2015-2016寒假学习

C++简单复习

在C++里面，true 和false可以和int 里面的0和1互换

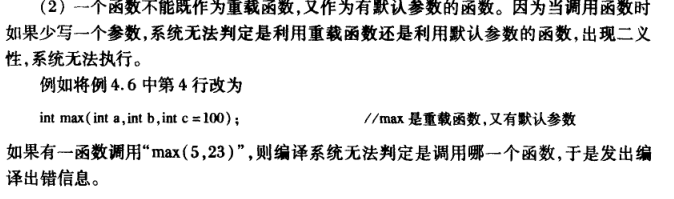
函数重载：函数名相同，而参数个数或者类型不同，可是程序还是很冗长，于是可以使用函数模板

C++里面函数模板：例如：

Template<typename T>

T 函数名（T 参数...）

一个函数不能同时作为重载函数和有默认参数的函数，否则会出现二义性



作用域在语句块上：

{int p;

}//p的有效范围

全局变量的缺点：

程序运行期间一直占用资源

程序的耦合性降低

降低清晰度

在C和指针里有内存和static的笔记再看一看

这里再学一学：

内存：程序区，静态存储区，动态存储区

内部函数：static 返回值类型 函数名（形参表）

不同文件的同名函数就互不干扰，只能被本文件的函数所调用

外部函数：extern 返回值类型 函数名（形参表）

如果没有extern默认为外部函数

接下来去看一看C++里面面向对象的探索

类是抽象的，不占内存，对象是具体的，占用存储空间

如果不加限定符，C++会默认为private

私有成员函数只能被本类中的其他成员函数所调用，而不能被类外调用，成员函数可以访问本类中的任何成员，

在类外定义成员函数方法：

返回值类型 类名：：函数名（）{函数体}

成员函数都不占用对象的存储空间，

引用对象成员语法：对象名.成员名 成员可以是函数，也可以是变量

也可以通过指针来存储对象的地址，这样指针就可以间接操作类里面的数据了

可以通过同一个类的对象的引用来访问对象里面的成员

软件工程最基本的原则就是将接口与实现分离

当改变数据时只改变类里面，而对外接口不会改变，就像iphone6升级为iphone6s，数据线还是一样的

类本身是一种抽象类型。不占用存储空间，无法容纳数据

构造函数（方法）

在定义对象的同时自动执行

不同对象可以进行不同的初始化，这就对构造函数进行重载了

构造函数名（类型1 形参1，类型2 形参2…）

这样定义对象时：

对象1（实参1，实参2…）

对象2（实参1，实参2…）

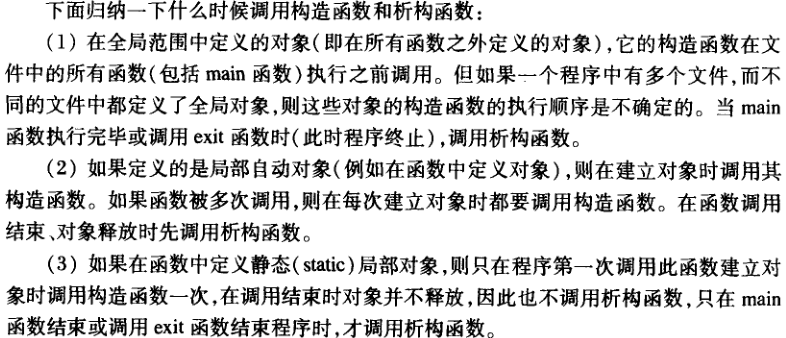
可以这样：构造函数名（类型 形参值=固定数值）；定义默认参数的构造函数，往往在类的里面去定义默认参数的构造函数

在声明对象时注意它可能有多个构造函数而出现歧义性

当main函数结束时才调用static局部对象的析构函数，析构函数用于撤销对象占用内存之前完成一些清理工作，它代表着对象生命期的结束，定义的方法：

~类名（）{函数体}

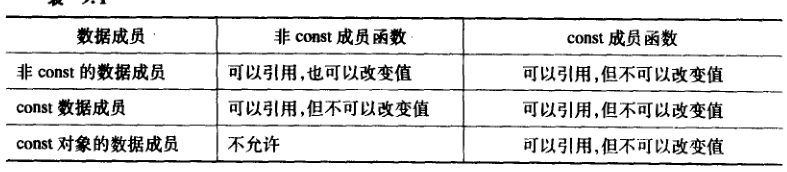
先构造的对象后析构，后构造的对象先析构，类似一个栈的存储结构



定义对象数组的方法： 类名 对象名[n]={

类名（参数1），类名（参数2）};分别调用两个构造函数

如果声明为const对象，只能调用const类型的函数，命名此函数的方法：返回值类型 函数名(参数…)const; 函数不允许修改对象的数值，被const声明的类数据只能用参数初始化表进行初始化，常成员函数可以引用任何数据成员，const数据成员可以被const成员函数引用，也可以被非const的成员函数引用



常成员函数不能调用另一个非const成员函数

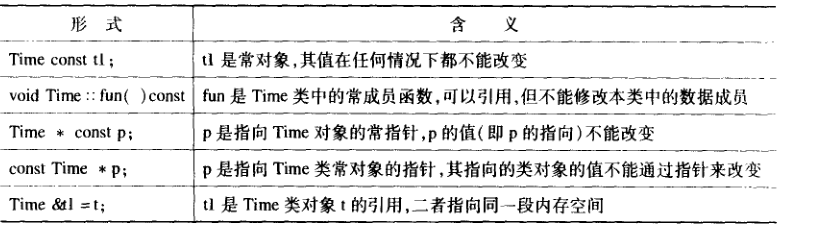
让一个指针始终指向同一个对象就要用到常指针，类名 \*const 指针变量名=&对象名，

可以把常指针当做为实参传递给函数。

Const 类型名 \*指针变量

Const指针可以指向任意变量，但是不能通过这个指针改变这个变量的数值

注意如果一个变量为const，只能用const指针来指向它



用new建立的动态对象一般不是用对象名，而是用指针访问，主要通过来建立和操作链表

建立一个动态对象方法： 类名 \*指针名=new 类名（初值表）;

