面向对象变成特点2.

了解各种域:

1.类域

C 作用域 ; C++引入类的概念

类：类型

类中各种作用域

class Point{

int x; int y;

public:

void show(); }; Point a;

2.作用域的理解

名称（标识符）具有作用域

变量名 变量作用域 ---> 变量名作用域

函数名

结构体名

别名

作用域：可使用的区域

3.作用域分类

根据位置，决定了名称的作用域

1).块作用域

通常，标识符的声明位置位于块中

这个名称仅能在这个块中使用

名称的使用必须在声明之后

{ int a; a = 20; }

通常：局部变量（静态 非静态）

2).文件作用域

通常，标识符的声明位置位于全局区（通常：main的上面）;这个名称仅能在整个文件中使用

通常：全局变量，别名，函数名，结构体名 。。。。。。

typedef unsigned int u32;

3).函数作用域

标签名只能放在函数中，不能放在全局区

4).函数原型作用域

int add(int a = 0, int b = 0);

4.类域

class Point // 类名作用域

{ int x; // 成员名作用域

int y;

void put(); };

Point ss; // 对象名作用域

1).类名作用域: 取决于声明的位置

1）全局类 <外部类> 通常

2）局部类 <内部类>

3）类中类 <嵌套类>

补充：类中通常会声明枚举型，用于表示该类中使用的特殊含义

2).成员名作用域

类作用域：整个类可以使用

3).对象名作用域

取决于声明的位置;道理同C语言普通变量

1. 运算符重载

1. 运算符重载:函数支持重载;运算符函数重载 ;运算符 == 函数

+ add

* sub

有些函数的功能与运算符的效果一致;此时，使用运算符来表示比使用函数名,更方便，更好理解如： 12 + 34; add(12, 34);

用户可以自定义大量的类；数据类；需要计算的类，才会使用运算

2. 运算符重载的理解:运算符实现了某种功能（运算规则）;12 + 34 ; 函数实现了某种功能add(12, 34)

strcpy(str1, str2); str1 = str2;可以将运算符看做是一种特殊的函数用运算符：简单、明了;Point a;Point b;

a - b;每个运算符都会对应一个约定俗成的功能. + 相加,- 相减;重载：必须功能相同; 原来存在，使用相同的名称（函数名 、运算符号）12 + 34;12.34 + 45.67 默认规则 ;只支持：基本类型;int +(int , int );

3.运算符重载规则：运算符是一种通俗、直观的函数，比如：int x＝2+3;语句中的“＋”操作符，系统本身就提供了很多个重载版本：int operator + (int，int);double operator + (double,double);

1).可以重载的运算符有：

双目运算符+ - \* / %

关系运算符== != < > <= >=

逻辑运算符|| && +

单目运算符+ - \* &

自增自减运算符++ --

位运算符| & ~ ^ << >>

赋值运算符= += -= \*= /= %= &= |= ^= <<= >>=

空间申请和释放new delete new[] delete[]

其他运算符() -> ->\* , []

2).不能重载的运算符有：

成员访问符.

成员指针访问运算符.\*

域运算符::

长度运算符sizeof

条件运算符号?:

3).不能臆造并重载一个不存在的运算符，如@, #，$等。

4.以成员函数形式重载运算符

成员函数形式的运算符声明和实现与成员函数类似，首先应当在类定义中声明该运算符，声明的具体形式为：返回类型 operator 运算符（参数列表）;

既可以在类定义的同时定义运算符函数使其成为inline型，也可以在类定义之外定义运算符函数，但要使用作用域限定符“::”，类外定义的基本格式为：

返回类型 类名::operator 运算符（参数列表）{……….}

5.以友元函数形式重载运算符

1).用成员函数重载双目运算符时，左操作数无须用参数输入，而是通过隐含的this指针传入，这种做法效率比较高

2).此外，操作符还可重载为友元函数形式，这将没有隐含的参数this指针。对双目运算符，友元函数有2个参数，对单目运算符，友元函数有一个参数。

3).重载为友元函数的运算符重载函数的声明格式为：friend 返回类型operator 运算符 (参数表);

两种方法比较：对于绝大多数可重载操作符来说，两种重载形式都是允许的。但对下标运算符[] 、赋值运算符=、函数调用运算符()、指针运算符->，只能使用成员函数形式重载。

