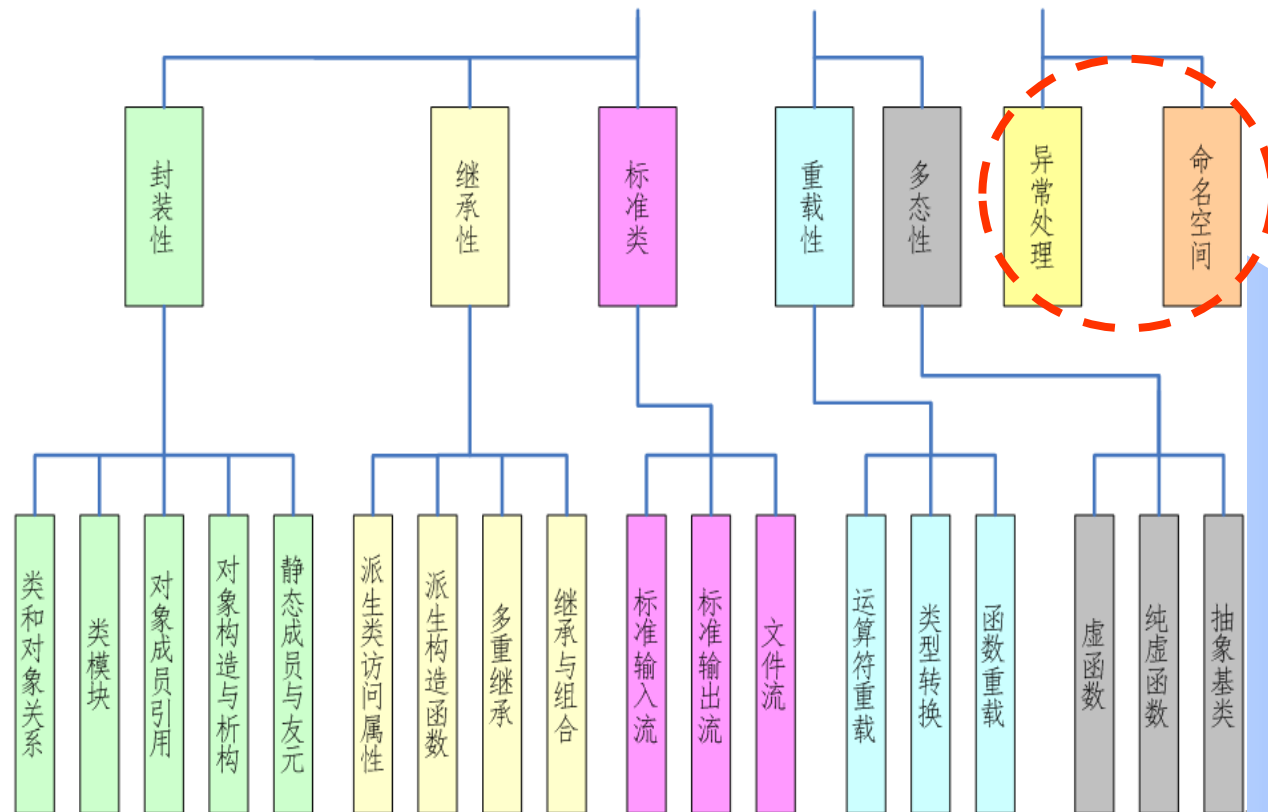


# 第11章 C++工具

## 11.1 异常处理

## 11.2 命名空间

C++的OOP程序 = 对象 + 消息 + 工具



# 重要通知

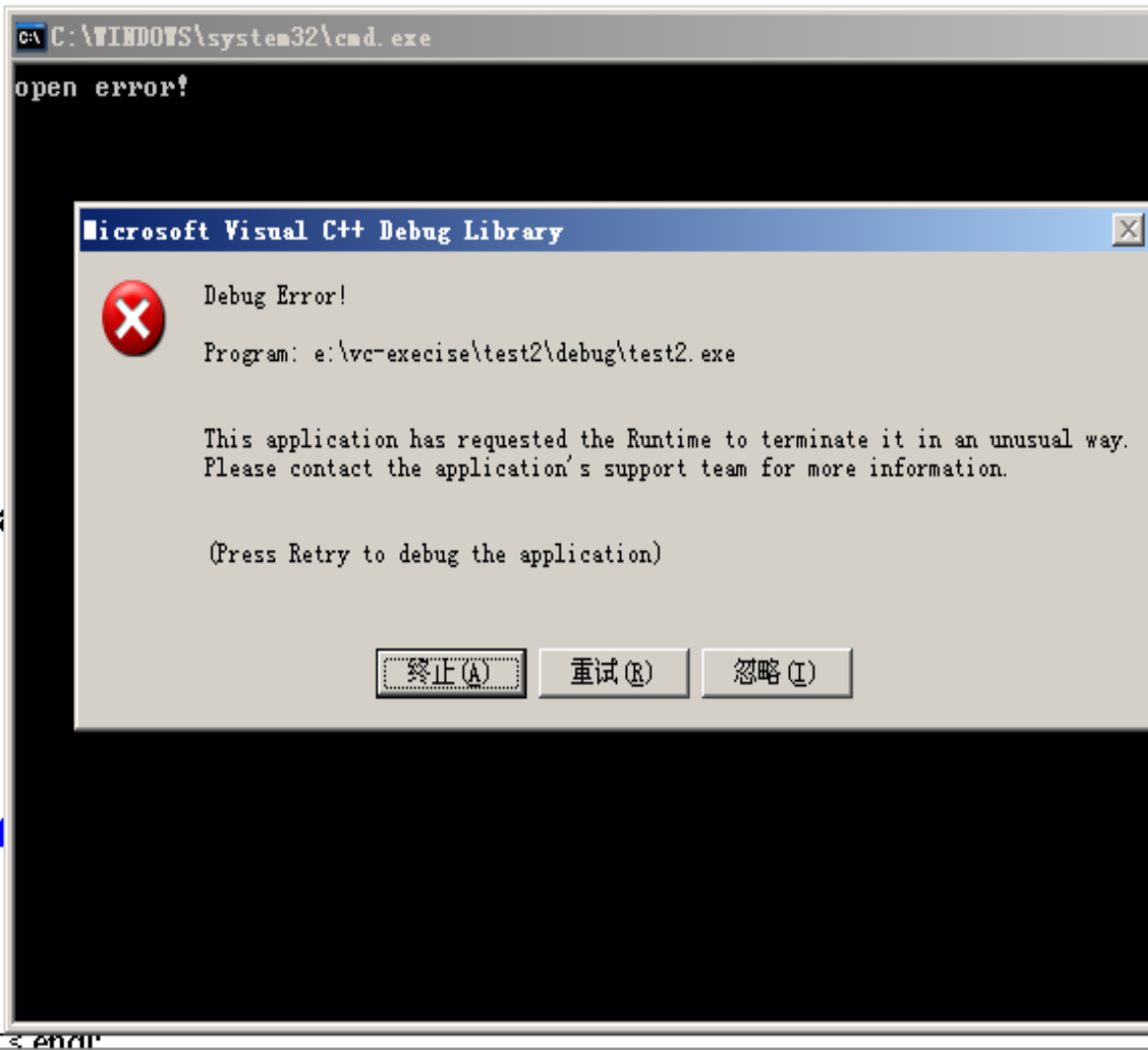
- 按照该课程教学大纲和学时安排，最后一次课计划在5月16号。
- 期末考试时间：待定（可能是15周末）。
- 小学期综合程序训练安排：最后一次课中确定。

