结构化程序设计和DITSF语言

钟万勰　裘春航　邓可顺 编著

科学出版社

结构化程序设计和DITSF语言

钟万勰　裘春航　邓可顺 编著

科学出版社

1988

内 容 简 介

本书结合程序设计方法学和程序正确性证明的思想，介绍大连工学院研制的结构化 FORTRAN 语言 DITSF，并以此为工具，阐述程序设计的基本原理和方法．内容包括：FORTRAN 语言，结构化程序的概念，结构化程序设计语言 DITSF，DITSF 语言翻译程序 DITSFTR 的设计与实现，DITSF 语言的应用举例，DITSF 语言在不同类型机器上的实现，结构化程序正确性证明等．

本书可作为大专院校学生、研究生的程序设计教材，也可供应用计算机的科研人员、工程技术人员和管理人员学习、参考．

结构化程序设计和 DITSF 语言

钟万勰　裘春航　邓可顺 编著

责任编辑　那莉莉　乐嘉敏

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

*

1988 年 1 月第 一 版　开本：787×1092　1/16

1988 年 1 月第一次印刷　印张：16 1/2

印数：0001—3,800　字数：381,000

ISBN 7-03-000037-4/TP・5

定价：3.90 元

前　言

随着电子工业的迅速发展，计算机硬件发展很快，而作为逻辑产品的软件，其发展速度相形之下却慢得多．这就导致了所谓“软件瓶颈”的现象，即人们的能力不足以编制出各种各样的软件以充分利用计算机资源．为了改变这种状态，除了增加对软件的投资外，还加强了对程序设计方法论和软件工具的研究．

要提高程序设计的效率，应该把程序的说明、设计、编写、调试和维护作为一个有机的整体来考虑．从这一基本观点出发，在程序质量评价标准、程序表示和程序设计概念本身等诸方面已发生了一系列的变化．总的来说，为了研制出可读性好、正确和可靠程度高、易于调试与维护的软件，程序设计已从强调灵活的技巧和局部效率向着注重程序结构化和使用完整的概念和方法的方向发展，并逐步形成了相应的程序设计方法论．结构化程序设计就是其中行之有效的代表．

结构化程序设计的最初思想见于 E. W. Dijkstra 1968 年给美国计算机学会通讯的一封题为“GOTO 语句是有害的”信件中．经过十几年的发展，这一思想现在已具有相当广泛的内容．按照结构化程序设计的观点，首先要求编程结构化，也就是采用几种恰有一个入口与一个出口的定型控制结构来展开程序．在此基础上，结构化程序设计需要在不同的抽象级别上，进行多次自上而下的设计、编程和调试，即经历多次综合．与传统的自下而上的方法相比，按照结构化程序设计研制的软件，通常其半成品的“能见度”较高，程序结构和计算结构比较符合并易于维护和发展．

与结构化程序设计相适应，我们应当用结构化程序设计语言来编程．本书将要介绍并使用的 DITSF 语言即为一种结构化的 FORTRAN 语言．

学会使用 DITSF 语言是很容易的，因为它的逻辑功能比较符合人的思维．尤其是对于熟悉 ALGOL60 程序设计的人员来说，只要学习一下 FORTRAN 的最基本规则，即可应用自如．另外，DITSF 程序与 ALGOL 程序间的转换也相当方便．

在写作过程中，我们力图使本书成为体现程序设计新思想的教材，因此只要有适当机会，便结合具体例子用程序逐步展开的方法，阐明如何从程序说明逐步精化到一个可在计算机上实现的程序．另外，在内容选择上也尽可能做到全面，其中包括各种类型的数值与非数值程序设计的问题．本书是在研制与使用 DITSF 语言的基础上，由讲授多次的教材改写而成．我们热忱地期望读者对本书的缺点或错误提出宝贵的批评意见．

