* 1. cout判断输入数据类型

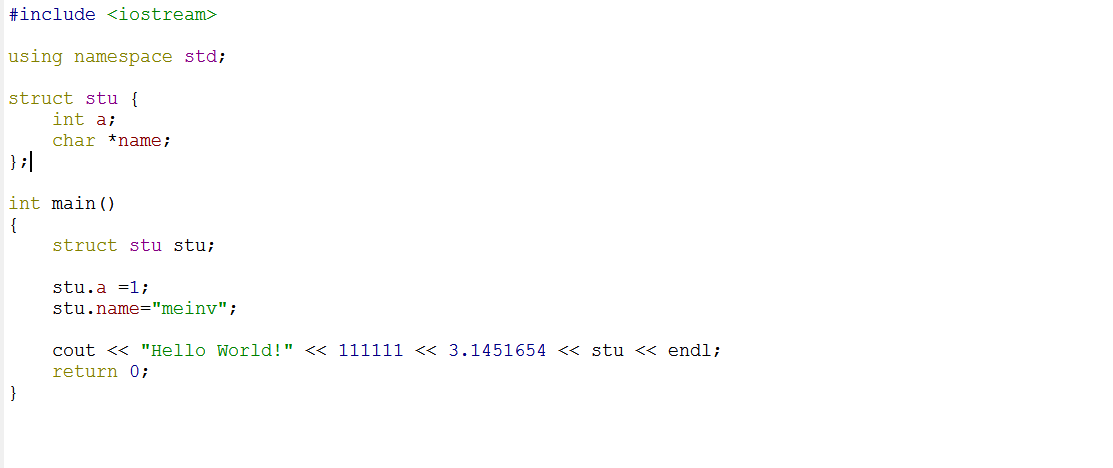
Q1:Cout如何判断输入数据的类型？

Cout的本质是对象，而操作符实际是Cout对象的一个成员函数，即Cout == Cout.operator << (类型)

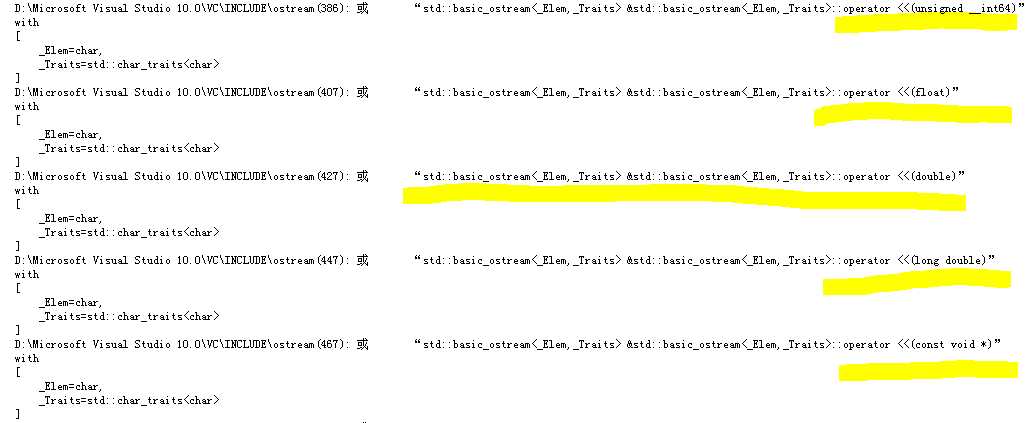
利用C++函数的可重载性，Cout可以即可实现各种输入数据的自动匹配。如：Cout.operator(int )，Cout.operator << (double)等等，实际也是如此。

如，下图是故意向Cout输入一个结构体变量时，编译器的报错信息，可以看到标准流库中，关于operator成员函数的重载。

构造的错误代码：



编译器（win7 64bit-Qtcreater2.5）报错：



* 1. 引用

引用本质是常量指针(对多数编译器来说)，创建引用的目的之一是规避指针可变性带来的风险。

所以，从C的角度理解引用，其本质是常量指针；从C++的角度理解引用

，它可以比喻成变量的别人，且编译器不会为它分配空间。

* 1. 类和对象
     1. 类和对象的区别

在C++里面，类和对象的区别在于编译器是否会为其分配空间，显然，类不会，而对象会。

推广到现实生活中，星球是类，地球是对象；大熊猫是类，成都动物园的大熊猫小美是对象，也就是说，类是存在意识中的概念或者定义，它不占据现实的物质空间，而对象必然是某种物质空间，或者说对象必然占据了一定的物质空间，比如地球，比如小美。

