编 译 技 术

F. R. A. 赫普古德 著

朱三元 曹东启 姚兆炜 译

内 容 简 介

目前电子计算机已成为科学、技术、经济、军事各个领域中广泛使用的强有力的计算工具．为了更有效、更方便地使用电子计算机，必须配置各种程序系统，其中最重要的系统之一就是编译系统．本书较全面地介绍了建立这种系统的常用方法和基本技术，内容简明、易读，是一本较好的入门书，可供设计、使用计算机的工作者以及有关专业人员参考学习之用．

F. R. A. Hopgood
COMPILING TECHNIQUES
Macdonald, American Elsevier Inc,
1970

编 译 技 术

F. R. A. 赫普古德 著
朱三元 曹东启 姚兆炜 译
*
科学出版社出版
北京朝阳门内大街 137 号
中国科学院印刷厂印刷
新华书店北京发行所发行 各地新华书店经售
*
1975 年 9 月第 一 版 开本：787×1092 1/32
1975 年 9 月第一次印刷 印张：5 1/16
印数：0001—26,450 字数：112,000
统一书号：13031·309
本社书号：479·13—1
定价：0.42 元

目　　录

第一章 绪 言

1.1 编译程序的定义

编译程序是计算机程序，它把面向问题语言（如 ALGOL，FORTRAN）的程序作为数据接收，并产生面向计算机的代码作为输出，这种代码还可能经汇编程序或装配程序作进一步的加工，交给计算机执行，其结果等价于面向问题语言的程序得出的结果．这个定义有点含糊，但也只能如此，因为有各种特点的计算机程序都采用编译程序这个词．

本书假定要编译的面向问题语言与机器无关，其复杂程度如 FORTRAN 或 ALGOL 语言那样．书中好多例子就是取材于这类语言的．虽然所讲的一些方法偏于科学语言，但大多数方法仍可应用于一般．编译程序的输出是以某假想计算机的机器指令定义的，这种计算机类似于目前普遍采用的单累加器的计算机．

1.2 编译程序的结构

一个编译程序可分为若干段，各段完成特定的工作．每一段都有特定的输入数据和输出结果，从而均可看作是一个子程序．在一些小型计算机上，为适应主存容量，程序的大小受到严格限制．通常把编译程序分为若干子程序，对整个程序来说，每个子程序在调用下一个子程序之前完成它自己的工作．若整个编译程序能充裕地放入主存贮器，那末，通常是轮流进入各个子程序，使得接近于编完一部分输入程序以

后，再审查输入程序的下一部分．虽然这两种方法要求完全不同的控制结构，但所采用的技巧并无很大差别．

通常使用一些附加设备，把面向问题语言的程序穿在纸带或卡片上，通过阅读器输入计算机．或者把部分程序保留在计算机内预先指定的单元中．在大多数系统里，有好几种提供程序的方法，因此，编译程序通常有一个开始段，其目的是为外界和编译程序其余部分之间提供一种交接形式．一旦经过开始段，对编译程序其余部分来说，用各种不同方法提供给它的同一程序就是等同的了．大部分程序设计语言在程序准备时允许加入某些多余符号．例如，时常插入空符号以助阅读，按需要插入若干新行符号．这些多余符号也被开始段删除，这种开始段通常称为词法分析．

完成词法分析后，就需要辨认组成程序的各个语句．若语句是复杂的，则把它分解成若干较简单的部分．一般说来，所定义的程序设计语言应使语法分析毋需知道语句的含义就能进行下去．编译程序的这一部分将证实程序在文法上是正确的，而其输出表示这种分析所建立的程序结构．

随后必须考查程序的每个语句，并产生出等价的计算机代码．对大多数语句来说，这种代码生成是简单的．某些语句类型常常有固定的生成代码序列，只有少数几个字段需要根据相应语句在程序中的确切形式而有所改变．这种固定的代码序列，已被存贮起来备用，称之为框架．

如果重点是要使占用的存贮尽可能地少，或使程序运行尽可能地快，那末，就要对上述所生成的简单代码进行优化．这样一来，需要对各语句进行更多的分析，并要查明前后语句间的相互影响．对科学语言来说，把表达式代码生成作得充分有效是至关重要的，因为无论在哪个程序中，它在全部代码里都占很大比例．

1.3 简　史

第一批高级语言的编译程序是在 1956 年到 1958 年产生的. 其中，最重要的是 IBM 704 机的 FORTRAN 编译程序. 这个早期的编译程序总共花了 18 个人年编成的，在某些方面，它所使用的技巧和目前所采用的一样灵活. 因为唯一的输入手段是穿孔卡片，并因为 FORTRAN 语言的语句辨认简单，所以，未对编译程序的词法分析提供特殊的技术. 但是，IBM 704 机只有三个变址寄存器，而设计目的之一是要产生和手编程序效能一样的代码，因此，大量的精力放在变址寄存器的分配算法和算术表达式的代码优化上面. 介绍这项工作的有 Sheridan[70] [1] 和 Backus[7] 的两篇文章.

