电子计算机介绍

数字计算机的编译程序构造

中册

D. 格里斯

科学出版社

数字计算机的编译程序构造

（中册）

D. 格里斯
曹东启 仲萃豪 姚兆煒 译

内 容 提 要

本书较全面系统地介绍了编译程序构造的主要技术、实用方法以及所需的基本理论知识.

全书共分二十一章. 前七章讲述编译程序的有关概念、形式语言理论、扫描程序以及各种语法分析方法. 随后九章介绍运行时存贮的组织、各种符号表、源程序的内部形式、语义程序、变量的存贮分配、错误校正以及解释程序等内容. 最后五章进一步讨论了代码生成、代码优化、宏功能的实现以及编写翻译程序的系统,并对建立编译程序时所应考虑的问题给出了若干扼要提示.

本书可供计算机软件工作者及有关专业人员学习参考之用.

David Gries

COMPILER CONSTRUCTION FOR DIGITAL COMPUTERS

John Wiley & Sons, Inc.

数字计算机的编译程序构造

(中册)

D. 格 里 斯

曹东启 仲萃豪 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

*

1976年7月第一版 开本:787×1092 1/32

1976年7月第一次印刷 印张:6 1/2

印数:0001—40,450 字数:147,000

统一书号:13031·445

本社书号:668·13—1

定价: 0.52 元

目　录

第八章　运行时的存贮组织

本章以三种熟知的高级语言为例，说明运行时的存贮组织问题．同时我们将对程序设计语言中所使用的许多概念进行解释．我们要给出类似于 FORTRAN 语言的一种简单存贮组织，类似于 ALGOL 语言的一种较复杂的存贮组织，以及带有结构、指示字变量和 ALLOCATE 与 FREE 语句的类似于 PL/1 语言必须建立的一种最复杂的存贮组织．

诚然，由于篇幅所限，我们不可能介绍这些语言的全部细节．所以本章的目的只介绍存贮管理的基本内容和某些一般观点．读者在学懂了这里所介绍的内容以后，应该能够对任何一种语言设计出自己所需的运行时存贮管理程序．通过阅读本章，应该弄清楚存贮管理的基本问题．在读本章时，应该对源语言彻底了解，否则会在某些地方由于看不懂而被忽视．

某些语言要求动态存贮分配方案，即按照随机方式分配若干块主存，然后释放它，再后又为其他目的而重新分配．有关这些方案的全面讨论，已超出本书范围．但为了圆满起见，在 8.10 中我们介绍一种存贮分配方法，并希望读者学一下 Knuth（68）所写的那些技术．现在，我们假定有两个程序 GETAREA（ADDRESS, SIZE）和 FREEAREA（ADDRESS, SIZE）．

第一个程序给调用的程序分配 SIZE 个单元，所分配单元的起始地址放在输出参数 ADDRESS 中．FREEAREA 是释放从地址 ADDRESS 开始的 SIZE 个单元．

在某些系统中，它虽分配存贮，但又不明显地释放存贮．当没有可用的存贮时，系统就调用**无用单元收集**程序，它找出现在确实已经不用的那些单元，并释放它们，它还压紧所用空间（即把正在用的数据移到相连单元），以便腾出大块相连单元成为空白存贮区，而不让它成为许多彼此分散的小块．在 8.10 中简短地讨论了这个问题；我们假定有一个程序 FREEGARBAGE 实现这个功能．

本章由下述几部分构成．在 8.1 中试图统一介绍**数据区**的概念．8.2 讨论属性单元，而 8.3 到 8.6 按照复杂程度，专门讨论了实现数据结构时所引起的各种问题．8.7 讨论了过程调用以及形式参数与实在参数之间的替换关系．然后再回到与 FORTRAN、ALGOL 有关的问题．在 8.10 中，讨论了动态存贮分配和无用单元收集，并以此结束本章．

