文件**包括磁盘文件、光盘文件和U盘文件。目前使用最广泛的是磁盘文件。
- **按用户使用角度分**：程序文件(**program file**); 数据文件(**data file**)。程序中的输入和输出的对象就是数据文件。
- **按文件中数据的组织形式分**：**ASCII**文件；二进制文件



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **ASCII文件和二进制文件的差别：**对于字符信息，在内存中是以**ASCII**代码形式存放的，因此，无论用**ASCII**文件输出还是用二进制文件输出，其数据形式是一样的。但是对于数值数据，二者是不同的。





第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **C++**提供低级I/O功能和高级I/O功能。
- **低级的I/O功能**：是以字节为单位输入和输出的，在输入和输出时不进行数据格式的转换。这种输入输出速度快、效率高，一般大容量的文件传输用无格式转换的I/O。但使用时会感到不大方便。
- **高级的I/O功能**：是把若干个字节组合为一个有意义的单位，然后以**ASCII**字符形式输入和输出。传输大容量的文件时由于数据格式转换，速度较慢，效率不高。



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **文件流**是以外存文件为输入输出对象的数据流。每一个文件流都有一个内存缓冲区与之对应。
- **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。
- **文件流与文件概念的区分**：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。
- **(1) ifstream类**：从istream类派生的。用来支持从磁盘文件的输入。
- **(2) ofstream类**：从ostream类派生的。用来支持向磁盘文件的输出。
- **(3) fstream类**：从iostream类派生的。用来支持对磁盘文件的输入输出。



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- C++与文件操作相关的类：
 - **(1) ifstream类**：从istream类派生的。用来支持从磁盘文件的输入。
 - **(2) ofstream类**：从ostream类派生的。用来支持向磁盘文件的输出。
 - **(3) fstream类**：从iostream类派生的。用来支持对磁盘文件的输入输出。
- 要以磁盘文件为对象进行输入输出，必须定义一个文件流类的对象，通过文件流对象将数据从内存输出到磁盘文件，或者通过文件流对象从磁盘文件将数据输入到内存。

`ofstream outfile;` //定义输出文件流对象



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- 建立了一个输出文件流对象，但是还未指定它向哪一个磁盘文件输出，需要在使用时加以指定。
- **文件的打开：**打开文件是指在文件读写之前做必要的准备工作，包括：**(1)**为文件流对象和指定的磁盘文件建立关联，以便使文件流流向指定的磁盘文件。**(2)**指定文件的工作方式。

- **方式一、使用open**

ofstream outfile;

//定义**ofstream**类(输出文件流类)对象**outfile**

outfile.open("f1.dat",ios::out);

//使文件流与**f1.dat**文件建立关联

可包含路径



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **文件的打开：**打开文件是指在文件读写之前做必要的准备工作，包括：**(1)**为文件流对象和指定的磁盘文件建立关联，以便使文件流流向指定的磁盘文件。**(2)**指定文件的工作方式。

- **方式二、在定义文件流对象时制定参数**

```
ofstream outfile ("f1.dat",ios::out);
```

```
//定义ofstream类(输出文件流类)对象outfile
```

```
//并使文件流与f1.dat文件建立关联
```

①每个打开的文件都有一个文件指针；

②如果打开失败，则返回0（使用open()函数）；或者流对象的值为0(使用构造函数打开)；

③可以用位或运算符“|”对输入输出方式进行组合；

④输入输出方式见后续表格。



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

名称	描述
ios::in	打开文件供读取
ios::out	打开文件，供写入
ios::app	写入的所有数据将被追加到文件的末尾 此方式使用 ios::out
ios::ate	写入的所有数据将被追加到文件的末尾 此方式不使用 ios::out
ios::trunk	删除文件原来已存在的内容（清空文件）
ios::nocreate (新版本I/O中无)	如果要打开的文件并不存在， 那么以此参数调用 open() 函数将无法进行。
ios::noreplace (新版本I/O中无)	如果要打开的文件已存在， 试图用 open() 函数打开时将返回一个错误
ios::binary	以二进制的形式打开一个文件



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **文件的关闭**：在对已打开的磁盘文件的读写操作完成后，应关闭该文件。即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。
- **方式**：用成员函数**close**。

outfile.close();

实际上是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。如，

outfile.open("f2.dat",ios::app|ios::nocreate);



