例子1 复数类----所有函数写的类的内部

```
#include<iostream>using namespace std;template<typename T>cl
ass Complex{ firend ostream & operator<<(ostream &out, Compl
ex &c3) {  cout << "a:" << " + " << "b:" << c3.b << endl;  r
eturn out; }public: Complex(T a=0,T b=0); {  this->a = a;  t
his->b = b; } Complex  operator+(Complex &c2) {  Complex tmp
(a + c2.a, b + c2.b);  return tmp; } void printComplex() {
cout << "a:" << a << "b:" << b << endl;     } private: T  a;
 T  b;};//??????????,??<<  >>???????,???????,????????,??????
??void main(){//?????????????,??????,c++???????? Complex<int
>    c1(1,2); Complex <int>   c2(3, 4); Complex<int>    c3=c
1+c2; cout << c3 << endl; system("pause"); return;}
```

例子2 复数类----所有函数写的类的外部,但在一个cpp里

```
#include<iostream>using namespace std;template<typename T>cl
ass Complex{ firend Complex  MySub<<(Complex &c1, Complex &c
2) {  Complex tmp(a + c2.a, b + c2.b);  return tmp; } firend
 ostream & operator<< <T> (ostream &out, Complex &c3); publi
c: Complex(T a, T b)  Complex  operator+(Complex &c2)  void
printComplex();private: T  a; T  b;};template<typename T>//?
??????,???????Complex<T>::Complex(T a,T b){ this->a = a; thi
s->b = b;}template<typename T>void  Complex::printComplex(){
 cout << "a:" << a << "b:" << b << endl;}template<typename T
>Complex<T> Complex<T>::operator+(Complex<T> &c2)//??????+??
???{ Complex tmp(a + c2.a, b + c2.b); return tmp;}template<t
ypename T>ostream & operator<<(ostream &out, Complex<T>&c3)/
/???????????{ cout << "a:" << " + " << "b:" << c3.b << endl;
 return out;}void main(){ Complex<int>    c1(1, 2); Complex
<int>   c2(3, 4); Complex<int>    c3 = c1 + c2; cout << c3 <
< endl; system("pause"); return;}
```

归纳以上的介绍,可以这样使用.声明类模板

1)先写出一个实际的类.由于其语义明确,含义清楚,一般不会出错.

2)将此类中准备改变的类型名(如int 要改为char)改用一个自己指定的虚拟类型名字

3)在类声明前加入一行,格式为:

template<class 虚拟类型函数>//注意本行末尾无分号

4)用类模板定义对象使用以下形式:

类模板名<实际类型名>对象名

类模板名<实际类型名>对象名(实参列表)

如:

Compare<int> cmp;

Compare<int> cmp(3,4);

5)如果在类模板外定义成员函数,应该写成类模板形式:

template<class 虚拟类型函数>

函数类型 类模板名<虚拟函数参数>::成员函数名(参数形参列表)(.....)

关于类模板的几点说明

1)类模板的类型参数可以有一个或多个,每个类型前面都必须加class,

如:

template<class T1,class T2>

class someclass

{.....}

定义对象时分别带入实际的类型名,如;

someclass<int,double>obj;

2)和使用类一样,使用类模板时要注意其作用域,只能在其有效作用域内用它定义对象

.

3)模板可以有层
次,一个类模板可以作为基类,派生
出派生类的模板,有关这方面的知识实际应用比较少,感兴趣的可以自行查阅.

类模板中的ststic关键字

从类模板比例实例化的每个模板类都有自己的类模板数据成员,该模板类的所有对象共享一个

ststic数据成员.

和非模板类的ststic数据成员一样,模板类的ststic数据成员,也应该在文件范围定义和初始化.

每个模板类都有自己的类模板和ststic数据成员副本.

