Polymorphism

一个function name有许多意思

C++里，virtual function提供这样的能力

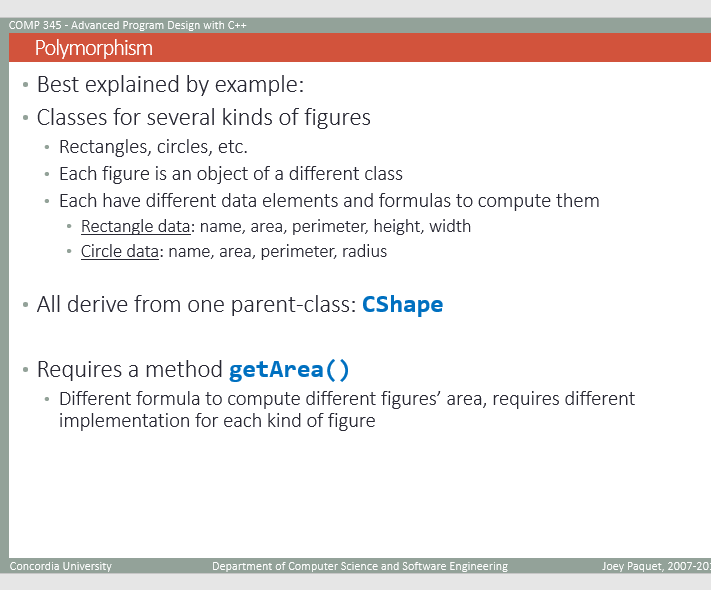
Object-oriented programming的基本原则之一

Virtual

虚拟的

利用virtual function来达到polymorphism

使用pure virtual function来创造抽象class

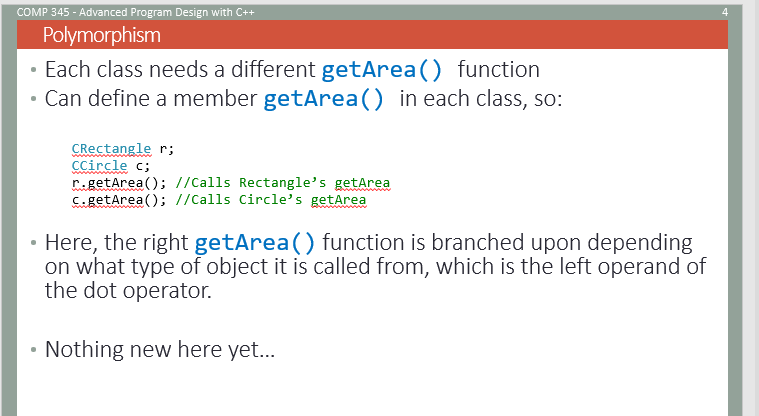


多态

三角形圆形都是继承于父类CShape

他们都需要method getArea()

然而他们计算面积方式是不同的，所以同样名字的function有了多个意思

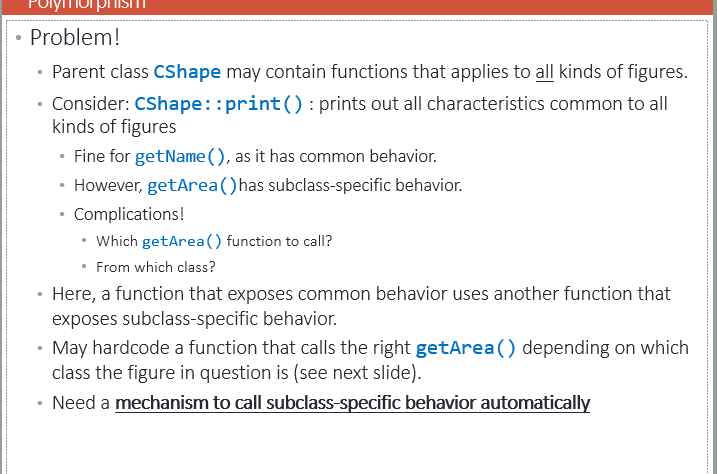


每一个class都需要getArea，

在各个子类描述getArea()