* + 1. 如何定义一个类

<C++里面，如何定义类（思想向）.xmind>

* 1. 类的成员函数

类的成员函数本质是个变量——指针变量，这个指针变量的初始化是在对象被创建时，就好比C结构体中定义的一个函数指针一样，只不过C中需要手动赋值，而C++由编译器自动完成。

C中函数指针的运用：

struct C\_STU {

int age;

string name;

void (\*c\_eat) (string );

};

void fun2(void)

{

C\_STU cstu1; //cpp支持不带struct关键词定义结构体（C中不允许）

C\_STU &cstu1\_1 = cstu1;

struct C\_STU cstu2;

cstu1.age = 20;

cstu1.c\_eat = c\_eat;

cstu1.c\_eat("cstu1 eat: 西红柿");

cstu2.age = 30;

cstu2.c\_eat = c\_eat;

cstu2.c\_eat("cstu2 eat: 红薯");

//对结构体进行引用

cstu1\_1.age = 20;

cstu1\_1.c\_eat = c\_eat;

cstu1\_1.c\_eat("cstu1\_1（引用） eat: 西红柿");

}

* 1. 成员函数指针

int (xxx::\*p)(int,int) = &xxx::func(); //获取成员函数的地址

STU stu;

Stu.\*p(a,b); //\*p相当于func

* 1. 常成员函数

Q4:构造函数是否可以申明为常成员函数？

* 1. 常成员，常对象

1. 类的常成员必须通过构造函数**初始化参数列表**进行初始化，可以通过不同的构造函数初始化；
2. 常对象的普通成员变量，是否需要初始化，取决于不同的编译器，如果未初始化常对象的普通成员变量，有的会报错，有的只会报警，然后编译自动为普通成员变量添加随机值。

而常对象的普通成员变量的初始化可以用构造函数初始化参数列表进行，也可以在构造函数中直接用this指针进行初始化。

1. 常对象只能访问常函数，常函数不能修改成员变量的值
   1. 对象的赋值和复制
      1. 复制构造函数

Stu stu1(10,20);

Stu stu2(stu1); //复制构造函数

Q1:创建对象stu2时，不会调用对应的构造函数，但销毁时会调用对应的析构函数。

A1:实际上创建stu2是不同会调用构造函数，会调用复制构造函数，复制构造是编译器默认添加，格式如下：

Stu::Stu(Stu &s) {

}

另外，此处可以访问到两个对象，其一this表示正在创建的对象；s表示传递的对象。

Q2:那么在函数传递参数和函数返回对象时，也是会调用复制构造函数？？？

A2：会

PS：对象的赋值和复制，在实际开发中主要用于函数的参数以及返回值

* + 1. 复制构造函数陷阱

使用复制操作时，特别注意复制对象的成员变量中是否存在**堆空间**，如果存在，不能使用默认的复制构造函数，否则会导致两个成员变量指向同一块堆空间，如此，无论先释放哪个，都会导致另外一个释放失败，甚至造成一些意想不到的情况。

总结：在CPP中，创建对象都是会调用构造函数（普通构造函数和复制构造函数）

* 1. 静态成员（static）
     1. 访问方式

两种：1和普通成员方式一致（对象.xx,引用.xx，指针->xx）；2是类名：：xx，如Stu::a = 10; （静态成员方法访问方式相同）

* + 1. 零散知识点

1 静态成员变量可作为多个对象的共享内存；

2 静态函数只需在申明时添加static关键字，在定义是无需添加；

3 静态成员变量必须在类体外初始化

类体内：类的定义域和类的申明域

4 静态成员方法不存在this指针，存在this指针本身就矛盾的，因为this指代一个对象，而静态成员方法不需要对象去调用。

* 1. 访问权限修饰符

Public,protected,private

控制成员自身被调用的位置。

访问权限修饰符的意义？

为了安全。实际开发中，绝大多数的成员变量都是私有的，通过公共的成员方法来访问私有的成员变量，实现私有成员变量的赋值和获取。而之所以要采取这种形式，原因是函数本身要比变量更加的安全，它是存在于代码段的，代码段是只读，所以相对于堆栈中的变量更加的安全。