对于如下代码：

complex c1(1.0, 2.0), cRes;

cRes = c1 + 5; //#1

cRes = 5 + c1; //#2

* 友元函数形式重载的都是合法的，可转换成：

cRes = operator+(c1, 5); //#1 合法

cRes = operator+(5, c1); //#2 合法

* 但成员函数形式的重载，只有语句#1合法，语句#2非法

cRes = c1.operator(complex(5)); //#1 可能合法

cRes = 5.operator(c1); //#2 非法，5不会隐式转换成complex

补充：有些运算符重载可以放在类外，也可以放在类内

建议：放在类内以下几点好处：1）参数方便2）访问成员方便3）将支持嵌入到类中，类去什么地方，支持就去什么地方

有些运算符重载只能放在类外 << >>

有些运算符重载只能放在类内 = [] () ->

核心：左操作数 Point a; a.=(12);

a = 12; 12 + a a + 12 cout << a; cout.<<(a);

1）如果左操作数固定为自定义类型：只能放在类内

2）如果左操作数固定为其他类型：只能放在类外

正常情况下，复制构造函数，析构函数、赋值运算符重载函数都不用写

就使用默认提供的。一旦类内部出现堆区空间（通常：指针成员），必须自定义复制构造、析

构、赋值运算符重载函数

3.特殊的运算符重载

1）赋值运算符：只能使用成员形式

2）插入运算符： 只能使用非成员形式（普通 + 友元）>>和<<

3）自增运算符:1.前置++ ++a; 2.后置++ a++;Point a(3, 5); a++;

前置和后置区别：后置形参多一个int

6.不同类型数据间的转换

1.类型转换的理解:本质：指一个数据的类型发生了变化

struct date{ int y; int m; int d; };

struct time{ int h; int m; int s; };

int --> double double --> int

(int)(10 / 2.5)

1）自动转换 <隐式转换>

条件：当计算时，两个数据的类型不匹配

支持：只支持基本类型之间的转换 （整型 实型）

特例：char short 只要做运算，就被转换成 int

方向：1).容量小 --> 容量大 2).精度低 --> 精度高 3).整型：有符号 --> 无符号

2）强制转换 <显式转换>

核心：你希望哪个数据的类型改变，就强制让它变

通常：用于控制计算按照我们预计的效果去执行 类型不匹配的赋值

支持：只支持基本类型之间的转换 （整型 实型）

struct date struct sss

{ {

int y; int y;

int m; int m;

int d; int d;}; };

int --> date date --> int

Point -> int int -> Point

2. C++中的类型转换：

C++引入了类概念，自定义类可以转换为其他类型;其他类型可以转换为自定义类

添加支持:int --> Point Point --> int

1）基本类型 --> 自定义类型 支持：在自定义类提供对应的构造函数

int -> Point Point a; int n; a = Point(10); 10 --> Point

注意：构造函数的本质是为了创建对象 :间接的为基本数据类型转换提供了支持:导致使用时，基本类型转换不可控; explicit 关闭隐式转换支持

2）自定义类型 --> 基本类型 支持：在自定义类提供对应的类型转换函数

Point --> int Point a(3, 8); int n; n = a; int = Point Point --> int 、

转换构造函数：作用将一个其他类型的数据转换成一个类的对象。如:complex（double r）{real=r;imag=0;} //作用是将double型的参数转换成complex类的对象。

注：转换构造函数只能有一个参数。如果有多个参数，它就不是转换构造函数。

3.类型转换函数（类对象转换其他类型的数据）

C++提供类型转换函数，类型转换函数的作用是将一个类的对象转换成另一个类型的数据。

1）格式: 普通函数： 返回值类型 函数名（形参列表）;

构造函数：类名（形参列表）;

运算符函数：返回值类型 operator符号（形参列表）;

类型转换函数： operator类型(){…};

class Point{ operator int(); Point --> int };

n = a.operator int();

\\ 可以通过operator int()这种类似操作符重载函数的类型转换函数来实现由自定义类型向其他类型的转换。如将point类转换成int类型等。

在类中定义类型转换函数的形式一般为： operator 目标类型名();

有以下几个使用要点：1.转换函数必须是成员函数，不能是友元形式或普通函数。2.转换函数不能指定返回类型，但在函数体内必须用return语句以传值方式返回一个目标类型的变量。3.转换函数不能有参数

注意：转换构造函数和类型转换函数和运算符+重载函数写一起会出现二义性；要使用类型转换函数需删去运算符重载函数。

1. 继承和派生

概念理解：

1.1).继承:在C++可重用性是通过继承实现。继承就是基于一个类生成另一个类的对象，以便使后者拥有前者的某些成员，再加上它自己的一些成员。(也就是在一个已存在类的基础上建立一个新的类)。