# 11.1 异常处理

```

1 #include <fstream>
2 #include <iostream>
3 using namespace std;
4 struct student
5 { char name[20];
6   int num;
7   int age;
8   char sex;
9 };
10 int main()
11 { student stud[3];
12   int i;
13   ifstream infile("stud1.dat",ios::bin);
14   if(!infile)
15   { cerr<<"open error!"<<endl;
16     abort();
17   }
18   for(i=0;i<3;i++)
19     infile.read((char*)&stud[i],sizeof student);
20   infile.close();
21   for(i=0;i<3;i++)
22   { cout<<"NO."<<i+1<<endl;
23     cout<<"name"<<stud[i].name<<endl;

```



```

1 #include <fstream>
2 #include <iostream>
3 using namespace std;
4 struct student
5 { char name[20];
6   int num;
7   int age;
8   char sex;
9 };
10 int main()
11 { student stud[3];
12   int i;
13   ifstream infile("stud1.dat",ios::binary);
14   //if(!infile)
15   // { cerr<<"open error!"<<endl;
16   // abort();
17   // }
18   for(i=0;i<3;i++)
19     infile.read((char*)&stud[i],sizeof(stud[i]));
20   infile.close();
21   for(i=0;i<3;i++)

```

C:\WINDOWS\system32\cmd.exe

NO.1

name: 昊↓  
num:1242232  
age:1242064  
sex:

NO.2

name:⊙  
num:1458424  
age:14  
sex:

NO.3

name:  
num:2045230001  
age:3864246  
sex:

请按任意键继续. . .

## 11.1 异常处理

- 所谓**异常** (Exception) 就是程序在运行时超出了程序员预计的某些特殊情况，不在正常的情况之列。我们的程序除了处理正常情况，还要对异常情况进行及时甄别，正确处理，以防引发更大的错误：

- 文件打开失败
- 内存分配失败
- 外部模块调用失败
- 非法指针
- 非法运算 (除数为0)
- 数组访问越界
- 函数输入、输出参数值超出预期范围
- 未初始化的变量使用
- 算法逻辑错误
- ...

```
int func() {
    int nLen;
    cin >> nLen; //输入成功吗?
    float * pf = //分配成功吗?
        new float[nLen];
    for (int i=0;
        i<nLen; i++)
        cin >> pf[i];
    //...
    return nLen;
}
```

# 造成异常原因

- 造成异常的原因有两种：

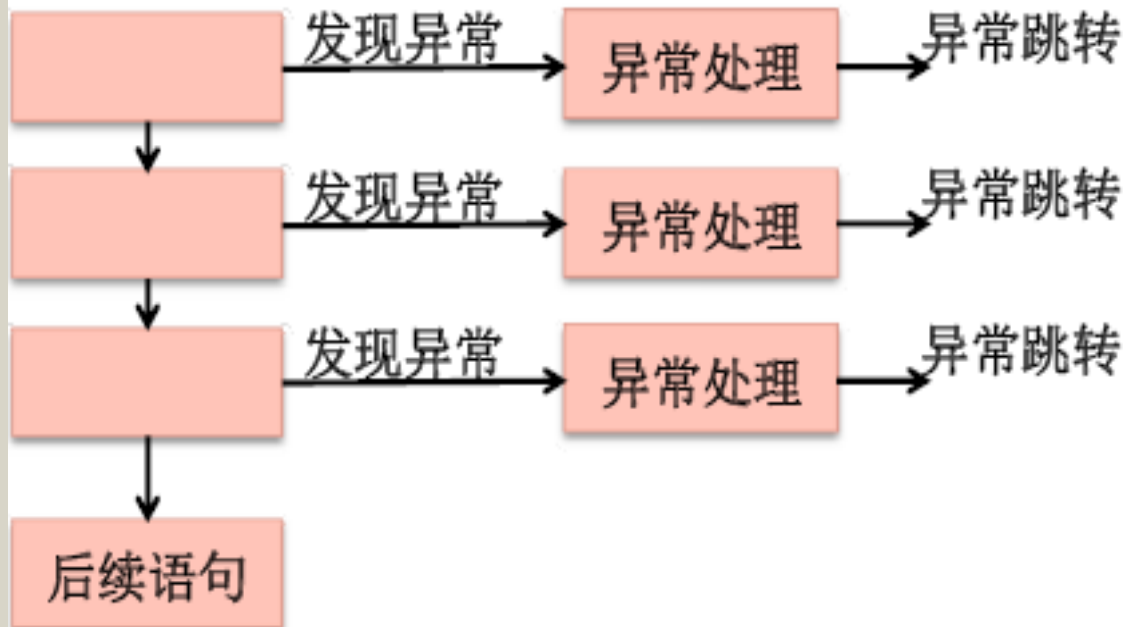
- 一是因为我们的**程序自身存在错误**，无法完成所要求的工作  
程序的错误包括编译时错误和运行时错误两种，即使编译通过的程序，仍然可能存在严重的错误和漏洞，导致运行时错误
- 二是因为我们的**程序所调用的外部模块发生错误**，致使我们的程序无法继续正常运行