作者

1986 年 5 月

目　　录

第一章 绪 论

1.1 计算机硬件与软件

电子计算机系统由计算机硬件和计算机软件组成．硬件是指数据处理中使用的物理设备，是计算机系统中的实际装置的总称．尽管计算机的种类繁多，然而组成其硬件的基本结构却是相似的．硬件一般包括运算器、控制器、内存贮器、外存贮器与输入输出设备等五大部分，如图 1.1 所示． 运算器是执行算术运算与逻辑运算的装置；控制器规定计算机执行指令的顺序，并根据指令中的信息送适当的控制信号到运算器和机器中的其它部分；存贮器用来存放数据以及指示计算机对数据如何进行操作的指令，其中内存贮器是可由运算器直接访问的存贮器，外存贮器又称后备存贮器，是指其它存贮数据和指令的媒体，如磁带、磁盘等．通常，运算器、控制器与内存贮器合在一起称为计算机的主机或中央处理机（CPU）．

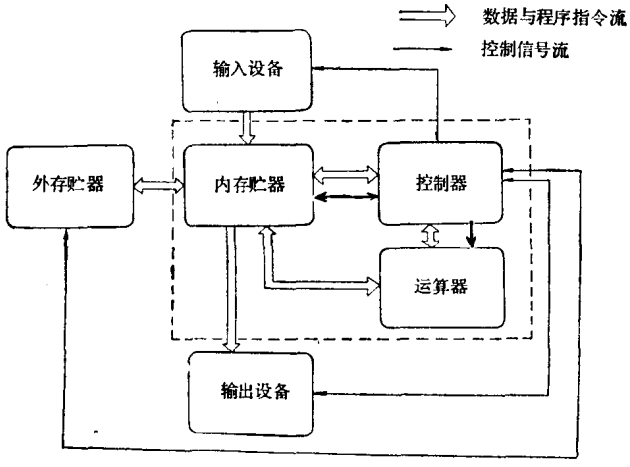


图 1.1

计算机软件是相对于硬件而言的一个概念，泛指为了使用计算机而需要的各种程序及其有关的文档资料，如使用说明书和设计说明书等．软件可分为系统软件与应用软件两大类．其中系统软件是为了提高计算机的使用效率，便于用户使用和维护计算机而研制的各种软件，如操作系统、各种语言的编译程序、故障检测和诊断程序、程序库等．至于应用软件，其范围更为广泛，凡是为解决某一特定问题而开发的各种软件都属此列，如有限元结构分析软件、情报资料自动检索软件等．应用软件的各个子领域为：数值计算软件、数据库管理系统、人工智能、自然语言处理、非常高级语言与自动程序设计系统及嵌入计算机的应用等．其中“嵌入计算机的应用”这一术语出现于七十年代中期，它用于控制宿主系统，如飞机、轮船、工厂的运行．

硬件提供计算机系统应用的基础，而软件扩大了它的功能．现在，软件的研制和生产

已成为一种新兴行业——软件工程.

到目前为止,计算机已经历了四个世代的发展，这四代计算机的硬件分别由电子管、晶体管、中小规模集成电路与大规模集成电路组成.

以大规模集成电路为硬件特征的第四代计算机的研制开始于七十年代初，至今还是兴旺时期. 在此期间出现了价格便宜的微型机与运算速度每秒超过亿次的巨型机.

1.2 程序设计概念的发展

随着电子工业的迅速发展,硬件发展得很快. 有关资料表明，自 1945 年以来硬件的成本每隔两、三年便下降一半. 但由于软件是通过程序设计而生成的一种逻辑产品,它主要靠人工研制,再加上软件平均复杂程度的增加,因此软件的发展速度相形之下便缓慢得多. 这就导致出现了所谓软件瓶颈，即出现了人们的能力不足以编制出各种各样的软件以充分利用计算机资源的现象[1,2]. 为了改变这种不合理状态，除了增加对软件的投资外,还在不断加强对程序设计方法论以及能缩短其它软件研制周期的软件工具的研究.