然后driver的时候两个getArea()实际上call的是不同的function

到这一步为止没啥新的



Cshape或许有所有子类都要用的function

例如print(0

假设print()要使用getName与getArea得到数据

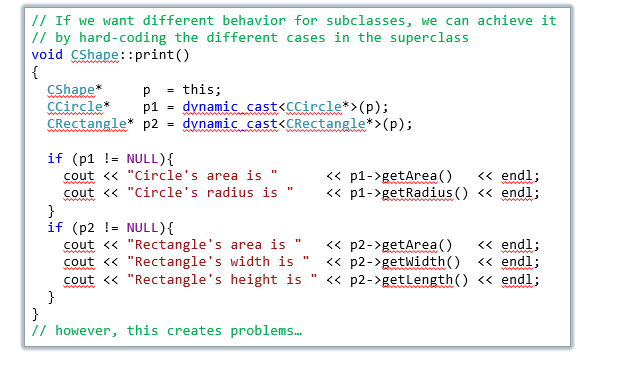
getName还好，大家都一样

但大家getArea不一样，不知道使用哪个function的getArea

你可以写一段比较难得代码来达到效果

或

我们需要新的机能来自动选择对应的子类behavior

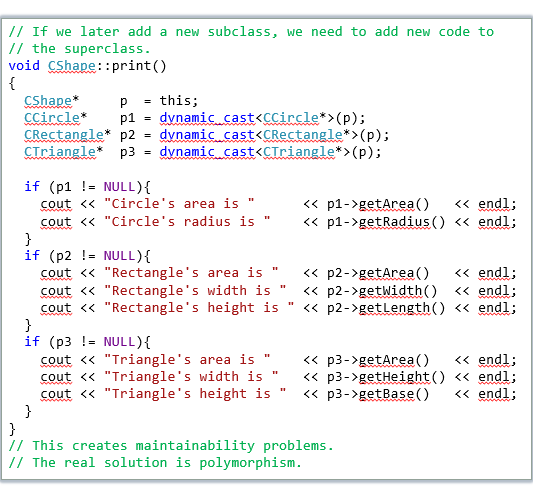


较难的代码，使用动态转换

然而这也是有问题的

假设我们描述了一个新的类别三角形，那么也会继承print()，然而print里面没有对应的三角形

所以每次我们新加东西就要改print，太憨批了

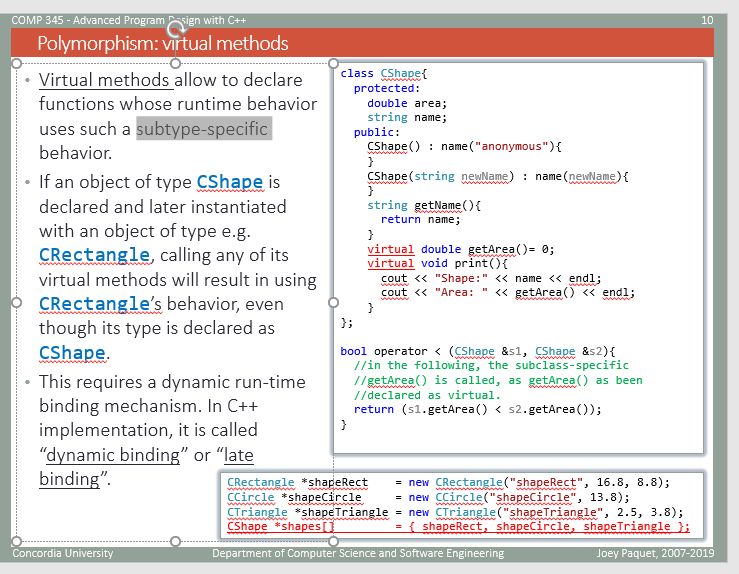


Virtual method

加了virtual 前缀的function允许在runtime（实际运行阶段）的时候使用子类同名function的behavior

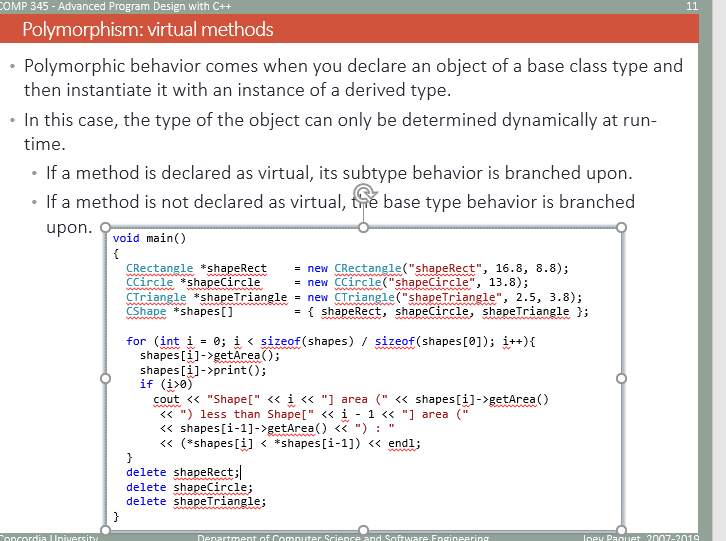
如果说一个object type一开始declare的时候是最基础的父类CShape （还没有使用constructor），然后才使用子类object例如CTriancle初始化，那么virtual method会使用Triangle里的描述

