**以存在的类为基础定义新的类，新类即捅有基类的数据成员和成员函数的一份复制品,这就是继承**

单一继承：只有一基类

多重继承：可以有多个基类

基类 也叫超类，父类

派生类 也叫子类

**类的派生类成员：可以访问protected的成员**

公有继承、保护继承和私有继承，也称为公有派生、保护派生和私有派生。

**class B {……};**

**class D : [private | protected | public] B**

**{**

**……**

**};**

**派生类从基类继承到的成员，维持基类成员的可访问性。**

* + 在private派生方式下，派生复制了基类全部成员，但复制到的成员在派生类中**全部变成了private成员**。

派生类的成员函数不能直接访问基类的private成员，但可以直接访问基类的public和protected成员，并且通过它们访问基类本身的private成员。

**（protected成员，不能在派生类外直接访问）**

**在protected派生方式下，派生复制了基类全部成员，但复制到public成员在派生类中变成了protected成员，其余成员的访问权限保持不变**

在派生方式下，派生类的继承到的成员函数都不能被外部函数访问，但在派生类中可以直接访问基类的public和protected成员，并且通过它们访问基类本身的private成员。



* **不能继承的基类内容**

**1.构造函数、析构函数**

**2.友员关系**

**3.针对基类定义的一些特殊运算符，如new等。**

如果不想让一个类作为其它类的基类，可以用final关键字阻止它被继承。

class D **final**:Base{……} //正确，D不能被继承

**派生类拷贝了基类数据成员和成员函数的一份副本，**

**派生类可以在继承基类成员的基础上**

* + 派生类可以增加新的数据成员和成员函数；
  + 重载从基类继承到的成员函数；
  + 覆盖（重定义）从基类继承到的成员函数；
  + 改变基类成员在派生类中的访问属性

**派生类不能继承基类的以下成员**

* 1. 析构函数。
  2. 基类的友元函数。
  3. 静态数据成员和静态成员函数。

**基类的同名成员函数会被派生类重载的同名函数所隐藏**

**通过派生类对象直接访问基类成员** cub1.setWidth(5);

**在派生类成员函数中直接访问基类成员**

void outData() {

Rectangle::outData(); //L2 派生类访问基类同名成员的方法

cout << "high=" << high << endl;

}

* + - **在派生对象中访问基类同名成员函数**
      1. **cub1.Rectangle::outData();**

用using声明使基类重载函数在派生类中可见。用法如下：

**using 基类名称::被隐藏成员函数名;（**隐藏基类同名的全部重载函数）

class D : public B {

public:

**using B::f1; //L1，使基类的3个f1函数在此区域可见**

void f1(char \* d) { cout << d << endl; }

};

使用using声明**调整个别基类成员在派生类中的访问权限**

using语句在public区域内即为public权限，在protected内即为protected权限，在private内则为private权限

using Base::x; //指定x为public权限

using Base::setxyz; //指定setxyz为public权限

* + 如果一个类继承了其它类，则它声明的友元也只能访问它自己的全体成员，包括它从基类继承到的public和protected成员。而它的基类和派生类并不认可这种友元关系，按照规则只能访问公有成员。

如果基类定义了静成成员，则在整个继承体系中只有该成员的唯一定义，为整个继承体系中的全体对象所共用。**将共享数据或计数器设置为基类的静态成员**

**基类类域 相互之间可以直接访问，不受定义先后次序的影响。**

**派生类的作用域嵌套在基类作用域的内层**

**默认构造函数（定义无参对象或对象数组）、拷贝构造函数（类对象作函数参数），赋值运算符函数（对象赋值），**移动构造函数，移动拷贝构造函数和移动赋值运符函数

当派生类有（多个）基类和（多个）对象成员，派生类的构造函数除了要负责本类成员的初始化外，还要调用基类和对象成员的构造函数

派生类构造函数名(参数表)**:基类构造函数名(参数表),对象成员名1(参数表),……**

{

……

}

C(int x, int y,int z) :B(x,y),c(z)

{ //L4

cout << "Constructing C:\t" << c << endl;

