单继承—

继承前提

1. 派生类拥有除了基类的构造函数和赋值重载操作符重载函数以外 的所有成员。
2. 定义派生类前 基类一定一定要有定义，不能只是声明
3. 友元关系不继承，基类与派生类各自的友元关系无必然联系

继承细节

1. protected继承 缓解了继承与封装的矛盾。 （既可以使基类的对象私有不被实例用户直接使用，又能被派生类使用。）
2. -使用基类的对象就两类：①实例化基类对象然后使用 ②被派生类继承使用。
3. 基类和派生类中同名的成员 ，派生类调用时，默认隐藏基类的使用派生类自己的，除非用域名受限（注意：只要同名了就会隐藏 管你成员函数里的参数一不一样）也可以使用using 类名：：函数名 来引入基类的成员函数 。

继承方式

基类中private的成员 在类外或者实例对象绝对不能访问了（在类中还是能访问），但是protected的可以 见上。

继承方式可以调整 private继承的不能调整调整方式记住。 基类和派生类的同名函数不能再调整，如果能调整也是直接所有同载的函数继承方式都被调整了。

Public 继承 时 派生类也叫子类型 子类型即后面虚函数，动态绑定等一系列的前提。

具体来说 基类指针可以指向派生类对象，派生类可以赋值给基类（忽略派生类新增的成员），派生类对象可以使用基类成员函数。

派生类初始化和赋值 初始化 以B->M->D 顺序 调用构造函数。 析构顺序相反 。

如果要调用基类的自定义构造函数 需要在派生类的构造函数的初始化表列中指出

调用基类的拷贝构造函数----

通过this指针转换成指向父类的this指针，可以直接用于对基类成员赋值

聚集能避免封装与继承的矛盾，也实现代码复用

C++ 默认静态绑定 。。

虚函数---动态绑定

只有类的成员函数才能是虚函数。

静态成员函数不能成为虚函数。

构造函数不能是虚函数。

析构往往是虚函数。

Override 跟hide 区别。

基类名受限，基类构造函数中的虚函数调用，非虚函数（只是重名的） 都采用静态绑定。

而且对虚函数的动态绑定往往要指针和引用来实现。

想要用基类指针来访问派生类的成员 先对指针进行类型转换。（不安全，可能会修改基类空间外的值） 转换前可以用dynamic\_cast<要转换的派生类的指针>(原本指向基类的指针)来检验，此操作 合法会返回转换后的指针，非法会返回NULL。

纯虚函数与抽象类

包含纯虚函数的类就是抽象类，抽象类不能实例化对象！（抽象类作为基类）

抽象类的作用？----为派生类提供基本框架和公共对外接口，派生类应当对抽象基类的所有纯虚成员函数进行实现

对于 类 有时候可以通过 特殊的方式 绕过访问控制访问到不可访问的成员，如指针转换（原本指向类的指针强制转成指向数据成员的指针，这时候私有成员可以被访问到。）

要避免这个问题可以使调用的时候形参不传类的指针，而传类的抽象基类的指针，这样使用的时候动态绑定到该类，但是想绕过访问控制就不行了，因为不知道此时这个指针指向谁，指向哪些数据，可以避免 绕过访问控制的 指针访问。

多继承。

必要性 1.用单继承实现的时候要重新定义不被单继承继承，但是也想被继承的东西，混乱。

2.用聚集无法体现类之间的子类型关系。

多继承派生类的定义

命名冲突 多继承的派生类 的 多个基类中 如果有同名成员 访问会出现二义性。

解决方法：直接用基类名受限

重复继承（菱形继承）： 要把被重复继承的那个基类定义为虚基类。

（虚基类 注意事项：1.构造函数由最新派生出的类的构造函数调用。

2.虚基类的构造函数优先非虚基类的构造函数执行。

类和对象

用栈的实例 ----封装和数据抽象的必要性。

多态性的体现 1.一名多用，重载---函数名重载和操作符重载（在同一个作用域用同样的名字对不同的程序实体命名）

2.类属性----一个程序实体对多种类型的数据进行操作

类属函数---一个函数能对多种类型的参数进行操作，

类属类型---一个类型能描述多种类型的数据

以上都是静态绑定

以下常常采用动态绑定

对象类型的多态----子类既属于父类也属于子类（子类赋值给父类，父类不能赋值给子类）

对象标志的多态-----父类指针可以引用或者指向子类（但是有时候是不安全的，因为可能会修改不属于父类的东西）

消息的多态。

多态的好处：容易实现程序上层代码的复用，使得程序扩充容易。

增强语言可扩展性。

类常常称为 “抽象数据类型”