* 1. 友元（friend）

使得访问权限修饰符无效。

Q1:友元是否对成员变量也有效果？

Q2:友元的运用：运算符的实现？

* 1. 继承和派生

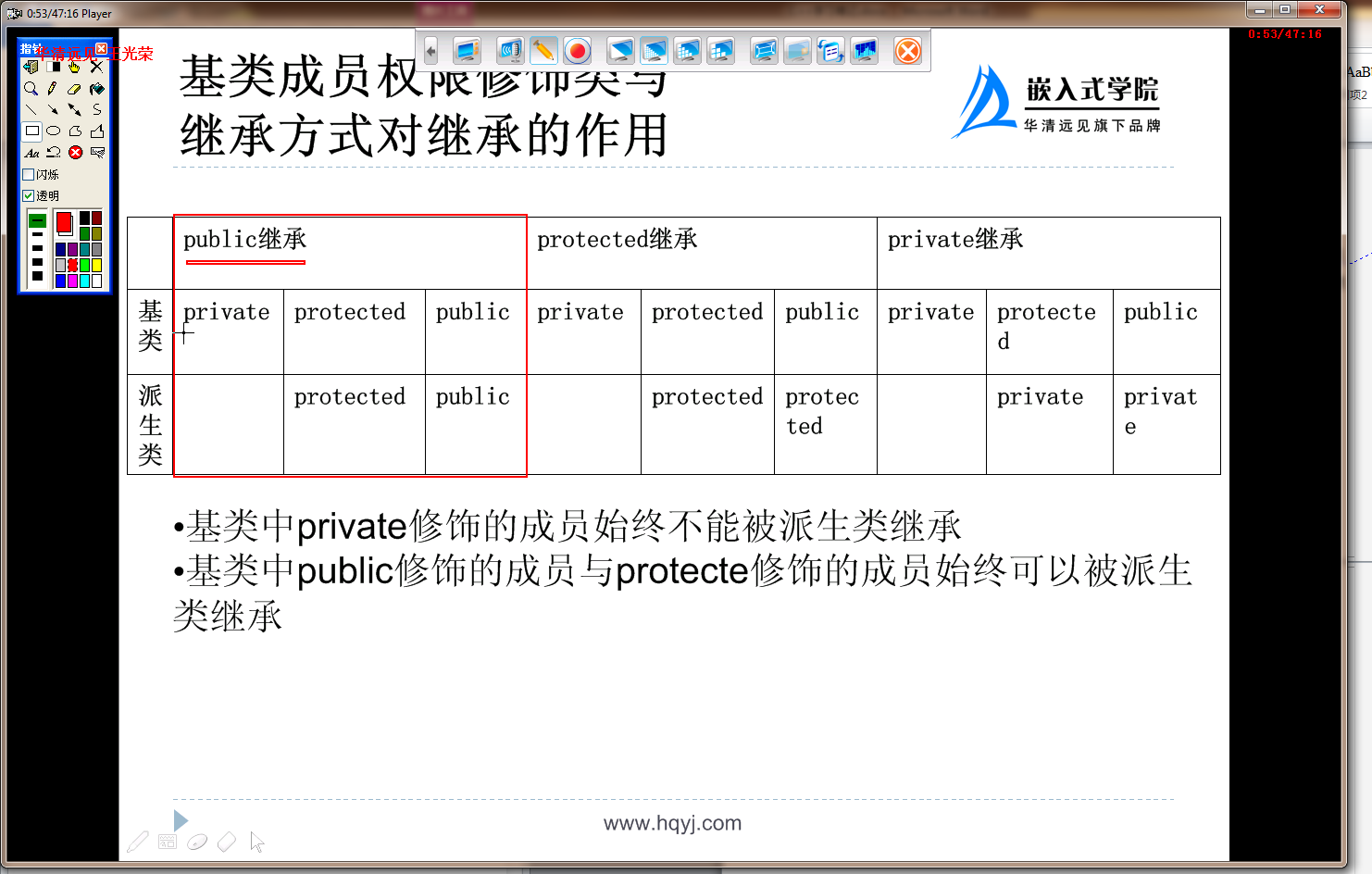
继承类型：

1. 包含关系（只能是单个基类）：派生类具备所有基类的性质；
2. 遗传关系（限制继承：可以有多个基类）：派生类只能具备部分基类的性质，如父母和儿子的关系，儿子只能继承部分父亲的性质，如男人，也只能继承部分母亲的性质，如长相等等，如图所示：