2).类的继承：一个新类从已有的类型那里获取其已有的特性，这个称为类的继承。

3).单继承：一个派生类只从一个基类派生，这称为单继承。

4).多重继承：一个派生类有两个或多个基类的称为多重继承。

继承的概念：继承；派生；基类；派生类；

基类：已有的类称为基类或者父类。

派生类：新建立的类称为派生类或者子类。

如： 马 （基类）

公马 母马 （派生类）

白公马 黑公马 白母马 黑母马 （派 生类）

5).继承的步骤：派生类生成过程包含3个步骤：资源：成员

吸收基类的成员 （将基类的成员拿过来，变成自己的成员）

改造基类的成员 （自己添加一些与基类成员同名的成员）

添加新的成员 （添加一些基类所没有的资源）

构造函数、析构函数是不能继承

每个类都有自己的构造函数、析构函数，表示自己的构造过程

6).继承格式：class 派生类名 : 派生方式 基类名

{

.....

};

class 跑车 : public 车

{

};

class Point

{

int x;

int y;

};

class Point3d : public Point

{

};

2.派生：就是从已有的类（父类）产生一个新的子类称为类的派生。

1).派生方式（继承方式）：派生有public（公用）、protected（受保护）、private（私有的）三种方式，不同的派生方式下，派生类对基类成员的访问权限以及外部对基类成员的访问权限有所不同.

1.公用继承：基类的公有成员和保护成员在派生类中保持原有访问属性，其私有成员仍为基类私有。2.私有继承：基类的共有成员和保护成员在派生类中成了私有成员。其私有成员仍为基类私有。3.受保护的继承：基类的公有成员和保护成员在派生类中成了保护成员，其私有成员仍为基类私有。保护成员的意思是，不能被外界引用，但可以被派生类的成员引用。

2)..派生类成员

访问权限：private成员是私有成员，只能被本类的成员函数所访问，派生类和类外都不能访问。

public成员是公有成员，在本类、派生类和外部都可访问

protected成员是保护成员，只能在本类和派生类中访问。是一种区分血缘关系内外有别的成员。

派生类的访问权限规则如下：

基类的private成员在外部和派生类中均不可访问

private派生使得基类中的所有成员都变成private成员，（在外部和其进一步的派生类中都不可访问）

protected派生使得基类中的所有非private成员权限都降一级：public降为protected，protected降为private

public派生时，基类的所有成员在派生类中的访问权限不变

在派生类中

1.如果访问的是基类的成员，就要考虑权限问题（public protected可访问，private不可访问）

基类的私有成员虽然不可访问，但是可以通过非private成员去间接访问

2.如果访问的是自己的成员，就不存在权限问题

访问基类成员的格式：

子类对象名. 基类名::成员名

A

s.x;

s.A::add

3.在能访问的前提条件下：

1).访问基类的成员函数，这个基类的成员函数只能访问基类的成员

2).如果访问成员时，成员仅有一份时，不需要指定源自哪个基类

3).如果访问成员时，成员有多份相同的

1.自己有，默认优先使用自己的

2.自己没有，就要指定源自哪个基类的

相同：数据成员：变量名相同

函数成员：函数名相同，形参相同

继承：必须满 “是” 的关系 跑车 是 车 电脑 是 电器

是否有访问权限:有权限：直接写成员名 x = 20;

成员在类中是否唯一 :唯一：直接写 :x = 20;

不唯一：成员是否是新增成员

是：直接写 x = 20;

否：指定源自哪个基类A::x = 20; B::z = 30; 无权限：不能直接写成员名，只能间接访问:setX(20);

1. 公用继承：在定义一个派生类时将基类的继承方式指定为public的，称为公用继承。，用公用继承的方式建立的派生类称为公用派生类，其基类称为公用基类。

公共基类的成员在派生类中的访问属性：

|  |  |  |
| --- | --- | --- |
| 在基类的访问属性 | 继承方式 | 在派生类中的访问属性 |
| Private（私有）  Public（公用）  Protected（保护） | Public（公用）  Public（公用）  Public（公用） | 不可访问  Public（公用）  Protected（保护） |

1. 私有继承：在声明一个派生类时将基类的继承方式指定为private的称为私有继承。用私有继承方式建立的派生类称为私有派生类，其基类称为私有基类。私有基类的公用成员和保护成员在派生类中的访问属性相当于派生类中的私有成员（即派生类的成员函数能访问它们，而在派生类外不能访问它们）私有基类的私有成员在派生类中成为不可访问的成员，只有基类的成员函数可以引用它们。（私有基类的成员可以被基类的成员函数访问，但不能被派生类的成员函数访问）

私有基类的成员在私有派生类中的访问属性

|  |  |  |
| --- | --- | --- |
| 在基类的访问属性 | 继承方式 | 在派生类中的访问属性 |
| Private（私有）  Public（公用）  Protected(保护) | Private(私有)  Private（私有）  Private（私有） | 不可访问  Private（私有）、  Private（私有） |

