1.7 TINY样本语言与编译器

任何一本关于编译结构的书如果不包括编译过程步骤的示例就不能算完整。本书将会多次用从现有的语言（如C、C++、Pascal和Ada）中抽取的实例来讲解。但是仅用这些实例来描述编译器的各个部分是如何协调一致的却不够。因此，写出一个完整的编译器并对其操作进行注释仍是很必要的。

描述真实的编译器非常困难。“真正的”编译器——也就是希望在每天编程中用到的——内容太复杂而且不易在本教材中掌握。另一方面，一种很小的语言（其列表包括10页左右的文本）的编译也不可能准确地描述出“真正的”编译器所需的所有特征。

为了解决上述问题，人们在（ANSI）C中为小型语言提供了完整的源代码，一旦能明白这

种技术，就能够很容易地理解这种小型语言的编译器了。这种语言称作 TINY，在每一章的示例中都会用到它，它的编译代码也很快会被提到。完整的编译代码汇集在附录 B中。

还有一个问题是：选择哪一种机器语言作为 TINY编译器的目标语言？为现有的处理器使用真正的机器代码的复杂性使得这个选择很困难。但是选择特定的处理器也将影响到执行这些机器生成的目标代码。相反地，可将目标代码简化为一个假定的简单处理器的汇编语言，这个处理器称为TM机（tiny machine）。在这里只是简单地谈谈，详细内容将放在第8章（代码生成）中。附录C有C的TM模拟程序列表。

1.7.1 TINY 语言

TINY的程序结构很简单，它在语法上与 Ada或Pascal的语法相似：仅是一个由分号分隔开的语句序列。另外，它既无过程也无声明。所有的变量都是整型变量，通过对其赋值可较轻易地声明变量（类似FORTRAN或BASIC）。它只有两个控制语句：if语句和repeat语句，这两个控制语句本身也可包含语句序列。If语句有一个可选的else部分且必须由关键字end结束。除此之外，read语句和write语句完成输入/输出。在花括号中可以有注释，但注释不能嵌套。

TINY的表达式也局限于布尔表达式和整型算术表达式。布尔表达式由对两个算术表达式的比较组成，该比较使用 <与=比较算符。算术表达式可以包括整型常数、变量、参数以及 4个整型算符 +、—、*、/，此外还有一般的数学属性。布尔表达式可能只作为测试出现在控制语句中——而没有布尔型变量、赋值或I/O。

程序清单1-1是该语言中的一个阶乘函数的简单编程示例。这个例子在整本书中都会用到。

程序清单1-1 一个输出其输入阶乘的TINY语言程序

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if x > 0 then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { output factorial of x }
end
```

虽然TINY缺少真正程序设计语言所需要的许多特征——过程、数组且浮点值是一些较大的省略——但它足可以用来例证编译器的主要特征了。

1.7.2 TINY编译器

TINY编译器包括以下的C文件，（为了包含而）把它的头文件放在左边，它的代码文件放在右边：

```
globals.h          main.c
util.h             util.c
scan.h             scan.c
parse.h            parse.c
```

```
symtab.h        symtab.c
analyze.h       analyze.c
code.h          code.c
cgen.h          cgen.c
```

除了将**main.c** 放在**globals.h** 的前面之外，这些文件的源代码及其行号都按顺序列在附录B中了。任何代码文件都包含了**globals.h** 头文件，它包括了数据类型的定义和整个编译器均使用的全程变量。**main.c** 文件包括运行编译器的主程序，它还分配和初始化全程变量。其他的文件则包含了头/代码文件对、在头文件中给出了外部可用的函数原型以及在相关代码文件中的实现（包括静态局部函数）。**scan**、**parse**、**analyze** 和**cgen** 文件与图1-1中的扫描程序、分析程序、语义分析程序和代码生成器各阶段完全相符。 **util** 文件包括了实用程序函数，生成源代码（语法树）的内部表示和显示列表与出错信息均需要这些函数。 **symtab** 文件包括执行与TINY应用相符的符号表的杂凑表。**code**文件包括用于依赖目标机器（将在1.7.3节描述的TM机）的代码生成的实用程序。图1-1还缺少一些其他部分：没有单独的错误处理器或文字表且没有优化阶段；没有从语法树上分隔出来的中间代码；另外，符号表只与语义分析程序和代码生成器交互（这将在第6章中再次讨论到）。

虽然这些文件中的交互少了，但是编译器仍有4遍：第1遍由构造语法树的扫描程序和分析程序组成；第2遍和第3遍执行语义分析，其中第2遍构造符号表而第3遍完成类型检查；最后一遍是代码生成器。在**main.c**中驱动这些遍的代码十分简单。当忽略了标记和编辑时，它的中心代码如下（请参看附录B中的第69、77、79和94行）：