8.1 数据区和区头向量

数据区是一组相连单元（一段磁心存贮），我们暂且不管数据究竟是按什么方法把它们逻辑地归并在一起的．一个数据区中所有单元在源程序中常常（但并不总是这样）有相同的**定义域**；它们能由同样一组语句所引用（换言之，定义域可以是一个分程序或过程体）．对类似的概念，Wegner（68）曾用过术语**现役记录**，但我们仍旧采用数据区这个简短的术语．

FORTRAN 公用语句中说明的各个变量将分配在单独的数据区中，即空白公用数据区．在 ALGOL 中，主程序所说明的简单变量的运行时单元以及主程序中所需要的临时变量的单元一起形成一个数据区．

在编译时，任何变量的运行时单元都能由一对有序数（数

据区编号，区距)表示，其中数据区编号是分配给数据区的唯一编号，区距是相对于该数据区起点变量的地址．于是，对于编号为 3 的数据区，它的第一个单元表示为 (3, 0)，第二个单元表为 (3, 1) 等等．当然，在我们生成引用某个变量的代码时，必须把它转换成变量的实际地址．通常我们把数据区的**基始地址**（即第一个单元的机器地址）放在一个寄存器中，并通过寄存器的内容加上区距来引用该变量．于是，这一对有序数(数据区编号，区距)就转换成另一对数(基始地址，区距)了．

数据区分为两类：**静态**的和**动态**的．在运行前，静态数据区有一组分配给它的固定单元．在整个运行过程中，这些单元总是分配给它用的．因此，静态数据区中的变量都通过绝对地址来引用，而不是由（基始地址，区距）这一对数来引用．

一个动态数据区在运行时不是一成不变的，有时引入，有时退出．每当数据区退出时，该数据区中的所有值便失去意义．例如，存放在 ALGOL 过程中变量的数据区就是这种情形．当调用过程时，就调用 GETAREA，对其固定长的数据区分配存贮．当从过程返回调用程序之前，它调用 FREEAREA，撤消所分配的空间．注意，每次分配给过程数据区的存贮单元不一定是相同的．但是，GETAREA 总送回所分配的数据区基始地址，而过程又总把这个地址放在同一单元中，譬如说 BA．这样，数据区中任何变量的地址总是 CONTENTS(BA) + OFFSET，即 BA 中的内容加上区距．如果递归地调用过程，那末，它在主存中将有几个数据区付本，每调用过程一次，就分配给一个，这就满足了我们的要求，每次调用过程时，它应有一个新区起作用．这样的数据区并不属于过程本身，而是属于过程的**执行**．

```
BEGIN PROCEDURE A;
      BEGIN PROCEDURE B; BEGIN ··· END;
            PROCEDURE C; BEGIN ··· END;
      END;
      PROCEDURE D; BEGIN ··· END;
  :
END;
```

图 8.1 一个 ALGOL 程序的过程结构

在执行过程的任一位置，有可能引用几个数据区中的变量．例如，考虑具有图 8.1 所示过程结构的 ALGOL 程序．如果主程序和每个过程都有自己的数据区，那末，当执行过程 B 时，我们能引用过程 A，B 和主程序的数据区．我们按照引用它们的先后顺序去安排它们，把所有能引用的数据区地址收集在一个**区头向量**中．对上例来说，当执行过程 B 时，区头向量将是

区头向量

主程序的数据区地址
过程 A 的数据区地址
过程 B 的数据区地址

于是，当产生数据区(即该数据区获得存贮空间)时，我们就有一个固定地方存贮它们的地址．为此，我们必须回答下面一些问题：区头向量本身在哪里指出？我们按照怎样的顺序把数据区地址安放在区头向量中？当我们讨论某个语言的实现时就要回答这些问题．

一般来说，在运行过程中，我们必须采用几个区头向量，为执行程序的各不同部份所用．在上例中，假设过程 B 调用过程 D．此时，我们必须构造一个区头向量，它仅包含 D 和主程序的数据区地址，以便在 D 中使用它们．但是仍旧要保留原先的区头向量；当 D 做完后返回到 B 时，再次用到它．**现役区头向量**总是用以指出当前所引用的各数据区．