关于算术表达式的语法分析和语义分析的大量不同算法，是在 1960 年前后出现的. 其中包括很多优先类型的方法，同时还包括一些不太常用的算法. 已定义的一些算法中，有些是从右到左分析的，有些是在扫描语句时不断改变方向的. 这些早期方法的历史请见 Randell 和 Russell[61].

1960 年的 ALGOL 报告是第一个被广泛接受的语言定义，它有严格定义的语法. 该报告连同定义中所使用的 BNF[2] 表示法，促进了语法分析及其有关问题的研究. 这些工作延续至今. 语法分析部分能使我们很好地理解所用的技巧，并对各种技巧的优缺点给予恰当的评价.

由于 ALGOL 语言是作为没有给出任何机器表示的国际标准接受的，这就导致了近几年来把更多的精力放在词法分析上. 例如，在英国已有十几种不同的 ALGOL 语言外部表示

---

1) 见参考资料.

2) BNF 即 Backus Normal Form 的缩写，称为 "Backus 范式". 可参阅 Naur, *CACM*, 1960, 5, 299—314.

法，并在这方面作了不少工作，使之有助于程序的互换.

编译技术方面的许多早期文章，已由 Rosen[63] 收集在一起了.

第二章　数据结构

2.1 引　言

在介绍编译程序的各部分之前，有必要了解编译程序所用的抽象数据结构，还应了解如何用具体的存贮器来实现它．我们可以把编译过程设想成：取进某种输入数据结构并改造它，以产生输出数据结构．在某种意义上讲，这种结构既等价于输入形式，又比输入形式更合于我们的意图．输入数据通常称为源程序，输出数据则称为结果代码．编译程序简单的形式之一是：从外部设备（例如卡片阅读器）读进输入数据结构，随后编译程序产生机器代码程序作为输出（放在计算机的存贮器内以备执行）．好多编译程序在得到最后的输出形式之前，输入数据结构往往要被改造成为几种内部数据结构．编译程序本身可能还要求存放一些附加信息，或为内部存放的信息交替地驱动而运转．最后，在用特定程序设计语言编写的源程序中所定义的数据结构，必须把它表示成某种存贮结构的形式（以执行该程序的计算机为背景）．在设计一个编译程序时，需要解决的问题有：诸如找出每一阶段所用的最明显的抽象数据结构，把抽象数据结构表示为最有效的内部存贮结构等．

2.2 抽象数据结构

我们把数据结构定义为一组规则和约束条件，它们表示数据块之间存在的关系．这种结构并未涉及可能出现的各个

数据块是什么．在某种意义上讲，只要求它们保持这种结构，然而，包含在数据项内的任何信息是不依赖于该结构的．这里使用了项这个术语去表示一个抽象数据结构的块．项本身也可以是其他的数据结构，这样就建立了数据结构的层次集合．

2.2.1 串（*string*）

串是项的有序集．它可以是变长的，也可以是定长的．每项只有其相邻项的信息，因此，取一特定项时，必须从串的一端开始，顺序进行查找．在串上定义的运算有

(a) 两串的并置；

(b) 对两串作项对项的比较；

(c) 把串分成几部分．

2.2.2 数组

数组 A 是这样排列的项的集合：使得一个有序整数集唯一定义数组每项的位置，并提供直接取每项的方法．用 ALGOL 表示法，数组 A 的一个项可写为 $A[j_1, j_2, j_3, \cdots, j_n]$．如果有序整数集的长度为 n，那末，该数组称为 n 维的．有序集中各个整数称为下标．每个下标可以用任意方式定义其界限，而该界限可由若干互不相交的子界限组成．

