第 一 章

电子数字计算机概述

电子数字计算机是由电子元件构成的一种现代化计算工具，它好象一个自动化的数字加工厂，用数字作为原料，经过加工、运算，得出数字形式的成品．由于电子数字计算机具有运算速度快、存贮容量大、计算精度高等特点，因此，可以利用它来完成大量繁复的计算工作，解决科学技术、工程设计以及其他各个领域中的计算问题．

我国自行设计制造的709机和TQ-16机是目前使用比较广泛的两台中型电子数字计算机．

第一节 709电子数字计算机和TQ-16电子数字计算机简介

709计算机是以集成电路为主要元件的通用电子数字计算机，平均每秒钟可以完成十二万次运算．机器配置有自动化程序的编译系统——用机器语言编写成的一个加工程序，它能够把用算法语言写成的源程序编排、翻译成用机器语言表示的目标程序．TQ-16计算机基本上是对部分功能作了改进的709计算机，除了编译系统外，它还配置有简易的管理系统，具有如下三种功能：

1．使用外部设备的管理；

2．接受并处理算题人员对机器的干预；

3．指示机器运行状态，以便于算题人员对出错、故障的监督、维护和处理．

709 计算机、TQ-16 计算机都由内存、运控和外部设备三大部分组成：

内存 是指内存贮器，它用来存贮程序、原始数据和计算结果．709 机和 TQ-16 机的内存贮器是由磁芯体构成，它好比是一座大楼房，由许多个称为单元的房间所组成，单元的总数称为计算机的内存容量．磁芯体的单元是由许多个能够表示二进制数字的小元件串联而成的，小元件的个数称为单元的字长．另外，为了存取信息的需要，磁芯体的单元还人为地依次编上一个号码，称为单元的地址．

709 机和 TQ-16 机的内存容量是 32768，单元地址依次编号为 0～32767（八进制数编号时为 000000～077777）．每个单元的字长为 48 个二进位．

运控 是运算器和控制器的总称．机器的各种运算都在这里进行，各种命令都由这里发出．这一部分中，同算题人员直接有关的设备，709 机是一个人工控制台，TQ-16 机是一个使用控制台（包括控制台面板和控制台打字机）．利用这些设备可以使机器停止运算，启动工作，输入、输出信息以及改变单元内容或运算次序等．

外部设备 包括光电机、排印机、磁鼓和磁带机四种设备．

光电机用来输入程序和原始数据．排印机用来输出运算结果和程序．目前，709 机和 TQ-16 机都配有两台光电机和两台排印机．

磁鼓和磁带统称为机器的外存贮器．每台磁鼓有 14336 个单元．与磁鼓相比，磁带容量更大，可达几十万个单元，但

存取时间较慢. 目前，709机未配磁带机，只配有四台磁鼓，算题人员可以使用三台，磁鼓单元统一编号为0～43007(八进制数编号时为000000～123777). TQ-16机配有三台磁鼓、两台磁带机，磁鼓编号为0、1、2，算题人员可以使用1号鼓、2号鼓，磁鼓单元分别编号为0～14335(八进制编号时为000000～033777). 磁带机编号为0、1，对算题人员一般只开放0号磁带机，以组分段形式存取信息.

机器各部分的联系示于图1.

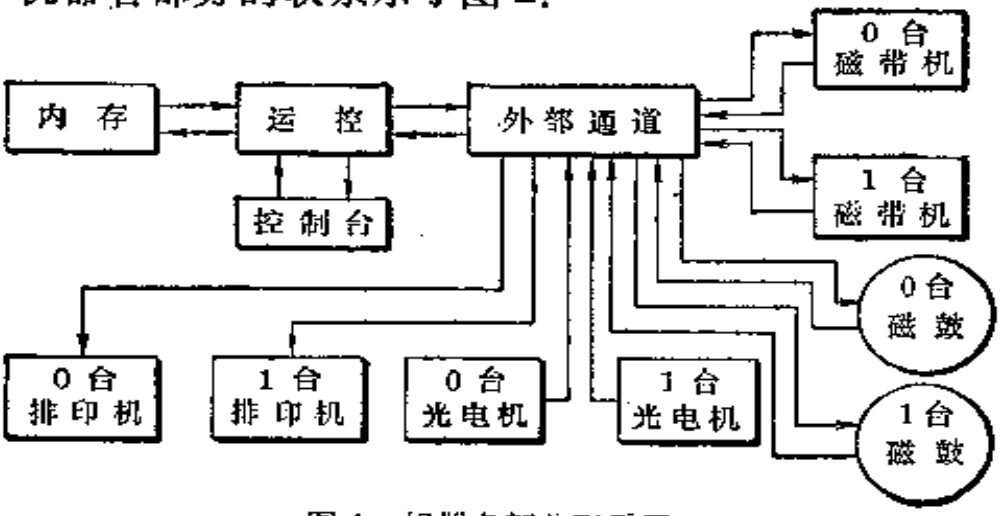


图1 机器各部分联系图

第二节 电子数字计算机上数的表示形式

电子数字计算机加工、运算的对象是表示为浮点形式的二进制数. 这种形式的数同我们日常接触的十进制数不一样，因此，要用计算机来解算生产实际问题，就要了解数在计算机里的各种表示形式，以及它们之间的转换方法.

一 十进制，二进制，八进制

我们日常生活中，用得最多的数是表示为十进制形式的

数，叫做十进制数．这种数的每一位数字都是0、1、2、3、4、5、6、7、8、9十个数字中的一个，而且，数中不同位置上的数字具有不同的含义．

例如，十进制数2.375表示为

$$2.375_{(十)}=2\times10^{0}+3\times10^{-1}+7\times10^{-2}+5\times10^{-3},$$

其中，自左至右的十进位数字2、3、7、5分别是10^{0}、10^{-1}、10^{-2}、10^{-3}数位上的数字．每一位上的数字满了十，就向高一位上进一，所以，十进制就是“逢十进一”的计数制．

除了十进制数之外，我们还经常碰到许多非十进制数．譬如，计算时间的时候，60秒钟作为1分钟，60分钟作为1小时，这就是六十进制．在使用电子数字计算机时，我们将更多地碰到二进制数和八进制数．

二进制数就是数的每一位数字都由0、1两个数字中的一个所构成，而且，数中不同位置上的数字具有不同的含义．

例如，二进制数10.011表示为

$$\begin{aligned}10.011_{(二)}&=1\times2^{1}+0\times2^{0}+0\times2^{-1}+1\times2^{-2}+1\times2^{-3}\\&=2.375_{(十)},\end{aligned}$$

其中，自左至右的二进位数字1、0、0、1、1分别是2^{1}、2^{0}、2^{-1}、2^{-2}、2^{-3}数位上的数字．每一位上的数字满了二，就向高一位上进一，所以，二进制就是“逢二进一”的计数制．

从本例看出，二进制中的10.011就等于十进制中的2.375．

由于二进制数的每一位数字只有0或1，形式很简单，运算也方便，所以，大多数计算机都采用二进制作为数的表示形式．当然，这种二进制形式的数使用时也有不便之处，譬如，它的数位很长，书写起来很累赘，而且，它同习惯使用的十进制数关系也不明显等．为了弥补这些缺点，我们引进数的八

进制形式. 因为，用八进制形式表示一个数时，它所占的位数同用十进制表示时相差不大，而且，数的八进制形式同二进制形式有简明的换算关系.

八进制数就是数的每一位数字都是 0、1、2、3、4、5、6、7 八个数字中的一个，而且，数中不同位置上的数字具有不同的含义.

例如，八进制数 2.3 表示为

$$2.3_{(八)}=2\times8^0+3\times8^{-1}=2.375_{(十)},$$

其中，2 是 8^0 数位上的数字，3 是 8^{-1} 数位上的数字. 每一位上的数字满了八，就向高一位上进一，所以，八进制就是"逢八进一"的计数制.

从本例看出，八进制表示中的 2.3，等于十进制表示中的 2.375，也等于二进制表示中的 10.011.

二 数制的转换

(一)数的十进制形式同八进制形式的互化

任何一个数 a，在十进制里的表示形式与八进制里的表示形式是不一样的，但它们代表的数值大小是相同的，所以这两种数的表示形式之间有一定的转换关系.

先讲数的十进制形式转换成八进制形式的方法.

1. 整数的化法——"除八取余"法则.

[例] 把十进制整数 $75_{(十)}$ 转换成八进制形式的整数.

假设转换后的八进制整数是 $k_nk_{n-1}\cdots k_{0(八)}$，则应有

$$75_{(十)}=k_n\times8^n+k_{n-1}\times8^{n-1}+\cdots+k_0\times8^0.$$

我们从这个式子中把 k_n、k_{n-1}、$\cdots$、k_0 确定出来，为此，等式两端除以 8，得

$$9+3\times8^{-1}=k_n\times8^{n-1}+k_{n-1}\times8^{n-2}+\cdots+k_0\times8^{-1}.$$

比较等式两端得

$$k_0=3,$$

$$9=k_n\times 8^{n-1}+k_{n-1}\times 8^{n-2}+\cdots+k_1\times 8^0.$$

将第二式两端除以 8，得

$$1+1\times 8^{-1}=k_n\times 8^{n-2}+k_{n-1}\times 8^{n-3}+\cdots+k_1\times 8^{-1}.$$

再比较等式两端得

$$k_1=1,$$

$$1=k_n\times 8^{n-2}+k_{n-1}\times 8^{n-3}+\cdots+k_2\times 8^0.$$

又从第二式比较两端得

$$k_2=1.$$

从而

$$75_{(十)}=(k_2k_1k_0)_{(八)}=113_{(八)}.$$

2．小数的化法——“乘八取整”法则.

[例] 把十进制小数 $0.825_{(十)}$ 转换成八进制形式的小数.

假设转换后的八进制小数是 $0.k_1k_2\cdots k_n\cdots_{(八)}$，则应有

$$0.825_{(十)}=k_1\times 8^{-1}+k_2\times 8^{-2}+\cdots+k_n\times 8^{-n}+\cdots.$$

我们从这个式子中把 k_1、k_2、…、k_n、… 确定出来，为此，等式两端乘 8，得

$$6+0.6=k_1+k_2\times 8^{-1}+\cdots+k_n\times 8^{-n+1}+\cdots.$$

比较等式两端得

$$k_1=6,$$

$$0.6=k_2\times 8^{-1}+k_3\times 8^{-2}+\cdots+k_n\times 8^{-n+1}+\cdots.$$

第二式两端乘 8，得

$$4+0.8=k_2+k_3\times 8^{-1}+\cdots+k_n\times 8^{-n+2}+\cdots.$$

比较等式两端得

$$k_2=4,$$

$$0.8=k_3\times 8^{-1}+k_4\times 8^{-2}+\cdots+k_n\times 8^{-n+2}+\cdots.$$

第二式两端乘 8，得

$$6+0.4=k_3+k_4\times 8^{-1}+\cdots+k_n\times 8^{-n+3}+\cdots.$$

比较等式两端得

$$k_3=6,$$

$$0.4=k_4\times 8^{-1}+k_5\times 8^{-2}+\cdots+k_n\times 8^{-n+3}+\cdots$$

$$\cdots\cdots\cdots\cdots.$$

如此一直下去，可以求出 k_1，k_2，k_3，…．从而，

$$0.825_{(+)}=(0.k_1k_2k_3\cdots)_{(八)}=0.6463146314\cdots_{(八)}.$$

3．一般十进制数的化法．

先用上面两种方法，把一般十进制数的整数和小数分别转换成相应的八进制形式，再综合两部分的结果，就得到一般十进制数的八进制形式．

[例] 把十进制数 $75.825_{(+)}$ 转换成八进制形式．

先把十进制整数 $75_{(+)}$ 按“除八取余”法则转换成八进制形式的整数 $113_{(八)}$：

$$\begin{array}{r|l} 8 & 75 \\ \hline 8 & 9 \quad 3 \\ \hline & 1 \quad 1 \end{array}\uparrow$$

再把十进制小数 $0.825_{(+)}$ 按“乘八取整”法则转换成八进制形式的小数 $0.6463146314\cdots_{(八)}$：

$$\begin{array}{r} 0.825 \\ \times\quad 8 \\ \hline 6.600 \\ \times\quad 8 \\ \hline 4.800 \\ \times\quad 8 \\ \hline 6.400 \\ \vdots \end{array}\downarrow$$

最后，把两部分结果合并起来，即得

$$75.825_{(十)}=113.6463146314\cdots_{(八)}.$$

如果要把一个八进制形式的数转换成十进制形式，则可以根据八进制数中各位数字的含义直接得出.

[例] 把八进制数 $15.6_{(八)}$ 化为十进制形式，只要按公式直接计算即可：

$$15.6_{(八)}=1\times8^1+5\times8^0+6\times8^{-1}=13.75_{(十)}.$$

(二)数的八进制形式同二进制形式的互化

由于 $8=2^3$，所以，八进制数的一位可以用二进制的三位来表示：

八进制	0	1	2	3	4	5	6	7
二进制	000	001	010	011	100	101	110	111

由此，数的八进制形式化为二进制形式时，可以采用“一位拉三位”的方法.

[例] 八进制数 $113.6463_{(八)}$ 化为二进制形式时有

$$113.6463_{(八)}=001001011.110100110011_{(二)}.$$

反过来，数的二进制形式化为八进制形式时，应以小数点作为基准，往左、往右都是用“三位并一位”，不足三位用“0”补.

[例] 二进制数 $1110.011111_{(二)}$ 化为八进制形式时有

$$001110.011111_{(二)}=16.37_{(八)}.$$

三 数的浮点形式

上面提到的数的各种表示形式，有一个共同的特点，就是它们的小数点位置都是固定的. 例如，$75.825_{(十)}$ 中，小数点固定在数字 5 与 8 之间；再如，$110.011_{(二)}$ 中，小数点固定在左起第三、第四位数字中间；$75.32_{(八)}$ 中，小数点固定在左起第二、第三位数字中间等等. 这种小数点具有固定位置的数

叫做定点数.

另外,对于任何一个数来说,除了可以表示为定点形式外,还可以表示成浮点的形式,即表示成小数点是浮动的形式.

[例] 十进制数 $75.825_{(+)}$ 可以表示为

$$75.825_{(+)}=+10^{-2}\cdot 0.75825_{(+)}$$
$$=+10^{+3}\cdot 0.075825_{(+)}=\cdots.$$

[例] 二进制数 $110.011_{(二)}$ 可以表示为

$$110.011_{(二)}=+2^{+11}\cdot 0.110011_{(二)}$$
$$=+2^{+100}\cdot 0.0110011_{(二)}=\cdots.$$

[例] 八进制数 $75.32_{(八)}$ 可以表示为

$$75.32_{(八)}=+8^{+2}\cdot 0.7532_{(八)}$$
$$=+8^{+3}\cdot 0.07532_{(八)}=\cdots.$$

一般地,任何一个数 x 都可以表示为

$$x=\pm N^{\pm p}\cdot q,$$

这里, N——数制的底数.十进制数的底数 N 等于 10,二进制数的底数 N 等于 2,八进制数的底数 N 等于 8.

p——称为"阶数",它是 N 进制表示的正整数.

q——称为"尾数",它是 N 进制表示的正小数,即

$$0\leqslant q<1.$$

若数的浮点形式中,有 $\frac{1}{N}\leqslant q<1$,则称该浮点数为 N 进制浮点规格化数. 例如,二进制浮点数 $+2^{+101}\cdot 0.101001$ 是浮点规格化的数,而 $+2^{+110}\cdot 0.0101001$ 是二进制非规格化的浮点数.

四 计算机单元中数的表示形式

电子数字计算机要求把参与运算的数表示为二进制浮点

形式

$$x = \pm 2^{\pm p} \cdot q,$$

其中，阶数 p 是二进制正整数，尾数 q 是二进制正小数. 在709机和TQ-16机上限定

$$-1000000 \leqslant \pm p < 1000000,$$

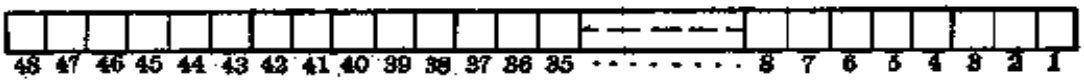
$$0 \leqslant q < 1.$$

且若 $q \geqslant \frac{1}{2}$，则称该数为二进制浮点规格化数.

电子数字计算机为什么要把数表示为二进制浮点形式呢？这是因为：

第一，二进制数在计算机单元里容易表示.

如前所述，计算机存放数据的单元是由一串互相联系着的小元件构成的. 每一个单元有48个小元件，即



单元的每一个小元件都具有两种不同的稳定状态(如晶体管的饱和或截止). 如果规定一种状态代表1，另一种状态代表0，那么，每一个小元件就可以表示一个二进位数字，因而，在确定正负号位和小数点位置后，一个单元中小元件任何一种稳定状态的组合，就代表着一个有限位长度的二进制数. 譬如，在一个具有48个小元件的单元中，若把第48位定为符号位(用0代表正，1代表负)，小数点位置固定在第47、48位之间，这样，这个单元就能把任何一个具有47个二进位数字的小于1的定点数表示出来.

第二，浮点形式可以扩大数的表示范围. 如果单元中表示的是定点数，则数的表示范围就会受到限制，例如上面的例子就只能把二进制的小数表示出来. 为了扩大数在单元中的

表示范围，可以采用小数点不固定的浮点形式.

在 709 机和 TQ-16 计算机单元中，二进制数规定表示为如下的浮点形式：

$$x_{(二)}=\pm 2^{\pm p_1p_2p_3p_4p_5p_6}\cdot 0.q_1q_2\cdots q_{39}q_{40},$$

其中，阶数 $p_1p_2p_3p_4p_5p_6$ 是二进制正整数，尾数 $0.q_1q_2\cdots q_{39}q_{40}$ 是二进制正小数，且

$$-1000000\leqslant\pm p_1p_2p_3p_4p_5p_6<1000000,$$

$$0\leqslant 0.q_1q_2\cdots q_{39}q_{40}<1.$$

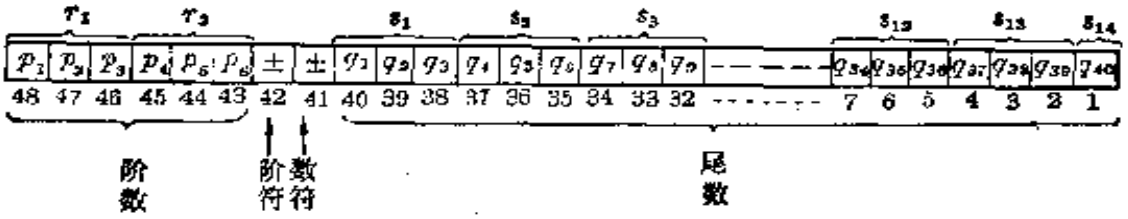
这个数如果用八进制浮点形式表示，则为

$$x_{(八)}=\pm 2^{\pm r_1r_2}\cdot 0.s_1s_2\cdots s_{13}s_{14},$$

其中，阶数 r_1r_2 是八进制正整数，尾数 $0.s_1s_2\cdots s_{13}s_{14}$ 是八进制正小数，且

$$-100\leqslant\pm r_1r_2<100,\ 0\leqslant 0.s_1s_2\cdots s_{13}s_{14}<1,\ s_{14}\leqslant 1,$$

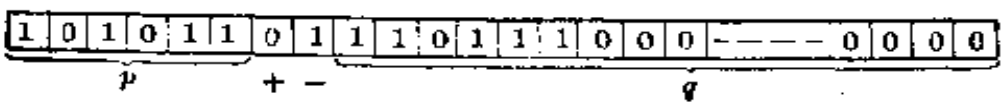
二进制浮点数在计算机单元中的形式是



其中，阶符和数符规定用 0 表示正，用 1 表示负，另外还规定，当阶符为负时，阶数要用 $1000000-|p_1p_2p_3p_4p_5p_6|$ 来代替.

现在举几个例子.

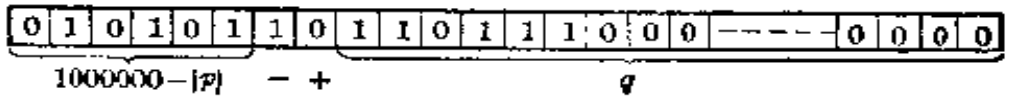
[例] 二进制数 $x_{(二)}=-2^{+101011}\cdot 0.11011100\cdots 0$ 在计算机单元中表示为



写成八进制形式，即

$$x_{(八)} = -2^{+53} \cdot 0.67000000\cdots0.$$

[例] 二进制数 $x_{(二)} = +2^{-101011} \cdot 0.11011100\cdots0$ 在计算机单元中表示为



写成八进制形式，即

$$x_{(八)} = +2^{-53} \cdot 0.67000\cdots0.$$

由于计算机单元的字长是有限的，因而，它能表示的二进制浮点规格化数的数值范围是：

$$2^{-63} \cdot \frac{1}{2} \leqslant |x| \leqslant 2^{+63} \cdot (1-2^{-40}).$$

这一范围在十进制计数制中，大约等于 $10^{\pm19}$.

对于绝对值小于 $2^{-68} \cdot 1/2$ 的数，计算机将表示为机器零：$2^{-64} \cdot 0$. 若在运算过程中，出现大于机器所能表示的最大的数，则计算机将产生溢出停机.

最后，我们用一个例子写出把一个十进制数化成二进制浮点规格化形式并置入单元的步骤.

[例] 把 $-53.14_{(十)}$ 化成二进制浮点规格化形式后置入单元.

1. 把 $-53.14_{(十)}$ 化成八进制形式.

先用"除八取余"法则把 $53_{(十)}$ 化成八进制整数 $65_{(八)}$：

$$\begin{array}{r|l} 8 & 5\ 3 \\ \hline & 6\ \ 5 \end{array} \uparrow$$

再用"乘八取整"法则把 $0.14_{(十)}$ 化成八进制小数 $0.107\cdots_{(八)}$：

$$
\begin{array}{r}
0.14 \\
\times\quad 8 \\
\hline
\boxed{1}.12 \\
\times\quad 8 \\
\hline
\boxed{0}.96 \\
\times\quad 8 \\
\hline
\boxed{7}.68 \\
\times\quad 8 \\
\hline
\vdots
\end{array}
$$

合并上面两项结果，得 $-53.14_{(十)}$ 相应的八进制形式

$$-65.107\cdots_{(八)}.$$

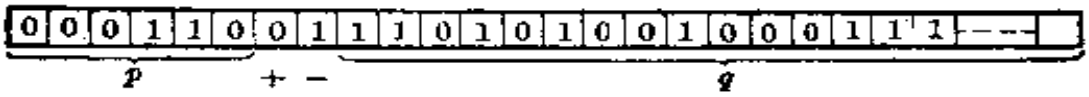
2．用“一位拉三位”方法把八进制数 $-65.107\cdots_{(八)}$ 转换为二进制形式：

$$-65.107\cdots_{(八)}=-110101.001000111\cdots_{(二)}.$$

3．把二进制数 $-110101.001000111\cdots_{(二)}$ 改写成浮点规格化形式，并置入单元．

$$-110101.001000111\cdots_{(二)}$$
$$=-2^{+000110}\cdot 0.110101001000111\cdots.$$

从而，在计算机单元中表示成



五　数的二-十进制形式

如前所述，程序运算的对象是表示为浮点形式的二进制数，这个规定对用惯十进制数的人来说是十分不便的，为了避免直接同二进制浮点数打交道，所以引进数的二-十进制浮点形式．

数的二-十进制形式，就是十进制数在计算机单元中的表

示形式，它把十进制数中的每一个数字都用二进制数字来表示．由于十进制数字同二进制数字的关系是

十 进 制	0	1	2	3	4	5	6	7	8	9
二 进 制	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

因此，任何一个十进制数，譬如 $319_{(+)}$，就可以写出它的二-十进制形式：001100011001．再如，$0.408_{(+)}$ 的二-十进制形式是 0.010000001000．

一般地，对任何一个十进制浮点规格化数

$$x_{(+)} = \pm 10^{\pm p} \cdot q,$$

它可写为

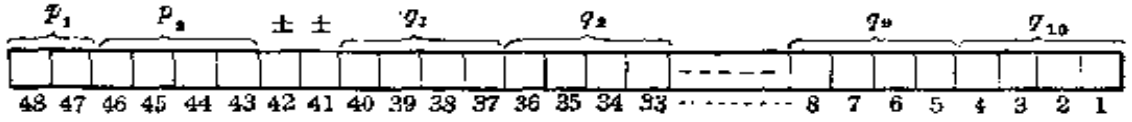
$$x_{(+)} = \pm 10^{\pm p_1 p_2} \cdot 0.q_1 q_2 \cdots q_9 q_{10},$$

或

$$p_1 p_2 \pm \pm q_1 q_2 \cdots q_9 q_{10},$$

这里，规定取10位有效数字，并限定 $p_1 p_2 < 19$，p_1、p_2、q_i($i=1, 2, \cdots, 10$)都是十进制数字．

从而，相应的二-十进制浮点形式规定这样写出：p_1 用两个二进制数位表示，p_2 和 q_i($i=1, 2, \cdots, 10$)都用四个二进制数位表示，阶符和数符都用 0 表示正、1 表示负．这个数在计算机单元中表示成



引入数的二-十进制浮点规格化形式之后，任何一个需要输入的十进制浮点数就可以用它来代替．这种二-十进制浮点规格化数被输进机器之后，机器里的〈10→2〉专用程序就自动把它转换成二进制浮点规格化形式，从而，输入数也就可以

作为工作程序的运算对象了.

现在,举几个例子.

[例] 把十进制数 $-61.129_{(+)}$ 表示成二-十进制浮点规格化形式.

因为

$$-61.129_{(+)}=-10^{+02}\cdot 0.61129,$$

即

$$0201611290\cdots 0.$$

它在计算机单元中可表示为



[例] 把十进制数 $+0.0001234567898_{(+)}$ 表示成二-十进制浮点规格化形式,并写出它在计算机单元中的形式.

因为

$$+0.0001234567898=+10^{-03}\cdot 0.1234567898,$$

即

$$03101234567898.$$

所以,它在计算机单元中的形式是



第三节 电子数字计算机解算生产实际问题的全过程

电子数字计算机解算生产实际中的计算问题,大致要经过下面几个步骤.

一 构造数学模型,选择计算方法

一个生产过程、一条基本运动规律、一种技术设计活动等

等，它们总是可以先经过分析、试验之后，用一系列数学算式来描述，这种对实际问题进行的数学描述，称为数学模型．举个简单例子来说，设要计算半径为 r_1，r_2，r_3，…，r_{100} 的一百块圆铁板的面积之和 S．这一问题的数学模型可以描述为

$$S=\pi\sum_{i=1}^{100}r_i^2.$$

构造了数学模型之后，就得选择运算简单、工作量节省，并能保证精度要求的计算方法．目前，数值分析学科的研究成果，已经为各种数学问题的解算提供了大量行之有效的方法(对于常用的一些计算方法，将在第三章中给出它们的标准程序)．

本例的计算方法，可以简单地描述如下：

先计算半径 $r_i(i=1, 2, \cdots, 100)$ 平方值之和 S．这可先使 $0\Rightarrow S$，然后逐个把 r_i 的平方值 r_i^2 累加到 S 中去，即

$$S+r_1^2\Rightarrow S,\ S+r_2^2\Rightarrow S,\ \cdots,\ S+r_{100}^2\Rightarrow S.$$

最后，计算

$$\pi\cdot S\Rightarrow S,$$

则 S 就是所求的结果．

二　计算过程的程序设计

计算方法中的计算公式、原始数据和计算步骤，构成计算问题的完整计算过程．为了让计算机来实现这个过程，还得把它们编写成程序，这个工作常常分两步进行．

(一)框图设计

用圆框、方框、指向线及文字说明描述计算过程的图式叫框图，框图设计有助于进行正确的程序设计．当然，对于简单的计算过程，这一步是可以省略的．上面例子的计算过程可以用图 2 的框图描述．

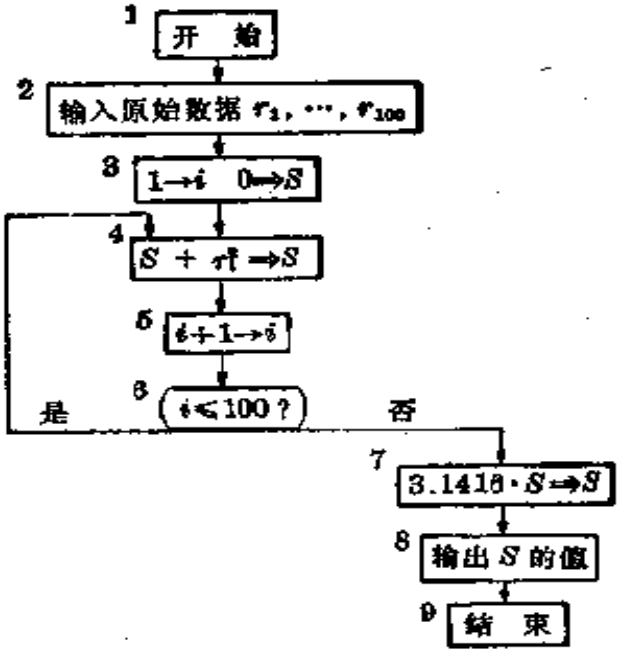


图 2

框 1　表示程序的开始.

框 2　输入原始数据 $r_i(i=1, 2, \cdots, 100)$.

框 3　循环参数 i 置 1, 变量 S 置 0.

框 4　把 r_i 的平方值累加到变量 S 中去.

框 5　修改循环参数 i.

框 6　循环参数 i 与循环终值 100 比较, 若 $i \leqslant 100$, 则转到框 4.

框 7　计算 $S=\pi \cdot \sum_{i=1}^{100} r_i^2$.

框 8　输出 S 的值.

框 9　表示程序的结束.

(二)程序设计

用算法语言把计算过程描述出来的工作, 称为程序设计. 参照上面框图设计出来的程序是

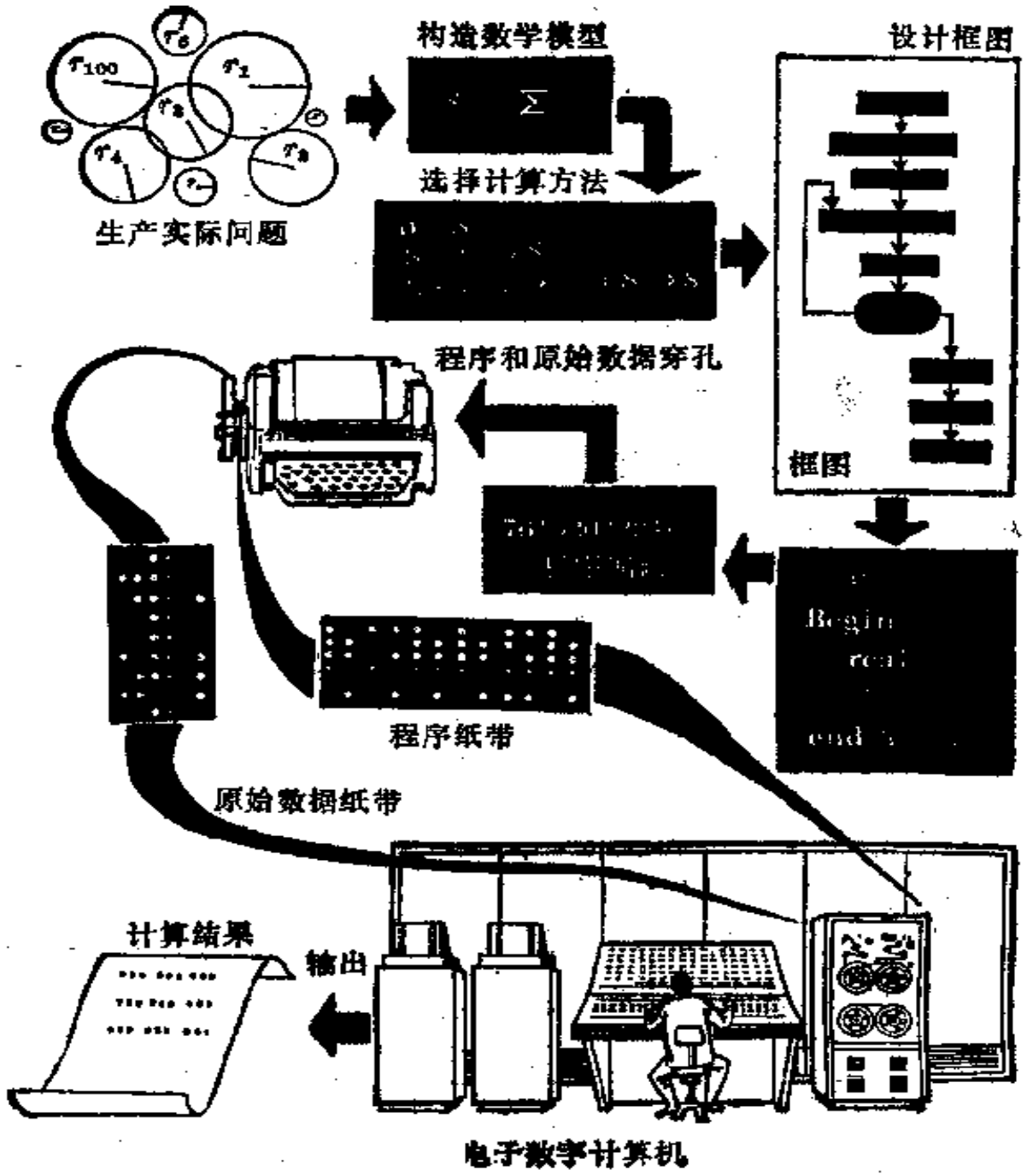

构造数学模型
设计框图
选择计算方法
生产实际问题
程序和原始数据穿孔
框图
程序纸带
原始数据纸带
计算结果
输出
电子数字计算机

图 3

```
begin
  real S;
  integer i;
  array r[1:100];
  #read(0, '10', r);
    S:=0;
  for i:=1 step 1 until 100 do
    S:=S+r[i]×r[i];
  S:=3.1416×S;
  #print (0, '10', S)
end 55…5
```

三　程序的信息化及其在计算机上的执行

用算法语言描述一个完整计算过程的程序通常称为源程序．为了把源程序输进计算机，必须把它信息化，即按照一定的编码系统把源程序编成数码，并穿孔在纸带上．代表源程序的纸带被输进机器以后，机器里的编译系统就对它进行语法检查，再编排并翻译成用机器语言表示的目标程序，最后，计算机执行目标程序并得出计算结果．

如果计算过程包含原始数据，则数据也应该进行信息化，即把它们穿成数据纸带．

源程序和原始数据信息化的具体方法详见第四章第一节．

综上所述，计算机解算生产实际问题的全过程，可由图 3 来说明．

习 题 一

1. 将下列十进制数转换为八进制浮点规格化形式(按单元字长截取尾数):

(1) 4096;　　(2) 0.1;

(3) 3.1415926;　　(4) −5.78;

(5) −245.326;　　(6) 0.0031;

(7) −0.00358;　　(8) 1000.

2. 将下列十进制数写成浮点规格化形式:

(1) 78.543;　　(2) 3000.001;

(3) 0.1×10^{-17};　　(4) -5.38×10^{9}

(5) −0.000153;　　(6) 78.7878;

(7) 400;　　(8) −205.

第 二 章

程序的编写方法

第一章第三节中已经讲到，任何一个要用电子数字计算机进行计算的问题，都必须预先编写好它的计算程序，本章将详细介绍采用算法语言编写程序的方法。

第一节　程序的初步认识

什么是计算问题的程序？我们从一个简单的例子入手，介绍它的基本内容和结构。

[例]　已知 x, a 的值，计算 $y=\frac{x^2}{a+3}$ 的值。

首先，我们应该说明，x, a, y 都是表示在实数范围内取值的变量，写成

real x, a, y.

然后，将变量 x, a 的十进制浮点形式的值，输入到机器内存贮器里代表变量 x、a 的单元中去，写成

$\#$read(0, '10', x, a).

变量 x, a 的给定值输进机器后，接着就按计算公式进行运算，写成

$y:=x\times x/(a+3)$.

最后，将计算得到的结果 y 按十进制浮点形式打印在输

出纸上，写成

#print(0, '10', y).

以上就是采用算法语言对这个计算问题进行的全部描述，如果把它们写在一起并添加某些符号，就构成了计算问题的源程序：

```
begin
  real x, a, y;                    ……说明部分
    #read(0, '10', x, a);
    y:=x×x/(a+3);                  ……语句部分
    #print(0, '10', y)
end 55…5
```

对这个简单例子的源程序作初步分析，可以得出如下几点认识：

1．整个源程序是由英文字母、阿拉伯数字以及数学记号等基本符号构成；

2．源程序的基本结构大致有如下形式：

begin

对变量作出说明的部分；

描述计算过程的部分；

end 55…5

其中，对变量作出说明的部分称为程序的说明部分；描述计算过程的部分称为程序的语句部分。

3．源程序的说明部分和语句部分须置于 **begin** 和 **end 55…5**（三十个以上）之中，它们分别作为程序开始和结束的信号。

第二节　算法语言的基本成分

一　基本符号

（一）二十六个英文字母（计算机对大、小写字母是不加区

别的，例如，A 和 a 代表同一符号）：

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；

（二）十个阿拉伯数字：

0, 1, 2, 3, 4, 5, 6, 7, 8, 9；

（三）两个逻辑值：

true([truɪ]真)，　**false**([fɔɪls]假)；

（四）四十六个定义符号：

1．二十三个运算符，包括

六个算术运算符：

＋(加)，－(减)，×(乘)，/(除)，÷(整除)，↑(乘幂)；

三个逻辑运算符：

∨(逻辑加)，∧(逻辑乘)，¬(逻辑非)；

六个关系运算符：

＜(小于)，≤(小于等于)，＝(等于)，≥(大于等于)，
＞(大于)，≠(不等于)；

八个顺序运算符：

goto([goutuɪ]转向)，　**if**([if]如果)，
then([ðen]那末)，　**else**([els]否则)，
for([fɔɪ]对于)，　**do**([duɪ]执行)，
stop([stop]停机)，　**pause**([pɔɪz]暂停)；

2．八个括号：

(　(圆开)，　)(圆闭)，

圆开、圆闭构成一对圆括号，

[　(方开)，　](方闭)，

方开、方闭构成一对方括号，

‘　(引号)，

两个引号构成一对行括号，

begin([bi'gin]开始)， **end**([end]结束)

构成一对语句括号，

ε([epsilon])，

两个 ε 构成一对字符括号。

3．八个说明符：

real([riel]实型)，**integer**(['intidʒə]整型)，**boolean**(['bu:lin]布尔型)；**array**([ə'rei]数组)，**switch**([switʃ]开关)，**procedure**([prə'siidʒə]过程)，**ku**([ku]库过程)，**fe**([fei]翻译)；

4．八个分隔符：

10(小拾)，·(点号)，:(冒号)，,(逗号)，;(分号)，井(井号)，**step**([step]步长)，**until**([ən'til]直到)；

5．一个分类符；

value(['vælju:]值)。

二 标识符，常量，变量，标准函数

(一)标识符

标识符是给变量、数组、开关、过程以及标号取名用的符号，它是由英文字母或英文字母与数字组成的符号序列，但序列的开头符号必须是英文字母。

标识符的例子如下：

a, *a*4, Max, beta, *aij* 等。

但是下面的符号序列不能看作标识符：

C-1 (出现有英文字母和数字以外的符号)，

2*a* (不是以英文字母开头)。

标识符中符号的个数可以是任意的，但是在机器上规定

限于前八个符号有区分意义，例如 *ymaxaijkl* 与 *ymaxaijkm* 只是第九个符号不同，因此这两个标识符在机器上被认为是相同的.

程序中需要使用的标识符可以按照规则随意选定，但是不能用同一个标识符表示两个不同的量(除非它们不在同一个作用范围内).

(二)常量

程序中的常量是指常数值和逻辑值.

常数值的书写形式同通常的十进制数的书写形式没有多大的差别，例如，0.53, −28.105, 3000, +4.3, 0, 10.0 等. 除此之外，算法语言还允许把常数值写成指数形式，即带阶的浮点形式. 例如 $_{10}3$, $_{10}-4$, $-5.6_{10}+4$, $3.1_{10}2$ 等. 但是必须指出，小拾符号 $_{10}$ 和它后面的整常数值组成一个整体，它表示以 10 为底的整数次方.

再举几个以浮点形式书写的常数值例子：

$$-0.05_{10}3 \quad (=-0.05\times10^{3}=-50),$$

$$+1.2_{10}+4 \quad (=1.2\times10^{4}=12000),$$

$$3_{10}-2 \quad (=3\times10^{-2}=0.03).$$

逻辑值只有两个，即 **true**(真)，**false**(假). 它们用来表示逻辑运算公式的计算结果.

例如，设 $a>0$ 是一个逻辑运算公式，若给出 $a=2$，则逻辑运算公式 $a>0$ 的计算结果为逻辑值 **true**；若给出 $a=-3$，则计算结果为逻辑值 **false**.

(三)变量

如果一个量的值是不确定的，则称这个量为变量.

例如，$b+3.1416\times c$，其中 3.1416 是常量，b，c 是变量.

变量有两种形式：简单变量和下标变量.

所谓简单变量，就是指它的值取一个数或一个逻辑值的单个变量. 我们用一个标识符表示一个简单变量，这个标识符称为简单变量标识符. 例如，计算公式 $s=20\times b$ 中，标识符 s, b 所代表的变量都是简单变量，所以它们都是简单变量标识符.

所谓下标变量，是指由一组数或一组逻辑值中各个分量所代表的变量，它具有形式

$$A[i_1,\ i_2,\ \cdots,\ i_n],$$

其中，A 代表全体分量的总名称，是一个标识符，叫数组标识符；$i_1,\ i_2,\ \cdots,\ i_n$ 称为下标变量的下标. 规定下标变量的下标写在方括号之中，并用逗号分隔开.

下标变量中下标的个数称为数组的维数. 具有一个下标的下标变量，相应的数组称为一维数组，具有 n 个下标的下标变量，相应的数组称为 n 维数组.

例如，若用数组标识符 x 表示一个向量的分量 $x_1, x_2, \cdots, x_n$，则各个分量都是一个下标变量，分别写成：$x[1], x[2], \cdots, x[n]$. 显然，x 是一个一维数组.

再如，若取数组标识符 M 表示矩阵

$$M=\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1q} \\ m_{21} & m_{22} & \cdots & m_{2q} \\ \vdots & \vdots & \cdots & \vdots \\ m_{p1} & m_{p2} & \cdots & m_{pq} \end{bmatrix},$$

则矩阵的每一个元素就是一个下标变量，分别写为 $M[1,\ 1]$, $M[1,\ 2], \cdots, M[p,\ q]$. 显然，$M$ 是一个二维数组.

下标变量的下标可以是常数、简单变量、下标变量或算术表达式，但它们都限于取整数值，若不为整数，则机器自动把小数部分删除.

(四)标准函数

在计算公式中，常常会遇到一些初等函数的计算，例如，$\sin x+\cos y+0.1-\ln m$. 当用算法语言描述这个计算公式时，允许把这些初等函数直接写进去，但在书写时，除了写出该初等函数的专用标识符外，还规定，在初等函数符号的前面要写一个#号. 另外，初等函数的自变量要用圆括号括起来.

例如，上面的式子可以写成

$$\#\sin(x)+\#\cos(y)+0.1-\#\ln(m).$$

机器中允许使用下列二十二个标准函数：

标准函数	数学含义
#abs(E)	E 的绝对值，
#entier(E)	不大于 E 的最大整数
#sign(E)	E 的符号，即 $\#\mathrm{sign}(E)=\begin{cases}1 & \text{若 } E>0\\ 0 & \text{若 } E=0\\ -1 & \text{若 } E<0\end{cases}$
#sin(E)	E 的正弦值；$\lvert E\rvert\leqslant\pi\cdot 2^{40}/4$
#cos(E)	E 的余弦值，$\lvert E\rvert\leqslant\pi\cdot 2^{40}/4$
#arc sin(E)	E 的反正弦值，$\lvert E\rvert\leqslant 1$
#arc tg(E)	E 的反正切值
#ln(E)	E 的自然对数，$E>0$
#exp(E)	E 的指数函数，$E<63\cdot\ln 2$
#sqrt(E)	E 的平方根，$E\geqslant 0$
#cubrt(E)	E 的立方根
#ranum(E)	产生一个[0，1]中均匀分布的伪随机数，这里 E 只有形式上的作用，可任取一值

（续表）

标准函数	数学含义
#tg(E)	E 的正切值，$\|E\| \leqslant \pi \cdot 2^{40}/4$
#axp(E, F)	E 的 F 次方值，其中要求： $E>0$，$-64\ln 2 < F\ln E < 63\ln 2$
#ranump(E)	它将 #ranum(E) 的伪随机数置成初态，然后转一下 #ranum(E) 取得伪随机数，其中 E 只有形式上的作用，可任取一值
#wg(E)	把弧度值 E 转换成如下形式的度、分、秒值（称为度、分、秒的十进制数表示形式）： $\pm a_m a_{m-1}\cdots a_1 \cdot b_1 b_2 \cdots b_n \quad (a_m \neq 0)$ 它表示 $\pm a_m a_{m-1}\cdots a_1^\circ \quad \pm b_1 b_2'$ $\pm b_3 b_4 \cdot b_5 \cdots b_n''$
#gw(x)	把十进制数表示形式的度、分、秒值 $x = \pm a_m a_{m-1}\cdots a_1 \cdot b_1 b_2 \cdots b_n$ 转换成弧度值
#bo(E, F)	#bo(E, F)$=$ 负数　若 E 值的第[$\|F\|$]位上为 1 正数(>0)　若 E 值的第[$\|F\|$]位为 0
#ew(E, F)	E 值左移（当 $F\geqslant 0$）或右移（当 $F<0$）[$\|F\|$]位的结果
#A(E, F)	#A(E, F)$=E$ 值 $\theta_1(1, 1, \cdots, 1, f_1, 1, \cdots, 1)$
#v(E, F)	#v(E, F)$=E$ 值 $\theta_2(0, 0, \cdots, 0, f_1, 0, \cdots, 0)$
#f(E, F)	#f(E, F)$=E$ 值 $\theta_3(0, 0, \cdots, 0, f_1, 0, \cdots, 0)$

注意：

(1) 标准函数的自变量 E、F 可以是常数、变量或算术表达式. 记号[$|F|$]表示 F 的绝对值的整数部分.

(2) 计算三角函数时，自变量均以弧度为单位.

(3) #A(E, F)、#v(E, F)、#f(E, F)三个标准函数的值都是 E 的二进制代码值同表中特定形式的二进制代码

值执行按位代码运算所得的结果. 特定形式的二进制代码中 f_1 表示第[|F|]位(限于[|F|]≤48)的二进制代码,

$$f_1=\begin{cases}1, & \text{当 } F<0,\\ 0, & \text{当 } F\geqslant 0.\end{cases}$$

二进制代码运算的规则是

x_i	y_i	$x_i\ \theta_1\ y_i$	$x_i\ \theta_2\ y_i$	$x_i\ \theta_3\ y_i$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

三 表 达 式

表达式是由常量、变量(简单变量或下标变量)、函数(包括标准函数)、圆括号及运算符所组成的, 它代表一个有数学含义的式子. 表达式有算术表达式和逻辑表达式两种.

算术表达式是对常数、变量和函数进行算术运算的式子. 算术表达式运算的结果是一个数值.

算术表达式中允许使用的运算符有

+(加)、-(减)、×(乘)、/(除)、÷(整除)、↑(乘幂).

算术表达式的例子:

算术表达式	数学含义	算术表达式	数学含义
3.1416	3.1416	((i+j)/k)↑2	$\left(\frac{i+j}{k}\right)^2$
a	a	c+(f+a)/(x+2)	$c+\frac{f+a}{x+2}$
7+m[i]↑3	$7+m_i^3$		

如果两个算术表达式用关系运算符连结起来，就构成一个关系式．若设 E_1、E_2 为算术表达式，则 $E_1<E_2$，$E_1\leqslant E_2$，$E_1=E_2$，$E_1\geqslant E_2$，$E_1>E_2$，$E_1\neq E_2$ 都是关系式．关系式的值是一个逻辑值．例如 $a\times b>0$ 对于具有相同符号的任意 a、b 值都得逻辑值 **true**，对于具有相异符号的任意 a、b 值都得逻辑值 **false**．

逻辑表达式，或叫布尔表达式，是对逻辑值、变量和关系式进行逻辑运算的式子．逻辑表达式的运算结果是一个逻辑值：**true** 或 **false**．

逻辑表达式中允许使用的逻辑运算符有

$\vee$（逻辑加）、$\wedge$（逻辑乘）、$\neg$（逻辑非）．

逻辑运算符的运算规则是

x	y	$x \vee y$	$x \wedge y$	$\neg x$
true	true	true	true	false
true	false	true	false	false
false	true	true	false	true
false	false	false	false	true

逻辑表达式的例子：

true，

u，

$a>0\vee(\neg b)\wedge d[i]+F>0$．

书写表达式时有几点应注意：

(1) 表达式中的每一个字符，必须写在同一条横线上．例如，$\frac{a}{b}+c$ 应写成 $a/b+c$．乘号 $\times$ 不能省写，也不能写为“$\cdot$”，例如，$2\times k-1$ 不能写成 $2k-1$ 或 $2\cdot k-1$．

(2) 表达式中允许象普通数学公式中那样使用圆括号，但不允许使用方括号.

(3) 机器在执行表达式时，是从左向右按如下优先级次序执行操作：

计算函数的值，

↑，

×，/，÷，

+，−

⩽，<，=，≠，>，⩾，

¬，

∧，

∨.

同一级的操作是从左做到右. 遇到圆括号时，机器优先算出左圆括号和对应的右圆括号之间的表达式的值.

例如，设 $a=36, b=2048, c=2, d=3, e=2, f=4, g=5, h=15$，则表达式

$$a+b/c\uparrow d\times e-(f\times g+h)$$

的执行次序和执行结果(分别以带圈的和不带圈的数表示)如图 4 所示.

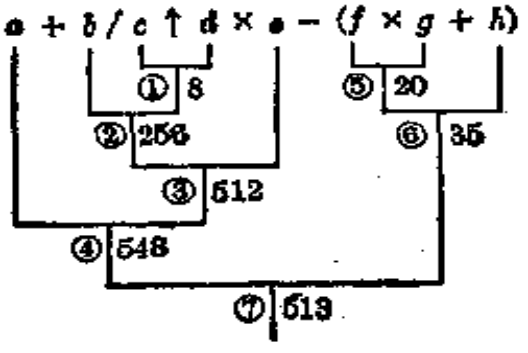


图 4

又如，逻辑表达式

$$u\geqslant 2\vee(L\neq 5\vee w\wedge z-4<0)\wedge(\neg a_1=0)$$

对于给定值：$u=2$, $a_1=0$, $L=5$, $w=$ **true**, $z=5$ 的执行次序和执行结果(分别用带圈和不带圈的数表示)如图 5 所示.

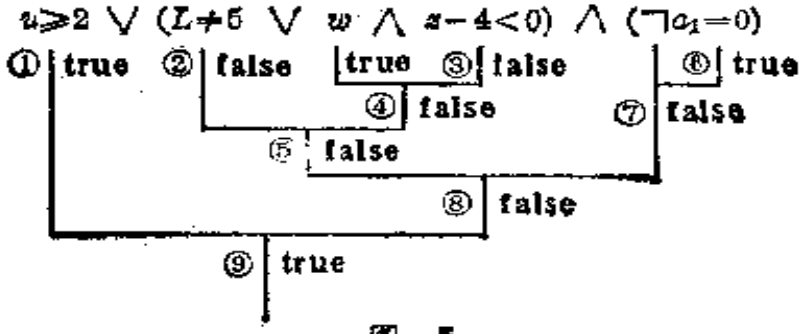


图 5

(4) 表达式中两个运算符不允许并置在一起，应该用圆括号把它们分开. 例如，$a/-b\uparrow -2$ 必须写成 $a/(-b)\uparrow(-2)$.