//调用问题：1.不能通过派生类对象引用从私有基类继承过来的任何成员。2.派生类的成员函数不能访问私有基类的私有成员，但可以访问私有基类的公用成员函数可以调用基类的公用成员函数，但不能引用基类的私有成员。

1. 保护成员和保护继承

由protected声明的成员称为受保护成员（保护成员）。与private一样是用来声明成员的访问权限的。受保护成员不能被类外访问，这与私有成员相似。不同在于保护成员可以被派生类的成员函数引用。在定义一个派生类时将基类的继承方式指定为protected的称为保护继承。用保护继承的方式建立的派生类称为保护派生类，其基类称为受保护的基类（简称保护基类）

特点：保护基类的公有成员和保护成员在派生类中都成了保护成员，其私有成员仍为基类私有（也就是把基类原有的公有成员也保护起来不让类外任意访问。）

注意：如果基类声明了私有成员，那么任何派生类都是不能访问它们的，如果希望在派生类中能访问它们，应当把他们声明为保护成员。（一个类中声明了保护成员，就意味着该类可能要用作基类，在派生类中会访问这些成员）。

基类成员在派生中的访问属性：

|  |  |  |
| --- | --- | --- |
| 在基类的访问属性 | 继承方式 | 在派生类中的访问属性 |
| Public（公用）  Private（私有）  Protected（保护） | Protected（保护）  Protected（保护）  Protected（保护） | Protected（保护）  不可访问  Protected（保护） |

1. 保护基类的所有成员在派生类中都被保护起来，类外不能访问，其公用成员和保护成员可以被其派生类的成员函数访问，私有成员则不可以访问。
2. 比较私有继承和保护继承（私有派生类和保护派生类）发现，在直接派生类中，以上两种继承方式的作用实际相同的，即在类外不能访问任何成员，而在派生类中可以通过成员函数访问基类中的公有成员和保护成员。但是如果继续派生，在新的派生类中，两种继承方式作用就不同了。（如果以公用继承方式派生出一个新派生类，原来私有基类中的成员在新派生类中都成为不可以访问的成员，无论在派生类内或派生类外都不能访问，而原来保护基类中的公用成员和保护成员在新派生类中为保护成员，可以被新派生类的成员函数访问
3. 在派生类中，成员的4种访问属性

|  |  |  |  |
| --- | --- | --- | --- |
| 派生类中访问属性 | 在派生类中 | 在派生类外部 | 在下一层公用派生类中 |
| 公用  保护  私有  不可访问 | 可以  可以  可以  不可以 | 可以  不可以  不可以  不可以 | 可以  可以  不可以  不可以 |

派生类外部就是在建立派生类对象的模块中，在派生类范围之外

如果被派生类继续派生，则在不同的继承方式下，成员所获得的访问属性是不同的。三种继承访问方式如上述表所标。

A

1. 多级派生时的访问属性

如图所示派生关系：类A为基类，类B是类A的派生类，类C是类B的派生类

B

则类C也是类A的派生类。类B称为类A的直接派生类，类C称为类A的间接

派生类。类A是类B的直接基类，是类C的间接基类。

C

如：class A{public: int i ; //基类

Private : int k;

Protected: void f1(); int j;

};

Class B:public A{ public: void f2(); //public派生类

Private: int m;

Protected: void f3();

};

Class C:protected B{public: void f4(); ///protected派生类

Private: int n;

};

各成员在不同类中的访问属性

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | i | F1() | j | k | F2() | F3() | m | F4() | n |
| 基类A  公用派生类B  保护派生类C | 公用  公用  保护 | 保护  保护  保护 | 保护  保护  保护 | 私有  不可访问  不可访问 | 公用  保护 | 保护  保护 | 私有  不可访问 | 公用 | 私有 |

4.派生类的构造函数和析构函数

派生时，构造函数和析构函数是不能继承的，为了对基类成员进行初始化，必须对派生类重新定义构造函数和析构函数，并在派生类构造函数的初始化列表中调用基类的构造函数。

由于派生类对象通过继承而包含了基类数据成员，因此，创建派生类对象时，系统首先通过派生类的构造函数来调用基类的构造函数，完成基类成员的初始化，然后再执行派生类的构造函数对派生类中新增的成员进行初始化。

1. 单基派生类的构造函数

派生类构造函数的一般格式为：

派生类名(总参数表): 基类构造函数(参数表)

