手把手教你构建 C 语言编译器（1）- 设计

Table of Contents

这是“手把手教你构建 C 语言编译器”系列的第二篇，我们要从整体上讲解如何设计我们的 C 语言编译器。

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

首先要说明的是，虽然标题是编译器，但实际上我们构建的是 C 语言的解释器，这意味着我们可以像运行脚本一样去运行 C 语言的源代码文件。这么做的理由有两点：

1. 解释器与编译器仅在代码生成阶段有区别，而其它方面如词法分析、语法分析是一样的。
2. 解释器需要我们实现自己的虚拟机与指令集，而这部分能帮助我们了解计算机的工作原理。

编译器的构建流程

一般而言，编译器的编写分为 3 个步骤：

1. 词法分析器，用于将字符串转化成内部的表示结构。
2. 语法分析器，将词法分析得到的标记流（token）生成一棵语法树。
3. 目标代码的生成，将语法树转化成目标代码。

已经有许多工具能帮助我们处理阶段1和2，如 flex 用于词法分析，bison 用于语法分析。只是它们的功能都过于强大，屏蔽了许多实现上的细节，对于学习构建编译器帮助不大。所以我们要完全手写这些功能。

所以我们会依照以下步骤来构建我们的编译器：

1. 构建我们自己的虚拟机以及指令集。这后生成的目标代码便是我们的指令集。

2. 构建我们的词法分析器
3. 构建语法分析器

编译器框架

我们的编译器主要包括 4 个函数：

1. `next()` 用于词法分析，获取下一个标记，它将自动忽略空白字符。
2. `program()` 语法分析的入口，分析整个 C 语言程序。
3. `expression(level)` 用于解析一个表达式。
4. `eval()` 虚拟机的入口，用于解释目标代码。

这里有一个单独用于解析“表达式”的函数 `expression` 是因为表达式在语法分析中相对独立并且比较复杂，所以我们将它单独作为一个模块（函数）。下面是相应的源代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <string.h>

int token;            // current token
char *src, *old_src;  // pointer to source code string;
int poolsize;         // default size of text/data/stack
int line;             // line number

void next() {
    token = *src++;
    return;
}
```

```
void expression(int level) {
    // do nothing
}

void program() {
    next();                  // get next token
    while (token > 0) {
        printf("token is: %c\n", token);
        next();
    }
}


int eval() { // do nothing yet
    return 0;
}

int main(int argc, char **argv)
{
    int i, fd;

    argc--;
    argv++;

    poolsize = 256 * 1024; // arbitrary size
    line = 1;

    if ((fd = open(*argv, 0)) < 0) {
        printf("could not open(%s)\n", *argv);
        return -1;
    }

    if (!(src = old_src = malloc(poolsize))) {
        printf("could not malloc(%d) for source area\n", poolsize);
        return -1;
    }

    // read the source file
    if ((i = read(fd, src, poolsize-1)) <= 0) {
        printf("read() returned %d\n", i);
        return -1;
```

```
    }
    src[i] = 0; // add EOF character
    close(fd);

    program();
    return eval();
}
```

上面的代码看上去挺复杂，但其实内容不多。它的流程为：读取一个文件（内容为 C 语言代码），逐个读取文件中的字符，并输出。这里需要的是注意每个函数的作用，后面的文章中，我们将逐个填充每个函数的功能，最终构建起我们的编译器。

本节的代码可以在 Github 上下载，也可以直接 clone

```
git clone -b step-0 https://github.com/lotabout/write-a-C-interpreter
```

这样我们就有了一个最简单的编译器：什么都不干的编译器，下一章中，我们将实现其中的`eval`函数，即我们自己的虚拟机。