从一般意义上讲,程序设计是一个包括程序(软件)的说明、设计、编程、调试与维护等内容的全过程.

程序的说明,是指给程序的功能(做什么)以明确的陈述. 显然,程序的功能完全由程序本身(如何做)规定，因此说明是重复的. 这种重复性构成了验证程序(如何做)及其说明(做什么)之间的一致性(即程序正确性)证明的基础. 程序的设计包括数据结构与算法的设计、文档的编制、模块的划分等. 设计能帮助我们进一步弄清问题的基本结构. 所谓调试(或称测试)是在经过选择的输入下,由程序执行的结果来推断程序行为性质的过程. 调试只能说明有错,而不能证明无错. 虽然，近十五年来已发展了各种非形式化、半形式化与形式化的证明方法,其中包括用完全形式化的办法建立起来的计算机验证系统,但这种途径是很费钱的. 据估计，为了证明程序是正确的需要付出十倍于编写程序所需的费用. 因此，程序正确性证明当前尚只用于出错后付出代价很大的场合. 发现程序错误以及进一步对其查错、改错的主要手段仍然是调试与维护. 图 1.2 给出了大型软件正确率与完成时间的关系曲线. 它表明对大型复杂系统,只要出错率Δ小到一定限度,就被认为可以了.

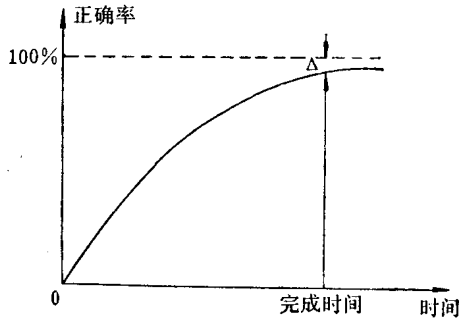


图 1.2

近年来的统计数字表明“大约有 70% 以上的现在程序设计是在维护旧的程序”. 到七十年代中期,人们已逐步认识到软件的维护是软件研制中的一个关键领域，并注意到为了提高程序设计的效率,降低软件研制费用,必须把程序的说明、设计、编程、调试与维护等几方面作为互有影响的整体来考虑. 从这些基本观点出发，程序设计已从强调灵活的技巧和局部效率向着强调**程序结构化**和使用**系统的概念与方法**的方向发展. 程序设计发展到今天已不再单纯是一项技术,而是一门科学了. 因此,不应当把局部效率与特殊

技巧作为评价程序好坏的基础．与此同时，逐渐形成了一套评价程序好坏的客观标准，它们大致可描述如下：

- 程序的可读性
- 正确性与易证明性
- 易于调试与维护
- 可靠性
- 易移植性

这些标准之间是互相联系的，其中每一项只是从一个侧面来评价程序质量．可读性好的程序，指的是一种比较容易为本人或同行们理解的程序．这样的程序的正确性程度也往往较高，同时也便于调试与维护．仔细阅读程序虽然不能作为正确性证明，但这也是防止一些错误（包括程序说明中的错误）的有效方法．通常，人们把可读性作为一条主要的评价标准．实际上，有经验的程序设计人员已自觉地避免使用各种特殊的技巧，而尽量采用系统化的、简明易懂的方法．

程序的可靠性是一个比较广泛的概念，因为它包括正确性与容错性这两个方面．如前所述，正确性只涉及程序能否完成预先说明的功能．但实际情况是复杂的，在程序执行过程中会发生难以预料的情况，其中有运行环境的突然变化，出现不满足要求的因素（如不合格的原始数据）以及由于难以做到或错误的功能说明等而引起的非正常状态． 所谓容错性能，就是在这些非常情况下，程序能继续执行一些合理操作的功能．

程序的易移植性，指的是在一种类型机器上发展的程序容易移植到另一种类型机器上．这在程序（软件）日益商品化的今天，更显得重要了．为此，应使所设计的程序尽可能做到与机器无关，而把不可避免的、与具体机器有关的内容屏蔽在极少的几个低级模块中，以便在移植时只作很少的改动即能适应不同类型的机器．