每一个程序都需要依赖外部模块提供的输入输出、函数运算等系统功能，我们的程序在调用这些外部函数时，可能会遇到错误

- 在程序设计过程中，一个优秀的程序员应该能**正确预料程序运行中的各种异常情况，并对它们进行系统、完善的处理**

# 异常处理方式1

- 对于预期可能会发生的错误，我们一般采用以下程序结构来处理：



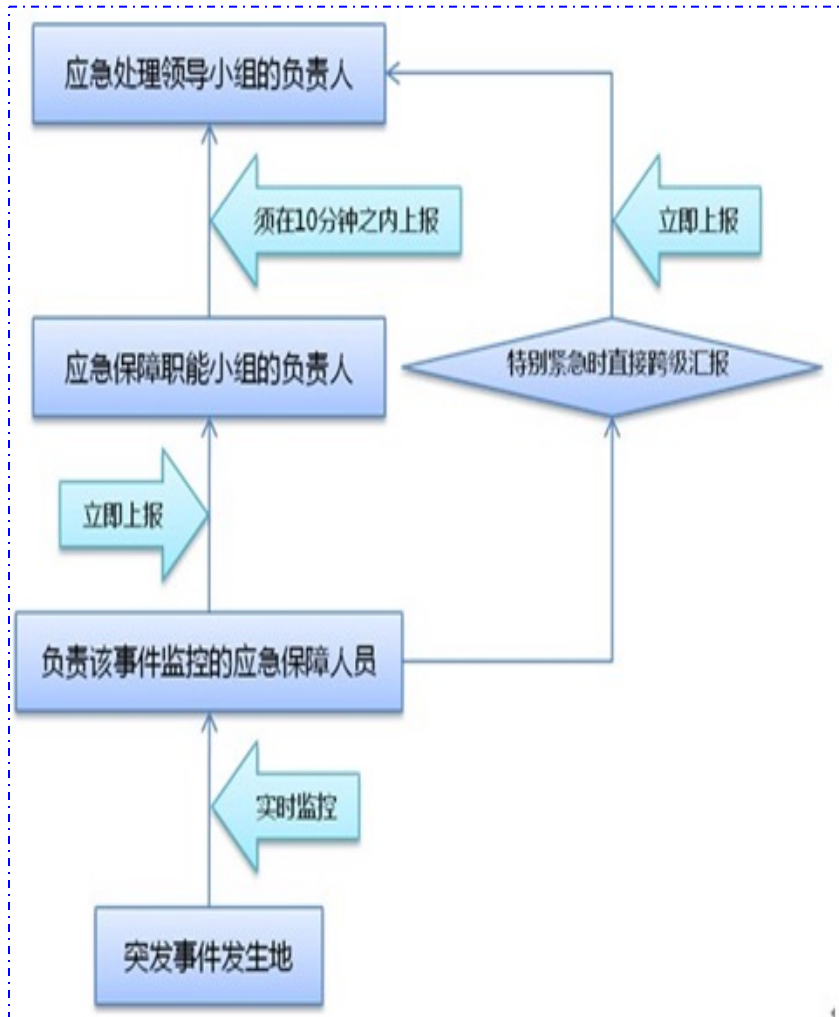
传统异常处理方法虽然解决了程序意外崩溃问题，却带来了代码混乱问题

```

int func() {
    int * p =
        new int[100];
    if (p == NULL)
        return -1;

    ifstream fs;
    fs.open("f1.txt");
    if (!fs) {
        delete [] p;
        return -2;
    }
    ...
    return 1;
}
    
```

## 异常处理方式2



### 实验场景

某班正在做C++实验；A同学“老师，我程序第10行编译通不过，咋办？”；B同学“老师，我程序第11行编译通不过，咋办？”；C同学“老师，我程序第12行编译通不过，咋办”……老师一头雾水，咋都找我呀？

D同学安慰老师说：“这是您教的C++出错处理的标准规范呀”。



## ◆ C++异常处理方式2(结构化处理) 的基本思想

- (1) 发现与处理分离机制。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，可提高效率。
- (2) 逐级上报机制。如果在执行一个函数过程中出现异常，发出一个信息给它上一级(即调用它的函数)，上级捕捉到信息后进行处理。如果上一级函数也不能处理，就再传给其上一级。如此逐级上报。如果到最高一级还无法处理，最后调用 `Terminat ()` 终止程序。

## ◆ C++异常处理语句

- (1) Try(发现)：把需要检查的语句放在try块中；
- (2) Catch(处理)：捕捉异常信息，如捕捉到就处理它；
- (3) Throw(定义)：当出现异常时发出一个异常信息。