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- 文件的操作--对ASCII文件的操作
- 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。读写操作可以用以下两种方法：
 - (1)用<<和>> 来输入输出标准类型的数据。
 - (2)用前面介绍的文件流的put,get,getline等成员函数进行字符的输入输出。



例**13.11** 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;
}
```



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。



例**13.12** 从例**13.11**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];
```



```
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)



例13.13 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
```




```
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);
 }
```



```
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)



磁盘文件**f3.dat**的内容虽然是**ASCII**字符，但人们是不能直接看到的，如果想从显示器上观看磁盘上**ASCII**文件的内容，可以采用以下两个方法：

(1) 在**windows**环境下,用记事本打开。**F3.dat**显示为

NEWBEIJINGGREATOLYPICCHINA

(2) 编一程序将磁盘文件内容读入内存，然后输出到显示器。可以编一个专用函数。

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



```
void display_file(char *filename)
{ifstream infile(filename,ios::in|ios::nocreate);
 if(!infile)
  {cerr<<"open error!"<<endl;
   exit(1);}
 char ch;
 while(infile.get(ch))
cout.put(ch);
 cout<<endl;
 infile.close();
}
```

然后在调用时给出文件名即可：

```
int main( )
{display_file("f3.dat");//将f3.dat的入口地址传给形参filename
 return 0;
}
```

运行时输出**f3.dat**中的字符：

NEWBEIJINGGREATOLYPICCHINA



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- **文件的操作--对二进制文件的操作**
- 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**`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。

- 用成员函数**`read`**和**`write`**读写二进制文件

`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);`



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- 文件的操作--对二进制文件的操作
- 用成员函数**read**和**write**读写二进制文件
- 例6：将一批数据以二进制形式存放在磁盘文件中。
- 例7：将例6中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。



例13.14 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",1004,1
7,'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。

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。



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( );
 }
```



```
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



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));`



第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

- 文件的操作--对二进制文件的操作
- 与指针有关的流成员函数：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，这是默认值。

- ios::cur 指针当前的位置。

- ios::end 文件末尾。

```
infile.seekg(100);
```

```
//文件指针向前移到100字节位置
```

```
infile.seekg(-50,ios::cur);
```

```
//文件指针从当前位置后移50字节
```

```
outfile.seekp(-75,ios::end);
```

```
//文件指针从文件后移50字节
```




第12讲 其他：输入输出流和容错机制

❖ 二、文件操作和文件流

■ 文件的操作--对二进制文件的操作

- 随机访问二进制文件：可以利用上面的成员函数移动指针，随机地访问文件中任一位置上的数据，还可以修改文件中的内容。

- 例8：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],stud[1]和stud[2]中
    cout<<stud1[i/2].num<<" "<<stud1[i/2].name<<"
"<<stud1[i/2].score<<endl;
    //输出stud1[0],stud[1]和stud[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。



第12讲 其他：输入输出流和容错机制

❖ 三、字符串流

- 字符串流：也称内存流。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。
- 字符串流也有相应的缓冲区，开始时流缓冲区是空的。
 - 如果向字符数组存入数据，流缓冲区中的数据不断增加。待缓冲区满了(或遇换行符)，再存入字符数组；
 - 如果是从字符数组读数据，先将字符数组中的数据送到流缓冲区，然后从缓冲区中提取数据赋给有关变量。



第12讲 其他：输入输出流和容错机制

❖ 三、字符串流

- 在字符数组中可以存放字符，也可以存放整数、浮点数以及其他类型的数据。

在向字符数组存入数据之前，要先将数据从二进制形式转换为**ASCII**代码存放在缓冲区，再从缓冲区送到字符数组；

从字符数组读数据时，先将字符数组中的数据送到缓冲区，在赋给变量前要先将**ASCII**代码转换为二进制形式。总之，流缓冲区中的数据格式与字符数组相同。



第12讲 其他：输入输出流和容错机制

❖ 三、字符串流

- 文件流类有`ifstream`, `ofstream`和`fstream`, 字符串流类有`istrstream`, `ostrstream`和`strstream`.
 - ◆ **相同：**都是`ostream`, `istream`和`iostream`类的派生类，操作方法是基本相同的。向内存中的一个字符数组写数据就如同向文件写数据一样。



第12讲 其他：输入输出流和容错机制

❖ 三、字符串流

- 文件流类有`ifstream`, `ofstream`和`fstream`, 字符串流类有`istrstream`, `ostrstream`和`strstream`。

◆ 不同：

- (1) 输入输出时数据与内存中的一个存储空间打交道，而不是外存文件。
- (2) 字符串流关联的不是文件，而是内存中的一个字符数组，因此不需要打开和关闭文件。
- (3) 每个文件以“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‘\0’作为结束符，全部数据后要加此字符。
- (4) 没有`open`成员函数，因此要在建立字符串流对象时通过给定参数来确立字符串流与字符数组的关联。即通过调用构造函数来解决此问题。



1. 建立输出字符串流对象

ostream类提供的构造函数的原型为

