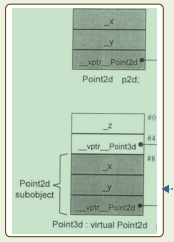
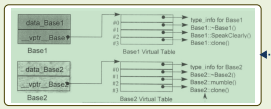
对象-Function语意

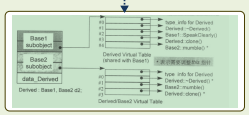
- inline函数 → 调用处直接扩展
  - 过程 → 判断函数inline能力
    - **inline能力不足** → 将其转换为static函数，进行调用
    - 能力足，进行转换 → 1. 形式参数： 可能需要引入临时变量
      2. 局部变量：名称独一无二， 函数调用的封闭区段内使用
      - *过度使用* → 1. 代码量暴增
        2. 如果存在inline嵌套，可能无法扩展
- 分类
  - static 函数 → 为static 变量设计
    - 特殊性 → 只能访问static成员，不提供访问nonstatic 成员接口（没有默认this指针）
      用对象调用函数与用类调用函数效果一致，
      编译器直接mangle为外部函数使用
  - 1. nonstatic 函数
    2. static 函数
    3. virtual 函数
    - nonstatic 函数 → 普通成员函数
      - **调用要求： 与普通函数代价一致**
        调用方式：与普通函数也一致，根据函数名寻址 → 1. 补全函数名， 重写为唯一外部函数(通常加上类名)
        2. 扩展函数参数（[const] class* this ）
        3. 成员函数转换为非成员函数【外部函数】
    - 指向成员函数的指针 → 根据函数类型有所不同，但使用时都可以当作函数使用
      funcpt = &Class::func, obj.*funcpt || pointer->*funcpt
      - *nonstatic 函数* → 类函数的地址
      - *虚拟函数* → 函数在虚表中的slot
        需要进行特殊转换
    - **virtual 函数** → **多态的核心**
      - *调用的方式* → 指针与对象调用方式不同
        - 指针访问 → 经由虚函数表获得函数地址，进而调用（也需要传入this）
        - 对象访问 → 需要访问的函数已经是确定的， 不需进行虚拟机制决议来确定访问哪个函数，所以访问方式与nonstatic函数一样，根据函数名来访问
      - 布局的考虑 → 如何实现执行期基类指针指向派生类，并调用派生类的函数
        - **基础布局** → 1. 如果有虚函数则创建虚函数表并保存指向该表指针
          2. 每个虚函数对应此表中的一个slot, 通过vptr 与 slot 获得该函数
          - 函数对应的slot → 应该与函数声明的顺序相关
          - *虚拟继承* → 虚拟继承中的虚基类与派生类的起始地址并不一致【offset】
            需要类似于多重继承中非首个基类指针的转换工作
            - 图解 →



          - **继承一个有虚函数的基类： vptr不变， 虚函数slot不变**
            - 多重继承 → 所有基类中的vptr成为派生类中的vptr， vptr不变
              派生类会根据自己虚函数的情况重写虚表
              这样基类指针指向派生类时，使用的虚表便是派生类的虚表
            - **重写虚表的规则** → 1. 若是不重新定义虚函数，则将父类中虚函数地址拷贝到虚表中
              2. 若是重新定义虚函数， 则写入自己的虚函数地址
              【1.2 保证了派生类与基类访问同样虚函数的形式是一样的， 函数SLOT不变】
              3. 新建虚函数或从其他基类继承的虚函数， 新的虚函数地址也会被写入到虚表中， 新的虚函数与基类没有关联， 不会产生冲突。
            - 1. 对于派生类虚函数表【可能与某个基类共享】，需要将所有继承而来的虚函数都写入表中， 供派生类指针直接使用
              2. 其余的非共享基类虚函数表， 重写所有派生重定义的虚函数覆盖原有函数地址
              - 图解 →



              - 原始基类
              - 派生类中布局 →



              - **基类指针被赋值为派生类对象地址** → 基类指针赋值为派生类对象地址时，需对当前this指针【当前this指针值指向派生类起始位置】进行偏移调整, 使其指向当前基类的起始位置： this + sizeof(base1)， 否则对非多态对象的引用都错
                - 在调用虚函数时产生问题 → 在调用虚函数时， 向虚函数传递的指针就是指向基类的而不是派生类的地址，需要二次调整，将其调整到指向派生类起始地址
                - cfont → 将虚函数表SLOT存储的函数地址变为一个聚合体，包含函数地址与偏移值
                - thunk → 在需要转换的虚函数表的slot中，将存储的虚函数地址值改为一小段thunk代码实现指针转换