手把手教你构建 C 语言编译器（0）- 前言

Table of Contents

“手把手教你构建 C 语言编译器” 这一系列教程将带你从头编写一个 C 语言的编译器。希望通过这个系列，我们能对编译器的构建有一定的了解，同时，我们也将构建出一个能用的 C 语言编译器，尽管有许多语法并不支持。

手把手教你构建 C 语言编译器系列共有10个部分：

1. 手把手教你构建 C 语言编译器（0）——前言
2. 手把手教你构建 C 语言编译器（1）——设计
3. 手把手教你构建 C 语言编译器（2）——虚拟机
4. 手把手教你构建 C 语言编译器（3）——词法分析器
5. 手把手教你构建 C 语言编译器（4）——递归下降
6. 手把手教你构建 C 语言编译器（5）——变量定义
7. 手把手教你构建 C 语言编译器（6）——函数定义
8. 手把手教你构建 C 语言编译器（7）——语句
9. 手把手教你构建 C 语言编译器（8）——表达式

10. 手把手教你构建 C 语言编译器（9）——总结

在开始进入正题之前，本篇是一些闲聊，谈谈这个系列的初衷。如果你急切地想进入正篇，请跳过本章。

为什么要学编译原理

如果要我说计算机专业最重要的三门课，我会说是《数据结构》、《算法》和《编译原理》。在我看来，能不能理解“递归”像是程序员的第一道门槛，而会不会写编译器则是第二道。

（当然，并不是说是没写过编译器就不是好程序员，只能说它是一个相当大的挑战吧）

以前人们会说，学习了编译原理，你就能写出更加高效的代码，但随着计算机性能的提升，代码是否高效显得就不那么重要了。那么为什么要学习编译原理呢？

原因只有一个：装B。

好吧，也许现在还想学习编译原理的人只可能是因为兴趣了。一方面想了解它的工作原理；另一方面希望挑战一下自己，看看自己能走多远。

理论很复杂，实现也很复杂？

我对编译器一直心存敬佩。所以当学校开《编译原理》的课程后，我是抱着满腔热情去上课的，但是两节课后我就放弃了。原因是太复杂了，听不懂。

一般编译原理的课程会说一些：

1. 如何表示语法（BNF什么的）
2. 词法分析，用什么有穷自动机和无穷自动机
3. 语法分析，递归下降法，什么 `LL(k)`，LALR 分析。
4. 中间代码的表示
5. 代码的生成
6. 代码优化

我相信绝大多数（98%）的学生顶多学到语法分析就结束了。并且最重要的是，学了这么多也没用！依旧帮助不了我们学习编译器！这其中最主要的原因是《编译原理》试图教会我们的是如何构造“编译器生成器”，即构造一个工具，根据文法来生成编译器（如 lex/yacc）等等。

这些理论试图教会我们如何用通用的方法来自动解决问题，它们有很强的实际意义，只是对于一般的学生或程序员来说，它们过于强大，内容过于复杂。如果你尝试阅读 lex/yacc （或 flex/bison）的代码，就会发现太可怕了。

然而如果你能跟我一样，真正来实现一个简单的编译器，那么你会发现，比起可怕的《编译原理》，这点复杂度还是不算什么的（因为好多理论根本用不上）。

项目的初衷

有一次在 Github 上看到了一个项目（当时很火的），名叫 c4，号称用 4 个函数来实现了一个小的 C 语言编译器。它最让我震惊的是能够自举，即能自己编译自己。并且它用很少的代码就完成了一个功能相当完善的 C 语言编译器。

一般的编译器相关的教程要么就十分简单（如实现四则运算），要么就是借助了自动生成的工具（如 flex/bison）。而 c4 的代码完全是手工实现的，不用外部工具。可惜的是它的代码初衷是代码最小化，所以写得很乱，很难懂。所以本项目的主要目的：

1. 实现一个功能完善的 C 语言编译器
2. 通过教程来说明这个过程。

c4 大致500+行。重写的代码历时一周，总共代码加注释1400行。项目地址: Write a C Interpreter。

声明：本项目中的代码逻辑绝大多数取自 c4，但确为自己重写。

做好心理准备

在写编译器的时候会遇到两个主要问题：

1. 繁琐，会有许多相似的代码，写起来很无聊。
2. 难以调试，一方面没有很好的测试用例，另一方面需要对照生成的代码来调试（遇到的时候就知道了）。

所以我希望你有足够的耐心和时间来学习，相信当你真正完成的时候会像我一样，十分有成就感。

PS. 第一篇完全没有正题相关的内容也是希望你能有所心理准备再开始学习。

参考资料

最后想介绍几个资料：

1. Let's Build a Compiler 很好的初学者教程，英文的。
2. Lemon Parser Generator，一个语法分析器生成器，对照《编译原理》观看效果更佳。

由于本人水平一般，文章、代码难免会有错误，敬请批评指正！

最后祝你学得愉快。