```
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。

```
ostream strout(ch1,20);
```

建立输出字符串流对象strout，并使strout与字符数组ch1关联(通过字符串流将数据输出到字符数组ch1)，流缓冲区大小为20。



2. 建立输入字符串流对象

istream类提供了两个带参的构造函数，原型为

```
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)。可以用以下语句建立输入字符串流对象：

```
istream strin(ch2);
```

作用：建立输入字符串流对象**strin**，将字符数组**ch2**中的全部数据作为输入字符串流的内容。

```
istream strin(ch2,20);
```

流缓冲区大小为**20**，因此只将字符数组**ch2**中的前**20**个字符作为输入字符串流的内容。



3. 建立输入输出字符串流对象

stringstream类提供的构造函数的原型为

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

可以用以下语句建立输入输出字符串流对象:

```
stringstream strio(ch3,sizeof(ch3),ios::in|ios::out);
```

作用: 建立输入输出字符串流对象, 以字符数组**ch3**为输入输出对象, 流缓冲区大小与数组**ch3**相同。

字符串流类是在头文件**stringstream**中定义的, 因此程序中在用到**istringstream**, **ostringstream**和**stringstream**类时应包含头文件**stringstream**



例13.17 将一组数据保存在字符数组中。

```
#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

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。



例13.18 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，然后再存放回字符数组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];  
a  
cout<<"array a:";  
for(i=0;i<10;i++)  
    cout<<a[i]<<" ";  
cout<<endl;  
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;}  
ostream strout(c,sizeof(c));  
for(i=0;i<10;i++)  
    strout<<a[i]<<" ";  
strout<<ends;  
cout<<"array c:"<<c<<endl;  
return 0;  
}
```

//从字符数组**c**读入**10**个整数赋给整型数组**a**

//显示整型数组**a**各元素

//用起泡法对数组**a**排序

//建立输出串流对象**strout**并与字符数组**c**关联

//将**10**个整数存放在字符数组**c**

//加入'\\0'

//显示字符数组**c**



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。



与字符串流关联的字符数组相当于内存中的临时仓库，可以用来存放各种类型的数据(以ASCII形式存放)，在需要时再从中读回来。它的用法相当于标准设备(显示器与键盘)，但标准设备不能保存数据，而字符数组中的内容可以随时用ASCII字符输出。它比外存文件使用方便，不必建立文件(不需打开与关闭)，存取速度快。但它的生命周期与其所在的模块(如主函数)相同，该模块的生命周期结束后，字符数组也不存在了。因此只能作为临时存储空间。

第12讲 其他：输入输出流和容错机制



❖ 三、异常处理

- 程序错误：1)语法错误;2)运行错误。
- 1)语法错误由编译程序检查。
- 2) 程序能通过编译，可以运行。但在运行过程中会出现异常，得不到正确的运行结果，甚至导致程序不正常终止，或出现死机现象。--隐蔽的错误，调试困难。
- **异常和异常处理：**只要程序出现与人们期望的情况不同，都可以认为是异常，并对它进行异常处理。因此，所谓异常处理指的是对运行时出现的差错以及其他例外情况的处理。
- **异常处理的任务：**在设计程序时，应当事先分析程序运行时可能出现的各种意外的情况，并且分别制订出相应的处理方法。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **C++异常处理的策略：**如果在执行一个函数过程中出现异常，可以不在本函数中立即处理，而是发出一个信息，传给它的上一级(即调用它的函数)，它的上级捕捉到这个信息后进行处理。如果上一级的函数也不能处理，就再传给其上一级，由其上一级处理。如此逐级上送，如果到最高一级还无法处理，最后只好异常终止程序的执行。**--异常的发现与异常处理分离，异常逐层上传处理。**
- **优点：**使底层的函数专门用于解决实际任务，而不必再承担处理异常的任务，以减轻底层函数的负担，而把处理异常的任务上移到某一层去处理。这样可以提高效率。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **C++异常处理的机制：**检查(**try**)、抛出(**throw**)和捕捉(**catch**)。
- 把需要检查的语句放在**try**块中，**throw**用来当出现异常时发出一个异常信息，而**catch**则用来捕捉异常信息，如果捕捉到了异常信息，就处理它。
 - **throw**的语法：
throw表达式;