在编译程序和程序设计语言中所使用的是数组集的某一特殊子集，即所谓矩形数组集．矩形数组的每个下标界限是不变的和连读的．例如，ALGOL 说明

array $A[1{:}10, 0{:}5, 3{:}7]$

定义了一个三维数组．它的下标界限分别是 10，6，5，其典型元为 $A[3, 4, 5]$．

2.2.3 排队和栈（*Queue* 和 *Stack*）

排队和栈是动态改变的数据结构．任何时候它们都包含一组有序的项．对排队的情形来说，如果添加一项，就把它放在组的末端，而且只能从定义该排队组的前面取出或移掉项．

列在排队中的第一项是唯一可取出的项，而且必须先移掉它．对栈的情形来说，添加的项同样是放在组的末端，不同的是要从组的末端取出和移掉项．此时最后添加的项是唯一可取出的项，并最先被移掉．项可以是可变长度的，并在项中指明其长度．

2.2.4 表

表由一组项组成，和每项相联系的是唯一的名字或关键字．通常，每项是由它的关键字连同与该关键字有关的一些信息所组成．给出项的关键字及其有关信息就能把项添到表中．反之，通过关键字能从表中取出相应的项来．

2.2.5 树

树是由一组结点组成的结构．每个结点(或者项)有这样的特点：除它所带的信息外，它还包含低层结点的指示字，在树的最低层，结点指的是叶子，叶子是由树外的数据结构组成的．层的思想是基于下述事实得来的：每株树必有一最顶的结点，它再无别的结点指示它(通常称为树的根)，树中也没有结点能指示前面已定义的结点．后一条件保证了从根到每一个结点只有唯一的通路．第一层结点是树根所指的那些结点，第二层的结点是第一层结点所指示的那些结点，等等．

图 2.1 用图解方法表示 $A*B+C*D$ 的二叉树（每个结点都正好指示两个低一层的结点）．字母 A，B，C，D 表示

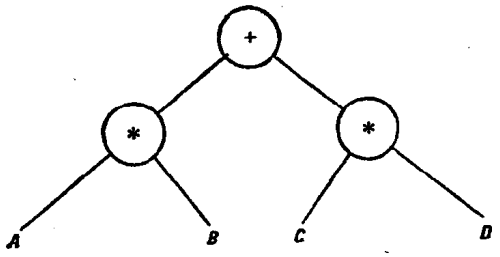

图 2.1

叶子，圆圈内的符号表示与该结点有关的信息．通常采用树根在顶部、叶子在底部的方法去表示树，这和自然界中的树有所不同．＊号用来表示乘法．

2.2.6 方向图

除低层结点可指示高层结点外，方向图是一种类似于树的结构．所以，一个结点可有几个结点指示它．因为现在已看不清图中两个结点之间的联线方向，所以，通常在图中至少要标出向上的指示线．放宽对树的一些规则的限制，就表明有可能存在一条从某结点出发再回到该结点的通路，即通常所谓的环路．方向图有一个特殊的子类是无环方向图．这和树的定义并不一致，因为，它仍然允许两个结点指示同一个结点，而这一点在树的定义中是不允许的．例如，ALGOL 语言的复合语句：

```
        begin
          a: = b: = c: = 2;
first:  if a > b then z: = 4 else c: = 1;
          if z > 3 then go to next;
          b: = 1;
          go to first;
next:   z: = 1;
        end
```

这个语句可用图 2.2 的方向图表示(图见 9 页)．

2.3 内部存贮结构

计算机都具备一组有序字组成的存贮器，这也是计算机本身唯一可用的存贮结构．通常，利用程序和子程序，可在计算机的基本结构上组成更高级的存贮结构，对使用子程序的人来说，好象计算机本身就有这种结构．有很多定义得相当

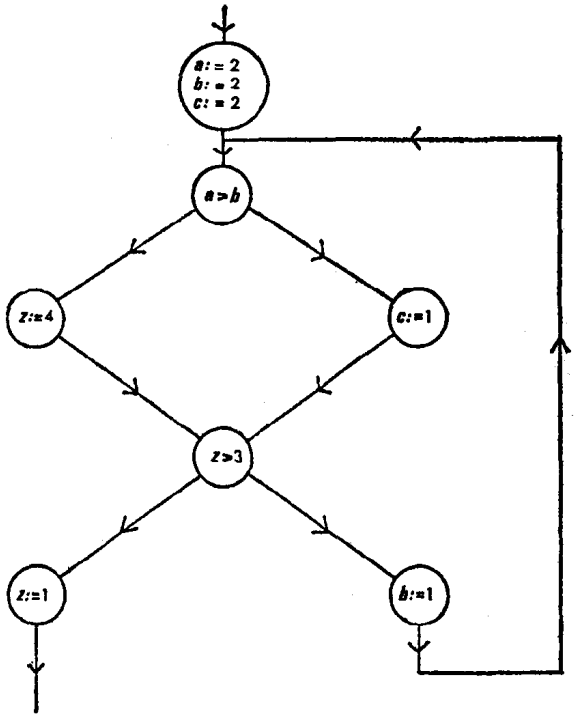

图 2.2

好的内部存贮结构的集合已被人们采用. 虽然，通常把它们设计成等价于抽象的存贮结构，或者，至少能容易映象为抽象结构，但是，它们仍受计算机存贮的基本结构的影响.