8.1 的练习

1. 当执行图 8.1 的过程C时，当前的区头向量应是什么样子？当执行过程A时，又是什么样子呢？当执行图 8.5 (p.241) 的过程D时，当前区头向量应是什么样子？当执行过程B时，又是什么样子呢？

8.2 属性单元

如果在编译时编译程序已经知道变量的全部属性，那末，编译程序就能根据这些属性产生引用这些变量所需的代码．但是在许多情况下，这些信息是在运行时动态指出的．例如，在 ALGOL 中，数组各维的上下界在编译时还不一定知道．而且，在某些语言中，实在参数无需在类型上恰好与形式参数相对应．此时因为编译程序必须考虑属性的各种可能的组合，从而不能生成简单、有效的代码．

为了解决这个问题，编译程序不仅要对变量分配空间，而且要对其**属性单元**（韦氏大字典第七版中把它定义为 *gauge*, *pattern* 或 *mold*）分配空间，该属性单元说明了在运行时所确定的属性．因为运行时已经知道属性是什么以及如何变化，那末，就要在属性单元中填上或改变其属性．

让我们举一个简单例子加以说明，假定形式参数是一个简单变量，而对应的实在参数可以具有不同类型，传送给过程的实在参数可看成具有下面的形式：

属性单元　0 = 实数，1 = 整数，2 = 布尔量等
值地址(或值本身)

当过程中引用形式参数时，该过程必须检查或解释这个属性单元，然后实现所需的类型转换．当然，可以调用另一个程序以实现这种转换工作．

在许多情况下，由于不知道变量的空间属性，所以编译程序不能对这些变量的值分配存贮空间．例如 ALGOL 中的数组就属于这种情形．编译程序所能做的工作只是在数据区中，对那些与变量相联系的、固定长的属性单元分配存贮空间．在运行时，当知道空间属性时，就调用 GETAREA 去分配存贮空间，并把这个存贮地址放在属性单元中．以后总是通过这个属性单元去引用相应的值．

结构和记录也需要有属性单元，实际上，它们需要更复杂的属性单元用以表明分量和子分量间是如何联系的．这些将在 8.6 中讨论．

允许运行时改变的属性越多，运行时所必须做的工作也就越多．把 FORTRAN 同 ALGOL 或 PL/1 相比，它之所以能编出更有效的程序，其原因就是 FORTRAN 在编译时已经知道了所有属性，从而无需分配属性单元，也无需对它们进行解释．

8.3 基本数据类型的存贮分配

必须把源程序的数据类型映像到等价的机器的数据类型．对某些类型来说，映像之间存在一一对应关系（如整数、实数）；对另一些类型来说，也许需要几个机器字．简短地说，有如下几种：

1. 整型变量通常占有数据区中的一个字或一个单元；其值按机器内部的标准整数形式存贮．

2. 实型变量通常占用一个字．如果要求高精度，那末可以用几个字，例如有些机器可用“双倍字”浮点格式．在没有浮点数表示形式的机器上，可用一个字存放指数；另一个字存放规格化的尾数，然后必须用子程序解释的办法来实现浮点

运算.

3. 布尔量或逻辑变量占用一个字,用零表示"假"值,非零或 1 表示"真"值. "真"、"假"值的具体表示,可以通过机器所能用的逻辑运算指令加以确定. 我们也可以用一个二进位表示每个布尔变量,并且尽可能地把许多布尔变量或常数合并在一个字中.

4. **指示字**是(引用)另一个值的地址. 在某些情形下,必须把一个指示字表示成两个相邻单元,一个指向(或就是)属性单元,属性单元指出了当前值所在地址,另一个指出真正的值. 如果在编译时对指示字的每一使用还不能决定所引用值的类型时,那末就要用两个单元.

8.4 数组的存贮分配

这里,我们假定数组或向量的每个元素都占有一个存贮单元. 多于一个单元的更一般情形留给读者考虑.

向量

通常把数据区中的一片相连单元分配给向量的元素,并按递增或递减顺序存放它们. 采用何种表示将根据所用的机器及其指令系统而定. 例如,在 IBM 7090 上,从基始地址**减去**下标寄存器的内容得到一个有效地址,于是,数据元素应按递减顺序存放了.