□ 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。先写出没有异常处理时的程序。

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 int main()
5 { double triangle(double, double, double);
6   double a, b, c;
7   cin >> a >> b >> c;
8   while(a > 0 && b > 0 && c > 0)
9     { cout << triangle(a, b, c) << endl;
10      cin >> a >> b >> c; }
11   return 0; }
12
13 double triangle(double a, double b, double c)
14 { double area;
15   double s = (a + b + c) / 2;
16   area = sqrt(s * (s - a) * (s - b) * (s - c));
17   return area; }

```

C:\WINDOWS\system32\cmd.exe

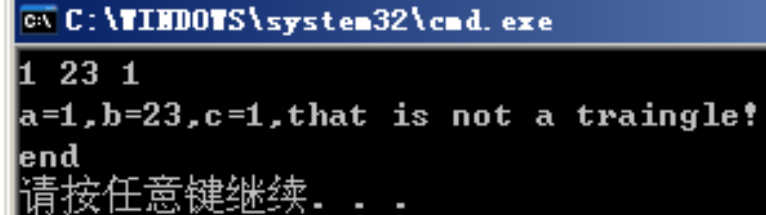
1 0 6

请按任意键继续. . .

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 int main()
5 { double triangle(double, double, double);
6   double a, b, c;
7   cin >> a >> b >> c;
8   try
9   { while(a > 0 && b > 0 && c > 0)
10     { cout << triangle(a, b, c) << endl;
11       cin >> a >> b >> c; }
12   }
13   catch(double)
14   { cout << "a=" << a << ", b=" << b << ", c=" << c << ", that is not a triangle!" << endl; }
15   cout << "end" << endl;
16   return 0; }
17
18 double triangle(double a, double b, double c)
19 { double s = (a + b + c) / 2;
20   if (a + b <= c || b + c <= a || c + a <= b) throw a;
21   return sqrt(s * (s - a) * (s - b) * (s - c)); }

```



```

C:\WINDOWS\system32\cmd.exe
1 23 1
a=1,b=23,c=1,that is not a traingle!
end
请按任意键继续. . .

```

# 结构化异常处理的一般格式

1) throw语句一般形式:

**throw 表达式;** //在（下级）函数中定义上报错误类型

2) try-catch的结构:

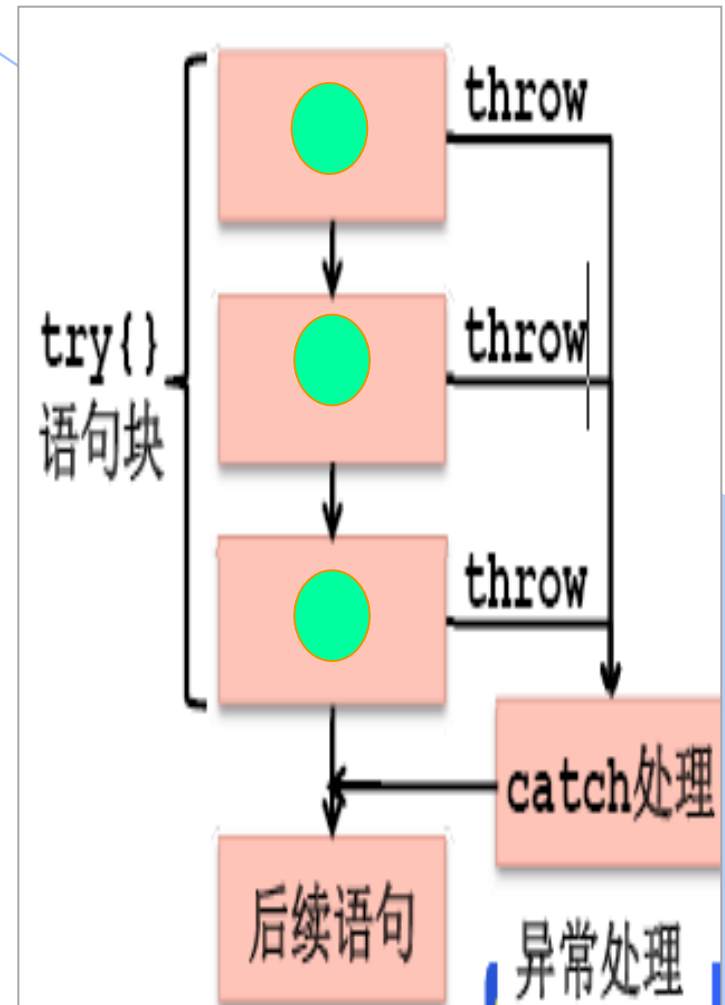
**try**

**{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}**

//在（调用）函数中捕获（被调用）函数定义且上报的错误类型



## 小 结

(1) 执行流程：当throw抛出异常信息后（该函数结束执行），先在本函数找与之匹配catch，如无try-catch结构或找不到匹配catch，则转到最近上级try-catch结构。如最终找不到匹配catch块，则系统会调用函数terminate（）来终止程序运行。

(2) try和catch结构的整体性。catch块须紧跟try块。二者之间不能插入其他语句。但在一个try-catch结构中，可只有try块而无或多个catch块。

(3) Catch检测的只是信息类型。因此catch参数可只标明类型，如catch(double)。如没有指定类型，而是删节号“...”，则表示可捕捉任何类型异常信息。如写变量catch（b），可实现throw（a）中a 到 b值传递。

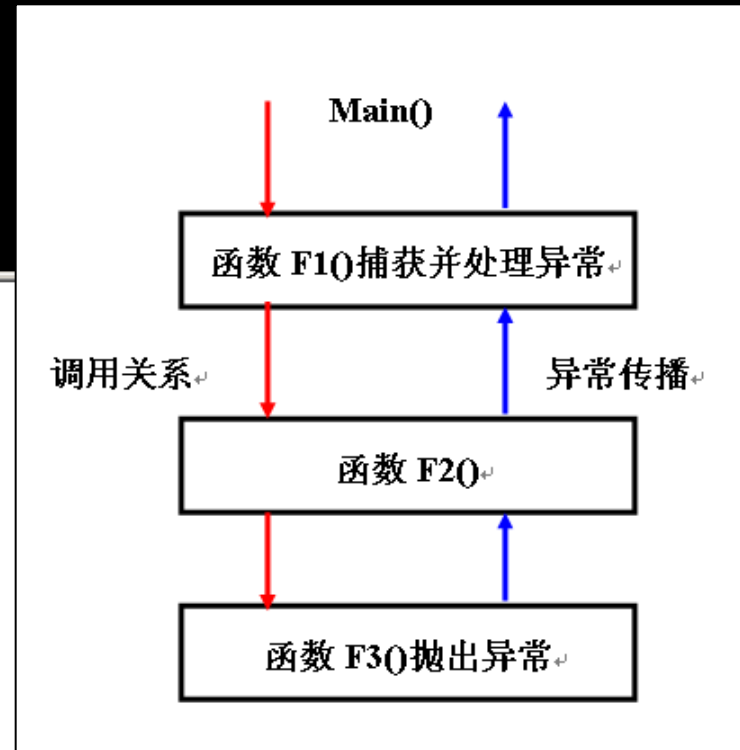
(4) throw可不包括表达式，则表示“不处理此异常，请上级处理”（调用栈向上传播）。

