



下载APP



开篇词 | 时至今日，如何更好地拥抱现代 C 语言？

2021-12-06 于航

[课程介绍 >](#)**讲述：于航**

时长 11:15 大小 10.32M



你好，我是于航，目前在 PayPal 做软件研发与技术管理工作。

也许一些同学对我比较熟悉，我之前曾在极客时间开设过一门《WebAssembly 入门课》，并且还是多个 WebAssembly、C++ 相关每日一课视频的作者。而今天，我又设计了一门新课，想要带你从不同的视角来学习 “C” 这门语言。

我相信来学习这门课的大部分同学，都或多或少接触过一些 C 语言的基础知识。但是，我认为掌握 C 语言的基本语法并不困难，更重要的是能够灵活、高效地使用这门语言，并**通过观察语言背后机器的执行细节，来深入了解关于编译优化、程序执行，以及计算机结构等其他相互关联的知识。**



作为 WebAssembly 技术的研究者和使用者，C 语言是我在工作中使用最多的语言之一。由于 C 语言语法简单、抽象层次较低，我能够通过它在进行原型验证时精确地控制程序的运行状态。另一方面，在接触操作系统、Unix 系统编程、语言运行时，以及系统库等相关内容时，我更深切地感受到了了解 C 语言对于深入理解这些内容的重要性。

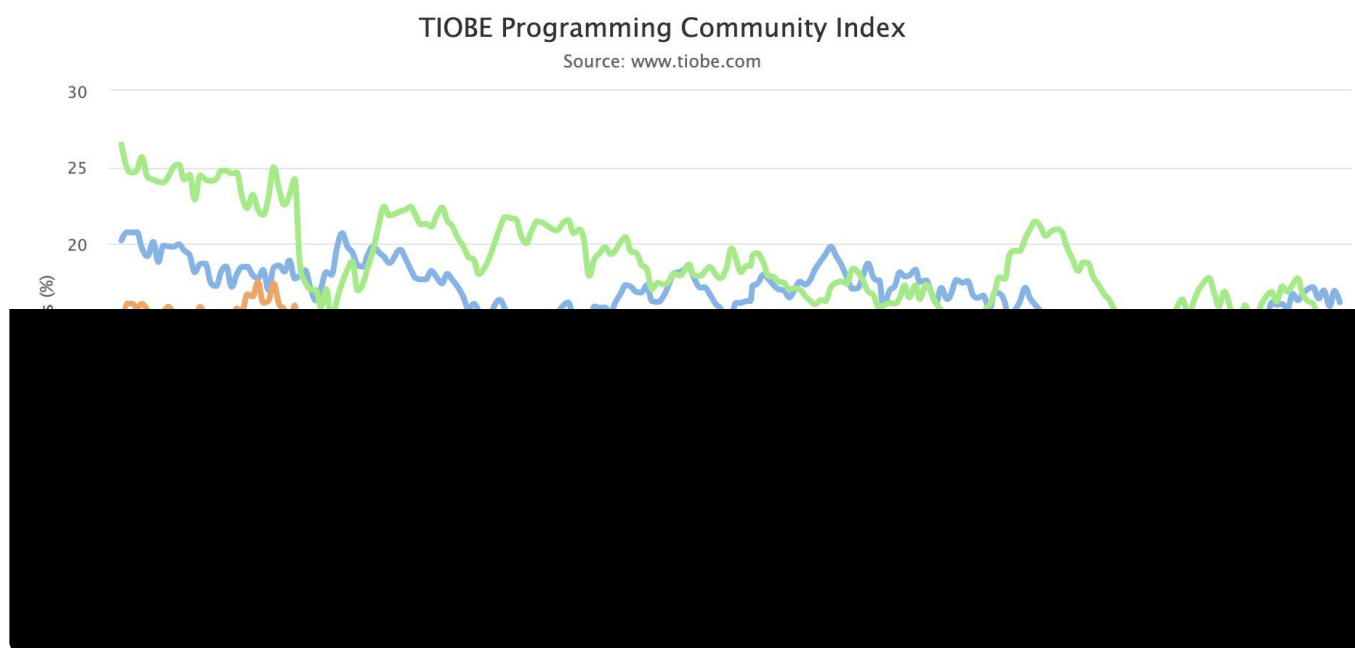
因此，这门课将不会介绍 C 语言的语法细节，而是结合 C 核心语法、汇编代码，以及计算机体系结构等相关知识，来**讲述 C 语言、应用程序和操作系统三者之间的协作关系**。