 - **try-catch**的语法结构为：
try{
 被检查的语句}
catch(异常信息类型 [变量名]){
 进行异常处理的语句}

第12讲 其他：输入输出流和容错机制



❖ 三、异常处理

- **例1：**给出三角形的三边 a, b, c ，求三角形的面积。只有 $a+b>c, b+c>a, c+a>b$ 时才能构成三角形。设置异常处理，对不符合三角形条件的输出警告信息，不予计算。
 - 无异常处理版本
 - 有异常处理版本

throw抛出什么样的数据由程序设计者自定，可以是任何类型的数据。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

■ **throw; try-catch**的注意事项：

- (1)被检测的函数必须放在**try**块中，否则不起作用；
- (2)**try**块和**catch**块作为一个整体出现，**catch**块必须紧跟在**try**块之后，不能单独使用，在二者之间也不能插入他语句。

但是在一个**try-catch**结构中，可以只有**try**块而无**catch**块。即在本函数中只检查而不处理，把**catch**处块放在其他函数中。

- (3)**try**和**catch**块中必须有用花括号括起来的复合语句，即使花括号内只有一个语句，也不能省略花括号。
- (4)一个**try-catch**结构中只能有一个**try**块，但却可以有多个**catch**块，以便与不同的异常信息匹配。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **throw; try-catch**的注意事项:

(5) **catch**的写法:

方式一、只写异常信息的类型名。如，**catch(double)**。

如果需要检测多个不同的异常信息，应当由**throw**抛出不同类型的异常信息(**catch**只检查所捕获异常信息的类型，而不检查它们的值)

异常信息可以是**C++**系统预定义的标准类型，也可以是用户自定义的类型(如结构体或类)。如果由**throw**抛出的信息属于该类型或其子类型，则**catch**与**throw**二者匹配，**catch**捕获该异常信息。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **throw; try-catch**的注意事项：

(5) **catch**的写法：

方式二、除了指定类型外，还指定变量名。

如，**catch(double d)**

此时如果**throw**抛出的异常信息是**double**型的变量**a**，则**catch**在捕获异常信息**a**的同时，还使**d**获得**a**的值。什么时候需要这样做呢？有时希望在捕获异常信息时，还能利用**throw**抛出的值，如

```
catch(double d){  
    cout<<"throw"<<d; }  

```

当抛出的是类对象时，有时希望在**catch**块中显示该对象中的某些信息。这时就需要在**catch**的参数中写出变量名(类对象名)。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

■ **throw; try-catch**的注意事项:

(5) **catch**的写法:

方式三、没有指定异常信息的类型，使用省略“...”。如，
catch(...) {**cout**<<“OK”<<**endl**;}

表示可以捕捉任何类型的异常信息，并输出“OK”。
这种**catch**子句应放在**try-catch**结构中的最后，相当于“其他”。如果把它作为第一个**catch**子句，则后面**catch**子句都不起作用。

方式四、在某些情况下，在**throw**语句中不包括任何表达式。如，**catch (...)** {**throw**;}

表示“我不处理这个异常，请上级处理”。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

■ **throw; try-catch**的注意事项：

(6) try-catch结构可以与throw出现在同一个函数中，也可以不在同一函数中。当throw抛出异常信息后，首先在本函数中寻找与之匹配的catch，如果在本函数中无try-catch结构或找不到与之匹配的catch，就转到离开出现异常最近的trycatch结构去处理。

(7)如果throw抛出的异常信息找不到与之匹配的catch块，那么系统就会调用一个系统函数terminate，使程序终止运行。

异常使用三部曲

❖ 1. 框定异常(**try** 语句块)

- 在祖先函数处，框定可能产生错误的语句序列，它是异常的根据，若不框定异常，则没有异常。

❖ 2. 定义异常处理(**catch** 语句块)

- 将出现异常后的处理过程放在**catch**块中，以便当异常被抛出，因类型匹配而捕捉时，就处理之。

❖ 3. 抛掷异常(**throw** 语句)

- 在可能产生异常的语句中进行错误检测，有错就抛掷异常

异常处理的实现机制

- ❖ 1 若有异常则通过throw操作创建一个异常对象并抛掷。
- ❖ 2 将可能抛出异常的程序段嵌在try块之中。控制通过正常的顺序执行到达try语句，然后执行try块内的保护段。
- ❖ 3 如果在保护段执行期间没有引起异常，那么跟在try块后的catch子句就不执行。程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。
- ❖ 4 catch子句按其在try块后出现的顺序被检查。匹配的catch子句将捕获并处理异常（或继续抛掷异常）。
- ❖ 5 如果匹配的处理器未找到，则运行函数terminate将被自动调用，其缺省功能是调用abort终止程序。

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