在易于调试与维护这一指标中，包括程序的易修改性和可扩展性．由于程序的改错与完善是调试与维护中经常要做的工作，因此要求程序易修改，即要求组成程序的各模块之间的联系尽可能弱．另一方面，为了提高竞争性，程序应具有良好的可扩展性能，以适应客观实际发展的需要．

1.3 结构化程序设计的概念

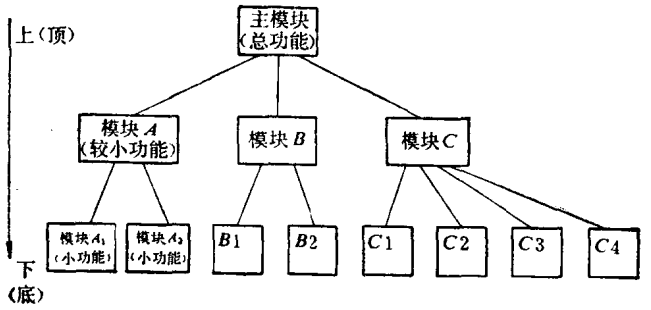
结构化程序设计的最初思想见于 E. W. Dijkstra 1968 年给 ACM 通讯的一封题为“GOTO 语句是有害的”信件中． 1972 年他又提出结构化程序的主要功能是保持正确性证明的可行性的观点[3,4]．经过十几年的发展，结构化程序设计已具有很广泛的内容，大体上可以归纳为如下几点：

（1）编程书写结构化．采用几种恰有一个入口与一个出口的定型结构来展开程序（见 3.3.1 节），从而使程序呈现为一种逻辑清楚的分层结构，使人易懂并保持正确性证明的可行性．

（2）主要采用自上而下、逐步精化的设计方法，即先全局后局部、先整体后细节、先抽象后具体的设计方法．但实际问题往往是比较复杂的，为了提高效率，常常伴随使用关键部分优先考虑的设计方法．

(3) 用模块式结构来组织程序，如图 1.4 所示(详见 3.5 节)。

(4) 避免过多使用 GOTO 语句，特别是逆转的 GOTO 语句。



(a)

输 入 (Input)	处 理 (Process)	输 出 (Output)
入口条件	功能: 1.····· 2.····· 3.····· ·····	出口条件

(b)

图 1.3 (a) 模块划分图；(b) 每一模块的接口与功能

与结构化程序设计的要求相适应，应当使用结构化程序设计语言来设计与编写程序。结构化程序设计语言又称为第二代高级程序设计语言。这种语言可以通过两种途径产生。其一是重新设计新的结构化程序设计语言，例如 PASCAL；第二种途径是在原有高级程序设计语言的基础上增加一些新的语句结构，以加强逻辑功能，改善表达能力。由此把它改造成为结构化程序设计语言，例如结构化 FORTRAN 语言。

大家知道，FORTRAN 语言是最早流行的一种高级语言，但正如 E. W. Dijkstra 所指出的那样，它是婴儿时期的产物，存在不少弱点，不适合于进行结构化程序设计。另一方面，FORTRAN 语言有比较统一的国际标准，各种类型的计算机都配有相当完善的 FORTRAN 语言编译程序，在过去的二十多年中又积累了大量的用 FORTRAN 语言编写的软件。因此，改进 FORTRAN 语言，使之适应结构化程序设计的特点与要求是一件很有意义的工作。

为了推进我国的结构化程序设计与研制大型软件的需要，我们参考美国的一个结构化 FORTRAN 文本[10]，并结合我国广大程序设计人员熟悉的 ALGOL 60 语言，设计了一个结构化 FORTRAN 语言文本 DITSF 及其相应的翻译程序 DITSFTR。本书的以后各章将系统介绍 DITSF 与 DITSFTR。