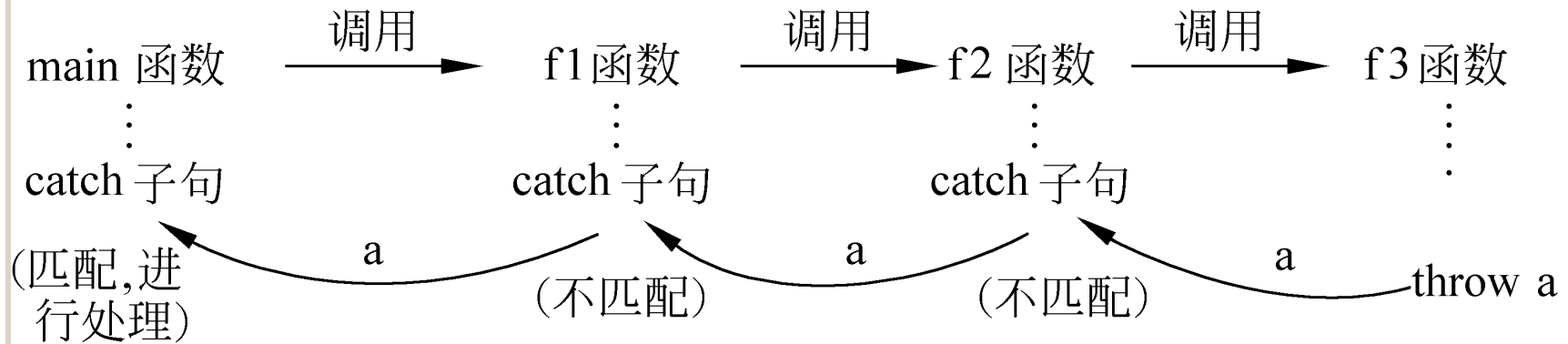
```

1  #include <iostream>
2  using namespace std;
3  int main()
4  { void f1();
5    try
6      { f1(); }
7      catch(double)
8        { cout<<"OK0!"<<endl; }
9        cout<<"end0"<<endl;
10     return 0; }
11
12 void f1()
13 { void f2();
14   try
15     { f2(); }
16     catch(char)
17       { cout<<"OK1!"<<endl; }
18       cout<<"end1"<<endl; }
19
20 void f2()
21 { void f3();
22   try
23     { f3(); }
24     catch(int)
25       { cout<<"OK2!"<<endl; }
26       cout<<"end2"<<endl; }
27 void f3()
28 { double a=0;
29   try
30     { throw a; }
31     catch(float)
32       { cout<<"OK3!"<<endl; }
33       cout<<"end3"<<endl; }
  
```

```

C:\WINDOWS\system32\cmd.exe
OK0!
end0
请按任意键继续. . .
  
```





**□ 提问：** 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，则程序运行结果是什么？

```

C:\WINDOWS\system32\cmd.exe
OK3!
end3
end2
end1
end0
请按任意键继续. . .
  
```

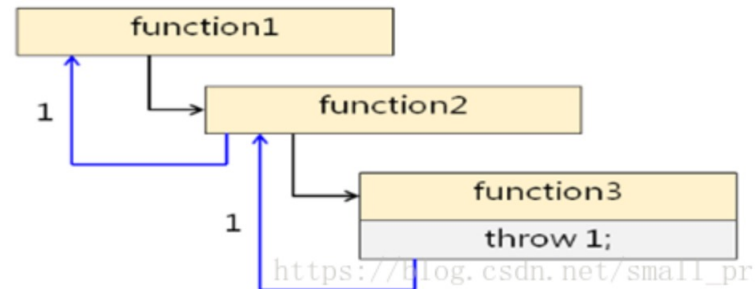
```

1  #include <iostream>
2  using namespace std;
3  int main()
4  {void f1();
5   try
6   {f1();}
7   catch(double)
8   {cout<<"OK0!"<<endl;}
9   cout<<"end0"<<endl;
10  return 0;}
11
12 void f1()
13 {void f2();
14  try
15  {f2();}
16  catch(char)
17  {cout<<"OK1!";}
18  cout<<"end1"<<endl;}
19
20 void f2()
21 {void f3();
22  try
23  {f3();}
24  catch(int)
25  {cout<<"Ok2!"<<endl;}
26  cout<<"end2"<<endl;}
27 void f3()
28 {double a=0;
29  try
30  {throw a;}
31  catch(double)
32  {cout<<"OK3!"<<endl;throw;}
33  cout<<"end3"<<endl;}
  
```

C:\WINDOWS\system32\cmd.exe

```

OK3!
OK0!
end0
请按任意键继续. . .
  
```





# 良好的程序设计素养

(1) 为了函数调用时的阅读性，建议声明函数时列出可能抛出的异常类型，缺省是指可以任何类型的异常信息

例：`double triangle(double, double, double) throw(double);`

例：`double triangle(double, double, double) throw(int, float, char);`

其中，异常信息可是标准或自定义类型数据。

(2) 如果想声明一个不抛出异常的函数，则可写为：

例：`double triangle(double, double, double) throw();`

(3) 异常指定必须同时出现在函数声明和函数定义的首行中。否则，编译系统报告“类型不匹配”。

```

1 #include <iostream>
2 using namespace std;
3 void MyFunc(void);
4
5 class Expt
6 {public:
7     Expt(){};
8     int ShowReason() const
9     {cout<< "Expt类异常"<< endl;
10      return 0; }
11 };
12
13 void MyFunc()
14 { int d;
15  cout<<"输入一个数到d"<<endl;
16  cin >>d;
17  if( d<0)
18  {cout<<"1.在MyFunc () 中抛出Expt类异常"<<endl;
19   throw Expt();}
20  cout<<"程序正常运行， d="<<d<<endl;
21 }
22
23 int main()
24 {cout<<"在main函数中"<<endl;
25 try
26 {MyFunc();}
27 catch(Expt E)
28 { cout<<"2.在处理MyFunc()的异常 "<<endl;
29  E.ShowReason ();}
30 return 0;

```

C:\WINDOWS\system32\cmd.exe

在main函数中  
输入一个数到d  
-2

1.在MyFunc () 中抛出Expt类异常  
2.在处理MyFunc()的异常  
Expt类异常  
请按任意键继续. . .