这在C++里叫动态binding



这里用virtual的就可以先直接=0

下面virtual也看着写



第四行就是declare In base type

前三行就是以 derived type实装

如果function 有virtual前缀，那么就使用子类的描述

如果没有virtual，就使用父类的描述。

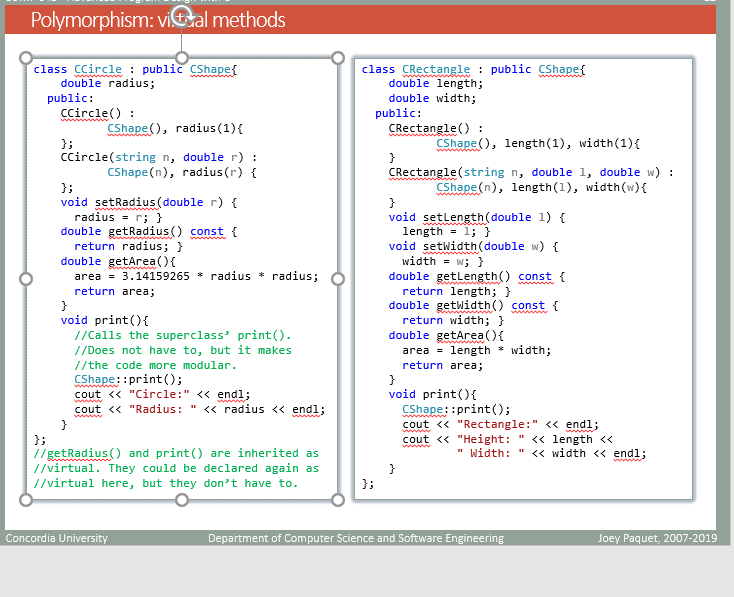
总结

Declare 子类 初始化子类，子类

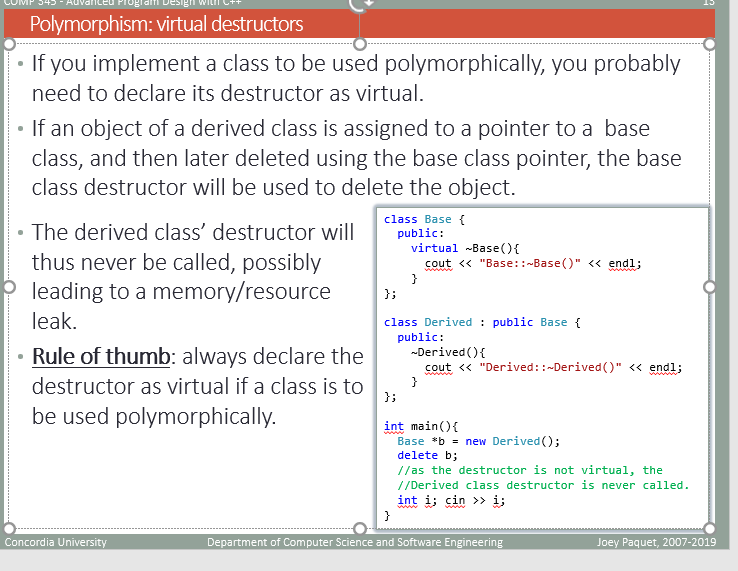
Declare 父类，初始化子类

有virtual 子类

没virtual 父类



继承的时候也可以借用superclass的print，没必要但让code更更容易阅读



如果你的class是多态的，那么他的destructor应该是virtual的

如果一个子类的objecct被赋予了一个父类类型的pointer，然后通过父类类型的pointer删除，这时会用父类的destructor。

那么子类的的destructor可能因此就没被叫到，可能会memory leak

原则就是如果是多态，那么父类的destructor尽量virtual

Pure virtual methods

C++里有一些virtual method可以是pure virtual

任意class，只要有哪怕一个pure virtual method,就是abstract class

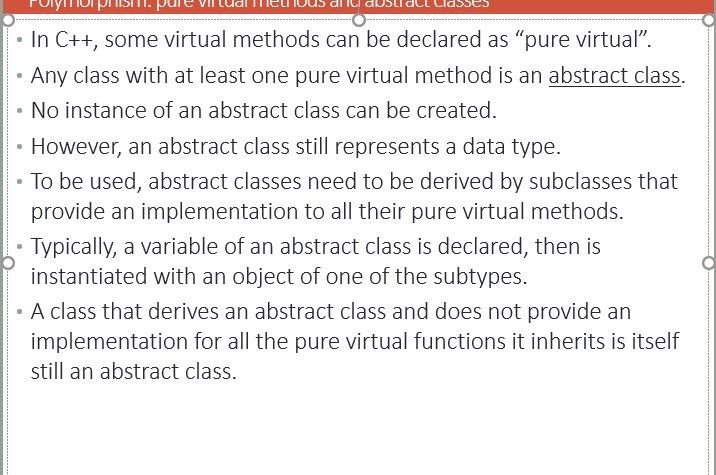
Abstract class并不能创造一个instance(object)