DITSF 是通过翻译程序或“前处理器” DITSFTR 来实现的。DITSFTR 装在带有 FORTRAN 语言的计算机上，它能自动将 DITSF 源程序翻译成该机的 FORTRAN 语

言源程序，整个工作流程如图 4.1 所示． 近几年来，DITSFTR 已先后安装在西门子 7738，7760，IBM 370-138，IBM 4341，CYBER 18-20，CYBER 170，180，PDP 11-23，11-34，11-70，WANG 2200-VS，FELIX C-512，ND-500，M-150H，ACOS-4，EC-1040，CROMEMCO Z80，DJS-153，DPS-8，DPS-6 和 IBM-PC 及其兼容机上．

第二章 FORTRAN 语言

FORTRAN 语言的研究是五十年代初在美国 IBM 公司支持下开始的，1956 年首次公开发表并在 IBM 704 计算机上实现，称之为 FORTRAN I. 以后它发展为 FORTRANII (1958年) 和 FORTRAN IV(1962年). 自 1962 年 5 月起，对这种算法语言进行了标准化工作. 1966 年 3 月美国标准协会制定了美国标准 FORTRAN(相当于 FORTRAN IV) 和美国标准基本 FORTRAN(相当于 FORTRAN II)等两个标准文本. 1972 年 7 月国际标准化组织制定了推荐标准，分三级：完全级(一级)，中间级(二级)和基本级(三级). 现在一般所称 FORTRAN IV 就相当于 ISO 标准的完全级.

各种程序设计语言本身是不依赖于机器的，但是用这些语言编写的软件，最终还是要在一台具体的计算机上执行. FORTRAN 语言也不例外. 事实上，没有任何一个 FORTRAN 语言的编译程序是不多不少、完完全全地实现了标准规定的. 由于计算机的功能差异，输入输出设备的配套和管理方式不同，因此许多编译程序都对标准文本作了这样或那样的扩充和限制. 特别是由于计算机的不断发展，算法语言不可避免地要引进新的概念增加新的功能. 我们不打算结合一台具体的机器讲，而是介绍 ISO 标准完全级 FORTRAN(以下简称标准 FORTRAN IV)，如果没有特别说明，凡是提到的FORTRAN 都是指 ISO 标准完全级 FORTRAN.

2.1 引 言

2.1.1 计算机存贮信息的单位

一个程序编写者，在编写程序时必然面临两方面的问题，即用什么算法和什么数据结构. 可以简单地说，算法和数据结构相结合就是一份程序. 数据的实现是与计算机存贮信息的单位以及信息的内部表示紧密相关的，而且因机器不同而异. 因此，有必要介绍一下计算机存贮信息的单位.

一切数值、字母、符号以及它们的各种组合，都是计算机处理的对象，称之为信息. 信息在计算机内部表示成一串串的数码. 计算机内部用的不是通常的十进制数，而是二进制数. 所以，计算机内部存贮信息的最小单位就是一个二进制位 (bit).

计算机的存贮器就象一长列房间，里面存放信息. 为了从中取出或向里放入信息，就要把这些房间编号，即编址. 如果认为一个二进制的位就是一个房间而编一个号，那么一台中等容量的计算机的内存贮器就会有几百万个二进制位，编号量太大，查找地址也费时间. 特别是很多有实际意义的信息，如数值、字符、机器指令等，都不是用一个二进制位所能表示的，往往要用许多个二进制位才能表示. 所以用二进制位作为编址的最小单位显然是不必要的，也是不好的.

有的计算机用“计算机字”，简称“字”作为编址单位，如 TQ-16 机，CDC CYBER 18-

20 机，CYBER 170，180 系列机，IBM 1130 机，ICL 1900 系列机[1]. 一个字由若干个连续排列的二进制位组成，二进制位的个数称为字长. 不同机器的字长是不一样的，如 TQ-16 机的字长为 48 位，CYBER 18-20 机和 IBM 1130 机的字长为 16 位，ICL1900 系列机的字长为 24 位，还有的机器字长为32位或 60 位的. 显然，字长越长，所表示的数值的位数就越多，精度就越高，但机器的存贮量就越少.

