1. 静态联编和动态联编
   1. 多态分类
      1. 静态多态 函数重载
      2. 动态多态 虚函数 继承关系
   2. 静态联编
      1. 地址早绑定 编译阶段绑定好地址
   3. 动态联编
      1. 地址晚绑定 ，运行时候绑定好地址
   4. 多态
      1. 父类的引用或指针指向子类对象
2. 多态原理解析
   1. 当父类中有了虚函数后，内部结构就发生了改变
   2. 内部多了一个 vfprt
      1. virtual function pointer 虚函数表指针
      2. 指向 vftable 虚函数表
   3. 父类中结构 vfptr &Animal::speak
   4. 子类中 进行继承 会继承 vfptr vftable
   5. 构造函数中 会将虚函数表指针 指向自己的虚函数表
   6. 如果发生了重写，会替换掉虚函数表中的原有的speak，改为 &Cat::speak
   7. 深入剖析，内部到底如何调用
   8. ((void(\*)()) (\*(int\*)\*(int\*)animal))();
   9. 猫吃鱼的函数调用（编译器的调用）
3. 多态案例 – 计算器案例
   1. 早期方法 是不利于扩展
   2. 开发有原则 开闭原则 -- 对扩展开放 对修改关闭
   3. 利用多态实现 – 利于后期扩展，结构性非常好，可读性高， 效率稍微低，发生多态内部结构复杂
4. 抽象类 和 纯虚函数
   1. 纯虚函数写法 virtual void func() = 0;
   2. 抽象类型
   3. 抽象类 不可以实例化对象
   4. 如果类 继承了抽象类， 必须重写抽象类中的纯虚函数
5. 虚析构和纯虚析构
   1. 虚析构
      1. virtual ~类名() {}
      2. 解决问题： 通过父类指针指向子类对象释放时候不干净导致的问题
   2. 纯虚析构函数
      1. 写法 virtual ~类名() = 0
      2. 类内声明 类外实现
      3. 如果出现了纯虚析构函数，这个类也算抽象类，不可以实例化对象
6. 向上类型转换和向下类型转换
   1. 基类转派生类
      1. 向下类型转换 不安全的
   2. 派生类转 基类
      1. 向上类型转换 安全
   3. 如果发生多态
      1. 总是安全的
   4. 父类中如果写了虚函数，而子类没有任何重写，有意义吗？
      1. 没有意义