然而abstract class仍然代表着一个数据类型

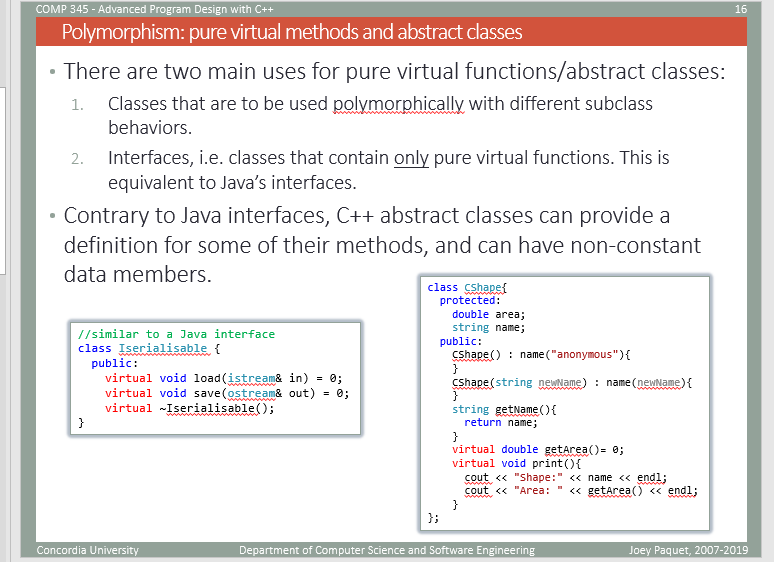
为了使用abstract class，我们必须使用一个继承子类，这个子类必须实装了所有pure virtual method

通常你可以declare一个抽象类的variable然后用一个子类的object 实例化

一个class如果继承了abstract class但并没有实装所有pure virtual function，那么这个子类class仍然是abstract class