{

//函数体

}

必须将基类的构造函数放在派生类的初始化表达式中，以调用基类构造函数完成基类数据成员的初始化，派生类构造函数实现的功能，或者说调用顺序为：1.完成对象所占整块内存的开辟，由系统在调用构造函数时自动完成。2.调用基类的构造函数完成基类成员的初始化。3.若派生类中含对象成员、const成员或引用成员，则必须在初始化表达式中完成其初始化。4.派生类构造函数体被执行。

2.单基派生的析构函数：

当对象被删除时，派生类的析构函数被执行。析构函数同样不能继承，因此，在执行派生类析构函数时，基类析构函数会被自动调用。执行顺序是先执行派生类的析构函数，再执行基类的析构函数，这和执行构造函数时的顺序正好相反。

3.有子对象的派生类构造函数：

类中的数据成员还可以包含类对象，可以在声明一个类时包含这样的数据成员。

如：class student{ public:

Protected： int num; string name; char sex;

};

Class student1:public student{ public: void show();

Private: student montor;//定义子对象 int age ; string addr;

};

派生类构造函数的任务应该包括3个部分：1.对基类数据成员的初始化；2.对子类对象数据成员初始化3。对派生类数据成员初始化。

定义派生类构造函数的一般形式：

派生类构造函数（总参数表）：基类构造函数名（参数表），子对象名（参数表）{派生类中新增数据成员初始化语句}

执行派生类构造函数的顺序是：

1).调用基类构造函数，对基类数据成员初始化2).调用子对象构造函数，对子对象数据成员初始化.3).再执行派生类构造函数本身，对派生类数据成员的初始化。

4.多重继承和多层派生时的构造函数

派生类只有一个基类时，称为单基派生，在实际运用中，经常需要派生类同时具有多个基类，这种方法称为多基派生或多重继承，下图是双基继承的示例，在实际应用中，还允许使用三基甚至是更多基继承。

C++充许一个派生类同时继承多个基类，这种操作称为多重继承。

A

1. B

C

1.声明多重继承的方法：

C++中，声明和定义具有两个以上基类的派生类

与声明单基派生类的形式类似，只需将要继承的多个基类用逗号分开即可

如：

派生类名(参数表):基类名1(参数表1),基类名2(参数表2),…,基类名n(参数表n)

{

private:

新增私有成员列表;

public:

新增公开成员列表;

};

例如，从A类和B类派生出C类的方式如下：

class C : public A, public B

{

//……

};

2.多重继承派生类（多基派生）的构造函数：

多基派生时，派生类的构造函数格式如（假设有N个基类）：

派生类构造函数名(总参数表)：基类1构造函数名(参数表1)，基类2构造函数(参数表2)，……，基类名N(参数表N)

{

派生类中新增数据成员初始化语句 //函数体

}

//多基派生和单基派生构造函数完成的任务和执行顺序并没有本质不同，唯一一点区别在于首先要执行所有基类的构造函数，再执行派生类构造函数中初始化表达式的其他内容和构造函数体，各基类构造函数的执行顺序与其在初始化表中的顺序无关，而是由定义派生类时指定的派生类顺序决定的

3.多重继承（多基派生）引起的二义性问题：

一般来说，在派生类中对基类成员的访问应当具有唯一性，但在多基继承时，如果多个基类中存在同名成员的情况，造成编译器无从判断具体要访问的哪个基类中的成员，则称为对基类成员访问的二义性问题。

如：class A{public: int a;void display();}; class B{public:int a; void display(); };

Class C:public A,public B{public: int b; void show();} ;//公用继承;

//出现同名情况。

解决： 1.可以用基类名来限定： C c1;

C1.A::a=3;//引用c1对象中的基类A的数据成员

C1.A::display();//引用c1对象中的基类A的成员函数display

2. 如果派生类C中的成员函数show访问基类A的display和啊a。可以不必写对象名二直接写

A::a=3; A::display();

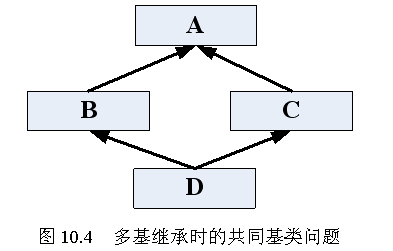
3.同名覆盖

C类：int a;int A::a; int B::a; void display(); void A::display(); void B::display();

//基类的同名成员在派生类中被屏蔽，成为“不可见”的，或者派生类新增的同名成员覆盖了基类中的同名成员。所以如果在定义派生类对象的模块中通过对象名访问同名的成员，则访问的是派生类的成员。注意：只有在函数名和参数个数相同，类型相匹配的情况下才发生同名覆盖。否则是函数重载。