删除一个动态对象：delete 指针名

建立和删除动态对象数组：

类名 \*指针名 =new类名[对象个数];

delete[]指针名;

对象之间赋值时一方不能含有动态分配的数据

对象的复制是一个无中生有的过程，具体方法：类名 对象2（对象1）；

其中自动调用复制构造函数

静态数据成员为所有对象共用，它只占一份空间，它的空间独立于所有对象之外，不占用对象的空间，它只能在类外初始化，静态数据成员初始化的方法：

数据类型 类名：：静态数据成员名=初值；

（不用加static）

通过类名引用的方法：类名：：静态成员名 同理静态成员函数也是这么调用

也可以用对象来引用

非静态成员函数有this指针，**静态成员函数没有this指针**，静态成员函数不能访问本类中的非静态成员，静态成员函数可以直接访问静态数据成员，

友元函数的举例：

#include<iostream>

using namespace std;

class Time{

public :

Time(int,int,int);

friend void display(Time &);

private :

int hour,min,sec;

};

Time::Time(int h,int m,int n){

hour=h;

min=m;

sec=n;

}

void display(Time &t)//不需要加类名和：：

{

cout<<t.hour<<t.min<<t.sec<<endl;//必须通过对象名来引用数据

}

int main(){

Time t1(10,13,12);

display(t1);

}

接下来看一看友元成员函数：

#include<iostream>

using namespace std;

class Date;//对类提前声明

class Time{

public:

Time(int,int,int);

void display(Date &);

private:

int hour;

int min;

int sec;

};

class Date {

public :

Date(int,int,int);

friend void Time::display(Date &);

private:

int mon;

int day;

int year;

};

Time::Time(int h,int m,int s)

{

hour=h;

min=m;

sec=s;

}

void Time::display(Date &d)

{

cout<<d.mon<<d.day<<d.year<<endl;

cout<<hour<<min<<sec<<endl;

}

Date::Date(int m,int d,int y)

{

mon=m;

day=d;

year=y;

}

int main()

{

Time t1(10,13,56);

Date d1(12,23,2016);

t1.display(d1);

}

友元类B中的所有函数都是A类中的友元函数，可以访问A类的所有成员，在A类的定义体中定义方法： friend 类名；可是A类无法访问B类的私有数据成员，友元关系不可以传递，

类模板：template<class 虚拟类型参数>

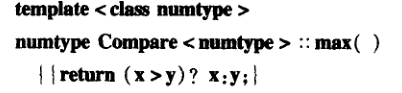


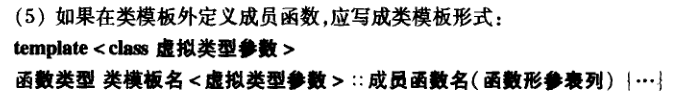
声明类模板对象的方法：

类名<实际数据类型> 对象名（参数）；



在类模板外面定义成员函数的方法：





运算符的重载：（JAVA里面没有运算符的重载）

函数类型 operator 运算符名称（形参列表）

{函数体}

复数相加：

#include<iostream>

using namespace std;

class Comp {

public:

Comp() { real = 0; imag = 0; }

Comp(int r,int i)

{

real = r;

imag = i;

}

Comp operator +(Comp &c2);

void display();

private:

int real;

int imag;

};

Comp Comp::operator+(Comp&c2) {

Comp c1;

c1.real = real + c2.real;

c1.imag = imag + c2.real;

return c1;

}

void Comp::display() {

cout << "(" << real << "," << imag <<"i"<< endl;

}

int main() {

Comp c1(3, 4);

Comp c2(5, -10);

Comp c3;

c3 = c1 + c2;

cout << "c1=";

c1.display();

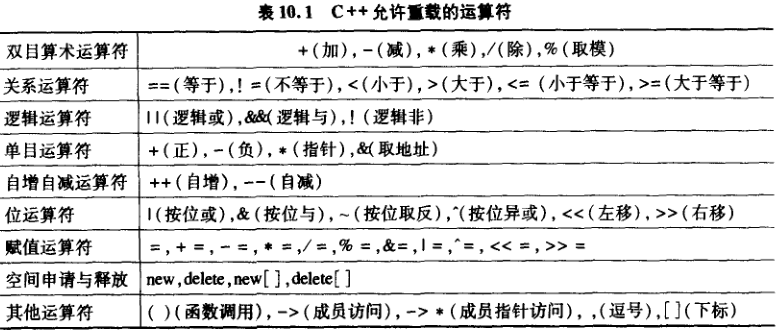
cout << "c2=";

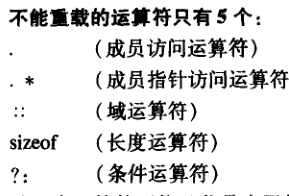
c2.display();

cout << "c3=" << endl;

c3.display();

}





重载不能改变运算符的优先级和结合性，不能改变运算对象的个数，重载运算符必须和用户自定义的对象一起使用，参数必须至少为一个对象（或类对象的一个引用）

运算符重载后也要和原来的用法类似，

运算符重载函数可以作为类成员函数和友元函数

#include<iostream>

using namespace std;

class Comp {

public:

Comp() { real = 0; imag = 0; }

Comp(int r,int i)

{

real = r;

imag = i;

}

friend Comp operator +(Comp &c1,Comp &c2);

void display();

private:

int real;

int imag;

};

Comp operator+(Comp&c1,Comp &c2) {

return Comp(c1.real + c2.real, c1.imag + c2.imag);

}

void Comp::display() {

cout << "(" << real << "," << imag <<"i)"<< endl;

}

int main() {

Comp c1(3, 4);

Comp c2(5, -10);

Comp c3;

c3 = c1 + c2;

cout << "c1=";

c1.display();

cout << "c2=";

c2.display();

cout << "c3=" << endl;

c3.display();