‘



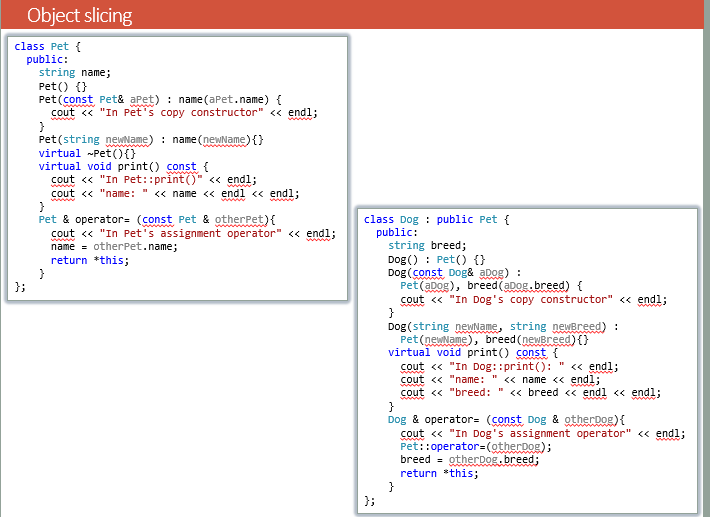
有两种方法使用 pure virtual function

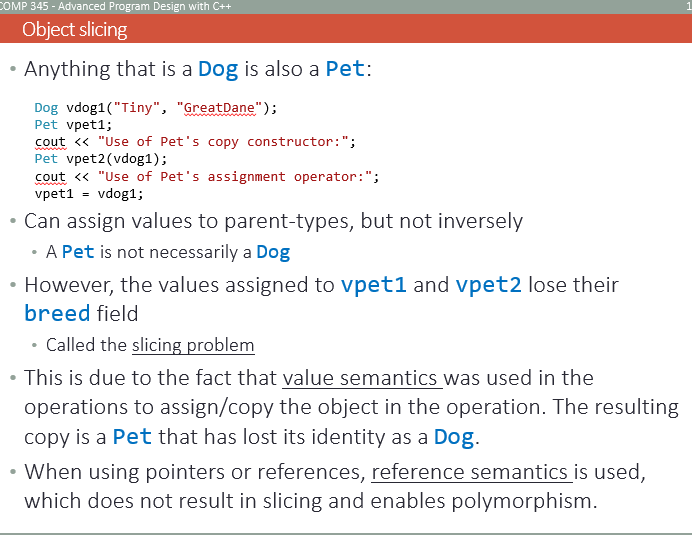
1. 那种将要被子类多态使用的父类

2.作为接口使用，class只包含pure virtual function,与java中的interface相似

然而与java 接口不同的是，C++抽象类可以描述一部分的method，并且也可以有非constant 数据类型

Object slicing: value semantics and reference semantics





假设dog是pet的子类，

首先我们能让父类=子类，给父类赋值一个子类

但不能子类=父类， 因为父类pet不见的一定是dog

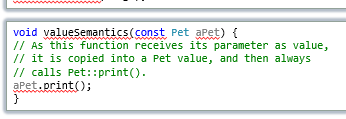
然而在赋值过程中，会损失dog里的breed信息

这叫做slicing problem

这是因为在copy操作中使用的值语义（value semantics也就是版本）。结果就是用pet里的copy constructor让他丧失了dog里的属性

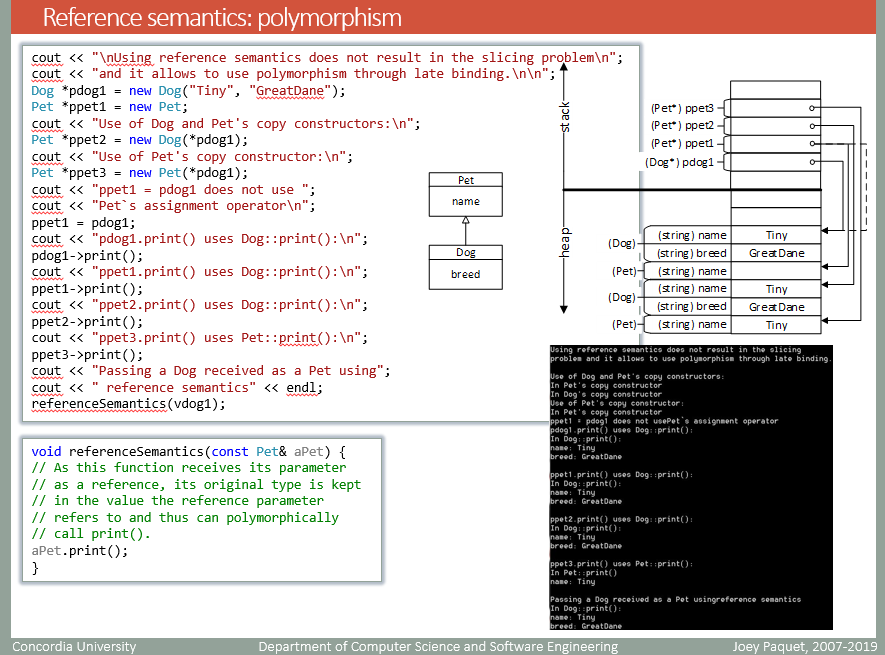
为了减少这个现象，应该使用Pointer或者reference,也就是reference semantics(使用pointer的版本)，这样就不会slicing并允许多态