还有一些机器，如 IBM 360/370，SIEMENS 7.000 系列机等，从节省内存出发，并不是以字作为编址的基本单位，而是以比字短的字节（byte）作为编址的基本单位，同时保留字的概念. 每个字节从逻辑上讲是由 8 个连续的信息位组成，最多可表示 256 种不同的信息状态. 我们可以根据所编的地址向任意一个字节存取信息. 每个字由连续的 4 个字节组成，4 个字节的字称为"整字"或"全"字. 相对于整字还有半字和双字，分别由连续的 2 个字节和 8 个字节组成. 整字、半字或双字的地址分别是它们的第一个字节的地址. 为了寻址方便，规定整字、半字或双字的地址数应分别能被 4，2 或 8 整除，记住这一点是非常重要的.

常用的一种说明一台计算机内存单元总数(容量)的单位是 K，其中 K = 1024. 假如某台计算机的内存容量是 32K 字，那么它的内存容量为 32 × 1024 = 32768 个以字为单位的单元. 例如 SIEMENS 7738 主机内存贮器的容量为 512KB，这里 B 是字节 byte 的缩写，它的内存容量为 512 × 1024 = 524288 个字节，相当于 131072 个字(字长为 32 位).

2.1.2 FORTRAN 语句分类及程序结构

语句分类

任何一个 FORTRAN 源程序都是由若干个语句组成的. FORTRAN 语句分为可执行语句和非执行语句两大类. 可执行语句指出所要执行的实际操作，它们在编译之后生成一系列的机器指令. 非执行语句用以向编译程序提供所需要的信息，例如程序结构、数据类型、存贮区分配、数据初值以及输入输出格式等等. 非执行语句不生成机器指令，它的作用是为可执行语句的执行作好准备. 语句的一览表如下：

- 可执行语句
  - 赋值语句
    - 算术赋值语句*
    - 逻辑赋值语句
    - 标号赋值语句
  - 控制语句
    - 转语句
      - 无条件转语句*
      - 计算转语句
      - 赋值转语句
    - 条件语句
      - 算术条件语句*
      - 逻辑条件语句*
    - 循环语句*
    - 调用语句*

1) ICL 是英国国际计算机有限公司 International Computers Limited 的缩写，该公司生产 ICL 1900 和 ICL 2900 系列计算机.

- 返回语句*
- 继续语句*
- 停语句*
- 暂停语句
- 输入输出语句
  - 读语句*
  - 写语句*
  - 输入输出辅助语句.

非执行语句
- 函数子程序语句*
- 子例程子程序语句*
- 数据块子程序语句*
- 说明语句
  - 类型语句*
  - 维数语句*
  - 公用语句
  - 等价语句*
  - 外部语句
- 格式语句*
- 数据初值语句
- 语句函数定义语句

其中带＊号的是应掌握的基本语句. 除了算术赋值语句、逻辑赋值语句和语句函数定义语句外，FORTRAN 语言的其它语句都是以某一个特定的字，例如 INTEGER, IF, CALL 或 DO 等开始的. 我们把区别语句类型的这些字称为关键字(注意，关键字不是 ISO 标准 FORTRAN IV 语言中的术语). 在 FORTRAN 语言中没有要保留的关键字，也就是说，组成关键字或关键字一部分的字符串，也可以作为符号名或符号名的一部分来使用[1]. 因此，编译程序将根据整个语句的形式来判断是否是关键字，从而属于何种语句.