}

继承

声明继承的方法：

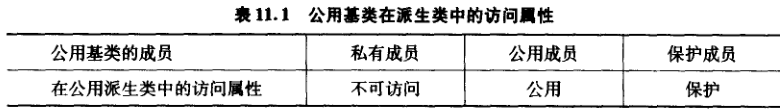
class 子类名:继承方式 父类名

{子类成员

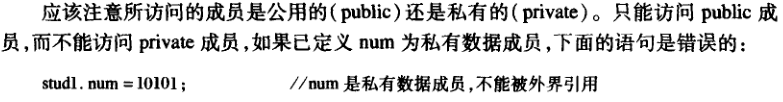
};

C++默认为private继承

公有继承：

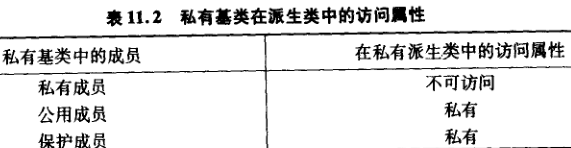


私有数据成员只能被同一类中的成员函数访问，共用成员可以被外界访问，

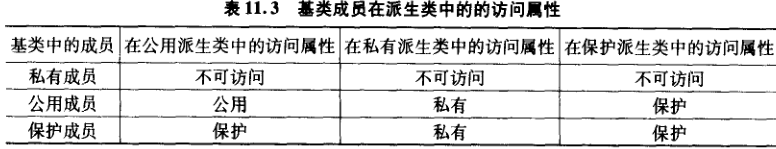


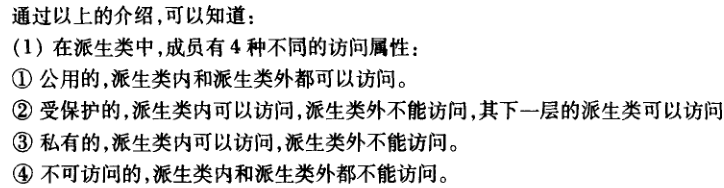
父类成员函数只能访问父类成员， 不能访问子类成员，子类外可以访问子类的共有成员，不能访问子类的私有成员

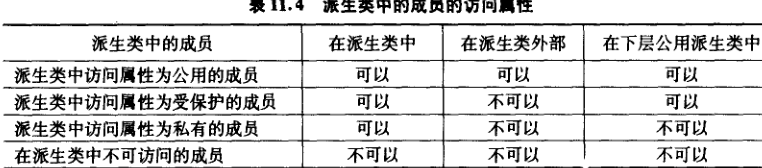
私有继承：



保护继承：







子类构造函数的做法：

子类构造函数名（总参数列表）：父类构造函数名（参数列表）//参数列表不包含数据类型

{子类中新增加的数据成员初始化语句}

子对象：在子类中定义的对象，看下面的程序：

#include<iostream>

#include<string>

using namespace std;

class Student {

public:

Student(int n, string nam)

{

num = n;

name = nam;

}

void display()

{

cout << "num:" << num << endl << "name" << name << endl;

}

protected:

int num;

string name;

};

class Stu1 :public Student

{

public:

Stu1(int n, string nam, int n1, string nam1, int a, string ad) :Student(n, nam), mon(n1, nam1)

{

age = a;

addr = ad;

}

void show()

{

cout << "This stu is:" << endl;

display();

cout << "age:" << age << endl;

cout << "address:" << addr << endl;

}

void showmon()

{

cout << "mon is:" << endl;

mon.display();

}

private:

Student mon;

int age;

string addr;

};

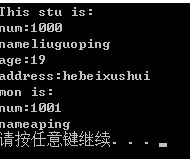
int main() {

Stu1 stu1(1000, "liuguoping", 1001, "aping ", 19, "hebeixushui");

stu1.show();

stu1.showmon();

}



子类构造函数包含三个部分：

对父类数据，子类数据，子对象数据初始化

接下来看一看派生类的构造函数：

#include<iostream>

#include<string>

using namespace std;

class Stu {

public:

Stu(int n, string nam)

{

num = n;

name = nam;

}

void display() {

cout << "num=" << num << endl;

cout << "name=" << name << endl;

}

protected:

int num;

string name;

};

class Stu1 :public Stu {

public :

Stu1(int n, string nam, int a) :Stu(n, nam)

{

age = a;

}

void show() {

display();

cout << "age=" << age << endl;

}

private:

int age;

};

class Stu2 :public Stu1 {

public :

Stu2(int n, string nam, int a, int s):Stu1(n, nam, a)

{

score = s;

}

void showall() {

show();

cout << "score=" << score << endl;

}

private:

int score;

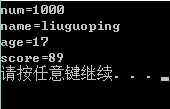
};

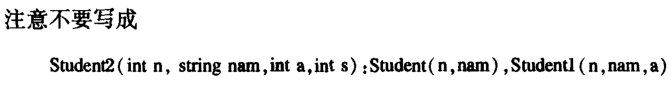
int main() {

Stu2 stu(1000, "liuguoping ", 17, 89);

stu.showall();

}





先初始化父类构造函数，再依次往下去初始化

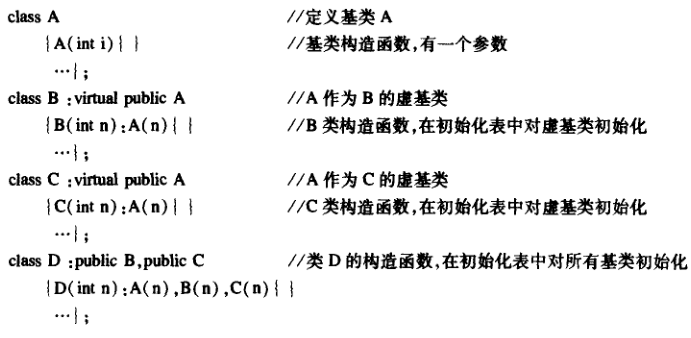
析构函数和构造执行顺序相反

虚基类：

在声明子类的同时声明继承方式，具体声明方法：

class 子类名：virtual 继承方式 基类名（父类名）

虚基类的构造函数书写方法：



规定在最后的子类中对直接父类初始化，还要给虚基类初始化

编译系统只对最后的构造函数调用

下面看一看这个程序：

#include<iostream>

#include<string>

using namespace std;

class Person {

public:

Person(string nam, char s, int a)

{

name = nam;

sex = s;

age = a;