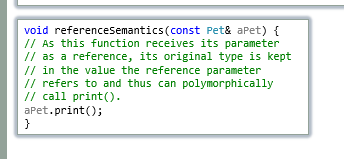
然后的两张图



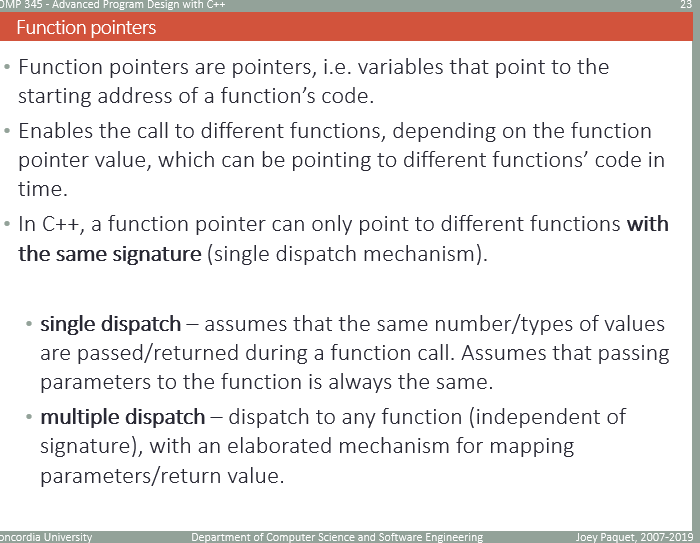
用值value作为parameter，损失数据







用reference没损失数据



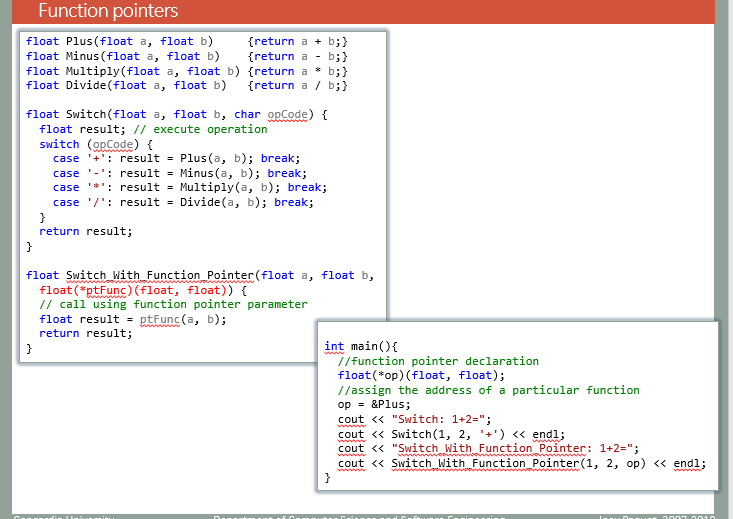
function Pointer指的是指向function code开头地址的pointer

允许call不同的function，取决于function pointer的值

C++中一个function pointer只允许指向有同样signature的不同function

sungle dispatch: passing parameter顺序类型总是一样

multiple dispatch::可以dispatch派遣到任意function，通过详尽描写机制



float(\*ptFunc)(float, float)

float指的是要Point的function的return type, (float,float是要指的function的参数)

\*pfFunc,\*代表着是pointer,pfFunc是这个pointer的名字

main中制定了一个pointer op

指向function Plus

这样第一个就是民工switch

第二个就是使用function pointer的高端选手

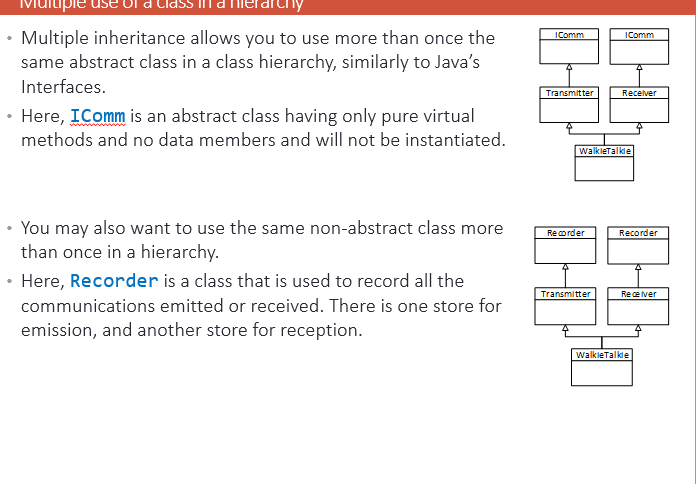
a,b，pointer

结果就是把ab带入pointer所指的function

float(\*op) (参数)

多重继承允许你使用多个抽象类

这里Icomm是一个只有一个pure virtual的class，没有object可以实例化

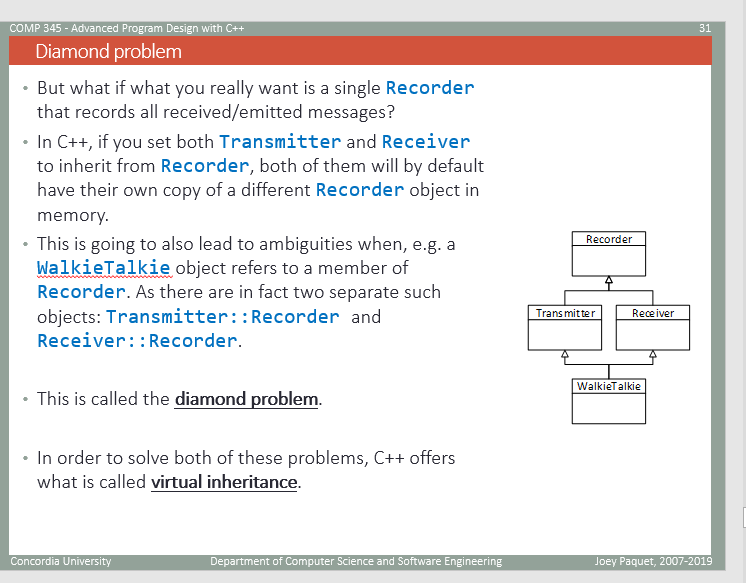


Recorder是一个基本类，记录器，用来收集信息

一个recorder用来收集emission，另外一个用来收集reception

那么我们可以让一个recorder既收集received又收集emitted message?