创建对象前类的数据成员不占内存，一般也不能直接赋初值。

数据类型可以是除void外的任意数据类型

如果数据成员 在类之前没被定义完（如成员对象）Error，

但是可以是指针或引用。

非成员函数称为全局函数。

成员函数的内部定义外部定义。、

数据封装带来的好处：类的数据表示的变化不会影响使用者。

对象的创建 直接和间接；

直接 A a， A a[100]

间接 A\*p=new A

This 指针

成员函数中对类成员的访问是通过this进行的。

每个成员函数都有一个缺省参数

构造函数 不带参数或者都带有默认值的叫默认构造函数

构造函数一般public 如果是private那就只能在本类和友元中创建该类对象

创建动态对象数组的时候只能用默认构造函数。

成员初始化表---一般用于初始化 有常值和引用的成员 的对象

如果数据成员有常量或者引用，但是类中没有定义构造函数 ，系统也不会生成非法的默认构造函数，相当于这个类非法，不能创建对象。/

初始化表中赋值按照类中成员定义先后，构造函数体内按照函数内赋值先后。

Free不会调用 析构函数，malloc不会调用构造函数。

创建对象的时候申请了一些资源没有归还的话就要自定义析构函数归还资源

Delete p （p为数组的话） 只会删掉第一个元素。

显式调用 析构函数 只归还对象申请的空间，不会撤销对象。如 a.~string（）；

拷贝构造函数 第一个参数最好是const & 类 后面可以跟其他参数 但是都必须带默认值。

调用拷贝构造函数的情形 1.定义对象 直接用拷贝构造函数来初始化。

2.形参类型不是类的指针或者引用的时候，传入对象作为参数时会调用拷贝构造函数创建形参 。

3.对象作为函数返回值的时候（返回值不是引用或者指针）会调用拷贝构造函数对其初始化。

对于隐式拷贝构造函数 系统默认给的 对普通成员就常规拷贝，对成员对象会调用成员对象的拷贝构造函数拷贝

自定义拷贝构造函数 1，涉及指针成员时，可能会指向同一块内存

2.值传递中可能改变 指针成员指向的东西 ，比如作为函数形参时。

类中由成员对象时 如果用的是隐式拷贝构造函数，那对成员对象调用的是他们自己的拷贝构造函数。

但如果是调用自定义的类的拷贝构造函数，成员对象调用的就是默认构造函数，除非在初始化表自己指出。

封装 继承是面向对象程序设计的两个主要特征。

类构成了模块的基本单位/

常成员函数

对象的成员函数分类 1. 会修改对象状态的

2.仅获取对象状态的。适合用常成员函数 ，避免修改成员。

常成员函数中指针的指向可以改变，且不会报错！！！ 通常认为改变指针指向也会出现问题。

传入对象的指针或者常量对象的引用 （提高效率）！

成员函数加上const修饰符的两个作用 1.在const成员函数定义的地方 提醒编译器不该改变成员的值。

2.调用的时候告诉编译程序 此函数不会改变 成员的值。

静态数据成员

全局变量缺乏保护，但是类之间又要共享。全局变量也能够实现但是不安全。

往往类外定义并初始化 （不绝对。

Class{

Static int a;

}

Int A:: a=0;

静态成员函数只能访问静态成员数据和静态成员函数

没有this指针 访问可以通过对象也可以直接类名受限。

C++支持的观点 类也是对象。

友元能访问该类 的私有和保护成员。

友元类 友元函数 友元成员函数 友元不具有传递性。

矩阵与向量 相乘 的例子 multiply 作为友元函数 提高效率。

转移构造函数的参数只能是右值引用

A(A&& a)

{

}

实参只能是临时对象 或者 即将消亡的对象 编译程序会自动 发现。

指针作为形参 1.提高传输效率

2.能够改变实参的值。

指向常量的指针 例如 const int \* p 不能改变 p指向的数据

Const int \* p

Const int x=0;

