1、组合与继承的基本概念

组合：把自定义类的对象作为新类的数据成员（即对象成员）。
该对象成员必须用新类的**构造函数的初始化列表**来初始化。

继承：在已有的类（基类）的基础上创建新的类（派生类）。体现了代码重用的特性。
单继承：一个派生类只有一个基类（多继承不在本门课的学习范围内）。

可以用派生类对象给基类对象赋值，此时基类对象的数据成员获得派生类对象中对应数据成员的值；也可以用派生类对象的地址给基类指针赋值，或将基类引用定义为派生类对象的别名，此时通过基类指针、基类引用只能访问从基类继承的成员（除非声明了虚函数）。

2、继承方式与访问权限

基类的私有成员在派生类中是隐藏的，不能被派生类新增的成员函数访问。
派生类的成员函数可以通过调用基类的保护或公有成员函数来操作基类的私有成员。
基类的保护成员和公有成员可以被派生类的成员函数访问，且如果继承方式是私有继承，那么它们成为派生类的私有成员；如果继承方式是保护继承，那么它们成为派生类的保护成员。
只有通过**公有继承方式**继承的**基类的公有成员**，才是派生类的公有成员。

3、继承层次中的构造与析构函数

派生类不继承基类的构造函数和析构函数。
如果基类没有默认的构造函数（即基类的构造函数都有参数），那么在派生类的初始化列表中**必须**显式地调用基类的构造函数。
【但是派生类的析构函数不用显示地调用基类的析构函数，这是自动调用的】

构造函数的执行分两个阶段：①执行初始化列表；②执行构造函数体。
在第①阶段中，先执行基类的构造函数（包括它的初始化列表和函数体），再执行派生类新增成员的初始化。派生类的新增成员按照它们在类的定义中声明的顺序来依次初始化。如果新增成员中有对象成员，那么在给它初始化时会调用其类型的构造函数。
在第②阶段中，不对（1）从基类继承的成员、（2）对象成员、（3）常量成员进行赋值。

析构函数的调用顺序与构造函数相反。如果多个对象同时消亡，那么先定义的对象后析构。

对于继承层次，在构造派生类的对象时，从根类的构造函数开始，逐层执行。
派生的对象析构时，调用析构函数的顺序与构造相反，最终调用根类的析构函数。

4、派生类重定义基类的成员函数

派生类新增的成员函数可以与基类的成员函数同名，在这种情况下，派生类只拥有新增的函数，即对基类的成员函数进行了重定义。
要在派生类的成员函数中调用被重定义的基类函数，必须加上 **基类的类名::** 。

5、多态性与虚函数

多态性：不同类的对象实现同名的功能（即同名的成员函数）时执行不同的操作。

静态联编/静态绑定：编译时已决定用哪一个函数实现某一功能。
动态联编/动态绑定：直到运行时才决定用哪一个函数来实现功能。【需要虚函数】

运行时的多态性（基于动态联编）的优点：
①“基类指针->虚函数()”这一代码可以运用于继承层次中各类对象的各不相同的操作；
②即使之后增加了新的类，也不用修改“基类指针->虚函数()”这一代码。

虚函数：在基类中用关键词 virtual 说明，并在派生类中重新定义。在派生类中重新定义时，其函数原型，包括**返回类型、函数名、参数个数与参数类型的顺序**都必须与基类中的原型完全相同。
派生类在对基类的虚函数重定义时，关键字 virtual 可以写也可以不写。不管 virtual 写或者不写，该函数都被认为是虚函数（最好是在重定义时写上 virtual，以易于理解）。

构造函数不能是虚函数。
析构函数可以虚函数且最好是虚函数。**析构函数会继承虚函数性质。**

```
class Derived : public Base {
      int *s;
   public:
      Derived() { s = new int [10]; }
      ~Derived() { delete [] s; } // 如果~Base 是虚函数，那么~Derived 也是虚函数
};
Base *p = new Derived; // 执行 Derived 的构造函数
delete p; // 如果~Base 不是虚函数，会执行 Base 的析构函数，导致内存泄漏
          // 如果~Base 是虚函数，会执行 Derived 的析构函数，释放动态数组
```

6、纯虚函数与抽象类

纯虚函数的声明：virtual 类型 函数名（参数表）= 0;
纯虚函数没有定义，也没有函数体，因此不能执行纯虚函数。
纯虚函数在基类中说明为纯虚函数，并在某个派生类（不一定是直接派生类）里重定义。

如果一个类中有一个纯虚函数（包括从基类继承而没有重定义的），则该类为抽象类。
不能建立抽象类的对象。抽象类也不能用作参数类型、函数返回类型或显式转换类型。
可以建立抽象类的指针或引用，指向它的派生类的对象，从而实现运行时的多态性。