不用virtual就是上面的例子



在C++中，如果你让两者都继承了recorder，那么两者都会有默认的不同的继承下来的recorder object

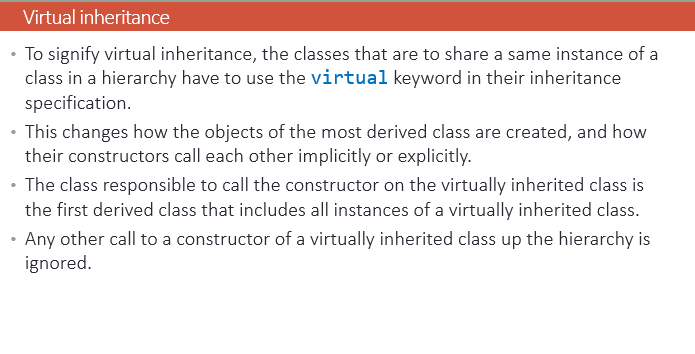
这就会增加混淆

假如创造walkie talkie 想使用recorder的信息，你不知道指的是哪个recorder

这就叫做diamond problem

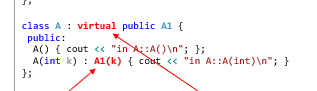
为了解决这个问题，C++允许virtual inheritance

创造了两个recorder object



为了表明是virtual inheritance

必须加上virtual前缀



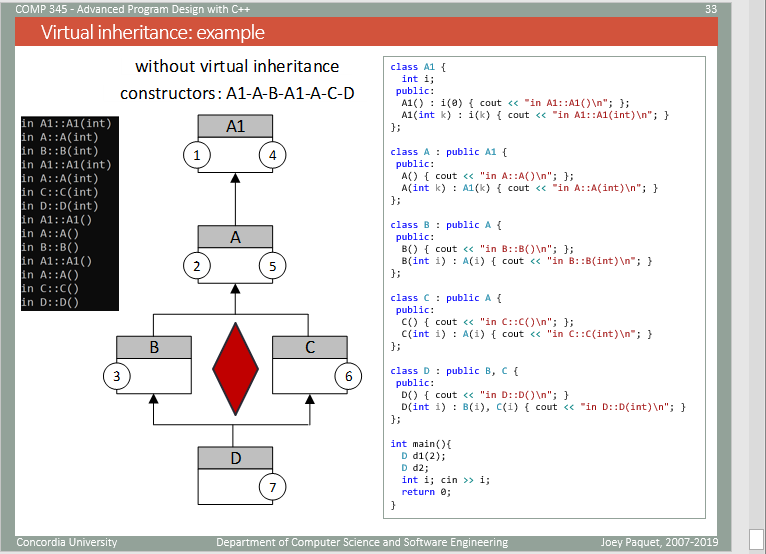
假设有一个父类

子类是virtual public XXXXX

然后

如果父类constructor被call过一次，

那么virtual 类中就不能再call这个父类constructor，会被忽略·



没有virtual inharitance的

第一个D1 2

他应该输出

b2

c2

DDDDDD

然后

A2

BBBBBBBBBBB

A2

CCCCCCC

DDDDDDD

然后

A1A1A1A1A1

AAAAAAAAAAA

BBBBBBBBBB

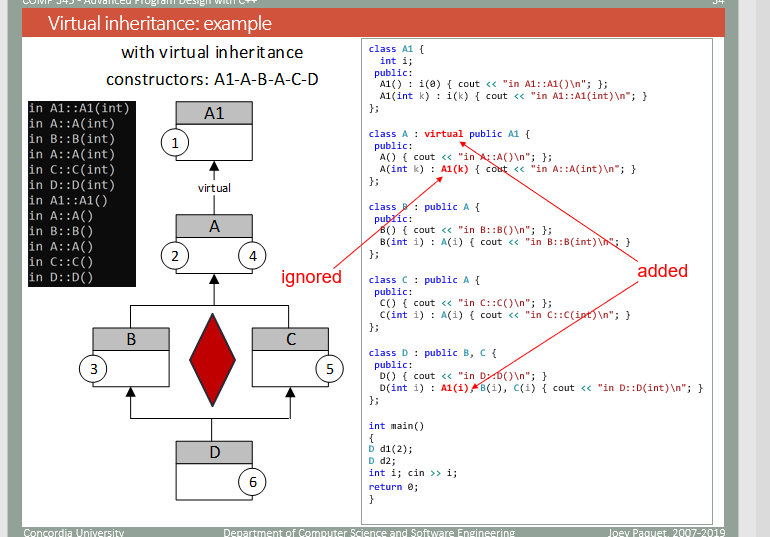
A1A1A1

AAAAAAAAA

CCCCCCCCCCCC

DDDDDDDDDDD

下面的那个d2不去管它

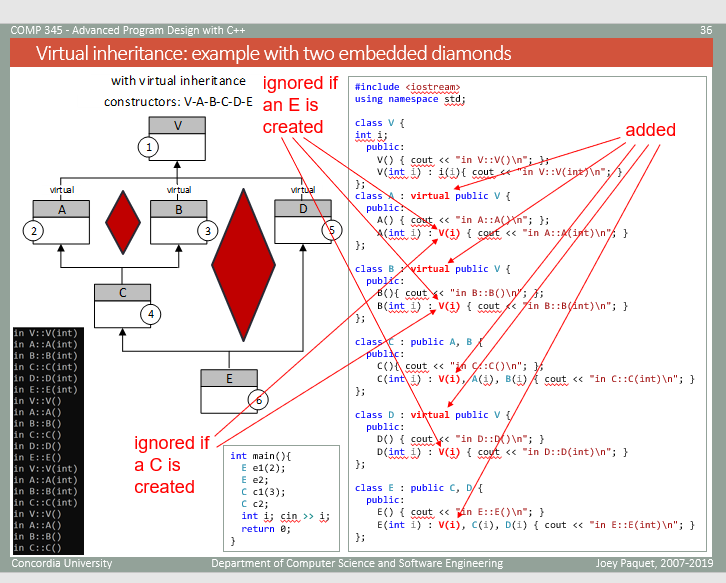