程序结构

FORTRAN 语言是块结构，或者说是段结构. 每个 FORTRAN 源程序是由一个或几个相对独立的程序段组成的. 这些程序段在 FORTRAN 语言中称之为程序单位. 程序单位分为主程序段和子程序段，子程序段又分为过程子程序段和说明子程序段，而过程子程序段又划分为函数子程序段和子例程子程序段:

- 程序单位
  - 主程序段
  - 子程序段
    - 过程子程序段
      - 函数子程序段
      - 子例程子程序段
    - 说明子程序段

在介绍程序段的结构之前先讲一下有关的两个名词: 程序部分和程序体. 程序部分是由数据初值语句和语句函数语句、可执行语句、格式语句等组成的 FORTRAN 语句系列，其中必须要有一个可执行语句. 程序体是由说明语句、格式语句、程序部分和结束行

1) 例如:在语句 DO = B*C − A 中，DO 不是关键字(又称专用字)，而是一个符号名.

END 组成的语句系列,其中说明语句和格式语句可有可无.

主程序段是由程序体构成的,子程序段是由子程序语句和程序体组成的. 所以,不论是主程序段还是子程序段,都是以 END 行作为其结束标志,目的是通知编译程序,该程序段的全部语句到此结束了. 对于主程序段, ISO 标准并没有规定其开头语句,对于子程序段,规定了开头语句(即子程序语句). FORTRAN 编译程序将根据结束行、有无子程序语句以及怎样的子程序语句来区分是何种程序单位.

所以,一份 FORTRAN 源程序或者只由一个主程序段组成,或者是由一个主程序段和若干个子程序段组成. 这就是说,一份 FORTRAN 源程序必须且只须包含一个主程序段.

关于在同一份 FORTRAN 源程序中各程序段的前后次序, ISO 标准 FORTRAN 并没有什么规定,可以随便放. 但是,不少具体机器的编译程序要求把主程序段放在整个程序的最前面,即为第一段. 因此,我们建议大家写程序时,最好把主程序放在最前面,这样就不受机器条件的限制,减少转机带来的麻烦. 另外,从主程序是整个程序的主控段来看,把它放在整个程序的最前面也是合适的.

组成一个 FORTRAN 程序的各程序段是相互独立的. 每个程序段是一个独立的编译单位,其中所使用的所有标识符和语句标号等只局限于在本程序段内有意义,与其它程序段无关. 所以,即使在不同的程序段内使用了相同的标识符,一般说来它们所代表的量是不相同的.各程序段之间的数据,是靠虚元和实元(即形参和实参)的结合或公用存贮区来传递的,这些将在下面详细介绍. 各程序段之间的关系是: 主程序可以调用各子程序,某子程序可以调用其它子程序; 任何子程序不能直接或间接地调自己,即不允许递归调用.

FORTRAN 语言的这种分段结构的另一个优点是,可以使程序的编写和调试分段并行地进行. 一个大型程序,可以根据问题的具体内容或处理方法划分为若干个相对独立的部分,事先商定好各程序段之间的接口,就可以由几个人同时编写和调试,最后进行联合调试. 局部的修改只涉及到有关的程序段,不影响大局. 对于一些常用的计算方法或专业性的数据处理方法,可以把它们编成相应的子程序段,放在子程序库中供用户选用. 对于已经调通的程序,在进行局部功能扩充或修改时,只需要修改和调试这一部分,其它部分原封不动,甚至可以不必进行编译等等.

2.1.3 FORTRAN 语言的字符集

标准 FORTRAN IV 的字符共有 47 个,分三大类.

字母字符(共 26 个):

A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z

数字字符(共 10 个):

0 1 2 3 4 5 6 7 8 9

特殊字符(共 11 个),见表 2.1.