C 语言过时了吗？

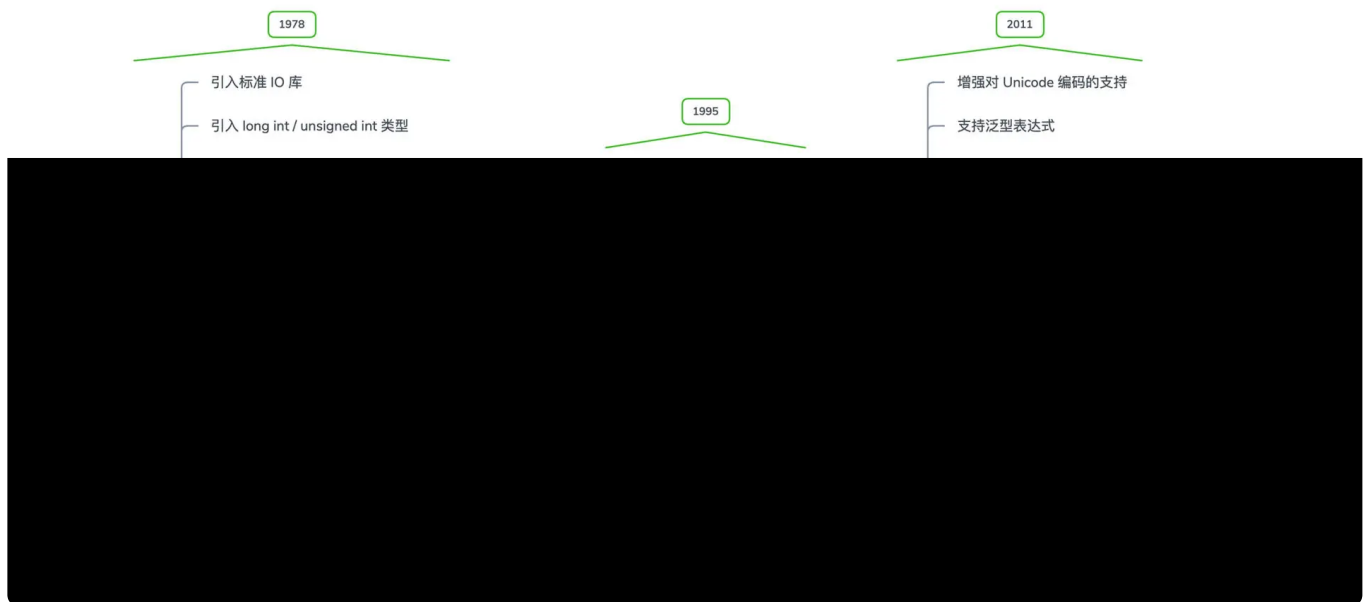
说到学习 C 语言，很多人都会有这样的问题：在新编程语言层出不穷的今天，C 语言已经诞生了这么久，会不会马上就要过时了？

对此，我的回答是：C 语言还远远没有过时，相反，学习 C 语言是非常有必要的。

从下图（图片来自 <https://www.tiobe.com/>）中可以看到，自 2000 年以来，C 语言便一直处于 TIOBE 编程语言榜单的前两名。作为全球最知名的，反映编程语言热门程度的榜单，TIOBE 指数直接表明了哪些语言是应该被及时掌握的。它对世界范围内编程语言的整体走势有着重要意义。