使用了virtual inheritance

D中call了祖父A1第一次，B(2),C(2)

然后A就不能再call

所以B(2)就是A(2)BBBBBBBBBBBB.,A(2)前半部分被忽略



这里e1(2)创造了第一个E

祖父V被call了一次，那么接下来A B D的V constructor都会被忽略，

C++使用virtual function table来实行late binding，

每一个包括或继承了virtual function的class 都被分配给一个virtual function table

vtable本身是一个静态array包含了function pointers

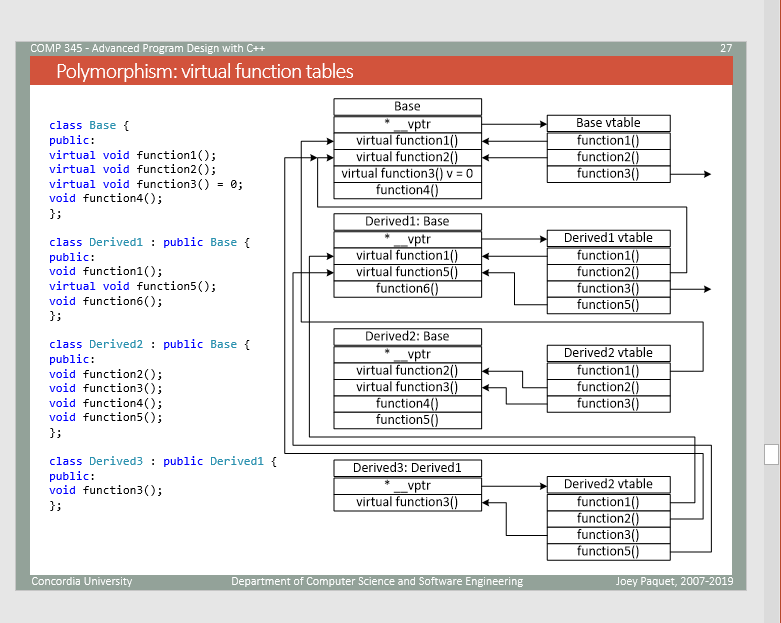
每一个virtual function都会有一个对应的function pointer

每一个pointer指向的都是最底层的class

对于pure virtual function来说，他可以默认vtable contain的是NULL pointer.

每一个通过有virtual function的class创造出来的object都有他自己的vtable

当这个object被付给父类祖雷时，他自己的子类vtable可以用来解决function call问题



function 3有=0，是pure virtual function，所以不用vtable不用指向原来的virtual function

别的vtable都要指向原来的virtual function

vptr要指向vtable

父类时virtual的function子类无论怎么蹦跶·还是virtual

弗雷没有才能自由，