(5) 使用整除符号 $\div$ 时，譬如 $a\div b$，它的结果值由机器自动取为整数，即

$$a\div b=\#\mathrm{sign}(a/b)\times\#\mathrm{entier}(\#\mathrm{abs}(a/b)).$$

(6) 运算符号 $\uparrow$ 后面限于使用取整数值的量，例如 $a\uparrow 2$, $c[i]\uparrow 5$, $r\uparrow(-4)$, $d\uparrow i$(i 应为整数值，若出现非整数值，则机器自动取其整数部分)等.

表达式除上面的一种形式外，还有另外一种带条件的形式，称为条件算术表达式和条件逻辑表达式. 条件算术表达式的值由含在表达式中的逻辑条件的判断而确定出来. 相应地，前面介绍的不含判断条件的算术表达式就称为简单算术表达式.

条件算术表达式的例子：

$$\textbf{if}\ a>0\ \textbf{then}\ 3\ \textbf{else}\ x+5.$$

它表示 $a>0$ 时算术表达式的值为 3，否则(即 $a\leqslant 0$ 时)

算术表达式的值为 $x+5$.

又如:

if $c+d<0$ **then** #sin(x) **else** #cos(x).

它表示 $c+d<0$ 时算术表达式取值 $\sin x$, 否则算术表达式取值 $\cos x$.

条件算术表达式的一般形式为

if B **then** E_1 **else** E_2,

这里, B 为逻辑表达式, E_1 为简单算术表达式, E_2 为算术表达式. 由于 E_2 也可以为条件算术表达式, 所以, 条件算术表达式可以有多套的形式, 如

if B_1 **then** E_1 **else if** B_2 **then** E_{21} **else** E_{22},

其中, B_1, B_2 都是逻辑表达式, E_1, E_{21} 为简单算术表达式, E_{22} 为算术表达式.

类似地, 条件逻辑表达式的一般形式为

if B_1 **then** B_2 **else** B_3,

其中, B_2 为简单逻辑表达式, B_1, B_3 为逻辑表达式. 条件逻辑表达式也可以是多套式的, 这里就不详述了.

习 题 二

1. 下列符号是否标识符?

$3a$, $5mx$, m_{14}, $tb(3)$, $cy-1$, y_6,
sin, b[2], #sqrt, φ, xa, beta.

2. 在程序中是否允许出现下列形式的数?

10^2, $5.7_{10}-8$, $3_{10}3.6$, $-1.6_{10}0.9$.

3. 在程序中如何书写下列算式?

$\cos(\sqrt{a^2+b^2+c^2})$, $(\sqrt{a_{ij}})^{\cos b}$,

$x_i^3-\dfrac{\sin(a^5-b^5)}{7.2\cdot10^{-8}-c}$, $\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$,

$$(\ln\sqrt{x^2+y^3}-e^{\cos^3 x})^{5/2}.$$

4. 说明下列表达式的数学含义：

(1) ＃sin(alpha)×＃sin(beta)↑2
＋＃sqrt(alpha×alpha＋＃arctg(x[n]))；

(2) ＃exp(a＋b×＃ln(x[i]))。

5. 设 $a=2$, $b=2$, $c=1$, $d=0$，试求下列逻辑表达式的值：

(1) $a+b>3 \vee d>0 \wedge (c\neq 0 \vee c+d\times a\uparrow 2\geqslant 7)$；

(2) $a=2 \vee (\neg b>0) \wedge (a-b)/c/b\neq 0$。

6. 指出下列表达式中的错误并改正之：

(1) $-b+$sqrt$(b\times b-4\times a\times c)/2a$；

(2) $a(i)+$＃arcsin$x+$＃tg$(y)-u\times$＃lgβ；

(3) $[(x+y)\times$＃exp$(z)]\uparrow 2+1$；

(4) $\dfrac{a\cdot b-(\cos f)\uparrow -5}{k-h}$。

第三节　基本语句

算法语言中的基本语句是描述计算公式、实现程序转向以及进行数据交换等工作的独立成分。利用各种基本语句就可以构成程序的语句部分，把一个计算问题的完整计算过程描述出来。为了区分程序语句部分中各个独立的基本语句，语句之间必须用分号；来分隔。

基本语句有赋值语句、转向语句、条件语句、循环语句、停机语句、空语句、输入语句、输出语句、过程语句。另外，在709计算机上还增加了一个十翻二语句。

下面逐个介绍基本语句，其中过程语句在第五节中专门讨论。

一　赋值语句

在使用基本语句描述计算公式时，常常需要把一个计算

公式的计算结果赋值给一个或几个变量，这时，就要用到赋值语句.

赋值语句的一般形式是

$$A:=E.$$

其中，A 表示简单变量或下标变量；E 是表达式；$:=$ 是赋值号.

赋值语句的作用是把赋值号右端的表达式的值赋给左端的变量 A.

如果 E 是算术表达式，则 A 必须是取数值的变量；如果 E 是逻辑表达式，则 A 必须是取逻辑值的变量.

赋值语句的例子：

赋 值 语 句	含 义
$k:=3.1416$	常数值 3.1416 赋给变量 k
$x:=y$	变量 y 的值赋给变量 x
$M[j]:=\#\sin(v)+u[j]$	$\sin v+u_j$ 的值赋给下标变量 M_j
$i:=i+1$	变量 i 的值加 1，其结果作为 i 的新值
$u:=a>b$	u 取关系式 $a>b$ 代表的逻辑值， 即：若 $a>b$ 成立，u 取 **true** 若 $a>b$ 不成立，u 取 **false**
$be:=a=1\vee x>2$	be 取逻辑表达式 $a=1\vee x>2$ 代表的逻辑值

有时，表达式的值需要同时赋给几个变量，这只要把每一个变量标识符依次写在表达式的左边，其间用赋值号：＝联系起来即可. 这种形式的赋值语句叫多重赋值语句.

例如

$$A:=B[j]:=c:=i+5.$$

它表示，表达式 $i+5$ 的值依次赋给变量 c, B, 和 A.

在赋值语句的一般形式中，若 E 是条件算术表达式（或条件逻辑表达式），则称这种赋值语句为条件赋值语句.

例如，在图 6 所示的曲线中，x 所对应的函数值由下式计算：

$$y=\begin{cases}0, & \text{当 } |x|\geqslant R,\\ \sqrt{R^2-x^2}, & \text{当 } |x|<R.\end{cases}$$

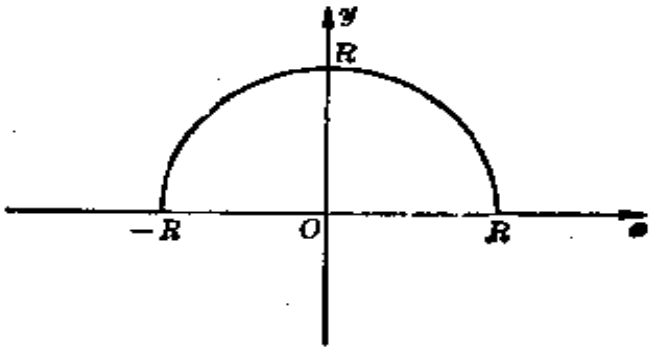


图 6

使用赋值语句时可写为

$y:=$ **if** #abs(x)$\geqslant R$ **then** 0 **else** #sqrt($R\times R-x\times x$).

二 转向语句

对任何一个完整程序的语句部分，计算机总是按照语句书写的自然次序逐个执行的. 如果要求从一个语句跳过某些语句，转向另一个语句去执行，这时就得使用转向语句.

为了提供转向语句的转入位置，程序中任何一个语句的前面可以加上一个标号. 标号也是一个标识符（当然它不能取得和其他标识符相同，除非它们不在同一个作用范围之内）. 标号同语句之间要用冒号来分隔，例如，

L: $y:=-a+b/0.25$;

start: #read(0, '10', H)

等.

转向语句有两种形式:

(1) **goto** L;

(2) **goto** $S[E]$.

第一种形式的转向语句实现转向具有标号 L 的那个语句继续运算. 例如,有一组语句:

$$a:=b+c;\quad d:=m\times \#\sin(u);\quad \cdots;$$

$$w:=y\uparrow 3-z\uparrow 3;\quad \cdots$$

如果要求机器在执行语句 $a:=b+c$ 后,跳过后面的一些语句而去执行语句 $w:=y\uparrow 3-z\uparrow 3$,这时,可先在语句 $w:=y\uparrow 3-z\uparrow 3$ 前面加上一个标号,例如 P_1,再在语句 $a:=b+c$ 后面写上第一种形式的转向语句 **goto** P_1 就可以了,即

$$a:=b+c;\quad \textbf{goto}\,P_1;\quad d:=m\times \#\sin(u);\quad \cdots;$$

$$P_1: w:=y\uparrow 3-z\uparrow 3;\quad \cdots.$$

第二种形式的转向语句,其转向目标 $S[E]$ 是一个开关变量,其中 E 可以是正整数、变量或算术表达式(它们的值必须是正整数,若不是正整数,则计算机自动取其整数部分),开关变量 $S[E]$ 的结构象一个一维的下标变量,但它与数组不一样,它的值不是一个数,而是一个标号,即 $S[E]$ 根据下标 E 的值可以确定出一个相应的标号来,E 值与标号的对应关系由程序中的说明部分用开关说明描述. 因此,第二种形式的转向语句必须与开关说明配套使用(详见第四节).

三 条件语句

在一些实际问题的计算公式中,有时需要根据某些条件的判别,才能决定执行什么运算,例如

$$y=\begin{cases}0, & \text{当 } x\leqslant a \text{ 时},\\ (x-a)^3, & \text{当 } x>a \text{ 时}.\end{cases}$$

这种计算公式可用条件语句描述为

if $x\leqslant a$ **then** $y:=0$ **else** $y:=(x-a)\uparrow 3$

或

if $x>a$ **then** $y:=(x-a)\uparrow 3$ **else** $y:=0$.

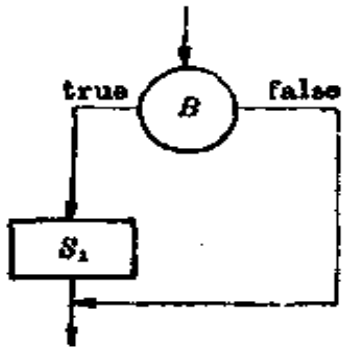
条件语句有如下两种形式：

(1) **if** B **then** S_1;

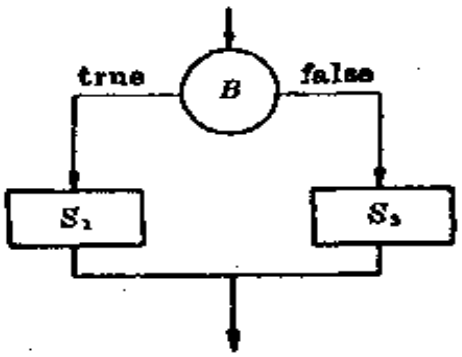
(2) **if** B **then** S_1 **else** S_2.

其中，B 是一个逻辑表达式；S_1 与 S_2 分别是一个语句，但规定 S_1 不能又是一个条件语句.

条件语句的执行效果是，当逻辑表达式 B 的值为 **true** 时，执行语句 S_1；若 B 的值为 **false**，则执行语句 S_2（第二种形式的条件语句）或不执行任何语句（第一种形式的条件语句）. 执行过程如图 7 所示.



第一种形式的条件语句



第二种形式的条件语句

图 7

[例 1] 计算 $y=\sqrt[4]{x}+\dfrac{\sin x}{\sqrt{z}}$ $(x\geqslant 0,\ z>0)$.

应用第一种形式的条件语句，上式的计算可写成

$$\text{if } x \geqslant 0 \wedge z > 0 \text{ then } y := \#\text{sqrt}(\#\text{sqrt}(x)) + \#\sin(x)/\#\text{sqrt}(z).$$

[例 2] 计算

$$y = \begin{cases} 1+x, & \text{当 } x > -1 \text{ 时,} \\ 2x, & \text{当 } x \leqslant -1 \text{ 时.} \end{cases}$$

用第二种形式的条件语句，上式的计算可写成

$$\text{if } x > -1 \text{ then } y := 1+x \text{ else } y := 2 \times x,$$

也可写成

$$\text{if } x \leqslant -1 \text{ then } y := 2 \times x \text{ else } y := 1+x.$$

[例 3] 计算

$$y = \begin{cases} 0, & \text{当 } x = z \text{ 时,} \\ z, & \text{当 } x < z \text{ 时,} \\ x, & \text{当 } x > z \text{ 时.} \end{cases}$$

用第二种形式的条件语句，上式的计算可写为

$$\text{if } x = z \text{ then } y := 0 \text{ else if } x < z \text{ then } y := z \text{ else } y := x.$$

在这个条件语句中，逻辑表达式 B 是 $x=z$，语句 S_1 是 $y:=0$，语句 S_2 又是一个条件语句：

$$\text{if } x < z \text{ then } y := z \text{ else } y := x.$$

上述计算公式还可写成其他形式，如

$$\text{if } x < z \text{ then } y := z \text{ else if } x > z \text{ then } y := x \text{ else } y := 0$$

等.

这种语句的一般形式是

$$\text{if } B_1 \text{ then } S_1 \text{ else if } B_2 \text{ then } S_2 \text{ else } S_3.$$

它是多套式的，其中 B_1，B_2 是逻辑表达式. 一般来说，当机器遇到多套式条件语句时，自左至右依次计算条件语句中逻辑表达式的值，直至找到 true 值的那个逻辑表达式为止，接着就执行紧跟在这个逻辑表达式后面的那个语句，原来多

套式条件语句的后面部分就不再分析；如果条件语句中所有的逻辑表达式的值都为 **false**，则执行最后一个 **else** 后面的语句(若没有 **else**，则执行本语句后面跟着的语句).

例如，对于下列语句：

if B_1 **then** S_1 **else if** B_2 **then** S_2 **else** S_3; S,

它的执行情况为：

若 B_1 为 **true**，则不管 B_2 的值，执行 S_1，再执行 S.

若 B_1 为 **false**，B_2 为 **true**，则执行 S_2，再执行 S.

若 B_1，B_2 均为 **false**，则执行 S_3，再执行 S.

用框图表示如图 8.

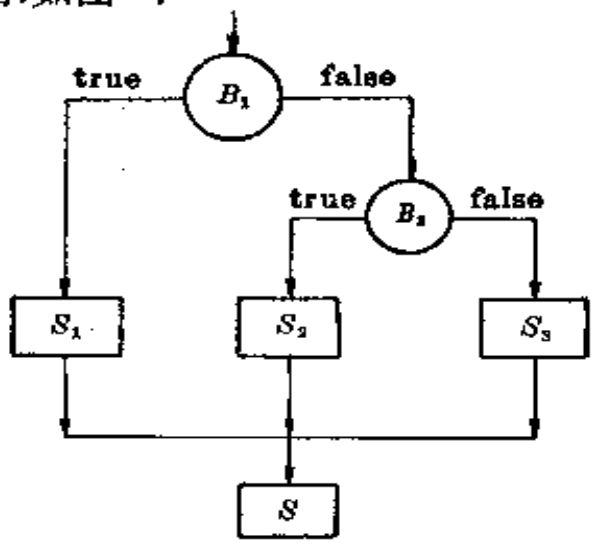


图 8

另外，在某些计算问题中，写出条件语句时，它的执行语句往往需要包含多于一个语句，这时，可以用语句括号 **begin end** 把它们括起来，将它们看成一个完整的新语句，称为复合语句，它的一般形式为

begin S_1; S_2; …; S_N **end**.

其中 S_1, S_2, …, S_N 都是语句，每个语句结束时均须加上分号，但 **end** 前的那个分号可以省略.

[例 4] 计算问题：

$$y=\begin{cases}1+x, & \text{当 } x\geqslant 0 \text{ 时},\\ 2x, & \text{当 } x<0 \text{ 时},\end{cases}$$

$$y_1=\begin{cases}x/3, & \text{当 } x\geqslant 0 \text{ 时},\\ 4-x, & \text{当 } x<0 \text{ 时}.\end{cases}$$

这个问题用第二种形式的条件语句可写为

if $x\geqslant 0$ **then begin** $y:=1+x$; $y_1:=x/3$ **end**

else begin $y:=2\times x$; $y_1:=4-x$ **end**.

这里，条件语句的执行语句 S_1 和 S_2 都是一个复合语句，它们分别为

begin $y:=1+x$; $y_1:=x/3$ **end**

和

begin $y:=2\times x$; $y_1:=4-x$ **end**.

四 循 环 语 句

循环语句是具有广泛应用的基本语句之一，程序借助于循环语句可以实现计算问题中包含的多次重复的计算. 现在我们用一个计算实例来说明循环语句的用法和意义.

[例] 柴油机的曲柄连杆机构如图 9 所示. 设曲轴以角

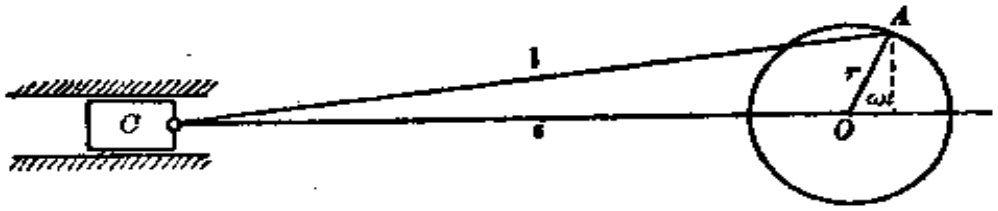


图 9

速度 ω 旋转，曲轴半径 $OA=r$，连杆长 $AC=l$，则活塞 C 的运动规律为

$$s=\sqrt{l^2-r^2\sin^2\omega t}-r\cos\omega t.$$

如果要计算当 $t=t_1,\ t_2,\ t_3$ 时相应的 s 值 $s_1,\ s_2,\ s_3$，我们可用下面三个赋值语句来表示：

```
s[1]:=#sqrt(l×l-r×r×#sin(omega×t[1])↑2)
        -r×#cos(omega×t[1]);
s[2]:=#sqrt(l×l-r×r×#sin(omega×t[2])↑2)
        -r×#cos(omega×t[2]);
s[3]:=#sqrt(l×l-r×r×#sin(omega×t[3])↑2)
        -r×#cos(omega×t[3]);
```

显然，这样的写法很不经济，因为这里把同一类型的计算过程重复书写了三次。可以设想，如果要对更多的 t 值来计算相应的 s 值，那将更是不胜其烦。为了克服这种书写上的繁复，我们引入循环语句，将上面三式统一写成

```
for i:=1, 2, 3 do
s[i]:=#sqrt(l×l-r×r×#sin(omega×t[i])↑2)
        -r×#cos(omega×t[i]).
```

这里新引入的变量 i 叫循环参数。“for i:=1, 2, 3 do”是“对 i 逐个取值 1, 2, 3 分别执行”的意思，do 后面的一个语句“$s[i]:=\cdots$”是循环语句的工作部分，叫循环体。整个循环语句执行的运算是，循环参数 i 逐个取值 1, 2, 3, 分别执行一次 do 后面的循环体。

显然，上面形式的循环语句的引入，简化了原来语句的书写形式。但是，如果要计算 $t=t_0,\ t_1,\ \cdots,\ t_{100}$ 所对应的 s 值，那末，即使这样写也是很麻烦的，仅“for i:=0, 1, 2, $\cdots$, 100 do”就得写上长长的几行。为此，还需要引进循环语句的另一种形式：

```
for i:=0 step 1 until 100 do
s[i]:=#sqrt(l×l-r×r×#sin(omega×t[i])↑2)
```

$$-r\times\#\cos(\text{omega}\times t[i]).$$

它表示，对循环参数 i 取值从 0 起每次增加 1 直到 100，分别执行一次 **do** 后面的循环体．这里的 0 称为循环参数的初态，1 称为步长，100 称为终态．

现在写出循环语句的一般形式：

(1) **for** $v:=E_1, E_2, \cdots, E_n$ **do** S;

(2) **for** $v:=E_1$ **step** dE **until** E_n **do** S.

其中，v 是循环参数，它是一个简单变量；S 是任一语句或分程序（见下节），称为循环语句的循环体；$E_1, E_2, \cdots, E_n$ 和 E_1 **step** dE **until** E_n 称为循环表，其中 $E_1, E_2, \cdots, E_n$ 和 E_1、dE、E_n 均可是常数值，也可以是变量或算术表达式．在第二种形式的循环语句中，E_1, dE, E_n 分别是循环参数的初态、步长、终态．

循环语句的作用是，它多次重复执行 **do** 后面的循环体 S，第一次执行时，$v=E_1$，第二次执行时，$v=E_1+dE$（第一种循环语句中是 $v=E_2$），第三次时，$v=E_1+2dE$（第一种循环语句中是 $v=E_3$），等等，直到满足条件

$$\#\text{sign}(dE)\times(v-E_n)>0$$

（第一种循环语句是对 $v=E_n$ 执行 S 以后）为止．

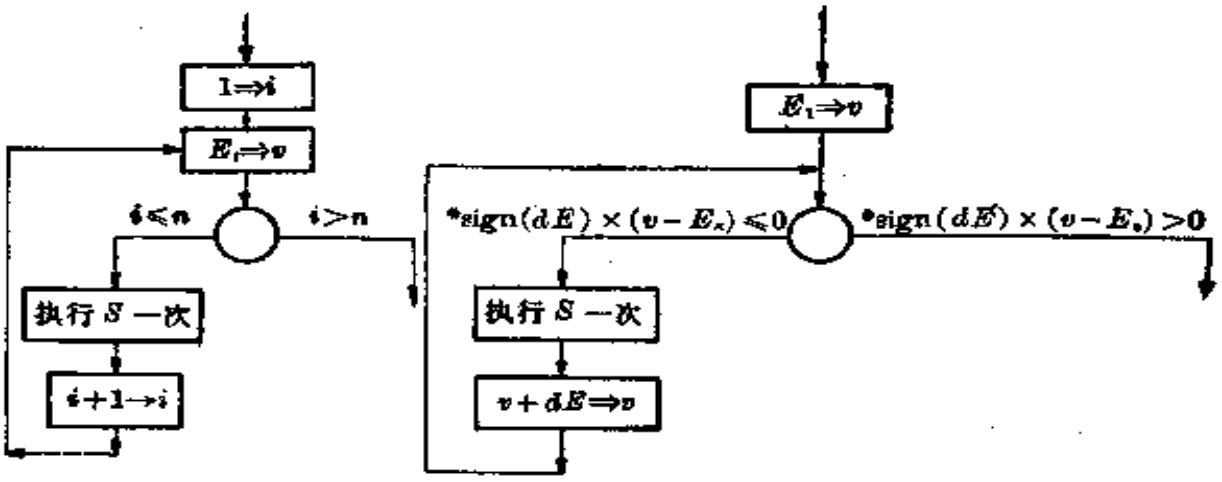
如果第二种循环语句开始执行时，就满足条件

$$\#\text{sign}(dE)\times(v-E_n)>0,$$

则循环体 S 将不被执行．

循环语句的执行过程如图 10 所示．由框图可以看出，循环语句总是可以用赋值语句、条件语句、转向语句的组合来代替，但前者书写时要简洁得多，所以应该尽可能地使用循环语句．

下面举几个例子．



第一种形式的循环语句　　第二种形式的循环语句

图 10

[例 1]　计算函数 $y=\sin\sqrt{x}$ 在 $x=0.1,\ 0.95,\ 7.24,\ 18$ 四个点上的函数值.

若把计算结果用数组标识符 y 来表示，则用第一种形式的循环语句可以把计算公式写为

```
i:=1;
for x:=0.1, 0.95, 7.24, 18 do
begin y[i]:=#sin(#sqrt(x)); i:=i+1 end.
```

如果把给定的 x_i 值也看作一个数组，用数组标识符 x 表示，则 x_i 中的每一个数就用一个下标变量表示，如 $x[1]$ 的值代表 0.1，$x[2]$ 的值代表 0.95，…等. 从而，此例也可用第二种形式的循环语句写成

```
for i:=1 step 1 until 4 do y[i]:=#sin(#sqrt(x[i])).
```

[例 2]　求 50 个数 $a_1,\ a_2,\ \cdots,\ a_{50}$ 的和.

由于在求和过程中，每一次都是前一次的结果加上新的一项，所以这是一个重复计算的过程，可用循环语句来表达. 为此取简单变量 S 表示诸数之和，则求和的计算过程可写成

$S:=0;$

for $i:=1$ **step** 1 **until** 50 **do** $S:=S+a[i]$.

这里，赋值语句"$S:=0$"不可省略，这就和我们用手摇计算机进行计算时必须先把存放结果的地方清洗为零一样.

[例 3] 已知曲线 L 在等距节点 $x=x_i$ 上的函数值 $y_i(i=1, 2, \cdots, 20)$，试用梯形法则求面积 S.

从图 11 看出，面积 S 是由各个小面积 A_i 组合而成的，即

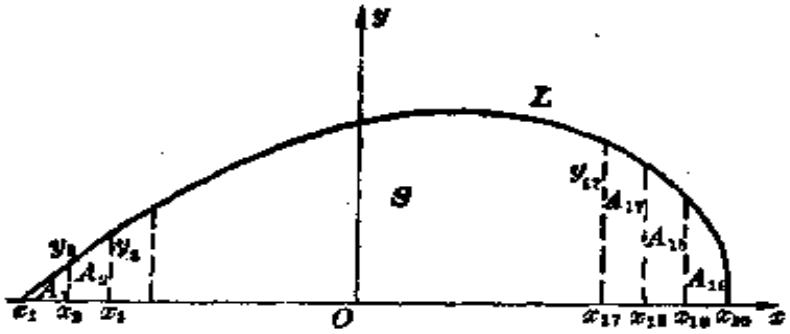


图 11

$$S=A_1+A_2+\cdots+A_{19}.$$

每个小面积 $A_i(i=1, 2, \cdots, 19)$ 按梯形公式计算，其值为

$$A_i=\frac{1}{2}(y_{i+1}+y_i)(x_{i+1}-x_i) \quad (i=1, 2, \cdots, 19).$$

这样，面积 S 的计算可用循环语句写成

$S:=0;$

for $i:=1$ **step** 1 **until** 19 **do**

$S:=S+(y[i+1]+y[i])\times(x[i+1]-x[i])/2.$

[例 4] 给定矩阵 $A=(a_{ij})$ 和向量

$B=(b_j) \quad (i=1, 2, \cdots, 10;\ j=1, 2, \cdots, 12)$,

求 $X=AB$，用语句写出计算过程.

根据矩阵和向量的乘法规则，有

$$x_i = a_{i1}b_1 + a_{i2}b_2 + \cdots + a_{i12}b_{12} \quad (i = 1, 2, \cdots, 10).$$

对每个固定的 i，这个计算过程可用循环语句写成

```
x[i]:=0;
for j:=1 step 1 until 12 do
x[i]:=a[i, j]×b[j]+x[i].
```

由于对不同的 i，所写出的语句形式是一致的，所以整个计算过程可以用两重循环的语句写为

```
for i:=1 step 1 until 10 do
  begin x[i]:=0;
    for j:=1 step 1 until 12 do
    x[i]:=a[i, j]×b[j]+x[i]
  end.
```

这里，语句

```
x[i]:=0; for j:=1 step 1 until 12 do
        x[i]:=a[i, j]×b[j]+x[i]
```

都受循环参数 i 控制，所以要用语句括号 **begin end** 把它们括起来，构成一个复合语句，作为 i 循环语句的循环体. 如果不加语句括号，那末关于参数 i 循环时，就只执行前一个语句 $x[i]:=0$，这样就不能完成所要求的运算.

为了正确使用循环语句，最后要指出下列几点：

(1) 循环语句中的执行语句 S 如果多于一个语句，必须用语句括号 **begin end** 把它们括起来，构成一个复合语句，这个复合语句就是循环语句的循环体.

(2) 同一个循环语句中，允许同时存在两种形式的循环表，这只要把这些循环表按所要求的执行次序排列在一起，并用逗号将它们彼此分开就可以了.

例如，计算函数 $y = x + x^5$ 在

$x=-18.4,\ -6.532,\ -3.1;$

$0,\ 1,\ 2,\ \cdots,\ 50;\ -100,\ -98,\ \cdots,\ -20$

上的函数值，就可用三张循环表排列在一起写成一个循环语句：

```
i:=1;
for x:=-18.4, -6.532, -3.1, 0 step 1 until 50,
        -100 step 2 until -20 do
begin y[i]:=x+x↑5; i:=i+1 end.
```

循环语句在执行过程中，依次对循环表执行循环运算，直到循环表用完为止.

(3) 允许在循环语句内部，对循环表中的$E_1, E_2, E_3, \cdots, E_n$和$dE$进行修改，例如

```
for i:=1 step m+n until m-n+10 do
    begin a[i]:=c×d[i]; n:=n+1 end.
```

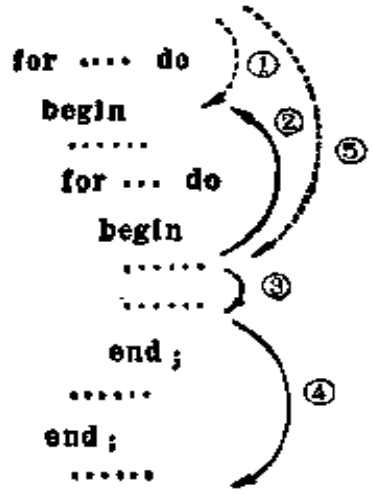
(4) 不允许使用转向语句从循环语句的循环体外转入循环语句的循环体内的任何语句；但是并不禁止用转向语句作相反方向的转移，即从循环语句的循环体内转到循环语句的循环体外.

例如，下面一段程序是不允许的：

```
…; goto P;
for i:=1 step 1 until 10 do
begin …; P:a[i]:=c×d[i]; … end.
```

反之，下面一段程序是允许的：

```
for i:=1 step 1 until 10 do
  begin …; goto P; … end;
  …
```



允　许：②，③，④

不允许：①，⑤

图　12

P: $k:=k+1$; ….

允许和不允许的转向方式见图 12.

五 停机语句

使程序的运算立即中止下来的语句叫停机语句. 它可以写在程序中任何需要停机的地方. 停机语句的使用, 可以使我们便于观察运算情况, 随时掌握程序工作的进程.

停机语句有两种形式:

(1) **stop**;

(2) **pause**.

第一种形式叫无条件停机语句. 一当它被执行时, 程序的运算就立即停止. 若再给出启动信息, 就从它后面的语句开始继续运算.

[例] 下面一段程序中,

…, $k:=h\times f$; **stop**; $h:=27\times l/p$; …

机器执行完赋值语句 $k:=h\times f$ 后就中止执行任何语句, 实现停机. 若给出启动信息, 机器就从赋值语句 $h:=27\times l/p$ 继续执行下去.

第二种形式的停机语句叫条件停机语句. 它在控制台上的"条件停机"扳键扳向上方时, 同无条件停机语句的作用一样, 执行停机; 否则, 它不执行停机任务而继续往下工作.

六 空 语 句

空语句就是不执行任何运算的语句, 它也没什么表示形式. 设置空语句的目的主要是为了便于使用转向语句.

[例] 已知 $y=f(x)$ 在 n 个点上的值 $y_k=f(x_k)(k=1, 2, \cdots, n)$, 它们不全为零, 试求出 y_k 中的最大绝对值 M, 并指

出相应的 x_k 的序号 k.

设用数组标识符 x、y 分别表示 x_k、y_k ($k=1, 2, \cdots, n$)，则计算过程可用如下语句写出：

```
M:=0;
for i:=1 step 1 until n do
    begin a:=#abs(y[i]);
      if a≤M then goto L;
      M:=a; k:=i;
  L:end.
```

这里，标号为 L 的语句是一个空语句.

七 输 入 语 句

各种计算语句的运算对象是存放在内存贮器里的代码或数，为了把它们从机外或外存贮器引入内存贮器，需要用到输入语句.

输入语句根据输入时所涉及的设备，须区分三种情形来书写.

1. 由光电机把数据纸带上的数据输入内存贮器变量单元时，输入语句的一般形式是

$$\#\mathrm{read}(0, \text{`}\Delta_1\text{'}, \Delta_2).$$

其中，Δ_2 指明输入数据应该依次存放到哪些变量中去，Δ_2 可以是一个或几个简单变量标识符或数组标识符，各变量标识符之间要用逗号分隔，而且，它们的书写次序必须同纸带上的数据次序相对应. Δ_1 指明数据输入的格式，709 机上安排了如下三种格式：

(1) 浮点十进制格式，简称 10 格式($\Delta_1=10$).

(2) 浮点八进制格式，简称 8 格式($\Delta_1=8$).

(3) 符号 4 格式，或称普通十进制格式($\Delta_1=10$，此时，0 的位置改写为 9).

TQ-16 机提供了如下五种格式：

(1)、(2) 同 709 机.

(3) 符号 6 格式，或称 N 格式($\Delta_1=N$).

(4) 符号 4 格式，或称普通十进制格式($\Delta_1=N$).

(5) 紧凑格式，或称 Zn 格式($\Delta_1=N$).

例如，设 a 是简单变量标识符，X 是数组标识符，则输入语句：

$$\#\text{read}(0,\ \text{`10'},\ a,\ X)$$

执行从光电机上把穿孔纸带的一个和随后的一组十进制浮点数输入到 a 和 X 所代表的变量单元中去.

2. 由外存贮器磁鼓中保存的代码或数，读到内存贮器变量单元时，输入语句的一般形式是

$$\#\text{read drum}(u,\ M,\ \Delta_2).$$

这种语句也叫读鼓语句，其中，u 是外存贮机磁鼓的台号；M 是用十进制计数的磁鼓中的单元地址，它允许是正整数或是简单变量(机器取其绝对值的整数值[$|M|$]作为单元的地址)；Δ_2 是内存贮器中的一串变量标识符(简单变量标识符或数组标识符)，两个标识符之间要用逗号分隔.

读鼓语句的作用是，它执行把外存贮器第 u 台磁鼓(u 值对 709 机没有意义；TQ-16 机限于取值 1、2)中开始地址为[$|M|$](709 机限于取值 0～43007；TQ-16 机限于取值 0～14335)的单元内容读入内存贮器，依次存放到 Δ_2 部分指明的变量单元中去. 这里，标识符所代表的变量个数将决定磁鼓内读出数据的个数.

例如，读鼓语句

#read drum(1, 150, a, b),

假定 a 为简单变量标识符，b 为包含 100 个下标变量的数组标识符，则机器执行这一语句时，就把 1 号磁鼓中从单元地址为 150 开始的 1+100=101 个数据读到内存贮器中变量 a 和 b 所代表的单元中去.

计算程序中的读鼓语句，往往是同后面将要讲到的写鼓语句联系起来使用的，因为磁鼓里保存的数据或代码是用写鼓语句写入的，因此，数据或代码从磁鼓中哪里读出，要根据写鼓语句曾经把数据或代码写到哪里来决定.

3．把外存贮器磁带中保存的代码或数，读到内存贮器时，输入语句的一般形式是：

#read tape(u, M, A_2)

这种语句也叫读带语句．目前，限于 TQ-16 机上使用．其中，u 是外存贮器磁带机的台号；M 是用十进制计数的磁带中的组号；A_2 是内存贮器中的数组标识符.

读带语句的作用是，它执行把外存贮器第 u 台磁带机(限于取值 0、1)第 M 组(限于取值 0～4000)单元的内容读入内存贮器，存放到 A_2 指明的数组中去．这里，数组标识符所代表的变量个数将决定磁带内读出数据的个数.

例如，读带语句

#read tape (0, 2, A)

假定数组标识符 A 包含 100 个下标变量，则机器执行这一语句时，就把 0 台磁带机中第 2 组单元内容(100 个数)，读入内存贮器中数组标识符 A 所代表的单元中去.

计算程序中的读带语句，往往是同后面将要讲到的写带语句联系起来使用的，因为，磁带里保存的数据或代码是用写带语句写入的，因此，数据或代码从磁带中哪里读出，要根据

写带语句曾经把数据或代码写在哪里来决定，而且规定，读带语句中的数组必须同写带语句中的数组具有相同的体积.

八 输 出 语 句

计算程序的中间或最终计算结果，是用输出语句把它们印在输出纸上的. 另外，利用输出语句还可以把内存贮器中变量的值写到外存贮器磁鼓或磁带中去.

同输入语句相仿，输出语句也要根据输出时所涉及的设备，区分三种情况来书写.

1. 把内存贮器里的数据，通过排印机打印在输出纸上时，输出语句的一般形式是

$$\#\text{print}(0, '\Delta_1', \Delta_2).$$

其中，Δ_1 指明输出数据取何种格式印在输出纸上，Δ_2 指明内存贮器中哪些变量的值需要输出，Δ_2 可以是一个或几个简单变量标识符或数组标识符，标识符之间须用逗号隔开，它们的书写次序决定了输出数据在输出纸上的印出次序.

输出语句中，常用的输出格式有下面十一种.

(1) 10 格式或叫十进制数格式.

Δ_1 的形式是

10.

10 格式的作用是，它把 Δ_2 部分的变量标识符的值按固定位数的十进制浮点形式印在输出纸上.

下面列出几个例子.

Δ_1	变 量 的 值	输 出 数 的 形 式
10	1234.56783	04++1234567830
10	−376.45611	03+−3764561100
10	0.0025	02−+2500000000

按 10 格式输出的十进制浮点数的一般形式是

$$p_1p_2\pm\pm q_1q_2q_3q_4q_5q_6q_7q_8q_9q_{10}$$
$$(=\pm 10^{\pm p_1p_2}\cdot 0.q_1q_2q_3q_4q_5q_6q_7q_8q_9q_{10})$$

其中，p_1，p_2，$q_i(i=1, 2, \cdots, 10)$ 都是十进制数字，左面一个土代表阶数的正负号，右面的一个土代表尾数的正负号.

(2) 8 格式或叫八进制数格式.

Δ_1 的形式是

8.

8 格式的作用是，它把 Δ_2 部分的变量标识符的值按固定位数的八进制浮点形式印在输出纸上.

下面列出几个例子.

Δ_2	变量的值	输出数的形式
8	0.1	75-+65146314631461
8	3.1416	02++62207771344441
8	−25	05+-62000000000000

按 8 格式输出的八进制浮点数的一般形式是

$$p_1p_2\pm\pm q_1q_2q_3q_4q_5q_6q_7q_8q_9q_{10}q_{11}q_{12}q_{13}q_{14}$$
$$(=\pm 2^{\pm p_1p_2}\cdot 0.q_1q_2q_3q_4q_5q_6q_7q_8q_9q_{10}q_{11}q_{12}q_{13}q_{14})$$

其中，p_1，p_2，$q_i(i=1, 2, \cdots, 14)$都是八进制数字，左面一个土代表阶数的正负号，右面一个土代表尾数的正负号.

(3) E 格式或叫十进制浮点格式.

它的一般形式是

$$Ew\cdot d.$$

其中，E 是十进制浮点格式的记号；w 是输出的每一个数所占用字符位置的个数；d 是输出数小数部分的十进制数字位数. 这里，必须满足 $w\geqslant d+7$.

E 格式的作用是，它把 Δ_3 部分对应变量标识符的值按如下十进制浮点形式印在输出纸上：

$$\underbrace{\underbrace{\text{␣}\cdots\text{␣}\pm 0.\underbrace{q_1q_2\cdots q_d}_{d\text{位}}10\pm p_1p_2}_{d+7\text{位}}}_{w\text{位}} \quad (=\pm 10^{\pm p_1p_2}\cdot 0.q_1q_2\cdots q_d)$$

下面列出几个例子.

Δ_1	变 量 的 值	输 出 数 的 形 式
$E15.8$	1234.567	␣0.12345670_{10}␣04
$E14.6$	−123.4567	␣−0.123456_{10}␣03
$E11.4$	−0.00825	−0.8250_{10}−02

这里，␣表示输出纸上空出一个字符的位置，以后不再说明.

E 格式的特点归纳为

(i) 输出数中阶数占两个字符，阶和数的正负号各占一个字符(空格代表正，"−"代表负)，阶的底数$_{10}$占一个字符，小数部分占 $d+2$ 个字符.

(ii) 输出数在 w 个字符位置中从右开始放起，在 $w>d+7$ 时，左边自动空出多余的字符位置(见例子第二行).

(iii) 当变量值有效数字的位数不足 d 位时，输出数自动用 0 在右方补足(见例子第三行)；多于 d 位时，输出 d 位，没有舍入(见例子第二行).

E 格式的优点是，输出数能够保证有效数字，适于在输出数数值范围难以估计的情况下采用. 其缺点是输出数的形式看起来不太方便.

(4) F 格式或叫十进制定点格式.

一般形式是

$$Fw\cdot d.$$

其中，F 是十进制定点格式的记号；w 是输出数总的占用字符个数；d 是输出数小数部分的十进制数字位数. 这里，必须满足 $w \geqslant d+2+l$，其中 l 是所有输出变量值中整数部分的最多位数. 当全部输出变量的绝对值均小于 1 时，l 为 1.

F 格式的作用是，它把 A_2 部分对应的变量标识符的值按如下十进制定点形式印在输出纸上：

$$\underbrace{\underbrace{\text{␣}\cdots\text{␣}\pm\underbrace{e_1e_2\cdots e_l}_{l\text{位}}\cdot\underbrace{q_1q_2\cdots q_d}_{d\text{位}}}_{d+l+2\text{位}}}_{w\text{位}}$$

下面列出几个例子.

A_1	变量的值	输出数的形式
$F15.8$	1234.567	␣␣1234.56700000
$F14.6$	−123.4567	␣␣␣−123.456700
$F11.4$	−0.00825	␣␣␣␣−0.0083

F 格式的特点归纳为

(i) 输出数的小数部分占 d 个字符，数的符号与小数点各占一个字符，整数部分占 l 个字符.

(ii) 输出数在 w 个字符位置中从右开始放起，在 $w>d+2+l$ 时，左边自动空出多余的字符位置(见例子)；而当 $w<d+2+l$ 时，整数部分位数不够，这时机器自动停机，若给出人工启动信息，机器就自动改用 E 格式输出该值.

(iii) 变量值有效数字的位数不足 d 位时，输出数自动用 0 在右方补足(见例子第一、二行)；多于 d 位时，输出 d 位，且执行第 $d+1$ 位向第 d 位的四舍五入(见例子第三行).

F 格式的优点是，输出数的形式同习惯使用的十进制形式一致，便于阅读. 其缺点是对所有输出变量的值，要有一个

预先的大致估计.

(5) I 格式或叫整数格式.

它的一般形式是

$$Iw.$$

其中，I 是整数格式的记号；w 是输出整数的总字符数(包括正负号).

整数格式的作用是，它把 A_2 部分对应的变量标识符的值按如下整数形式印在输出纸上：

$$\underbrace{\pm e_1 e_2 \cdots e_{w-1}}_{w\text{位}}$$

下面列出几个例子.

A_1	变 量 的 值	输 出 数 的 形 式
$I8$	1234	␣␣␣␣1234
$I3$	−37	−37
$I1$	−37	−37
$I5$	−123.45	␣−123

整数格式的特点归纳为

(i) 输出数总是占有 w 个字符：一个符号位和 $w-1$ 个数字位.

(ii) 如果变量的值不足 w 位，则左边填空格(见例子第一、四行)；如果多于 w 位，这时机器自动停机，若给出人工启动信息，机器就自动用 I 格式按实际位数输出该数(见例子第三行).

(iii) 具有非整数值的变量，允许使用整数格式输出其整数部分.

(6) H 格式或叫静态字符格式.

它的一般形式是

$$wHa_1a_2\cdots a_w.$$

其中，H 是静态字符格式的记号；w 是输出字符的总个数；$a_1a_2\cdots a_w$ 是被印出的 w 个字符(它们不能包含逗号，和引号 ')。

H 格式主要是用来打印标题的，它的作用是把记号 H 后面写出的 w 个字符印在输出纸上。显然，对 H 格式并不需要 Δ_2 部分提供变量标识符。

例

Δ_1	输出形式
8*H*Shanghai	SHANGHAI
7*H*Max(y_1)	MAX(*Y1*)

(7) A 格式或叫动态字符格式。

它的一般形式是

$$rAw.$$

其中，A 是动态字符格式的记号；w 是输出字符的总个数；r 是重复次数，它可以对应 Δ_2 部分的数组标识符。

A 格式的作用是，它把 Δ_2 部分对应变量标识符的单元代码所代表的字符自左至右印出 w 个(六个二进位数码，即两个八进位数码表示一个字符，对应关系见第四章图 26)。

例

Δ_1	变量单元的代码形式	输出字符形式
5*A*	100101000001111001010100111010…	$e_1(t)$
5*A*	000000001010000010100111001100…	0.2−*A*
3*A*	001001001001000001…	991

A 格式和 H 格式不同之处在于，它可将动态变化的变量以字符方式印出。

(8) x 格式或叫空格格式.

它的一般形式是

$$wx.$$

其中，x 是空格格式的记号；w 是空出字符位置的个数.

x 格式的作用是，它在输出纸上自左至右连续空出 w 个字符的位置. 它一般同其他格式(除 10 格式、8 格式外)联合使用，作为分隔输出纸上几种输出数据或字符之用. 它也不需要 A_3 部分提供变量标识符.

[例] 设 $a=-1234.5678$, $b=-0.034$. 则输出语句

#print(0, '$F14.6$, $4x$, $E9.2$', a, b)

输出形式为

␣␣−1234.567800 ␣␣␣␣ −0.34$_{10}$−01

(四个空格)

(9) /格式或叫换行格式.

它的一般形式是

$$w/.$$

其中，/是换行格式的记号；w 是换行的次数.

换行格式的作用是，它在输出纸上从上到下连续换行 w 次. 它也不需要 A_3 部分提供变量标识符.

[例] 如果 a, b 取上例中的值，则

#print(0, '$F14.6$, $3/$, $F14.6$', a, b)

输出形式为

␣␣− 1 2 3 4.5 6 7 8 0 0␣……␣

␣␣␣……␣ } 二行空格

␣␣␣……␣

␣␣␣␣␣␣−0.0 3 4 0 0 0……␣

(10) S 格式或叫连印格式.

它的一般形式是

$$wSa.$$

其中，S 是连印格式的记号；w 是字符连印个数；a 是连印 w 次的字符(a 不能是逗号和分号).

S 格式也不需要 A_2 部分提供变量标识符，它的作用就是把 a 位置上的字符在输出纸上连印 w 个. 例如，输出语句

#print(0, '7SK')

的输出形式是

$$\underbrace{KKKKKKK}_{7\text{个}K}$$

在使用以上十种输出格式时，尚须注意以下几点：

1) 同一个输出语句中，8、10 格式只能单独使用，其他各种格式可以混合使用. 每一个输出格式必须同变量标识符一一对应(不要求变量的输出格式除外).

2) 输出语句在印出每一个变量标识符所代表的值时，总是从左开始印起. 若使用 8 格式，则每行印出三个八进制浮点数；若使用 10 格式，则每行印出四个十进制浮点数. 一行印不完的变量值，自动换行从左端印起. 输出纸上每行规定限印 80 个字符，因此，在使用格式输出时，应预先设计好印出式样. 下面是一个设计不当的例子.

变量的值	输出语句形式	输出形式
$a=-1420.53$	#print(0, '73x, 2$Ha=$, $F8.2$, 'a)	73位 ␣␣…␣ $a=-1420$ $.53$

3) 输出语句的 A_2 部分，若有连续 r 个变量标识符使用相同要求的输出格式(指 E、F、I 及 A 格式)，如

#print(0, '$F14.6$, $F14.6$, $I2$, $I2$, $I2$', a, b, c, d, e), 则允许把它简写成

#print(0, '$2F14.6$, $3I2$', a, b, c, d, e).

其中，输出格式前面的正整数 2 和 3，称为该格式的重复次数，通常用记号 r 表示. 对于 8、10 格式，则无需写出重复次数 r，就能达到多个变量标识符的输出.

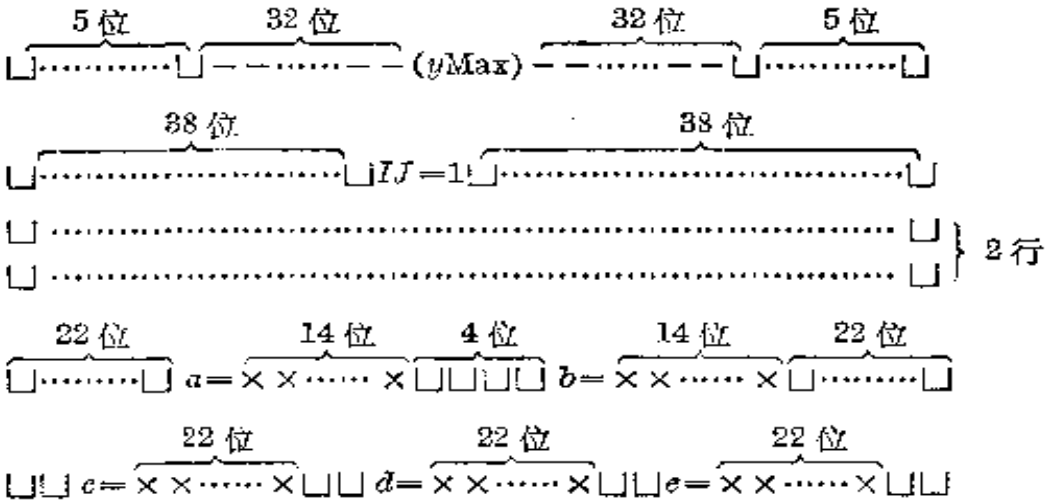
4) 在空格格式 wx、换行格式 $w/$、字符格式 wH、动态字符格式 rAw 中，当 $w=1$ 时，这个 1 可以省略.

现在举一个输出语句的例子.