C:\WINDOWS\system32\cmd.exe

在main函数中  
输入一个数到d  
2

程序正常运行， d=2  
请按任意键继续. . .

```
#include <iostream>
using namespace std;
void MyFunc( void );
class CMyException
{public:
    CMyException() {};
    ~CMyException() {};
    const char *ShowExceptionReson() const
    { return "Exception in CMyException class. ";};
}
class CDoctorDemo
{public:
    CDoctorDemo();
    ~CDoctorDemo();};
CDoctorDemo::CDoctorDemo()
{    cout << "Constructing CDoctorDemo." << endl;}
CDoctorDemo::~~CDoctorDemo()
{    cout << "Destructing CDoctorDemo." << endl;}
void MyFunc()
{    CDoctorDemo D;
    cout << "In MyFunc(). Throwing CMyException ." << endl;
    throw CMyException();}
```

```
int main()
{ cout << "In main." << endl;
  try
  { cout << "In try block, calling MyFunc()." << endl;
    MyFunc(); }
  catch( CMyException E )
  {cout << "In catch handler." << endl;
    cout << "Caught CMyException exception type: ";
    cout << E.ShowExceptionReson() << endl; }
  catch( char *str )
  {cout << "Caught some other exception: " << str << endl; }
  cout << "Back in main. Executio
  return 0; }
```

运行结果

**In main.**

**In try block, calling MyFunc().**

**Constructing CDoctorDemo.**

**In MyFunc(). Throwing CMyException**

**Destructing CDoctorDemo.**

**In catch handler.**

**Caught CMyException exception type: Exception  
in CMyException class.**

**Back in main. Execution resumes here.**

## 第3方函数类外处理的封装

```

10 void func(int i)
11 {
12     if( i < 0 )
13     {
14         throw -1;
15     }
16
17     if( i > 100 )
18     {
19         throw -2;
20     }
21
22     if( i == 11 )
23     {
24         throw -3;
25     }
26
27     cout << "Run func..." << endl;
28 }

```

```

30 void MyFunc(int i)
31 {
32     try
33     {
34         func(i);
35     }
36     catch(int i)
37     {
38         switch(i)
39         {
40             case -1:
41                 throw "Invalid Parameter";
42                 break;
43             case -2:
44                 throw "Runtime Exception";
45                 break;
46             case -3:
47                 throw "Timeout Exception";
48                 break;
49         }

```

```

53  int main(int argc, char *argv[])
54  {
55      // Demo();
56
57      try
58      {
59          MyFunc(11);
60      }
61      catch(const char* cs)
62      {
63          cout << "Exception Info: " << cs << endl;
64      }
65
66      return 0;
  
```



## 11.2 命名空间

- 命名作用域：C++引入可由用户命名作用域，用来解决程序中同名冲突问题；
- 作用域类型：文件（编译单元）、函数和复合语句、类作用域
- 不同作用域中可定义相同名字变量，互不干扰；
- ◆ 提问：如何扩展一个变量的作用域？

```

1 class A           //声明A类
2 {public:
3   void fun1();    //声明A类中的fun1函数
4   private:
5   int i; };
6 void A::fun1()    //定义A类中的fun1函数
7 {
8   //
9 }
10 class B          //声明B类
11 {public:
12   void fun1();    //B类中也有fun1函数
13   void fun2();
14 };
15 void B::fun1()   //定义B类中的fun1函数
16 {
17   //
18 }
```

- 可通过extern声明同一程序中的两个文件中的同名变量是同一个变量；
- 如果在文件B中有以下声明：  

extern int a; 表示文件B中的变量a是在其他文件中已定义的变量。

```

1 #include <iostream>
2 using namespace std;
3 #include "header1.h"
4 int main()
5 {
6     Student stud1(101,
7         "Wang",18);
8     stud1.get_data();
9     cout<<fun(5,3)<<endl;
10    return 0;
11 }

```

```

C:\WINDOWS\system32\cmd...
101 Wang 18
2.82843
请按任意键继续. . .

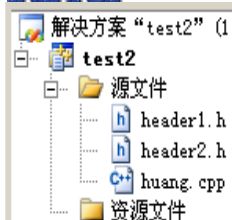
```

```

1 //例14.4中的头文件header1
2 #include <string>
3 #include <cmath>
4 using namespace std;
5 class Student
6 {
7 public:
8     Student(int n,string nam,int a)
9     {
10         num=n;name=nam;age=a;
11     }
12     void get_data();
13 private:
14     int num;
15     string name;
16     int age;
17 };
18 void Student::get_data()
19 {
20     cout<<num<<" "<<name<<" "<<age<<endl;
21 }
22 double fun(double a,double b)
23 {
24     return sqrt(a+b);
25 }

```





```
1 #include <iostream>
2 using namespace std;
3 #include "header1.h"
4 #include "header2.h"
5 int main()
6 { Student stud1(101,"Wang",18);
7   stud1.get_data();
8   cout<<fun(5,3)<<endl;
9   return 0;
10 }
11
12
```

```
1 #include <string>
2 #include <cmath>
3 using namespace std;
4 class Student
5 { public:
6   Student(int n,string nam,
7     char s)
8     { num=n;name=nam;sex=
9       s;}
10   void get_data();
11 private:
12   int num;
13   string name;
14   char sex;
15 };
16 void Student::get_data()
17 { cout<<num<<" "<<name<
18   <<" "<<sex<<endl; }
19 double fun(double a,double
20   b)
21 { return sqrt(a-b);}
```

```
1 #include <string>
2 #include <cmath>
3 using namespace std;
4 class Student
5 { public:
6   Student(int n,string nam,int a)
7     { num=n;name=nam;age=a;}
8   void get_data();
9 private:
10   int num;
11   string name;
12   int age;
13 };
14 void Student::get_data()
15 { cout<<num<<" "<<name<<" "<<age<
16   <<endl;
17 }
18 double fun(double a,double b)
19 { return sqrt(a+b);}
```