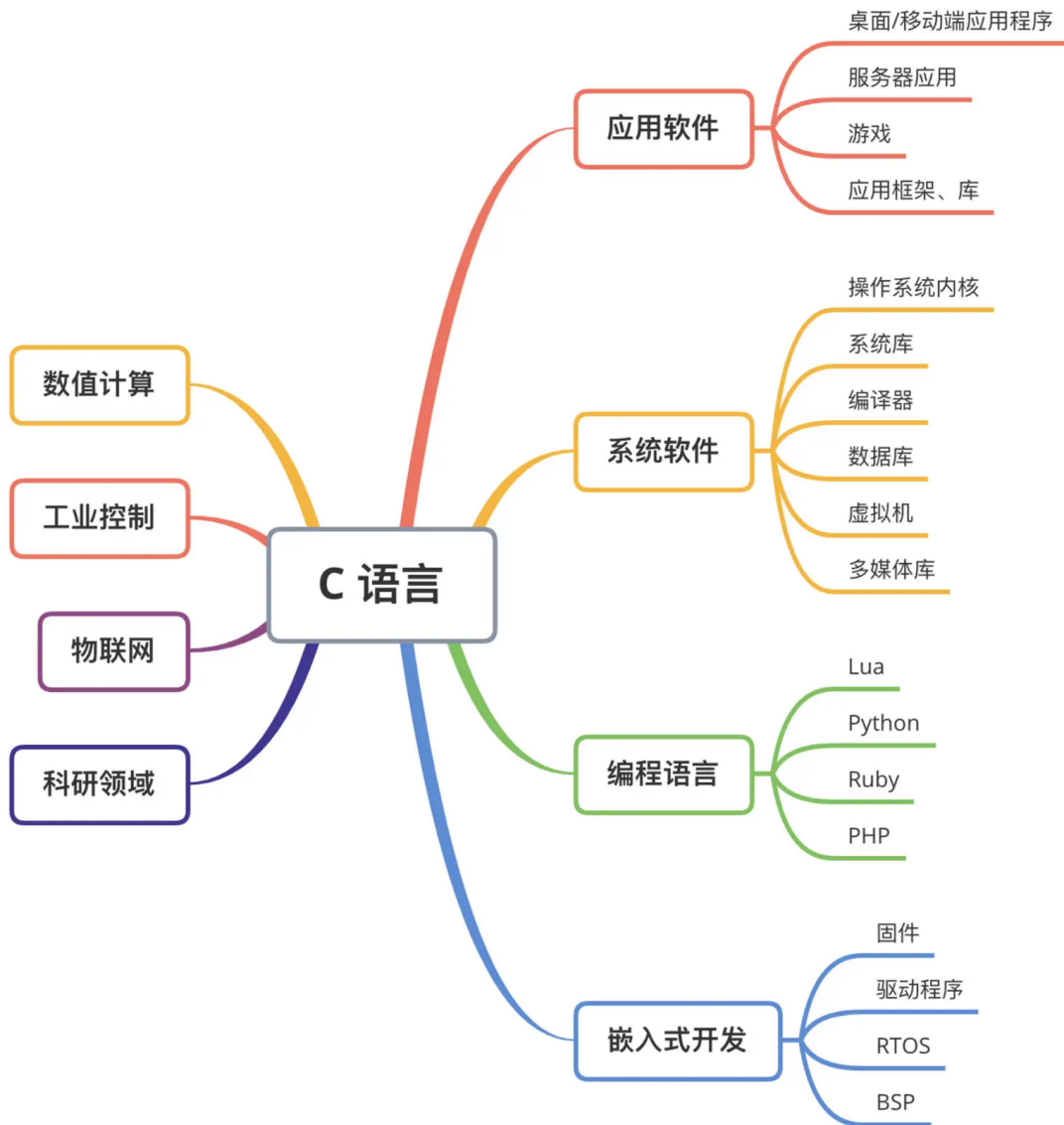
不仅如此，C 语言本身也处在不断的“进化”过程中。下图展示了 C 语言诞生至今的几次重大版本的发布内容。可以看到，C 语言在发展的同时也在尽可能保持着自身的精简。而最近 C2x 标准草案的制定也证明了时至今日的 C 语言，仍然“老当益壮”。



总之，C 语言远远没有过时。其实，**C 语言问世几十年来，一直都是使用最广泛的编程语言之一。**

作为一种静态编译型语言，C 语言有着其自身所适合的应用场景。确实，我们无法使用它来编写 Web 应用，也无法使用它来高效快捷地构建深度学习应用。但你要知道的是，用于支持这些应用正常运行的系统组件，乃至操作系统内核，都是使用 C 语言编写的。**现如今，这个世界上几乎所有重要的软件都与 C 有着直接或间接的关系。**

C 语言被广泛应用于实现操作系统、嵌入式系统应用、编译器、数据库、驱动程序，以及服务器应用等较为底层和基础的系统级程序。除此之外，C 语言在诸如数值计算、工业控制、物联网，乃至科研领域也有着重要的应用。



那么，为什么 C 语言使用如此广泛呢？我认为原因有两个，一是它有着远优于大部分其他语言的程序精确控制能力，二是它高效的运行时性能。


精确控制程序

C 语言在设计上对平台独立的低层次汇编指令进行了适度的抽象，在大多数情况下，它的代码可以直接映射到硬件平台上的机器指令。因此，我们能够更加灵活地控制程序的具体表现行为。