[例] 输出语句

#print(0, '$5x$, $32S$-, $6H(y\max)$, $32S$-, $43x$,
$4HIJ=1$, $3/$, $22x$, $2Ha=$, $F14.6$, $4x$, $2Hb=$,
$F14.6$, $24x$, $2Hc=$, $E22.8$, $2x$, $2Hd=$,
$E22.8$, $2x$, $2He=$, $E22.8$', a, b, c, d, e);

在输出纸上印出的形式是



(11) L 格式，或者叫竖印格式，它的一般形式是

$$Lr_1G_1w_1\cdot d_1;\ \cdots\cdots;\ r_kG_kw_k\cdot d_k.$$

其中，L 是竖印格式的记号；$G_iw_i\cdot d_i(i=1, 2, \cdots, k)$称为 L

格式的子格式，这里 G_i 可以是 E、F、I、A 格式的任何一种；w_i，d_i 为格式 G_i 所要求的字符位数（当 G_i 为 I、A 格式时就只有 w_i）；r_i 为相应子格式的重复次数。

L 格式一般是用来输出几组数据并设计成竖式表格时采用的，它的输出形式可描述如下：

将各子格式在 A_2 部分所对应的变量内容，在输出纸上自左至右各印成竖的一条。每条所占的字符宽度为各自对应的 w_i 值。各条的第一个数印在同一横行上，变量少的竖条印完后往下自动留出空白，直到变量最多的竖条印完为止。

[例] 假设标识符 a、b、c、d 分别是具有 5、3、4、2 个下标变量的数组，它们的值是

$$a[1]=3,\ a[2]=4,\ a[3]=5,\ a[4]=6,\ a[5]=7,$$
$$b[1]=25.34,\ b[2]=18.26,\ b[3]=3.05,$$
$$c[1]=-0.0052,\ c[2]=-0.041,\ c[3]=0.2,$$
$$c[4]=8,\ d[1]=-0.24,\ d[2]=73.28.$$

输出语句

#print(0, '$LF20.2$; $I8$; $E20.1$; $F20.2$', b, a, c, d)

的输出形式是

15位　　7位　12位　　15位　　12位 $=80-\overline{w}$ 位

␣…␣25.34␣…␣3␣…␣−0.5$_{10}$−02␣…␣−0.24␣…␣

␣…␣18.26␣…␣4␣…␣−0.4$_{10}$−01␣…␣7.3␣28␣…␣

␣…␣␣3.05␣…␣5␣…␣␣0.2$_{10}$␣00␣…␣␣␣…␣

␣……␣6␣…␣␣0.8$_{10}$␣01␣……␣

␣……␣7␣……␣

在使用 L 格式时，其中子格式 $G_i(i=1, 2, \cdots, k)$ 所对应的变量标识符个数应满足

$$\sum_{i=1}^{k} r_i \leqslant 8.$$

另外，一般情况下还应满足

$$\bar{w}=\sum_{i=1}^{k} r_i w_i = 80.$$

但当 $\bar{w}\neq 80$ 时，在下列两种情形下，机器能自行调整输出形式，使之保持整齐.

若 $\bar{w}>80$，但 $\bar{w}-80<w_k$，即最右一竖条的数据不能排入一行时，则将它排在次一行的最右方(左方留出空白)(见图 13).

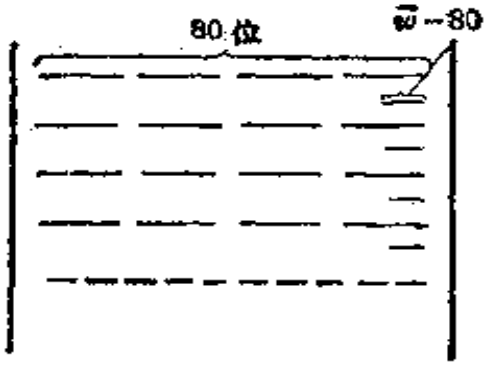


图 13

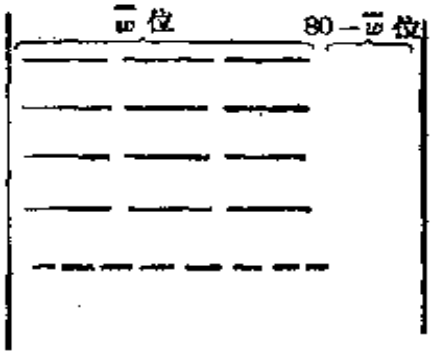


图 14

若 $\bar{w}<80$，但 $80-\bar{w}<w_1$，则输出形式相当于把打印纸当作右边空出 $80-\bar{w}$ 个字符位置来使用(见图 14).

2. 把内存贮器变量单元里的代码或数，写到外存贮器磁鼓里，输出语句的一般形式为

$$\#\,\mathrm{write\,drum}(u,\ M,\ A_2).$$

它也称为写鼓语句.

其中，u 是外存贮器磁鼓的编号；M 是用十进制计数的磁鼓单元地址，它允许是正整数或是简单变量(机器取其绝对值的整数值 $[|M|]$ 作为其单元地址)；A_2 是内存贮器中的

变量标识符(简单变量标识符或数组标识符)，两个标识符之间要用逗号分隔开.

写鼓语句的作用是，它执行把 Δ_2 部分指明的内存贮器变量标识符所代表的值，写到外存贮器 u 号磁鼓 (u 值对 709 机没有意义；TQ-16 机限于取值 1、2) 中开始地址为 [|M|] (709 机限于取值 0～43007；TQ-16 机限于取值 0～14335)的区域中去.

[例] 写鼓语句

#write drum(2, 1000, a, b)

执行把内存贮器中变量标识符 a, b 所代表的值，依次写到 2 号磁鼓中地址从 1000 号开始的单元中去.

3. 把内存贮器里的代码或数，写到外存贮器磁带时，输出语句的一般形式是

#write tape(u, M, Δ_2).

这种语句也叫写带语句. 目前，限于 TQ-16 机上使用. 其中，u 是外存贮器磁带机的台号；M 是用十进制计数的磁带中的组号，它可以是正整数，或是简单变量(机器取其绝对值的整数值作为组号)；Δ_2 是内存贮器中的数组标识符.

写带语句的作用是，它执行把 Δ_2 部分指明的内存贮器数组标识符所代表的值，写到外存贮器第 u 台磁带机(限于取值 0、1)第 M 组(限于取值 0～4000)单元中去.

例如，写带语句

#write tape(0, 2, A)

表示将数组 A 的内容写入 0 台磁带机，并注明是 2 组.

在使用写带语句时必须注意的是，磁带是以组分段的，每组可大可小，包含若干单元，写带的组号无须连续，但必须从

小到大，可以使用的组号是 0～4000.

九 十翻二语句

在 709 机上为了把十进控制台变量的值从二-十进制浮点形式转换成二进制浮点形式，所以增加了一个十翻二语句，其一般形式是

$$\text{fe}\ H0000i,$$

其中，$H0000i(i=0、1、2、3)$ 是十进控制台变量（关于十进控制台变量的用法见第四章第二节）.

最后，举两个例子说明怎样用基本语句的组合来描述计算过程.

[例 1] 在齿轮设计中，经常会遇到一种叫做渐开线的曲线(图 15). 从单位圆上坐标为(1, 0)点出发的渐开线方程是

$$\begin{cases} x=\cos t+t\sin t, \\ y=\sin t-t\cos t. \end{cases}$$

试在半径为 4 的圆内，对 t 从 0 开始每隔 0.1 求曲线坐标值

$$(x, y).$$

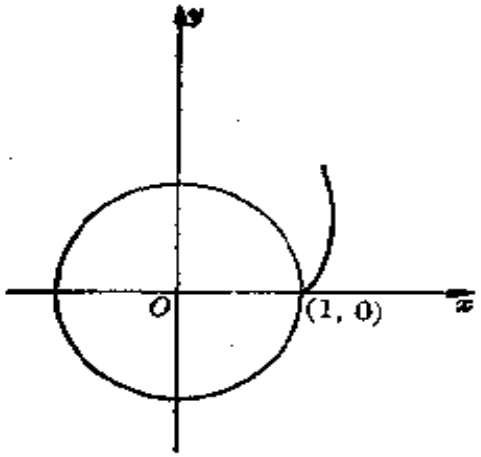


图 15

从 $t=0$ 开始，0.1 为步长，计算一次渐开线方程，即可求出相应的 x，y 值，这是一个循环计算的过程，但是，循环变量 t 的终值取什么？

问题给出了一个循环是否结束的判定条件，即

$$x^2+y^2<4^2.$$

利用这个条件，计算过程可用框图描述如图 16.

把框图用基本语句写出来就是

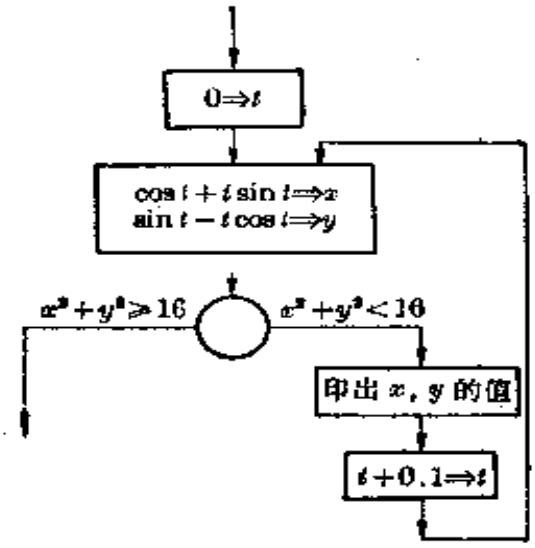


图 16

```
t: =0;
L1: x: =#cos(t)+t×#sin(t);
    y: =#sin(t)-t×#cos(t);
  if x×x+y×y<16 then
    begin #print(0, '2F14.6', x, y);
      t: =t+0.1; goto L1 end;
```

[例 2] 迭代公式

$$\begin{cases} x^{(n+1)}=x^{(n)}-\dfrac{x^{(n)}\cdot x^{(n)}-a}{2x^{(n)}}, \\ x^{(0)}=1, \end{cases}$$

$n=0, 1, \cdots$, 直到条件 $|x^{(n+1)}-x^{(n)}|<\varepsilon$ 为止，是求 a 的平方根值的计算过程. 现给定 $a=1, 2, \cdots, 100$, $\varepsilon=10^{-4}$, 试求出它们的平方根值.

可以看出，对于给定的一个 a 值，迭代公式对迭代次数来说，是一个循环计算过程，不过，作为循环结束的判定条件不是循环的次数，而是变量 x 前后两次迭代值的接近程度，即判定条件 $|x^{(n+1)}-x^{(n)}|<\varepsilon$ 是否满足.

另外，对变量 a 来说，整个计算过程将是又一个循环过程，循环变量 a 的初值、步长、终值分别是 1、1、100，执行语句 S 是迭代公式.

由此，计算问题可用框图描述为

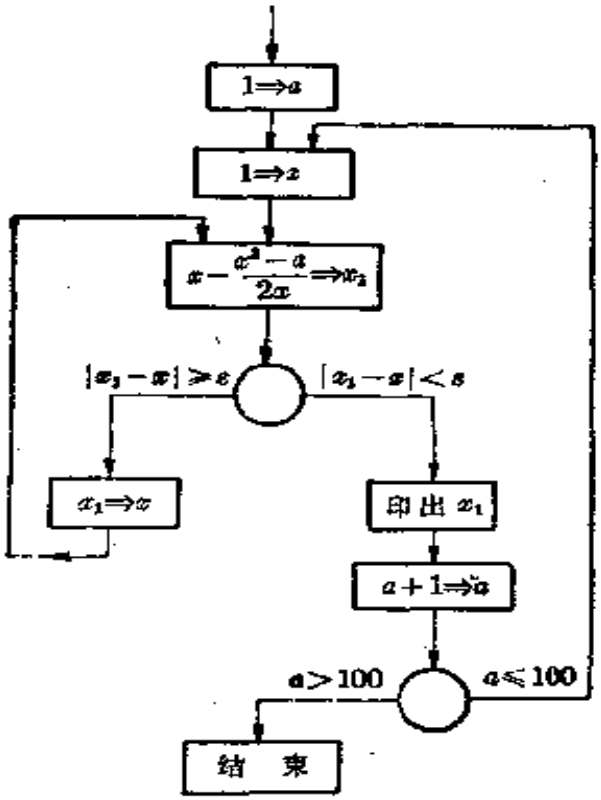


图 17

用语句写出来是：

```
for a:=1 step 1 until 100 do
begin x:=1;
  L1: x1:=x-(x×x-a)/(2×x);
    if #abs(x1-x)≥10-4 then
      begin x:=x1; goto L1 end;
    #print(0, 'F14.6', x1)
end;
```

习 题 三

1. 用赋值语句写出下列计算公式：

(1) $x=ab^3+c\cdot\ln(1+c^2)$;

(2) $x=\dfrac{-b+\sqrt{b^2-4ac}}{2a}$;

(3) $y=z=ax^4+|x^2\operatorname{tg}x-a|$.

2. 用循环语句写出下列各式的计算过程：

(1) 对 $x=100, 101, 102, \cdots, 150$, 求

$$y=ax^2\cos\left(\frac{x}{200}\pi\right)-\sqrt{x};$$

(2) 对 $x=0, 2, 4, \cdots, 50$, 求

$$y=\frac{\ln(1+\sqrt{x})}{1+x^2};$$

(3) 求105个数 $a_1, \cdots, a_{105}$ 的立方和: $D=\sum_{i=1}^{105}a_i^3$;

(4) 对 $a=2, 3, 4, 5, 6, 10, 24$, 求

$$x=\frac{a_i+\sqrt{a_i^2-1}}{a_i+1};$$

(5) 对 $a=a_0, a_1, a_2, a_3$, 求

$$w=(x_1^2-a)(x_2^2-a)\cdots(x_{100}^2-a).$$

3. 用条件语句描写下面两图中的曲线.

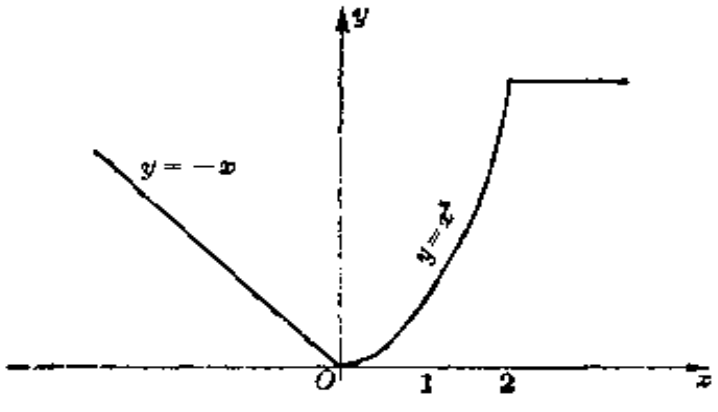


图 18

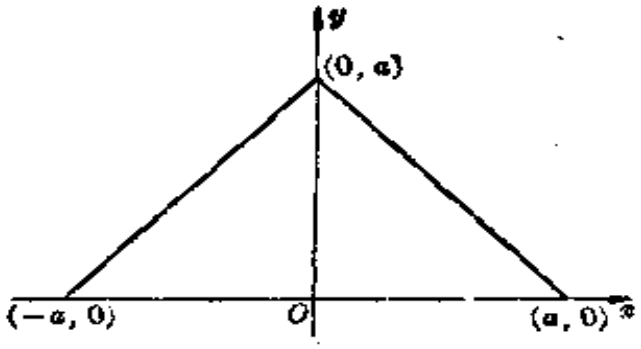


图 19

4. 用语句写出求 $a_0, \cdots, a_{50}$ 中正数之和 s 和负数之和 t 的计算公式.

5. 用语句写出下列各式:

(1) 给出 100 个数 $x_1, x_2, \cdots, x_{100}$, 求

$$D=\sqrt{\frac{(x_1-a)^2+(x_2-a)^2+\cdots+(x_{100}-a)^2}{100}};$$

(2) 给出一组数 $y_0, y_1, \cdots, y_{100}$, 求

$$D=y_1+y_2+\cdots+y_{n-1}+\frac{y_0+y_n}{2}.$$

6. 已知摆线的参数方程为

$$\begin{cases} x=t-\sin t, \\ y=1-\cos t, \end{cases}$$

要求对 $0\leqslant x<1$ 一段弧线, 从原点起, 对 t 每隔 0.1 求 x, y 的值. 试用语句写出其计算过程.

7. 已知两组数 $p_i, q_i (i=1, 2, \cdots, 100)$. 要求在方程 $x^2+p_ix+q_i=0$ 有实根时求出它的两个根, 并印出. 试用语句写出其计算过程.

8. 已知 $y=f(x)$ 在 n 个点 x_k 上的值 $y_k (k=1, 2, \cdots, n)$, 试求出 y_k 的最大绝对值 M, 并指出相应的 x_k 的值.

9. 已知

$$e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots,$$

试用算法语言描述下列计算过程:

(1) 求此级数前 10 项之和;

(2) 求此级数和, 精确到 10^{-5}.

10. 执行下列诸语句后，m、n、P之值为多少？

```
m:=1; n:=1;
for i:=1 step 1 until 6 do
  begin p:=m+n; m:=n; n:=p end;
```

11. 指出下列诸语句的书写错误：

(1) **if** $0<x<2$ **then** $y:=$#sqrt$(1-(x-1)\uparrow 2)$;

(2) $y:=$**if** $0\leqslant x$ **then** #sqrt$(1-(x-1)\uparrow 2)$;

(3) **if** $x<2$ **then if** $x>0$ **then** $y:=$#sqrt$(1-(x-1)\uparrow 2)$;

(4) **if** $x<0$ **then goto** L_1;

```
for i:=1, 2, 3 do
  L1:y:=#sqrt(x+i);
```

(5) #print(0, '$F14.6$, $E8.3$', x);

(6) #print(0, '10, $F14.6$', c, d);

(7) #print(0, '2$F14.7$', a, b[i]).

12. 设x、y[1:3]是一维数组，试用$F14.6$格式输出其分量成如下形式：

$$\left.\begin{array}{l} x_1=\cdots\cdots x_2=\cdots\cdots x_3=\cdots\cdots \\ y_1=\cdots\cdots y_2=\cdots\cdots y_3=\cdots\cdots \end{array}\right\}\text{空出两行字符位置}$$

(其中'……'均表示输出数所占用的字符位置)。

第四节　说明和分程序

程序中的说明，好象一本书的索引一样，它列出程序中的全部标识符，并且指明这些标识符的性质。机器根据说明部分所列出的资料，进行诸如给变量分配存贮单元、确定运算对象（同类型的变量才能进行运算）等工作。

程序的说明部分必须对语句部分中出现的标识符写出相应的说明（标号标识符例外），不能遗漏，也不能重复。各种形

式的说明之间必须用分号来分隔. 同一类型说明中的各个变量标识符的次序可以任意排列.

程序的说明一共有四种类型: 简单变量说明、数组说明、开关说明、过程说明. 本节介绍前面三种说明,过程说明将在下一节中专门介绍.

一 简单变量说明

简单变量说明是用来说明简单变量的, 在说明中同时指出变量的类型.

根据简单变量的不同性质,简单变量说明有三种形式:实型量说明、整型量说明、布尔量说明.

(一)实型量说明

实型量说明的一般形式是

$$\textbf{real}\ a,\ b,\ \cdots,\ z.$$

其中, **real** 是实型量说明的记号; a, b, $\cdots$, z 是被说明的实型简单变量标识符,它们之间用逗号分隔.

实型量说明的含义是,被说明的简单变量标识符

$$a,\ b,\ \cdots,\ z$$

所代表的量都规定在实数范围内取值. 实型量说明中的变量称为实型量,例如,在上一节第 64 页的例 1 中,语句部分出现的简单变量 t, x, y 都是在实数范围内取值的, 所以, 对这些变量标识符所作的说明应该是实型量说明,写成

$$\textbf{real}\ t,\ x,\ y.$$

现在, 我们根据程序的两个组成部分——说明部分和语句部分,再添加上程序的开始和结束信号,就可以写出这个计算问题的完整程序如下:

```
begin
说明部分  real t, x, y;  ……简单变量说明
语句部分  t:=0;  ……赋值语句
          L1:x:=#cos(t)+t×#sin(t);  ……赋值语句
          y:=#sin(t)-t×#cos(t);  ……赋值语句
          if x×x+y×y<16 then
            begin #print(0, '2F14.6', x, y);  } 条件语句
            t:=t+0.1; goto L1 end;
end 55…5
```

（二）整型量说明

整型量说明的一般形式是

integer a, b, …, z.

其中，**integer** 是整型量说明的记号；a, b, …, z 是被说明的整型简单变量标识符，它们之间用逗号分隔.

整型量说明的含义是，被说明的简单变量标识符 $a, b, \cdots, z$ 所代表的量都规定在整数范围内取值. 整型量说明中的变量称为整型量，例如，在上一节第 65 页的例 2 中，语句部分出现的简单变量 a 是在整数范围内取值的，所以，对它所作的说明应该是整型量说明，写成

integer a.

对于语句部分出现的另外两个简单变量 x, x_1，它们是在实数范围内取值的，所以，对它们所作的说明应该是实型量说明，写成

real x, x_1.

现在，对照程序的基本结构，就可以写出这个计算问题的完整程序如下：

```
begin
说明部分  integer a;  ……整型量说明
          real x, x1;  ……实型量说明
```

```
语句部分{ for a:=1 step 1 until 100 do                          }循环语句
          begin x:=1;
            L1:x1:=x-(x×x-a)/(2×x);
            if #abs(x1-x)≥10-4 then
              begin x:=x1; goto L1 end;
            #print(0, 'F14.6', x1)
          end
end 55…5
```

程序中对整型量和实型量分开说明，这样就便于阅读，但在 709 机和 TQ-16 机中，由于对整数和实数的表示作了相同的处理，所以在程序中这两种说明是可以相互替代的．例如，上面程序中的说明部分也可以写成

real a, x, x_1;

或

integer a, x, x_1;

等．

（三）布尔量说明

布尔量说明的一般形式是

boolean a, b, …, z.

其中，**boolean** 是布尔量说明的记号；a, b, …, z 是被说明的布尔简单变量标识符，它们之间用逗号分隔．

布尔量说明的含义是，被说明的简单变量标识符

a, b, …, z

所代表的量都规定在逻辑值范围内取值，即限于取 **true**(真)或 **false**(假)．

[例] 给定两组整数值：

$x=20, 19, \cdots, 1$, $y=100, 102, \cdots, 138$,

试对满足 $x+y$ 为偶数的(x, y)计算 $z=\sqrt{x^2+y^2}$．

这个问题的关键是要找出执行计算公式 $z=\sqrt{x^2+y^2}$ 的

判别条件．这只要利用下面的逻辑表达式：

$$(x+y)/2=(x+y)\div 2.$$

若逻辑表达式的值为 **true**，则知 $x+y$ 为偶数；若逻辑表达式的值为 **false**，则知 $x+y$ 不是偶数．如果我们把这个逻辑表达式的值赋给简单变量 u，显然，应该说明简单变量标识符 u 为布尔量，相应的说明写成

boolean u.

这个计算问题的完整程序可以写为

```
begin
    boolean u;  ..... 布尔量说明
    integer x, y;  ..... 整型量说明
    real z;  ..... 实型量说明
    for x:=20 step -1 until 1 do
      for y:=100 step 2 until 138 do
        begin u:=(x+y)/2=(x+y)÷2;
          if u then
          begin z:=#sqrt(x×x+y×y);
            #print(0, '3F14.6', x, y, z);
          end
        end
end 55…5
```

（说明部分：布尔量说明、整型量说明、实型量说明；语句部分：循环语句）

二　数组说明

数组说明是用来说明代表一组下标变量的数组标识符的，在说明中同时指出相应下标变量中下标的个数（即数组的维数）及下标取值的上、下界．由于机器对实型量和整型量不加区别，所以，作为数组标识符所代表的下标变量的类型，也总认为是实型量．

数组说明有两种形式：固定数组说明和半动态数组说明．

(一)固定数组说明

先举一个例子. 在上一节循环语句的例 4 中，描述计算过程的语句一共使用了三组下标变量：

$$A[i, j],\ B[j],\ x[i]$$

$$(i=1, 2, \cdots, 10;\ j=1, 2, \cdots, 12),$$

对代表这三组下标变量的数组标识符所作的说明写为

array $A[1{:}10, 1{:}12],\ B[1{:}12],\ x[1{:}10]$.

这里，**array** 是数组说明的记号. 这个数组说明表示：数组标识符 A 代表了 10×12 个具有两个下标的下标变量 $A[i, j]$，它的第一个下标 i 的取值范围是 1 到 10，第二个下标 j 的取值范围是 1 到 12. 数组标识符 B 代表了 12 个具有一个下标的下标变量 $B[j]$，下标 j 的取值范围是 1 到 12. 数组标识符 x 代表了 10 个具有一个下标的下标变量 $x[i]$，下标 i 的取值范围是 1 到 10. 这三组下标变量在机器里的存放次序是：

$A[1, 1],\ A[1, 2],\ \cdots,\ A[1, 12],$

$A[2, 1],\ A[2, 2],\ \cdots,\ A[2, 12],$

$\cdots,\ \cdots,\ \cdots,\ \cdots,\ \cdots,\ \cdots,\ \cdots,\ \cdots,\ \cdots,$

$A[10, 1],\ A[10, 2],\ \cdots,\ A[10, 12];$

$B[1],\ B[2],\ \cdots,\ B[12];$

$x[1],\ x[2],\ \cdots,\ x[10]$.

显然，这里说明的数组标识符 A, B 和 x，它们所代表的下标变量的个数都是完全确定的，所以，称这种数组说明为固定数组说明.

现在，我们对照程序的结构写出这个例子的完整程序如下：

```
begin
说明部分 { array A[1:10, 1:12], B[1:12], x[1:10]; …… 固定数组说明
         { integer i, j; …… 整型量说明
语句部分 { #read(0, '10', A, B); …… 输入语句
         { for i:=1 step 1 until 10 do
             begin x[i]:=0;
               for j:=1 step 1 until 12 do
                 x[i]:=x[i]+A[i, j]×B[j]
             end;                                  } …循环语句
         { #print(0, 'F14.6', x) …… 输出语句
end 55…5
```

固定数组说明的一般形式是

$$\textbf{array}\ A[\Delta_1],\ B[\Delta_2],\ \cdots,\ Z[\Delta_n].$$

其中，**array** 是数组说明的记号；A, B, …, Z 是被说明的数组标识符；Δ_1, Δ_2, …, Δ_n 是相应的数组界偶表，它们均由若干对界偶构成，每对界偶是由用冒号相联的两个整常数(称为界偶的上、下界)构成，界偶表中的各对界偶之间要用逗号分隔，而且规定，界偶的上界值不能小于下界值。

界偶表中界偶的对数，就是数组的维数，即，只有一对界偶的数组是一维数组，有两对界偶的数组是二维数组，……。显然，数组的维数也就是相应下标变量中下标的个数。例如，固定数组说明

array $R[1:100]$, $H[0:1, 1:9]$, $S[1:2, 0:1, 1:60]$

中，R 是一维数组，H 是二维数组，S 是三维数组，它们分别代表如下排列次序的三组下标变量：

$R[1]$, $R[2]$, …, $R[100]$;

$H[0, 1]$, $H[0, 2]$, …, $H[0, 9]$,

$H[1, 1]$, $H[1, 2]$, …, $H[1, 9]$;

$S[1, 0, 1]$, $S[1, 0, 2]$, …, $S[1, 0, 60]$,

$S[1, 1, 1], S[1, 1, 2], \cdots, S[1, 1, 60],$

$\cdots, \cdots, \cdots, \cdots, \cdots, \cdots, \cdots, \cdots, \cdots, \cdots,$

$S[2, 1, 1], S[2, 1, 2], \cdots, S[2, 1, 60].$

现在举一个使用固定数组的程序例子.

[例] 设 a 和 b 是分别代表 50 个下标变量的数组标识符，试把它们的值对换一下，即

$a[1] \rightleftarrows b[1], a[2] \rightleftarrows b[2], \cdots, a[50] \rightleftarrows b[50].$

这个计算问题的完整程序是

```
begin
    array a[1:50], b[1:50];  ..... 固定数组说明
    integer i;               ..... 整型量说明
    real c;                  ..... 实型量说明
    #read(0, '10', a, b);    ..... 输入语句
    for i:=1 step 1 until 50 do
      begin c:=a[i];
            a[i]:=b[i];
            b[i]:=c
      end;                   ..... 循环语句
    #print(0, '2F14.6', a, b) ..... 输出语句
end 55...5
```

最后，还须指出，数组说明中界偶的上、下界值是相对固定的，例如，由 $x_1, x_2, \cdots, x_{10}$ 十个分量构成的数组，在说明中可以写成 **array** $x[1:10]$，也可以写成 **array** $x[0:9]$ 或 **array** $x[-3:6]$ 等. 但一当数组说明的形式确定以后，则下标变量同下标值的对应关系也就相应地被确定了，如第一种形式中，$x_1=x[1], x_2=x[2], \cdots, x_{10}=x[10]$，第二种形式中，$x_1=x[0], x_2=x[1], \cdots, x_{10}=x[9]$，第三种形式中，$x_1=x[-3], x_2=x[-2], \cdots, x_{10}=x[6]$ 等.

另外，对于三维以下的数组，若它们在同一个数组说明中

具有相同的界偶表，则在书写说明时可予简化，例如，数组说明

array $A[1:15]$, $B[1:15]$,
$C[0:5, 1:10]$, $D[0:5, 1:10]$

允许简化为

array A, $B[1:15]$, C, $D[0:5, 1:10]$.

(二)半动态数组说明

先举一个例子. 设有矩阵

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

如果矩阵 A 的行数 m，列数 n 都是简单变量，即 m、n 的值是可以变化的，这时，下标变量的个数随 m、n 的变化而改变，下标的取值范围是不确定的，因此，在对代表下标变量 $A[i, j]$ 的数组标识符 A 进行说明时，就不能再用固定数组说明. 在这种情况下，为了对数组标识符 A 进行说明，使机器能为相应下标变量开列存贮单元，就得给机器指出该数组标识符所代表的下标变量的最多个数. 譬如，对于 $m\leqslant 10$，$n\leqslant 8$ 的情形，$A[i, j]$ 就最多有 $10\times 8=80$ 个，这个代表下标变量最多个数的值，称为数组的最大体积. 对数组标识符 A 的说明规定写为

array $A(80)[1:m, 1:n]$.

显然，这个数组说明中界偶的上界值是可变的，即数组标识符 A 代表的下标变量的个数是变动的，所以称这种数组说明为半动态数组说明.

半动态数组说明的一般形式是

array $A(P_1)[\Delta_1]$, $B(P_2)[\Delta_2]$, …, $Z(P_n)[\Delta_n]$.

其中，**array** 是数组说明的记号；A, B, …, Z 是被说明的数组标识符；用圆括号括起来的 P_1, P_2, …, P_n 是相应数组标识符 A, B, …, Z 的最大体积，即相应数组所代表的下标变量个数的上界，它们都是正的整常数；Δ_1, Δ_2, …, Δ_n 是相应数组的界偶表，它们的含义同固定数组说明中的一样，不过这里界偶的上、下界不限于整常数，也可以是简单变量，但不能是算术表达式，这一点是和下标变量中对下标的规定不相同的.

为了使半动态数组界偶中的上、下界有确定的数值，就必须在说明之前给出它们的数值，通常，可以在半动态数组说明的前面，利用输入语句或其他计算语句使界偶的上、下界获得确定的数值. 譬如，利用输入语句时，写成

```
#read(0, '10', m, n);
  ..........
  ..........
begin
  array A(80)[1:m, 1:n];
  ..........
  ..........
  ..........
end.
```

再如，利用赋值语句(设 E_1、E_2 为算术表达式)时，写成

```
m:=E1; n:=E2;
  ..........
  ..........
begin
```

```
        array A(80)[1:m, 1:n];
        ..........
        ..........
        ..........
        ..........
    end,
```

等.

必须指出，上面两种确定界偶上、下界的方法中，代表界偶上界的简单变量标识符 m、n，也应该预先加以说明，因此，在使用半动态数组说明时，整个程序将是分层的：

```
第一层 { begin
         integer m, n;
           #read(0, '10', m, n);
           (或 m:=G00001; n:=G00002;)
         第二层 { begin
                    array A(80)[1:m, 1:n];
                    ..........
                    ..........
                  end;
                  ..........
         end 55…5
```

这种分层进行说明的形式，大大地扩展了程序的结构，我们将在后面加以详细讨论.

下面举一个使用半动态数组说明的程序例子.

[例] x 是具有 n 个分量 x_i ($i=1, 2, \cdots, n$, $n\leqslant 50$) 的向量，试编写求模公式：$\text{norm}=\sqrt{x_1^2+x_2^2+\cdots+x_n^2}$ 的程序.

计算问题的完整程序如下：

```
begin
说明部分  integer n; ····· 整型量说明
          #read(0, '10', n); ····· 输入语句
          begin
            array x(50) [1:n]; ····· 半动态数组说明
            integer i; ····· 整型量说明
            real norm; ····· 实型量说明
语句部分      #read(0, '10', x); ····· 输入语句
              norm:=0; ····· 赋值语句
              for i:=1 step 1 until n do
                norm:=norm+x[i]↑2; } 循环语句
              norm:=#sqrt(norm); ····· 赋值语句
              #print(0, 'F14.6', norm) ····· 输出语句
          end
end 55···5
```

（第二层 / 第一层）

这里，由于给定向量的分量个数 n 是不确定的，所以对代表 n 个分量的数组标识符 x 所作的说明使用了半动态数组说明，且由于 $n\leqslant 50$，所以最大体积取为 50. 半动态数组说明中界偶的上界值用赋值语句给出.

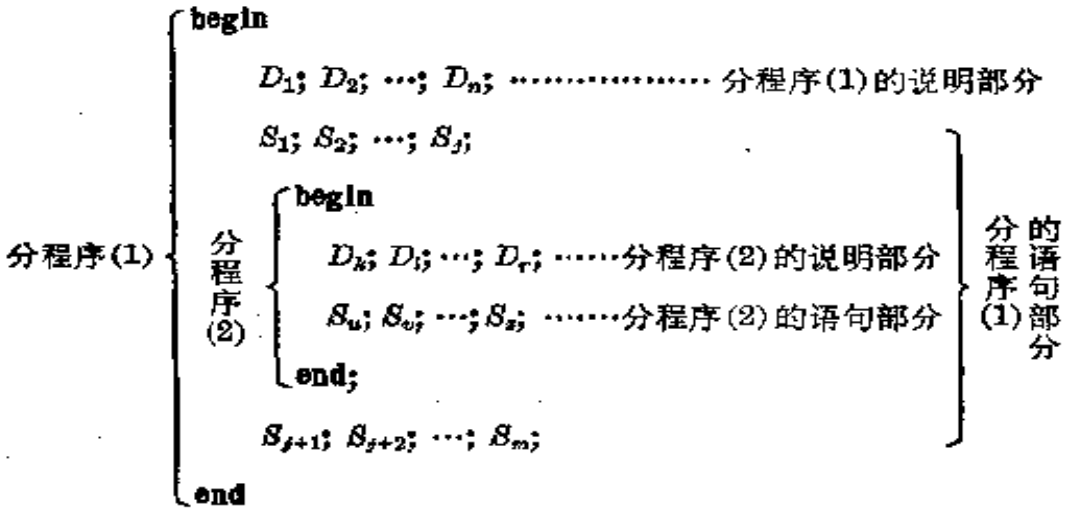
三 分 程 序

前面已经看到，在使用半动态数组说明时，需要将程序写成分层说明的形式. 一般地，一个比较复杂的程序，也常常需要写成分层的形式，其中每一层程序表达了一个完整的数学含义，称为分程序.

分程序的一般形式是

```
begin
  D1; D2; ···; Dn; ····· 分程序的说明部分
  S1; S2; ···; Sm; ····· 分程序的语句部分
end
```

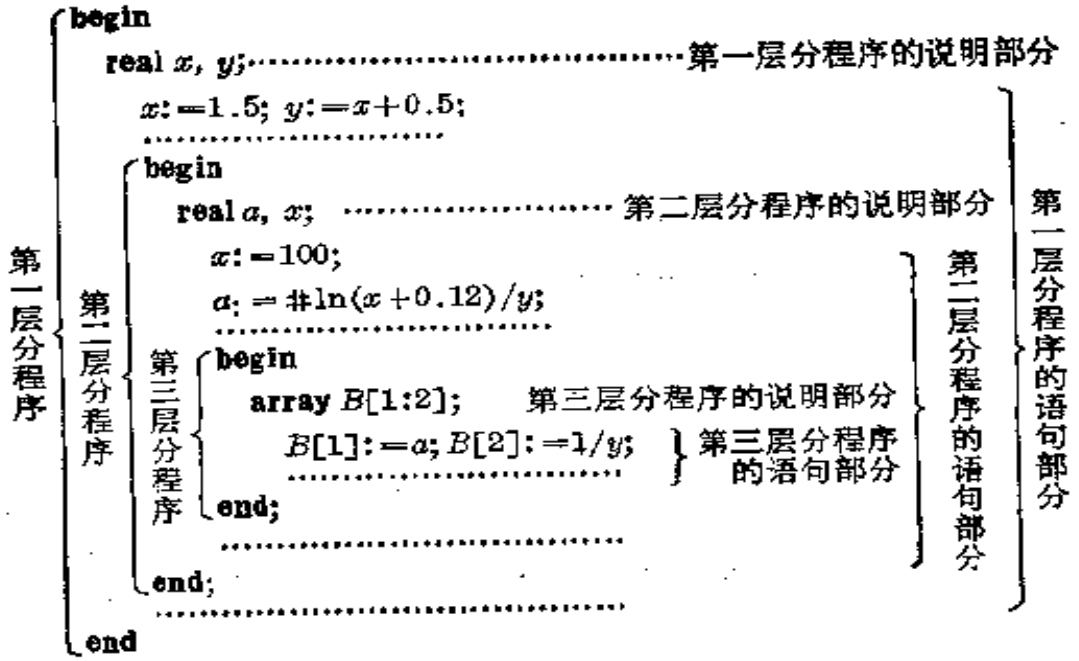
其中，D_1，D_2，…，D_n 是说明；S_1，S_2，…，S_m 是语句或分程序，也就是说，分程序内部允许再出现分程序，例如，



为了确定完整程序中多个分程序的层次，我们根据分程序所处的相对位置，把处在内层的叫做内层分程序，处在外层的叫做外层分程序，并且把最外层的分程序作为第一层，从外向里依次计算分程序的层次．对于并列位置上的分程序它们的层次是相同的．

本节开头我们曾强调指出，程序中出现的每一个标识符都必须在它的说明部分进行说明．然而对于分程序来说，它的说明部分不一定要对其内部出现的标识符都加以说明，其中一部分标识符可以在包含该分程序的任何一个外层分程序中说明．这时，在分程序头部说明的量，称为该分程序的局部量，分程序中出现的而未在该分程序中说明的量，称为该分程序的非局部量．显然，局部和非局部都是相对的，对某一分程序来讲是局部的量，但当其在内层分程序中出现时却成了内层分程序的非局部量．

我们举一个包含三层分程序的程序例子．



在这个例子中，对第三层分程序来说，B 是局部量，a、y 是非局部量。对第二层分程序来说，a、x 是局部量，y 是非局部量.

分程序的局部量只限于在该分程序内部有意义，但分程序的非局部量在该分程序的内、外都有意义. 因此，作为分程序中局部量的标识符，在该分程序的外面，它可以用来标识其他的变量.

例如，

```
第一层分程序 begin
               real a;
               array b[1:2];
                 a:=5; b[1]:=11; b[2]:=12;
               第二层分程序 begin
                              real b;
                                b:=0;
                                L1:#print(0, '2I2', a, b)
                            end;
               L2:#print(0, '2I2', a, b)
             end
```

在这个程序中，对第二层分程序来说，b 是局部量，a 是非局部量，因此，标号为 L_1 的输出语句被执行时将印出数值：5，0．对第一层分程序来说，a、b 都成了局部量，因此，标号为 L_2 的输出语句被执行时将印出数值：5、11、12．特别要指出的是，作为第二层分程序的局部量 b，它的标识符在第一层分程序中用作数组标识符，这是完全允许的。

再如，

```
第一层分程序 {
  begin
    real d;
    第二层分程序(1) {
      L1: begin
            real a, b, i;
            a:=2; b:=3;
            i:=a+b; d:=3×i
          end;
    }
    第二层分程序(2) {
      L2: begin
            array a[1:5];
            real c;
            integer i;
            #read(0, '10', a);
            c:=0;
            for i:=1 step 1 until 5 do
              c:=c+a[i]
          end;
    }
    d:=d+2
  end
}
```

在这个程序中，L_1，L_2 是两个并列形式的分程序；a，i 分

别是 L_1 分程序和 L_2 分程序的局部量，即它们分别在本层分程序内部有意义．另外，L_1 分程序中的 d，对 L_1 分程序来说是非局部量，所以，它在 L_1 分程序的内、外均有意义．从而，程序执行过程中变量 d 将先获得数值 $3(2+3)=15$，然后获得数值 $15+2=17$．

程序中分程序的引入及局部量的使用，使每一个完整的数学含义可以用一个分程序来描述，这样，从程序结构上来看，将是十分清晰的；另外，分程序的采用往往还能节省变量在机器中的存贮单元，因为，局部于几个分程序的量，它们是重合使用着某些单元的。

例如，

```
begin
  integer i; real z1, z2;
  array x[1:200], y[1:150];
    #read(0, '10', x, y);
    z1:=0;
    for i:=1 step 1 until 200 do
      z1:=z1+x[i]↑3;
    z2:=0;
    for i:=1 step 1 until 150 do
      z2:=z2+y[i]↑4;
    #print(0, '2F14.6', z1, z2)
end.
```

显然，这个程序中的变量 i, z_1, z_2, x, y 在机器中将占有 $1+1+1+200+150=353$ 个单元．如果把程序改写成下面分程序的形式，那么就可以达到节约单元的目的。

```
第一层分程序 { begin
                integer i; real z1, z2;
                第二层分程序(1) { L1:begin
                                    array x[1:200];
                                    #read(0, '10', x);
                                    z1:=0;
                                    for i:=1 step 1 until 200 do
                                      z1:=z1+x[i]↑3
                                  end;
                第二层分程序(2) { L2:begin
                                    array y[1:150];
                                    #read(0, '10', y);
                                    z2:=0;
                                    for i:=1 step 1 until 150 do
                                      z2:=z2+y[i]↑4;
                                  end;
                #print(0, '2F14.6', z1, z2)
              end
```

这个程序中，L_1、L_2 是两个并列的分程序. L_1 分程序中，i、z_1 是非局部量，x 是局部量，它们在机器中占有 1+1+200=202 个单元；L_2 分程序中，i、z_2 是非局部量，y 是局部量，且 i 就是 L_1 分程序中同一个量 i，所以，它在机器中将占用一个单元. 另外，由于 y 是局部于 L_2 分程序的量，它在机器中将自动重用 L_1 分程序中 x 所占用的单元. 这样，整个程序中变量占用的单元将是 1+1+1+200=203 个.

最后还须指出，分程序对转向语句的使用有一个限制，即利用转向语句进入分程序时，不允许越过分程序的头部 **begin** 转入分程序内部的任何语句. 然而，相反方向的转向却是允许的，即分程序内部的转向语句允许转出分程序进入外层分程序的任一语句. 例如，下面一段程序是不允许的：

goto L_1;

L_0: **begin**

D_1; D_2; …; D_m;

S_1; S_2; …; S_j; L_1:S_k; …; S_n

end

其中，D_1, D_2, …, D_m 是分程序的说明；S_1, S_2, …, S_n 是分程序中的语句.

然而，下面一段程序却是允许的：

begin

D_1; D_2; …; D_m;

S_1; S_2; …; S_j; **goto** L_1; S_k; …; S_n

end;

……

L_1: S

其中，D_1, D_2, …, D_m 是分程序的说明；S_1, S_2, …, S_n 是分程序的语句；S 是语句.

关于使用分程序时各种允许和不允许的转入、转出方案，同使用循环语句时对转向语句的限制一样.

四 开关说明

开关说明是为第二种转向语句确定转向标号的.

开关说明的一般形式是

switch $S:=L_1, L_2, \cdots, L_n$.

其中，**switch** 是开关说明的记号；S 是开关标识符；L_1, L_2, …, L_n 是用逗号分隔的一串标号，称为开关表. 开关表中标号的排列次序是根据程序的转向要求设计而成的.

开关说明怎样为第二种转向语句确定转向标号呢？

如前所述，第二种形式的转向语句 **goto** $S[E]$ 中，开关变量 $S[E]$ 根据 E 的动态值可以确定出一个标号来，这个标号规定由标识符为 S 的开关说明来提供，它就是开关表中从左数起的第 E 个标号，从而，转向语句就把这个标号作为自己的转向目标.

例如，若设开关说明是

$$\textbf{switch}\, G := A,\ B,\ C,\ D,$$

则对转向语句 **goto** $G[E]$ 来说，当 E 值等于 2 时，就相当于 **goto** B，当 E 值等于 3 时，就相当于 **goto** C 等，如果 E 的值超出了相应开关表中标号的个数，则程序将导致运算错误.

必须强调，开关说明是说明的一种，应该放在程序的说明部分，而转向语句则是语句的一种，应该在程序的语句部分使用. 开关说明同第二种形式转向语句配合起来就可以实现程序的多头转向，使程序的结构更加紧凑. 现在举两个使用开关说明的程序例子.

[例 1] 给定一组数 $x_i(i=1,\ 2,\ \cdots,\ 6)$，试按下面公式计算相应的 $y_i(i=1,\ 2,\ \cdots,\ 6)$：

$$y_i=\begin{cases} x_i+1, & i=1,\ 4,\\ 2x_i, & i=2,\ 5,\\ x_i/3, & i=3,\ 6. \end{cases}$$

我们用数组标识符 x、y 分别表示给定数据和计算结果，并设计一个开关说明(设 R 为它的开关标识符)，再利用第二种转向语句 **goto** $R[i]$ 就可以写出计算问题的程序如下：

```
begin
  说明部分{ switch R:=L1, L2, L3, L1, L2, L3; ····· 开关说明
            integer i; ····· 整型量说明
            array x, y[1:6]; ····· 数组说明
```

```
语句部分 { #read(0, '10', x); ····· 输入语句
           i:=1; ····· 赋值语句
      L0: goto R[i]; ····· 第二种形式转向语句
      L1: y[i]:=x[i]+1; goto L4; ····· 赋值语句、转向语句
      L2: y[i]:=2×x[i]; goto L4; ····· 赋值语句、转向语句
      L3: y[i]:=x[i]/3; ····· 赋值语句
      L4: i:=i+1; ····· 赋值语句
          if i≤6 then goto L0; ····· 条件语句
          #print(0, 'F14.6', y) ····· 输出语句
end 55…5
```

[例 2] 设

$$g(x)=\begin{cases}\arcsin(ax+b), & 0<x<1,\\ \sum_{i=0}^{10} c_i x^i, & 1\leqslant x<2,\\ \ln(d^2-x), & 2\leqslant x<3.\end{cases}$$

试对已知的 $x_1, \cdots, x_{20}(0<x_i<3)$，求 g 的值，并要求当 $g(x)$ 无意义时打印符号 NO.

使用开关说明，完整的程序可写成

```
begin
说明部分 { real a, b, d, g; ····· 实型量说明
           integer e, i, j; ····· 整型量说明
           array c[0:10], x[1:20]; ····· 固定数组说明
语句部分 { #read(0, '10', a, b, d, c, x); ····· 输入语句
           for j:=1 step 1 until 20 do
           { 分程序
             begin switch P:=L1, L2, L3; ····· 开关说明
                 e:=#entier(x[j])+1; ····· 赋值语句
                 goto P[e]; ····· 转向语句
             L1: if #abs(a×x[j]+b)>1 then
                 #print(0, '2HNO')
                 else g:=#arcsin(a×x[j]+b); } …条件语句
                 goto L4; ····· 转向语句
             L2: g:=0; ····· 赋值语句
                 for i:=10 step -1 until 0 do
                 g:=c[i]+g×x[j]; } …循环语句
                 goto L4; ····· 转向语句
             L3: if d×d-x[j]≤0 then
                 #print(0, '2HNO')
                 else g:=#ln(d×d-x[j]); } …条件语句
             L4: #print(0, '2Hj=, I3, 15x,
                 2Hg=, F14.6, 'j, g); } …输出语句
             end
           } 循环语句
end 55…5
```

以上三种类型的说明和下一节将要介绍的过程说明，构成了程序说明部分的全部内容，对这些类型的说明，我们可以根据计算程序的具体要求来选用. 说明部分中，各种类型的说明的次序可以是任意的，但必须注意，作为程序的说明部分，必须紧跟在 **begin** 的后面，因此，如果在语句部分的任何位置上需要引入说明部分，那就必须在这个说明前面添加一个 **begin**，而在后面的适当地方添加一个 **end**. 这样，实际上是引进了一层新的分程序.

习 题 四

1. 设有数组 $a_i(i=1, 2, \cdots, 10)$，$d_{ij}(i=1, 2, 3;\ j=1, 2, \cdots, 10)$，$e_{ij}(i=1, 2, 3;\ j=1, 2, \cdots, 10)$，$f_{ijk}(i=1, 2, 3;\ j=1, 2;\ k=1, 2, \cdots, 5)$，试写出其数组说明.
2. 写出下列计算过程的完整程序：
   (1) 给出正整数 $a_i(i=1, 2, \cdots, 100)$，求其奇数之和 N 及偶数之和 M；
   (2) 设 $A=(a_{ij})$，$B=(b_{jk})$，$C=(c_{ik})$，求

$$c_{ik}=\sum_{j=1}^{n} a_{ij}b_{jk} \quad (i=1, 2, \cdots, m;\ k=1, 2, \cdots, l)$$

$$n, m, l \leqslant 10.$$

3. 试确定下面一段分程序从开始运算到结束时印出些什么数，并指出哪些是局部量，哪些是非局部量？

```
(1) begin
      real x, y;
        y:=1;
      L: x:=y+1; y:=x+y;
        begin
          real x;
            x:=y+1; y:=x+y;
```

```
        if x<10 then goto L;
        #print (0, '2I3', x, y)
      end;
      if x<10 then goto L;
      #print(0, '2I3', x, y)
    end
```

(2)
```
    begin
      switch R:=L, M, M;
        i:=0;
      L:x:=0;
      M:i:=i+1; x:=3×x+1;
        #print(0, 'F14.6', x);
        if i≤3 then goto R[i]
    end
```

4. 求复合函数 $z=f(g(x))$ 在 $x=0.6, 1.25, 2.3, 1.8, 0.45$ 时的函数值. 已知:

$$y=g(x)=\begin{cases}\operatorname{tg}(ax)+b, & \text{当 } 0\leqslant x\leqslant 1,\\ a^2-b^2x^2, & \text{当 } 1<x\leqslant 2,\\ \ln(a^2+b^2x^2), & \text{当 } 2<x\leqslant 3,\end{cases}$$

$$z=f(y)=(\cos^2 y+e^y)\sqrt{a^2+b^2},$$

其中 a, b 为常量, 试采用开关说明和第二种转向语句形式写出计算过程的完整程序.

5. 计算问题:

$$s_1=a+\sum_{i=1}^{10}r_i,\quad s_2=b+\sum_{i=1}^{20}s_i^2,$$

$$s_3=c+\sum_{i=1}^{30}t_i^5,\quad M=s_1+s_2+s_3,$$

其中, a, b, c 是给定常数; $r_i(i=1, 2, \cdots, 10)$, $s_i(i=1, 2, \cdots, 20)$, $t_i(i=1, 2, \cdots, 30)$ 是原始数据; M, s_1, s_2, s_3 是所求结果.

