手把手教你构建 C 语言编译器（7）- 语句

Table of Contents

整个编译器还剩下最后两个部分：语句和表达式的解析。它们的内容比较多，主要涉及如何将语句和表达式编译成汇编代码。这章讲解语句的解析，相对于表达式来说它还是较为容易的。

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

语句

C 语言区分“语句”（statement）和“表达式”（expression）两个概念。简单地说，可以认为语句就是表达式加上末尾的分号。

在我们的编译器中共识别 6 种语句：

1. `if (...) <statement> [else <statement>]`
2. `while (...) <statement>`
3. `{ <statement> }`
4. `return xxx;`
5. `<empty statement>`;
6. `expression;` (expression end with semicolon)

它们的语法分析都相对容易，重要的是去理解如何将这些语句编译成汇编代码，下面我们逐一解释。

IF 语句

IF 语句的作用是跳转，跟据条件表达式决定跳转的位置。我们看看下面的伪代码：

```
if (...) <statement> [else <statement>]

  if (<cond>)                    <cond>
                                 JZ a
    <true_statement>   ===>      <true_statement>
  else:                          JMP b
a:                           a:
```

```
    <false_statement>          <false_statement>
b:                         b:
```

对应的汇编代码流程为：

1. 执行条件表达式 `<cond>`。
2. 如果条件失败，则跳转到 `a` 的位置，执行 `else` 语句。这里 `else` 语句是可以省略的，此时 `a` 和 `b` 都指向 IF 语句后方的代码。
3. 因为汇编代码是顺序排列的，所以如果执行了 `true_statement`，为了防止因为顺序排列而执行了 `false_statement`，所以需要无条件跳转 `JMP b`。

对应的 C 代码如下：

```
if (token == If) {
    match(If);
    match('(');
    expression(Assign);  // parse condition
    match(')');

    *++text = JZ;
    b = ++text;

    statement();         // parse statement
    if (token == Else) { // parse else
        match(Else);

        // emit code for JMP B
        *b = (int)(text + 3);
        *++text = JMP;
        b = ++text;
```

```
        statement();
    }

    *b = (int)(text + 1);
}
```

While 语句

While 语句比 If 语句简单，它对应的汇编代码如下：

```
a:                      a:
   while (<cond>)         <cond>
                          JZ b
    <statement>           <statement>
                          JMP a
b:                      b:
```

没有什么值得说明的内容，它的 C 代码如下：

```
else if (token == While) {
    match(While);

    a = text + 1;

    match('(');
    expression(Assign);
    match(')');

    *++text = JZ;
    b = ++text;

    statement();

    *++text = JMP;
    *++text = (int)a;
```

```
    *b = (int)(text + 1);
}
```

Return 语句

Return 唯一特殊的地方是：一旦遇到了 Return 语句，则意味着函数要退出了，所以需要生成汇编代码 `LEV` 来表示退出。

```
else if (token == Return) {
    // return [expression];
    match(Return);

    if (token != ';') {
        expression(Assign);
    }

    match(';');

    // emit code for return
    *++text = LEV;
}
```

其它语句

其它语句并不直接生成汇编代码，所以不多做说明，代码如下：

```
else if (token == '{') {
    // { <statement> ... }
    match('{');

    while (token != '}') {
        statement();
    }
```

```
        match('}');
    }
    else if (token == ';') {
        // empty statement
        match(';');
    }
    else {
        // a = b; or function_call();
        expression(Assign);
        match(';');
    }
```

代码

本章的代码可以在 Github 上下载，也可以直接 clone

```
git clone -b step-5 https://github.com/lotabout/write-a-C-interpreter
```

本章的代码依旧无法运行，还剩最后一部分没有完成：`expression`。

小结

本章讲解了如何将语句编译成汇编代码，内容相对容易一些，关键就是去理解汇编代码的执行原理。

同时值得一提的是，编译器的语法分析部分其实是很简单的，而真正的难点是如何在语法分析时收集足够多的信息，最终把源代

码转换成目标代码（汇编）。我认为这也是初学者实现编译器的一大难点，往往比词法分析/语法分析更困难。

所以建议如果没有学过汇编，可以学习学习，它本身不难，但对理解计算机的原理有很大帮助。