初学者编程入门：学习C 的最大难度 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/257/2021_2022__E5_88_9D_E5_AD_A6_E8_80_85_E7_c67_257592.htm 困难度之一“ C 是个难学易用的语言 ”，这句话相信很多人都心有戚戚。C 的学习难度，一在于语言本身太多的“ 幕 ”，另一个就在于 "paradigm shift"（思考模式的移转）。传统语言如 C, Pascal, Basic, Fortran...，除了模样看起来稍有不同，基本上都是函式 call 来 call 去，大同小异，很容易掌握。你想做的动作，在 code 中都看得一清二楚。你所看不到的，也不过就是编译器为你的函式加上用以处理堆叠的一小段码（prologue 和 epilogue），这一小段码基本上做的是 housekeeping 工作，你没看到也没有关系，并不影响你对程式逻辑的思考。C 不一样，C 有太多和程式逻辑息息相关的动作是编译器为我们加上去的。换句话说 C 编译器为我们“ 加码 ”。如果不识清这一节，学习C 有如雾里看花，雾非雾，花非花。编译器为我们的 C 程式加了什么码呢？很多！物件诞生时 ctor 会被唤起，物件死亡时 dtor 会被唤起，这都是加码的结果。ctor 中设定vtpr 和 vtbl，这也是加码的结果。new 单一物件时会产生 memory block cookie，new 物件阵列时会产生一个内部结构记录着 object size 和 class ctor...，这也都是布幕后的工作。可以说，程式码中看不到而却必须完成的所有与程式逻辑有关的动作，统统都是 C 编译器加码后的结果。当“ 继承 ”发生，整个情况变得稍微复杂起来。“ 多重继承 ”又更复杂一些，“ 虚拟继承 ”又再更复杂一些。这些布幕后的主题，统可归类为所谓的 C object model（物件模型）。如果不知道这些底

层机制，你就只能够把 "make destructors virtual in base classes" 或 "never treat arrays polymorphically"这类规则硬背下来，却不明白它的道理。用一样东西，却不明白它的道理，林语堂如是说：“ 不高明 ”。只知道 how，不知道 why。困难度之二C 的第二个学习难度在于 "paradigm shift"（思考模式的移转）。别说自己设计classes 了，光使用别人的classes，就都是一种思考模式和行为模式的移转。MFC（或 OWL 或 VCL ）programmer 必然能够领略并体会其中的意思。使用所谓的 application framework（一种大型的、凝聚性强的、有着物件导向公共基础建设的 class library），你的码和 framework 之间究竟是怎样的关系呢？framework 提供的一大堆可改写的虚拟函式的意义与价值究竟在哪里呢？为什么 framework 所设计的种种美好性质以及各式各样的演算法竟然可以施行于我们自己设计的 class types 身上呢？framework 被设计时，也并不知道我们的存在。这正是物件导向中的多型（polymorphism ）的威力。稍早所说的 C 物件模型，偏属程式设计的低层面；这里所说的思考模式移转，则是程式设计的高层面。能够把新思维模式的威力发挥得最淋漓尽致的，当推物件导向的 polymorphism（多型）和 generalization（泛型）。如果你没有使用这两项特性，等于入 C 宝山却空手而归。反覆磨炼，循环震荡想像 C 是一把用来解决程式问题的刀，要它坚韧，要它锋利，就必须经过多次的回火，在高热和骤冷之间炼。初学 C 语法（syntax）之后，你应该尽快尝试体验 polymorphism（大致而言也就是虚拟函式的运用）。等到对 OOP 的精神有了大局掌控的能力，但对 C 的许多小细节不甚清楚，就是回到C 物件模型 炼的时机。成长，是在高阶（polymorphism）

和低阶（object model）之间反覆震荡，才能够震荡到更高的位阶，而不是平平庸庸于中阶（C syntax）的一滩死水。