下面写出了这个计算问题的程序，试指出其中使用分程序、半动态数组及开关说明的错误，并写出这个计算问题的正确程序。

```
begin
  real a, b, c;
  integer i;
    #read(0, '10', a, b, c);
    for i:=1 step 1 until 3 do
      begin real M;
            integer j;
            array x[1:10×i], s[1:3];
            switch K:=L1, L2;
        #read(0, '10', x);
        goto s[i];
        L1:s[1]:=a;
          for j:=1 step 1 until 10 do
            s[1]:=s[1]+x[j];
          goto L4;
        L2:s[2]:=b;
          for j:=1 step 1 until 20 do
              s[2]:=s[2]+x[j]↑2;
          goto L4;
        L3:s[3]:=c;
          for j:=1 step 1 until 30 do
              s[3]:=s[3]+x[j]↑5;
        L4:end;
          M:=0;
          for i:=1 step 1 until 3 do
              M:=M+s[i];
          #print(0, '2F14.6', s, M)
end 55…5
```

第五节　过　　程

我们已经知道，循环语句可以实现同一计算过程的多次重复计算，因而它在程序设计中是一种常用的语句．但是，使用循环语句并不是缩短程序的唯一途径，特别是对于同一个计算过程用多种不同参数进行计算的情形，常常采用另一种更加有效的程序设计技术——过程．

过程，它包括过程说明和过程调用两个方面．前者是说明的一种形式，放在程序的说明部分；后者是语句的一种形式，或是一个函数命名符，在程序的语句部分使用．过程说明和过程调用配合使用，就可以达到紧缩程序的目的．

一　过程的例子

[例 1]　计算边长为 x、y、z 的三角形面积 A 的公式是

$$A=\sqrt{s(s-x)(s-y)(s-z)},$$

其中，$s=\frac{1}{2}(x+y+z)$．

试利用这个公式，求图 20 中三个三角形的面积之和：

$$\mathrm{sum}=A_1+A_2+A_3.$$

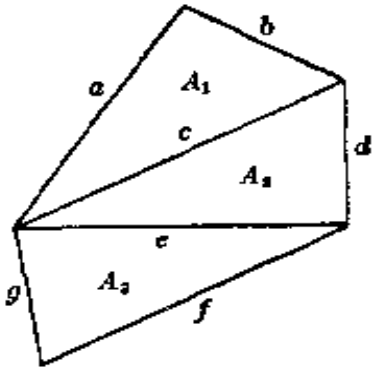


图　20

我们先用一般的方法写出这个计算问题的程序：

```
        begin
说明部分  real a, b, c, d, e, f, g, s, sum;
          #read(0, '10', a, b, c, d, e, f, g);
          s:=(a+b+c)/2;
          sum:=#sqrt(s×(s-a)×(s-b)×(s-c));
语句部分  s:=(c+d+e)/2;
          sum:=#sqrt(s×(s-c)×(s-d)×(s-e))+sum;
          s:=(e+f+g)/2;
          sum:=#sqrt(s×(s-e)×(s-f)×(s-g))+sum;
          #print(0, 'F14.6', sum)
        end 55…5
```

分析这个程序就可以看出，它的语句部分有三组形式相同的语句，即

第一组：

$s:=(a+b+c)/2;$

$\mathrm{sum}:=\#\mathrm{sqrt}(s\times(s-a)\times(s-b)\times(s-c));$

第二组：

$s:=(c+d+e)/2;$

$\mathrm{sum}:=\#\mathrm{sqrt}(s\times(s-c)\times(s-d)\times(s-e));$

第三组：

$s:=(e+f+g)/2;$

$\mathrm{sum}:=\#\mathrm{sqrt}(s\times(s-e)\times(s-f)\times(s-g));$

如果把这三组语句写成公用的形式，则有

$s:=(x+y+z)/2;$

$Q:=\#\mathrm{sqrt}(s\times(s-x)\times(s-y)\times(s-z)).$

这里，x、y、z 是公用的参数，如果相应地代入 a、b、c，则公用语句组就是原来的第一组语句；代入 c、d、e，则就是原来的第二组语句；代入 e、f、g，则就是原来的第三组语句。另

外，s 是不依赖于三组语句的中间变量标识符，Q 是存放计算结果的变量. 由此看来，用公用参数描述的公用语句组是完全可以代替原来三组语句的. 为此，我们把这个公用语句组添加一些符号后，构成一个独立的程序块，称为过程说明，其形式是

```
real procedure Q(x, y, z);
    value x, y, z;
    real x, y, z;
  begin
    real s;
      s:=(x+y+z)/2;
      Q:=#sqrt(s×(s-x)×(s-y)×(s-z))
  end.
```

这里，**real procedure** 是过程说明的类型；Q 是过程标识符；圆括号里用逗号分隔的公用参数 x、y、z 叫形式参数；**value** x, y, z 称为值部分；**real** x, y, z 称为区分部分；由公用语句组构成的分程序 **begin real** s; …**end** 称为过程体.

过程说明是说明的一种形式，所以把它放在程序的说明部分. 由于程序中有了这个过程说明，原来语句部分的三组形式相同的语句组就可以分别用一个称为函数命名符的记号来代替，即它们的功能可以由函数命名符调用相应过程说明来完成，这三个函数命名符的形式分别写成

$$Q(a,\ b,\ c);$$
$$Q(c,\ d,\ e);$$
$$Q(e,\ f,\ g).$$

这里，Q 是过程标识符. 圆括号里用逗号分隔的参数 a, b, c; c, d, e; e, f, g 分别同过程说明中的形式参数一一对

应，它们是三个语句组的计算参数，称为实在参数.

从而，采用过程的程序设计方法就可以在达到同样计算结果的前提下把程序改写成更加紧凑的形式：

```
begin
  real procedure Q (x, y, z); 过程类型、过程标识符、形式参数
    value x, y, z; ····· 值部分
    real x, y, z; ····· 区分部分
      begin
        real s;
          s:=(x+y+z)/2;
          Q:=#sqrt(s×(s-x)×(s-y)×(s-z))
      end;
  real a, b, c, d, e, f, g, sum; ·····实型量说明
    #read(0, '10', a, b, c, d, e, f, g); ·····输入语句
    sum:=Q(a, b, c); ·····赋值语句
    sum:=Q(c, d, e)+sum; ·····赋值语句
    sum:=Q(e, f, g)+sum; ·····赋值语句
    #print(0, 'F14.6', sum) ·····输出语句
end 55···5
```

（过程说明 = 过程类型、过程标识符、形式参数 + 值部分 + 区分部分 + 过程体；说明部分 = 过程说明 + 实型量说明；语句部分 = 输入语句 + 赋值语句 + 赋值语句 + 赋值语句 + 输出语句）

[例 2] 给定已知数：

$$a_i,\ b_i \quad (i=1,\ 2,\ \cdots,\ 50),$$
$$c_i,\ d_i \quad (i=1,\ 2,\ \cdots,\ 100),$$
$$e_i,\ f_i \quad (i=1,\ 2,\ \cdots,\ 150),$$

试计算：

$$x_i=a_i+b_i \quad (i=1,\ 2,\ \cdots,\ 50),$$
$$y_i=c_i+d_i \quad (i=1,\ 2,\ \cdots,\ 100),$$
$$z_i=e_i+f_i \quad (i=1,\ 2,\ \cdots,\ 150).$$

我们先用数组标识符 a、b、c、d、e、f 代表给定数据，x、y、z 代表计算结果，则计算问题的程序可以写为

```
begin
说明部分 { integer i;
           array x, a, b[1:50], y, c, d[1:100], z, e, f[1:150];
语句部分 { #read(0, '10', a, b, c, d, e, f);
           for i:=1 step 1 until 50 do
             x[i]:=a[i]-b[i];
           for i:=1 step 1 until 100 do
             y[i]:=c[i]+d[i];
           for i:=1 step 1 until 150 do
             z[i]:=e[i]+f[i];
           #print (0, '3F14.6', x, y, z)
end 55…5
```

这个程序象例1一样，语句部分也存在着用同一个计算公式分别对三组不同参数：x, a, b; y, c, d; z, e, f 进行计算的语句组

$$\textbf{for}\ i:=1\ \textbf{step}\ 1\ \textbf{until}\ n\ \textbf{do}$$
$$t[i]:=R[i]+s[i].$$

但这里同例1不同之点是，作为计算过程的结果是一组值而不是一个值. 为此，我们把公用计算过程的语句组写成另一种形式的过程说明：

```
procedure SL(n, R, s, t);
  value n;
  integer n;
  array R, s, t;
    begin
      integer i;
        for i:=1 step 1 until n do
          t[i]:=R[i]+s[i]
    end.
```

这里，**procedure** 是过程说明的类型，SL 是过程标识符. 圆括号里用逗号分隔的公用参数 n, R, s, t 叫形式参数. **value** n 称为值部分. **integer** n; **array** R, s, t 称为区分部分. 由公用语句组构成的分程序 **begin integer** i; … **end** 称为过程体.

我们把这个过程说明放在程序的说明部分，原来三个重复语句组的地方分别用三个称为过程语句的语句来代替，即它们的功能可以由过程语句调用相应过程说明来完成. 这三个过程语句的形式分别是

$$SL(50,\ a,\ b,\ x);$$
$$SL(100,\ c,\ d,\ y);$$
$$SL(150,\ e,\ f,\ z).$$

这里，SL 是过程标识符. 圆括号里用逗号分隔的参数 50, a, b, x; 100, c, d, y; 150, e, f, z 分别是三个语句组的计算参数，它们同相应过程说明中的形式参数一一对应，称为过程语句的实在参数.

从而，原来的程序就改写成过程的形式：

```
begin
  procedure SL(n, R, s, t); … 过程类型、过程标识符、形式参数
    value n; ..... 值部分
    integer n; .....
    array R, s, t; ..... } 区分部分
      begin
        integer i;
          for i:=1 step 1 until n do
            t[i]:=R[i]+s[i]
      end;                                  } 过程体
  array x, a, b[1:50], y, c, d[1:100], z, e, f[1:150]; 数组说明
```

说明部分 ｜ 过程说明

语句部分
```
    #read(0, '10', a, b, c, d, e, f); ····· 输入语句
    SL(50, a, b, x); ····· 过程语句
    SL(100, c, d, y); ····· 过程语句
    SL(150, e, f, z); ····· 过程语句
      #print(0, '3F14.6', x, y, z) ····· 输出语句
end 55···5
```

二 过程的一般形式

由上面两个例子可以看出，过程是由过程说明和过程调用两部分构成的. 过程说明好比一个公用的程序块，它用语句描述了一个完整的计算过程. 通过过程语句或函数命名符来调用它，就可以使这个计算过程实现对不同参数组进行计算的要求.

下面介绍过程两部分的一般形式.

(一)过程说明

过程说明的一般形式是[注]

<过程类型><过程标识符><形式参数部分>;

<值部分>;

<区分部分>;

<过程体>.

现在分别加以说明.

1. <过程类型>. 过程类型是指明该过程属于何种类型的记号. 常用的过程有函数过程和一般过程两种，它们的类型记号分别写成

real procedure——函数过程,

procedure——一般过程.

[注] 这里使用的尖括号，是为了把每一个独立的符号部分分隔开来，并非算法语言中的基本符号.

前面例1中采用的是函数过程；例2中采用的是一般过程．这两种过程的区别在于：

(1) 函数过程的计算结果限于一个数值，这个值在过程体中必须赋给过程标识符；一般过程的运算结果可以是一个数值，也可以不是一个数值，这个结果必须在过程体中存放到代表计算结果的形式参数或某个非局部量中去．

(2) 函数过程用函数命名符来调用，这种函数命名符不是一个独立的语句，而可以看作一个函数变量，如果它在表达式中出现，则只能作为最左边的分量来使用．例如，若设$f(x)$为一个函数命名符，则我们不能写

$$a+f(x),$$

而应该写

$$f(x)+a.$$

一般过程用过程语句来调用．过程语句是一种独立的语句，可以象使用其他语句那样来使用它．

应该指出，无论函数过程或一般过程，它们的过程标识符都不能在本过程体中任何表达式里出现．

2．<过程标识符>．过程标识符是给过程取名的．对于函数过程来说，它还将被赋予计算结果的值．同变量标识符一样，过程标识符可以任意选定，但不能同其他标识符重用，除非它们有不相交的作用域．

3．<形式参数部分>．形式参数部分列出计算过程所依赖的全部参数，它的一般形式是

$$(z_1, z_2, \cdots, z_n),$$

其中，圆括号里用逗号分隔的标识符称为形式参数，它们的排列次序是可以任意的．作为形式参数的标识符允许同其他变量标识符相同．形式参数是表示过程所依赖的参数所占有的

位置. 例如，前面例 1 中的形式参数 x、y、z 表示计算公式中自变量 a、b、c, c、d、e 或 e、f、g 所占的位置. 例 2 中形式参数 n, R, s, t 分别表示计算公式中的给定数据个数 50、100、150, 给定数据 a、c、e 和 b、d、f 及计算结果 x、y、z 所占的位置.

过程说明一般都包含形式参数部分，但也有一些过程可以不具有形式参数部分，称为无参数过程. 实际上，无参数过程说明中出现的量可分为两种，一种是该过程说明的过程体内部说明的，另一种是在包含该过程说明的外层分程序中说明的.

[例] 已知方程

$$f(x)=x^3-x^2-x-2=0$$

在区间[0, 3]中有一个实根，试用牛顿迭代公式求出它.

牛顿法求根的迭代公式是

$$x^{(n)}=x^{(n-1)}-\frac{f(x^{(n-1)})}{f'(x^{(n-1)})}, \quad n=1, 2, \cdots.$$

若取迭代初值 $x^{(0)}=0$, 精确度要求为 10^{-5}, 则计算过程可用框图描述如图 21.

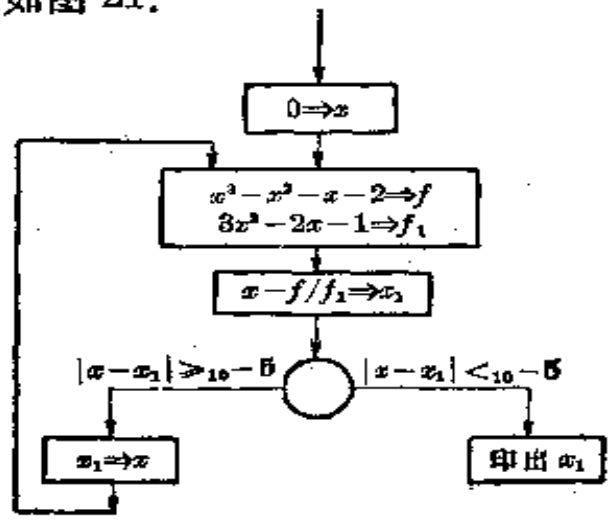


图 21

若把计算函数值 $f=x^3-x^2-x-2$ 及导数值 $f_1=3x^2-2x-1$ 的公式写成过程说明，则计算问题的程序是

```
begin
  real x, x1, f, f1;                                  实型量说明
  procedure ff1;                                      过程类型、过程标识符
    begin                                             } 过程体 } 无参数过程说明
      f:=x↑3-x×x-x-2;
      f1:=3×x×x-2×x-1
    end;
  x:=1;                                               赋值语句
L1: ff1;                                              无参数过程语句
  x1:=x-f/f1;                                         赋值语句
  if #abs(x-x1)≥10-5 then                             } 条件语句
    begin x:=x1; goto L1 end;
  #print(0, 'E14.6', x1)                              输出语句
end 55…5
```

这里，程序中的过程说明是无参数过程的形式。过程体中出现的变量 x, f, f_1 都是非局部量，它们在外层分程序中已经进行了说明，因此，它们在过程体的内外都有意义。由于过程说明是无参数的，所以调用它的过程语句也是无参数的，叫无参数过程语句。程序中语句部分的过程语句 ff_1 就是无参数过程语句。

4. <值部分>. 值部分列出了在过程说明中被处理为值参数的那些形式参数。值部分的一般形式是

$$\textbf{value}\ z_i,\ z_j,\ \cdots,\ z_l.$$

其中，value 是值部分的记号。列于值部分的形式参数 z_i, z_j, …, z_l 称为值参数。没有处理为值参数的形式参数称为换名参数。值部分中值参数的排列次序是可以任意的。

把某些形式参数处理为值参数可以缩短过程的机器执行时间，提高程序质量，这一点我们将在后面过程调用部分举例

说明，所以应当尽可能地把形式参数处理成值参数，即把它们放在值部分．但是根据规定，并非所有的形式参数都能处理成值参数的．譬如，代表一个数组或过程的形式参数以及作为计算结果的形式参数都不能处理成值参数．一般来说，对于作为自变量的形式参数，当它是简单变量时，就可以处理成值参数．

5．<区分部分>．区分部分是指明每一个形式参数性质的部分．常用的区分有六种类型（设 $z_1, z_2, \cdots, z_n$ 为形式参数）：

real $z_1, z_2, \cdots, z_n$;
integer $z_1, z_2, \cdots, z_n$;
boolean $z_1, z_2, \cdots, z_n$;
array $z_1, z_2, \cdots, z_n$;
procedure $z_1, z_2, \cdots, z_n$;
real procedure $z_1, z_2, \cdots, z_n$.

从外表来看，区分与说明很相似，但实际上它们完全是两回事．说明是用来指明程序中用到的诸量的具体性质的；而区分部分则仅仅是过程说明中对形式参数进行分类．在区分部分中，各种区分之间，必须用分号来分隔．各种区分以及每种区分类型中的形式参数，它们排列的先后次序都是可以任意的．

[例] 编写一个计算多项式

$$s=\sum_{i=0}^{n} a_i x^i$$
$$=(\cdots((a_n x+a_{n-1})x+a_{n-2})x+\cdots+a_1)x+a$$

值的函数过程说明．

我们把多项式最高次数 n，自变量 x，存放多项式系数的

数组 a 取作形式参数，则函数过程说明写为

```
real procedure poly(n, x, a);          过程类型，过程标识符，形式参数
  integer n;
  real x;                              } 区分部分
  array a;
begin
  integer i;
  real s;
  s:=0;                                } 过程体
  for i:=n step -1 until 0 do
    s:=s×x+a[i];
  poly:=s
end
```

这里，区分部分 **integer** n; **real** x; **array** a 把过程说明的全部形式参数 n, x, a 分类列出，但不必指明这些形式参数的具体性质，诸如数组标识符 a 的界偶表是什么等.

6．<过程体>. 过程体是过程说明的主要成分，它用语句把计算过程描述出来. 过程体可以由语句或分程序构成，如果过程说明具有形式参数，则这些形式参数应该在过程体中出现. 除此之外，过程体中还可以出现对该过程体而言的局部量或非局部量.

以上六个部分就是一个过程说明的构成成分. 一般来说，编写一个计算过程的过程说明都应该包含这六个部分. 但是，在有些具体问题的过程说明中也可以缺少其中某些部分，例如前面提到的无参数过程的情形，就缺少形式参数部分. 然而，在任何情况下，过程类型的记号、过程标识符以及过程体这三个部分总是不可缺少的.

（二）过程调用

过程说明是用函数命名符或过程语句来调用的. 函数命

名符用来调用函数过程说明，过程语句用来调用一般过程说明；它们的一般形式是

$$A(x_1, x_2, \cdots, x_n).$$

这里，A 是过程标识符；圆括号里用逗号分隔的 $x_1, x_2, \cdots, x_n$ 称为实在参数，它们是被说明过的标识符，可以是简单变量标识符、数组标识符或过程标识符. 如果某一实在参数在相应过程说明中对应的形式参数区分为实型简单变量，且在过程体中作为自变量来使用，则它也允许是一个算术表达式.

必须强调，函数命名符和过程语句的标识符，实在参数的个数、性质、位置必须同被调用的过程说明的标识符，形式参数的个数、性质、位置保持一致.

函数命名符或过程语句是怎样调用过程说明的呢？具体步骤如下：

1. 赋值. 在过程说明中列在值部分的全部形式参数(即值参数)里，赋以过程语句(或函数命名符)中相应实在参数的值或算术表达式的值.

2. 换名. 把过程体中未列在值部分的全部形式参数(即换名参数)，换成过程语句或函数命名符中相应实在参数的标识符或算术表达式.

3. 执行. 将经过赋值、换名后的过程体执行一次.

现在举一个例子来说明这个调用步骤，并结合考察值参数在调用过程中的作用.

[例] 已知复数

$$t = r + is, \ z_k = x_k + iy_k, \quad k = 1, 2, 3,$$

试计算

$$w = u + iv = z_1(z_2 \cdot z_3 + t).$$

我们先编一个复数乘法的一般过程说明. 由于

$$(a_1+ia_2)(b_1+ib_2)=(a_1b_1-a_2b_2)+i(a_1b_2+a_2b_1)$$
$$=c_1+ic_2,$$

若把复数的实部和虚部 a_1, a_2, b_1, b_2, c_1, c_2 取为形式参数，则过程说明可编成

```
procedure CM(a1, a2, b1, b2, c1, c2); 过程类型、过程标识符、形式参数
  real a1, a2, b1, b2, c1, c2; ……区分部分
    begin
      c1:=a1×b1-a2×b2;
      c2:=a1×b2+a2×b1
    end                                  } …过程体
```

显然，这里形式参数 a_1, a_2, b_1, b_2, c_1, c_2 都是换名参数。

如果把这个过程说明放在计算程序的说明部分，那么在程序的语句部分连续用两个过程语句调用它，就可以获得计算结果。现在写出计算程序如下：

```
begin
  说明部分 {
    procedure CM(a1, a2, b1, b2, c1, c2); …过程类型、标识符、形式参数
      real a1, a2, b1, b2, c1, c2; ……区分部分
        begin
          c1:=a1×b1-a2×b2;
          c2:=a1×b2+a2×b1
        end;                             } …过程体
    real x1, y1, x2, y2, x3, y3, r, s, u, v, xh, yh; … 简单变量说明
  语句部分 {
    #read(0, '10', x1, y1, x2, y2, x3, y3, r, s); …… 输入语句
    CM(x2, y2, x3, y3, xh, yh); ……过程语句
    CM(x1, y1, xh+r, yh+s, u, v); …… 过程语句
    #print(0, '2F14.6', u, v) ……输出语句
end 55…5
```

机器执行本程序的次序是，先执行输入语句，然后，依次用两个过程语句调用过程说明，最后，执行输出语句。第一个过程语句 $CM(x_2, y_2, x_3, y_3, x_h, y_h)$ 调用相应过程说明的具体步骤如下：

1．赋值．由于过程说明中形式参数都是换名参数，没有值参数，因此，机器不进行赋值工作．

2．换名．机器把过程体中的全部换名参数用过程语句中相应的实在参数来代替，即

$$x_2\to a_1,\ y_2\to a_2,\ x_3\to b_1,\ y_3\to b_2,\ x_h\to c_1,\ y_h\to c_2.$$

换名以后的过程体成为

begin

$x_h:=x_2\times x_3-y_2\times y_3;$

$y_h:=x_2\times y_3+y_2\times x_3$

end.

3．执行．机器执行赋值、换名后的过程体，就在实在参数 x_h、y_h 中得到了计算问题的中间结果 $z_2\cdot z_3$．

机器执行第一个过程语句后，接着就执行第二个过程语句 $CM(x_1,\ y_1,\ x_h+r,\ y_h+s,\ u,\ v)$，调用相应过程说明的具体步骤是

1．赋值．由于过程说明中形式参数都是换名参数，所以，赋值工作不进行．

2．换名．机器把过程体中的全部换名形式参数用过程语句中相应实在参数来代替，即

$$x_1\to a_1,\ y_1\to a_2,\ x_h+r\to b_1,\ y_h+s\to b_2,\ u\to c_1,\ v\to c_2.$$

换名以后的过程体成为

begin

$u:=x_1\times(x_h+r)-y_1\times(y_h+s);$

$v:=x_1\times(y_h+s)+y_1\times(x_h+r)$

end.

3．执行．机器执行赋值、换名后的过程体，就在实在参数 u、v 中得到计算问题的最终结果 $z_1(z_2\cdot z_3+i)$．

从过程语句调用过程说明的具体步骤中可以看出，第一个过程语句调用过程说明时需要换名十次；第二个过程语句调用过程说明时，除了同样要换名十次外，还要计算

$$x_k+r,\ y_k+s$$

各两次．为什么机器要这样做呢？主要是因为

$$a_1,\ a_2,\ b_1,\ b_2,\ c_1,\ c_2$$

都是换名的形式参数，它们只是某些参变量所占位置的标志，在机器中是不占有存贮单元的，如果不换名，则在计算

$$c_1:=a_1\times b_1-a_2\times b_2$$

和

$$c_2:=a_1\times b_2+a_2\times b_1$$

时就不知道到什么存贮单元里去取 a_1，a_2，b_1，b_2 的值，所以对这类形式参数必须换名．那么，能否不换名而就找到运算分量的值呢？为了解决这个问题，只要对 a_1，a_2，b_1，b_2 都开列出存贮单元，并在执行过程体前就赋以过程语句中相应实在参数的值，这样，a_1，a_2，b_1，b_2 就不需要换名了．为此，列出一个专门的部分——值部分，并把这些形式参数放在这一部分，称为值参数．

引进值参数后，计算程序中的过程说明就写成

```
procedure CM(a1, a2, b1, b2, c1; c2); ……过程类型、标识符、形式参数
  value a1, a2, b1, b2; ……值部分
  real a1, a2, b1, b2, c1, c2; ……区分部分
    begin
      c1:=a1×b1−a2×b2;
      c2:=a1×b2+a2×b1
    end
```

……过程体

从而，机器执行第一个过程语句

$$CM(x_2,\ y_2,\ x_3,\ y_3,\ x_k,\ y_k)$$

时，调用过程说明的具体步骤是

1．赋值．由于过程说明中形式参数 a_1，a_2，b_1，b_2 是值参数，所以机器就把相应的实在参数的值赋给它们，即等价于执行 $a_1:=x_2$; $a_2:=y_2$; $b_1:=x_3$; $b_2:=y_3$;

2．换名．形式参数 c_1、c_2 未列在值部，所以它们是换名参数，因而就换名为相应实在参数，即

$$x_k\to c_1,\ y_k\to c_2.$$

3．执行．最后机器执行赋值、换名后的过程体

begin

$x_k:=a_1\times b_1-a_2\times b_2$;

$y_k:=a_1\times b_2+a_2\times b_1$

end.

从而在实在参数 x_k，y_k 中得到计算问题的中间结果 z_2，z_3．容易看出，这里只有两次换名，四次赋值，而原来却要十次换名．

类似地，机器执行第二个过程语句

$$CM(x_1,\ y_1,\ x_k+r,\ y_k+s,\ u,\ v)$$

时，只要进行两次换名、四次赋值以及计算 x_k+r，y_k+s 各一次，与原来换名十次及计算 x_k+r，y_k+s 各两次相比，工作量要大大节省．

由此看来，把某些形式参数处理成值参数可以缩短程序的执行时间，它是提高程序质量的一个有效方法．

最后，我们举一个包含过程说明及其调用的程序例子．

[例]　给出三组数

$$x_i\quad(i=1,\ 2,\ \cdots,\ 10),$$

$$y_i\quad(i=1,\ 2,\ \cdots,\ 20),$$

$$z_i\quad(i=1,\ 2,\ \cdots,\ 30),$$

试分别求出其按绝对值为最大的数 a, b, c, 并计算

$$R=a+b+c.$$

由于分别求三组数按绝对值为最大的数的计算过程在形式上是相同的，而且，作为计算结果的值仅一个，所以我们把它写成函数过程的形式。另外，为了使编出的过程说明能适用于三个具体的计算过程，所以取如下参变量作为过程说明的形式参数：n——给定数据的个数，P——存放给定数据的数组。这里，形式参数 n 可以处理成值参数。

这样，计算公式 $R=a+b+c$ 中变量 a, b, c 的值，就可以用三个函数命名符调用过程说明来得到。但是，根据赋值语句中函数命名符只能作为表达式最左边分量使用的规定，这个计算公式就得用三个赋值语句来写出。

现在写出这个计算问题的程序：

```
begin
  real procedure max(n, P); 过程类型、过程标识符、形式参数
    value n;                                  ……值部分
    integer n;                                }…区分部分
    array P;
      begin                                   }局部量说明
        real s;                               }  过程体
        integer i;                            }  过程说明
        s:=P[1];
        for i:=2 step 1 until n do
          if #abs(P[i])>#abs(s) then s:=P[i];
        max:=s
      end;
  real a, b, c, R;                            ……实型量说明
  array x[1:10], y[1:20], z[1:30];            ……数组说明
    #read(0, '10', x, y, z);                  ……输入语句
    a:=max(10, x);                            ……赋值语句
    b:=max(20, y);                            ……赋值语句
    c:=max(30, z); R:=a+b+c;                  ……赋值语句
    #print(0, '10', a, b, c, R)               ……输出语句
end 55…5
```

三　常用算法的标准过程

在使用计算机解算数学问题时，某些数值计算方法是常被用到的，例如，求解线性代数方程组的消去法，高斯数值求积法，求解常微分方程组的龙格—库塔法等等．对于这些常用的计算方法的程序，如果对每一个应用它的计算问题都要由算题人员自行编写，这将是很不经济的．为此，可以选择一些常用的计算方法，并预先确定某些规格、标准和要求，把它们编成过程说明的形式，称为常用算法的标准过程（第三章中列出了部分常用算法的标准过程）．算题人员需要用到它们的时候，只要把它们抄写在计算程序的说明部分，并按照标准过程的要求，写出调用它的过程语句或函数命名符就可以了．

现在举一个调用常用算法标准过程的程序例子．

[例]　试求图 22 电路中各电阻上的端电压

$$v_i \quad (i=1,\ 2,\ 3,\ 4,\ 5,\ 6).$$

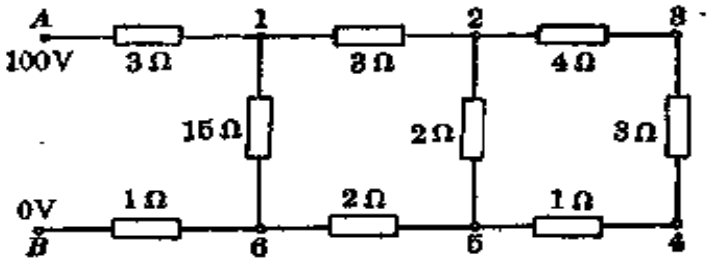


图　22

根据电学知识，电路图中各节点上的电流堆积应为 0，如对节点 1 应有

$$i_{s1}+i_{21}+i_{61}=\frac{100-v_1}{3}+\frac{v_2-v_1}{3}+\frac{v_6-v_1}{15}=0,$$

即

$$11v_1-5v_2-v_6=500.$$

类似地，列出其他节点上的公式则得

$$\begin{cases} 11v_1 - 5v_2 - v_6 = 500, \\ 4v_1 - 13v_2 + 3v_3 + 6v_5 = 0, \\ 3v_2 - 7v_3 + 4v_4 = 0, \\ v_3 - 4v_4 + 3v_5 = 0, \\ v_2 + 2v_4 - 4v_5 + v_6 = 0, \\ 2v_1 + 15v_5 - 47v_6 = 0. \end{cases}$$

若表示为矩阵形式，则有

$$KV = F,$$

其中

$$K = \begin{bmatrix} 11 & -5 & 0 & 0 & 0 & -1 \\ 4 & -13 & 3 & 0 & 6 & 0 \\ 0 & 3 & -7 & 4 & 0 & 0 \\ 0 & 0 & 1 & -4 & 3 & 0 \\ 0 & 1 & 0 & 2 & -4 & 1 \\ 2 & 0 & 0 & 0 & 15 & -47 \end{bmatrix},$$

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_6 \end{bmatrix}, \quad F = \begin{bmatrix} 500 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

这是含有六个未知量的线性代数方程组的求解问题. 我们可以选择常用算法标准过程："严格选大元消去法解线性代数方程组标准算法"(见第三章五)，来调用求解之.

整个计算程序写为

```
begin
  说明部分 { array K[1:6, 1:6], F, x1, x2[1:6]; ……数组说明
             procedure GJ(n, a, b, eps, ik, jk);
               value n, eps; ……值部分
               integer n;
               real eps;
               array a, b, ik, jk;
                 begin
                   integer i, j, k, l, i0, j0;
                   real c, d;
                   ……
                 end;
             } 区分部分 + 过程体 = 标准过程说明
  语句部分 { #read(0, '10', K, F); ……输入语句
             GJ(6, K, F, 10-3, x1, x2); ……过程语句
             #print(0, 'F14.6', F) ……输出语句
             }
end 55…5
```

这里，程序中说明部分的“标准过程说明”就是“严格选大元消去法解线性代数方程组标准算法”的过程说明，是由第三章五照抄进来的，其中，语句部分中的过程语句

$$GJ(6,\ K,\ F,\ {}_{10}-3,\ x_1,\ x_2)$$

是按照“使用说明”的要求写出的．机器执行这个语句时，就调用标准过程说明，从而在代表计算结果的实在参数 F 中得到方程组的解．这里，应该注意，过程语句的标识符，实在参数的位置、个数、性质必须同标准过程说明的标识符，形式参数的位置、个数、性质保持一致．

从上面的例子看出，选用常用算法标准过程，可以减少程序设计的工作量，所以，应该尽量采用这种程序设计的方法．

另外，为了进一步方便算题人员使用常用算法标准过程，机器上还建立了一个“常用算法标准过程库”，即预先把若干个常用算法的标准过程存放到机器的外存贮器里，构成一个

标准过程库，库中的常用算法标准过程叫库过程. 目前，第三章中带有(*)的标准程序，已经列入"常用算法标准过程库"中，算题人员如果需要选用某个库过程，就只要在程序的说明部分给出如下形式的信息：

ku<库过程标识符>

或

ku<库过程标识符><形式参数部分>.

机器遇到这个信息，就把相应库过程说明调到计算程序的说明部分，这样，语句部分的过程语句(或函数命名符)就象以往一样调用该过程说明. 例如，上面的例子调用库过程"严格选大元消去法解线性代数方程组"的方法，整个程序可以写成

```
begin
  array K[1:6, 1:6], F, x1, x2[1:3];
  ku GJ; (或 ku GJ(n, a, b, eps, ik, jk); )
    #read(0, '10', K, F);
    GJ(6, K, F, 10-3, x1, x2);
    #print(0, 'F14.6', F)
end 55…5
```

其中，**ku** GJ 或 **ku** $GJ(n, a, b, \text{eps}, ik, jk)$是库过程说明；$GJ(6, K, F, 10-3, x_1, x_2)$ 是调用库过程说明的过程语句.

习 题 五

1. 试分别用两种过程说明的形式描述

$$f(x)=x^2+\sqrt{1+2x+3x^2},$$

然后，分别编出下面计算公式的程序：

$$\alpha=\frac{\sin y}{y^4+\sqrt{1+2y^2+3y^4}},\ \beta=\alpha^2-1+4z^2+\sqrt{1+4z+12z^2}.$$

2. 编写计算 n 阶矩阵 (a_{ij}) $i, j=1, 2, \cdots, n$, 主对角线元素之和的函数过程说明.

3. 指出下面过程说明中的错误:

```
real procedure Mv(n, a, b, c);
    value n, a, b, c;
  begin real c;
    integer a, b[1:n];
      c:=0;
      for i:=1 step 1 until n do
        c:=c+a[i]×b[i]
  end
```

4. 下面过程说明中的形式参数能否处理为值参数? 为什么?

```
procedure plus(x, y, s);
  real x, y, s;
    begin
      x:=x+1; y:=y+2; s:=s+3
    end.
```

5. 指出下面过程说明中的局部量与非局部量:

```
real procedure f(a, b, n, p, q);
  real p, q, n;
  array a, b;
    begin
      real i, c;
        c:=0;
        i:=n×(n+1)/2;
        begin
          real n;
            for n:=1 step 1 until i do
              c:=(a[n]×p+b[n]×q)↑n+c
        end;
```

$f:=c$

end.

6．编写计算多项式值

$$s=\sum_{i=0}^{n}a_ix^i$$

的过程说明，然后计算：

$$w=\sum_{k=0}^{2m+1}c_k(y^2+s^2)^k,$$

其中

$$m<50.$$

7．设函数：

$$f(x)=\begin{cases}2-1.5x, & |x|\leqslant 1,\\ x/2, & |x|>1.\end{cases}$$

试计算它在区间$[a, b]$上傅里叶展开系数：

$$A_k=\frac{1}{n}\sum_{i=0}^{2n-1}f\left(a+i\,\frac{b-a}{2n}\right)\cdot\cos\left(\frac{2k\pi}{b-a}\left(a+i\,\frac{b-a}{2n}\right)\right),$$

$$B_k=\frac{1}{n}\sum_{i=0}^{2n-1}f\left(a+i\,\frac{b-a}{2n}\right)\cdot\sin\left(\frac{2k\pi}{b-a}\left(a+i\,\frac{b-a}{2n}\right)\right),$$

$$k=0, 1, \cdots, n-1,\ n<100.$$

第六节　程序的例子

[例 1]　给定半径 $R_1=110$ 厘米的圆铁板一块，若以其圆心为中心，逐次以 $R_2=1, 2, \cdots, 100$ 为半径，对原铁板进行圆切割，试分别求留下部分圆环的面积.

显然，需要求的圆环面积一共有 100 个，为此，用 S 代表具有 100 个下标变量的数组标识符，以保存这 100 个圆环面积的值. 另外，由于这 100 个圆环面积是随切去的圆铁板半径 R_2 的改变而改变的，所以，取 R_2 为逐次执行下列圆环面积公式的循环变量：

$$S[R_2]=3.1416\times(12100-R_2\times R_2)$$

$$R_2=1,\ 2,\ \cdots,\ 100.$$

从而，计算的程序是

```
begin
    integer R2;
    array S[1:100];
  for R2:=1 step 1 until 100 do
      S[R2]:=3.1416×(12100-R2×R2);
    #print(0, 'F14.6', S)
end 55…5
```

[例 2]　已知一个光学系统(图 23)的结构参数：

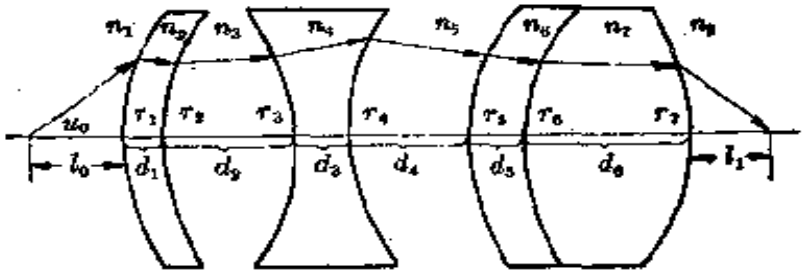


图　23

$r_k(k=1,\ 2,\ 3,\ \cdots,\ 7)$是各镜面的曲率半径；

$d_k(k=1,\ 2,\ \cdots,\ 6)$是镜面之间的距离；

$n_k(k=1,\ 2,\ \cdots,\ 8)$是介质的折射率；

u_0是数值孔径；l_0是物距，试求截距l_1。

根据光学原理，截距l_1可由如下循环公式求得：

$$i=\frac{l_2-r_k}{r_k}u_1,\ \text{第一次计算时，取}\ l_2=l_0,\ u_1=u_0,$$

$$i_1=\frac{i\cdot n_k}{n_{k+1}},$$

$$u_1=u_0+i-i_1,$$

$$l_1=r_k\left(1+\frac{i_1}{u_1}\right),$$

$$l_2=l_1-d_k,$$

$$k=1,\ 2,\ \cdots,\ 7.$$

这个计算问题的程序可以写为

```
begin
  real l2, u1, i, i1, l1, u0, l0;
  integer k;
  array r[1:7], d[1:6], n[1:8];
  #read(0, '10', r, d, n, l0, u0);
  l2:=l0; u1:=u0;
  for k:=1 step 1 until 7 do
    begin i:=(l2-r[k])×u1/r[k];
          i1:=i×n[k]/n[k+1];
          u1:=u0+i-i1;
          l1:=r[k]×(1+i1/u1);
          if k≠7 then l2:=l1-d[k]
    end;
  #print(0, 'F14.6', l1)
end 55…5
```

[例 3] 许多计算问题经常需要作矩阵相乘的运算，试编写具有如下两种功能的矩阵相乘的过程说明：

(1) $[C]_{m\times l}=[A]_{m\times n}\cdot[B]_{n\times l}$;

(2) $[C]_{m\times l}=[A]^{T}_{n\times m}\cdot[B]_{n\times l}$.

设矩阵 $[A]$、$[B]$、$[C]$ 分别用数组标识符 A、B、C 表示，m、n、l 分别是它们的行、列数，若取 ind 为区分两种功能的形式参数(假定，$ind=0$ 时完成(1)；$ind=1$ 时完成(2))，则过

程说明可编写为：

```
procedure MTM(m, n, l, A, B, C, ind);
  value m, n, l, ind; integer m, n, l, ind;
  array A, B, C;
  begin integer i, j, k;
    for i:=1 step 1 until m do
      for j:=1 step 1 until l do
        begin C[i, j]:=0;
          for k:=1 step 1 until n do
            C[i, j]:=(if ind=0 then A[i, k]*B[k, j]
              else A[k, i]*B[k, j])+C[i, j]
        end
  end;
```

[例 4] 某计算问题具有 n 个原始数据，它们都是不超过三位有效数字的正整数. 为了节省内存单元，拟采用一个单元 x 存放三个数 a, b, c 的紧凑格式进行存贮，即：

$$x=0000\underbrace{\times\times\times}_{a}\underbrace{\times\times\times}_{b}\underbrace{\times\times\times}_{c}0$$

这样，程序中要调用数据 a、b 或 c 时，必须先把存放它们的紧凑格式单元 x 进行分解. 试编写这个分解程序.

由于紧凑格式单元的分解是程序中经常要进行的工作，所以把它编成过程说明的形式. 为此，若把取紧凑格式单元的变量 x 及分解结果的单元变量 a、b、c 作为形式参数，则过程说明可编写为：

```
procedure dec(x, a, b, c); value x; real x, a, b, c;
  begin real S;
    S:=x*10^3; a:=#entier(S);
```

```
        S:=(S-a)*10^3; b:=#entier(S);
        S:=(S-b)*10^3; c:=#entier(S+0.5)
    end;
```

[例 5] 工程计算中常常遇到一些复杂的函数关系，很难用一个公式来表达. 假定已经知道了在自变量 $x_i(i=1, 2, \cdots, n)$ 上相应的函数值 $y_i(i=1, 2, \cdots, n)$，且设 x_i 是从小到大的次序排列的，则可以编写一个线性插值公式的过程说明，求出任一个 $x_p \in [x_1, x_n]$ 上的函数值 y_p.

若用标识符 x、y 分别表示给定的 n 个自变量值和相应的函数值，则其分量分别是

$x[1], x[2], \cdots, x[n]$ 和 $y[1], y[2], \cdots, y[n]$.

现在的问题是要寻找一个下标值 $i_0 \in [1, n]$，使

$$x_{i_0} \leqslant x_p \leqslant x_{i_0+1}.$$

如果找到了这个 i_0，就可以利用如下的线性插值公式：

$$y_p = y_{i_0} + \frac{y_{i_0+1} - y_{i_0}}{(x_{i_0+1} - x_{i_0})}(x_p - x_{i_0})$$

求出 y_p.

如何找满足条件 $x_{i_0} \leqslant x_p \leqslant x_{i_0+1}$ 的 i_0 呢？可以利用循环语句，从 $i=1$ 开始直到 $n-1$ 为止，逐个判别条件是否满足，若满足，则当时的 i 值即为所找的 i_0.

为了使这个计算过程具有通用性，我们把它编成函数过程说明的形式，并把标识符 x_p, y_p, n, x, y 取为形式参数. 从而有

```
real procedure Li(n, x_p, x, y);
        value n, x_p;
        integer n;
        real x_p;
```

```
        array x, y;
    begin
        integer i; real y_p;
      for i:=1 step 1 until n-1 do
        if x[i]≤x_p∧x_p≤x[i+1] then
          begin y_p:=y[i]+(y[i+1]-y[i])
            ×(x_p-x[i])/(x[i+1]-x[i]); goto Lend;
  L:L i:=y_p end.
```

[例 6] 给定 n 个数 $d_1, d_2, \cdots, d_n$(设 $n\leqslant 100$)及其对应的 n 个 n 维向量 $A_1, A_2, \cdots, A_n$. 试按从小到大的次序对 d_i 进行排队，排队时，要求相应的 n 个 n 维向量也随 d_i 的调整跟着一起调整.

这个问题可以看作是进行 n 次在 j 个数中寻找最小数的过程，我们采用逐个比较的办法来实现，需注意的是，其中 j 的个数将随寻找步骤的逐次进行而逐次减 1.

若设 d_i 存于数组 $d[1:n]$、A_i 按列存于数组 $A[1:n, 1:n]$，则整个程序可以写为：

```
begin
    integer n;
    #read(0, '10', n);
  begin
    integer i, j, k; real p;
    array d(100)[1:n], A(10000)[1:n, 1:n];
    #read(0, '10', d, A);
    for i:=1 step 1 until n do
      begin k:=i; p:=d[i];
        for j:=i+1 step 1 until n do
```

```
            if d[j]<p then begin k:=j; p:=d[j] end;
          if k≠i then begin d[k]:=d[i]; d[i]:=p;
          for j:=1 step 1 until n do
            begin P:=A[j, i]; A[j, i]:=A[j, k];
              A[j, k]:=P end end
        end
      end
    end 55…5
```

[例 7] 已知脉冲衰减计算公式:

$$F(x, y, z)=x+\sum_{n=1}^{1200}\frac{2}{n\pi}\sin n\pi x\cdot\cos 2n\pi y\cdot e^{-n\sqrt{\pi}z},$$

试对参数

$x=0.02\xrightarrow{(0.02)}0.50;$

$y=0\xrightarrow{(0.01)}0.50;$

$z=0,\ 0.005,\ 0.01,\ 0.05,\ 0.1,\ 0.2,\ 0.4, 0.6, 0.8, 1.0$

计算出相应的函数值 $F(x, y, z)$.

显然,这是一个四层相套的循环计算过程,它的计算工作量是较大的,为了提高程序质量,可以利用关系式:

$$\sin n\pi x=\sin(n+100)\pi x=\cdots=\sin(n+1100)\pi x,$$

$$\cos 2n\pi y=\cos 2(n+100)\pi y=\cdots=\cos 2(n+1100)\pi y,$$

以减少 $\sin n\pi x$, $\cos 2n\pi y$ 的计算量.

另外,对级数中的 $e^{-n\sqrt{\pi}z}$,也可根据指数函数的特点,把中间结果充分利用起来,以节省运算工作量.

整个计算的程序可以写为

```
begin
  real x, y, z, w, y1;
  integer i, j, k, m, n;
```

```
array F[0:50], x1, z0[1:100], z1, z2[1:1200];
  for i:=1 step 1 until 10 do
    begin
      switch p:=L1, L2, L3, L4, L5,
        L8, L5, L6, L6, L6;
      goto p[i];
      L1: for n:=1 step 1until 1200 do
            z1[n]:=1; goto L7;
      L2: for n:=1 step 1 until 1200 do
            z1[n]:=#exp(-0.005×#sqrt(n));
            goto L7;
      L3: for n:=1 step 1 until 1200 do
            z1[n]:=z1[n]×z2[n]; goto L7;
      L4: for n:=1 step 1 until 1200 do
            z1[n]:=#exp(-0.05×#sqrt(n));
            goto L7;
      L5: for n:=1 step 1 until 1200 do
            begin z2[n]:=z1[n];
            z1[n]:=z1[n]×z1[n] end; goto L7;
      L6: for n:=1 step 1 until 1200 do
            z1[n]:=z1[n]×z2[n];
      L7: for k:=1 step 1 until 100 do
            begin z0[k]:=0;
              for m:=0 step 1 until 11 do
                begin z:=100×m+k;
                      z0[k]:=z0[k]+z1[z]/z
                end
```

```
            end;
        for x:=0.02 step 0.02 until 0.50 do
          begin
          w:=3.141593×x;
          for n:=1 step 1 until 100 do
          x1[n]:=#sin(n×w);
          for j:=0 step 1 until 50 do
          begin y:=0.01×j; y1:=6.283186×y;
            F[j]:=0;
            for n:=1 step 1 until 100 do
            F[j]:=F[j]+x1[n]×
            #cos(n×y1)×z0[n];
            F[j]:=x+2×F[j]/3.141593
          end;
          #print(0,'2Hi=,F7.4,5x,2Hx=,F5.2,
            5x,/,F14.8', i, x, F)
          end
      end
  end 55…5
```

[例 8] 结构力学计算问题中，常常需要求解含有大量未知数的线性代数方程组

$$AX=B,$$

其中，$A=(a_{ij})(i,\ j=1,\ 2,\ \cdots,\ n)$ 是系数矩阵；$B=(b_{ir})$、$X=(x_{ir})(i=1,\ 2,\ \cdots,\ n;\ r=1,\ 2,\ \cdots,\ m)$ 分别是右端项和未知量.

在很多情况下，A 还是对称、正定的带状稀疏矩阵，例如，下面的五阶矩阵就具有这种形式：

$$A=\begin{bmatrix} 3 & & (\text{对称}) & & \\ 1 & 5.2 & & & \\ 0 & 0 & 3.4 & & \\ 0 & 0.2 & -1.3 & 2.5 & \\ 0 & 0 & 2 & 0 & 6 \end{bmatrix}.$$

试编写一个求解这类线性方程组的过程说明.

对于这类方程组，可以采用"改进的平方根算法"求解，步骤是：

1．对矩阵 A 进行分解：

$$A=LDL^T,$$

其中，L 是 n 阶下三角矩阵，L^T 是 L 的转置矩阵，D 是对角矩阵，且 $D=\mathrm{diag}(1/l_{11},\ 1/l_{22},\ \cdots,\ 1/l_{nn})$，$L$ 的元素 l_{ij} 由下式计算：

$$l_{ij}=a_{ij}-\sum_{s=1}^{j-1}\frac{l_{is}\cdot l_{js}}{l_{ss}} \qquad (i=1,\ 2,\ \cdots,\ n;\ j=1,\ 2,\ \cdots,\ i).$$

A 被分解后，原方程就化为等价的两个方程组：

$$LDY=B \quad \text{和} \quad L^TX=Y.$$

2．由方程组 $LDY=B$，求出中间变量 Y，其元素 y_{ir} 的计算公式是

$$y_{ir}=b_{ir}-\sum_{s=1}^{i-1}\frac{l_{is}\cdot y_{sr}}{l_{ss}} \qquad (i=1,\ 2,\ \cdots,\ n;\ r=1,2,\cdots,m).$$

3．由方程组 $L^TX=Y$，求出最终解 X，其元素 x_{ir} 的计算公式是：

$$x_{ir}=\left(y_{ir}-\sum_{s=i+1}^{n}l_{si}\cdot x_{sr}\right)\Big/l_{ii}$$

$$(i=n,\ n-1,\ \cdots,\ 1;\ r=1,\ 2,\ \cdots,\ m).$$

在程序编写时，为了减少运算量，上述计算公式均采用按行的运算次序进行．另外，为了节省存贮量，把 A 中下三角

部分的元素按行从第1个非零元素开始逐个压缩存放于一维数组 $A[1:v_a]$ 中(v_a 为下三角部分属于变带区域里的元素总数. 这里,变带区域指的是由 A 中每一行第一个非零元素同主对角线元素围成的部分),A 中每一行中主对角线位置上的元素在 $A[1:v_a]$ 中的序号存于数组 $d[0:n]$(约定 $d[0]=0$). 例如,上面例子中的矩阵存贮形式是:

$A[1:10]=(3,\ 1,\ 5.2,\ 3.4,\ 0.2,\ -1.3,\ 2.5,\ 2,\ 0,\ 6)$

$d[0:5]=(0,\ 1,\ 3,\ 4,\ 7,\ 10)$.