输出

显示以下输出 (S): 生成

```
1>正在编译...
1>huang.cpp
1>e:\vc-exercise\test2\test2\header2.h(5) : error C2011: 'Student' : 'class' type redefinition
1>e:\vc-exercise\test2\test2\header1.h(5) : see declaration of 'Student'
1>e:\vc-exercise\test2\test2\header2.h(14) : error C2027: use of undefined type 'Student'
1>e:\vc-exercise\test2\test2\header1.h(5) : see declaration of 'Student'
1>e:\vc-exercise\test2\test2\header2.h(15) : error C2065: 'num' : undeclared identifier
1>e:\vc-exercise\test2\test2\header2.h(15) : error C2065: 'name' : undeclared identifier
1>e:\vc-exercise\test2\test2\header2.h(15) : error C2065: 'sex' : undeclared identifier
1>e:\vc-exercise\test2\test2\header2.h(17) : error C2084: function 'double fun(double,double)' already has a body
1>e:\vc-exercise\test2\test2\header2.h(17) : see previous definition of 'fun'
```



## 11.2 命名空间

◆ 命名空间(namespace)：由应用程序来命名的内存区域。通过命名空间机制可以把一些**全局实体**分别放在各自空间中，从而来实现与其他全局实体区分。

◆ 例：

namespace ns1 //指定命名空间ns1

{int a;

double b;

}



内存空间

◆ 命名空间必须遵循先定义，后使用原则；

◆ 命名空间的变量使用：借助空间名和作用域分辨符“::”，如ns1::a, ns1::b。此法为：命名空间限定(qualified)。

## 命名空间申明一般格式：

**namespace 空间名**

**{变量(可以带有初始化)；**

**常量；**

**函数(可以是定义或声明)；**

**结构体；**

**类；**

**模板；**

**命名空间(命名空间中又定义命名空间，即嵌套)**

**}**

例： namespace ns1

{const int RATE=0.08; //常量

double pay;

double tax( ) }

例：

```
namespace ns1
{const int RATE=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;}
}
```

如果想输出命名空间ns1中成员的数据，使用方法：

```
cout<<ns1::RATE<<endl;
cout<<ns1::pay<<endl;
cout<<ns1::tax()<<endl;
cout<<ns1::ns2::age<<endl;
```

## 11.2 使用命名空间解决名字冲突

```

1 #include <iostream>
2 #include "header1.h"
3 #include "header2.h"
4 int main()
5 {Ns1::Student stud1(101,
  "Wang",18);
6  stud1.get_data();
7  cout<<Ns1::fun(5,3)<
  <endl;
8  Ns2::Student stud2(102,
  "Li","f");
9  stud2.get_data();
10 cout<<Ns2::fun(5,3)<
  <endl;
11 return 0;
12 }

```

```

C:\WINDOWS\system32\c...
1 101 Wang 18
1 2.82843
1 102 Li f
1 1.41421
请按任意键继续. . .

```

```

1 #include <string>
2 #include <cmath>
3 namespace Ns2
4 {class Student
5  {public:
6   Student(int n,string nam,
  char s)
7   {num=n;name=nam;sex=s;}
8   void get_data();
9   private:
10  int num;
11  string name;
12  char sex;};
13
14 void Student::get_data()
15 {cout<<num<<" "<<name<
  <" "<<sex<<endl;}
16 double fun(double a,double
  b)
17 {return sqrt(a-b);}}
18

```

```

1 using namespace std;
2 #include <string>
3 #include <cmath>
4 namespace Ns1
5 {class Student
6  {public:
7   Student(int n,string nam,int a)
8   {num=n;name=nam;age=a;}
9   void get_data();
10  private:
11  int num;
12  string name;
13  int age; };
14 void Student::get_data()
15 {cout<<num<<" "<<name<<" "<<age<
  <endl;
16 }
17 double fun(double a,double b)
18 {return sqrt(a+b);} }
19

```

## 11.2 引用命名空间成员的方法

### (1) 方法一：限定法

用空间名和作用域分辨符进行限定。即命名空间名::成员名

### (2) 方法二：别名法

给命名空间取一个别名，代替较长空间名。

如: namespace Television //声明命名空间  
{...}

namespace TV = Television; //别名TV与Television等价

### (3) 方法三：using 空间名::空间成员名

例. using ns1::Student;

Student stud1(101, "Wang", 18); //相当于ns1::Student

注意: using语句开始，到using所在作用域结束

### (4) 方法四：using namespace 空间名

例. using namespace ns1;

Student stud1(101, "Wang", 18); //指ns1中的Student

cout<<fun(5, 3)<<endl; //是命名空间ns1中的fun函数



```

1  #include <iostream>
2  using namespace std;
3  int iNum = 1000;
4  namespace MyName
5  {int iNum = 200;
6  int Add(int iNum) {iNum = iNum + iNum; return iNum;}}
7
8  // YourName.h
9  namespace YourName
10 {int iNum = 30000;
11 int Add(int iNum) {iNum = iNum + iNum; return iNum;}}
12 void main(void)
13 { cout << MyName::Add(10)+iNum << endl;
14   cout << MyName::Add(10)+iNum << endl;
15   namespace N2 = YourName;
16   cout << N2::Add(10)+N2::iNum << endl;
17   using namespace YourName;
18   cout << Add(10)+MyName::iNum << endl;
19   cout << MyName::Add(10)+iNum << endl;
20 }
21