假定我们采用通常习惯的递增顺序. 于是,由 ARRAY A[1:10] 定义的数组元素是按顺序 A[1], A[2], ···, A[10] 存放.

矩阵

有几种存贮二维数组的方法. 通常方法是按行(即先按行排序)把它们存放在数据区中;也就是说,对 ARRAY

A[1:M, 1:N] 所说明的数组是按下列顺序排列的:

A[1, 1], A[1, 2], ···, A[1, N], A[2, 1], ···,

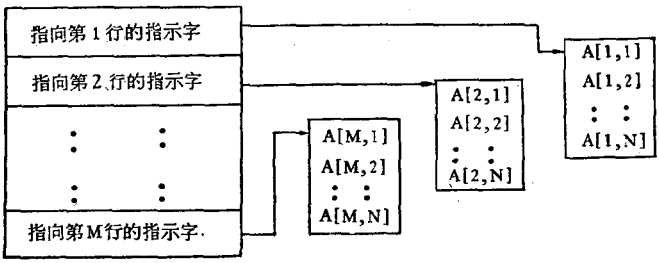
A[2, N], ···, A[M, 1], ···, A[M, N].

上面顺序指出了元素 A[i, j] 被分配在地址 ADDRESS(A[1, 1]) + (i − 1)*N + (j − 1) 中,我们将它写为

(8.4.1) (ADDRESS(A[1, 1]) − N − 1) + (i*N + j)

第一个量是一个常数，而且仅需计算一次. 于是，为了找出 A[i, j]，要求作一个乘法和两个加法. IBM FORTRAN IV 要求数据是按列存放的.

第二种方法对每一行都要分配一个单独数据区，而且还要有一个由指示字组成的向量,用以指向这些数据区的位置. 每行元素按递增顺序存放在一片相连单元中. 于是，说明 ARRAY A[1:M, 1:N] 将产生



当数组本身存贮在各独立的数据区时，指向这些行的指示字向量也存放在与该数组有关的数据区中. 数组元素 A[i, j] 的地址是 CONTENTS(向量的地址 + i − 1) + (j − 1).

这种方法不要求乘法运算，这是我们之所以采用它的一个理由. 另一个理由是无需把所有行同时存放在主存中. 为此,指向行的指示字还要包含一个值,当相应行不在存贮器中时,它引起硬设备或软设备中断. 当发生中断时,就把所需要的行取到主存中,代替另一行. 在 B5500 计算机上 Burroughs

扩充的 ALGOL 编译程序就采用了这种方法．如果所有的行都在主存中，那末，采用行指示字向量要占用较多的空间．

第三种方法是，当已知矩阵是稀疏矩阵（即大部份元素是 0）时采用的方法，此时，可以采用散列地址方案，它根据数组元素 A[i, j] 的值 i 和 j，把数组元素**散列**在相对小的表中（见第九章散列表方法）．在该表中仅记录非零元素．

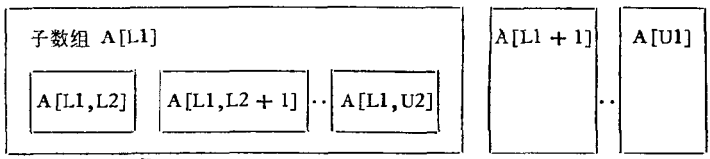
多维数组

譬如说，我们对如下 ALGOL 说明考虑多维数组的存贮分配和引用：

ARRAY A[L1:U1, L2:U2, ···, Ln:Un]

下面要介绍的一种方法是把二维情形所介绍的第一种方法加以推广而成的一般方法，当然它也适用于一维情形．

子数组 A[i, *, ···, *] 是按顺序 A[L1, *, ···, *], A[L1 + 1, *, ···, *], ···, A[U1, *, ···, *] 存放的．在每个子数组 A[i, *, ···, *] 内是子子数组 A[i, L2, *, ···, *], A[i, L2 + 1, *, ···, *], ···, A[i, U2, *, ···, *]．对每维都如此重复．于是，当我们按递增顺序排列各数组元素时，最末尾的下标改变得最快：



现在问题是如何去引用数组元素 A[i, j, k, ···, l, m]．假设

d1 = U1 − L1 + 1, d2 = U2 − L2 + 1, ···,

dn = Un − Ln + 1.

