并非偏见也驳“驳’C语言已经死了’”PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/286/2021_2022__E5_B9_B6_E9_9D_9E_E5_81_8F_E8_c67_286800.htm STL的代码并不优雅，缺乏functional programming机制支持的C 对于实现algorithm非常的牵强，比方我要find(v.begin(), v.end(), compare).的时候(v是一个自定义的结构)，我必须在函数外面写一个比较函数，如果要带一些上下文的话还得写一个functor类，非常的丑陋不堪，实用性大打折扣。而FP系的语言来说，可以非常自然的写一个匿名函数。STL里所标榜的容器，算法等概念，在FP里早就原生支持了，而且要优雅的多。至于NT代码这个我没看过不好说，但是据说代码里有不少当初程序员留下来的抱怨BUG及设计失误的话。 >> 内存管理是程序设计中最经典的话题。GC无疑是内存管理一个伟大的变革，但是我只是把它看作内存管理的一个解决方案，而认为不是唯一的解决方案。比GC更加优雅的方案不见得没有。我比较倾向于在特定的情况下选择合适的内存管理方案，而不是没有任何选择的余地，而这正是C/C 的伟大之处。 所有那些GC语言(如Java、C#等)均把这个解决方案强加给程序员，这一定程度上来说减轻了程序员的负担，但是也同时约束了程序员的主观能动性。"分配内存和释放内存在C语言中都是很慢的"?不知道作者从哪里获得的结论。 实话说我也不喜欢GC，没有GC的C也可以工作的很好，但是对于FP系的语言来说没有GC是无法正确工作的，所以我还是得接受GC这个东西。当然我更喜欢的是将两者互相结合的方式。 >> C/C 语言本身确实没有太多MultiThead的支持，这种情况在C 0x出来后可望

改变。但是，请记住C/C 永远倾向于你使用成熟的库来解决问题。 C/C 不能适应未来多核时代的发展，这个会是它没落的最大原因。库不能真正的解决问题，我们需要的是在语言层面的进一步发展。 >> 指针是C/C 过于灵活的体现。使用指针的代码可以写得很丑陋，但一样可以很优雅。这一点上用何种语言不会有区别。我相信，可以写出优雅的Java代码，那么也一定可以写出同样优雅的C/C 代码。而反之则未必(因为有些C 某些范式是Java所不能支持的)。C/C 语言中的选择太多，这的确是令人困惑的，但不见得是劣势。我对C/C 程序员的建议是，多了解和使用C 标准库，而不是过于纠缠指针相关的细节。 >> 算法优化是程序设计的关键。但是通常情况下，所有语言(包括C/C )的程序员研究的是关键路径的优化。研究*p 是不是比p[i]快?我相信这是标准库的实现者要考虑的事情。所不同的是，C/C 程序员也可以和标准库的作者一样去考虑这些细节，而其他语言的程序员被剥夺了这个权利。 说到优化，话题就多了。我曾经向C#的Dictionary中插入了1亿条整数(从1万多个文本文件中读入)，结果发现程序运行了整整一个下午仍然没有完成。而我改用C 的std::map，20分钟就搞定了。再试试对50万条自定义的结构体数据进行排序，我相信你和我一样，会深深喜欢上C 的的高效而优雅。多年以前程序员们还在C程序里面内联汇编以实现代码级的优化，但是如今已经没有人这么做了，因为CPU越来越复杂了，大多数情况编译器做的比手工的要好。现如今的Java/.NET的JIT引擎也已经能够达到非常高的优化水平，在性能上C代码的优势已经越来越不明显了。对于未来而言代码级的优化也已经不再是重点，哪个语言可以适应多核的发

展，谁就将成为性能的王者。 >> 新生的语言，必然会在吸收旧的语言上基础上进行改进。看一个语言的生命力，并不在于看它某些地方存在的不足。事物会发展，并趋于完善。相信C 0x出来后，C/C 语言又将获得新的生命力。单看Java、C#等几个新一代的语言，其中有如此多的C 烙印，就证明了C/C 的影响是巨大的。动不动说一门语言死了，是一种浅薄。 说一门语言死了，不是说完全消失，而是退出主流开发语言行列，逐渐的被边缘化，这些年鼓吹C/C 的人已经越来越少了，在很多开发领域C/C 的地位已经被Java、.net、脚本语言等所取代。C 0x出不出来已经不重要了，倒是C /CLI的出现带给C 一些新意，不过虽然我很欣赏C /CLI，但是它不会成为主流。在多核到来的时候目前编程语言还没做好准备，未来我们要面临的不是2核4核而是百核千核这样的规模，这不光要在算法领域继续发展，编程语言也要来一次重大的变革才能适应这种发展，至于方向在哪里，FP系的语言或许会给你带来一些启示。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com