```
int Div(int x,int y);
```

```
int main()
```

```
{ try
```

```
{ cout<<"5/2="<<Div(5,2)<<endl;
```

```
  cout<<"8/0="<<Div(8,0)<<endl;
```

```
  cout<<"7/1="<<Div(7,1)<<endl;
```

```
}
```

```
catch(int)
```

```
{ cout<<"except of deviding zero.\n";
```

```
}
```

```
cout<<"that is ok.\n";
```

```
}
```

```
int Div(int x,int y)
```

```
{ if(y==0) throw y;
```

```
  return x/y;
```

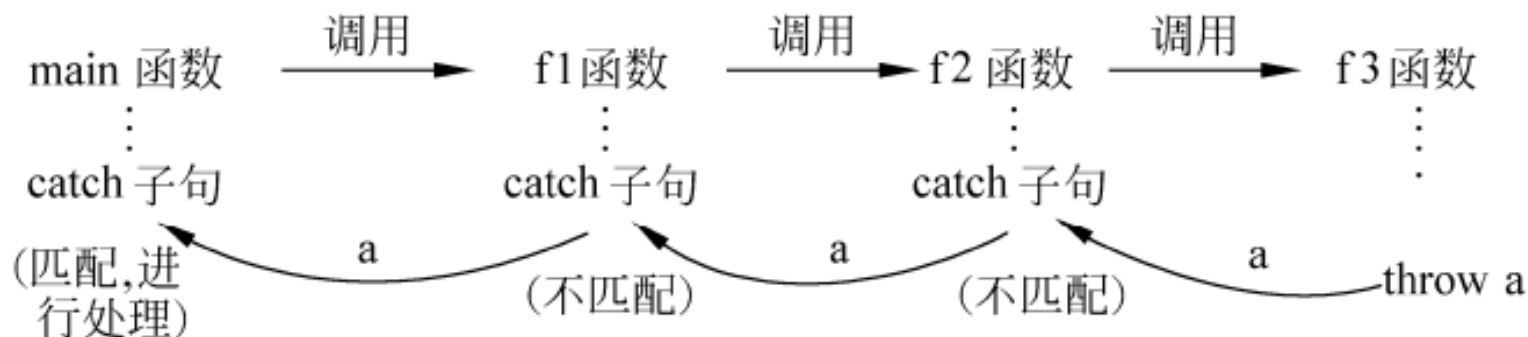
```
}
```

例12-1处理
除零异常

第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **例2：**在函数嵌套的情况下检测异常处理。



◆如果将f3函数中的**catch**子句改为**catch(double)**，而程序中其他部分不变，输出？

◆如果此基础上再将f3函数中的**catch**块改为如下，输出？

