1. 静态联编动态联编
   1. 静态多态和动态多态
   2. 静态多态：函数重载，运算符重载
   3. 动态多态：
      1. //先有继承关系
      2. //父类中有虚函数，子类重写父类中的虚函数
      3. //父类的指针或引用 指向子类的对象
   4. 静态多态在编译阶段绑定地址，地址早绑定，静态联编
   5. 动态多次在运行阶段绑定地址，地址晚绑定，动态联编
2. 多态原理
   1. 当父类写了虚函数后，类内部结构发生改变，多了一个vfptr
   2. vfptr 虚函数表指针 ---- > vftable 虚函数表
   3. 虚函数表内部记录着 虚函数的入口地址
   4. 当父类指针或引用指向子类对象，发生多态，调用是时候从虚函数中找函数入口地址
   5. 虚函数 关键字 virtual
   6. 利用指针的偏移调用 函数
      1. ((void(\*)()) (\*(int \*)\*(int \*)animal)) ();
      2. typedef void( \_\_stdcall \*FUNPOINT)(int);
      3. (FUNPOINT (\*((int\*)\*(int\*)animal + 1)))(10);
3. 多态案例 - 计算器案例
   1. 设计抽象计算器类，分别实现加减乘计算，继承于抽象计算器类，重写虚函数
   2. 利用多态可以调用不同计算器
   3. 多态的好处
      1. 代码可读性强
      2. 组织结构清晰
      3. 扩展性强
   4. 开闭原则： 对扩展进行开放 对修改进行关闭
4. 纯虚函数和抽象类
   1. 语法： virtual int getResult() = 0;
   2. //如果一个类中包含了纯虚函数，那么这个类就无法实例化对象了,这个类通常我们称为 抽象类
   3. //抽象类的子类 必须要重写 父类中的纯虚函数，否则也属于抽象类
5. 虚析构和纯虚析构
   1. 虚析构语法：
      1. virtual ~Animal(){}
      2. 如果子类中有指向堆区的属性，那么要利用虚析构技术 在delete的时候 调用子类的析构函数
   2. 纯虚析构语法：
      1. virtual ~Animal() = 0;
      2. Animal::~Animal(){ .. }
      3. //纯虚析构 需要有声明 也需要有实现
      4. //如果一个类中 有了 纯虚析构函数，那么这个类也属于抽象类，无法实例化对象了
6. 向上类型转换和向下类型转换
   1. 父转子 向下类型转换 不安全
   2. 子转父 向上类型转换 安全
   3. 如果发生多态，那么转换永远都是安全的
7. 重载、重写、重定义
   1. 重载
      1. 函数重载
      2. 同一个作用域下，函数名称相同，参数个数、顺序、类型不同
   2. 重写
      1. 子类重写父类中的虚函数，函数返回值、函数名、形参列表完全一致称为重写
   3. 重定义
      1. 子类重新定义父类中的同名成员函数，隐藏掉父类中同名成员函数，如果想调用加作用域
8. 多态案例2 - 电脑组装案例
   1. 利用多态实现 电脑组装