矩阵 A 按照这种"一维变带存贮"后,其任一变带区域里的元素 a_{ij} 在 $A[1:v_a]$ 中的序号$\langle a_{ij}\rangle$及任一行第1个非零元素所在列的列序号 $\langle m_i\rangle$,分别可用下面公式算出:

$$\langle a_{ij}\rangle = d[i]-i+j,$$

$$\langle m_i\rangle = i-(d[i]-d[i-1])+1.$$

按照上面约定,则可以把程序写为:

```
procedure LDLT(n, m, A, d, B); value n, m;
    integer n, m; array A, d, B;
  begin integer i, di, mi, s, j, dj, mj, ij, r, is; real c;
    for i:=1 step 1 until n do
      begin di:=d[i]-i; mi:=d[i-1]-di+1;
        for s:=mi step 1 until i-1 do
          begin C:=A[di+s]/A[d[s]];
            for j:=s+1 step 1 until i do
              begin dj:=d[j]-j; mj:=d[j-1]-dj
                +1; ij:=di+j;
                if s≥mj then
                  A[ij]:=A[ij]-C*A[dj+s]
              end;
```

```
            for r := 1 step 1 until m do
              B[i, r] := B[i, r] - C*B[s, r]
          end
      end;
    for i := n step -1 until 1 do
      begin di := d[i] - i; mi := d[i-1] - di + 1;
        for r := 1 step 1 until m do
          B[i, r] := B[i, r]/A[d[i]];
        for s := mi step 1 until i - 1 do
          begin is := di + s;
            for r := 1 step 1 until m do
              B[s, r] := B[s, r] - A[is]*B[i, r]
          end
      end
end;
```

第 三 章

常用数值方法的标准程序

在使用电子数字计算机解算实际计算问题时，为了避免编写程序工作的重复劳动，一般都选用常用数值方法的标准程序. 这里选编部分常用数值方法的标准程序，写成过程说明的形式，供算题人员选用. 其中，带有记号(*)的程序已存于 709 机和 TQ-16 机的"常用算法标准程序库"中，使用者可按调用库过程的方法直接调用它们.

一 一元三点插值(*)

(一)功能

对于给定的一元函数 $y=f(x)$ 的 n 个节点值 $x_1<x_2<\cdots<x_n$，相应的函数值 y_1, y_2, $\cdots$, y_n，程序选取最靠近插值点 u 的相邻三个节点值 x_i, x_{i+1}, x_{i+2} 及相应的函数值 y_i, y_{i+1}, y_{i+2}，用拉格朗日三点插值公式:

$$v=\sum_{k=i}^{i+2}\left(\prod_{\substack{j=i\\ j\neq k}}^{i+2}\frac{x-x_j}{x_k-x_j}\right)y_k$$

求出插值点 u 所对应的函数值 v.

(二)使用说明

语句形式: Lag1(n, u, x, y, v).

其中，n 表示节点个数，值参数；u 表示插值点，值参数；$x[1:n]$, $y[1:n]$ 分别表示节点值和相应函数值；v 表示插值

结果.

(三)程序

```
procedure Lag 1 (n, u, x, y, v); value n, u;
    integer n; real u, v; array x, y;
  begin integer i; real x0, x1, x2;
    for i:=1 step 1 until n-3 do
      if u≤x[i+1] then goto L; i:=n-2;
L: if i≠1∧u-x[i]<x[i+1]-u then i:=i-1;
    x0:=x[i]; x1:=x[i+1]; x2:=x[i+2];
  v:=(u-x1)*(u-x2)/((x0-x1)*(x0-x2))*y[i]
      +(u-x0)*(u-x2)/((x1-x0)*(x1-x2))*y[i+1]
      +(u-x0)*(u-x1)/((x2-x0)*(x2-x1))*y[i+2]
  end;
```

二　二元三点插值(*)

(一)功能

对于给定二元函数 $z=f(x, y)$ 的 x 坐标，y 坐标的节点值 $x_1<x_2<\cdots<x_n$，$y_1<y_2<\cdots<y_m$，相应的函数值 $z_{ij}(i=1, 2, \cdots, n;\ j=1, 2, \cdots, m)$，程序选取最靠近插值点$(u, v)$的相邻九个节点值 $x_r(r=i, i+1, i+2)$、$y_s(s=j, j+1, j+2)$及其相应的函数值 $z_{rs}(r=i, i+1, i+2;\ s=j, j+1, j+2)$，用拉格朗日三点插值公式：

$$W=\sum_{r=i}^{i+2}\sum_{s=j}^{j+2}\left(\prod_{\substack{k=i\\k\neq r}}^{i+2}\frac{x-x_k}{x_r-x_k}\right)\left(\prod_{\substack{l=j\\l\neq s}}^{j+2}\frac{y-y_l}{y_s-y_l}\right)z_{rs}.$$

求出插值点(u, v)所对应的函数值 W.

(二)使用说明

语句形式：Lag 2(n, m, u, v, x, y, z, w)。

其中，n, m 分别表示 x, y 方向的节点个数，值参数；u, v 分别表示 x, y 方向插值点的值，值参数；$x[1:n]$, $y[1:m]$ 分别表示 x, y 方向的节点值；$z[1:n, 1:m]$表示节点上的函数值；w 表示插值结果.

(三)程序

```
procedure Lag 2 (n, m, u, v, x, y, z, w); value n, m,
    u, v; integer n, m; real u, v, w; array x, y, z;
  begin integer i, j, ii, jj; real x0, x1, x2, y0, y1,
      y2; array ax, ay[0:2];
    for i:=1 step 1 until n-3 do
      if u≤x[i+1] then goto L1; i:=n-2;
L1: for j:=1 step 1 until m-3 do
      if v≤y[j+1] then goto L2; j:=m-2;
L2: if i≠1∧u-x[i]<x[i+1]-u then i:=i-1;
    if j≠1∧v-y[j]<y[j+1]-v then j:=j-1;
    x0:=x[i]; x1:=x[i+1]; x2:=x[i+2];
    y0:=y[j]; y1:=y[j+1]; y2:=y[j+2];
    ax[0]:=(u-x1)*(u-x2)/((x0-x1)*(x0-x2));
    ax[1]:=(u-x0)*(u-x2)/((x1-x0)*(x1-x2));
    ax[2]:=(u-x0)*(u-x1)/((x2-x0)*(x2-x1));
    ay[0]:=(v-y1)*(v-y2)/((y0-y1)*(y0-y2));
    ay[1]:=(v-y0)*(v-y2)/((y1-y0)*(y1-y2));
    ay[2]:=(v-y0)*(v-y1)/((y2-y0)*(y2-y1));
    w:=0;
  for ii:=0, 1, 2 do
    for jj:=0, 1, 2 do
    w:=w+ax[ii]*ay[jj]*z[i+ii, j+jj]
```

end;

三　龙贝方法计算单重积分(*)

(一)功能

对于积分

$$I=\int_a^b f(x)dx,$$

程序采用龙贝方法求值，公式是

$$T_{0k}=\frac{(b-a)}{2^{k+1}}\left[f(a)+2\sum_{i=1}^{2^k-1}f\left((b-a)\frac{i}{2^k}+a\right)+f(b)\right],$$

$$k=0,\ 1,\ 2,\ \cdots$$

$$T_{mk}=\frac{4^m\cdot T_{m-1\,k+1}-T_{m-1\,k}}{4^m-1},\quad m=1,\ 2,\ 3,\ \cdots$$

$$k=0,\ 1,\ 2,\ \cdots$$

$$\lim_{\substack{m\to\infty\\ k\text{固定}}}T_{mk}=\lim_{\substack{k\to\infty\\ m\text{固定}}}T_{mk}=I.$$

其中，T_{0k}，T_{1k} 分别为积分步长等于 $(b-a)\cdot 2^{-k}$ 时的梯形公式和辛甫生公式。

(二) 使用说明

函数命名符形式：Romberg(*fct*, *a*, *b*, *n*, *t*).

其中，*fct* 表示计算被积函数 $f(x)$的函数过程，由使用者自行编写；*a* 为积分下限、*b* 为积分上限，*n* 表示所需要的精确度，一般取 $n=4$、5 即可，*a*、*b*、*n* 都是值参数；*t*[1:*n*+1]表示所需的一组工作单元。

(三)程序

```
real procedure Romberg (fct, a, b, n, t);
  value a, b, n; real a, b; integer n;
  array t; real procedure fct;
    begin real l, u, m, u1, u2; integer f, h, j, nn;
```

```
l:=b-a; u1=fct(a); u2:=fct(b);
t[1]:=(u1+u2)/2; nn:=1;
for h:=1 step 1 until n do
  begin u:=0 m:=l/(2×nn);
    for j:=1 step 2 until 2×nn-1 do
      u:=fct(a+j×m)+u;
    t[h+1]:=(u/nn+t[h])/2; f:=1;
    for j:=h step -1 until 1 do
      begin f:=4×f;
        t[j]:=t[j+1]+(t[j+1]-t[j])/
          (f-1)
        end;
      nn:=2×nn
    end;
  Romberg:=t[1]×l
end;
```

四　高斯法计算二重积分、三重积分

(一)功能

本程序用高斯积分法计算二重、三重积分：

$$GI_2=\int_a^b\int_{g_1(x)}^{g_2(x)} f(x,\ y)\,dy\,dx,$$

$$GI_3=\int_a^b\int_{g_1(x)}^{g_2(x)}\int_{p_1(x,y)}^{p_2(x,y)} f(x,\ y,\ z)\,dz\,dy\,dx.$$

其中，a、b 是有界常数，$g_1(x)$，$g_2(x)$，$p_1(x,\ y)$，$p_2(x,\ y)$分别是关于 x 和 x、y 的有界单值连续函数，f 在积分区域内连续有界。

求 GI_2, GI_3 的计算公式是：

$$GI_2\approx(b-a)\sum_{i=1}^{m}\Big[u_i(b_1-a_1)\sum_{j=1}^{m}u_j f(a+(b-a)t_i,\ a_1+(b_1-a_1)t_j)\Big];$$

$$GI_3\approx(b-a)\Big\{\sum_{i=1}^{m}u_i(b_1-a_1)\Big[\sum_{j=1}^{m}u_j(b_2-a_2)\times\sum_{k=1}^{m}u_k f(a+(b-a)t_i,\ a_1+(b_1-a_1)t_j,\ a_2+(b_2-a_2)t_k)\Big]\Big\}.$$

其中，$a_1=g_1(a+(b-a)t_i)$，$b_1=g_2(a+(b-a)t_i)$，

$a_2=p_1(a+(b-a)t_i,\ a_1+(b_1-a_1)t_j)$，

$b_2=p_2(a+(b-a)t_i,\ a_1+(b_1-a_1)t_j)$，

t_i 和 u_i 是 [0, 1] 区间的高斯积分公式的节点和权系数，它可以用高斯-勒让德求积公式的节点 z_i 和权系数 w_i 按

$$t=(z+1)/2,\quad u=w/2$$

求得.

(二)使用说明

语句形式：

二重积分：$GI2(m,\ a,\ b,\ g_1,\ g_2,\ f,\ t,\ u)$；

其中，m 表示积分节点数，值参数；a、b 表示外层积分限，值参数；$g_1(x)$、$g_2(x)$表示求内层积分限的函数过程；$f(x,\ y)$表示求被积函数值的函数过程；$t[1:m]$、$u[1:m]$表示积分节点和权系数. $g_1(x)$, $g_2(x)$, $f(x,\ y)$三个函数过程由用户自编，t 和 u 的值可查附表提供.

三重积分：$GI3(m,\ a,\ b,\ g_1,\ g_2,\ p_1,\ p_2,\ f,\ t,\ u)$；

其中，m、a、b、$t[1:m]$、$u[1:m]$的含义同二重积分. $g_1(x)$、$g_2(x)$、$p_1(x,\ y)$、$p_2(x,\ y)$ 分别表示求积分限的函数过程；

$f(x, y, z)$表示求被积函数值的函数过程，均由用户自编. t 和 u 的值可查附录提供.

(三)程序

二重积分:

```
real procedure GI2(m, a, b, g1, g2, f, t, u);
  value m, a, b;
    real a, b; integer m; array t, u;
      real procedure g1, g2, f;
  begin real a1, b1, c, e1, e2, x; integer i, j;
    e1:=0; c:=b-a;
    for i:=1 step 1 until m do
      begin x:=a+c*t[i]; a1:=g1(x);
        b1:=g2(x)-a1; e2:=0;
        for j:=1 step 1 until m do
          e2:=f(x, a1+b1*t[j])*u[j]+e2;
        e1:=b1*e2*u[i]+e1
      end;
    GI1:=c*e1
  end;
```

三重积分:

```
real procedure GI3(m, a, b, g1, g2, p1, p2, f, t, u);
  value m, a, b;
    integer m; real a, b; array t, u;
      real procedure g1, g2, p1, p2, f;
  begin integer i, j, k; real a1, a2, b1, b2, c, e, e1,
    e2, x, y;
    e:=0; c:=b-a;
```

附表　高斯-勒让德求积公式的节点 x 和权系数 w 表

（x_i 关于零对称，绝对值相同的 x_i 所对应的 w_i 相等）

m	$\pm x_i$	w_i
1	0	2
2	0.57735 02691 89629	1.0
3	0.77459 66692 41483 0	0.55555 55555 55556 0.88888 88888 88889
4	0.86113 63115 94053 0.33998 10435 84856	0.34785 48451 37454 0.65214 51548 62546
5	0.90617 98459 38664 0.53846 93101 05683 0	0.23692 68856 56189 0.47862 86704 99366 0.56888 88888 88889
6	0.93246 95142 03152 0.66120 93864 66265 0.23861 91860 83197	0.17132 44923 79170 0.36076 15730 48193 0.46791 39345 72691
7	0.94910 79123 42759 0.74153 11855 99394 0.40584 51513 77397 0	0.12948 49661 68870 0.27970 53914 89277 0.38183 00505 05119 0.41795 91833 73469
8	0.96023 98564 97536 0.79666 64774 13627 0.52555 24099 16329 0.18343 46424 95650	0.10122 85362 90376 0.22238 10344 53374 0.31370 66458 77887 0.36268 37833 78362
9	0.96816 02395 07626 0.83603 11073 26636 0.61337 14327 00590 0.32425 34234 03809 0	0.08127 43883 61574 0.18064 81606 94857 0.26061 06964 02935 0.31234 70770 40003 0.33023 92550 01260
10	0.97390 65785 17172 0.86506 33666 88985 0.67940 95682 99024 0.43339 53941 29247 0.14887 43389 81631	0.06667 13443 08868 0.14945 13491 50581 0.21908 63625 15982 0.26926 67193 09996 0.29552 42247 14753
11	0.97822 86581 46057 0.88706 25997 68095 0.73015 20055 74049 0.51909 61292 06812 0.26954 31559 52345 0	0.05566 85671 16174 0.12558 03694 64905 0.18629 02109 27734 0.23319 37645 91990 0.26280 45445 10247 0.27292 50867 77901

（续表）

m	$\pm x_i$			w_i		
12	0.98156	06342	46719	0.04717	53363	86512
	0.90411	72563	70475	0.10693	93259	95318
	0.76990	26741	94365	0.16007	83285	43346
	0.58731	79542	86617	0.20316	74267	23066
	0.36783	14989	98180	0.23349	25365	38355
	0.12523	34085	11469	0.24914	70458	13403
13	0.98418	30547	18588	0.04048	40047	65316
	0.91759	83992	22978	0.09212	14998	37728
	0.80157	80907	33310	0.13887	35102	19787
	0.64234	93394	40340	0.17814	59807	61946
	0.44849	27510	36447	0.20781	60475	36389
	0.23045	83159	55135	0.22628	31802	62897
	0			0.23255	15532	30874
14	0.98628	38086	96812	0.03511	94603	31752
	0.92843	48836	63574	0.08015	80875	59760
	0.82720	13150	69765	0.12151	85706	87903
	0.68729	29084	11685	0.15720	31671	58194
	0.51524	86363	58154	0.18553	83974	77938
	0.31911	23689	27890	0.20519	84637	21296
	0.10805	49487	07344	0.21526	38534	63158
15	0.98799	25180	20485	0.03075	32419	96117
	0.93727	33924	00706	0.07036	60474	88108
	0.84820	65834	10427	0.10715	92204	67172
	0.72441	77313	60170	0.13957	06779	26154
	0.57097	21726	08539	0.16626	92058	16994
	0.39415	13470	77563	0.18616	10000	15562
	0.20119	40939	97435	0.19843	14853	27112
	0			0.20257	82419	25561
16	0.98940	09349	91650	0.02715	24594	11754
	0.94457	50230	73233	0.06225	35289	38648
	0.86563	12023	87832	0.09519	85116	82498
	0.75540	44083	55003	0.12462	89712	55534
	0.61787	62444	02644	0.14959	59888	16577
	0.45801	67776	57227	0.16915	65193	95003
	0.28160	35507	79259	0.18260	34150	44924
	0.09501	25098	37637	0.18945	06104	55068

```
for i:=1 step 1 until m do
  begin x:=a+c*t[i]; a1:=g1(x);
    b1:=g2(x)-a1; e1:=0;
    for j:=1 step 1 until m do
      begin y:=a1+b1*t[j]; a2:=p1(x, y);
        b2:=p2(x, y)-a2; e2:=0;
        for k:=1 step 1 until m do
          e2:=f(x, y, a2+b2*t[k])*u[k]+e2;
        e1:=b2*e2*u[j]+e1
      end;
    e:=b1*e1*u[i]+e
  end;
GI3:=c*e
end;
```

五　严格选大元消去法解线性代数方程组(*)

(一)功能

对于 n 阶线性代数方程组

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i=1, 2, \cdots, n,$$

程序在整个系数矩阵找最大模元素作为主元素，采用约当消去法求出解 $x_1, x_2, \cdots, x_n$. 求解过程中，当逐次取得的主元小于给定的 eps 值时，程序出现条件停机(如果控制台上条件停机扳键拨上的话)，并可继续计算.

(二)使用说明

语句形式：$GJ(n, a, b,$ eps, $ik, jk)$.

其中，n 表示方程组阶数，值参数；$a[1{:}n, 1{:}n]$表示元素按行

存放的系数矩阵；$b[1:n]$表示方程组右端项，最后存放方程组的解 $x_1, x_2, \cdots, x_n$；eps 表示最大主元所不能小于的数值，值参数；$ik, jk[1:n]$表示所需的两套工作单元.

(三)程序

```
procedure GJ(n, a, b, eps, ik, jk);
    value n, eps; integer n; real eps;
      array a, b, ik, jk;
  begin integer i, j, k, l, i0, j0; real c, d;
    for k:=1 step 1 until n do
      begin c:=0;
        for i:=1 step 1 until n do
          begin
            for l:=1 step 1 until k-1 do
              if i=ik[l] then goto L2;
            for j:=1 step 1 until n do
              begin if #abs(a[i, j])≤#abs(c)
                then goto L1;
                  c:=a[i, j]; i0:=i; j0:=j;
              L1: end;
        L2: end;
        if #abs(c)≤eps then pause;
        ik[k]:=i0; jk[k]:=j0;
        for j:=1 step 1 until n do
          begin
            for l:=1 step 1 until k do
              if j=jk[l] then goto L4;
            if a[i0, j]=0 then goto L4;
```

```
        d:=a[i0, j]:=a[i0, j]/c;
        for i:=1 step 1 until n do
          begin if i=i0 then goto L3;
            a[i, j]:=a[i, j]-a[i, j0]×d;
          L3: end;
      L4: end;
      d:=b[i0]:=b[i0]/c;
        for i:=1 step 1 until n do
        begin if i=i0 then goto L5;
          b[i]:=b[i]-a[i, j0]×d;
      L5: end;
        for i:=1 step 1 until n do a[i, j0]:=0
    end;
  for k:=1 step 1 until n do
    a[1, jk[k]]:=b[ik[k]];
  for k:=1 step 1 until n do
    b[k]:=a[1, k]
end;
```

六　共轭斜量法解线性代数方程组(*)

(一)功能

对于 n 阶线性代数方程组

$$AX=B,$$

这里，$A=(a_{ij})$，$X=(x_i)$，$B=(b_i)$，$i, j=1, 2, \cdots, n$，程序按如下步骤求出其解或近似解：

1）给出方程组解的初值 X_0。

2）计算 $r_0=B-AX_0$，$P_0=A^Tr_0$，A^T 为 A 的转置。

3) 重复下列公式的计算：

$$\alpha_i=\frac{(AP_i,\ r_i)}{(AP_i,\ AP_i)},$$

$$X_{i+1}=X_i+\alpha_iP_i,$$

$$r_{i+1}=r_i-\alpha_iAP_i,$$

$$\beta_i=\frac{(A^Tr_{i+1},\ A^Tr_{i+1})}{(A^Tr_i,\ A^Tr_i)},$$

$$P_{i+1}=A^Tr_{i+1}+\beta_iP_i.$$

直至 $i=n-1$ 或剩余值变得足够小为止.

(二)使用说明

语句形式：$CG(x,\ r,\ n,\ \text{mat},\ p,\ \text{temp})$.

其中，n 是方程组阶数，值参数；$x[1:n]$ 开始时置迭代初值，结束时存放方程组解；$r[1:n]$ 开始时置方程右端 B，结束时存放余量；P、temp$[1:n]$ 为两套所需用的工作单元组；mat 表示矩阵与向量乘积的一个一般过程说明，由使用者自行编写，其形式如下：

```
procedure mat (tra, dat, res);
    boolean tra; array dat, res;
  begin
    B: = if tra then At else A;
    res: = B × dat
  end.
```

这里，A 为方程组的系数矩阵，At 表示 A 的转置阵.

(三)程序

```
procedure CG(x, r, n, mat, p, temp);
    value n;
    integer n; array x, r, p, temp;
```

```
  procedure mat;
begin
  real alpha, beta, atrsq;
  integer iter;
  real procedure dot (u, v);
      array u, v;
    begin
        integer i; real sum;
      sum:=0;
      for i:=1 step 1 until n do
          sum:=sum+u[i]×v[i];
      dot:=sum
    end;
  procedure combine (f, c, g, h);
    value c;
    real c; array f, g, h;
  begin
      integer i;
    for i:=1 step 1 until n do
      h[i]:=f[i]+c×g[i]
  end;
    L0: for iter:=0 step 1 until n do
    begin if iter≠0 then
      begin mat (false, x, temp);
        combine (r, -1, temp, r);
        mat (true, r, p); atrsq:=dot(p, p)
        end else begin mat (true, r, temp);
```

```
        beta: =dot (temp, temp)/atrsq;
        combine (temp, beta, p, p);
        atrsq: =beta×atrsq end;
          if atrsq=0 then goto L_n;
        mat (false, p, temp);
        alpha: =dot (temp, temp);
          if alpha=0 then goto L_n;
        alpha: =dot (r, temp)/alpha;
        combine (x, alpha, p, x);
        combine (r, -alpha, temp, r)
      end;
  L_n: end;
```

七　高斯-塞德尔迭代法解线性代数方程组(*)

(一)功能

对于 n 阶线性代数方程组

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i=1, 2, \cdots, n,$$

程序采用高斯-塞德尔迭代法求其解：$x_1, x_2, \cdots, x_n$. 计算步骤是：

$$0 \Rightarrow x_i \Rightarrow y_i, \quad i=1, 2, \cdots, n.$$

$$y_i = \left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}y_j\right) \Big/ a_{ii}, \quad i=1, 2, \cdots, n.$$

判别：$|y_i - x_i| < \mathrm{tol}\ (i=1, 2, \cdots, n)$ 满足时迭代结束.

(二)使用说明

语句形式：$GS\ (n, a, b, \mathrm{tol}, x, y)$.

其中，n 是方程组阶数，值参数；$a[1:n, 1:n]$ 表示元素按行存放的系数矩阵；$b[1:n]$ 表示方程组右端；tol 表示迭代要求的绝对误差，值参数；x，$y[1:n]$ 表示所需的两套工作单元，迭代结束后在 $y[1:n]$ 中得到方程组的解 x_1，x_2，…，x_n。

如果系数矩阵的对角元优于其他元素，则迭代收敛，否则迭代过程可能发散。

(三)程序

```
procedure GS (n, a, b, tol, x, y);
    value n, tol; real tol; integer n;
      array a, b, x, y;
  begin integer i, j, k;
    for i:=1 step 1 until n do x[i]:=y[i]:=0;
L1: for i:=1 step 1 until n do
    begin y[i]:=b[i];
      for j:=1 step 1 until n do
        y[i]:=if i=j then y[i] else
          y[i]-a[i, j]×y[j];
      y[i]:=y[i]/a[i, i]
    end;
  for k:=1 step 1 until n do
    if ##abs (y[k]-x[k])≥tol then
      begin for i:=1 step 1 until n do
          x[i]:=y[i];
        goto L1
      end
  end;
```

八　部分主元消去法解高阶线性代数方程组

(一)功能

对于 n 阶线性代数方程组：

$$\sum_{j=1}^{n} a_{ij}x_{jr}=b_{ir} \quad (i=1,\ 2,\ \cdots,\ n;\ r=1,\ 2,\ \cdots,\ m),$$

程序采用压缩存贮的部分主元素消去法求解.

算法的第 i 步是，首先将 $a_{ij}(j<i)$ 化为 0，在余下的 $a_{ij}(j\geqslant i)$ 中选取按模最大的元素作主元，且把它所在的列同第 i 列交换，同时记录列标. 用 a_{ii} 遍除 $a_{ij}(j>i)$；其次，把 $a_{ki}(k<i)$ 化为 0. 这样，就把 A 的 i 阶主子阵化为单位阵 I（只要隐含表示而不需存贮），余下需要存贮和处理的只是 A 的前 i 行的右边的 $n-i$ 列的元素. 因为，$\underset{1\leqslant i\leqslant n}{\mathrm{Max}}\{i(n-i)\}=n^2\div 4$，所以，只需 $n^2\div 4$ 个内存单元存贮 A 的元素，可以求解较高阶数的线性方程组.

(二)使用方法

语句形式：$PPS(n,\ m,\ A,\ B,\ C,\ D,\ ep,\ CA)$.

其中，n 表示方程组阶数，值参数；m 表示方程组右端项列数，值参数；$A[1{:}n]$ 表示存放系数矩阵第 i 行元素的工作单元；$B[1{:}n,\ 1{:}m]$ 表示方程组的右端项，过程被调用后存放方程组的解；$C[1{:}n]$、$D[1{:}n\div 4]$ 表示两套工作单元；ep 表示主元绝对值所不能小于的值；$CA(i,n,A)$ 表示形成或从外存调入系数矩阵第 i 行元素于数组 $A[1{:}n]$ 中的过程，由使用者自编.

(三)程序

```
procedure PPS (n, m, A, B, C, D, ep, CA);
    value n, m, ep;
    integer n, m; real ep; array A, B, C, D;
```

```
    procedure CA;
begin integer i, j, k, l, t, p, q; real s, r;
  for i:=1 step 1 until n do
    begin CA (i, n,A); t:=n+1-i; q:=i-1;
      if i=1 then
        for j:=1 step 1 until n do c[j]:=j;
      for j:=1 step 1 until q do
        begin k:=c[j];
          L1: if k<j then begin k:=c [k];
            goto L1 end;
            if k>j then begin S:=A[j];
              A[j]:=A[k]; A[k]:=S end
          end;
        r:=0;
      for j:=i step 1 until n do
        begin l:=j-i+1; s:=A[j];
          for k:=1 step 1 until q do
            begin s:=s-a[k]*d[l]; l:=l+m
              end;
          a[j]:=s;
          if #abs (s)>r then
            begin r:= #abs(s); p:=j end
        end;
      if r<ep then stop;
      if p≠i then begin s:=a[i]; a[i]:=a [p];
        a[p]:=s; j:=c[i]; c[i]:=c[p];
          c[p]:=j; k:=1; p:=p-i;
```

```
for j:=1 step 1 until q do
  begin s:=d[k]; d[k]:=d[k+p];
    d[k+p]:=s; k:=k+t end end;
r:=1/a[i];
for j:=i+1 step 1 until n do
  a[j]:=r*a[j];
for j:=1 step 1 until m do
  begin s:=b[i, j];
    for k:=1 step 1 until q do
      s:=s-a[k]*b[k, j]; b[i, j]:=s*r
  end;
l:=1;
for j:=1 step 1 until q do
  begin s:=d[l];
    for k:=2 step 1 until t do
      d[l-j+k-1]:=d[l+k-1] -s*a[i
        +k-1];
    for k:=1 step 1 until m do
      b[j, k]:=b[j, k]-s*b[i, k];
    l:=l+t
  end;
p:=n-i; t:=p*q;
for k:=1 step 1 until p do
  d[t+k]:=a[i+k];
if i=n then
  for j:=1 step 1 until m do
    begin
```

```
                for k:=1 step 1 until n do
                  a[c[k]]:=b[k, j];
                for k:=1 step 1 until n do
                  b[k, j]:=a[k]
              end
          end
      end;
```

九　高阶对称正定稀疏线性代数方程组的分块解法

(一)功能

对高阶线性代数方程组

$$AX=B,$$

其中，A 是 n 阶对称正定稀疏矩阵，X 和 B 是 $n\times m$ 阶矩阵，程序采用分块的方法，把"用改进平方根法解对称正定稀疏线性方程组(一维变带宽存贮)"的算法引伸到外存中去，从而有效地解决具有多个右端项的高阶对称正定稀疏线性方程组的求解问题.

分块解法的基本思想是，预先将系数矩阵 A 划分为 p_n 个行块(每个行块的大小按内存容量而定，划块时第 p 个行块限于同第 $p-1$、$p+1$ 个行块各构成一个公共相交三角块. 图 24 是 $p_n=4$ 的分块示意图). 然后，程序按如下步骤求解：

1．对 $p=1\sim p_n$ 个矩阵块元素 A_p 逐块进行分解，并求出相应部分的中间变量 Y_p. 这个过程中，程序根据预给信息 $nw[p]$，把第 p 个相关公共三角块 T_p 及相应对角元位置记鼓(p_n 个矩阵行块共有 p_n-1 个相关公共三角块，第 p_n-1 个相关公共三角块可直接参加回代计算，无须记鼓)，记鼓信息约定：

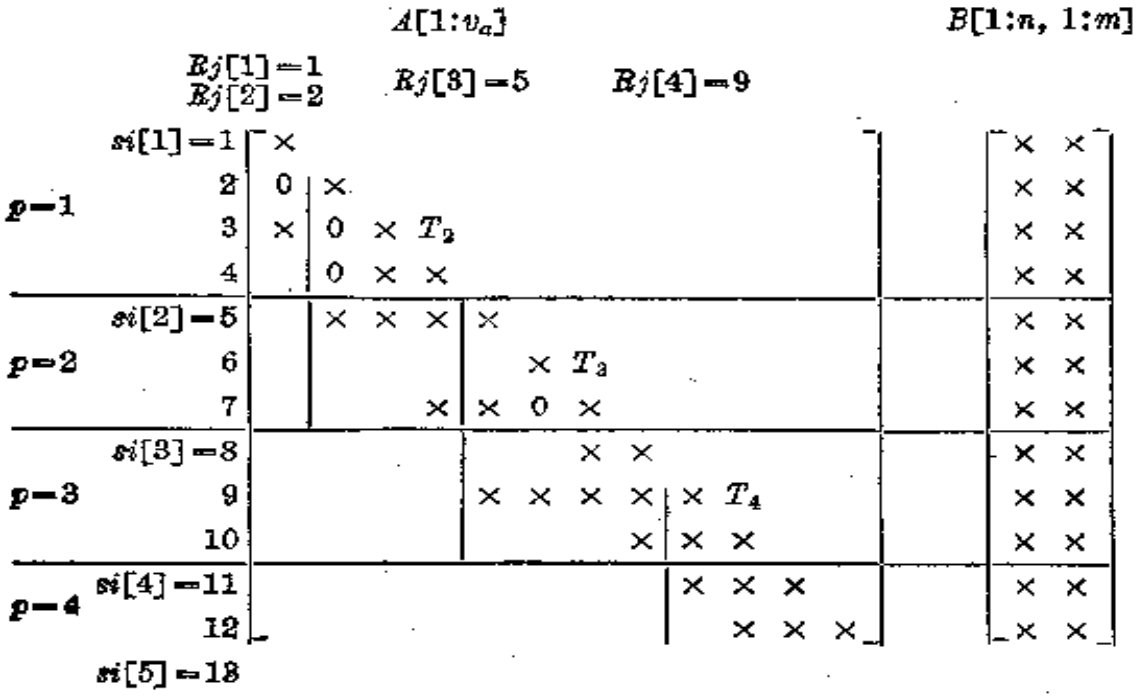


图 24　分块解法示意图

$$mw[p]=\begin{cases}<0, \text{表示第 } p \text{ 个相关公共三角块 } T_{p+1} \text{ 及相应对角元位置不记数.}\\ \geqslant 0, \text{表示第 } p \text{ 个相关公共三角块 } T_{p+1} \text{ 及相应对角元位置记于起始地址为 } mw[p] \text{ 的数区.}\end{cases}$$

2．对 $p=p_n\sim 1$ 个矩阵块元素 A_p（经步骤 1 分解过）及相应中间变量 Y_p，逐块进行回代计算求出相应部分的解 X_p. 这个过程中，程序根据记数信息 $mw[p]$ 由外存调入第 p_0 ($p_0\leqslant p-1$)个相关公共三角块，在此基础上重新形成分解后的第 p 个矩阵块元素 A_p 及相应对角元位置.

3．上面求得的 X_p，合在一起就得原方程的解 X.

(二)使用说明

语句形式：$LDLT_0(n,\ m,\ pn,\ vd3,\ va3,\ d3,\ A3,\ d,\ A,\ B,\ Si,\ Rj,\ mw,\ CAd)$.

其中，n、m、p_n 分别表示方程组阶数、右端项列数及分块总数，都是值参数；$d3[0:vd3]$、$A3[1:va3]$ 分别表示存放第 $p(p=1\sim p_n-2)$ 个相关公共三角块 T_p 的对角元位置和元素（按"一维变带存贮"形式）的工作单元（这里，$vd3$、$va3$ 分别是其体积的上限、是值参数）；$d[0:vd]$、$A[1:va]$ 分别表示存放第 $p(p=1\sim p_n)$ 个矩阵块（相关公共三角块加矩阵行块）对角元位置和元素（按"一维变带存贮"形式）的工作单元（这里，vd、va 分别表示其体积的上限，使用者可在主程序直接给定）；$B[1:n, 1:m]$ 开始存放右端项，过程调用后，在此得到方程组的解；$S_i[1:p_n+1]$ 表示第 $p=1\sim p_n+1$ 个矩阵行块的起始行号（在整体矩阵中，且约定 $S_i[p_n+1]=n+1$）；$R_j[1:p_n]$ 表示第 $p=1\sim p_n$ 个矩阵行块中最左非零元素所在的列号（在整体矩阵中）；$nw[1:p_n-1]$ 表示第 $p=1\sim p_n-1$ 个相关公共三角块的预给记数信息（约定，$nw[p]<0$ 表示第 p 个相关公共三角块对角元位置 $d[0:vd3]$ 及元素 $A3[1:va3]$ 不予记数；$nw[p]\geqslant 0$ 表示记于始地址为 $nw[p]$ 的数区），使用者应根据数区容量适当划块，然后恰当给出记数信息；$CAd(p, Si, Rj, A, d)$ 表示按"一维变带存贮"形式形成第 p 个矩阵行块元素于 $A[d[S_i[p]-R_j[p]]]+1\sim A[d[S_i[p+1]-R_j[p]]]$ 中，相应对角元位置于 $d[S_i[p]-R_j[p]+1]\sim d[S_i[p+1]-R_j[p]]$ 中的过程，由使用者编写。

（三）程序

```
procedure LDLT0 (n, m, pn, vd3, va3, d3, A3, d,
  A, B, Si, Rj, nw, Cad);
    value n, m, pn, vd3, va3; integer n, m, pn,
      vd3, va3;
      array d3, A3, d, A, B, Si, Rj, nw; procedure
```

```
        CAd;
    begin integer p, gn, p0, i, pp;
      procedure TA3(p); value p; integer p;
        begin integer n3, h3, i, ni, j;
          n3:=Si[p+1]-Rj[p+1];
          h3:=Rj[p+1]-Rj[p];
          for i:=1 step 1 until n3 do
            begin ni:=d[h3+i]-d[h3+i-1];
              if i<ni then ni:=i; d3[i]:=d[i]:=
                d[i-1]+ni;
              for j:=ni-1 step -1 until 0 do
                A3[d[i]-j]:=A[d[i]-j]:=A[d[h3
                  +i]-j]
            end
        end;
    procedure LDLT1 (p, ind); value p, ind; integer
      p, ind;
      begin integer nB, ne, dB, i, di, mi, idB, S, j,
        dj, mj, ij, r, iS, SdB; real C;
        nB:=Si[p]-Rj[p]+1; ne:=Si[p+1]-Rj[p];
          dB:=Rj[p]-1;
        if ind≠-1 then begin
          for i:=nB step 1 until ne do
            begin di:=d[i]-i; mi:=d [i-1] -di+1;
              idB:=i+dB;
              for S:=mi step 1 until i-1 do
                begin C:=A[di+S]/A[d[S]];
```

```
                for j:=S+1 step 1 until i do
                  begin dj:=d[j] -j; mj:=d[j-1]
                    -dj+1; ij:=di+j;
                    if S≥mj then A[ij]:=A[ij] -
                      C*A[dj+S]
                  end;
                if ind=1 then
                  for r:=1 step 1 until m do
                    B[idB, r]:=B[idB, r] -C*B [S
                      +dB, r]
              end
            end end else begin
          for i:=ne step -1 until nB do
            begin di:=d [i] -i; mi:=d [i-1]-di+1;
              idB:=i+dB;
              for r:=1 step 1 until m do
                B[idB, r]:=B[idB, r]/A[d[i]];
              for S:=mi step 1 until i-1 do
                begin iS:=di+S; SdB:=S+dB;
                  for r:=1 step 1 until m do
                    B[SdB, r]:=B[SdB, r]-
                      A[iS]*B[idB, r]
                end
            end end
        end;
      d[0]:=d3[0]:=0;
      for p:=1 step 1 until pn do
```

```
    begin #print (0, '2Hp=, i3', p);
      CAd(p, Si, Rj, A, d); LDLT1(p, 1);
      if p=pn then goto L1; TA3(p);
      if nw[p]≥0 then begin
        gn:=nw[p]; #write drum (0, gn, d3, A3) end
    end;
L1: for p:=pn step -1 until 1 do
    begin #print (0, '2Hp=, i3', p); if p=pn then
      goto L3;
      for p0:=p-1 step -1 until 1 do
        if nw[p0]≥0 then goto L2; p0:=0;
    L2: if p0≠0 then begin
        gn:=nw[p0]; #read drum (0, gn, d3, A3);
        for i:=1 step 1 until va3 do A[i]:=A3[i];
        for i:=0 step 1 until vd3 do d[i]:=d3[i]
                    end;
      for pp:=p0+1 step 1 until p do
        begin CAd(pp, Si, Rj, A, d); LDLT1 (pp, 0);
          if pp≠p then TA3(pp)
        end;
    L3: LDLT1 (p, -1)
      end
    end;
```

十　用消去法求矩阵的逆阵和行列式值(*)

(一)功能

对于非异矩阵 A，程序采用消去法求出其逆阵 A^{-1}，同时

得出 A 的行列式值. 消元过程中逐次取得的主元小于给定的 eps 值时，出现程序停机.

(二)使用说明

语句形式: inversion (n, a, b, c, z, eps, de).

其中，n 表示矩阵的阶数，值参数；$a[1:n, 1:n]$表示元素按行存放的系数矩阵，最后存放 A 的逆阵 A^{-1}；b、c、$z[1:n]$表示所需的三套工作单元；eps 表示最大主元所不能小于的数值，值参数；de 存放行列式值，为了防止溢出，调用前应在相应实在参数中置以适当的比例因子 f，从而，最后在 de 中得到的行列式值是

$$f \cdot |A|.$$

(三)程序

```
procedure inversion (n, a, b, c, z, eps, de);
    value n; real eps, de; integer n;
      array a, b, c, z;
  begin real w, y; integer i, j, k, l, p;
    for j:=1 step 1 until n do z[j]:=j;
    for i:=1 step 1 until n do
      begin k:=i; y:=a[i, i]; l:=i-1; p:=i+1;
        for j:=p step 1 until n do
          begin w:=a[i, j];
            if #abs (w)>#abs (y) then
              begin k:=j; y:=w end
          end;
        de:=de×y;
        if #abs (y)<eps then stop; y:=1/y;
        for j:=1 step 1 until n do
```

```
                begin c[j]:=a[j, k]; a[j, k]:=a[j, i];
                  a[j, i]:=-c[j]×y;
                  b[j]:=a[i, j]:=a[i, j]×y
                end;
              a[i, i]:=y; j:=z[i]; z[i]:=z[k];
                z[k]:=j;
              for k:=1 step 1 until l, p step 1 until n do
              for j:=1 step 1 until l, p step 1 until n do
                  a[k, j]:=a[k, j]-b[j]×c[k]
            end;
          for i:=1 step 1 until n do
            begin
              L1:k:=z[i]; if k=i then goto L2;
              for j:=1 step 1 until n do
                begin w:=a[i, j]; a[i, j]:=a[k, j];
                  a[k, j]:=w end;
                p:=z[i]; z[i]:=z[k]; z[k]:=p;
                  de:=-de; goto L1;
              L2: end
        end;
```

十一　用雅可比方法求对称矩阵特征值和特征向量(*)

(一)功能

对于 n 阶实对称矩阵 A，程序采用雅可比法求出 n 个特征值及其对应的 n 个特征向量.

(二)使用说明

语句形式：Jacobi (a, s, n, rho).

其中，n 表示矩阵的阶数，值参数；$a[1:n, 1:n]$表示元素按行存放的矩阵，最后在对角线位置上得到特征值；$s[1:n, 1:n]$的每一列，存放相应于 a 中特征值的特征向量；rho 表示计算结果的精确度，值参数；

(三)程序

```
procedure Jacobi (a, s, n, rho);
    value n, rho; integer n; real rho; array a, s;
  begin real n1, n2, thr, mu, ga, st, ct, it, v1, v2, v3;
    integer i, j, p, q, ind;
    for i:=1 step 1 until n do
      for j:=1 step 1 until i do
      if i=j then s[i, j]:=1 else
        s[i, j]:=s[j, i]:=0;
      it:=0;
      for i:=2 step 1 until n do
        for j:=1 step 1 until i-1 do
          it:=it+2×a[i, j]↑2;
        n1:=#sqrt (it); n2:=(rho/n)×n1;
        thr:=n1; ind:=0;
      L0:thr:=thr/n;
      L1: for q:=2 step 1 until n do
        for p:=1 step 1 until q-1 do
          if #abs (a[p, q])≥thr then
            begin ind:=1; v1:=a[p, p];
              v2:=a[p, q]; v3:=a[q, q];
              mu:=0.5×(v1-v3);
              ga:=if mu=0 then -1 else -#sign
```

```
                (mu) × v2/#sqrt (v2↑2+mu↑2);
            st:=ga/#sqrt (2×(1+#sqrt
                (1−ga↑2)));
            ct:=#sqrt (1−st↑2);
            for i:=1 step 1 until n do
              begin it:=a[i, p]×ct−a[i, q]×st;
                a[i, q]:=a[i, p]×st+a[i, q]×ct;
                a[i, p]:=it;
                it:=s[i, p]×ct−s[i, q]×st;
                s[i, q]:=s[i, p]×st+s[i, q]×ct;
                s[i, p]:=it
              end;
            for i:=1 step 1 until n do
              begin a[p, i]:=a[i, p];
                a[q, i]:=a[i, q] end;
            a[p, p]:=v1×ct↑2+v3×st↑2−2×v2
              ×st×ct;
            a[q, q]:=v1×st↑2+v3×ct↑2+2×v2
              ×st×ct;
            a[p, q]:=a[q, p]:=(v1−v3)
              ×st×ct+v2×(ct↑2−st↑2)
          end;
        if ind:=1 then begin ind:=0; goto L1
      end else
        if thr>n2 then goto L0
end;
```

十二　求实对称矩阵全部特征值和特征向量的QL算法

(一)功能

对于实对称矩阵 A 的特征值问题

$$AX=\lambda X,$$

程序按如下二步求出全部特征值和特征向量：

1．用豪塞豪德(Householder)变换把 A 化为三对角矩阵 T.

2．用QL算法求 T 的全部特征值及对应的 A 的特征向量.

(二)使用方法

语句形式：QL(n, A, d, e, ep).

其中，n 表示矩阵 A 的阶数，值参数；$A[1:n, 1:n]$开始按行存放矩阵元素，过程调用后按列存放所求特征向量(第 i 列元素对应于第 i 个特征向量)；$d[1:n]$存放所求特征值的数组；$e[1:n]$是工作单元组；ep 表示一个小量，一般可取 $_{10}-10$.

(三)程序

```
procedure QL(n, A, d, e, ep); value n, ep;
    integer n; real ep; array A, d, e;
  begin integer i, j, k, l, m;
    real f, g, h, hh, c, p, r, s, b;
  for i:=n step -1 until 2 do
    begin l:=i-2; f:=A[i, i-1]; g:=0;
      for k:=1 step 1 until l do
        g:=g+A[i, k]*A[i, k];
      h:=g+f*f;
      if g≤ep then begin e[i]:=f; h:=0; goto
```

```
      skip end;
    l:=l+1; g:=e[i]:=if f≥0 then
      -#sqrt(h) else #sqrt (h);
    h:=h-f*g; A[i, i-1]:=f-g; f:=0;
    for j:=1 step 1 until l do
      begin A[j, i]:=A[i, j]/h; g:=0;
        for k:=1 step 1 until j do
          g:=g+A[j, k]*A[i, k];
        for k:=j+1 step 1 until l do
          g:=g+A[k, j]*A[i, k];
        e[j]:=g/h; f:=f+g*A[j, i]
      end;
    hh:=f/(h+h);
    for j:=1 step 1 until l do
      begin f:=A[i, j]; g:=e[j]:=e[j]-hh*f;
        for k:=1 step 1 until j do
          A[j, k]:=A[j, k]-f*e[k]-g*A[i, k]
      end;
  skip:d[i]:=h
  end;
  d[1]:=e[1]:=0;
  for i:=1 step 1 until n do
    begin l:=i-1; if d[i]≠0 then
      for j:=1 step 1 until l do
        begin g:=0;
          for k:=1 step 1 until l do
            g:=g+A[i, k]*A[k, j];
```

```
            for k:=1 step 1 until l do
              A[k, j]:=A[k, j]-g*A[k, i]
          end;
        d[i]:=A[i, i]; A[i, i]:=1;
        for j:=1 step 1 until l do
          A[i, j]:=A[j, i]:=0
      end;
    for i:=2 step 1 until n do e[i-1]:=e[i];
    e[n]:=b:=f:=0;
    for l:=1 step 1 until n do
      begin j:=0; h:=ep*(#abs(d[l])+
        #abs(e[l]));
    if b<h then b:=h;
    for m:=l step 1 until n do
      if #abs (e[m])≤b then goto cont;
cont: if m=l then goto root;
nextit: if j=30 then stop; j:=j+1;
    g:=d[l]; p:=(d[l+1]-g)/(2*e[l]); r:=#sqrt
      (1+p*p); d[l]:=e[l]/(if p<0 then p-r else
      p+r); h:=g-d[l];
    for i:=l+1 step 1 until n do d[i]:=d[i]-h;
      f:=f+h;
    p:=d[m]; c:=1; s:=0;
    for i:=m-1 step -1 until l do
      begin g:=c*e[i]; h:=c*p;
        if #abs(p)≥#abs (e[i]) then
        begin c:=e[i]/p; r:=#sqrt (1+c*c);
```

```
            e[i+1]:=s*p*r;
          s:=c/r; c:=1/r end else begin
            c:=p/e[i]; r:=#sqrt(1+c*c);
            e[i+1]:=s*e[i]*r; s:=1/r; c:=c/r end;
        p:=c*d[i]-s*g; d[i+1]:=h+s*(c*g+s*d[i]);
        for k:=1 step 1 until n do
          begin h:=A[k, i+1]; A[k, i+1]:=
            s*A[k, i]+c*h; A[k, i]:=c*A[k, i]-s*h
          end
        end;
        e[l]:=s*p; d[l]:=c*p; if #abs(e[l])>b
          then goto nextit;
      root: d[l]:=d[l]+f
      end
    end;
```

十三　求实对称矩阵部分(或全部)特征值、特征向量的吉文斯-豪塞豪德方法

(一)功能

对于 n 阶实对称矩阵 A 的特征值问题

$$AX=\lambda X,$$

程序按如下步骤求其特征值、特征向量.

1．用豪塞豪德 (Householder) 正交相似变换 $C=PAP^T$ 将 A 化为三对角矩阵 C.

2．根据 $C-\lambda I$ 的主子式构成多项式的施斗姆 (Sturm) 序列，利用二分法求出 C 的按代数值从小到大的 ne 个特征值.

3．用反迭代法求对应于 $nv(nv \leqslant ne)$ 个特征值的特征向量.

(二)使用方法

语句形式：*GH* (*n*, *ne*, *nv*, *eps*, *A*, *E*, *v*, *B*, *C*, *P*, *Q*, *R*, *W*, *Y*, *in*).

其中，*n* 表示矩阵 *A* 的阶数，值参数；*ne*、*nv*$(1 \leqslant nv \leqslant ne \leqslant n)$分别表示要求的特征值和特征向量的个数，值参数；*eps* 表示精度要求，值参数；*A*[1:*n*, 1:*n*] 表示按行存放 *A* 元素的数组；*E* [1:*ne*] 表示存放所求特征值的数组（按代数值从小到大次序排列）；*v*[1:*n*, 1:*nv*]表示存放所求特征向量的数组（*nv* 列分别对应于前 *nv* 个特征值）；*B*[1:*n*]、*C*[1:*n*]分别表示存放三对角矩阵次、主对角线元素的数组；*P* [1:*n*]、*Q* [1:*n*]、*R*[1:*n*]、*W*[1:*n*]、*Y*[1:*n*+2]、*in*[1:*n*]表示六套工作单元.