}

**当基类或成员对象所属类只含有带参数的构造函数时，即使派生类本身没有数据成员要初始化，它也必须定义构造函数**

Line(int a,int b,int l):Point(a,b) { ***//构造函数初始化列表***

len=l;

cout<<"Constructing Line,len ..."<<len<<endl;

}

* **当具有下述情况之一时，派生类可以不定义构造函数。**
  1. **基类没有定义任何构造函数。**
  2. **基类具有缺省参数的构造函数。**

**基类具有无参构造函数**

**如果派生类的基类同时也是另外一个类的派生类，则每个派生类只负责它的直接基类的构造函数调用。**

**当派生类存在虚基类时，所有虚基类都由最后的派生类负责初始化**

构造函数继承

**using Base::Base; //继承基类构造函数**

如果基类有多个构造函数，则using语句会在派生类中为**每个基类构造函数生成一个与之对应的构造函数**，并具有与基类构造函数相同的访问权限

用using**声明其它成员**只是使该成员在指定的派生类权限区域可见，**并不生成代码**。而用**using继承基类构造函数**，则会使编译器在派生类中**生成基类构造函数的一份副本**。

**派生类构造函数构造原则和次序**

基类构造函数→对象成员构造函数→派生类构造函数

当有多个对象成员时，将按它们在派生类中的声明次序调用

**派生类**的赋值函数和拷贝构造函数，以及移动赋值和移动构造函数不但要执行派生类成员的拷贝和移动，而且**还要负责基类部分数据成员的拷贝和移动**

**公有派生方式下，凡是需要基类对象的地方都可以使用派生类对象**

派生类与基类赋值

从派生类对象中复制其基类子对象并将之赋值给基类对象。

以下两种操作并不存在从派生类向基类的类型转换：

* 1. 在把派生类对象赋值给基类对象
  2. 用派生类对象初始化基类对象

**可以把派生类对象转换成基类对象，不能把基类对象转换成派生类对象**

* + 把派生类对象赋值给基类对象或用派生类对象初始化基类对象，完成赋值或初始化操作后**，基类对象与派生对象就没有关系了**
  + 把基类对象的指针或引用绑定到派生类对象时，指针或引用从来就没有生成新对象，**它们操作的是派生类对象内部的基类子对象。**

**形参是基类对象**

* + **不能修改实参对象的值。**

**形参是基类对象的引用或指针**

* + **能够修改实参对象的值**

不能把基类对象赋值给派生类对象；不能把基类对象的地址赋值给派生类对象的指针；也不能把基类对象作为派生对象的引用

* **不论以哪种方式把派生类对象赋值给基类对象，都只能访问到派生类对象中的基类子对象部份的成员，不能访问派生类的自定义成员。**

**多继承：**

**class 派生类名:[继承方式] 基类名1,[继承方式] 基类名2,**

**…**

**{**

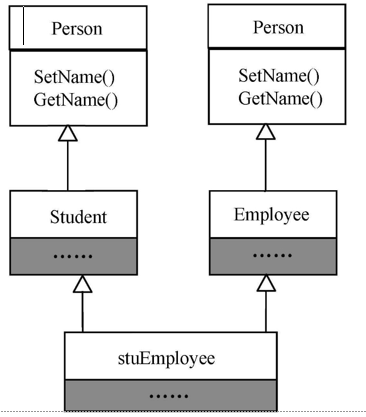
**……**

**};**

两个不同基类拥有同名成员时，容易产生名字冲突 **mi.A::f(); //L2：正确**

**虚拟继承：**

引入的虚拟继承的原因

派生类间接继承同一基类使得间接基类（Person）在派生类中有多份拷贝，引发二义性**虚拟继承使基类在派生类中只存在一份拷贝，解决了基类数据成员的二义性问题**

class 派生类名:virtual [继承方式] 基类名1，virtual [继承方式] 基类名2,…{

派生类成员声明与定义;

};

关键字virtual限定继承方式，将基类指定为虚拟基类，就使**该基类的成员在派生类中只有一份拷贝**

**构造函数初始列表中构造虚基类**

**在虚拟继承方式下，虚基类则由最终派生类的构造函数负责初始化**

**组合**则体现了类之间的另一种关系，是指一个类可以包容另外的类，即用其他类来定义它的对象成员