5.虚基类（共同基类）

多基派生中，如果在多条继承路径上有一个共同的基类，如图所示，，在D类对象中，会有来自两条不同路径的共同基类（类A）的双重拷贝。

1. 虚基类的初始化

声明虚基类方法：class A{…..}; class B:virtual public A{…….} ;class C:virtual public A{………};

//声明类B是类A的公用派生类，A是B的虚基类// 声明类C是类A的公用派生类，A是C的虚基类

声明虚基类的一般形式：class 派生类名：virtual继承方式 基类名

虚基类初始化：如果在虚基类中定义了带参数的构造函数，而且没有定义默认构造函数，则在其所有派生类（直接派生类或间接派生类的派生类中），通过构造函数的初始化表对虚基类进行初始化。

如:class A{A (int i){}………..};class B:virtual public A{B(int n):A(n){}…………};class

C:virtual public A{ C(int n):A(n){}…………..}; class D:public B,public C{D(int n):A(n),B(n),C(n){}…………};

//这里C++编译系统只执行最后的派生类对虚基类的构造函数的调用，而忽略虚基类的其他派生类（B类，C类）对虚基类的构造函数的调用，这就保证虚基类的数据成员不会被多次初始化。

注意：z在最后的派生类中不仅要负责对其直接基类进行初始化，还要负责对虚基类初始化

2.虚基类的作用：

C++提供虚基类的方法，使得在继承间接共同基类时只保留一份成员。

3.虚基类的二义性

共同基类和多基派生的共同作用，使得在派生类中会出现多个共同基类的拷贝，这很容易带来二义性问题如：D类通过B类和C类各继承了一次A类的函数SetX()和Print()，从而产生了二份拷贝，进而导致了二义性

解决：使用关键字virtual将共同基类A声明为虚基类，可有效解决上述问题。在定义由共同基类直接派生的类（示例中的类B和类C）时，使用下列格式定义：

class 派生类名 : virtual 派生方式 基类名

A{SetX()、Print()}

B{}

A{SetX()、Print()}

C{}

{

//类定义

};

D{}

4.虚基派生的构造函数和析构函数

对普通的多层继承而言，构造函数的调用是嵌套的，如由C1类派生C2类，C2类又派生C3类时，C1C2C3，有：

C2(总参数表):C1(参数表)

C3(总参数表):C2(参数表)

而对图10-6所示的虚基派生来说，如果按照上述规则，应该有：

B(总参数表):A(参数表)

C(总参数表):A(参数表)

D(总参数表):B(参数表),C(参数表)

这样“A(参数表)”将被执行2次，这显然不行，因为根据虚基派生的性质，类D中只有一份虚基类A的拷贝，因此A类的构造函数在D类中只能被调用一次。其实实际代码是：

B(总参数表):A(参数表)

C(总参数表):A(参数表)

D(总参数表):B(参数表),C(参数表),A(参数表)

其中A(参数表)是编译器隐式自动加上去的。并且在B(参数表),C(参数表)里不再调用虚基类A的构造函数。

当然我们也可以在D类构造函数的初始化列表中显式列出对虚基类A构造函数的调用。这样可以指定传入参数的值。

这种机制保证了不管有多少层继承，虚基类的构造函数必须且只能被调用一

6.基类与派生类的相互转换:（将派生类的值赋给基类对象）

在用到基类对象的时候可以用其子对象代替。

1.派生类对象可以向基类对象赋值（可以用子类（公用派生类）对象对其基类对象赋值）2.派生类对象可以替代基类对象向基类对象的引用进行赋值或初始化。3.如果函数的参数时基类对象或基类对象的引用，相应的实参可以用子类对象。4.派生类对象的地址可以赋给指向基类对象的指针变量，（指向基类对象的指针变量也可以指向派生类对象）

1).单基继承的类型适应

2).多基继承的类型适应

3).公共基类的类型适应

4).虚基类的类型适应

注意：1。只能用子类对象对其基类对象赋值，而不能用基类对象对其子类对象赋值。（因为基类对象不包含派生类的成员，无法对派生类赋值）2。同一基类的不同派生类对象之间也不能赋值。

7.继承与组合

面向对象设计的难点在于类的设计，而不是对象的创建，就像工业生产中图纸是关键，有了图纸，产品可以很容易地创建出来。在程序设计中，如何处理类派生和类组合一直是个很让人觉得折磨的问题。

前面已提及继承的重要性，使得代码结构清晰，大大提高了程序的可复用性，因此，容易犯的错误就是把继承当成灵丹妙药，不管三七二十一拿来继承一下再说，其实，在面向问题空间的对象组织方面，不只有继承，还有对象组合，更高阶的结构还有聚合等