}

protected:

string name;

char sex;

int age;

};

class Tea :virtual public Person {

public :

Tea(string nam, char s, int a, string t) :Person(nam, s, a) {

tee = t;

}

protected:

string tee;

};

class Stu :virtual public Person {

public:

Stu(string nam, char s, int a, double sc) :Person(nam, s, a) {

score = sc;

}

protected:

double score;

};

class Grad :public Tea, public Stu {

public:

Grad(string nam, char s, int a, string t, double sc, double w) :Person(nam, s, a), Tea(nam, s, a, t), Stu(nam, s, a, sc) {

wage = w;

}

void show() {

cout << "name=" << name << endl;

cout << "age=" << age << endl;

cout << "sex=" << sex << endl;

cout << "score=" << score << endl;

cout << "tee=" << tee << endl;

cout << "wage=" << wage << endl;

}

private:

double wage;

};

int main() {

Grad grad1("liuaping", 'f', 20, "student", 90, 123);

grad1.show();

}

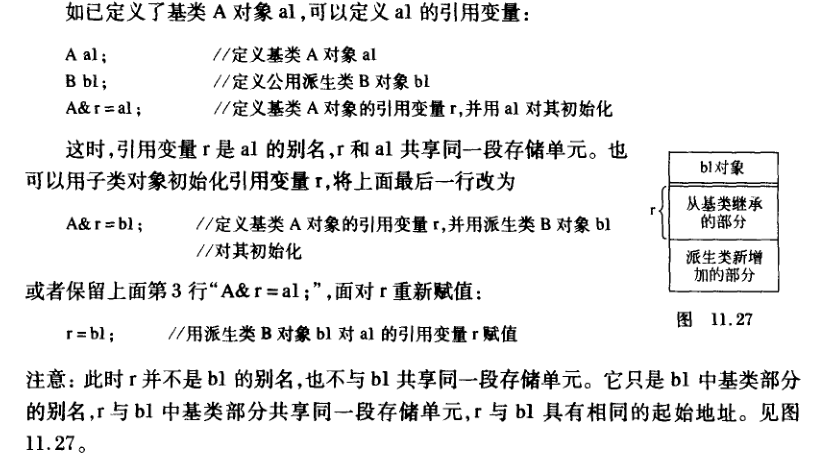


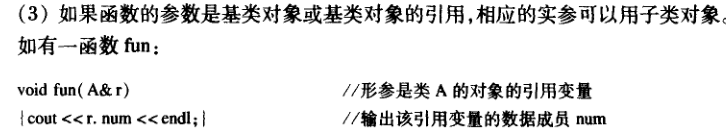
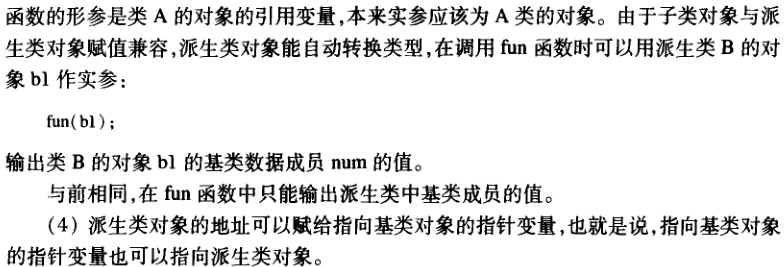
为避免讨厌的二义性问题，我伟大的Java根本没有多重继承，只支持单继承

可以通过子类来给父类赋值，，给数据赋值，对成员函数不管

赋值时舍弃子类新增加的成员

子类对象可以替代父类对象向父类对象的引用进行赋值



通过指向基类对象的指针，只能访问子类中的父类成员，不能访问子类新增加的成员

如：

#include<iostream>

#include<string>

using namespace std;

class Stu {

public:

Stu(int, string, double);

void display();

private:

int num;

string name;

double score;

};

Stu::Stu(int n, string nam, double s) {

num = n;

name = nam;

score = s;

}

void Stu::display() {

cout << "num=" << num << endl;

cout << "name=" << name << endl;

cout << "score=" << score << endl;

}

class Grad :public Stu {

public:

Grad(int, string, double, double);

void display();

private:

double pay;

};

Grad::Grad(int n, string nam, double s, double p) :Stu(n, nam, s) {

pay = p;

}

void Grad::display() {

Stu::display();

cout << "pay=" << pay << endl;

}

int main() {

Stu stu1(1001, "aping,", 90.00);

Grad grad1(2001, "Kastelo", 99.9, 99.999);

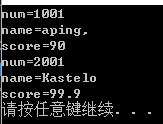
Stu \*p = &stu1;

p->display();

p = &grad1;

p->display();

}



可以通过虚函数和多态性解决

（C++东西太多了，博大精深）

继承与组合以后用到再学

**多态性**

#include<iostream>

#include<string>

using namespace std;

class Point {

public:

Point(double x = 0, double y = 0);

void setPoint(double,double);

double getx()const { return x; }

double gety()const { return y; }//只允许函数引用类数据

friend ostream & operator <<(ostream &, const Point &);

protected:

double x, y;

};

class Circle :public Point{

public:

Circle(double a = 0, double b = 0, double r = 0);

void setbanjing(double);

double getbanjing()const;

double area()const;

friend ostream & operator <<(ostream &, const Circle &);

protected:

double radius;

};

class Cylinder :public Circle {

public:

Cylinder(double x = 0, double y = 0, double r = 0, double h = 0);

void setHigh(double);

double getHigh()const;

double area()const;

double vloment()const;

friend ostream & operator <<(ostream &, const Cylinder &);

protected:

double high;

};

Point::Point(double a, double b) {//初始化

x = a;

y = b;

}

void Point::setPoint(double a,double b) {//赋值

x = a;

y = b;

}

ostream &operator <<(ostream &output, const Point &p) {

output << "{" << p.x << endl << p.y << endl;

return output;

}

Circle::Circle(double a, double b, double r) :Point(a, b) {

radius = r;

}

void Circle::setbanjing(double r) {

radius = r;

}

double Circle::getbanjing()const {

return radius;