(三)程序

```
procedure GH (n, ne, nv, eps, A, E, V, B, C, P,
  Q, R, W, Y, in);
    value n, ne, nv, eps; integer n, ne, nv;
      real eps;
    array A, E, V, B, C, P, Q, R, W, Y, in;
  begin integer i, j, k, m, i1, n1, n2, aa;
    real s, ss, t, l, kk, lam, norm, al, u;
    boolean first;
  LH1: n1:=n-1; n2:=n-2; t:=#Ranump(0);
    if n≤2 then goto LH3;
    for i:=1 step 1 until n2 do
      begin ss:=0; i1:=i+1;
        for j:=i1 step 1 until n do
```

```
      ss: = ss + A[j, i] * A[j, i];
    s: = #sqrt (ss); if A[i1, i] < 0 then s: = -s;
      C[i]: = A[i, i]; B[i]: = -s; al: = 0;
    if s = 0 then goto LH2;
    al: = 1/(ss + s * A[i1, i]); W[i1]: = A[i1, i]:
      = A[i1, i] + s;
    for j: = i + 2 step 1 until n do
      W[j]: = A[j, i];
    for j: = i1 step 1 until n do
      begin t: = 0;
        for k: = i1 step 1 until n do
          t: = t + A[j, k] * W[k];
        P[j]: = t * Al
      end;
    kk: = 0;
    for j: = i1 step 1 until n do
      kk: = kk + W[j] * P[j]; kk: = kk * al/2;
    for j: = i1 step 1 until n do
      q[j]: = P[j] - kk * W[j];
    for j: = i1 step 1 until n do
      for k: = j step 1 until n do
        A[k, j]: = A[j, k]: =
          A[j, k] - (q[j] * W[k] + q[k] * W[j]);
  LH2: A[i, i]: = al
  end;
LH3: b[n1]: = A[n, n1]; c[n1]: = A[n1, n1];
  c[n]: = A[n, n]; b[n]: = 0;
```

```
LB1: for i:=1 step 1 until n do
  W[i]:=b[i]*b[i];
  norm:=#abs(c[1])+#abs(b[1]);
  for i:=2 step 1 until n do
    begin t:=#abs(c[i])+#abs(b[i])+
      #abs(b[i-1]);
      if norm<t then norm:=t
    end;
  l:=-norm;
for i:=1 step 1 until ne do E[i]:=norm;
for k:=1 step 1 until ne do
  begin u:=E[k];
    LB2: lam:=(l+u)/2; s:=l*u;
      if s≤0 then s:=(u-l)*10-4 else begin
        t:=(#abs(u)+#abs(l))/2;
          s:=#abs(u/t-l/t) end;
      if s≤eps then e[k]:=lam else begin
        aa:=0; i:=1;
    LB3: s:=c[i]-lam;
    LB4: if s≥0 then aa:=aa+1;
        if s=0 then begin i:=i+2; goto LB7
          end; i:=i+1;
        if i>n then goto LB8;
        if W[i-1]<#abs(s) then goto LB5;
          ss:=s/W[i-1];
        if #abs(ss)<10-15 then goto LB6;
    LB5: s:=c[i]-lam-W[i-1]/s; goto LB4;
```

```
        LB6: if s<0 then aa:=aa+1; i:=i+1;
        LB7: if i≤n then goto LB3;
        LB8: if n-aa<k then l:=lam else
                 begin u:=lam;
                 m:=if n-aa<ne then n-aa else ne;
                 for j:=k+1 step 1 until m do
                   e[j]:=lam end;
               goto LB2 end
      end;
    LW1: if nv≤0 then goto LW5;
      y[n+1]:=y[n+2]:=0;
      for i:=1 step 1 until nv do
        begin
          for j:=1 step 1 until n do
            begin R[j]:=C[j]-e[i]; q[j]:=B[j];
              P[j]:=0; y[j]:=1
            end;
          first:=true;
          for j:=1 step 1 until n1 do
            begin if R[j]=0∧B[j]=0 then begin
              R[j]:=10-15; W[j]:=0; in[j]:=0;
                goto LW2 end;
              if #abs (R[j])<#abs (B[j]) then
                begin S:=R[j]/B[j]; in[j]:=1;
                R[j]:=b[j]; t:=R[j+1];
                R[j+1]:=q[j]; q[j]:=t;
                t:=q[j+1];q[j+1]:=p[j];p[j]:=t
```

```
      end else begin S:=B[j]/R[j];
      in[j]:=0 end;
      R[j+1]:=R[j+1]-S*q[j];
        q[j+1]:=q[j+1]-S*p[j];
        W[j]:=S;
LW2: end;
  if R[n]=0 then R[n]:=10-15;
  if i≠1∧#abs (e[i]-e[i-1])≤
    norm*2↑(-24) then
    for k:=1 step 1 until n do
      y[k]:=2*#Ranum (0)-1;
LW3: for k:=n step -1 until 1 do
  begin LW4: S:=y[k]-q[k]*y[k+1]
    -p[k]*y[k+2];
    if #abs (S)<#abs (R[k]) then
      y[k]:=S/R[k] else
      begin SS:=R[k]/S; if #abs (SS)
        >10-4 then
        y[k]:=-S/R[k] else begin
          for j:=1 step 1 until n do
            y[j]:=y[j]*10-3;
          goto LW4 end end
  end;
if first then begin first:=false;
  for j:=1 step 1 until n1 do
    if in[j]=1 then begin t:=y[j];
      y[j]:=y[j+1]; y[j+1]:=
```

```
                    t-W[j]*y[j+1] end else
                    y[j+1]:=y[j+1]-W[j]*y[j];
                    goto LW3 end;
                for j:=n2 step -1 until 1 do
                begin t:=0;
                  for k:=j+1 step 1 until n do
                    t:=t+A[k, j]*y[k];
                  t:=t*A[j, j];
                  for k:=j+1 step 1 until n do
                    y[k]:=y[k]-t*A[k, j]
                end;
              t:=0;
              for j:=1 step 1 until n do
                if t<#abs (y[j]) then t:=#abs (y[j]);
              if t>0 then
                for j:=1 step 1 until n do
                  v[j, i]:=y[j]/t
            end;
        LW5: end;
```

十四　用幂法和穷举法求实矩阵的部分特征值和特征向量

(一)功能

对于实矩阵 A 的特征值问题

$$AX=\lambda X,$$

程序按下列方法求出 $ne(\leqslant n)$个实的特征值(从大到小)和相应的特征向量:

1．用幂法求 $A_1=A$ 的最大特征值和特征向量，即利用迭代公式：

$$A_1X^{(k)}=\lambda^{(k+1)}X^{(k+1)}\quad(k=0,\ 1,\ 2,\ \cdots),$$

当 k 充分大时，使 $X^{(k)}=X^{(k+1)}$，则可取 $\lambda_1=\lambda^{(k+1)}$，$X_1=X^{(k+1)}$ 作为 A 的最大特征值和特征向量．

2．用穷举法求 A 的第 2 特征值和特征向量．即构造矩阵：

$$A_2=A_1-\lambda_1X_1Y_1^T.$$

其中，λ_1、X_1 为 A_1 的最大特征值和特征向量，Y_1 是 A_1^T 的对应于最大特征值 μ_1 的特征向量，且有 $(X_1Y_1)=1$．

然后，用幂法求 A_2 的最大特征值和特征向量．则它们就是 A_1 的第 2 特征值 λ_2 和特征向量 X_2．

3．类似地，依次可求 A 的前 $ne(\leqslant n)$ 个特征值和特征向量．

（二）使用方法

语句形式：*power*(*n*, *ne*, *A*, *ep*, *u*, *X*, *Xk*, *Xk1*)．

其中，*n* 表示矩阵 *A* 的阶数，值参数；*ne* 表示需求特征值的个数，值参数；*A*[1:*n*, 1:*n*] 按行存放矩阵元素的数组；*ep* 表示迭代的精度要求，值参数；*u*[1:*ne*]、*X*[1:*n*, 1:*ne*]分别存放所求特征值和特征向量（按列存放）的数组，*Xk*[1:*n*]、*Xk1*[1:*n*] 表示两套工作单元；调用时，*Xk*[1:*n*]中预置迭代初始向量．

（三）程序

```
procedure power(n, ne, A, ep, u, X, Xk, Xk1);
  value n, ne, ep;
    integer n, ne; real ep; array A, u, X, Xk,
      Xk1;
  begin integer i1, i2, j1, i, j; real ep1, b, d, s;
```

```
for j:=1 step 1 until ne do
  begin u[j]:=0;
    for i:=1 step 1 until n do
      X[i, j]:=0
  end;
for i1:=1 step 1 until ne do
  begin i2:=1;
    L1:j1:=0;
    L2:ep1:=b:=0;
  for i:=1 step 1 until n do
    begin xk1[i]:=0;
      for j:=1 step 1 until n do
        xk1[i]:=xk1[i]+A[i, j]*xk[j];
      if #abs (xk1[i])>#abs (b) then
        b:=xk1[i]
    end;
  for i:=1 step 1 until n do
    begin xk1[i]:=xk1[i]/b;
      d:=#abs (xk1[i]-xk[i]);
      if ep1<d then ep1:=d
    end;
  if j1=0∨ep1>ep then begin j1:=j1+1;
    for i:=1 step 1 until n do
      xk[i]:=xk1[i]; goto L2 end;
  for i:=1 step 1 until n do
    for j:=i step 1 until n do
      begin S:=A[i, j]; A[i, j]:=A[j, i];
```

```
            A[j, i]:=S end;
        i2:=-1*i2; if i2<0 then begin u[i1]:=b;
          for i:=1 step 1 until n do
            X[i, i1]:=xk1[i]; goto L1 end;
          S:=0;
          for i:=1 step 1 until n do
            S:=S+xk1[i]*x[i, i1];
          for i:=1 step 1 until n do
            for j:=1 step 1 until n do
              A[i, j]:=A[i, j]-u[i1]*X[i,
                i1]*xk1[j]/S
      end
    end;
```

十五　求实矩阵全部特征值、特征向量的QR算法

(一)功能

本程序按如下步骤求实矩阵A的全部特征值、特征向量：

1．用对角阵相似变换减小A的范数.

2．用初等相似变换把A化为上准三角形矩阵.

3．用带原点移动的QR算法求上准三角形矩阵的全部特征值．然后再求A的全部特征值、特征向量.

(二)使用方法

语句形式：QR(n, Ba, eps, A, WR, Wi, V, D, cn).

其中，n表示矩阵A的阶数，值参数；Ba表示2、10、16中任取一值的值参数；eps表示满足$1+eps>1$的机器最小整数，值参数；$A[1:n, 1:n]$表示按行存放矩阵A元素的数组；$WR[1:n]$、$Wi[1:n]$分别表示存放所求特征值实部和虚部的数

组；$V[1:n, 1:n]$表示存放所求特征向量的数组，如果第 i 个特征值是实的，则 v 的第 i 列元素是其相应的特征向量；如果第 i、$i+1$ 个是一对复共轭特征值，则 v 的第 i、$i+1$ 列元素分别是这对特征值中虚部为正的那个特征值所对应的特征向量的实部和虚部。$D[1:n]$ 表示一组工作单元；$cn[1:n]$ 表示存放每一特征值所化的迭代次数。

（三）程序

```
procedure QR (n, Ba, eps, A, Wr, Wi, v, d, cn);
    value n, Ba, eps; integer n, Ba; real eps;
      array A, Wr, Wi, v, d, cn;
  begin integer i, j, k, l, no, m, na, its, en;
    real B2, c, f, g, r, s, x, y, p, q, t, w, z, ra, sa,
      vr, vi, norm, low, upp;
    procedure cdiv (xr, xi, yr, yi, zr, zi);
      value xr, xi, yr, yi;
        real xr, xi, yr, yi, zr, zi;
      begin real h;
        if #abs (yr)>#abs (yi) then begin h:=
          yi/yr; yr:=h*yi+yr; zr:=(xr+h*xi)/yr;
          zi:=(xi-h*xr)/yr end else begin h:=
          yr/yi; yi:=h*yr+yi; zr:=(h*xr+xi)/yi;
          zi:=(h*xi-xr)/yi end
      end;
    procedure exc (m); value m; integer m;
      begin d[m]:=j;
        if j≠m then begin
          for i:=1 step 1 until k do
```

```
                begin f:=A[i, j]; A[i, j]:=A[i, m];
                  A[i, m]:=f end;
              for i:=l step 1 until n do
                begin f:=A[j, i]; A[j, i]:=
                  A[m, i]; A[m, i]:=f end
                    end
        end;
    B2:=Ba*Ba; l:=1; k:=n;
L1: for j:=k step -1 until 1 do
    begin R:=0;
      for i:=1 step 1 until j-1, j+1 step 1
        until k do
        R:=R+#abs (A[j, i]);
      if R=0 then begin exc (k); k:=k-1;
        goto L1 end
    end;
L2: for j:=l step 1 until k do
    begin c:=0;
      for i:=l step 1 until j-1,
        j+1 step 1 until k do
        c:=c+#abs (A[i, j]);
      if c=0 then begin exc (l); l:=l+1; goto L2
        end
    end;
low:=l; upp:=k;
    for i:=l step 1 until k do d[i]:=1;
iter: no:=0;
```

```
for i:=l step 1 until k do
  begin c:=R:=0;
    for j:=l step 1 until i-1,
      i+1 step 1 until k do
      begin c:=c+#abs (A[j, i]);
        R:=R+#abs (A[i, j]) end;
    g:=R/Ba; f:=1; S:=c+R;
  L3: if c<G then begin f:=f*Ba; c:=c*B2;
    goto L3 end;
    g:=R*Ba;
  L4: if c≥G then begin f:=f/Ba; c:=c/B2;
    goto L4 end;
    if (c+R)/f<0.95*S then begin G:=1/f;
      d[i]:=d[i]*f; no:=1;
    for j:=l step 1 until n do
      A[i, j]:=A[i, j]*G;
    for j:=1 step 1 until k do
      A[j, i]:=A[j, i]*f end
  end;
if no=1 then goto iter;
for m:=low+1 step 1 until upp-1 do
  begin i:=m; x:=0;
    for j:=m step 1 until upp do
      if #abs (A[j, m-1])>#abs (x) then
        begin x:=A[j, m-1]; i:=j end;
    cn[m]:=i;
    if i≠m then begin
```

```
        for j:=m-1 step 1 until n do
          begin y:=A[i, j]; A[i, j]:=A[m, j];
            A[m, j]:=y end;
        for j:=1 step 1 until upp do
          begin y:=A[j, i]; A[j, i]:=
            A[j, m]; A[j, m]:=y end end;
      if x≠0 then
        for i:=m+1 step 1 until upp do
          begin y:=A[i, m-1]; if y≠0 then
            begin y:=A[i, m-1]:=y/x;
            for j:=m step 1 until n do
              A[i, j]:=A[i, j]-y*A[m, j];
            for j:=1 step 1 until upp do
              A[j, m]:=A[j, m]+y*A[j, i]
                end
          end
    end;
  for i:=1 step 1 until n do
    begin for j:=1 step 1 until n do
      v[i, j]:=0;
      v[i, i]:=1
    end;
  for i:=upp-1 step -1 until low+1 do
    begin j:=cn[i];
      for k:=i+1 step 1 until upp do
        v[k, i]:=A[k, i-1];
      if i≠j then begin
```

```
                for k:=i step 1 until upp do
                  begin v[i, k]:=v[j, k];
                    v[j, k]:=0 end;
                v[j, i]:=1 end
            end;
          for i:=1 step 1 until low-1,
            upp+1 step 1 until n do
            begin Wr[i]:=A[i, i]; Wi[i]:=0;
              on[i]:=0 end;
          en:=upp; t:=0;
        nextw: if en<low then goto fin;
          its:=0; na:=en-1;
        nextit:
          for l:=en step -1 until low+1 do
          if #abs (A[l, l-1])≤eps*(#abs(A[l-1,
            l-1])+#abs (A[l, l])) then goto cont1;
          l:=low;
        cont1:x:=A[en, en]; if l=en then goto onew;
          y:=A[na, na]; W:=A[en, na]*A[na, en];
            if l=na then goto Twow;
          if its=30 then begin on[en]:=31; stop end;
          if its=10 ∨ its=20 then begin t:=t+x;
            for i:=low step 1 until en do
              A[i, i]:=A[i, i]-x;
              S:=#abs(A[en, na])+#abs(A[na, en-2]);
              x:=y:=0.75*S; W:=-0.4375*S*S end;
                its:=its+1;
```

```
    for m:=en-2 step -1 until l do
      begin z:=A[m, m]; R:=x-z; S:=y-z;
        P:=(r*S-W)/A[m+1, m]+A[m, m+1];
          Q:=A[m+1, m+1]-z-r-S; r:=
          A[m+2, m+1]; S:=#abs (P)+#abs
          (Q)+#abs(R); P:=P/S; Q:=Q/S;
          R:=R/S; if m=l then goto cont2;
        if #abs (A[m, m-1])*(#abs (Q)+#abs
          (R))<=eps*#abs (P)*(#abs (A[m-1,
          m-1])+#abs (z)+#abs (A[m+1, m+
          1])) then goto cont2
      end;
cont2:
    for i:=m+2 step 1 until en do A[i, i-2]:=0;
    for i:=m+3 step 1 until en do A[i, i-3]:=0;
    for k:=m step 1 until na do
      begin no:=if k≠na then 1 else 0;
        if k≠m then begin p:=A[k, k-1]; q:=
          A[k+1, k-1]; R:=if no=1 then A[k+
          2, k-1] else 0; x:=#abs (P)+#abs
          (Q)+#abs (R); if x=0 then goto cont3;
          P:=P/x; Q:=Q/x; R:=R/x end;
        S:=#sqrt (P*P+q*q+R*R);
          if P<0 then S:=-S;
        if k≠m then A[k, k-1]:=-S*x else
          if l≠m then A[k, k-1]:=-A[k, k-1];
          P:=P+S; x:=P/S; y:=Q/S; z:=R/S;
```

```
      Q:=Q/P; R:=R/P;
    for j:=k step 1 until n do
    begin P:=A[k, j]+Q*A[k+1, j];
      if no=1 then begin
      P:=P+R*A[k+2, j];
        A[k+2, j]:=A[k+2, j]-P*z end;
      A[k+1, j]:=A[k+1, j]-P*y;
        A[k, j]:=A[k, j]-P*x
    end;
  j:=if k+3<en then k+3 else en;
  for i:=1 step 1 until j do
    begin P:=x*A[i, k]+y*A[i, k+1];
      if no=1 then begin
      P:=P+z*A[i, k+2];
        A[i, k+2]:=A[i, k+2]-P*R end;
      A[i, k+1]:=A[i, k+1]-P*q;
        A[i, k]:=A[i, k]-P
    end;
  for i:=-low step 1 until upp do
    begin P:=x*v[i, k]+y*v[i, k+1];
      if no=1 then begin
      P:=P+z*v[i, k+2];
        v[i, k+2]:=v[i, k+2]-P*R end;
      v[i, k+1]:=v[i, k+1]-P*q;
        v[i, k]:=v[i, k]-P
    end;
cont3: end; goto nextit;
```

```
onew: WR[en] := A[en, en] := x+t; Wi[en] := 0;
  cn[en] := its; en := na; goto nextw;
twow: P := (y-x)/2; Q := P*P+W;
  z := #sqrt (#abs(Q));
  x := A[en, en] := x+t; A[na, na] := y+t;
    cn[en] := -its; cn[na] := its;
  if q>0 then begin z := if P<0 then P-z else
    P+z; Wr[na] := x+z; Wr[en] := S :=
    x-W/z; Wi[na] := Wi[en] := 0; x :=
    A[en, na]; R := #sqrt(x*x+z*z); P := x/R;
    Q := z/R;
  for j := na step 1 until n do
    begin z := A[na, j]; A[na, j] := Q*z+
      P*A[en, j]; A[en, j] := Q*A[en, j]-P*z
    end;
  for i := 1 step 1 until en do
    begin z := A[i, na]; A[i, na] := Q*z+P*A[i,
      en]; A[i, en] := Q*A[i, en]-P*z
    end;
  for i := low step 1 until upp do
    begin z := v[i, na]; v[i, na] := Q*z+P*v[i,
      en]; v[i, en] := Q*v[i, en]-P*z
    end end else begin Wr[na] := Wr[en] :=
      x+P;
  Wi[na] := z; Wi[en] := -z end;
  en := en-2; goto nextw;
fin: norm := 0; k := -1;
```

```
for i:=1 step 1 until n do
  begin for j:=k step 1 until n do
      norm:=norm+#abs (A[i, j]);
    k:=i
  end
for en:=n step -1 until 1 do
  begin P:=WR[en]; Q:=Wi[en]; na:=en-1;
    if Q=0 then begin m:=en; A[en, en]:=1;
      for i:=na step -1 until 1 do
        begin W:=A[i, i]-P; R:=A[i, en];
          for j:=m step 1 until na do
            R:=R+A[i, j]*A[j, en];
        if Wi[i]<0 then begin z:=W; S:=r
          end else begin m:=i; if Wi[i]=0
          then A[i, en]:=-r/(if W≠0 then W
          else eps*norm) else begin
          x:=A[i, i+1]; y:=A[i+1, i];
          Q:=(Wr[i]-P)↑2+Wi[i]↑2;
            A[i, en]:=t:=(x*S-z*r)/q;
            A[i+1, en]:=if #abs (x)>#abs (z)
            then(-R-W*t)/x else (-S-y*t)/z
            end end end else if q<0 then
            begin m:=na; if #abs (A[en, na])
            >#abs (A[na, en]) then begin
            A[na, na]:=-(A[en, en]-P)/
            A[en, na]; A[na, en]:=-q/A[en, na]
            end else
```

```
cdiv(−A[na, en], 0, A[na, na]−P, Q,
  A[na, na], A[na, en]);
    A[en, na]:=1; A[en, en]:=0;
for i:=na−1 step −1 until 1 do
  begin W:=A[i, i]−P;
    ra:=A[i, en]; Sa:=0;
    for j:=m step 1 until na do
      begin
        ra:=ra+A[i, j]*A[j, na];
        Sa:=Sa+A[i, j]*A[j, en]
    end;
  if Wi[i]<0 then begin z:=W;
    r:=ra; S:=Sa end
    else begin m:=i; if Wi[i]=0
      then
    cdiv (−ra, −Sa, W, q, A[i, na],
      A[i, en]) else begin
    x:=A[i, i+1]; y:=A[i+1, i];
    vR:=(WR[i]−P)↑2+Wi[i]↑2−
    Q*Q; vi:=(WR[i]−P)*2*q;
    if vR=0∧vi=0 then vR:=
      eps*norm*(#abs (W) + #abs (Q)
      + #abs (x) + #abs (y) + #abs
      (z)); cdiv (x*r−z*ra+Q*Sa,
      x*S−z*Sa−Q*Ra, vr, vi,
      A[i, na], A[i, en]); if #abs(x)>
      #abs (z) + #abs (Q) then begin
```

```
                A[i+1, na]:=(-ra-W*A[i,
                na]+Q*A[i, en])/x; A[i+1,
                en]:=(-Sa-W*A[i, en]-
                Q*A[i, na])/x end else
                cdiv(-r-y*A[i, na], -S-
                y*A[i, en], z, Q, A[i+1, na],
                A[i+1, en])
              end end
            end end
  end;
for i:=1 step 1 until low-1,
  upp+1 step 1 until n do
  for j:=i+1 step 1 until n do
    v[i, j]:=A[i, j];
for j:=n step -1 until low do
  begin m:=if j≤upp then j else upp;
    l:=j-1;
    if Wi[j]<0 then begin
      for i:=low step 1 until upp do
        begin y:=z:=0;
          for k:=low step 1 until m do
            begin y:=y+v[i, k]*A[k, l];
              z:=z+v[i, k]*A[k, j]
            end;
          v[i, l]:=y; v[i, j]:=z
        end end else if Wi[j]=0 then
      for i:=low step 1 until upp do
```

```
                begin z:=0;
                  for k:=low step 1 until m do
                    z:=z+v[i, k]*A[k, j];
                  v[i, j]:=z
                end
            end;
        for i:=low step 1 until upp do
          begin S:=d[i];
            for j:=1 step 1 until n do
              v[i, j]:=v[i, j]*S
          end;
        for i:=low-1 step -1 until 1,
          upp+1 step 1 until n do
          begin k:=d[i]; if k≠i then
            for j:=1 step 1 until n do
              begin S:=v[i, j]; v[i, j]:=v[k, j];
                v[k, j]:=S end
          end
    end;
```

十六　解广义特征值问题的子空间迭代法

(一)功能

对广义特征值问题

$$AX=\lambda BX,$$

程序采用子空间迭代法求其前 n_0 个特征值 $\lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_{n_0}$ 及相应的特征向量 $X_1, X_2, \cdots, X_{n_0}$. 步骤如下：

1. 用 n_0 个线性无关列向量构成初始迭代矩阵 X_0.

2．进行并行反幂迭代，第 k 次迭代公式是

$$A\overline{X}_k=BX_{k-1}.$$

3．形成投影矩阵

$$A_k=\overline{X}_k^TA\overline{X}_k,$$

$$B_k=\overline{X}_k^TB\overline{X}_k.$$

4．求解投影矩阵的特征值问题

$$A_kZ_k=\lambda_kB_kZ_k.$$

5．计算第 $k+1$ 次迭代矩阵

$$X_k=\overline{X}_kZ_k.$$

如此，$k=1, 2, \cdots$ 重复 2～5 的过程，只要 X_0 满足一定条件，必有：$\lambda_k\to\lambda$，$X_k\to X$.

在实际计算中，判定 $|\lambda_{jk}-\lambda_{j,k-1}|/\lambda_{jk}<s$ $(j=1, 2, \cdots, n_0)$，作为过程的结束.

（二）使用方法

语句形式：SBIEV $(n, n_0, n_1, ns,$ eps, $A, d, B, Ak,$ $Bk, WI, WJ, Y0, Y1, YC, DL, ER, QL)$；

其中，n 表示矩阵 A、B 的阶数，值参数；n_0 表示要求特征值、特征向量的个数，值参数；n_1 表示所选子空间的维数，值参数（一般可取 $n_1=\min\{2n_0, n_0+8\}$）；eps 表示特征值精度的控制值，值参数；$A[1:ns]$、$B[1:ns]$ 表示按“一维变带形式”存贮矩阵 A、B 元素的数组；$d[0:n]$ 表示存放对角元在一维数组中序号的数组（“一维变带存贮”形式见 p. 125）；$A_k[1:n_1, 1:n_1]$、$B_k[1:n_1, 1:n_1]$、$WJ[1:n_1]$、$DL[1:n_1]$、$er[1:n_1]$、$Y0[1:n, 1:n_1]$、$YC[1:n, 1:n_1]$ 为工作单元组，$WI[1:n_1]$ 的前 n_0 个元素和 $Y1[1:n, 1:n_1]$ 的前 n_0 列分别存放所求的特征值、特征向量.

另外，调用本过程时，要用到“求实对称矩阵全部特征值

和特征向量的 QL 算法”(见 p 156). 使用者须在源程序说明部分给出.

(三)程序

```
procedure SBIEV (n, n0, n1, ns, eps, A, d, B, Ak,
                 Bk, WI, WJ, Y0, Y1, YC, DL, ER, QL);
  value n, n0, n1, ns, eps;
  integer n, n0, n1, ns; real eps;
  array A, d, B, Ak, Bk, WI, WJ, Y0, Y1, YC,
        DL, ER, procedure QL;
begin integer i, j, k, i0, mi, j0, mj, mij, ij;
  real h1, h2, s, ys, yr, yp, ym, e, eps1;
  procedure MATT (in, jn, kn, X, Y, Z, G1, G2);
      value in, jn, kn, G1, G2;
        integer in, jn, kn, G1, G2;
      array X, Y, Z;
    begin integer i, j, k, j0;
      for i:=1 step 1 until in do
        begin j0:=if G1=0 then 1 else i;
          for j:=j0 step 1 until jn do
            begin Z[i, j]:=0;
              for k:=1 step 1 until kn do
                Z[i, j]:=if G2=0 then Z[i, j]+
                  X[i, k]*Y[k, j]
                  else Z[i, j]+X[k, i]*Y[k, j]
            end
        end
    end;
```

```
procedure MULT (nj, ni, d, B, Y, Z);
    value nj, ni; integer nj, ni;
      array d, B, Y, Z;
  begin integer i, j, k, mB;
    for j:=1 step 1 until nj do
      for i:=1 step 1 until ni do
        begin Z[i, j]:=0;
          for k:=1 step 1 until i do
            begin mB:=i-(d[i]-d[i-1])+1;
              if k≥mB then Z[i, j]:=Z[i, j]
                +B[d[i]-i+k]*Y[k, j]
            end;
          for k:=i+1 step 1 until ni do
            begin mB:=k-(d[k]-d[k-1])+1;
              if i≥mB then Z[i, j]:=Z[i, j]+
                B[d[k]-k+i]*Y[k, j]
            end
        end
    end;
L1: for i:=1 step 1 until n1 do
    WI[i]:=0;
  for i:=1 step 1 until n do
    begin Y1[i, 1]:=0;
      for j:=1 step 1 until n1 do
        Y0[i, j]:=if j=1 then 1 else 0
    end;
  for j:=2 step 1 until n1 do
```

```
      begin h1:=0;
        for i:=2 step 1 until n do
          if Y1[i, 1]=0 then begin
            h2:=B[d[i]]/A[d[i]];
              if h2>h1 then begin
            h1:=h2; i0:=i end end;
        Y1[i0, 1]:=i0; Y0[i0, j]:=1
      end;
L2:   YS:=YR:=0;
      for i:=1 step 1 until ns do
        begin if #abs (A[i])>ys then ys:=#abs
          (A[i]);
          if #abs (B[i])>yr then yr:=#abs(B[i])
        end;
      for i:=1 step 1 until ns do
        begin A[i]:=A[i]/ys; B[i]:=B[i]/yr end;
L3:   for i:=1 step 1 until n do
        begin i0:=d[i]-i; mi:=i-(d[i]-d[i-1])
          +1;
          for j:=mi step 1 until i do
            begin j0:=d[j]-j;
              mj:=j-(d[j]-d[j-1])+1;
              mij:=if mi≥mj then mi else mj;
              ij:=i0+j; S:=0;
              for k:=mij step 1 until j-1 do
                S:=S+A[i0+k]*A[d[k]]*A[j0+k];
              A[ij]:=A[ij]-S;
```

```
                    if j=i then goto L31;
                  A[ij]:=A[ij]/A[d[j]]
                end;
          L31: end;
        MULT (n1, n, d, B, Y0, Y1);
     L4: for i:=1 step 1 until n do
           for j:=1 step 1 until n1 do
             Y0[i, j]:=Y1[i, j];
     L5: for i:=1 step 1 until n do
         begin i0:=d[i]-i;
           mi:=i-(d[i]-d[i-1])+1;
           for j:=mi step 1 until i do
             begin ij:=i0+j; if j=i then goto L51;
               for k:=1 step 1until n1 do
                 Y1[i, k]:=Y1[i, k]-A[ij]*A[d[j]]
                   *Y1[j, k]
             end;
         L51: for k:=1 step 1 until n1 do
                Y1[i, k]:=Y1[i, k]/A[d[i]]
         end;
         for i:=n step -1 until 1 do
           begin i0:=d[i]-i; mi:=i-(d[i]-d[i-1])
             +1;
             for j:=mi step 1 until i-1 do
               for k:=1 step 1 until n1 do
                 Y1[j, k]:=Y[j, k]-A[i0+j]*Y1[i, k]
           end;
```

```
    MULT (n1, n, d, B, Y1, YO);
    MATT (n1, n1, n, Y1, Y0, Ak, 1, 1);
    MATT (n1, n1, n, Y1, YO, Bk, 1, 1);
L6: yp:=ym:=-0;
    for i:=1 step 1 until n1 do
      for j:=i step 1 until n1 do
      begin if #abs (Ak[i, j])>yp
        then yp:=#abs (Ak[i, j]);
        if #abs(Bk[i, j])>ym
          then ym:=#abs (Bk[i, j])
      end;
    for i:=1 step 1 until n1 do
      for j:=i step 1 until n1 do
        begin Ak[i, j]:=Ak[i, j]/yp;
          Bk[i, j]:=Bk[i, j]/ym end;
L7: for i:=1 step 1 until n1 do
        for j:=i step 1 until n1 do
          begin S:=Bk[i, j];
            for k:=i-1 step -1 until 1 do
              S:=S-Bk[i, k]*Bk[j, k];
            if j=i then e:=dl[i]:=#sqrt (s)
              else Bk[j, i]:=s/e
          end;
      for i:=1 step 1 until n1 do
        begin e:=dl[i];
          for j:=i step 1 until n1 do
            begin S:=Ak[i, j];
```

```
                for k:=i-1 step -1 until 1 do
                  S:=S-Bk[i, k]*Ak[j, k];
                Ak[j, i]:=s/e
              end
          end;
        for j:=1 step 1 until n1 do
          for i:=j step 1 until n1 do
            begin S:=Ak[i, j];
              for k:=i-1 step -1 until j do
                S:=S-Ak[k, j]*Bk[i, k];
              for k:=j-1 step -1 until 1 do
                S:=S-Ak[j, k]*Bk[i, k];
              Ak[i, j]:=S/dl[i]
            end;
L8:     QL(n1, Ak, WJ, er, 10-7);
        for j:=1 step 1 until n1 do
          for i:=n1 step -1 until 1 do
            begin S:=Ak[i, j];
              for k:=i+1 step 1 until n1 do
                S:=S-Bk[k, i]*Ak[k, j];
              Ak[i, j]:=S/dl[i]
            end;
L9:     for i:=1 step 1 until n1 do
            WJ[i]:=WJ[i]*yp/ym;
        for i:=1 step 1 until n do
          for j:=1 step 1 until n1 do
            Y0[i, j]:=Y1[i, j];
```

```
    MATT (n, n1, n1, YC, Ak, Y1, 0, 0);
    for j:=1 step 1 until n1 do
      begin S:=0;
        for i:=1 step 1 until n do
          if #abs(Y1[i, j])>#abs(s)
            then S:=Y1[i, j];
        for i:=1 step 1 until n do
          Y1[i, j]:=Y1[i, j]/S
      end;
L10:  eps1:=0;
    for i:=1 step 1 until n0 do
      begin S:=#abs((WJ[i]-WI[i])/WJ[i]);
          if S>eps1 then eps1:=S
        end;
    for i:=1 step 1 until n1 do
        WI[i]:=WJ[i];
    if eps1>eps then goto L4;
    MATT (n, n1, n1, Y0, Ak, Y1, 0, 0)
    end;
```

十七　求高次代数方程全部实根的牛顿迭代法(*)

(一)功能

给定实系数高次代数方程

$$p(x)=a_0x^n+a_1x^{n-1}+\cdots+a_n=0$$

(设 $a_0>0$),

程序用牛顿法按如下计算步骤求出全部实根:

1) 确定实根上界 $x_0=1+\sqrt[k]{\frac{B}{a_0}}$. 其中，$k\geqslant 1$ 是第一个负系数的下标，B 为所有负系数的绝对值的最大值，a_0 为最高次项的系数(设为正).

2) 迭代

$$x^{(i)}=x^{(i-1)}-\frac{f(x^{(i-1)})}{f'(x^{(i-1)})},$$

直到 $|x^{(i)}-x^{(i-1)}|\leqslant \mathrm{eps}\cdot|x^{(i)}|$ 为止.

3) 把方程降阶，继续求下一个根.

(二)使用说明

语句形式：$Nm(a, n, z, \mathrm{eps}, b, p, q)$.

其中，n 表示方程最高次数，值参数；$a[0:n]$表示方程系数；eps 表示要求满足的相对误差，值参数；$z[1:n]$中存放求得的实根；b、p、$q[0:n-1]$表示所需的三套工作单元.

(三)程序

```
procedure Nm(a, n, z, eps, b, p, q);
    value n, eps; integer n; real eps;
      array a, z, b, p, q;
  begin real aa, pp, qq, x0, x1; integer i, m, s;
    procedure horner (p, q, n, x, pp, qq);
      value n, x; real pp, x, qq; integer n;
        array p, q;
      begin real s, s1; integer i;
        s:=s1:=0;
        for i:=0 step 1 until n-1 do
          begin s:=s×x+p[i];
            s1:=s1×x+q[i] end;
        pp:=s×x+p[n]; qq:=s1
```

```
    end;
  p[0]:=aa:=a[0]; x0:=pp:=0;
    s:=#sign(a[0]);
  for i:=1 step 1 until n do
  if s×a[i]<0 then begin if pp=0 then pp:=i;
    if x0<#abs(a[i]) then x0:=#abs(a[i]) end;
  x0:=if pp=0 then 0 else
    1+#exp(#ln(#abs(x0/aa))/pp);
  for i:=0 step 1 until n-1 do
    b[i]:=(n-i)×a[i];
  for m:=1 step 1 until n do
    begin Li: horner(a, b, n, x0, pp, qq);
      x1:=x0-pp/qq;
      if #abs(x1-x0)>eps×#abs(x1)
        then begin x0:=x1; goto Li end;
      z[m]:=x1;
      pp:=b[0]:=b[0]-aa; q[0]:=pp;
      if m<n then begin
        for i:=1 step 1 until n-1 do
          begin pp:=p[i]:=x1×p[i-1]+a[i];
            pp:=b[i]:=b[i]-pp;
            q[i]:=x1×q[i-1]+pp
          end;
        horner(p, q, n-1, x1, pp, qq);
        x0:=x1-pp/qq end
    end
end;
```

十八　求实函数零点的赛勃法

(一)功能

本程序用赛勃(Chambers)法求实函数 $f(x)$ 的零点. 原理是: 设(x_{n-1}, y_{n-1}), (x_n, y_n), (x_n^*, y_n^*)是曲线 $y=f(x)$上不同的三点,作通过这三点的插值多项式:

$$x=x_{n-1}\frac{(y-y_n)(y-y_n^*)}{(y_{n-1}-y_n)(y_{n-1}-y_n^*)}+x_n\frac{(y-y_{n-1})(y-y_n^*)}{(y_n-y_{n-1})(y_n-y_n^*)}$$
$$+x_n^*\frac{(y-y_{n-1})(y-y_n)}{(y_n^*-y_{n-1})(y_n^*-y_n)}.$$

于是可取 $y=0$ 的 x 值作为迭代新值,即

$$x_{n+1}=\frac{x_n^*y_{n-1}y_n}{(y_n^*-y_{n-1})(y_n^*-y_n)}+\frac{x_{n-1}y_n^*y_n}{(y_{n-1}-y_n^*)(y_{n-1}-y_n)}$$
$$+\frac{x_ny_n^*y_{n-1}}{(y_n-y_n^*)(y_n-y_{n-1})},$$

其中,x_n^* 的函数关系可选为

$$x_n^*=\begin{cases}(x_{n-1}+x_n)/2, & |y_n-y_{n-1}|<\varepsilon,\\ x_n-\dfrac{x_n-x_{n-1}}{y_n-y_{n-1}}y_n, & |y_n-y_{n-1}|\geqslant\varepsilon.\end{cases}$$

然后,对每一迭代步作判别

1. $|x_{n+1}-x_n|/|x_n|<ep_1$,

2. $|f(x_{n+1})|$和$|f(x_{n+1})|\Big/\prod_{i=1}^{rtc}(x_{n+1}-x_i^*)$都小于 ep_2.

其中,rtc 是已求出的根数,x_i^* 是已求出的第 i 个根.

若1、2两式中有一式成立, 即认为 x_{n+1} 是求得的一个根.

(二)使用说明

语句形式: *chre*(*p1*, *p2*, *mx*, *nr*, *ep1*, *ep2*, *fct*, *ind*, *rt*);

其中, *p1*、*p2* 是预先给定的初值,值参数; *mx* 是允许的最大迭

代次数，值参数；*nr* 是要求根的个数，值参数；*ep*1、*ep*2 是允许误差，值参数；*fct* 是计算函数值的函数过程，形式要求是 **real procedure** *fct*(*x*)；*ind*[1:*nr*]是计算结果的指示信息：

$$ind[l]=\begin{cases}0, \text{表示第 } l \text{ 个根经 } mx \text{ 次迭代仍未满足精度},\\ 1, \text{表示第 } l \text{ 个根已经求得}, \ l=1, 2, \cdots, nr.\end{cases}$$

rt[1:*nr*]是存放所求零点的数组。

（三）程序

```
procedure chre (p1, p2, mx, nr, ep1, ep2, fct, ind,
  rt);
    value p1, p2, nr, mx, ep1, ep2; integer mx, nr;
      real p1, p2, ep1, ep2;
      array ind, rt; real procedure fct;
  begin real ot, dn, fot, x1, x2, x3, f1, f2, f3, ra,
    fu, t1;
    integer i, c1, rtc, itc;
      switch j:=m2, m3, m4, m7;
    star: for i:=1 step 1 until nr do rt[i]:=0;
      rtc:=0;
    m0: c1:=itc:=0; ot:=p1;
    m1: dn:=1;
      for i:=1 step 1 until rtc do
        begin ra:=ot-rt[i]; if #abs (ra)<10-7
          then begin
          if c1=0 then p1:=ot:=ot+0.1 else if
            c1=1 then
          p2:=ot:=ot+0.1 else goto fin; goto
            m1 end;
```

```
        dn: = dn*ra
      end;
  c1: = c1+1; t1: = fct(ot); fot: = t1/dn; goto j[c1];
m2: f2: = fot; ot: = p2; goto m1;
m3: f3: = fot; x2: = p1; x3: = p2;
m5: x1: = if #abs (f3-f2) <10-7 then
    (x3+x2)/2 else x3-f3*(x3-x2)/(f3-f2);
    ot: = x1; c1: = 2; goto m1;
m4: f1: = fot; if #abs (f1-f2) <10-7 ∨ #abs
    (f2-f3) <10-7 ∨ #abs (f1-f3) <10-7 then
    begin t1: = 2*x3-x2; goto m6 end else t1: =
    f2/(f1-f2)*f3/(f1-f3)*x1+f1/(f2-f1)*f3/
    (f2-f3)*x2+f1/(f3-f1)*f2/(f3-f2)*x3;
m6: itc: = itc+1; x2: = x3; f2: = f3; ot: = x3: = t1;
    c1: = 3; goto m1;
m7: f3: = fot; fu: = fct(x3);
      t1: = if #abs (x2) <10-8 then x3 else
        (x3-x2)/x2;
      if #abs (t1) ≤ep1 ∨ #abs (f3) ≤ep2 ∧ #abs
        (fu) ≤ep2 then goto fin;
        if itc≥mx then goto fim else
          goto m5;
    fin: ind[rtc+1]: = 1; goto m8;
    fim: ind[rtc+1]: = 0;
m8: rtc: = rtc+1; rt[rtc]: = x3;
      if rtc≥nr then goto en; goto m0;
en: end;
```

十九　用拟牛顿法解非线性方程组

(一)功能

对非线性方程组

$$f_i(x_1, x_2, \cdots, x_n)=0 \quad (i=1, 2, \cdots, n),$$

程序用拟牛顿法按如下步骤求其一组实根 $x_1, x_2, \cdots, x_n$.

给定 $|f_i|$ $(i=1, 2, \cdots, n)$ 所允许的误差 $\varepsilon>0$, 从初始向量 $X_0^{(0)}=(x_1^{(0)}, x_2^{(0)}, \cdots, x_n^{(0)})$ 出发, 设迭代进行到第 k 次:

$$X_0^{(k)}=(x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}),$$

如满足　　$|f_i(X_0^{(k)})|<\varepsilon \quad (i=1, 2, \cdots, n)$,

则 $X_0^{(k)}$ 即为一组解; 否则, 构造 n 个新点

$$X_i^{(k)}=(x_1^{(k)}, \cdots, x_{i-1}^{(k)}, x_i^{(k)}+h^{(k)}, x_{i+1}^{(k)}, \cdots, x_n^{(k)})$$
$$(i=1, 2, \cdots, n),$$

其中 $h^{(k)}$ 是第 k 步的步长参数, 然后对每一个 f_i, 在点 $X_0^{(k)}$ 和 $X_i^{(k)}$ 作新的一次内插多项式 $W_i^{(k)}(X)$, 使

$$W_i^{(k)}(X_j^{(k)})=f_i(X_j^{(k)})$$
$$(j=0, 1, \cdots, n;\ i=1, 2, \cdots, n).$$

显然, 这个多项式形如

$$W_i^{(k)}(X)=f_i(X_0^{(k)})+\sum_{j=1}^{n}\frac{[f_i(X_j^{(k)})-f_i(X_0^{(k)})]\cdot(x_j-x_j^{(k)})}{h^{(k)}}$$
$$(i=1, 2, \cdots, n).$$

我们把线性方程组

$$W_i^{(k)}(x_1, x_2, \cdots, x_n)=0 \quad (i=1, 2, \cdots, n)$$

的解作为原方程解的第 $k+1$ 次逼近.

继续迭代, 直至求出满足给定精度的迭代解.

(二)使用说明

语句形式: $NLSENL(n, h, w, ep, ps, a, x, y, f)$.

其中，n 表示方程组的方程个数，值参数；h 表示步长参数，值参数；w 表示每次迭代中乘 h 的因子(可取 0.1)，值参数；ep 表示每个方程所允许的误差，值参数；ps 表示方程组左端允许的最大绝对值，值参数；$a[1:n, 1:n]$，$v[1:n]$ 表示两组工作单元；$X[1:n]$ 开始存放初始向量，过程调用后存方程的近似解．$y[1:n]$ 存放方程组左端在点 X 上的值；$f(X, y)$ 表示计算方程组左端函数在 X 点上值存于 y 中的过程，由使用者自编．

调用方式：

repeat: $k:=NLSENL(n, h, w, ep, ps, A, v, X, y, f)$;
  if $k=-2 \vee k=-3$ **then goto** *repeat* **else**
    if $k=-1$ **then begin** 改变初始 X 或其他参数;
      goto *repeat* **end**
      else 取解 X;

k 的意义是

$$k=\begin{cases} -1, & \text{如方程组左边有一方程的绝对值大于 } ps; \\ -2, & \text{求解化约后的线性方程组为奇异方程组;} \\ -3, & \text{求解化约后的线性方程组的根的和为 1;} \\ m, & \text{达到精度要求时的迭代次数.} \end{cases}$$

(三)程序

```
real procedure NLSENL(n, h, w, ep, ps, A, v, x, y, f);
    value n, h, w, ep, ps; integer n;
      real h, w, ep, ps;
      array A, v, x, y; procedure f;
  begin integer m, i, j, k, b1, b2, i0;
    real al, r, c, t;
    m:=0;
  L1: b1:=1; b2:=0; f(x, y);
```

```
for i:=1 step 1 until n do
  begin v[i]:=r:=y[i]; r:=#abs (r);
    if b1=1∧r<ep then b1:=1 else b1:=0;
    if b2=1∨r>ps then b2:=1 else b2:=0;
  end;
if b1=1 then goto L2; if b2=1 then goto L3;
for i:=1 step 1 until n do
  begin r:=x[i]; x[i]:=r+h; f(x, y);
    for k:=1 step 1 until n do A[k, i]:=y[k];
      x[i]:=r
  end;
for k:=1 step 1 until n do
  begin c:=0;
    for i:=k step 1 until n do
      if #abs (A[i, k])>#abs (c) then begin
        c:=A[i, k]; i0:=i end;
    if #abs (c)≤10-10 then goto L4;
      if i0=k then goto L1;
    for j:=k step 1 until n do
      begin t:=A[k, j]; A[k, j]:=A[i0, j];
        A[i0, j]:=t end;
    t:=v[k]; v[k]:=v[i0]; v[i0]:=t;
L: c:=1/c;
  for j:=k+1 step 1 until n do
    begin A[k, j]:=A[k, j]*c;
      for i:=k+1 step 1 until n do
        A[i, j]:=A[i, j]-A[i, k]*A[k, j]
```

```
        end;
      v[k]:=v[k]*c;
      for i:=k+1 step 1 until n do
        v[i]:=v[i]-v[k]*A[i, k]
      end;
      for i:=n step -1 until 1 do
        for j:=i+1 step 1 until n do
          v[i] :=v[i]-A[i, j]*v[j];
      al:=1;
      for i:=1 step 1 until n do al:=al-v[i];
      if al=0 then goto L5; al:=h/al;
      for i:=1 step 1 until n do x[i]:=x[i]
        -v[i]*al; h:=h*w; m:=m+1; goto L1;
    L2: nlsenl:=m; goto L6;
    L3: nlsenl:=-1; goto L6;
    L4: nlsenl:=-2; goto L6;
    L5: nlsenl:=-3;
    L6: end;
```

二十　用吉尔方法解常微分方程组(定步长)(*)

(一)功能

对常微分方程组

$$\begin{cases} \dfrac{dy_i}{dy_1}=f_i(y_1, y_2, \cdots, y_n), & i=1, 2, \cdots, n, \\ y_i(y_{10})=y_{i0}, \end{cases}$$

其中，

$$y_1=x,\ y_{10}=x_0,\ f_1\equiv 1.$$

程序根据给定步长 h 及第 k 点上的函数值 y_{ik} 应用吉尔方法求出第 $k+1$ 点上的函数值 y_{ik+1}:

$$k_{ij}=f_{ij}=f_i(y_{1\,j-1},\ y_{2\,j-1},\ \cdots,\ y_{n\,j-1}),$$

$$y_{ij}=y_{i\,j-1}+h[a_j(k_{ij}-b_jq_{i\,j-1})],\quad j=1,\ 2,\ 3,\ 4,$$

$$q_{ij}=q_{i\,j-1}+3[a_j(k_{ij}-b_jq_{i\,j-1})]-c_jk_{ij},\quad i=1,\ 2,\ \cdots,\ n,$$

其中,

$$a_1=\frac{1}{2},\quad a_2=1-\sqrt{\frac{1}{2}},\quad a_3=1+\sqrt{\frac{1}{2}},\quad a_4=\frac{1}{6},$$

$$b_1=2,\quad b_2=b_3=1,\quad b_4=2,$$

$$c_1=c_4=\frac{1}{2},\quad c_2=1-\sqrt{\frac{1}{2}},\quad c_3=1+\sqrt{\frac{1}{2}},$$

$$q_{i0}(y_{10})=0,\quad q_{i0}(y_{1k})=q_{i4}(y_{1\,k-1}),\quad k=1,\ 2,\ \cdots.$$

(二)使用说明

语句形式: Gill (n, h, fct, be, y, k, q).

其中, n 表示方程组阶数, 值参数; h 表示积分步长, 值参数; $y[1:n]$ 开始存放第 k 点的函数值 y_{ik}, 最后放第 $k+1$ 点的函数值 y_{ik+1}; k, $q[1:n]$表示所需的两套工作单元; fct 表示计算方程组右端的一般过程说明, 它由 $y[1:n]$的值按方程组右端函数的形式求取 f_i 值送入 $k[1:n]$中; be 表示一个布尔量, 当第一次转入本过程时, be 对应的实在参数中须置 **true**, 从而程序使 q_{i0} 取 0, 并在转出本过程时, 自动在 be 中置 **false**, 以后每次转入本过程后, q_{i0} 就自动取上次运算的结果。

(三)程序

```
procedure Gill (n, h, fct, be, y, k, q);
    value n, h; integer n; real h;
    array y, k, q; boolean be; procedure fct;
  begin integer i, j; real aj, bj, cj, d;
```

```
array a, b, c[1:4];
if be then begin
  for i:=1 step 1 until n do q[i]:=0;
    a[1]:=c[1]:=c[4]:=0.5; b[1]:=b[4]:=2;
    b[2]:=b[3]:=1; a[4]:=1/6;
    a[2]:=c[2]:=1-#sqrt (0.5);
    a[3]:=c[3]:=1+#sqrt (0.5);
    be:=false end;
  for j:=1 step 1 until 4 do
    begin fct (y, k);
      aj:=a[j]; bj:=b[j]; cj:=c[j];
      for i:=1 step 1 until n do
        begin d:=aj×(k[i]-bj×q[i]);
          y[i]:=y[i]+h×d;
            q[i]:=q[i]+3×d-cj×k[i]
        end
    end
end;
```

二十一　用龙格-库塔法解常微分方程组(变步长)(*)

(一)功能

对常微分方程组

$$\begin{cases} \dfrac{dy_i}{dy_1}=f_i(y_1, y_2, \cdots, y_n), & i=1, 2, \cdots, n, \\ y_i(y_{10})=y_{i0}, \end{cases}$$

其中，

$$y_1=x,\ y_{10}=x_0,\ f_1\equiv 1,$$

程序采用龙格-库塔法（自动选取步长 h）由第 k 点的函数值 y_{ik} 算出第 $k+1$ 点的函数值 y_{ik+1}，并给出下一步计算的初始步长，其公式是：

$$k_{i1}=f_i(y_{1k},\ y_{2k},\ \cdots,\ y_{nk}),$$

$$k_{i2}=f_i\left(y_{1k}+\frac{hk_{11}}{2},\ \cdots,\ y_{nk}+\frac{hk_{n1}}{2}\right),$$

$$k_{i3}=f_i\left(y_{1k}+\frac{hk_{12}}{2},\ \cdots,\ y_{nk}+\frac{hk_{n2}}{2}\right),$$

$$k_{i4}=f_i(y_{1k}+hk_{13},\ \cdots,\ y_{nk}+hk_{n3}),$$

$$y_{ik+1}=y_{ik}+\frac{h}{6}(k_{i1}+2k_{i2}+2k_{i3}+k_{i4}).$$

计算过程是先取步长 h，算出 $y_{i\,k+1}^{h}$，再用步长 $h/2$，连续算二步，得到同一点上的值 $y_{ik+1}^{h/2+h/2}$，与 y_{ik+1}^{h} 比较，若不满足精度要求 eps，则再将步长缩小一半用二步算出新的值 $y_{ik+1}^{h/4+h/4}$，与 $y_{ik+1}^{h/2}$ 比较，再判别精度要求，过程直到满足为止.