1.组合：某类以另一个类对象作数据成员，称为组合，在逻辑上，如果类A是类B的一部分（a part of）或者说HAS-A（“有一个”），那么就不要从A类派生出类B，而应当采用组合的方式，《高质量C++编程指南》中“眼睛、鼻子、嘴巴、耳朵和头部”的范例很好地解释了组合的本质：眼睛、鼻子、嘴巴、耳朵分别是头部的一部分，头部并不是从眼睛、鼻子、嘴巴、耳朵继承来的。

//通过继承建立了派生类与基类的关系它是一种“是”关系，

//通过组合建立了成员类与组合类（复合类）的关系

//继承是纵向的，组合是横向的

1. 多态性与虚函数

1.多态性

多态性：向不同的对象发送同一个消息，不同的对象在接收时产生不同的行为（方法）。在C++中，多态性的表现形式之一是：具有不同功能的函数可以用同一个函数名，这样就可以实现一个函数名调用不同内容的函数。多态性是面向对象设计语言的基本特征。仅仅是将数据和函数捆绑在一起，进行类的封装，使用一些简单的继承，还不能算是真正应用了面向对象的设计思想。多态性是面向对象的精髓。多态性可以简单地概括为“一个接口，多种方法”，函数重载就是一种简单的多态，一个函数名（调用接口）对应着几个不同的函数原型（方法）。

多态的分类:

生物 动物 哺乳动物 微生物

人 男人 暖男

女人 渣男

犬科动物 狗 哈士奇

泰迪

植物 金银花

养（狗 n）

{

}

养（金银花 n）

{

}

1）多个类彼此不同，但是拥有共同的源基类

差异非常之大

多态性很明显

2）多个类，拥有直接基类

3）对象的多态

1.同类对象 属性的差异 导致的多态

2.不同类对象 行为的差异 导致的多态

多态性的分类：1.静态多态性2.动态多态性

特点：静态多态性是通过函数重载实现的。由函数重载和运算符重载形成多态性属于静态多态性。（编译时的多态性）在程序运行前就已决定了执行的函数方法。

动态多态性：不再编译时确定调用的是那个函数，而是在程序运行过程中才动态的确定操作所针对的对象爱那个。（运行时的多态性）动态的多态性是通过虚函数实现的。（同一类族中不同类的对象，对同一函数调用做出不同的响应）

联编：程序调用函数时，具体应使用哪个代码块是由编译器决定的。以函数重载为例，C++编译器根据传递给函数的参数和函数名决定具体要使用哪一个函数，称为联编

1).静态联编：编译器可以在编译过程中完成这种联编，在编译过程中进行的联编叫静态联编（static binding）或早期联编

2).动态联编：在一些场合下，编译器无法在编译过程中完成联编，必须在程序运行时完成选择，因此编译器必须提供这么一套称为“动态联编”（dynamic binding）的机制，也叫晚期联编（late binding），C++通过虚函数来实现动态联编

2.虚函数:

本质：函数 虚：virtual 如:virtual int add(int a, int b);

在基类中提供一个统一的接口，将这个接口函数 声明为虚函数然后，参数通常为 基类指针.

注意：1。只能用virtual声明类的成员函数，把它作为虚函数，而不能将类外的普通函数声明为虚函数。（因为虚函数的作用是充许在派生类中对基类的虚函数重新定义。所以只能用于类的继承层次结构中）。2.一个成员函数被声明为虚函数后，在同一类族中的类就不能再定义一个非virtual的但与该虚函数具有相同的参数（个数，类型）和函数返回值类型的同名函数。

1).虚函数作用：虚函数就是用来解决动态多态问题的。（在程序运行期间，用指针指指向某一派生类对象，这样就能调用指针指向的派生类对象中的函数而不会调用其它派生类中的函数）。

注意：虚函数的作用是充许再派生类中重新定义与基类同名的函数，并且可以通过基类指针或引用来访问基类和派生类中的同名函数。

2).虚函数使用方法：

1.在基类中用virtual声明成员函数。在类外定义虚函数时，不必在家virtual2。定义一个指向基类对象的指针变量，并使它指向同一类族中需要调用该函数的对象。3.通过该指针变量调用此虚函数，此时调用的就是指针变量指向的对象的同名函数。4.在派生类中重新定义此函数，函数名，函数类型，函数参数个数和类型必须与基类的虚函数相同，根据派生类的需要重新定义函数体。（当一个成员函数被声明为虚函数后，其派生类中的同名函数都自动成为虚函数，所以派生类重新定义该函数可加virtual，也可不加）