}

double Circle::area()const {

return 3.14\*radius\*radius;

}

ostream &operator <<(ostream &output, const Circle&c) {

output << "Center=" << c.x << endl << c.y << endl << "r=" << c.radius << endl << c.area() << endl;

return output;

}

Cylinder::Cylinder(double a, double b, double r, double h) :Circle(a, b, r), high(h) {};

void Cylinder::setHigh(double h) {

high = h;

}

double Cylinder::getHigh()const {

return high;

}

double Cylinder::area()const {

return 2 \* Circle::area() + 2 \* 3.14\*radius\*high;

}

double Cylinder::vloment()const {

return Circle::area()\*high;

}

ostream & operator <<(ostream &output, const Cylinder &cy) {

output << "center=" << cy.x << endl << cy.y << endl << cy.radius << cy.high << endl <<

cy.area() << endl << cy.vloment() << endl;

return output;

}

int main() {

Point p(3.1, 6.8);

cout << "x=" << p.getx() << "y=" << p.gety() << endl;

p.setPoint(8.8, 9.2);

cout << "p的数值为：" << p << endl;

Circle c(3.5, 6.4, 8.8);

cout << c.getx() << endl << c.gety() <<endl<< c.getbanjing()<<endl << c.area() << endl;

c.setbanjing(8.9);

c.setPoint(10, 10);

cout << "new circle" << c << endl;

Point &pr = c;

cout << "pr" << pr << endl;//这里应该不会出现子类的数据！

Cylinder cy1(3.1, 7.3, 3.3, 10);

cout << cy1.getx() << endl << cy1.gety() << endl << cy1.getbanjing() << cy1.getHigh() << cy1.area() << cy1.vloment() << endl;

cy1.setHigh(15);

cy1.setbanjing(2.5);

cy1.setPoint(1, 1);

cout << cy1 << endl;

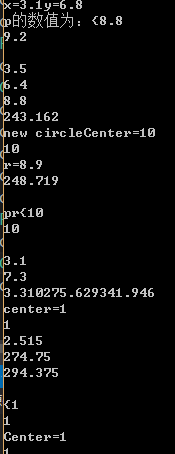
Point &pr1 = cy1;

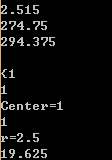
cout << pr1;

Circle &c1 = cy1;

cout << c1 << endl;

}

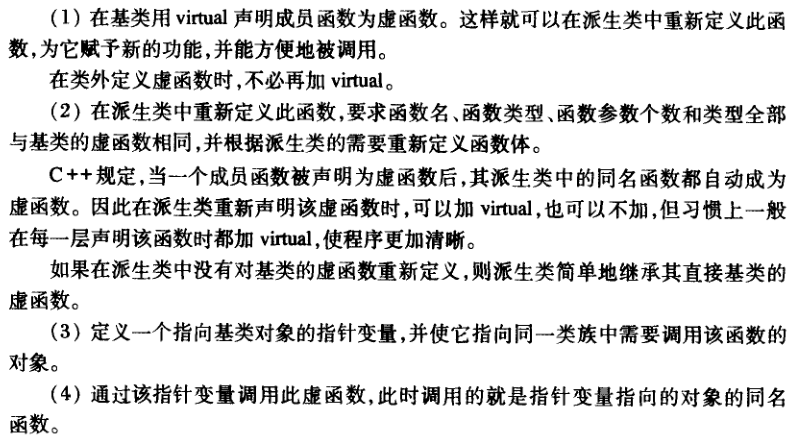




虚函数：

由虚函数实现的动态多态性是指同一类族中不同类的对象，对同一函数调用做出的不同响应

虚函数的使用方法：



#include<iostream>

#include<string>

using namespace std;

class Stu {

public:

Stu(int, string, double);

virtual void display();

protected:

int num;

string name;

double score;

};

class Grad :public Stu {

public:

Grad(int, string, double, double);

virtual void display();

private:

double pay;

};

Stu::Stu(int n, string nam, double s) { num = n, name = nam, score = s; }

void Stu::display() { cout << num << endl << name << endl << score << endl; }

void Grad::display() { cout << num <<endl<< name << endl<<score << endl << pay; }

Grad::Grad(int n, string nam, double s, double p) :Stu(n, nam, s), pay(p) {}

int main() {

Stu s1(1001, "liu", 99.0);

Grad gra1(100000, "guoping", 989.9, 888);

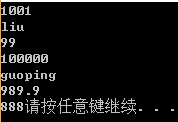
Stu \*p = &s1;

p->display();

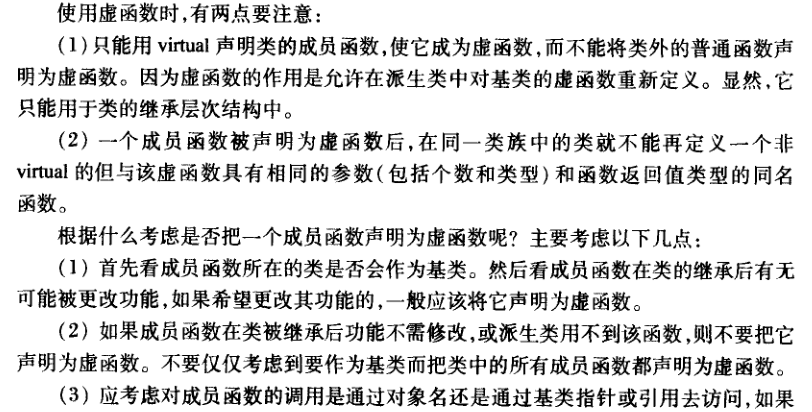
p = &gra1;

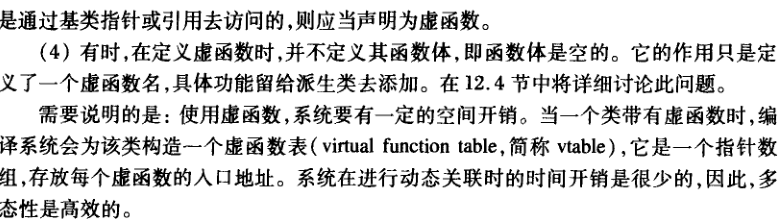
p->display();