1. 限制继承
   1. 继承方式

继承方式表格：



1. 继承方式表
   * 1. 继承方式详解

1 、派生类不会继承基类的**构造函数和析构函数**

2、私有的成员始终不能被派生类继承

3、共有继承，继承的基类成员权限不变；

受保护继承，继承的基类成员权限都是受保护的

私有继承，继承的继承成员权限都是私有的

PS:受保护和私有的成员，都只能在类体内进行访问。

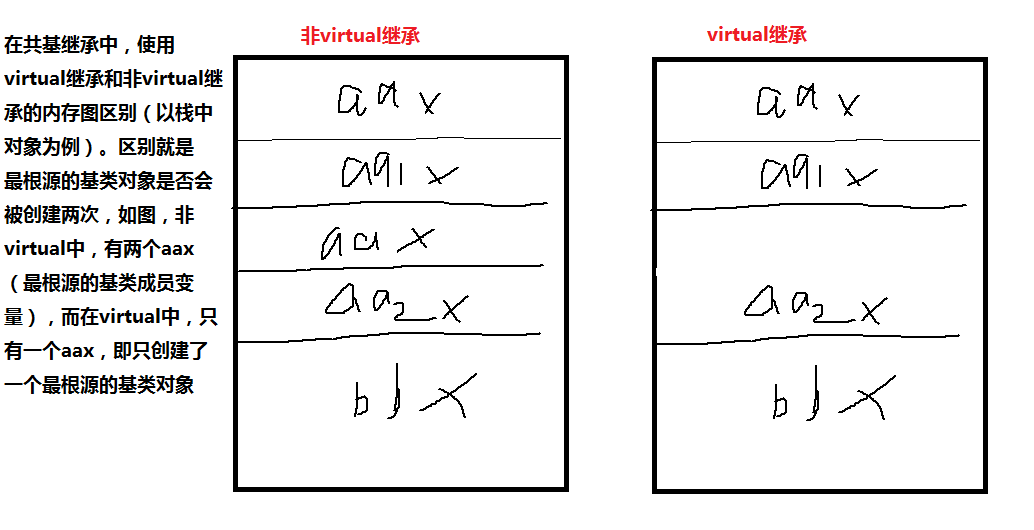
4、派生类创建对象时，会先创建基类的对象。多基对象，每个基类（间接和直接）的对象都会被创建。

* 1. 多基继承和共基继承的二义性，以及继承的本质

1、多级继承的二义性：继承的多个基类之间存在相同名字的成员时，采样**基类1：：成员**，基类2：：成员的方式区分；

2、共同基类产生的二义性：派生类的直接基类继承于同一基类时，此时会产生二义性，因为直接基类存在相同的成员（从同一基类中继承的），消除这种二义性，可以采样**virtual关键词**为直接基类继承的相同成员打上标记，这样派生类在继承时，就不会出现两个相同的成员；当然，也可以使用private继承来规避这个问题。

共基对象的内存图（栈）分配：



1. 共基继承内存图

3、继承的本质：继承就是把基类和派生类**重新组合成新的类型**，在构造对象时，也是通过这个新的组个类型来进行对象创建，所以，派生类可以访问到基类的成员（如果用this指针，甚至通过地址偏移访问所有的成员，包括被保护的成员）

4、C++赋值：同类对象可以相互赋值，派生类可以向基类赋值

* 1. 继承和组合，以及思考

Q1: 在共基继承中，只用一个vritual关键词修饰派生类，为何不能解决二义性？

1、组合和继承从内存角度上看，基本一致（除去二义性的特例），都是创建出新的组个类型，使用新的组合类型构造对象。

2、初始化表：

1）初始化成员变量：变量名（初始化值）；（特别的，如果成员变量是类变量，那么要注意对应类的构造函数使用）

2）初始化基类对象：类名（初始值）（这样做的原因是构造函数名字固定为类名，所以在创建基类对象时，需要使用类名（初始值）的格式）

Q2：继承和组合的运用（思想向）？

A2：组合用于可有可无的功能添加，比如打印机之于PC，所以类的本质属性，不能用组合来实现，而是需要自己写（直接写在类中）或者从父类中继承。

**PS：核心思想——少用继承，多用组合**

Q3:那如果我用了组合来表征一个类的本质属性，有什么不好呢？

A3:可扩展性和可维护性会非常差（为啥？）

PS：设计原则是符合可扩展 可维护，符合面对对象思想，符合高内聚，低耦合的思想的原则，大神们根据设计原则总结的编码规律即是**设计模式**，共23种（常见的工厂，单例，代理等等）。

Q4：继承和派生课后思考题（从面向对象思想的角度回答）

1、单基继承的使用时机：包含关系，能用is表达；

2、多基继承的使用时机：父母与儿子（限制继承）

3、组合的理解：在自己的世界里，你可以认为任何类属性都是你自定义类的本质，但在当你的类需要服务于现实世界时，就不能这么随意的定义这些属性，你必须想清楚你的类的属性哪些是本质的，哪些是可有可无的，对于本质的属性，你需要用继承或者写在类中，对于组合的属性，你可以将其独立出来，以单独一个类的形式扩展在你的类中。