我们来看个例子吧。使用 C 语言，我们甚至可以直接控制代码中某个变量值的存放位置，视情况来决定将它存放在栈内存中，还是寄存器中。如下图所示，这里，左右两个窗口中

相同背景颜色的代码行，表示了 C 代码与其对应的汇编代码。可以看到，左侧 C 代码中第三行变量 `x` 的值被存放到了 `ebx` 寄存器中。

在某些特殊场景下，我们甚至可以直接在 C 代码中嵌入汇编代码，以更细粒度的方式来控制程序的执行，或直接与更底层的硬件进行交互。比如在下面这段代码中，我们使用 `asm` 关键字嵌入了两行汇编代码，你能猜出它们做了什么吗？可以在评论区留下你的答案。

 复制代码

```
1 #include <stdio.h>
2 int main(void) {
3     int src = 1;
4     int dst;
5     asm ("mov %1, %0\n\t"
6         "add $1, %0"
7         : "=r" (dst)
8         : "r" (src));
9     printf("%d\n", dst);
10 }
```

C 语言更贴近底层硬件的这一特征，就使得**它非常适合被应用在需要细粒度控制资源，或与底层硬件打交道的场景中**。实际上，这其中有很多优秀项目都是你所熟知，甚至在日常工作中都会经常使用到的。比如代码版本管理工具 Git、高性能 Web 服务器 Nginx、高性能 NoSQL 数据库 Redis，以及最知名、代码行数最多的开源项目 Linux 内核等。因此，如果你想要读懂它们的设计、搞懂它们的原理，那了解 C 语言便是一个必不可少的过程。

需要注意的是，C 语言也保持了高级语言的部分特性，比如提供了接近于自然语言的语法和关键字，且源代码可以做到独立于机器的分发和使用。因此，使用 C 语言，我们既能享受到它作为高级语言时的自然语法和特性，又能够做到同低级语言一样，精确地控制程序的执行细节，甚至直接与硬件交互。

高效的运行时性能

很明显，使用 C 语言正确实现的程序可以享受到最高的运行时性能。而通过内联汇编，它甚至可以与直接编写的汇编代码相比肩。

需要注意一点：和其他语言不同，C 并未提供语言内置的诸如垃圾回收（GC）等可能导致额外运行时开销的特性。在提升了性能的同时，你也需要正确处理内存的分配与回收过程，以避免出现诸如内存泄露等问题。在这门课里，我会为你介绍如何正确地编写 C 代码，来避免类似的问题。

学习 C 语言，为什么是你修炼编程内功的必经之路？

这时候，有同学可能会问了：我并不想做嵌入式、操作系统这些底层开发，平时的开发工作感觉 Java、Go 这些语言也够用了，为什么还要学习 C 语言呢？

对此我想说的是：即使你不使用 C 语言进行开发，深入学习 C 语言，也是你修炼内功、成为编程和计算机高手的必经之路。

为什么这么说呢？主要有三个原因。

第一，C 语言作为一门简单通用的早期编程语言，是 Go、Objective-C、C#、Java 这些高级编程语言在设计时所参考的“**原型**”语言。可以说，C 语言就是众多编程语言中的“九阳神功”，相信在你深入了解 C 语言后，再去学习其他语言，也会变得轻松许多。

第二，我们上面也提到过，C 语言是目前众多流行操作系统、编译器、上层实用软件与各类系统组件，乃至嵌入式开发所使用的源语言。因此，学习 C 语言也让我们具有了能够去**探索优秀软件内部实现细节**的能力，而这通常也是优秀工程师提升自我实力的一种快捷方式。

第三，C 语言的抽象程度非常低，是最适合用来帮助理解计算机系统底层运作机制的语言。在学习如何高效使用 C 语言的过程中，你将会学习到有关高速缓存、内存、寄存器，以及函数调用等相关的内容，而这无疑对你提升自身实力有着巨大的帮助。总之，**深入学习 C 语言之后，我们就拥有了从“更低纬度”理解计算机运作机制的能力。**

网上流传的一个不太恰当的比喻是：学习 C 语言正如我们通过学习营养学、健康学来为自己合理地制定饮食计划。当然我们也可以选择直接购买市面上已经装配好的各类营养餐品，但当身体状况并没有按预期发展时，我们并不清楚问题出在哪里。并且，当需要实现一些特殊的定制化需求时，可能市面上的产品功能总会与我们的目标有所出入。