```

C:\WINDOWS\sys

1020

1020

31010

1210

1020

请按任意键继续。

全局空间变量

解决方案“huangtest” (1 个项)

huangtest

头文件

源文件

huangtest.cpp

资源文件

```

1  #include <iostream>
2  using namespace std;
3  int iNum = 1000;
4  namespace MyName
5  {int iNum = 200;
6   int Add(int iNum) {iNum = iNum + iNum; return iNum;}}
7
8  // YourName.h
9  namespace YourName
10 {int iNum = 30000;
11  int Add(int iNum) {iNum = ::iNum + iNum; return iNum;}}
12 void main(void)
13 { cout << MyName::Add(10)+::iNum << endl;
14   cout << MyName::Add(10)+iNum << endl;
15   namespace N2 = YourName;
16   cout << N2::Add(10)+N2::iNum << endl;
17   using namespace YourName;
18   cout << Add(10)+MyName::iNum << endl;
19   cout << MyName::Add(10)+iNum << endl;
20 }
21

```

解决... 类视图 属性...

输出

显示输出来源(S): 生成

```

1>----- 已启动全部重新生成: 项目: huangtest, 配置: Debug Win32 -----
1>正在删除项目“huangtest”(配置“Debug|Win32”)的中间文件和输出文件
1>正在编译...
1>huangtest.cpp
1>f:\教学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\huangtest\huangtest.cpp(19) : error C2872: “iNum”: 不明确的符号
1>    可能是“f:\教学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\huangtest\huangtest.cpp(3) : int iNum”
1>    或      “f:\教学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\huangtest\huangtest.cpp(10) : int YourName::iNum”
1>生成日志保存在“file:///f:/教学文档/程序设计基础/tsinghua.c/2010xia/练习/huangtest/huangtest/Debug/BuildLog.htm”
1>huangtest - 1 个错误, 0 个警告
===== 全部重新生成: 成功 0 个, 失败 1 个, 跳过 0 个 =====

```



## 11.2 使用命名空间成员的方法

□问题：为什么在我们前面写程序时，都需要加上

```
using namespace std;
```

□标准C++库的命名空间

标准C++头文件中函数、类、对象和类模板是在命名空间std中定义。因此，在程序中用到C++标准库时，需要使用std作为限定。在文件开头加入：

```
using namespace std;
```

这样，在std中定义和声明的所有标识符在本文件中都可以作为全局量来使用。但是应当绝对保证在程序中不出现与命名空间std的成员同名的标识符。

## 命名空间使用示意图

```
//header1.h  
namespace ns1 {  
    ...  
}
```

```
//header2.h  
namespace ns2 {  
    class Student {  
        ...  
    };  
}
```

```
//header3.h  
namespace ns3 {  
    ...  
}
```

```
//1.cpp  
#include "header1.h"  
using namespace ns1;  
//使用命名空间ns1内的所有  
//成员,并保证本文件不出现  
//与ns1内命名冲突的成员
```

```
//2.cpp  
#include "header2.h"  
using ns2::Student;  
//使用命名空间ns2内的成员  
//类Student,并保证本文  
//件不出现与Student命名  
//冲突的成员
```

```
//3.cpp  
#include "header3.h"  
using namespace ns3;  
//使用命名空间ns3内的所有  
//成员,并保证本文件不出现  
//与ns3内命名冲突的成员
```

## 第11次作业

- 本次作业只有1道必做题。在笔试之前交。
- 要求：在第10次作业第1题的基础上
  - (1) 增加一个文件打开或者读写异常处理类，实现文件访问的异常处理功能，即当文件打开不或读写成功时，程序将抛出异常信息，由主程序通过屏幕告诉用户错误原因；
  - 当然，还可以自由设计，处理其他方面的异常；
  - (2) 如果将第10次作业中的`use namespace std`注释掉，则整个程序该如何修改？

## 笑话一则：测试你是否适合当程序员（例外处理思维能力）

- 师问：“树上有10只鸟，猎人开枪打死了1只，还剩几只？” 生：“是无声手枪，还是其它没有声音的枪？”
- 师：“不是无声手枪，也不是其它没有声音的枪。” 生：“枪声有多大？”
- 师：“80-100分贝” 生：“那就是说，会震得耳朵疼？”
- 师：“是的” 生：“在那个地方，打鸟不犯法？”
- 师：“不犯” 生：“您确定那只鸟真的被打死啦？”
- 师：“确定，拜托，你只需告诉我还剩几只鸟就OK？” 生：“鸟里有没有聋子？”
- 师：“没有” 生：“其中有智力问题？就是呆傻到听到枪响都不知道要飞的？”
- 师：“没有，智商都在200以上！” 生：“有没有关在笼子里？”
- 师：“没有” 生：“有没有残疾或饿得飞不动的鸟？”
- 师：“没有，身体都倍棒！” 生：“算不算怀孕肚子里的小鸟？”
- 师：“都是公的。” 生：“都不可能怀孕？”
- 师：“我晕！绝对不可能！” 生：“打鸟的人有没有眼花？保证是10只？”
- 师：“10只” 生仍追问：“有没有傻到不怕死的？”
- 师：“都怕死” 生：“有没有是情侣的，一方被打中，另一个主动要陪着殉情的？”
- 师：“笨蛋！之前不是告诉你都是公的吗！” 生：“同性不可以相爱啊？”
- 师愤怒了：“…10只鸟的性取向都很正常！” 生：“会不会一射二鸟？”
- 师：“不会” 生：“一枪打仨呢？”
- 师：“不会” 生：“4呢？” 师：“更不会” 生：“5呢？”
- 师彻底崩溃：“尼玛，再说一遍，一枪只能打死1只！” 生：“…好吧，就是所有鸟都可以自由活动的？”
- 师：“不会，每只鸟都自由飞行” 生：“如果您的回答没有骗人话，” 满怀信心的说：“打死的鸟要是挂在树上没掉下来，那么就剩1只；如果掉下来，就1只不剩！”
- 终于等到学生的答案了，老师强忍着几乎倒地的晕眩感，颤抖地说：“你不用读小学了，直接去**当程序员**吧！”