```
catch(double)
{   cout<<"OK3!"<<endl;
    throw;   }
```



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **在函数声明中进行异常情况指定：**C++允许在声明函数时列出可能抛出的异常类型，使用户清楚所用的函数是否会抛出异常以及异常信息可能的类型。
- **`double triangle(double,double,double) throw(double);`**
表示函数只能抛出**double**类型的异常信息。

`double triangle(double,double,double) throw(int,double,float,char);`
表示**triangle**函数可以抛出**int,double,float**或**char**类型的异常信息。

- **`double triangle(double,double,double) throw();`****//throw无参**
声明一个不能抛出异常的函数，即使在函数执行过程中出现了**throw**语句，实际上也并不执行**throw**语句，并不抛出任何异常信息，程序将非正常终止。

第12讲 其他：输入输出流和容错机制



❖ 三、异常处理

- 在函数声明中进行异常情况指定
- 异常指定是函数声明的一部分，必须同时出现在函数声明和函数定义的首行中，否则在进行函数的另一次声明时，编译系统会报告“类型不匹配”。如果在声明函数时未列出可能抛出的异常类型，则该函数可以抛出任何类型的异常信息。



第12讲 其他：输入输出流和容错机制

❖ 三、异常处理

- **在异常处理中处理析构函数：** C++的异常处理机制会在**throw**抛出异常信息被**catch**捕获时，对有关的局部对象进行析构(调用类对象的析构函数)，析构对象的顺序与构造顺序相反，然后执行与异常信息匹配的**catch**块中的语句。
- 在**try**块(或**try**块中调用的函数)中定义了类对象，在建立该对象时要调用构造函数；
- 在执行**try**块 (包括在**try**块中调用其他函数)的过程中如果发生了异常，此时流程立即离开**try**块。这样流程就有可能离开该对象的作用域而转到其他函数，在此之前做好结束对象前的清理工作。

异常处理中的构造与析构

❖ 找到一个匹配的**catch**异常处理后

- 初始化参数。
- 将从对应的**try**块开始到异常被抛掷处之间构造（且尚未析构）的所有自动对象进行析构。
- 从最后一个**catch**处理之后开始恢复执行。

例12-2 异常处理时的析构

```
#include <iostream.h>
void MyFunc( void );
class Expt
{ public:
    Expt(){};
    ~Expt(){};
    const char *ShowReason() const
    { return "Expt类异常。";
    }
};

void MyFunc()
{ Demo D;
  cout<<"在MyFunc()中抛掷Expt类异常。"
  <<endl;
  throw Expt();
}
```

```
class Demo
{ public:
    Demo();
    ~Demo();
};