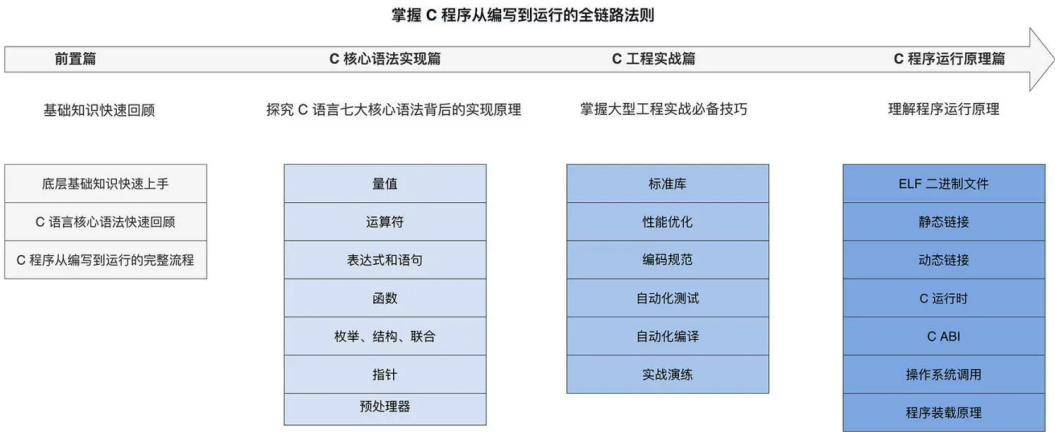
编程也是如此，相较于直接使用诸如 Python、Java 等高抽象粒度的编程语言，学习 C 语言能够让你从基础层面了解程序是如何工作的。**理解了计算机系统的底层运作机制，你在设计更复杂、性能更高的程序时，便能够得心应手、融会贯通。**

所以，要想深入理解计算机系统的运行原理，学习 C 语言是一个必经之路。

这门课是怎么设计的？

为了达到灵活、高效地使用 C 语言，并借此深入理解计算机系统运行原理的目的，只掌握 C 语言的基本语法是远远不够的。我们还需要**深入到 C 语言的内部，去了解一个 C 程序从编写到编译，再到被运行的整个流程细节**。只有做到“知其然”并“知其所以然”，方能运用自如，百战不殆。

为了做到这一点，我们将从 C 语言的核心语法开始，先来了解编译器是如何在机器指令层面实现它们的。紧接着，我们会把目光移到标准库。标准库是扩展 C 语言功能的一大利器，我们将会介绍现代 C 语言标准库中的一些重要功能，以及这些功能背后的运作机制。除此之外，如何利用计算机体系结构来编写高性能的 C 代码，也是工程化相关的重要内容。然后，随着 C 代码被编译，我们将会探讨二进制可执行程序是如何在与操作系统的协同工作下被运行的。经过这几个步骤后，你将会对一个 C 程序的完整生命周期有着更深刻的理解。



基于这个思路，这门课主要分为四个模块。

第一个模块是“**前置篇**”，我将为你讲解一些学习这门课所需要的基础知识。我们的课程中涉及到了有关计算机体系结构、汇编语言等较为底层的内容，因此我为你设计了一

讲“课前热身”，向你简单介绍与基本数据量单位、汇编语言，以及指令集寄存器有关的内容。在这一模块中，我还会用一段相对复杂的代码作为例子，来带你回顾 C 语言的核心语法，并介绍 C 程序从编写到运行的基本步骤。

第二个模块是“**C 核心语法实现篇**”。我会梳理 C 语言 7 大核心语法“背后的故事”，带你了解编译器如何在汇编层面实现这些语法。学完这个模块，你会对 C 程序的运行细节有着更深刻的理解，从而更好地掌握并优化程序运行。

第三个模块是“**C 工程实战篇**”。在这个模块中，你会学习到 C 语言在大型工程实战中的必备技巧，主要包括：快速掌握 C 标准库的重要功能，以及这些功能背后的实现原理；掌握编写高性能 C 代码、编码规范、结构化测试、结构化编译这些 C 项目工程化的实用技巧。

第四个模块是“**C 程序运行原理篇**”。我会为你介绍一个 C 程序是如何通过编译，并最终被操作系统运行的。程序的运行涉及到众多与操作系统的交互细节，你将在这个模块里详细了解它们。

这门课涉及的内容，我都是基于 x86-64 平台下的 Linux 系统进行介绍的。当然，对于 macOS 和 Windows 系统来说，某些细节会有所不同，但基本原理是相通的。

时至今日，C 语言作为最“古老”的编程语言之一，仍然“老当益壮”、生生不息。这一切靠的不是巧合，而是绝对的实力。**而要发挥 C 语言的最大威力，我们就不应该只简单了解它的语法，而应该在此基础上进一步了解代码如何被编译，程序如何被运行。**只有当完整的“链路”建立在脑海中时，你才对程序有了最完全的把控。那接下来，就跟我来一趟不一样的 C 语言之旅吧！

分享给好友，一起充电升级

👍 赞 18 💡 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。