}

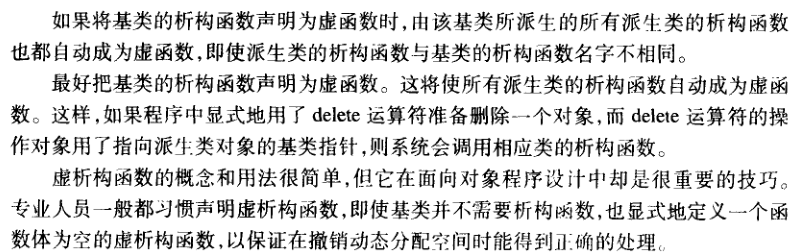


同一消息在不同的对象下有不同响应就是多态性



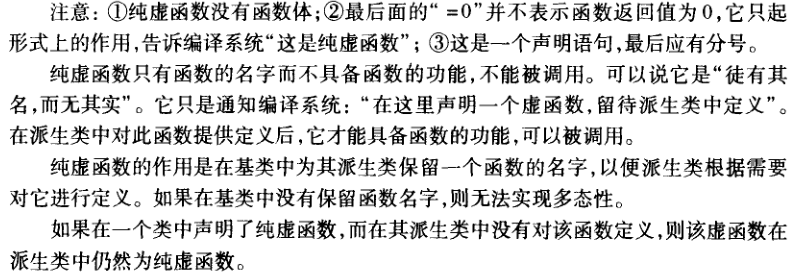






纯虚函数是在声明虚函数时被初始化为0的函数，方法：

Virtual 函数类型 函数名 （参数列表）=0；



凡是包含纯虚函数的类都是抽象类

抽象类为子类提供了接口

#include<iostream>

#include<string>

using namespace std;

class Sharp {

public:

virtual double area()const { return 0; }

virtual double vloment()const { return 0; }

virtual void sharpname()const = 0;

};

class Point :public Sharp {

public:

Point(double = 0, double = 0);

void setpoint(double, double);

double getx() const { return x; }

double gety()const { return y; }

virtual void sharpname()const { cout << "point:"; }

friend ostream &operator<<(ostream &, const Point &);

protected:

double x;

double y;

};

class Circle :public Point {

public:

Circle(double x = 0, double y = 0, double r = 0);

void setradius(double);

double getradius()const;

virtual double area()const;

virtual void sharpname()const { cout << "circle:"; }

friend ostream &operator<<(ostream &, const Circle &);

protected:

double radius;

};

class Cylinder :public Circle {

public:

Cylinder(double x = 0, double y = 0, double r = 0, double h = 0);

void sethigh(double);

virtual double area()const;

virtual double vloment()const;

virtual void sharpname()const { cout << "Cylinder:"; }

friend ostream & operator<<(ostream &output, const Cylinder &);

protected:

double high;

};

Point::Point(double a, double b) {

x = a;

y = b;

}

void Point::setpoint(double a, double b) {

x = a;

y = b;

}

ostream & operator<<(ostream &output, const Point &p) {

output << p.x << endl << p.y << endl;

return output;

}

Circle::Circle(double a, double b, double r) :Point(a, b) {

radius = r;

}

void Circle::setradius(double r){

radius = r;

}

double Circle::getradius()const { return radius; }

double Circle::area()const { return 3.14\*radius\*radius; }

ostream & operator<<(ostream &output, const Circle &c) {

output << c.x << endl << c.y << endl<<c.radius<<endl;

return output;

}

Cylinder::Cylinder(double a, double b, double r, double h) :Circle(a, b, r), high(h) {}

void Cylinder::sethigh(double h) {

high = h;

}

double Cylinder::area()const {

return 2 \* Circle::area() + 2 \* 3.14\*radius\*high;

}

double Cylinder::vloment() const{

return Circle::area()\*high;

}

ostream & operator<<(ostream &output, const Cylinder &cy) {

output << cy.x << endl << cy.y << endl<<cy.radius<<endl<<cy.high<<endl;

return output;

}

int main() {

Point point(8.8, 6.6);

Circle cir(2.4, 2.5, 6.6);

Cylinder cc(1.1, 2.2, 3.3, 4.4);

point.sharpname();

cout << point << endl;

cir.sharpname();

cout << cir << endl;

cc.sharpname();

cout << cc << endl;

Sharp \*s;

s = &point;

s->sharpname();

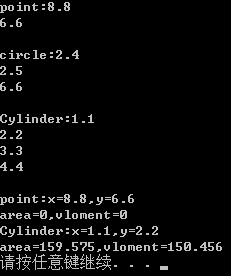
cout << "x=" << point.getx() << ",y=" << point.gety() << endl << "area=" << s->area() << ",vloment=" << s->vloment() << endl;

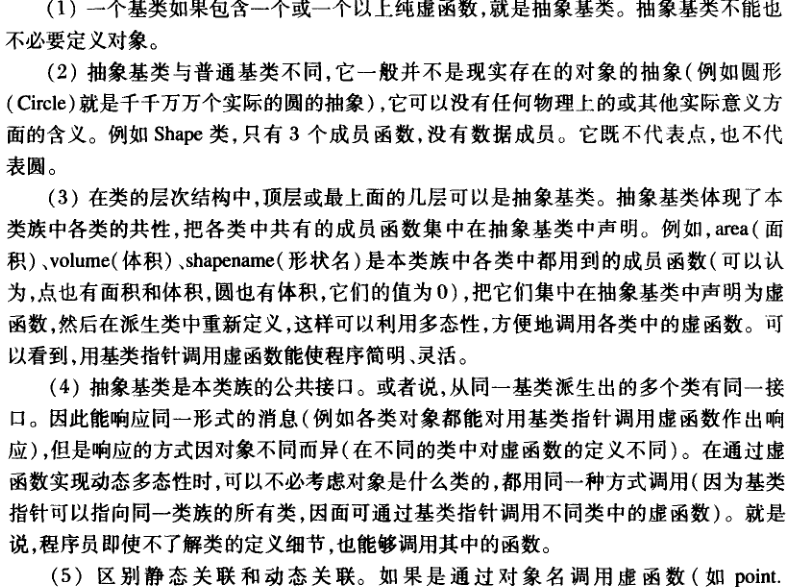
s = &cc;