```
syntaxTree = parse( );
buildSymtab (syntaxTree);
typeCheck (syntaxTree);
codeGen (syntaxTree, codefile);
```

为了灵活起见，我们还编写了条件编译标志，以使得有可能创建出一部分的编译器。如下是该标志及其效果：

标　志	设置效果	编译所需文件（附加）
NO_PARSE	创建只扫描的编译器	**globals.h, main.c, util.h, util.c, scan.h, scan.c**
NO_ANALYZE	创建只分析和扫描的编译器	**parse.h, parse.c**
NO_CODE	创建执行语义分析，但不生成代码的编译器	**symtab.h, symtab.c, analyze.h,analyze.c**

尽管这个TINY编译器设计得有些不太实际，但却有单个文件与阶段基本一致的好处，在以后的章节中将会一个一个地学到这些文件。

任何一个ANSI C编译器都可编译TINY编译器。假定可执行文件是**tiny**，通过使用以下命令：

```
tiny sample. tny
```

就可用它编译文本文件**sample.tny** 中的TINY源程序。（如果省略了**.tny**，则编译器会自己添加**.tny**后缀）。屏幕上将会出现一个程序列表（它可被重定向到一个文件之上）并且（如当代码生成是被激活的）生成目标代码文件**sample.tm**（在下面将谈到的TM机中使用）。

在编辑列表的信息中有若干选项，以下的标志均可用：

标 志	设 置 效 果
EchoSource	将TINY源程序回显到带有行号的列表
TraceScan	当扫描程序识别出记号时，就显示每个记号的信息
TraceParse	将语法树以线性化格式显示
TraceAnalyze	显示符号表和类型检查的小结信息
TraceCode	打印有关代码文件的代码生成跟踪注释

1.7.3 TM机

我们用该机器的汇编语言作为TINY编译器的目标语言。TM机的指令仅够作为诸如TINY这样的小型语言的目标。实际上，TM具有精减指令集计算机（RISC）的一些特性。在RISC中，所有的算法和测试均须在寄存器中进行，而且地址模式极为有限。为了使读者了解到该机的简便之处，我们将下面C表达式的代码

```
a [index] = 6
```

翻译成TM汇编语言（请读者将它与1.3节中相同语句假定的汇编语言比较一下）：

```
LDC 1, 0 ( 0 )  load o into reg 1
* 下面指令
* 假设index 在存储器地址10中
LD 0, 10 ( 1 )  load val at (10+R1 into R0
LDC 1, 2 ( 0 )  load 2 into reg 1
MUL 0, 1, 0     put R1 * R0 into R0
LDC 1, 0 ( 0 )  load 0 into reg 1
* 下面指令
* 假设a在存储器地址20中
LDA 1, 20 ( 1 ) load 20 + R1 into R0
ADD 0, 1, 0     put R1 + R0 into R0
LDC 1, 6 ( 0 )  load 6 into reg 1
ST 1, 0 ( 0 )   store R1 at 0 + R0
```

我们注意到装入操作中有3个地址模式并且是由不同的指令给出的：**LDC**是“装入常量”，**LD**是“由存储器装入”，而**LDA**是“装入地址”。另外，该地址通常必须给成“寄存器+偏差”值。例如“**10(1)**”（上面代码的第2条指令），它代表在将偏差10加到寄存器1的内容中计算该地址。（因为在前面的指令中，0已被装入到寄存器1中，这实际是指绝对位置10）⊖。我们还看到算术指令**MUL**和**ADD**可以是“三元”指令且只有寄存器操作数，其中可以单独确定结果的目标寄存器（1.3节中的代码与此相反，其操作是“二元的”）。

TM机的模拟程序直接从一个文件中读取汇编代码并执行它，因此应避免将由汇编语言翻译为机器代码的过程复杂化。但是，这个模拟程序并非是一个真正的汇编程序，它没有符号地址或标号。因此，TINY编译器必须仍然计算跳转的绝对地址。此外为了避免与外部的输入/输出例程连接的复杂性，TM机有内部整型的I/O设备；在模拟时，它们都对标准设备读写。

⊖ LDC命令也要求一个“寄存器+偏差”的格式；但由于TM汇编程序格式简单统一，也就忽略了寄存器，偏差本身也作为常量装入。

通过使用任何一个ANSI C编译器，都可将**tm.c** 源代码编译成TM模拟程序。假定它的可执行文件叫作**tm**，通过发出命令

```
tm sample.tm
```

就可使用它了。其中，sample. tm是TIMY编译器由sample. tny源文件生成的代码文件。该命令引起代码文件的汇编和装入，接着就可交互地运行 TM模拟程序了。例如：如果 sample. tny是程序清单1-1中的范例程序，则按以下计算就可得到7的阶乘：

```
tm sample. tm
TM simulation (enter h for help) ...
Enter command: go
Enter value for IN instruction: 7
OUT instruction prints: 5040
HALT: 0, 0, 0
Halted
Enter command: quit
Simulation done.
```