P=&x;

X=1;

此类型指针可以指向变量也可以指向常量，指向变量的目的是不修改变量的值。 好处：1.提高效率2.保证不改变实参的值。

对比普通指针 不能指向常量 const int x=0;

Int \* p=&x ERROR

Const A\* p;

p->xxx=xxx; 试图通过指向常量的指针变量来间接修改实参，错误

指针类型的常量 int \* const p=&x; p 不能被修改 （指向不改变） p=&y !!!!错误

Const int \* const p=&x; 指向常量的指针常量 既不能改x的值也不不能改p的指向。

指针作为函数返回值可能会出现的 问题 ：内存回收，堆区的内存被回收之后又被另一个调用 ，导致共用同一块堆区内存的出现问题。

类 抽象数据类型

类的数据成员可以是除了void以外的所以c++类型。

声明时未见定义的只能是引用或指针 否则报错

对象构成了面向对象程序设计的基本计算单位。

Struct 和union中也能定义函数。

把成员函数的实现放在类定义中是建议编译程序按内联函数处理他们。

在类中使用与成员函数标识符同名的全局标识符时，需要在全局标识符前面加上全局与解析符::

在所有函数外定义的对象称为全局对象，在函数内定义的对象称为局部对象。

类定义中的每个数据成员对该类的每个对象都有一个拷贝，而类定义中的成员函数对该类的所有对象只有一个拷贝。 This指针解决对 哪个对象的成员函数进行调用的问题

构造函数调用后不能再调用了，构造函数可以重载，带默认参数的和不带参数的统称默认构造函数 隐式提供的默认构造函数的作用是负责调用成员对象和基类的构造函数。（同理，隐式提供的默认析构函数也是）

显式调用析构函数 s.~string()

只要有提供构造函数，不管是不是默认构造函数，系统都不再隐式提供了

类有成员对象时调用隐式拷贝构造函数，成员对象也会调用其拷贝构造函数，如果类调用自定义的构造函数则成员对象调用默认构造函数，除非初始化表中指出

类作为模块的基本单位。

常成员函数对常对象调用成员函数的合法性进行判断。/

全局操作符重载函数往往需要作为要访问的成员的类的友元函数

加减乘除相等不等的重载

如果作为成员函数重载 ！！注意第一个参数！！如果第一个参数不是this就必须得作为全局函数重载！！！ Complex double+ complex 的实例

单目操作符 前置后置++

六大特殊重载操作符（非运算类）

隐式赋值操作符重载的原型 A& A::operator=(const A& a );

对普通成员常规赋值，对成员对象调用成员对象的赋值操作符重载进行赋值 。

不仅会导致拷贝构造函数类似的问题 还会内存泄漏！！

自己定义的赋值重载操作符不会调用成员对象的赋值重载操作符 ，所以定义的时候考虑进去 成员对象的 赋值

赋值 的重载 检查自身赋值 和指针的赋值 避免悬浮指针和内存泄漏

拷贝构造函数与赋值的区别，初始化时拷贝，重新赋值的时候用赋值

转移赋值操作符重载函数 A& operator=(A&& a)

[]的重载 检查下标越界

New，delete重载 必须是静态成员函数 且静态可以不写 主要重载的时申请和释放空间的功能 默认构造函数除了调用成员对象和基类的构造函数之外，不会对普通成员初始化

Void\* operator new(size\_t size)

Void operator delete(void\*p, size\_t size);

New 重载函数可以有多个，delete重载函数只能有一个，重载之后都不会调用系统提供的默认的了，要使动态对象消亡，必须直接调用对象类的析构函数。

函数调用重载 （）

可以将相应对象的类当作函数使用

主要用于函数对象 该对象除了函数还有状态。

匿名函数

->重载，实现智能指针。计数器。

A\* operator ->()

{ count++;

Return p\_a;

}

类型转换操作符的重载。 也可以通过 带一个参数的构造函数 实现。

Operator int()

{

}

类型转换出现二义性 ： 1 显式类型转换

2.A（int i）前加一个修饰符explicit 来解决 explicit 禁止把A(i)当作隐式类型转换。消除二义性。

小结， 面向对象程序设计实现了数据抽象封装，对程序模块化，复用性，易维护性等