上述 47 个基本字符的意义大都是明显的,少数则不然. 例如星号"*"、斜杠"/"和小数点"."等,在不同的场合下有不同的含义,今后遇到时再作具体说明. 空白字符在印刷

上是空白，为了醒目起见，书写时通常用记号"␣"表示．字符是 FORTRAN 程序中的最基本的书写和印刷单位，程序中一般不允许出现这 47 个字符以外的字符．但是，在下面要讲的文字常数或文字格式中，可以使用更多的为硬件所允许的字符．当然，出现基本符号以外的字符，会使程序在不同类型计算机上互换遇到麻烦，因此，一般不使用它们．

表 2.1

字　符	名　字	字　符	名　字
+	加　号	,	逗　号
−	减　号	(	左括号
*	星　号	)	右括号
/	斜　杠	␣	空　白
=	等　号	$	货币号
.	小数点		

FORTRAN 语言中使用的字母都是大写字母，不允许出现小写字母．在一般情况下，程序中出现的小写字母，都作为相应的大写字母来处理．在写程序时，要求注意某些字符的写法，为了易区分起见，通常将数 0 写作 ∅ 以区别于字母 O，字母 I 的上下两横要明显，数字 1 的上头要带尖等．

关于货币字符，ISO 标准 FORTRAN 文本没有规定具体的符号，不少进口计算机都以美元符号 "$" 来表示，国产机器可以用人民币符号 "¥" 来表示．写程序时，货币字符只允许出现在文字常数、H 型字段描述符等之中．

有些计算机还增加了两个特殊字符：&(与号)和'(单引号)，例如 SIEMENS 7730，7738机和 CYBER 18-20 机都是这样．而且，SIEMENS 7.000 系列 PRIME 550，FELIX C-512 等机还允许将货币字符 $ 当字母字符用．

字符是组成信息的基本单位．字符在计算机内是以代码的形式表示的．目前，国际上通用的代码主要有两种，一是 EBCDIC 码，它由 8 位二进制数组成，或者说由两个十六进制数组成，如表 2.2 所示．由于大多数打印机仅限于使用 39 到 88 个字符，而 8 位可以表示 $2^8 = 256$ 个不同的字符，因此表 2.2 还空了很多，而且还增加了一些具有特殊意义的字符(某些终端设备的键盘上，有表示这些字符的键)，如 DEL 表示清除 NL 表示换行(或回车)等．虽然表中有大写与小写英文字母，但从一个仅能生成 64 个字符的设备上输入，只能用大写字母．其中 16 进制中的 10，11，12，13，14，15 分别用 A，B，C，D，E，F 表示．一般，IBM 系统与 SIEMENS (西门子)系统的机器都能用 EBCDIC 码．

另一种是 ASCII 代码，即美国信息交换标准代码．由于表示一个字符所用的位数不同，ASCII 代码对不同的机器可能不一样．表 2.3 给出了 8 位 ASCII 代码表，表 2.4 给出了 IBM 360 系统的 ASCII-8 (8 位)与 EBCDIC 码的对照表．

2.1.4 FORTRAN 语言的命名规则

在编写程序的时候，会遇到各种对象，如变量、数组、子程序等，都要用名字来标识它们，以便区分和引用．这与平常生活中对各种事物起名字是一样的，也和数学中用字母代替各种数量值是类似的．

表 2.2 EBCDIC 码

十六进制 ★	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0★	NUL	SOH	STX	ETX		HT		DEL				VT	FF	CR	SO	SI
1★	DLE	DC1	DC2	DC3		NL	BS		CAN	EM			FS	GS	RS	US
2★						LF	ETB	ESC						ENQ	ACK	BEL
3★			SYN					EOT					DC4	NAK		SUB
4★	空格										¢	.	<	(	+	|
5★	&										!	$	*	)	;	¬
6★	-(减)	/									∧	,	%	−	>	?
7★											:	#	@	'	=	"
8★		a	b	c	d	e	f	g	h	i						
9★		j	k	l	m	n	o	p	q	r						
A★			s	t	u	v	w	x	y	z						
B★																
C★		A	B	C	D	E	F	G	H	I						
D★		J	K	L	M	N	O	P	Q	R						
E★	Blank		S	T	U	V	W	X	Y	Z						
F★	0	1	2	3	4	5	6	7	8	9						