Demo::Demo()
{
    cout<<"构造 Demo."<<endl;
}

Demo::~~Demo()
{
    cout<<"析构 Demo."<<endl;
}
```

```
int main()
{ cout<<"在main函数中。"<<endl;
  try
  { cout<<"在try块中，调用MyFunc()。" <<endl;
    MyFunc();
  }
catch( Expt E )
{ cout<<"在catch异常处理程序中。"<<endl;
  cout<<"捕获到Expt类型异常：";
  cout<<E.ShowReason()<<endl;
}
catch( char *str )
{ cout<<"捕获到其他的异常："<<str<<endl;
}
cout<<"回到main函数。从这里恢复执行。"
  <<endl;
return 0;
}
```

程序运行时输出:

在main函数中。

在try块中，调用MyFunc()。

构造 Demo.

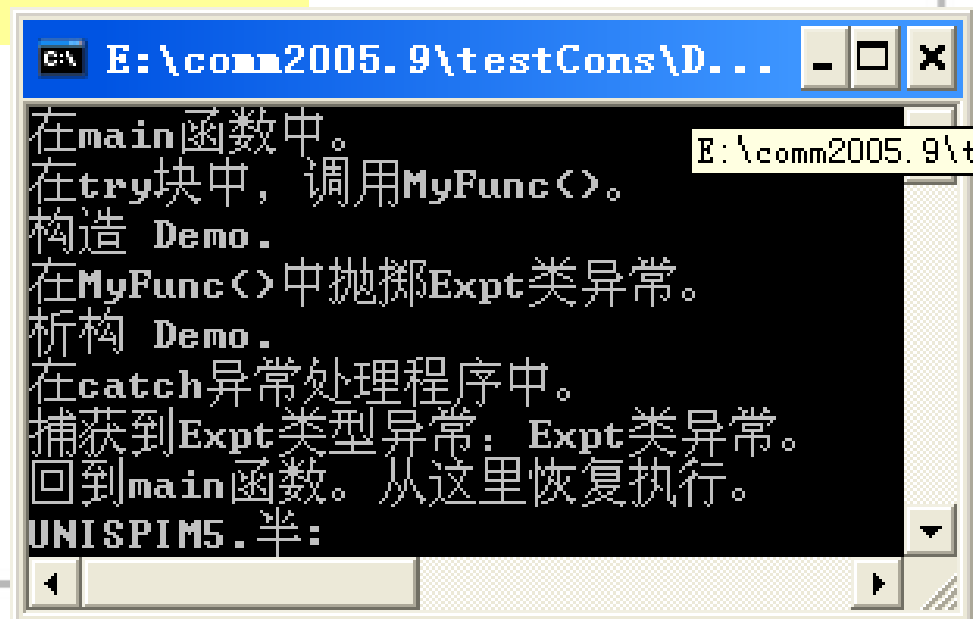
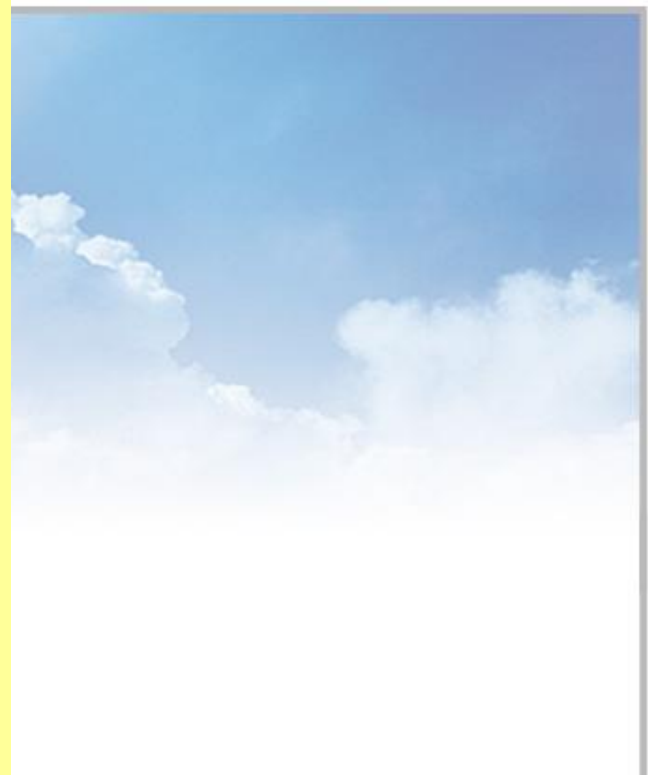
在MyFunc()中抛掷Expt类异常。

析构 Demo.

在catch异常处理程序中。

捕获到Expt类型异常： Expt类异常。

回到main函数。从这里恢复执行。



```
E:\comm2005.9\testCons\D...
在main函数中。
在try块中，调用MyFunc()。
构造 Demo.
在MyFunc()中抛掷Expt类异常。
析构 Demo.
在catch异常处理程序中。
捕获到Expt类型异常： Expt类异常。
回到main函数。从这里恢复执行。
UNISPI5.半:
```



第12讲 其他：输入输出流和容错机制

❖ 四、命名空间

■ 作用域：例1

不同作用域可以同名，加上类名的限定，不会发生冲突。

■ 名字冲突：例2

由于编译和连接导致作用域扩展，在相同作用域内同名（有两个或者更多的同名实体），产生冲突。

■ C++的解决方法：命名空间

- **命名空间**：实际上就是一个由程序设计者命名的内存区域。程序设计者可以根据需要指定一些有名字的空间域，把一些全局实体分别放在各个命名空间中，从而与其他全局实体分隔开来。

第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

- **命名空间的作用：**类似于操作系统中的目录和文件的关系。命名空间建立一些互相分隔的作用域，把一些全局实体分隔开来，以免产生名字冲突。
- 可以根据需要设置许多个命名空间，每个命名空间名代表一个不同的命名空间域，不同的命名空间不能同名。
- 全局变量可以理解为全局命名空间，独立于所有有名的命名空间之外，它不需要用**namespace**声明的，实际上是由系统隐式声明的，存在于每个程序之中。
- **语法：**

```
namespace 空间名称{  
    //变量声明  
}
```


第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

■ 命名空间的例：

```
namespace ns1//指定命名空间ns1
{
    int a;
    double b;
}
```

- 花括号内不仅可以包括变量，而且还可以包括以下类型：
①变量(可以带有初始化)；②常量；③函数(可以是定义或声明)；④结构体；⑤类；⑥模板；⑦命名空间(在一个命名空间中又定义一个命名空间，即嵌套。)。

第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

■ 命名空间的声明实例：