误差选取时规定：

当 $|y_{ik+1}^{h}|<A$ 时取绝对误差 $\delta_i=|y_{ik+1}^{h}-y_{ik+1}^{h/2+h/2}|$，

当 $|y_{ik+1}^{h}|\geqslant A$ 时取相对误差 $\delta_i=|(y_{ik+1}^{h}-y_{ik+1}^{h/2+h/2})/y_{ik+1}^{h}|$.

另外，程序在计算第 $k+1$ 点时能为计算第 $k+2$ 点值提供初始步长，设第 $k+1$ 点时的步长为 h^*，则规定：

当 $32\cdot\delta_{imax}<$eps，则取 $2h^*$ 作为计算第 $k+2$ 点值时的初始步长.

当 $32\cdot\delta_{imax}\geqslant$eps，则取 h^* 作为计算第 $k+2$ 点值时的初始步长.

（二）使用说明

语句形式：RK(n, h, fct, eps, a, y_0, y_1, y_2, y_3, fc, f)．其中，n 表示方程组阶数，值参数；h 开始存初始步长，最后存计算下一点值的初始步长；A 表示误差控制值；eps 表示要求

的计算精确度，值参数；$y_0[1:n]$开始放第 k 点的函数值 y_{ak}，最后放第 $k+1$ 点的函数值 $y_{1\,k+1}$；y_1, y_2, y_3, fc, $f[1:n]$表示所需五套工作单元；fct 表示计算方程右端的一般过程，它将 $fc[1:n]$中的值作为 y_i，按方程组右端函数的形式算出 f_i 值存进 $f[1:n]$中.

(三)程序

```
procedure RK(n, h, fct, eps, a, y0, y1, y2, y3, fc, f);
    value eps, n, a; integer n; real eps, a, h;
    array y0, y1, y2, y3, fc, f; procedure fct;
  begin integer k; real b, c, d; array e[1:5];
    procedure rkst (x, y); array x, y;
      begin integer k, j;
        for k:=1 step 1 until n do
          fc[k]:=y[k]:=x[k];
        for j:=1 step 1 until 4 do
          begin fct (fc, f);
            for k:=1 step 1 until n do
              begin d:=f[k]×h;
                fc[k]:=x[k]+e[j]×d;
                y[k]:=y[k]+e[j+1]×d/3
              end
          end
      end;
  e[1]:=e[2]:=e[5]:=0.5; e[3]:=e[4]:=1;
    rkst (y0, y3);
  L1: h:=h/2; rkst (y0, y1); rkst (y1, y2); b:=0;
    for k:=1 step 1 until n do
```

```
        begin c:=#abs (y2[k]-y3[k]);
          d:=#abs (y2[k]);
          if d>a then c:=c/d; if c>b then b:=c
        end;
      if b>eps then
        begin
          for k:=1 step 1 until n do
            y3[k]:=y1[k]; goto L1
        end;
      h:=2×h;
      for k:=1 step 1 until n do y0[k]:=y2[k];
      if b<eps/32 then h:=2×h
    end;
```

二十二　逐步回归分析程序

(一)功能

本程序使用逐步回归方法作多元回归分析，它求出：

1．回归方程

$$y=b_0+b_1x_1+\cdots+b_{n-1}x_{n-1}$$

中的回归系数 b_0，b_1，…，b_{n-1}.

2．选入因子对 y 的复相关系数及 y 的标准差 s.

3．回归方程的拟合值及与实测值的偏差.

(二)使用说明

语句形式：$SRA\ (m,\ n,\ F_{1sn},\ xy,\ y,\ A,\ xBa,\ sgm)$.

其中，m 表示样本数，值参数；n 表示因子个数加 1，值参数；F_{1sn} 表示 F 检验的临界值，值参数；$xy[1{:}n,\ 1{:}m]$ 表示存放因子 x_1，x_2，…，x_{n-1} 及预报量 $x_n=y$ 的数组；$y[1{:}m]$ 是一组

工作单元，存放回归方程的拟合值与拟合偏差；$A[1:n, 1:n]$表示相关系数矩阵；$xBA[1:n]$、$sgm[1:n]$表示两组工作单元，开始时分别存放计算所得的$\bar{x}$，σ，以后用来作为因子选上与否的信息，当x_i未选上时，$xBA[i]=i$，$sgm[i]=0$，一旦x_i被选上，$xBA[i]=0$，$sgm[i]=i$.

程序最后印出的i是选上因子的序号，b为对应该因子的回归系数，R为复相关系数，y_n为标准差.

(三)程序

```
procedure SRA (m, n, F1sn, xy, y, A, xBa, sgm);
  value m, n, F1sn;
    integer m, n; real F1sn;
      array xy, y, A, xBa, sgm;
  begin integer k, i, j, t, l;
    real c1, c2, R, yn, h, max, c, s, F1, v1;
    procedure mA;
    begin if A[k, k]<10-5 then
      #print (0, '2Hk=, i5', k); c1:=1/A[k, k];
      for i:=1 step 1 until n do
        if i≠k then begin c2:=A[i, k]*c1;
          for j:=1 step 1 until n do
            if j≠k then A[i, j]:=A[i, j]-
              A[k, j]*c2 end;
      for j:=1 step 1 until n do
        if j≠k then begin A[k, j]:=A[k, j]*c1;
          A[j, k]:=-A[j, k]*c1 end;
      A[k, k]:=c1; R:=#sqrt (1-A[n, n]);
        yn:=h*#sqrt (A[n, n]/(m-l-1))
```

```
    end;
  procedure im (ds); array ds;
    begin
      for i:=1 step 1 until n-1 do
        begin if ds[i]=0 then goto LL;
          v1:=A[i, n]*A[n, i]/A[i, i];
          if v1>max then begin max:=v1;
            k:=i end;
        LL: end
    end;
  procedure pp;
  begin for i:=1 step 1 until n do
      begin c:=0;
        for j:=1 step 1 until m do c:=c+xy[i, j];
        xBA[i]:=c/m; c:=0;
        for j:=1 step 1 until m do
          c:=c+(xy[i, j]-xBA[i])↑2;
        sgm[i]:=#sqrt (c)
      end;
    h:=sgm[n]
  end;
  L1: pp;
    for i:=1 step 1 until n-1 do
      for j:=i+1 step 1 until n do
        begin c:=0;
          for t:=1 step 1 until m do
            c:=c+(xy[i, t]-xBA[i])*(xy[j, t]-
```

```
                xBA[j]);
              A[j, i]:=A[i, j]:=c/(sgm[i]*sgm[j])
            end;
          for i:=1 step 1 until n do
            begin xBa[i]:=i; sgm[i]:=0;
              A[i, i]:=1 end;
          l:=s:=0;
L2:       if l=n-1 then goto L3;
          max:=0; im(xBA); F1:=max*(m-l-2)/
            (A[n, n]-max);
          if F1>F1m then begin xBA[k]:=0;
            sgm[k]:=k; l:=l+1; ma end else goto L3;
          if s=0 then begin s:=1; goto L2 end;
          max:=--10 18; im (sgm); F1:=-max*(m-l
            -1)/A[n, n];
          if F1≤F1m then begin sgm[k]:=0;
            xBA[k]:=k; l:=l-1; ma end;
          goto L2;
L3:       for i:=1 step 1 until n-1 do A[i, 1]:=sgm[i];
          pp; c:=0;
          for i:=1 step 1 until n-1 do
            if A[i, 1]≠0 then begin
              A[i, 2]:=s:=A[i, n]*sgm[n]/sgm[i];
              #print (0, '10x, 2Hi=, i5, 5x, 2HB
                =, F14.5', i, s); c:=c+s*xBA[i] end;
          s:=xBA[n]-c; #print (0, '21x, 3HB0=,
            F14.5, /, 10x, 2HR=, F8.5, 10x,
```

```
    3Hyn=, F14.5', s, R, yn);
  for j:=1 step 1 until m do
    begin l:=s;
      for i:=1 step 1 until n-1 do
        if A[i, 1]≠0 then
          l:=A[i, 2]*xy[i, j]+l;
      y[j]:=l
    end;
  #print(0, '8x, 2HF=, F6.2, /, F8.2', F1sn, y);
  for j:=1 step 1 until m do
    y[j]:=xy[n, j]-y[j];
  #print (0, 'F8.3', y)
end;
```

二十三　用改进的单纯形法解线性规划问题

(一)功能

本程序采用改进的单纯形法解一般的线性规划问题. 即求向量 $X=(x_1, x_2, \cdots, x_n)^T$, 使目标函数

$$f=cX=c_1x_1+c_2x_2+\cdots+c_nx_n,$$

在约束条件　　$x_j\geqslant 0 \quad (j=1, 2, \cdots, n),$

$$AX=B$$

下取极小值. 其中, X 和 B 分别是 n 维和 m 维列向量, c 是 n 维行向量, $A=(a_{ij})$ 为 $m\times n$ 矩阵, 并假定 $b_i\geqslant 0$ $(i=1, 2, \cdots, m)$.

(二)使用说明

语句形式: $Lp\ (m, n, A_1, X, IB, u, xj, ep, w, F)$.

其中, m、n 分别是 A 的行数和列数, 值参数; $A_1[1:p, 1:n+1]$

存放系数矩阵 A 的扩充矩阵($p=m+1$ 或 $p=m+2$, $\bar{a}_{ij}=a_{ij}$, $\bar{a}_{i\,n+1}=b_i$, $a_{m+1\,j}=c_j$, $i=1,2,\cdots,m$, $j=1,2,\cdots,n$); $X[1:p]$ 存放极小解; $IB[1:p]$ 存放关于人工变数的信息和基底描绘, 其中 $IB_i=0$ 或 1, $IB_i=0$ 表示第 i 个方程无需引入人工变数, $IB_i=1$ 表示需引入人工变数; ep 表示精度控制, 一般可取 $_{10}-7$; w 是计算结果指示信息, 若 $w=-1$, 表示给定问题无解; 若 $w=0$, 表示解是无界的; 若问题可行, 则求得问题的解, 存于 X 中, IB 中存基底描绘, 基底描绘和解是按相同次序排列的, 例如 $IB[5]=6$, 则 $x_6=x[5]$. $-X[m+1]$ 是目标函数值; $u[1:p,\ 1:p]$、$xj[1:p]$ 是所需两套工作单元; F 是区分是否有初始基底的信息, 若 $F=1$, 表示有初始基底, 此时输入矩阵 $\bar{A}$ 是 $A_1[1:m+1,\ 1:n+1]$; 若 $F=0$, 表示无初始基底, 此时输入矩阵 $\bar{A}$ 是 $A_1[1:m+2,\ 1:n+1]$.

若求极大值, 则可令 $f_1=-f$.

(三)程序

```
procedure Lp (m, n, A1, X, IB, u, xj, ep, w, F);
  value m, n, ep, F;
    integer m, n, w, F; real ep; array A1, X, IB,
      u, xj;
  begin integer p, i, j, k, l; real r, s;
    p:=if F=1 then m+1 else m+2;
      if F=1 then goto L1;
    for j:=1 step 1 until n+1 do
      begin A1[p, j]:=0;
        for i:=1 step 1 until m do
          if IB[i]=1 then
            A1[p, j]:=A1[p, j]-A1[i, j]
```

```
      end;
L1: for i:=1 step 1 until p do
    for j:=1 step 1 until p do
      if i=j then u[i, j]:=1 else u[i, j]:=0;
    for i:=1 step 1 until p do
      IB[i]:=n+i;
        w:=-1;
L2: for i:=1 step 1 until p do
      begin s:=0;
        for j:=1 step 1 until p do
          s:=s+u[i, j]*A1[j, n+1];
        X[i]:=s
      end;
    if F=1 then begin w:=0; goto L3 end;
    if X[p]+ep≥0 then begin F:=1; p:=p-1;
      w:=0 end;
L3: r:=k:=0;
    for j:=1 step 1 until n do
      begin s:=0;
        for i:=1 step 1 until p do
          s:=s+u[p, i]*A1[i, j];
        if s-r+ep<0 then
          begin r:=s; k:=j end
      end;
    if k=0 then goto L4;
    r:=10^18; l:=0;
    for i:=1 step 1 until p do
```

```
      begin s:=0;
        for j:=1 step 1 until p do
          s:=s+u[i, j]*A1[j, k];
        xj[i]:=s;
        if (s-ep>0)∧(i≤m) then begin
          if X[i]/s-r<0 then begin
            r:=X[i]/s; l:=i end end
        end;
      if l=0 then begin #print(0, '3HW=, i3', W);
        goto L5 end;
      for j:=1 step 1 until p do
        u[l, j]:=u[l, j]/xj[l];
      for i:=1 step 1 until p do
        if i≠l then for j:=1 step 1 until p do
            u[i, j]:=u[i, j]-xj[i]*u[l, j];
      IB[n+l]:=k; goto L2;
    L4: if w<0 then #print (0, '2Hw=, i2', w);
    L5: end;
```

二十四　快速傅里叶变换(FFT)程序

(一)功能

本程序利用快速傅里叶变换方法(FFT)计算复的或实的有限离散傅里叶变换，即计算

$$x_j=\sum_{k=0}^{N-1}F_k e^{-\frac{2\pi i}{N}kj}\quad(j=0,\ 1,\ \cdots,\ N-1).$$

其中，$N=2^M$(M为正整数)，$F_k(k=0,\ 1,\ \cdots,\ N-1)$可为复

数. 相应变换结果 $x_j(j=0, 1, \cdots, N-1)$一般也是复数.

(二)使用说明

语句形式. FFT (m, A, B).

其中，$m=2^N$(N 是变换的点数)，值参数；$A[0:n-1]$、$B[0:n-1]$分别存放 F_k 的实部和虚部，语句调用结束时，在此得到变换结果，即傅里叶系数的实部和虚部.

若变换值 F_k 为实数，则可令其虚部为零，从而得到实数的离散傅里叶变换结果.

(三)程序

```
procedure FFT (m, A, B); value m;
    integer m; array A, B;
  begin integer i, n, k, l, j, j1, j2, jj1, jj2, lc,
    nc1, nc2, n1, n2;
      real a1, b1, c, s, cb, sb, cc, sc, r1, r2, th;
    n:=2↑m;
    for l:=1 step 1 until m do
      begin lc:=(2↑l)÷2; nc1:=n÷lc; nc2:=nc1÷2;
        th:=lc*6.2831853/n; sc:=#sin (th);
          cc:=-2*(#sin (th/2))↑2;
        for k:=1 step 1 until lc do
          begin j1:=(k-1)*nc1; j2:=j1+nc2;
            s:=0; c:=1;
            for j:=0 step 1 until nc2-1 do
              begin jj1:=j1+j; jj2:=j2+j;
                r1:=A[jj1]; r2:=B[jj1];
                A[jj1]:=r1+A[jj2];
                B[jj1]:=r2+B[jj2];
```

```
                    a1:=r1-A[jj2]; b1:=r2-B[jj2];
                    A[jj2]:=a1*c+b1*s; B[jj2]:=
                    b1*c-a1*s; sb:=cc*s+sc*c+s;
                    cb:=cc*c-sc*s+c; s:=sb; c:=cb
                  end
              end
          end;
        j:=0; n1:=n-1; n2:=n÷2;
        for i:=0 step 1 until n1-1 do
          begin if j≤i then goto L1;
            r1:=A[j]; r2:=B[j]; A[j]:=A[i];
              B[j]:=B[i]; A[i]:=r1; B[i]:=r2;
          L1: k:=n2;
          L2: if j<k then goto L3; j:=j-k; k:=k÷2;
            goto L2;
          L3: j:=j+k
        end
    end;
```

第 四 章

计 算 实 习

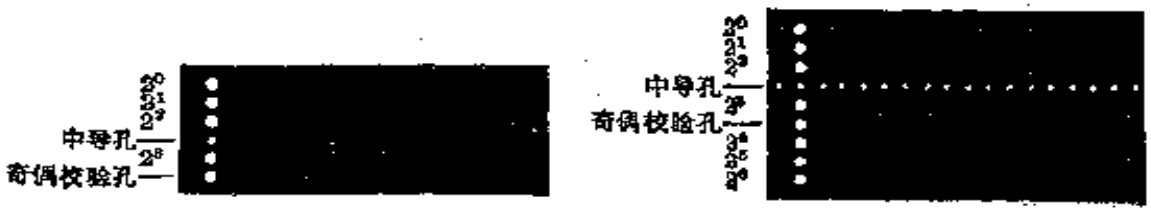
计算问题的源程序编好以后，接着就是使用电子数字计算机进行计算了。这个过程还需要经过两个步骤：一是穿孔，二是上机计算。

第一节 穿 孔

穿孔，就是把源程序和原始数据用穿孔机穿在纸带上。

穿孔纸带是一种狭长的纸质带子，TQ-16 机允许使用两种规格：五单位纸带和八单位纸带，709 机限于使用五单位纸带。

五单位纸带每一排有六个孔位：一个中导孔，一个奇偶校验孔，四个数码孔。八单位纸带每一排有九个孔位：一个中导孔，一个奇偶校验孔，七个数码孔。纸带及其孔位安排如图 25 所示。



五单位纸带　　八单位纸带

图 25　纸带及其孔位安排图

纸带上的中导孔起定位的作用. 奇偶校验孔用以校验输入代码的正确性: 如果同一排上孔的个数(包括奇偶校验孔本身, 但不包括中导孔)是奇数, 则该排孔所代表的信息就输入机器, 否则就认为有错(个别孔例外), 停止输入. 每一个中导孔所在排的数孔, 用以代表数码, 有孔为"1", 无孔为"0". 因此, 同一横排上数孔的不同组合状态, 就可以把数表示出来.

在纸带穿孔时, 有两种方式:

1. 五单位穿孔方式. 这种穿孔方式用一个奇偶校验孔和第1～4位数孔(依次表示 2^0、2^1、2^2、2^3). 若用前3位数孔表示一个八进制数字, 第4位数孔表示一个二进制数字, 那么, 同一排数孔的不同组合状态就可以把八进制数0～17(相当于十进制数0～15)表示出来, 用二排数孔就可以把八进制数00～77表示出来.

五单位穿孔方式可用五单位纸带或八单位纸带穿孔.

2. 八单位穿孔方式. 这种穿孔方式要用到一个奇偶校验孔和第1～7位数孔(依次表示 2^0、2^1、2^2、2^3、2^4、2^5、2^6). 若用第1～3位和第4～6位数孔分别表示一个八进制数字, 第7位数孔表示一个二进制数字(实际使用时, 除个别孔外它总是为0), 这样, 同一排上数孔的不同组合状态, 就可以把八进制数00～77表示出来.

八单位穿孔方式用八单位纸带在CFX-5型穿复校机上穿孔, 目前, 这种穿孔方式限于用来穿孔TQ-16机的源程序.

一 源程序穿孔

源程序是计算机对数据进行加工、运算的一系列命令, 为

了把它输进机器，必须穿成纸带。

(一) 源程序的编码

为了把源程序穿孔成纸带，先要把组成源程序的字符按照一定的编码系统编成数码。这里采用的是符号6编码方法，即每一个字符用两个八进制数码 C_1C_2 表示。字符同八进制数码 C_1、C_2 的对应关系见图26"字符编码表"。例如，字符 a 的编码是14，数字8的编码是10，加号+的编码是46，表达式 x_1+2 的编码是43014602等等。编码时需要注意的是，**begin** 等黑体词的两侧各应增编一个字符 ε，即按 εbeginε 进行编码。

C_1 \ C_2	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	..	10	*A*	*B*	*C*	*D*
2	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
3	*M*	*N*	O	*P*	*Q*	*R*	*S*	*T*
4	*U*	*V*	*W*	*X*	*Y*	*Z*	+	−
5	*	/	÷	↑	∨	∧	¬	<
6	⩽	=	⩾	>	≠	,	:	;
7	#	(	)	[	]	'	∈	␣

图26 字符编码表

为了减少源程序的穿孔工作量，机器允许把源程序中某些基本符号用缩写的形式进行编码，而且，缩写和不缩写的两种方案(或者它们的混合方案)穿成的源程序纸带是同等有效的。基本符号的缩写形式见图27。

现在，举一个源程序的编码例子。

基本符号	缩写形式	基本符号	缩写形式
begin	b	else	el
end	e	true	tr
if	i	false	fa
then	t	stop	st
goto	g	pause	pa
for	f	boolean	bo
step	s	switch	sw
until	u	#read	#W1
do	d	#print	#W2
real	r	#read drum	#W3
integer	in	#write drum	#W4
array	a	#read tape	#W5
procedure	p	#write tape	#W6
value	v		

图 27　基本符号缩写表

[例]　源程序

```
begin
  real x, y;
    x:=10-2; y:=x↑3;
    #print (0, '10', y)
end ∧∧……∧
```

这个源程序编成数码后的形式是

761520222431767635201427764365446743666113470267
44666143530367

70333524313771006575010075654472762031177655555
……55

同样，这个源程序也可写成缩写形式：

```
b
  r x, y;
    x: =10-2; y: =x↑3;
    #W2 (0, '10', y)
e ∧∧……∧
```

编码形式是：

761576763576436544674366611347026744666143530367
70420271006575010075654472762076555555……55

(二)源程序的穿孔

源程序编成数码以后，就可以用穿孔机把它穿成纸带.对于709机，限于使用五单位纸带，按五单位穿孔方式穿孔；对于TQ-16机，可以使用五单位纸带或八单位纸带按五单位穿孔方式穿孔.穿孔时用到的纸带孔型及名称如图28所示.

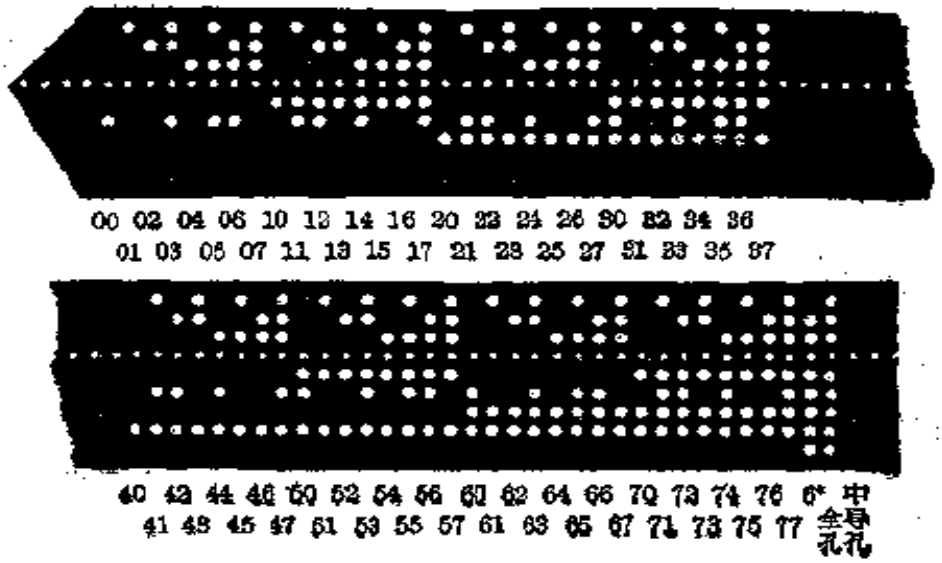
由此看出，五单位穿孔方式用两排孔或八单位穿孔方式用一排孔就可以把每一个符号6编码的字符表示出来.另外，根据特殊的需要，TQ-16机上，把中导孔和全孔看作为废孔，6* 孔用作起终信号；709机上，把空孔看作为废孔，止孔用作终信号.

源程序穿孔时，必须注意的一点是，TQ-16机源程序纸带的首尾必须分别加穿起终信号6*6*；709机源程序纸带的尾部必须加穿终止信号止.

对应于上面例子(缩写方案)的源程序，两种穿孔方式的源程序纸带如图29所示.



五单位纸带孔型



八单位纸带孔型

图 28 五单位纸带和八单位纸带孔型图

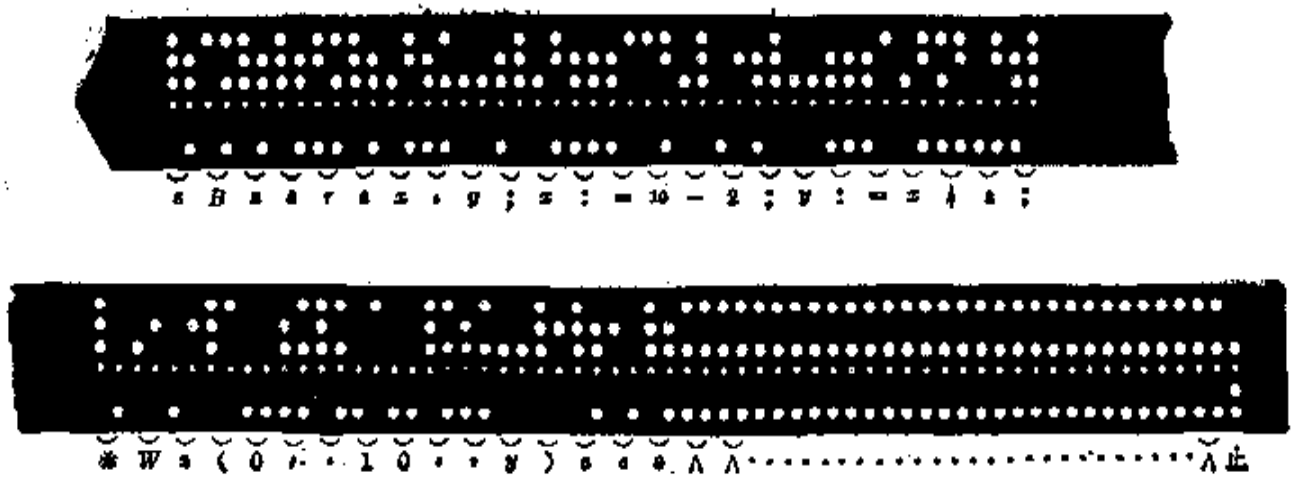
(三)源程序纸带的修改

源程序穿成纸带以后，如果发现错误需要修改，或者上机过程中需要作某些改动，这可以采取如下三种方法：

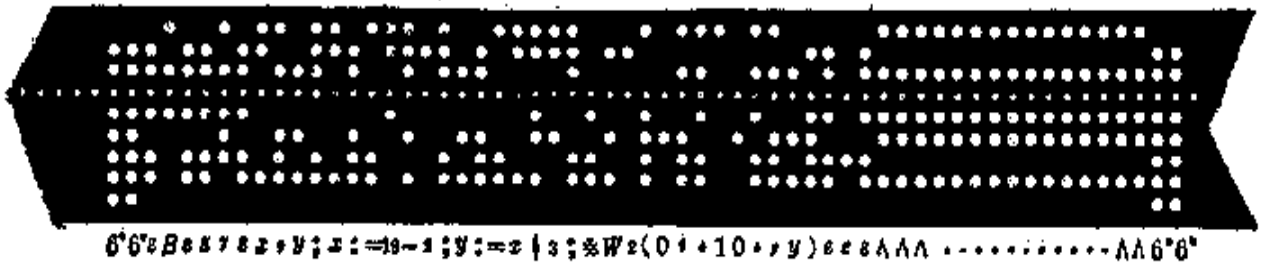
1．手工修改。按照字符编码规则，用剪贴、补孔的办法，手工修改源程序纸带。

2．电传修改。使用控制台打字机发布修改命令，由机器对已输入的源程序进行修改。这一方法只在上机过程中发现少量错误时才被采用，具体修改办法详见第二节四中“几种常用的操作技术”部分。

3．机器修改。由上机人员根据机器对源程序纸带进行检查所提供的出错信息，找出错误，写一个修改方案并穿成纸



五单位纸带的源程序



八单位纸带的源程序(按八单位方式穿孔)

图 29　五单位源程序纸带和八单位源程序纸带(按八单位方式穿孔)的图例

带(称为修改方案纸带)，接在源程序纸带前面，一起输进机器，由机器自动完成对源程序的修改工作.

以上三种修改源程序的方法，709 机限于使用第一种. TQ-16 机则常采用第三种.

下面介绍 TQ-16 机源程序修改方案的编写方法.

1. 页码，行码，字符位置，分号位置.

为了编写源程序的修改方案，并便于机器打印及人工查阅出错位置，在源程序中任何一个说明或语句的前面都可以加上页码和行码. 这些页码和行码同源程序一起穿孔并输入机器. 页码和行码只起标识程序中字符位置的作用，而对程序所描述的算法和执行流程毫无影响.

页码和行码的记号分别是：

$$\#\mathrm{YE}n: \qquad \#\mathrm{HA}m:$$

其中，YE、HA 分别是页和行的汉语拼音缩写. n、m 分别称为页号和行号，它们都是三位数字的十进制数，即

$$000 \leqslant n,\ m \leqslant 999.$$

必须注意的是，页号和行号都要写足三位数字，如 #YE001:，#HA025:，并且，页号以及相邻两个页码之间的行号，只许由小到大使用，但不要求连续. 图 30 是加有页、行码的源程序示意图.

有了页、行码的概念，就可以进一步把源程序中任一字符的字符位置或任一分号的分号位置确定下来. 分号位置或字符位置是这样规定的：先写出分号或字符所在的程序页号，再写出所在的行号，最后写出它们在该行中的序号(即从左数起第几个分号或字符). 例如，图 30 中第一张程序纸上字符 D 的字符位置是 001, 005, 004，第二张程序纸上右下角分号的分号位置是 003, 019, 003.

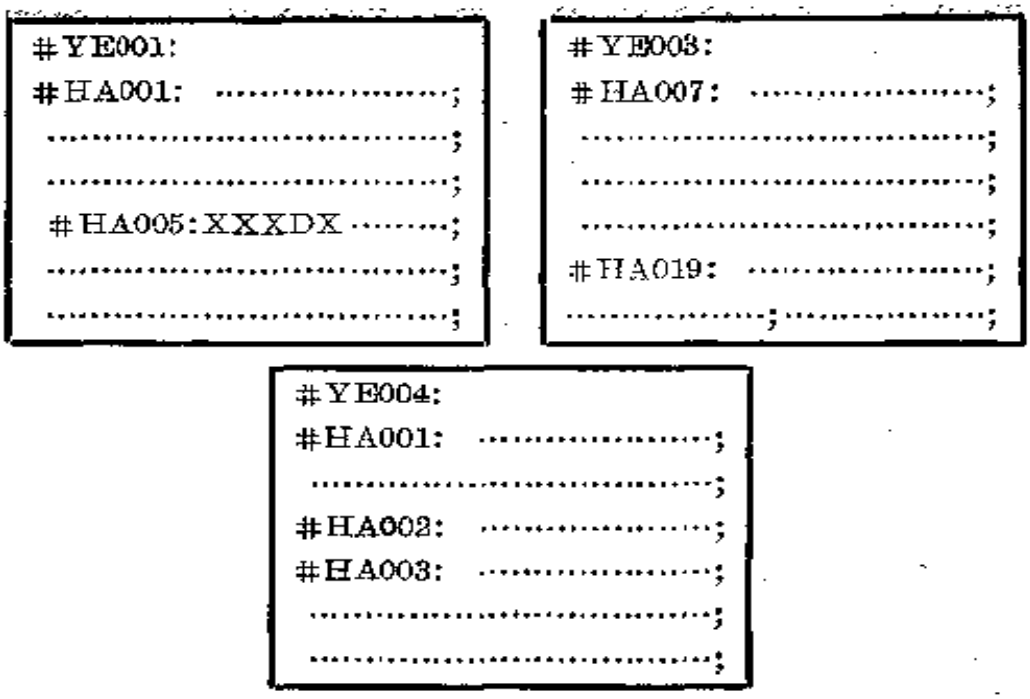


图 30　TQ-16机源程序书写形式示意图

2．修改方案的写法。

TQ-16 机安排有以下三种修改源程序的措施：

(1) 删除．使用这一措施可以将介于两个分号之间的一段源程序或这样的几段源程序全部删去．例如，要将源程序第 1 页第 3 行第 1 个分号到第 1 页第 4 行第 2 个分号之间的一段程序以及第 5 页第 7 行第 2 个分号到第 6 页第 12 行第 3 个分号之间的一段程序全部删去，则可以写成如下形式的删除命令：

#SC (001, 003, 001:001, 004, 002; 005, 007, 002:
006, 012, 003);

其中，SC 是删除的汉语拼音缩写．

删除命令的一般形式是：

#SC(被删除几段的起止分号位置)．

(2) 插入．使用这一措施可以将某些新内容插入到源程

序中某一个分号的后面. 例如, 要在源程序第 12 页第 31 行第 4 个分号后面插入一个停语句 stop; 则可以写成如下形式的插入命令:

#CR 012, 031, 004; *estops*; #WH;

其中, CR 和 WH 分别是插入和完的汉语拼音缩写.

插入命令的一般形式是:

#CR 分号位置; 插入内容 #WH;

(3) 修改. 使用这一措施可以将源程序从某一个字符起到某一字符止的所有字符换成另外一些内容. 例如, 要将源程序从第 25 页第 14 行第 32 个字符起到第 25 页第 15 行第 7 个字符止的所有字符换成一个赋值语句 $M:=K/R$; 则可以写成如下形式的修改命令:

#XG 025, 014, 032:025, 015, 007; $M:=K/R$;
#GB;

其中, XG 和 GB 分别是修改和改毕的汉语拼音缩写.

修改命令的一般形式是:

#XG 起止字符位置; 修改内容 #GB;

根据源程序的出错情况, 选用上面三种修改命令, 把它们连在一起, 就构成源程序的一个修改方案, 机器执行修改方案就可以把已经发现的错误全部改正.

在编写修改方案时应该注意的是, 源程序中诸错误要从前到后依次修改, 即出现在修改方案中的诸页号、行号、分号位置和字符位置必须是由小到大排列着, 互不相交. 另外, 对源程序的修改也只能是一次性的, 即如果修改过的程序在上机时又发现新的错误, 应当把新、旧错误统一起来, 重新编写修改方案再次上机.

带有修改方案的程序例子如下:

6*6* #XG 001, 003, 007;; #GB; ∧∧……∧6*6*

6*6* #YE: 001:

#HA 001:b

#HA 002: r x, y;

#HA 003: $x:=_{10}-2$, $y:=x\uparrow 3$;

#HA 004: #W2 (0, '10', y)

e∧∧……∧6*6*

其中，修改方案是为了把赋值语句 $x:=_{10}-2$ 后面的逗号改正为分号。

二 原始数据穿孔

对于具有原始数据的程序，它的原始数据是由程序执行输入语句输进机器的，为此，也应该把它穿成纸带。由于输入语句允许使用各种输入格式：浮点十进制格式(10 格式)、浮点八进制格式(8 格式)、符号 6 格式(N 格式)、符号 4 格式(普通十进制格式)、紧凑格式(Zn 格式)，因此，数据也应该按照相应格式的要求穿孔。但不管何种格式，原始数据一律用五单位穿孔方式穿孔，穿孔时用到的孔型如图 31 所示(以五单位纸带为例)：

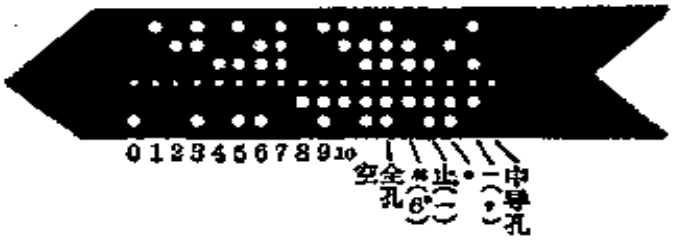


图 31 原始数据纸带孔型图(括号内是 TQ-16 机定义)

(一) 浮点十进制格式(10 格式)

这种输入格式要求把输入的十进制数，写成浮点规格化

形式 $\pm 10^{\pm p_1p_2}0.q_1q_2\cdots\cdots q_{10}$，然后按如下形式穿孔：

$$p_1p_2\ {}^{00}_{11}q_1q_2\cdots\cdots q_{10}$$

其中，$^{00}_{11}$ 分别表示阶数 p_1p_2 和尾数 $0.q_1q_2\cdots\cdots q_{10}$ 的正负号，正穿成 0，负穿成 1.

[例 1]　十进制数 −10.30987654 穿孔为



0201103098765 4

[例 2]　十进制数 0.00008 穿孔为



0 4 1 0 8 0 0 0 0 0 0 0 0

浮点十进制格式穿孔时必须注意下列几点：

1. 穿孔数据的有效数字为 10 位. 不足 10 位用 0 补足；多于 10 位，四舍五入，删成 10 位.

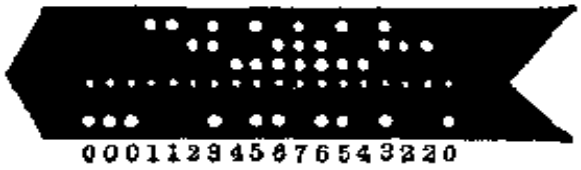
2. 为了便于阅读、校对，一般在两个穿孔数据之间穿一个分隔符号(709 机可选用空孔；TQ-16 机可选用全孔或中导孔).

3. 同一个输入语句中，对应的每一个数组构成一个数据段，相连的简单变量也构成一个数据段. 穿孔时，数据段之间必须留有 0.5 米左右的空白纸带.

(二)浮点八进制格式

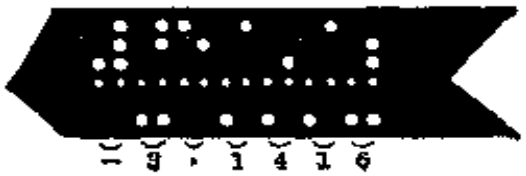
这种输入格式的要求同浮点十进制格式一样，只是被输入的八进制数应是 18 位的浮点规格化形式，且最后一位数字不应超过 1.

[例]　八进制数 −0.1234567654321 7 应穿孔成：



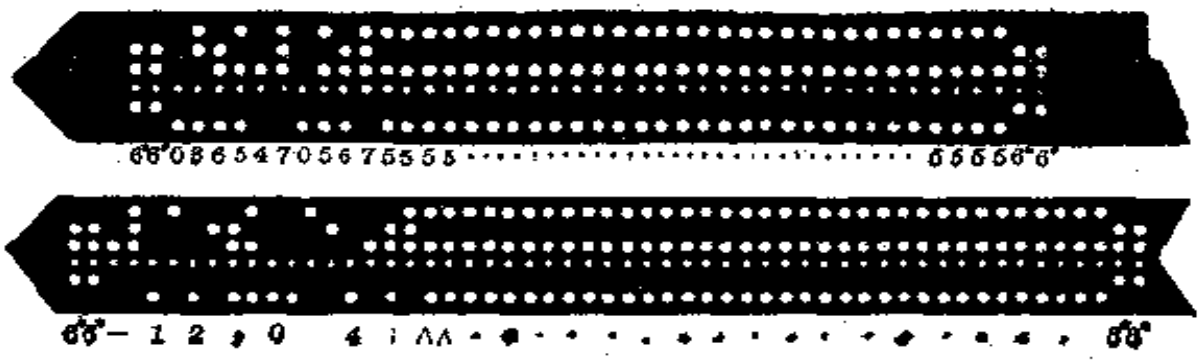
(三) 符号 6 格式，或称 N 格式(TQ-16 机)

符号 6 格式是一种与常用书写方式相同的穿孔格式，它按照源程序字符一样的符号 6 编码方法，一个字符穿成两个数码. 例如，-3.1416 穿孔成：



在使用符号 6 格式输入数据时，在一个输入语句中相连的所有简单变量构成一个或几个数据段，每个数组也构成一个或几个数据段. 每个数据段不得超过 180 个数，数与数之间用逗号分隔，最后一个数后面穿一个分号. 每一个数据段的首尾分别穿起终信号 6*6* 和 $\underbrace{\wedge\wedge\cdots\cdots\wedge}_{15\text{个以上}}$6*6*. 数据段之间也要留有 0.5 米的空白纸带.

例如，输入语句 #read (0, 'N', l, m, p)；其中，l、m 是简单变量，要输的数是 3、-5，p 是有两个分量的数组，要输入的数是 -12, 0.4，则对应于输入语句的数据纸带可穿成两个数据段：



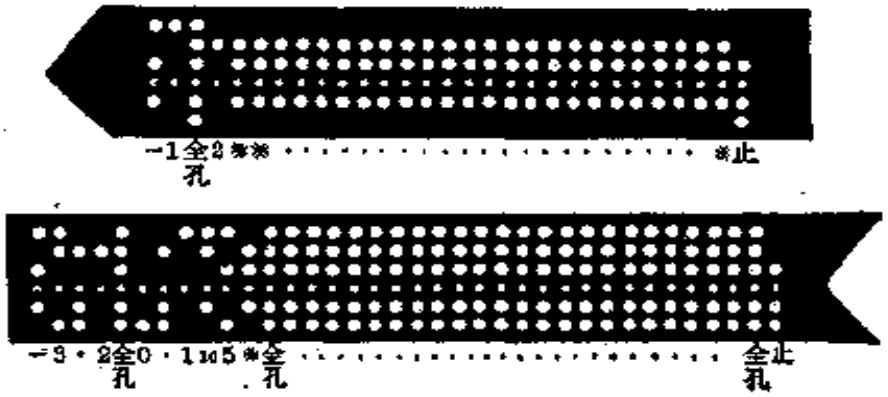
(四)符号4格式(普通十进制格式)

符号4格式输入的数据是普通的十进制浮点或定点数(删去+号)，例如-3.5，$0.4_{10}-5$，120等，数据按五单位方式穿孔，一个字符用一排孔表示．穿孔时，709机和TQ-16机分别提出要求如下：

1．709机．

(1) 输入语句形式：#read(9, '10', A_2)，这里，A_2为一串用逗号分隔的简变、数组标识符．

(2) 同一输入语句的全部数据可依次穿成一条数带．一条数带可分成若干数段(但不能将一个数分拆在两个数段内，且每段不能超过6816排孔)．穿孔时，相邻两个数之间需穿一个全孔，数段结束时穿24个以上#和一个止．数带结束时穿一个#、24个以上全孔和一个止．数段之间留1米以上的空白纸带．例如，数据-1、2、-3.2、$0.1_{10}5$四个数穿成两个数段时的形式是：

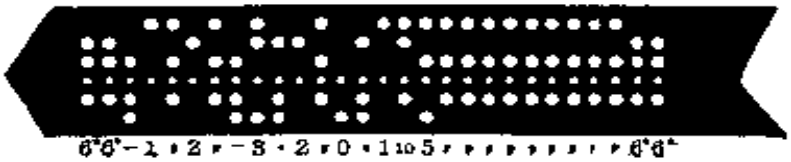


2．TQ-16机．

(1) 输入语句的形式：#read(0, 'N', A_2)，其中，A_2为一串用逗号分隔的简变、数组标识符．

(2) 程序的全部数据分为若干个数段(每段不能超过6100排孔)．每个数段由一个数组或同一输入语句中若干个

相连的简变组成. 穿孔时，数段中相邻两个数之间需穿一个逗号，结束时穿 9 个以上逗号. 同样，每个数段首尾分别加穿两个 6*. 例如，数据 −1、2、−3.2、$0.1_{10}5$ 四个数据穿成的数段，其形式为

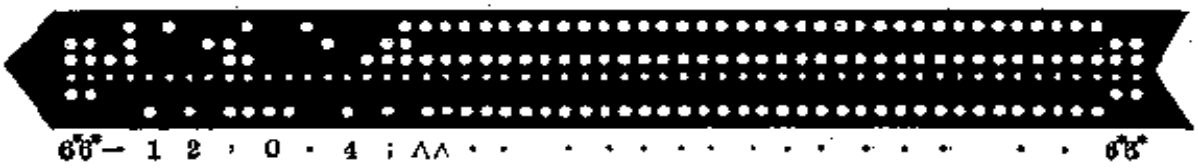


(五) 紧凑格式或称 Zn 格式(TQ-16 机)

紧凑格式是一种效率较高的数据输入格式. 使用方式介绍如下：

1. 数据按符号 6 编码穿孔. 同源程序的穿孔方式一样，例如，3.1416 穿孔成 031201040106，$2.6_{10}5$ 穿孔成 0212061305，等等.

2. 整个程序中的输入数据，不论是简单变量还是数组，不论分几个输入语句，都一个接一个穿在一起构成一条数带，数带中两个数之间穿一个逗号分隔，最后一个数后面穿一个分号，数带的首尾分别加穿起终信号 6*6* 和十五个以上 Λ 及 6*6*. 例如，程序的输入数据是 −12，0.4，则相应的数据纸带穿成：



3. 除此之外，Zn 格式尚可采用如下紧凑穿孔方式：

(1) 如果若干个相邻数相等，例如，数据中有如下一段：……10, 15, 15, 15, 15, 20, ……，则可简化为……, 10, 15 (4), 20……进行穿孔. 这里，圆括号前面的数是被重复的对

象，圆括号中的数指明重复的次数.

(2) 如果若干个相邻的数成等差数列，例如，数据中有如下一段：……，4，6，8，10，12，14，……，则可以简化为：……，4(6，2)，……进行穿孔，其中，圆括号前面的数是等差数列的首项. 圆括号中前面一个数是数列的项数，后面一个数是步长(它可以是任意实数)，两个数用逗号分隔.

(3) 如果若干组相邻的数，每组内容相等，例如，数据中有如下一段：……，8，4，2，1，8，4，2，1，8，4，2，1，……，则可以简化为：……，3[8，4，2，1]，……进行穿孔，其中，方括号前面的数表示循环的次数，方括号里是循环的内容.

(4) 紧凑和不紧凑的 Zn 穿孔形式，可以在同一条数据纸带中混合使用.

4. 使用 Zn 格式时，机器使用到 1 号鼓从 14335 向前的若干单元（等于输入数据的总个数），直到控打机印出-Su Wan-后，才完全释放这些单元，因此，若程序中使用写鼓语句，应注意不要和这些单元冲突.

第二节 上机计算

一 控制台简介

源程序和原始数据穿成纸带以后，接着的工作就是上机计算了. 上机时，算题人员与计算机直接联系的装置是控制台. 通过控制台，算题人员可以给出人工命令，了解程序的运行状况或采取必要的临时措施等. 下面分别介绍 709 机和 TQ-16 机的控制台.

(一)709 机控制台

709 机的控制台如图 32 所示. 控制台的面板上装有各

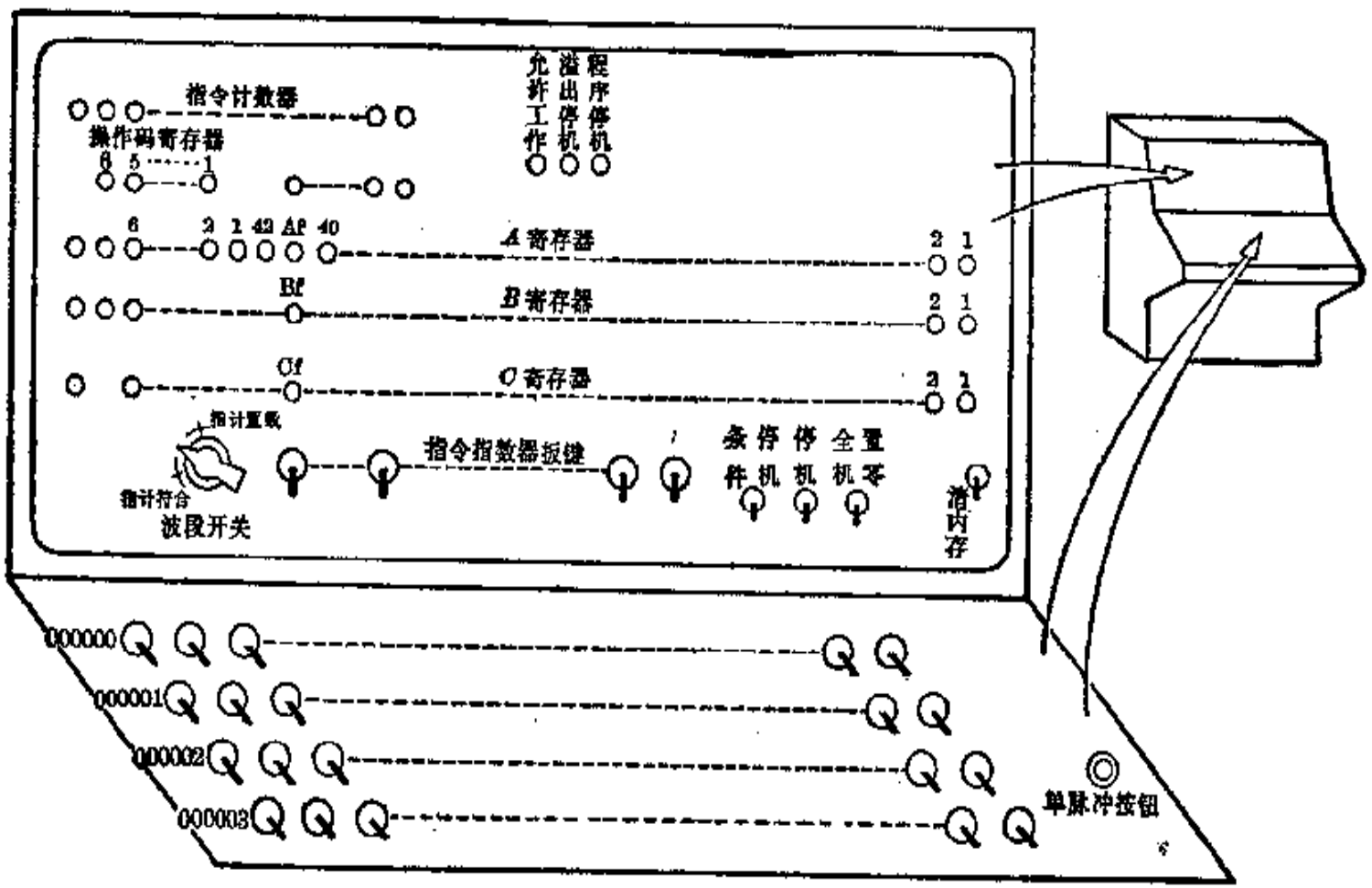


图 32　709 机控制台板面示意图

种开关、扳键、按钮和指示氖灯. 扳键和指示氖灯可以把数表示出来，规定：在指示氖灯表示数时，灯亮表示 1，灯暗表示 0；扳键表示数时，向上表示 1，向下表示 0. 这样，一排指示氖灯或一排扳键，按其状态的组合就可以表示一个浮点二进制数，为了书写方便，这个数常用八进制浮点形式写出.

下面对一些主要的指示氖灯、按钮、扳键及开关作一简单介绍.

1. 信号装置.

(1) 指令计数器. 它显示当前执行指令所在单元的地址. 第 0 位是 0 或 1，用来区分单元的左指令或右指令.

(2) 操作码寄存器. 它显示当前执行操作的性质.

(3) 运算器指示灯. 它由 A、B、C 三个寄存器组成. A 寄存器显示当前指令运算结果的内容. B 寄存器存放不规舍乘法的低位或不规舍除法的余数. 运算器从内存贮器读出或写入代码都要通过 C 寄存器.

(4) 停机指示灯.

溢出停机指示灯. 表示运算中因溢出而产生的停机.

程序停机指示灯. 它指示运算过程中正常的停机.

(5) 允许工作指示灯. 表示机器各部分均已处于可以开始工作的状态.

2. 操作设备.

(1) 单脉冲按钮. 是给出启动信息的装置. 若以短促的动作按一下单脉冲按钮，机器就从指令计数器中显示的单元地址开始执行运算(这个手续通常叫做启动).