例子

```
#include<iostream>using namespace std;template<typename T>cl
ass AA{public: static T m_a;private:};class AA1{public: stat
ic int m_a;private:};template<typename T>int  AA1::m_a = 0;c
lass AA2{public: static char m_a;private:};char  AA2::m_a =
0;void main(){ AA<int> a1, a2, a3; a1.m_a = 10; a2.m_a++; a3
.m_a++; cout << AA<int>::m_a << endl; AA<char> b1, b2, b3; b
1.m_a = 'a'; b2.m_a++; b2.m_a++; cout << AA<char>::m_a << en
dl; //m_a??????????,?????m_a system("pause"); return;}
```

异常问题

一、为什么要有异常——WHY？

1.通过返回值表达错误

像malloc会返回0或1.

局部对象都能正确的析构

层层判断返回值，流程繁琐

例子：

```
#include <iostream>#include <cstdio>using namespace std;int
func3 (void) {FILE* fp = fopen ("none", "r");//fopen???????
NULL?if (! fp)return -1;// ...fclose (fp);return 0;}int func
2 (void) {if (func3 () == -1)return -1;// ...return 0;}int f
unc1 (void) {if (func2 () == -1)return -1;// ...return 0;}in
t main (void) {//???????if (func1 () == -1) {cout << "??????
????" << endl;return -1;}// ...cout << "??????????" << endl;
return 0;}
```

2.通过setjmp/longjmp远程跳转

一步到位进入错误处理，流程简单

局部对象会失去被析构的机会

例子：

```
#include <iostream>#include <cstdio>#include <csetjmp> //?c?
????using namespace std;jmp_buf g_env; //jmp????c??????????
??????????????????????????????,???????????????class A {pub
lic:A (void) {cout << "A??" << endl;}~A (void) {cout << "A??
" << endl;}};void func3 (void) {A a;FILE* fp = fopen ("none"
, "r");if (! fp)longjmp (g_env, -1); //???????????????g_env
??-1??????????main???setjmp???// ...fclose (fp);}void func2
(void) {A a;func3 ();// ...}void func1 (void) {A a;func2 ();
// ...}int main (void) {if (setjmp (g_env) == -1) { //??????
??????????genv?0????????func1()??????fun3??longjmp???????g_e
nv??1?????g_env??cout << "??????????" << endl;return -1;}fun
c1 ();// ...cout << "??????????" << endl;return 0;}
```

---

3.异常处理

局部对象都能正确的析构

一步到位进入错误处理，流程简单

---

二、异常的语法——WHAT？

1.异常的抛出

throw 异常对象;

异常对象可以是基本类型的变量，也可以是类类型的对象。

当程序执行错误分支时抛出异常。

2.异常的捕获

try {

可能抛出异常的语句块;

}

catch (异常类型1 异常对象1) {

处理异常类型1的语句块;

}

catch (异常类型2 异常对象2) {

处理异常类型2的语句块;

}

...

catch (...) {

处理其它类型异常的语句块;

}

异常处理的流程，始终沿着函数调用的逆序，依次执行右花括号，直到try的右花括号，保证所有的局部对象都能被正确地析构，然会根据异常对象的类型，匹配相应的catch分支，进行有针对性的错误处理。

```
???#include <iostream>#include <cstdio>using namespace std;c
lass A {public:A (void) {cout << "A??" << endl;}~A (void) {c
out << "A??" << endl;}};void func3 (void) {A a;FILE* fp = fo
pen ("none", "r");if (! fp) { //???????????throw?????throw??
???cout << "throw?" << endl;throw -1; //??????throw????????
?????cout << "throw?" << endl;}cout << "???????" << endl;//
...fclose (fp);}void func2 (void) {A a;cout << "func3()?" <<
 endl;func3 (); //???????????cout << "func3()?" << endl;//
...}void func1 (void) {A a;cout << "func2()?" << endl;func2
(); //?????????cout << "func2()?" << endl;// ...}int main (v
oid) {try {cout << "func1()?" << endl;func1 (); //????func1?
???a?????????func2????a?????????func3????a??????????throw???
-1???func3???func3??a????????func2?????func2?a????????func1?
????func1?a???????????try?????????????????????????????catch?
???“????”?cout << "func1()?" << endl;}catch (int ex) {if (ex
 == -1) {cout << "??????????" << endl;return -1;}}// ...cout
 << "??????????" << endl;return 0;}
```