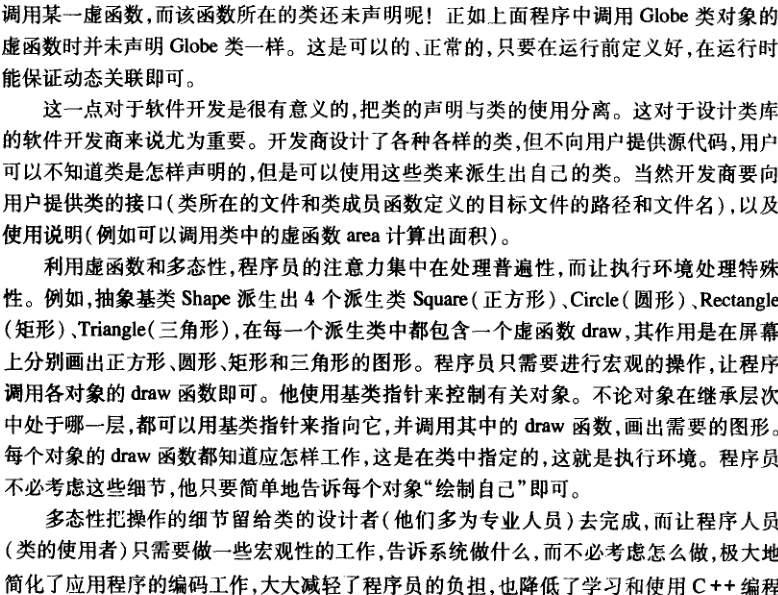
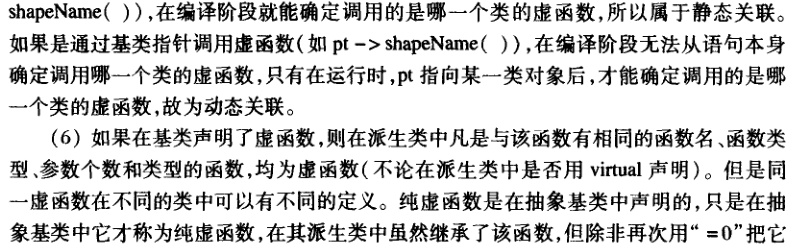
s->sharpname();

cout << "x=" << cc.getx() << ",y=" <<cc.gety() << endl << "area=" << s->area() << ",vloment=" << s->vloment() << endl;

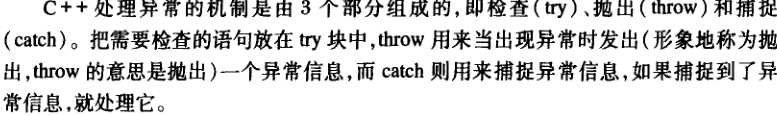
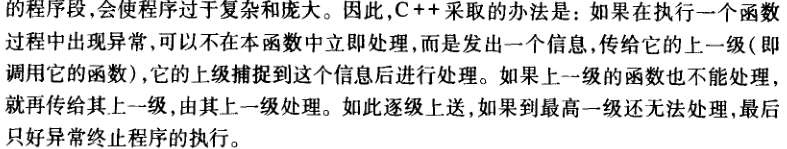
}