4、组合的不足：过多的依赖关系（高耦合）

* 1. 多态

1、隐藏：派生类将基类的同名成员隐藏，也就是说，派生类的对象在访问同名成员时，默认访问的是派生类的成员，基类的成员被隐藏掉，即用普通方法访问基类成员时就无法访问了，而要想访问被隐藏的基类同名成员，加上基类名词即可。

特别的，当派生类的同名成员方法和基类成员方法**完全相同**时，此时称这种实现叫做**重写**——派生类重写基类的成员方法（注意区别重载：重载是函数名相同，但参数或返回值不同）。

PS：实际开发中，重写占的比例较多，成员方法隐藏的相对较少

2、虚函数：语法上没有没强制规定基类和派生类的虚函数要成对对应，但实际开发中，建议成对对应。

Q：需要理解CPP成员方法的本质！

* + 1. 虚函数

1、回调：函数指针，后写的代码触发先写的代码调用后写的代码；

2、CPP中，虚函数用于回调

3、多态：一个代码多种状态，分成静态多态性和动态多态性。

静态多态性：重载；

动态多态性：指针访问虚函数（引用一样）

如：

void f1(基类 \*p) {

p->eat(); //当eat为虚函数时，最终具体会调用哪个eat函数，要根据传递的指针值确定。

}

f1(&基类对象)； //p->eat调用基类eat方法

f2(&派生类对象)； //P->eat调用派生类eat方法

4、虚析构函数的常见使用： 解决在堆中创建派生类时造成的内存泄露

* + 1. 抽象类（纯虚函数）

1、含有纯虚函数的类都叫抽象类，如果派生类未把所有的纯虚函数定义（只定义了其中几个），此时派生类依然是抽象类。

2、纯虚函数如果被定义（即初始化），那么纯虚函数将成为虚函数。特别的，当派生类将所有继承的纯虚函数都定义，那么此派生类将不是抽象类。

* + 1. 问题

Q1：虚函数的理解

Q2： 1）什么时候使用重写？

基类的方法不满足自身的实际业务需求时，派生类进行重写，而其中基本都是重写虚函数，重写普通函数无意义。无意义的原因在于，重写普通函数无法形成多态，反之，虚函数可以，为什么说虚函数可以形成多态？举个例子，比如定义一个学生类，其中有个eat的函数（虚函数），当男学生和女生学继承学生类并各自重写eat函数时，我们使用学生类的指针访问eat函数，则会根据传递的是男生类还是女生类对象指针来确定学生类指针访问的对象，这样，就形成了多态。

2）什么时候使用抽象类？

3）什么时候用组合？

整体的一部分（可有可无的），非本质，作为功能的扩充。

4）什么时候用继承？

当存在类型上的包含关系时，使用继承；为了解决代码的重用

Q3： 类的本质属性或者方法是适用于所有情况，还是适用于绝大部分情况？

Q4：虚函数和普通函数的区别（思想上，非语法上）

*PS:面试的时候，把面试官引导到自己擅长的领域里（哈哈，没毛病）*

* 1. 访问权限修饰符修饰构造函数，名称空间

访问权限修饰的构造函数和修饰普通的构造函数基本一致（区别看第1点）

1、派生类调用基类受保护的构造函数，只能在初始化表中进行。

2、名称空间的使用：主要用于写重名的函数和类名；

2、名称空间语法格式

1）xxx\_namespace::类名/函数名（适用于多个名称空间同时导入产生冲突）

2）using namespace xxx

using namespace xxx::xxx（名称空间嵌套）

* 1. 泛型/模板

1、模板的语法（定义）：

1）模板函数：模板用于函数中

template <class XX>

XX tml\_func(XX x, XX y) {

return (x+y);

}

其中：template是CPP关键字，class关键字换成typename。定义多个模板参数时，用逗号隔开，每个模板参数都需要有关键字class（或者typename）

2）模板类

template<typename t1,typename t2>

class Stu {

t1 tt1;

t2 tt2;

Stu(t1 x, t2 y);

}

template<typename t1,typename t2> //类模板的函数必须在函数体上添加模板类申明

Stu<t1,t2>::Stu(t1 x,t2 y) {

this->tt1 = x;

this->tt2 = y

}

//使用

Stu<int,String>stu1("xiaoming", 23);

总结：

a.模板类的意义就是可以让类的方法实现动态重载，即在编写程序时再确定类型，然后由编译器自动推导最终需要的参数类型。

b.模板类和模板函数大部分编译器只支持写在头文件中（习惯将定义/声明写在类中），此外，重复导入模板，不会出现多次定义的问题（编译器决定）。

2、模板的本质：模板本质就是**重载**，只不过重载是人为确定参数类型，而模板是在编译时由编译器自动推导参数类似（更加灵活）。