(2) 全机置零扳键. 此扳键上下一个来回，就可使所有寄存器置成零，即处于启动前的准备状态.

(3) 波段开关. 是一个具有指向的旋转式开关. 它有两

个用途，即把单元地址置于指令计数器及使机器实现符合停机.

(4) 指令计数器扳键. 它的用途是:

(i) 把单元地址置于指令计数器. 手续是先把单元地址拨在指令计数器扳键上，然后，把波段开关转向"指计置数"，再恢复原来位置.

(ii) 给出符合停机地址. 手续是先把符合停机的地址扳在指令计数器扳键上，然后，把波段开关转向"指计符合". 这样，当机器执行到该地址的指令时，就实现停机.

(5) 清内存扳键. 将扳键上、下打一个来回，可以把内存贮器中的全部单元清洗为零.

(6) 停机扳键、条件停机扳键. 若把停机扳键打向上方，机器即实现强制停机. 条件停机扳键打向上方时，当机器执行到程序中的暂停语句时，即实现停机.

(7) 控制台单元. 在控制台面板上，有四排扳键，分别代表机器的 000000、000001、000002、000003 四个存贮单元，称为控制台单元. 每个控制台单元有 48 个扳键，相当于 48 个二进制数位. 控制台单元有两种用途:

(i) 给出各种操作的命令，是进行人工操作的主要装置.

(ii) 作为程序中变量使用，称为标准变量，四个控制台单元对应四个标准变量. 它们在程序中可以无须说明，直接使用. 如果程序中使用的标准变量标识符是 $G0000i$ ($i=0, 1, 2, 3$)，则称为八进标准变量；如果程序中使用的标准变量标识符是 $H0000i$ ($i=0, 1, 2, 3$)，则称为十进标准变量. 程序在使用标准变量时，限于从它们中取值，不准向它们赋值. 而且，八进标准变量对应控制台单元中给出的值，应是二进制浮点规格化形式；十进制标准变量对应控制台单元中给出的值，

应是二-十进制浮点规格化形式，且在使用前还需经过十翻二语句。如果十进标准变量需换新值，则应在对应控制台单元中置上新的二-十进制值，并重新经过十翻二语句。

标准变量使用举例如下：

若源程序中有 $n(\leqslant 48)$ 个程序块，每块计算一个项目，但在每次算题时，仅要求计算其中某几项，这时，就可在每一块程序的入口处置下列形式的语句：

L_i: i #Bo ($G00003$, i)>0 t g L_j;

其中，L_i 为第 i 块程序头部的标号，而 $i=1,2,\cdots,n$, $j=i+1$.

这样，若某次上机要计算第 3、7、11 项，则只要在控制台单元 000003 中置第 3、7、11 位为 1，置其他位为 0.

又如，程序段：

Fe $H00002$; i $H00002>10$ t $M:=M/10$; …

的执行效果是，当控制台单元 000002 中置上大于 10（二-十进制浮点规格化形式）的任一值时，机器就执行赋值语句 $M:=M/10$；否则不执行.

再如，

i $G00003<0$ t #W2 (0, '10', n, ep);

其中，n, ep 是程序中关键性的变量（如迭代次数，误差值等）。为了获取中间结果或考察程序运行是否正常，在程序的适当位置，常常安排这种语句，它的执行效果是，当控制台单元 000003 中任置一个小于零的值（即第 41 位扳键扳向上方）时，机器就输出关键变量 n, ep 的值，否则就不输出.

(二)TQ-16 机控制台

TQ-16 机的控制台分为使用控制台和维护控制台两部分。使用控制台供上机人员使用，维护控制台则供维修人员使用.

使用控制台又包括控制台面板和控制台打字机(电传打字机)两部分,现分别作一简单介绍.

1. 控制台面板.

TQ-16 机的控制台面板如图 33 所示, 面板上有各种信号装置和操作设备:

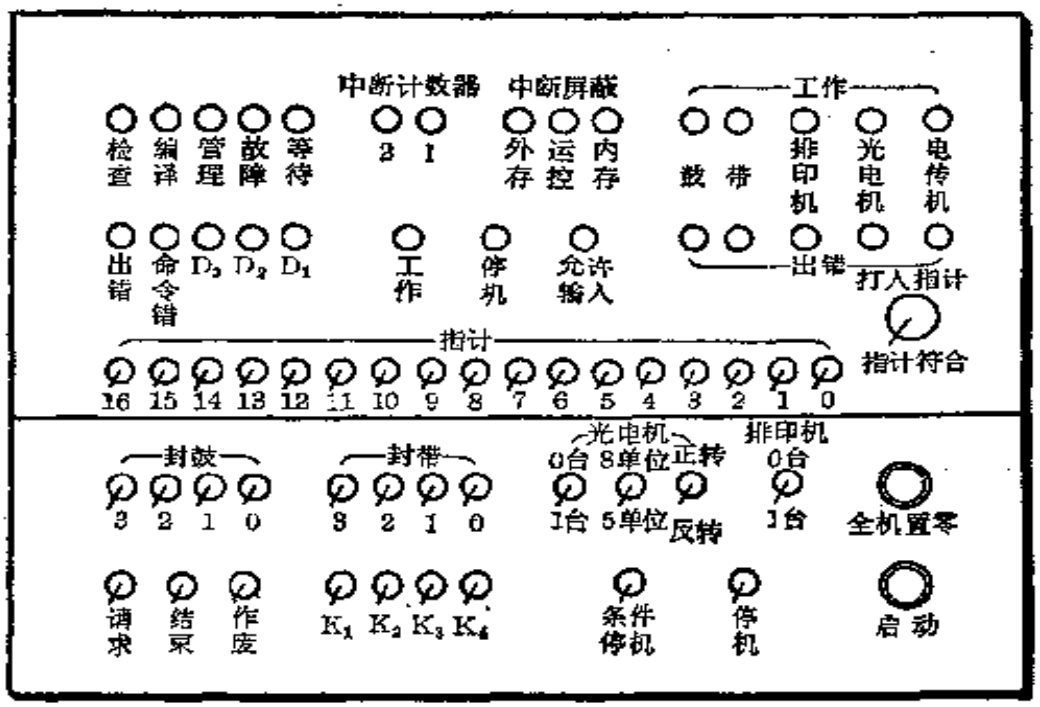


图 33 TQ-16 控制台面板示意图

(1) 信号装置.

(i) 机器状态灯:

停机灯: 灯亮表示目前正执行停机指令.

工作灯: 灯亮表示目前机器正在正常运行.

(ii) 软设备状态灯:

等待灯: 灯亮表示目前机器空闲,等待新任务.

管理灯: 灯亮表示目前机器处于管理程序工作状态, 灯暗表示目前机器处于目标程序工作状态.

故障灯：灯亮表示管理程序在运行过程中发生故障.

编译灯：灯亮表示机器目前处于编译程序工作状态.

出错灯：灯亮表示目标程序在运行中发生错误.

允许输入灯：灯亮表示控打机启动正常，允许输入命令.

命令错误灯：灯亮表示刚输入的命令有错误.

中断屏蔽灯：共三只灯，分别表示运控出错屏蔽，内存出错屏蔽，外部出错屏蔽.

中断计数器灯：共两只，分别表示机器当前的中断状态，它相当于两位二进制数，可以表示无中断、一级、二级、三级中断四种状态.

D_3、D_2、D_1 灯：相应于点灯语句中的三只灯.

(iii) 外部设备状态灯：鼓、带、排印机、光电机、电传机五种设备各有两只灯，上面灯亮表示该设备正在工作，下面灯亮表示该设备故障.

(iv) 检查灯，暂未定义.

(2) 操作设备.

(i) 起动和停机装置：

启动按钮：以短促的动作按一下，机器就从指令计数器中的单元地址开始执行运算.

全机置零按扭：以短促的动作按一下，可将全部寄存器置为零.

停机扳键：把扳键打向下方，机器即实现强制停机.

条件停机扳键：把扳键打向下方，则当机器执行暂停语句时实现停机.

指令地址扳键、打入指计及指计符合扳键：有时需要机器在预给的单元地址起、停，则使用这一套扳键；若要从预给的地址启动，可先把预给地址拨在指令地址扳键上(指令地址

扳键共17个，从右到左编号为0～16位，其中，0位扳键居中代表左指令，打下代表右指令，1～16位是地址码的二进制数位，扳键居中代表0，打下代表1），然后，把打入指计扳键置向上方一次，则指令地址扳键所代表的预给地址便被送入指令计数器，再按一下启动按钮，就实现从预给地址启动．若要机器在预给地址实现符合停机，可先把预给地址拨在指令地址扳键上，然后，将指计符合扳键打向下方，则当机器执行到预给地址时，就实现符合停机．

（ii）外部设备使用扳键：

光电机输入扳键：扳键打向下方为五单位输入方式，打向上方为八单位输入方式．

纸带正反转扳键：将此扳键打向下方一次，能使纸带从结束符号6*6* 开始倒转回起始符号6*6* 为止．

排印机选台扳键：扳键打向下方为1台，打向上方为0台．

光电机选台扳键：扳键打向下方为1台，打向上方为0台．

封鼓、封带扳键：共八只，分别对应0～3台鼓，0～3台带，扳键打向下方表示将该台鼓（带）封锁，即只能读出，不能写入．

（iii）命令输入扳键：包括请求、结束、作废扳键，它们的操作见本节之二．

（iv）另外，还有 K_1、K_2、K_3 扳键，若把它们打下，则改变相应标准变量的内容．K_4 暂无定义．

2．控制台打字机（控打机）．

TQ-16机用B-55型电传打字机作为控制台打字机，用以实现人机的联系，算题人员能够利用控制台打字机输入命令

或程序、输出命令执行的结果、各关节点的状态以及源程序等.

算题人员实现人机的联系，就是通过按动控打机上的键盘以发布命令的形式进行的，TQ-16 机控打机键盘如图 34 所示.

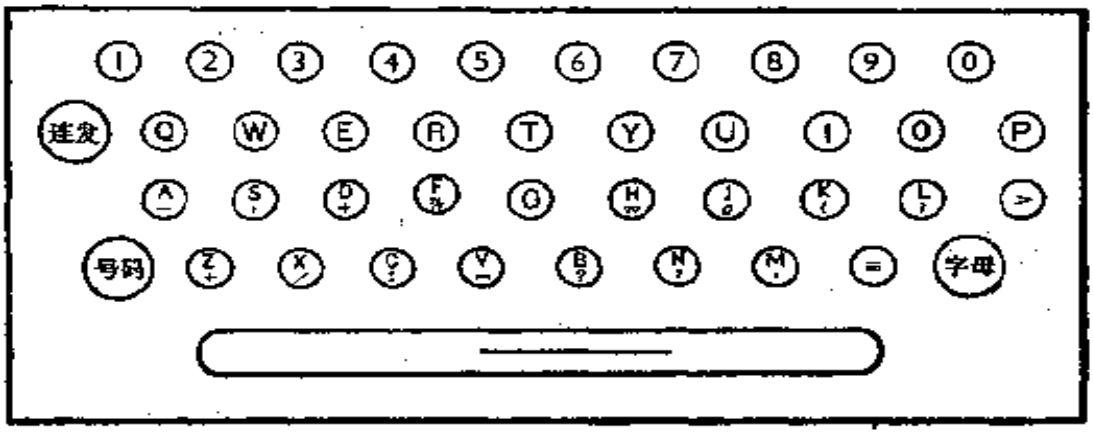


图 34　TQ-16 控制台打字机键盘示意图

键盘上共有五排键. 第一排键是数字 0～9，只有按过号码键后，才能按动此排键. 按此排键后，则印出相应的数. 按过字母键后，此排键是封锁的. 第二排键是字母 Q、W 等，只有按过字母键后才能按动此排键，按此排键后，则印出相应的字母. 按过号码键后，此排键是封锁的. 第三、四排键上都有两个符号. 如果按过号码键按动该两排上的键，则印出下面的符号；如果按过字母键按动该两排上的键，则印出上面的字母. 其中，＞是回车，＝是换行，它们对号码、字母均适用. 第五排只有一个长键，按此键后即印空白. 另外，再有一个连发键，按动此键就将上次按过键的内容连发，这两个键，对号码和字母均是公有的.

二　上机操作的基本技术

前面已经讲过，为了了解和控制程序运行的全过程，算题

人员需要随时对机器进行干预. 具体来说，这种干预在709机上是通过控制台面板上给出命令的方式实现的；在TQ-16机上则是通过控打机用发布命令的方式实现的. 这里，先介绍TQ-16机的命令形式及其输入方式. 709机的命令形式及输入方式将在下一节中结合具体操作技术给出.

(一)命令的一般形式

TQ-16机命令的一般形式是：

命令码　命令信息

其中，命令码由两个汉语拼音字母组成，如 *BX*，*BY* 等，它决定该命令的操作性质. 命令信息由字母、数字及分隔符组成(常用的分隔符有，·：—四种，其中，—不起语义作用，用以分隔信息的不同部分)，它决定该命令的操作内容. 有些命令只有命令码而无命令信息.

例如，命令 DS20, 002031. 表示要从内存地址为002031开始的一组单元中读出 $20_{(八)}$ 个数，按十进制形式在控打机上印出. 这里，DS是命令码，表示要执行的操作是从内存读出十进制数在控打机上印出. 数字是命令信息. 逗号，和点号·都是分隔符，逗号决定了它前面的数字是读出数的个数(八进制)，点号决定了它前面的数字是读出数的起始地址.

(二)命令的输入方法

算题人员根据命令的形式，按动控打机的键盘就可以把命令输进机器. 具体操作时，还将涉及四只灯(允许输入、命令错、电传机)和三个扳键(请求、结束、作废). 输入命令的操作顺序是：

首先，将请求扳键打向下方一次. 当允许输入灯亮时，就在控打机上打入命令，并在命令之末连打八个以上—. 然后，核对打印纸上的命令是否正确，若正确无误，则进行下一步操

作；若核对有错，则应将作废扳键打向下方一次，这时，控打机即印出—MLZF=477303，表示命令已被作废，然后，重新请求、打入命令，直到正确为止.

接着，将结束扳键打向下方一次，这时机器开始核对该命令是否符合命令之一般形式，若无错误，自动执行该命令，并由控打机印出 MLZX（命令执行）字样，至此，就完成了一道命令的输入和执行；若核对有错，机器将印出——MLZF（命令作废），并自动将命令作废，然后，需重新请求、打入命令.

三　上机操作的基本步骤

从输入程序到正式运算，人工操作的基本步骤如下：

（一）输入源程序

首先，把源程序纸带置于任一台光电输入机上，然后进行如下操作：

709 机	TQ-16 机
1．置控制台单元： "0" 34 007000 35 000000 "1" 36 000040 37 000010 "2" 00 000063 20 000010 其中，"2" 中的地址码 000063 和 000001 分别对应于第 30、29、26、25、1 位二进制扳键，算题人员可根据需要选拨它们，各二进位扳键含义是： 第 1 位：表示不要语法检查. 第 25 位：表示要输出参考信息表、内存分配表. 第 26 位：表示要输出源程序（统篇式）.	1．拨好光电机选台扳键和光电机输入方式扳键. 2．顺次按"全机置零"和"启动"按扭，此后，"等待"灯亮，控打机印出：—DENG DAI GONG ZUO—（等待工作） 3．发布准备编译命令： By FGYSCX 其中，By 为命令码，其余为命令信息，算题人员可根据需要选打它们（次序不论），各字符的含义是： F 表示修改方案和源程序按、单位方式穿孔.

（续表）

709 机	TQ-16 机
第29位：表示要输出信息表。 第30位：表示要输出源程序（页行式）。 2．将“全机置零”扳键打一个来回，再给出启动信息，机器就自动将源程序纸带输入内存贮器，然后从“标准程序库”中读入相应的过程说明嵌入源程序中相应位置（若源程序包含库过程），形成一个新的源程序，最后输出该源程序（若“2”中第26位或第30位扳键拨上）。	G表示有修改方案。 Y表示不要语法检查。 S表示要输出源程序（页行式）。 C表示要输出参考信息表、内存分配表。 X表示要输出信息表。 接着，机器就自动输入修改方案（若打出G的话），并打印出来。其次，自动输入源程序，并对源程序进行修改（若源程序中使用了库过程，则便从“标准程序库”中读入相应的过程说明，嵌入相应位置，形成一个新的源程序），最后输出该源程序（若打出S的话）。

（二）语法检查

源程序输入之后，接着，机器就对它进行语法检查（若给出需要检查的信息）。

若整个源程序没有语法错误，则机器实现停机（709机的停机地址是000722右；TQ-16机的条件停机地址是000325）。

若源程序有语法错误，则机器自动在排印机上印出错误信息，其形式是

709 机	Cuo Wu 09 08 L_1 …… L_1: eeoreI: =1 estepe 1 euntí leneDeA[i]: = …… …… （三行字符）
TQ-16 机	Yu FA Cuo Wu #YE003: #HA015: 09 08 L_1 ……; L_1: eeoreI: =1 estepe 1 euntilenedoeA[i]: = …… …… （三行字符）

这个语法错误信息表示：在印出字符行的第二行第 08 个字符附近存在第 09 类性质的语法错误 (Yu FA Cuo Wu)．错误发生处前面最邻近的标号是 L_1．根据这个语法错误信息，查阅“源程序语法错误表”(见附录二(一))，易知，这是由于把 *e*For*e* 误写(穿)成 *e*Eor*e* 所致．

机器输出语法错误信息后，一般是跳过这段程序(并印出跳跃信息：TIAO DAO)，继续对后面的程序进行检查，若再有错，则继续输出错误信息，直至查完整个源程序．

语法检查结束之前，对源程序中未经说明即被使用的标识符或标号也输出一行信息，形式是：

709 机	Cuo Wu 39 00 L_2 或 Cuo Wu 07 00 AA_2 L_3 (错误) (编号) (标号) (错误) (出错编号) (标识符) (标号)
IQ-16机	YuFACuoWu 3900 L_2 或 YuFACuoWu 07 00 AA_2 L_3 (语法错误) (编号)(标号) (语法错误)(出错编号)(标识符)(标号) QING TAO SHEN Du: 003; XING ZHI 004; (嵌套深度) (层数) (性质) (编号)

前者表明：使用了未经说明的标号 L_2(列在开关表中的标号算作对它的使用)．后者表明：嵌套于源程序第 3 分程序中使用了未经说明的数组标识符 AA_2．L_3 是它前面最邻近的标号．这里，标识符性质编号规定为：001 表示简单变量，002 表示布尔变量，004 表示数组变量，010 表示过程标识符，040 表示开关变量．

对于有语法错误的源程序，必须按照上节提供的修改源程序方法把错误改正，再次上机.

另外，语法检查过程中，机器还自动把程序的注解打印出来，例如，对程序段：……sThensseGOTOsL_2sendsJacobi;……，机器印出如下形式的注解：

ZHU JIE Jacobi

其中，ZHU JIE 是注解的汉语拼音.

(三) 编排

在语法检查通过的基础上，继续启动(若条件停机扳键未打下时，TQ-16 机将由上一步自动转入)，则机器就对源程序进行编排，即把源程序编排为中间语言程序——L_1 程序.

编排过程中，若继续发现语法错误，机器便自动停机(对 TQ-16 机尚在控打机上印出停机地址 ZJ＝073567)，启动后即印出错误信息，并继续进行编排，出错信息的形式是：

709 机	00……0××00……0×× …… ……}(二行字符) 其中，前一个八进制代码表示出错分程序的嵌套深度，后一个八进制代码表示出错性质编号．错误发生在第二行字符中．
TQ-16机	Yu FA Cuo Wu (语法错误)　#YE××××: (页)　#HA××××: (行) QING TAO SHEN Du:××× (嵌套深度)　BIAN HAO:××× (编号) …… ……}(二行字符)

同前面一样，根据这个语法错误信息，查阅“源程序语法错误表”(附录二(二))，即可确定语法错误的性质.

如果在源程序中，使用了未经说明的标识符，机器将继续

断定其为语法错误，此时，机器在控打机(TQ-16 机)或排印机(709 机)上印出如下形式的信息：

×××××××	×××××	×××	×××××××	×××××	×××
〈标识符〉	〈嵌套深度〉	〈性质〉	〈标识符〉	〈嵌套深度〉	〈性质〉

编排结束后，机器印出如下形式的参考信息表及内存分配表(若给出需要印出的信息)。

CAN KAO XIN XI　　(参考信息)

×××××××	×××××××	×××××××
〈源程序长度〉	〈L_2 中间程序长度〉	〈局部量存区长度〉

NEI CUN FEN PEI BIAO　　(内存分配表)

〈N〉 BIAO	×××××××	〈数表头地址〉
〈D〉 BIAO	×××××××	〈信息向量表头地址〉
〈S〉 BIAO	×××××××	〈开关表头地址〉
〈K〉 BIAO	×××××××	〈过程标识符表头地址〉
〈P〉 TOU	×××××××	〈目标程序头地址〉
〈V〉 TOU	×××××××	〈局部量头地址〉

××	×××××××	×××××××	××	×××××××	×××××××
〈局部量标识符〉	〈内存地址〉	〈标识符性质〉	〈局部量标识符〉	〈内存地址〉	〈标识符性质〉
⋮	⋮	⋮	⋮	⋮	⋮

(四) 翻译

编排结束以后，机器自动对 L_1 程序进行翻译工作。即把 L_1 中间语言程序翻译成机器语言程序——目标程序。

翻译过程中，机器若继续发现语法错误(一般是程序太长或变量太多等原因造成)，便自动停机(TQ-16 机在控打机上印出停机地址 ZJ＝×××××××，709 机则在指令计数器中显示停机地址)，根据停机地址，查阅"编译过程中非正常停机表"(附录三(一))就可确定出错误的性质。

翻译结束后，机器印出如下形式的信息表(若给出需要印

出的信息)：

XIN XI BIAO (信息表)

KAI SHI(开始)	×××××××	‹目标程序开始地址›
JIE SHU(结束)	×××××××	‹目标程序结束地址›
#YE×××: #HA×××:	×××××××	‹对应目标程序地址›
BIAO HAO(标号)	×××××××‹对应地址›××××	‹标号标识符›
TING JI(停机)	×××××××‹停机地址›×××××××××	‹停机指令›
ZAN TING(暂停)	×××××××‹暂停地址›×××××××××	‹暂停指令›
SHU RU(输入)	×××××××	‹输入指令地址›
SHU CHU(输出)	×××××××	‹输出指令地址›
DOU GU(读鼓)	×××××××	‹读鼓指令地址›
XIE GU(写鼓)	×××××××	‹写鼓指令地址›
DOU DAI(读带)	×××××××	‹读带指令地址›
XIE DAI(写带)	×××××××	‹写带指令地址›

……

此后，机器自动从外存读入标准子程序等，在内存形成目标程序的运行状态，并把它记入外存作为副本. 然后，709 机停机于 000022 右；TQ-16 机则在控打机上印出信息后停机.

-BIAN YE JIE SHU-(编译结束)

(五)执行目标程序

经上述步骤，机器就完成了把源程序编译成目标程序的工作(这个过程一般只需几十秒钟到二、三分钟时间). 然后，作好如下准备：

1. 若有原始数据，则应把数据纸带置于光电机上(TQ-16 机尚需拨好光电机选台扳键和输入方式扳键).

2. 若使用标准变量，则应给予赋值，TQ-16 机用写标准变量命令来实现；709 机则在相应控制台单元中给值.

3．若使用暂停语句，则应拨好条件停机扳键．

然后，给出启动信息(709 机按动单脉冲按钮；TQ-16 机根据程序中原始数据的输入格式用控打机发布启动命令：程序使用十、八格式和符号 6 格式时命令是 BS；使用符号 4 格式时命令是 FS；使用紧凑格式时命令是 Zn，或从 076520 打入指计后启动)，目标程序就开始正式运行．其后，机器根据程序中安排的输入、输出和停机语句，自动输入原始数据、输出中间、最终结果或实现停机．但有时也可能出现一些意外停机，这时应对照"非正常停机表"(附录三之(二)、(三))，查实原因，采取措施．

程序正常运算结束，709 机停机于目标程序结束地址；TQ-16 机则在控打机上印出信息-YAN SUAN JIE SHU-(运算结束)．

四　几种常用的操作技术

上面我们介绍了正常情况下从输入源程序到执行目标的操作步骤．除此之外，算题人员还可根据需要执行某些人工操作，现介绍如下．

(一)程序纸带的分段输入

对于较长的源程序纸带，可以按如下办法分段输入．

1．把源程序纸带任意分段穿孔为

709　机	TQ-16　机
$a_1b_1\cdots z_1$␣␣…␣ 止白 $a_2b_2\cdots$ z_2␣␣…␣ 止白 ……$a_nb_n\cdots z_n\wedge\wedge\cdots\wedge$ 止	6*6*$a_1b_1\cdots z_1$␣␣…␣6*6* 白 $a_2b_2\cdots$ z_2␣␣…␣6*6* 白 ……$a_nb_n\cdots z_n\wedge\wedge\cdots\wedge$6*6*

其中，$a_ib_i\cdots z_i$ 为第 i 段源程序字符，␣和∧的穿孔个数均应超过 15 个.

2．按如下操作进行分段输入：

709 机	TQ-16 机
按输入源程序的一般方法输入第 1 段，停机后，启动，输入第 2 段，……，直至输入第 n 段结束.	用控打机发布命令 BX FGYSCX 其中，命令信息中各字符含义同前.

若输入过程中，在第 m 段发生中断，则只要把纸带倒转到该段之首，然后从 000020(709 机)或 075120(TQ-16 机)启动重新输入.

(二)重算

如果由于某些原因，内存状态被破坏(其他部分完好)，这时可将原始数据纸带回转到首部，然后作如下操作：

709 机	TQ-16 机
置控制台单元： "0" 34 005000 35 000010 "1" 36 000040 37 000010 "2" 20 000010	用控打机发布命令： BZ

全机置零，启动.

这样，内存中重新获得目标程序的运行状态，启动后即进行重算.

(三)给出启动地址

程序运行中，若需从预给地址 K(或 $K_{右}$)继续计算，则应将机器停下，然后进行操作：

709 机	TQ-16 机
方法一：置控制台单元： "0" $\overset{20}{21}$ K 00 000000 全机置零，启动$\left(\begin{smallmatrix}20\\21\end{smallmatrix}\right.$为$\left.\begin{smallmatrix}K\\K_{右}\end{smallmatrix}\right)$. 方法二：将 $K(K_{右})$ 置于指令计数器扳键上，全机置零，波段开关转向指计置数后恢复原位，启动.	方法一：用控打发布命令： $QD\ K$. 若为 $K_{右}$，则命令信息 K 的第 1 位打 4，例如命令：QD402716，给出启动地址 $002716_{右}$. 方法二：将 $K(K_{右})$ 拨在指令地址扳键上，把打入指计扳键向上打一次，然后启动.

（四）复算

若程序需要对 n 组不同原始数据计算 n 次，则每次计算时可先将数据纸带准备好，然后作操作：

709 机	TQ-16 机
方法一：用（三）的给出启动地址 $K(K_{右})$ 的方法（这里 $K(K_{右})$ 是由信息表中查出的目标程序开始地址）. 方法二：用（二）的重算方法（若外存完好的话）.	方法一：根据程序输入数据的格式，分别用控打机发布命令： BS 或 FS 或 Zn， 方法二：用（三）的给出启动地址的方法或用（二）的重算的方法.

（五）读变量的值

在程序运行过程中，有时需要把某些局部量的值：$\langle d\rangle \xrightarrow{n_{(八)}} \langle d\rangle+n-1$ 印出来，其操作是：

709 机	TQ-16 机
方法一：置控制台单元： "0" 06 077701 20 000240 "1" 00 $\langle n\rangle_{(八)}$ 00 $\langle d\rangle$ "2" 01 全机置零，启动. 方法二：置控制台单元： "3" 00 $\langle n\rangle_{(八)}$ 00 $\langle d\rangle$ 然后，按（三）给出启动地址002151.再启动.	方法一：用控打机发布命令： DS n, d. 当 $n=1$ 时，命令可简打为： DS d. 则控打机按要求打印出相应的 $n_{(八)}$ 个数，此方法只适用于少量的数. 方法二：用控打机发布命令： DYG130: n, d. 则排印机将地址从 d 开始的 $n_{(八)}$ 个数带地址打印出来.

(六)写变量的值

在程序运行的过程中，有时需要把给定值 S 写入局部变量 d，其操作是：

709 机	TQ-16 机
置控制台单元： "0" 71 000002 04 〈d〉 "1" 01 "2" 〈 $S_{(A)}$ 〉 全机置零，启动。	用控打机发布命令： XS d: S. 这里，S 是欲写入的十进制浮点数形式。

(七)给标准变量赋值

709 机	TQ-16 机
直接把变量值置于标准变量所对应的控制台单元中，若程序使用八进制标准变量，则相应控制台单元应置浮点八进制规格化数；若程序使用十进制标准变量，则相应控制台单元应置二—十进制浮点规格化数，程序中应在使用它之前先经过十翻二语句：Fe $H0000i$; (i 可取值 0、1、2、3)。	在控打机上发布命令： XK Gi, S。Hi, S。 其中，i 可取值 0、1、2、3，表示标准变量的序号。G 表示八进制标准变量，命令把其后 S 值的八进制浮点数，直接赋予相应标准变量；H 表示十进制标准变量，命令把其后 S 值的十进制浮点数，规格化后赋予相应标准变量。 每次写入的数可以 1～4 个。(八进或十进标准变量) 例如，对四个标准变量 0，1，2，3 依次赋值 126，－34，50，－25，其中前两个用十进制浮点格式，后两个用八进制浮点格式，则命令形式为： XK $H0$, 03001260000000。 1, 020134000000000。 $G2$, 060062000000000000。 3, 050162000000000000。

(八)用控打机输入修改方案(TQ-16 机)

源程序第一次上机后，若只发现少量易于查到的语法错误，则可直接通过控打机发布修改命令，达到修改源程序的目的．具体操作步骤是：

1．将源程序纸带倒转到其首部，然后，全机置零，启动，使机器处于等待工作状态．

2．发布电传修改命令，形式是

FR 修改方案"FH"

其中，命令信息部分按前述源程序修改方案的编写形式一样要求．

3．发布编译命令：

By FGYSCX

从而，机器就能按电传输入的修改方案，对源程序进行修改，并继续以后的工作．

例如，要对源程序进行两个修改：把第 2 页第 21 行第 4 个字符 j 改为 $-j$；删除第 2 页第 23 行第 1 个分号到第 2 个分号之间的一个语句，则修改命令为：

FR #XG 002, 021, 004; $-j$ #GB; #SC (002, 023, 001:002, 023, 002); "FH"

附录一　源程序书写限制表

由于机器的内存容量以及"编译系统"内部的有关表格都是有限的，所以，在编写源程序时必须注意如下限制条件：

限制种类	限制值	备注
源程序长度	$114624_{(+)}$	字符总数
L_1 中间程序长度	$54000_{(入)}$	
目标程序最大长度	$60000_{(入)}$	当局部量较多时，可能小于该长度
数组段内数组个数	128	
固定数组维数	30	
半动态数组维数	10	
开关表总长	700	n 个标号的开关表占 $n+1$ 个单元
标号总数	1000	
源程序中常数总个数	830	
源程序中常数的限值	10^{19}	
过程说明总个数	255	
一个过程说明中形式参数个数	63	
八、十输入、出，读、写数语句中变量标识符个数	无限制	
A、I、F、E 格式输出语句中变量标识符个数	无限制	
A、I、F、E 格式输出语句中使用格式指示的个数	无限制	
输出语句中每一格式指示所占用总字符数	56	
多重赋值语句的重数	60	
循环表中 E_i 和 dE 的字符个数	128	
分程序嵌套层数	46	
过程体内分程序层数	不限	

附录二　源程序语法错误表

(一)语法检查过程中的出错编号和出错性质对照表

编号	出　错　性　质	编号	出　错　性　质
01	源程序未以 **begin** 开始	14	一般过程的标识符被赋值
02	**begin** 个数多于 **end** 个数	15	**real** 型、**integer** 型变量被赋予逻辑值
03	黑体词限制符出错	16	算术表达式开始符号出错
04	非黑体词分隔符号出错	17	运算分量的类型不一致
05	(与 [使用混淆	18	一般过程被用作运算分量
06	括号不成对出现	19	函数命名符不是用作表达式最左面的运算分量
07	该标识符未被说明	20	数组标识符被当作简单变量
08	该标识符被说明两次或标号标识符重复出现	21	数组的下标数和维数不相等
09	语句开始符号出错	22	下标变量的下标出错
10	语句或说明后的符号不对	23	布尔表达式出错
11	循环语句出错	24	非布尔量被作为布尔量使用
12	**then** 后面出现 **if**	25	第二种形式的转向语句出错
13	赋值语句中赋值号左部变量出错	26	标准函数、输入、出语句，读、写数带语句的符号后不是 (

（续表）

编号	出错性质	编号	出错性质
27	函数过程标识符未被赋值	42	输入或输出语句中变量标识符部分出错
28	简单变量说明中出错	43	输入或输出语句中的其他错误
29	数组说明中出错	44	标号说明表存区满
30	开关说明中出错	45	标号使用表存区满
31	过程说明中出错	46	标识符表存区满
32	过程语句的实在参数部分出错	47	数组段长度或界偶数超过限制数
33	形式参数未被区分	48	多重赋值语句中两端变量类型不一致
34	数组标识符区分为值参数	49	多重赋值语句重数超过限制数
35	该符号不能在实在参数部分出现	50	输出语句中输出格式和变量标识符不对应
36	过程说明中值部分或区分部分出错	51	标识符暂存表区满
37	形式参数与实在参数之类型或个数不一致	52	输出语句中B格式使用不正确
38	同一层中出现相同标识符的标号	53	'十翻二'语句出错
39	转向语句在本层及外层找不到转向标号	54	Z格式出错
40	输入、输出语句中格式部分出错	55	库过程说明出错
41	输入语句出错	56	二元标准函数出错(如#bo(E, F)等)

(二)编排过程中的出错编号和出错性质对照表

编号	出错性质	编号	出错性质
46	黑体词限制符出错	65	下推链存区满
47	数组说明中上界小于下界	66	鼓中的标号对应表满
50	形式参数被区分为开关	67	开关表存区满
51	同一分程序中标识符被说明二次	70	数组段表满
52	圆括号不配对	71	信息向量暂存表满
53	方括号不配对	72	信息向量表满
54	标准函数或输入、出语句的符号写错，页、行出错	73	数表存区满
55	缺少 end	74	内存分配表满
56	缺少 begin	75	辅栈存区满
57	源程序中标识符未说明	76	主栈存区满
60	源程序常数绝对值超过 $_{10}19$	77	L_1 程序(中间程序)满
61	半动态数组中上、下界变量未说明	00	界偶个数超过允许值

附录三　非正常停机表

(一)编译过程中非正常停机表

程序	停机地址	原　　因	处　理　意　见
修改	075133	<调整修改符>出错	下机
	075250 右	调整修改后，程序长度超过7680个机器字	下机
语法检查	000457 右	<标号说明表>满	下机
	000505 右	<标号使用表>满	下机
	001230	<标识符表>满	下机
	002247	<标识符暂存表>满	下机
编排	073567	语法错误	继续启动，打印出错信息后，进行后继工作
	073602	语法错误	下机
翻译	000433	程序主体部分未由 begin 引入	下机
	000424	<过程说明>个数超过允许值	下机
	000543 右	目标程序长度超过允许值	下机
	000550	状态表满	下机
	000552	辅助下推表满	下机
	000554 右	目标程序临时下推表满	下机
	000556 右	U[4]信息表满	下机
	000634	目标程序、局部量存区满	下机
	000641	标号表超过允许值	下机
	001354	状态表中有非法元素	继续启动，打印出错信息后，停机于001734
	001402	循环语句专用表(T)满	下机
	001516 右	循环语句专用表(IT)满	下机
	001524 右	循环语句专用表(T11)满	下机
	001730	算术运算符出错	继续启动，打印出错信息后，停机于001734
	001731	关系运算符出错	同上
	001732	逻辑运算符出错	同上
	001733	单边运算符出错	同上
	001734	翻译下标变量时出错	同上
	002053 右	开关表长度超过容许值	下机
	002071 右	改造开关表时出错	下机

（二）目标程序非正常停机表

程　序	停机地址	原　　因	处　理　意　见
过程控制	000146 右	目标程序值参数出错	下机
10→2	000226	〈10→2〉中遇大于 9 的数字	将该十进制数的准确值置于 000004 单元后，从本地址之右指令继续启动
半动态数	001045 右	〈界偶表〉求得之分量总数大于〈体积变量〉之值	下机
数组控制	001046	半动态数组上界值小于下界值	下机
格式加工	001540 右	F 格式中，所给 W 不足，即 $W<d+7$	继续启动，$d+7\Rightarrow W$，调用 E 格式
	001563 右	输出数 $\|x\|\geqslant 2^{40}$，不能使用 I 格式	继续启动，仍调用 I 格式
	001573 右	I 格式中，所给 W 小于该数值之实际有效的位数	继续启动，I 格式按实际有效数位输出
	002064	输出格式中使用了非法的格式指令	下机
	002110 右	L 格式中使用了非法的子格式	下机
	002254	N 格式之某一数值大于 $_{10}\pm 19$	继续启动，进行后继工作

（三）标准函数非正常停机表

程　序	停　机　地　址		原　因	处理意见
	709 机	TQ-16 机		
Arcsin(E)	077200	075170	$\|E\|>1$	下机
sqrt(E)	077313 右	075303 右	$E<0$	下机
Axp(E, F)	077331 右	075321 右	$E<9$	下机
ln(E)	077355 右	075345 右	$E\leqslant 0$	下机
exp(E)	077427	075417	$E\geqslant 63\ln 2$	下机
sin(E) 或 cos(E) 或 tg(E)	077511	075501	$\|E\|\geqslant\frac{\pi}{4}\cdot 2^{40}$	下机
Bo(E, F)	077631	075621	$\|F\|>48$	下机
A(E, F)	077655 右	075645 右	$\|F\|>48$	下机
v(E, F) 或 $f(E, F)$	077677 右	075667 右	$\|F\|>48$	下机

习 题 解 答

习 题 一

1. (1) $+2^{+15}\cdot 0.40000000000000$; (2) $+2^{-3}\cdot 0.63146314631461$;
   (3) $+2^{+9}\cdot 0.62207732321131$; (4) $-2^{+8}\cdot 0.56172702436561$;
   (5) $-2^{+10}\cdot 0.75251564570651$; (6) $+2^{-10}\cdot 0.62624536474160$;
   (7) $-2^{-10}\cdot 0.72517156726700$; (8) $+2^{+12}\cdot 0.764000000000$.
2. (1) $+10^{+02}\cdot 0.785430000$; (2) $+10^{+04}\cdot 0.300000100$;
   (3) $+10^{-17}\cdot 0.100000000$; (4) $-10^{+10}\cdot 0.538000000$;
   (5) $-10^{-08}\cdot 0.153000000$; (6) $+10^{+02}\cdot 0.787878000$;
   (7) $+10^{+08}\cdot 0.400000000$; (8) $-10^{+08}\cdot 0.205000000$.

习 题 二

1. 3a(否), 5mx(否), m_{1a}(是), tb(3)(否), cy−1(否), y_4(是), sin(是), b[2](否), #sqrt(否), φ(否), xa(否), beta(是).
2. 10^2(应写成 $_{10}2$), $5.7_{10}-8$(允许),
   $3_{10}3.6$, $-1.6_{10}0.9$($_{10}$ 后要求整数值).
3. #cos(#sqrt(a↑2+b↑2+c↑2)),
   #exp(#cos(b)×#ln(#sqrt(a[i, j]))),
   x[i]↑3−#sin(a↑5−b↑5)/(7.2_{10}−8−c),
   (−b+#sqrt(b×b−4×a×c))/2/a,
   (−b−#sqrt(b×b−4×a×c))/2/a,
   #sqrt((#ln(#sqrt(x×x+y×y))−#exp(#cos(s)↑2))↑5).
4. (1) $\sin\alpha\cdot\sin^2\beta+\sqrt{a^2+\mathrm{tg}^{-1}x_n}$;
   (2) $e^a\cdot x_i^b$.
5. (1) true; (2) true.
6. (1) (−b+#sqrt(b×b−4×a×c))/(2×a);

(2) a[i]+#arcsin(x)+#sin(y)/#cos(y)−u×#ln(b)/#ln(10);

(3) ((x+y)×#exp(z))↑2+1;

(4) (a×b−#cos(f)↑(−5))/(k−h).

习 题 三

1. (1) x:=a×b↑3+c×#ln(1+c×c);

(2) x:=(−b+#sqrt(b×b−4×a×c))/2/a;

(3) y:=z:=a×x↑4+#abs(x×x×#sin(x)/#cos(x)−a);

2. (1) **for** x:=100 **step** 1 **until** 150 **do**
y[x−99]:=a×x×x×#cos(x×3.1416/200)−#sqrt(x);

(2) **for** x:=0 **step** 2 **until** 50 **do**
y[x/2+1]:=#ln(1+#sqrt(x))/(1+x×x);

(3) #read(0, '10', a); D:=0;
for i:=1 **step** 1 **until** 105 **do**
D:=D+a[i]↑3;

(4) **for** a:=2 **step** 1 **until** 6, 10, 24 **do**
begin x:=(a+#sqrt(a×a−1))/(a+1);
#print (0, '2Ha=, i2, 15x, 2Hx=, F14.6', a, x) **end**;

(5) #read (0, '10', a, x);
for i:=0, 1, 2, 3 **do**
begin W:=1;
for j:=1 **step** 1 **until** 100 **do**
W:=W×(x[j]×x[j]−a[i]);
#print (0, '2Hi=, i1, 5x, 2HW=, F14.6', i, W)
end;

3. **if** x≤0 **then** y:=−x **else if** x≤2 **then**
y:=x×x **else** y:=4
if #abs(x)≥a **then** y:=0 **else** y:=a−#abs(x)

4. #read(0, '10', a);

```
        s:=t:=0;
        for i:=0 step 1 until 50 do
        if a[i]≥0 then s:=s+a[i] else t:=t+a[i];
5.  (1) #read (0, '10', a, x);
          D:=0;
        for i:=1 step 1 until 100 do
          D:=D+(x[i]-a)↑2;
        D:=#sqrt (D/100);
    (2)   #read (0, '10', y);
          D:=0;
          for i:=0 step 1 until 99 do
            D:=D+y[i]+y[i+1];
          D:=D/2;
6.        t:=0;
      L:x:=t-#sin(t); y:=1-#cos(t);
        #print (0, '2F14.6', x, y);
        if x≥0∧x<1 then begin t:=t+0.1;
            goto L end;
7.      #read(0, '10', p, q);
        for i:=1 step 1 until 100 do
          begin x1:=p[i]×p[i]-4×q[i];
            if x1≥0 then begin
                x1:=(-p[i]+#sqrt(x1))/2;
                x2:=(-p[i]-#sqrt(x1))/2;
                #print(0, '3, 5x, 2F14.6', i, x1, x2)end
          end;
8.        #read (0, '10', x, y);
          ymax:=y[1]; x1:=x[1];
        for k:=2 step 1 until n do
          if #abs(y[k])≥#abs(ymax)then begin
```

```
        y_max:=y[k]; x1:=x[k] end;
    #print(0, '2F14.6', x1, y_max);
```

9. (1)
```
    s:=y:=1;
    for n:=1 step 1 until 9 do
      begin y:=y×x/n; s:=s+y end;
    #print (0, 'F14.6', s);
```
(2)
```
    n:=s:=y:=1;
    L: y:=y×x/n; if#abs(y)>10-5 then begin
        s:=s+y; n:=n+1; goto L end;
      #print(0, 'F14.6', s);
```

10. 13, 21, 21,

11. (1) 逻辑表达式 $0<x<2$ 应写成 $0<x \wedge x<2$;
(2) 条件算术表达式中缺少 **else** 后面的算术表达式;
(3) 条件语句中 **then** 后面不能是条件语句;
(4) 不允许用转向语句转入循环语句的循环体;
(5) 输出格式和变量标识符不一一对应;
(6) 8、10 输出格式不能同其他输出格式在同一个输出语句中使用;
(7) 输出变量的标识符中不能有下标的形式。

12.
```
    x1:=x[1]; x2:=x[2]; x3:=x[3];
    y1:=y[1]; y2:=y[2]; y3:=y[3];
    #print(0, '3Hx1=, F14.6, 3Hx2=, F14.6, 3Hx3=,
      F14.6, 3/, 3Hy1=, F14.6, 3Hy2=, F14.6, 3Hy3=,
      F14.6', x1, x2, x3, y1, y2, y3);
```

习 题 四

1.
```
    array a[1:10], d, e[1:3, 1:10], f[1:3, 1:2, 1:5];
```

2. (1)
```
      begin
        array a[1:100];
        integer i, n, m;
```

```
        #read(0, '10', a); n:=m:=0;
        for i:=1 step 1 until 100 do
          if a[i]÷2=a[i]/2 then m:=m+a[i]
              else n:=n+a[i];
        #print(0, '2F14.6', m, n)
      end 55…5
```

(2)

```
      begin
        integer m, l, n;
        #read(0, '10', m, l, n);
        begin
          array a(100)[1:m, 1:n], b(100)[1:n, 1:l],
                c(100)[1:m, 1:l];
          integer i, j, k;
            #read (0, '10', a, b);
            for i:=1 step 1 until m do
              for k:=1 step 1 until l do
              begin c[i, k]:=0;
                for j:=1 step 1 until n do
                  c[i, k]:=a[i, j]×b[j, k]+c[i, k]
              end;
          #print (0, 'F14.6', c)
        end
    end 55…5
```

3. (1) 16, 31, 64, 127, 32, 127,

对外层来说，x, y 是局部量，对内层分程序说，x 是局部量，y 是非局部量。

(2) 1, 1, 4, 13,

对本分程序来说，i, x 都是非局部量。

4.

```
    begin
      integer i;
```

```
      real y, a, b, z, x;
      #read (0, '10', a, b);
      for x:=0.6, 1.25, 2.3, 1.8, 0.45 do
        begin switch s:=L1, L2, L3;
          i:=#entier(x)+1; goto s[i];
        L1: y:=#sin(a×x)/#cos(a×x)+b; goto L4;
        L2: y:=a×a-b×b×x×x; goto L4;
        L3: y:=#ln(a×a+b×b×x×x);
        L4: z:=(#cos(y)↑2+#exp(y))×#sqrt(a×a+b×b);
            #print (0, '2F14.6', x, z)
        end
    end 55···5
5.  begin
      real a, b, c, M; integer i; array z[1:3];
      #read (0, '10', a, b, c);
      for i:=1 step 1 until 3 do
        begin integer j, i1;
          i1:=10×i;
          begin array x(30)[1:i1];
              switch k:=L1, L2, L3;
            #read (0, '10', x); goto k[i];
          L1: z[1]:=a;
              for j:=1 step 1 until 10 do
                z[1]:=z[1]+x[j];
              goto L4;
          L2: z[2]:=b;
              for j:=1 step 1 until 20 do
                z[2]:=z[2]+x[j]↑2;
              goto L4;
          L3: z[3]:=c;
```

```
            for j:=1 step 1 until 30 do
              s[3]:=s[3]+x[j]↑5;
        L4: end
      end;
    M:=0;
    for i:=1 step 1 until 3 do
      M:=M+s[i];
    #print (0, '2F14.6', s, M)
  end 55…5
```

习 题 五

1. 函数过程说明的形式：

```
begin
  real y, s, a, B;
  real procedure f(x);
      value x; real x;
    f:=x×x+#sqrt (1+2×x+3×x×x);
  #read (0, '10', y, s);
  a:=f(y×y); a:=#sin(y)/a;
  B:=f(2×s)+a×a-1;
  #print (0, '2F14.6', a, B)
end 55…5
```

一般过程说明的形式：

```
begin
  real y, s, a, B, f1, f2;
  procedure f (x, y);
      value x; real x, y;
    y:=x×x+#sqrt (1+2×x+3×x×x);
  #read (0, '10', y, s);
    f (y×y, f1); f (2×s, f2);
```

```
        a:=#sin (y)/f1;
        B:=a×a-1+f2;
      #print (0, '2F14.6', a, B)
    end 55…5
```

2.
```
    real procedure s(n, a);
      value n; real n; array a;
    begin
      integer i; real s1;
      s1:=0;
      for i:=1 step 1 until n do
          s1:=s1+a[i, i];
      s:=s1
    end;
```

3.
```
    procedure MV (n, a, b, c);
      value n; real n, c; array a, b;
    begin integer i;
      c:=0;
      for i:=1 step 1 until n do
          c:=c+a[i]×b[i]
    end;
```

4. 不能，因为 x, y, s 都是作为存放结果的形式参数。

5. i 是过程体的局部量，c 对过程体来说是局部量，对其内层分程序来说是非局部量、n 是内层分程序的局部量，i 是非局部量。

6.
```
    begin
      real m, m1;
      #read (0, '10', m); m1:=2×m+1;
      begin
        real w, y, x; array c(100)[0:m1];
        real procedure poly (n, x, a);
            value x, n; real x, n; array a;
```

```
        begin real i, sum;
            sum:=0;
            for i:=n step-1 until 0 do
              sum:=sum×x+a[i];
            poly:=sum
        end;
      #read (0, '10', y, z, c);
        w:=poly (m1, y×y+z×z, c);
      #print (0, '2Hw=, F14.6', w)
    end
  end 55…5
7.  begin
      integer n, n1;
        #read (0, '10', n); n1:=n-1;
        begin array ak, bk (100)[0:n1];
          real a, b, h, h1, k, i, x, y;
          real procedure f (x); value x; real x;
            f:=if #abs (x)>1 then x/2 else 2-1.5×x;
          #read (0, '10', a, b);
          h:=(b-a)/2/n; h1:=6.2832/(b-a);
          for k:=0 step 1 until n-1 do
            begin ak[k]:=bk[k]:=0;
              for i:=0 step 1 until 2×n-1 do
                begin x:=a+i×h; y:=f(x); x:=k×h1×x;
                  ak[k]:=ak[k]+y×#cos(x);
                  bk[k]:=bk[k]+y×#sin(x)
                end;
              ak[k]:=ak[k]/n; bk[k]:=bk[k]/n
            end; #print (0, '2F14.6', ak, bk)
  end end 55…5
```

参 考 资 料

[1] 电子计算机常用算法。中国科学院沈阳计算技术研究所、后字414部队、北京工业大学计算站编。科学出版社，1976年。

[2] 简明数学手册。《简明数学手册》编写组编。上海教育出版社，1977年。

[3] 计算技术通讯 (1975, 1～2, 1977, 1)。上海计算技术研究所情报资料室。

[4] J. H. Wilkinson, C. Reinsch, Linear Algebra, 1971.

[5] 常用算法标准程序汇编。上海计算技术研究所一室编。即将出版。

[6] 数字计算机上用的数学方法 (I)、(II)。A. 拉尔斯登，H. S. 维尔夫等著。上海科学技术出版社，1967、1976年。

[7] 有限元素法续讲。复旦大学数学系应用数学教研组编。1976年。

[8] 回归分析与设计。上海师范大学数学系应用数学教研组编。1976年。

[9] 概率统计计算。中国科学院计算技术研究所三室概率统计组。1976年。

[10] TQ-16机编译系统使用说明。上海工业建筑设计院计算组编。1978年。

[11] ALGOL语言程序设计讲义 (TQ-16计算机)。武汉测绘学院李鸣山、胡笔蕊编。1978年。

[12] 709机编译系统使用手册。上海计算技术研究所编。1972年。

[13] 709机编译系统修改使用说明。上海计算技术研究所编。1977年。