每个内部存贮结构将由不可分的(就结构来说)存贮块组成，这些存贮块称为元. 这些元严格地等价于抽象结构的项. 每个元可有几个域，其中某些能用于定义结构. 就宽度而言，一个元可由机器字的一个二进位组成，但更常见的至少要有一个地址码的长度.

2.3.1 向量

在好多计算机中，向量是为计算机的基本存贮结构起的

名字．向量是计算机存贮器中一组相毗邻的元．向量是由它的基始地址、元的大小及其长度定义的(假定所有元具有同样的大小)．由向量的基始地址，根据元在向量中的位置，可直接取到该元．和向量密切对应的抽象数据结构是一维数组，而且几乎一定能把一维数组映象为向量．因此，用ALGOL的说明：

array $A[1:N]$

表示长度为 N、名字为 A 的向量是方便的，并把 $A[i]$ 看做是向量的元 ($i = 1, \cdots, N$)．

2.3.2 链表

链表是一组元，每个元由两个域组成．第二个域存放一个指向链表中下一个元的指示字．第一个域存放的是指出该元所定义的信息的指示字．信息可能是别的链表或某些外部数据结构．外部数据结构称为原子，我们假定它对链表的运算而言是不可分的．例如，字符号串 ABC 能用图 2.3 表示成一个链表．

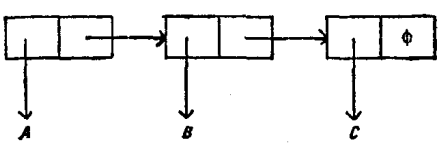

图 2.3

链表作为一个整体可用指示字 P 取用，P 指示链表的第一个元．链表末一个元的第二个域中，或者是一个写为 ϕ 的特殊登记符，表示空指示字；或者是一个指示链表开始元的指示字，这时，链表称为环．

具体实现时，链表元是由足以给出两个计算机地址的二进位所组成．某些计算机用一个机器字就能做到，有的就要

求两个机器字才行．值得注意的是：相邻元在计算机的存贮器中勿须连续，事实上，它们可以分散于整个计算机的存贮器中．有关链表运算的详细介绍，见 J. M. Foster 的“表格加工（*List Processing*）”一文．

链表是能灵活使用计算机连续存贮器的最简单的形式．

2.3.3 丛（*plex*）

虽然上面定义的链表可以表示更加复杂的抽象数据结构，诸如树和方向图．但这种表示时常是效率不高而又含糊．例如，图 2.1 中的树，可用图 2.4 中所示的结构表示成一个链表．

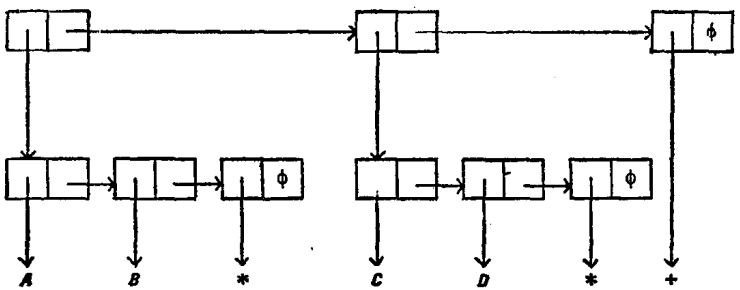

图 2.4

在这种情况下，一个更有效而又明显的存贮结构是丛．类似丛的存贮系统已经广泛使用了若干年，丛的精确定义和理论，以及丛的程序设计已由麻省理工学院的 D. T. Ross 作了总结．丛（拉丁文为 *plexus*，意即任何复杂结构，它包含由相关部分所组成的交错盘结的网络）是由一组称为珠（*bead*）的元组成的，此处每个元是计算机存贮器的 *N*-字向量．这 *N*-字块分成一组含有信息或指示其它珠的指示字的域．我们假定丛的每种类型是一种格式，这种格式规定各个域代表的是什么．又假定格式在外形上看是定义丛的珠所组成的集合．