```
namespace ns1{  
    const intRATE=0.08;//常量  
    double pay; //变量  
    double tax( ) //函数  
    { return a*RATE;    }  
    namespace ns2 { //嵌套的命名空间  
        int age;  
    }  
}
```

第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

■ 命名空间的声明实例：

如果想输出命名空间**ns1**中成员的数据，可以采用下面的方法：

```
cout<<ns1::RATE<<endl;
cout<<ns1::pay<<endl;
cout<<ns1::tax()<<endl;
cout<<ns1::ns2::age<<endl;
//需要指定外层的和内层的命名空间名
```

■ 使用命名空间解决冲突：修改例2

第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

■ 使用命名空间的方法：

命名空间名::命名空间成员名

(2) 使用**using**命名空间成员名

using ns1::Student;

using声明的有效范围是从**using**语句开始到**using**所在的作用域结束。如果在以上的**using**语句之后有以下语句：

Student stud1(101,"Wang",18);

//相当于ns1::Student

即， **ns1::Student stud1(101,"Wang",18);**



第12讲 其他：输入输出流和容错机制

❖ 四、命名空间

■ 使用命名空间的方法：

命名空间名::命名空间成员名

(2) 使用using命名空间成员名

`using ns1::fun;` //fun是属于命名空间ns1中的fun

`cout<<fun(5,3)<<endl;` //相当于`ns1::fun(5,3)`

注意：在同一作用域中用**using**声明的不同命名空间的成员中不能有同名的成员。例如

`using ns1::Student;` //Student是命名空间ns1中的

`using ns2::Student;` //Student是命名空间ns2中的

Student stud1; //此处的**Student**是哪个命名空间中的？

→产生了二义性，编译出错。



第12讲 其他：输入输出流和容错机制

❖ 四、命名空间

■ 使用命名空间的方法：

命名空间名::命名空间成员名

(3) 使用 **using namespace** 命名空间名

using namespace ns1;

声明了在当前作用域中要用到命名空间**ns1**中的成员，该命名空间的任何成员使用时都不必用命名空间限定。如，

Student stud1(101,"Wang",18);

//Student隐含指命名空间**ns1**中的**Student**

cout<<fun(5,3)<<endl;

//fun函数是命名空间**ns1**中的**fun**函数

■ 在声明的作用域中，命名空间**ns1**的成员就好像在全局域声明的一样。因此可以不必用命名空间限定。



第12讲 其他：输入输出流和容错机制

❖ 四、命名空间

- **无名的命名空间：**C++允许使用没有名字的命名空间。

```
namespace { //命名空间没有名字  
    void fun() //定义命名空间成员  
    {cout<<"OK."<<endl;}  
}
```

- 由于命名空间没有名字，在其他文件中显然无法引用，只在本文件的作用域内有效。
- 无名命名空间的成员fun函数的作用域为文件A。在文件A中使用无名命名空间的成员，不必(也无法)用命名空间名限定。如果在文件A中有以下语句:fun();则执行无名命名空间中的成员fun函数，输出"OK."。
- 在本程序中的其他文件中也无法使用该fun函数，也就是把fun函数的作用域限制在本文件范围中。

第12讲 其他：输入输出流和容错机制



❖ 四、命名空间

- **标准命名空间std**：解决C++标准库中的标识符与程序中的全局标识符之间以及不同库中的标识符之间的同名冲突。标准C++库的所有的标识符都是在一个名为**std**的命名空间中定义的，或者说标准头文件(如**iostream**)中函数、类、对象和类模板是在命名空间**std**中定义的。

- `std::cout<<"OK."<<endl;`

`//声明cout是在命名空间std中定义的流对象`

- `using namespace std;`

`//可以不必对每个命名空间成员一一进行处理，在文件的
//开头加入`



第12讲 其他：输入输出流和容错机制

❖ 四、命名空间

- 标准命名空间**std**
- 由于在**std**中定义的实体实在太多，有时程序设计人员也弄不清哪些标识符已在命名空间**std**中定义过，为减少出错机会，有的专业人员喜欢用若干个“**using** 命名空间成员”声明来代替“**using namespace** 命名空间”声明，如

```
using std::string;  
using std::cout;  
using std::cin;
```

为了减少在每个程序中都要重复写声明，程序开发者会把常用到的命名空间**std**成员的**using**声明组成一个头文件，然后在程序中包含此头文件。

C++程序设计 (II)

Thank You !



浙江工业大学计算机学院