即 di 是第 i 维中不同下标值的个数．根据 2 维情况，我们可以找出子数组 A[i, *, ···, *] 的开始位置

BASELOC + (i − L1) * d2 * d3 * ··· * dn.

其中 BASELOC 是第一个元素A [L1, L2, ···, Ln] 的地址，d2 * d3 * ··· * dn 是每个子数组 A[i, *, ···, *] 的长度．然后对该值再加上一项

$$(j - L2) * d3 * \cdots * dn$$

就能找到子子数组 A[i, j, *, ···, *] 的开始位置，重复这一方法，最终求得下标变量的地址为

$$\begin{aligned}(8.4.2)\quad & BASELOC + (i - L1) * d2 * d3 * \cdots * dn \\ & + (j - L2) * d3 * \cdots * dn \\ & + (k - L3) * d4 * \cdots * dn + \cdots \\ & + (l - L[n-1]) * dn + m - Ln.\end{aligned}$$

经因式分解后得到

$$(8.4.3)\qquad CONSPART + VARPART$$

其中

$$\begin{aligned}(8.4.4)\quad & CONSPART = BASELOC \\ & - ((\cdots((L1 * d2 + L2) * d3 + L3) * d4 \\ & + \cdots + L[n-1]) * dn + Ln)\end{aligned}$$

又

$$\begin{aligned}(8.4.5)\quad & VARPART = (\cdots((i * d2 + j) * d3 \\ & + \cdots + l) * dn + m\end{aligned}$$

因为 CONSPART 仅依赖上下界和数组的存贮位置，所以其值仅需计算一次，且应保存起来；VARPART 则依赖下标 i, j, ···, m 和每维中下标值的个数 d2, d3, ···, dn. 上式的计算看起来非常吓人，但是，化为这种形式之后，就可以通过执行下列语句很容易地把值计算出来，

```
VARPART: = 第一下标 (i)
VARPART: = VARPART * d2 + 第二下标 (j)
VARPART: = VARPART * d3 + 第三下标 (k)
    :                    :
VARPART: = VARPART * dn + 第 n 下标 (m)
```

于是，我们就有方法来对付这种麻烦了；采用上述方案计算下标变量地址时，编写程序或生存代码都是非常容易的．事实上，由于这种计算的迭代性质，生成引用任何多维数组的代码就像生成二、三维数组的代码一样容易．

在进行代码优化的情况下，最好采用没有对 di 进行因式分解的（8.4.2）（当然要把不变部份和可变部份分开），因为此式能把计算分成几个独立部份．在第十八章中，我们将看到这种方案．

内情向量

在 FORTRAN 中数组的上下界在编译时都是已知的．因此，编译程序把上下界和值 d1，d2，…，dn 作为常数，去分配数组的存贮以及生成引用数组元素的代码．在 ALGOL 和 PL/1 中，因为上下界有时要在运行时才能计算出来，所以不能使用上述方法．因此，在运行时需要一个数组的属性单元，给出必要的信息．我们把这个属性单元称为数组的**内情向量或信息向量**．在编译时内情向量具有固定长度，如此就能在数据区中给它分配存贮空间，使它与相应数组联系．一直到运行之前，还不能对数组本身分配存贮空间，但当进入说明该数组的分程序时，就能计算出数组的上下界，并调用数组的分配程序．该程序计算出所需要的单元个数，调用 GETAREA，按此长度分配数据区，并把所需的全部信息填到内情向量之中．图 8.3 给出了数组分配程序的框图．

需要把什么信息填到内情向量中呢？就上面所提出的 n 维方案来说，我们需要值 d2，…，dn 和 CONSPART．这是勉强够用的最少信息．如果在引用数组之前还要检查下标是否有意义，我们还应该包括它们自身的上下界．图 8.2 指出了可能要求有的信息．