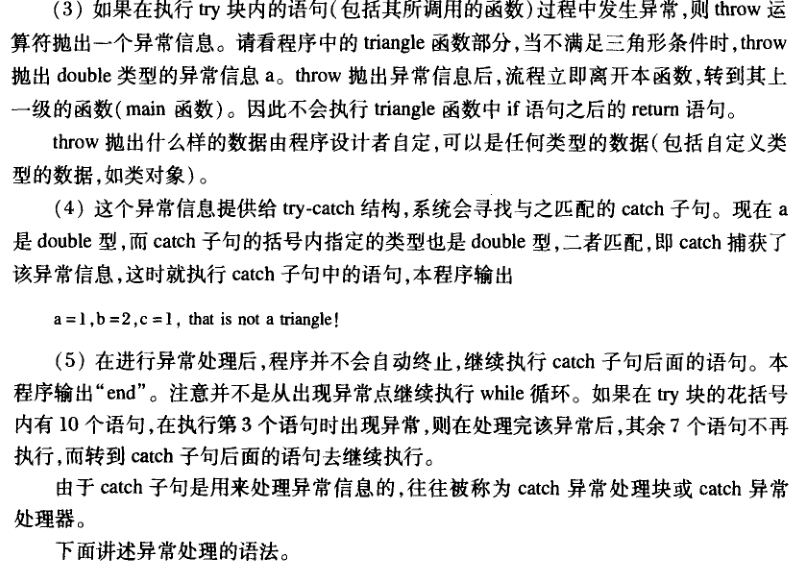


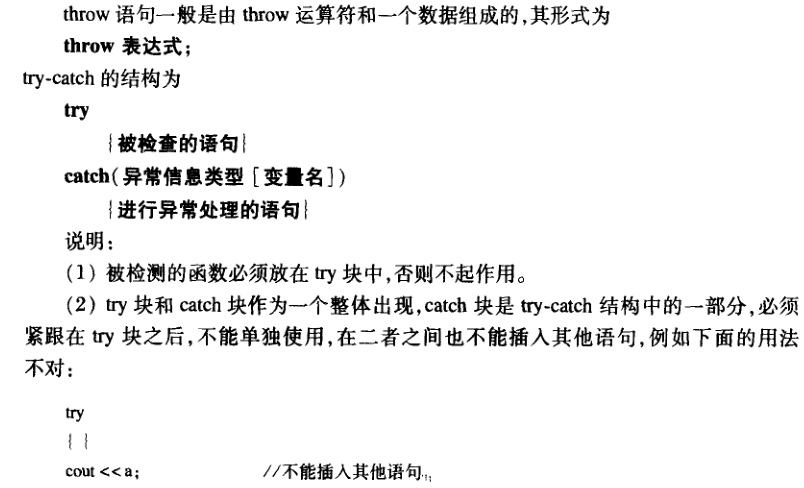




多态性是开启继承功能的钥匙

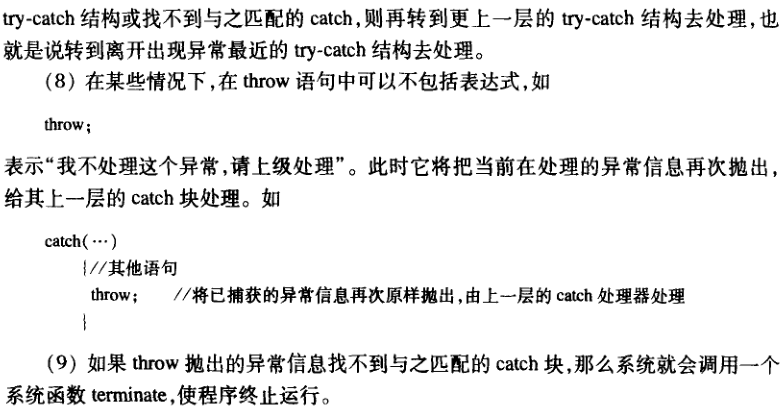


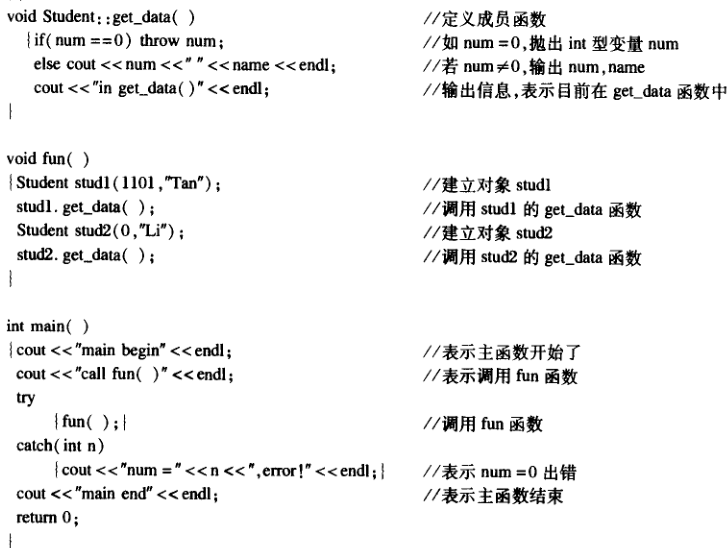
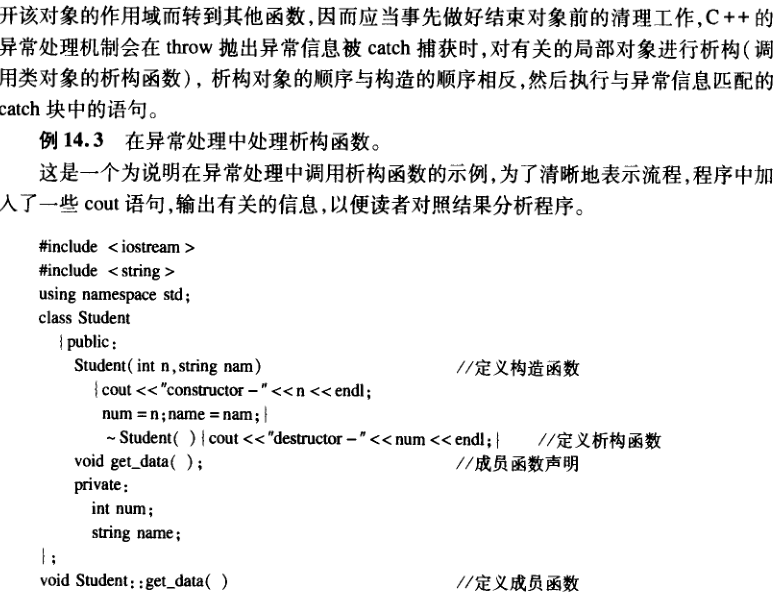
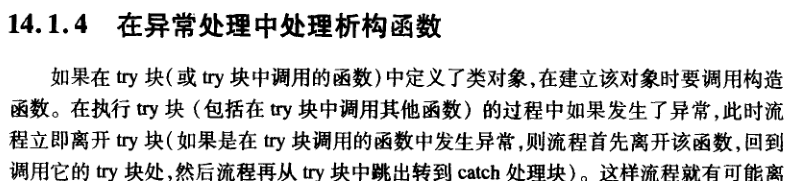
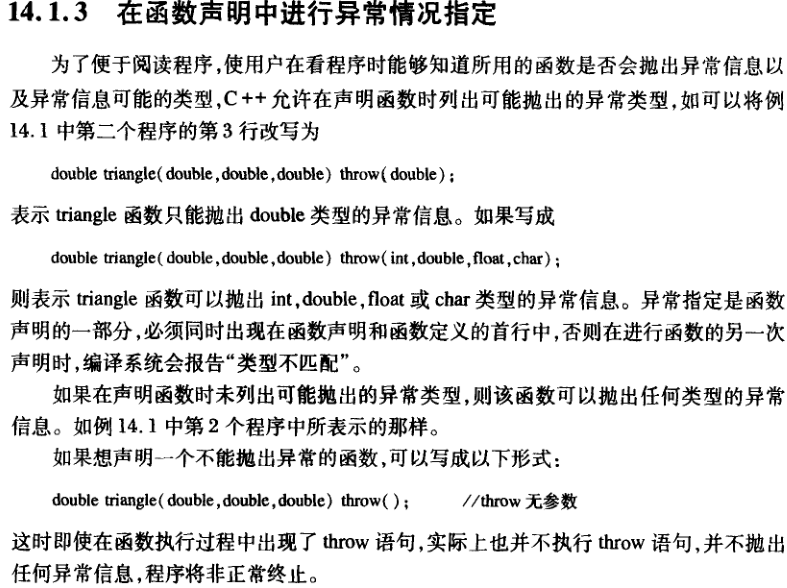












有的用到时再学，跟大神很远啊，任重而道远吧，努力根本不够，与时间赛跑！