3).虚函数的声明和初始化

虚函数的定义很简单，只要在成员函数原型前加一个关键字virtual即可。

如果一个基类的成员函数定义为虚函数，那么，它在所有派生类中也保持为虚函数；即使 在派生类中省略了virtual关键字，也仍然是虚函数。

派生类中可根据需要对虚函数进行重定义，重定义的格式有一定的要求：

* + 与基类的虚函数有相同的参数个数；
  + 与基类的虚函数有相同的参数类型；
  + 与基类的虚函数有相同的返回类型：或者与基类虚函数的相同，或者都返回指针（或引用），并且派生类虚函数所返回的指针（或引用）类型是基类中被替换的虚函数所返回的指针（或引用）类型的子类型（派生类型）。

4).什么情况下使用虚函数

1.看成员函数所在的类是否会作为基类。然后看成员函数在类的继承后有无可能被更改功能，如果希望更改其功能的，一般将其声明为虚函数。

2.如果成员函数在类被继承后功能无需修改，或派生类用不到该函数，则不要把它声明为虚函数。不要仅仅考虑到要作为基类而把类中的所有成员函数都声明为虚函数。

3.考虑对成员函数的调用是通过对象名还是通过基类指针或引用去访问，如果是通过基类指针或引用去访问的，则应当声明为虚函数。

4.在定义虚函数时，并不定义其函数体，即函数体时空的，它的作用只是定义了一个虚函数名，具体的功能留给派生类去添加。

5).虚函数的访问

1.对象名访问

2.引用访问

3.指针访问

4.类内访问

5.在构造函数或析构函数中访问

6).虚函数表vftable

1.C++中的虚函数的实现一般是通过虚函数表。如果类中包含有虚函数，在用该类实例化对象时，对象的第一个成员将是一个指向虚函数表(vftable)的指针(vfptr)。虚函数表是一块连续的内存，它的每一项都记录了一个虚函数的入口地址。如果类中有N个虚函数，那么其虚函数表将有N\*4字节的大小。它就像一个地图一样，指明了实际所应该调用的函数。

2.虚函数表是固定的东西，在编译的时候确定，虚函数指针vptr是在运行阶段确定的，而多态的实现是通过对象中的vptr指针指向不同的虚函数表实现的，在运行的时候指针指向是可以有变化的。

3.可以通过对象的地址得到这张虚函数表，然后就可以遍历其中的[函数指针](http://baike.baidu.com/view/1604730.htm)，并调用相应的虚函数

虚函数表

虚函数1入口地址

虚函数2入口地址

vfptr

……

3.纯虚函数与抽象类

1.纯虚函数：当在基类中无法为虚函数提供任何有实际意义的定义时，可以将该虚函数声明为纯虚函 数，它的实现留给该基类的派生类去做。基类中的纯虚函数为它的派生类提供统一的接口

方便多态的实现

2.定义和声明：纯虚函数是一种特殊的虚函数，其格式一般如下：

class 类名

{

virtual 类型 函数名 (参数表)=0;

…

};

纯虚函数不能被直接调用，仅提供一个与派生类一致的接口；如果在一个类中声明了纯虚函数，而在其派生类中没有对该函数定义，则该虚函数在派生类中仍为纯虚函数。

2.抽象类

1. 一个类可以包含多个纯虚函数。只要类中含有一个纯虚函数，该类便为抽象类。一个抽象类只能作为基类来派生新类，不能创建抽象类的对象，如例中的A类便是抽象类，创建A类的对象是非法的，如：

A a; //错误：A为抽象类

但可声明一个指向抽象类的指针，如：

A\* a=NULL;

A\* a=new B;

应注意：“A\* a=new A;”非法，因为该语句试图创建A的对象。

和普通的虚函数不同，纯虚函数不能被自动继承，在派生类中必须对基类中虚函数进行重定义，或者在派生类中再次将该虚函数声明为纯虚函数，否则编译器将提示错误信息。这说明，抽象类的派生类也可以是抽象类，只有在派生类中给出了基类中所有纯虚函数的实现时，该派生类便不再是抽象类。和纯虚函数一样，抽象类只起到提供统一接口的作用。

1. 只定义了protected型构造函数的类也是抽象类

对一个类来说，如果只定义了protected型的构造函数而没有提供public构造函数，无论是在外部还是在派生类中都不能创建该类的对象，但可以由其派生出新的类，这种能派生新类，却不能创建自己对象的类是另一种形式的抽象类

3. 对类层次中的同名成员函数来说，有3种关系：

1.重载（overload）、

2.覆盖（override）

3.隐藏（hide、oversee）