

前 言

近年来,有关计算机的书颇受欢迎,这对我国计算机的普及和应用,起着良好的推动作用。

但是,目前还有不少单位的微型计算机尚未充分发挥其应用的作用,其主要原因是对各种各样的微型计算机开发的还不够。为了帮助广大初、中级计算机应用人员以及学过 BASIC 语言的中、小学生掌握编制程序的方法和技巧,作者根据近年来从事计算机实践和教学经验,拟就上述问题作些补充,力求熔实用性、通用性、技巧性、趣味性于一炉,在编程方法、设计技巧和实际操作诸方面作一些探索性尝试。

本书强调普及和提高结合,实用和技巧兼顾,以满足不同层次读者的需求。

本书的资料主要来自作者多年编程和教学实践,同时还吸收了国内外的先进经验。书中阐述的方法、技巧、思路对其它程序设计语言都有参考价值,而实用程序可以直接付诸使用或稍作修改后引用。

在本书编写过程中,得到南京大学大气科学系邹进上教授的鼓励与支持,并提出不少宝贵意见,在此谨表谢意。

由于作者学识疏浅,成书仓促,错误缺点在所难免,敬请广大读者不吝指教。

编 者

1987年8月25日

目 录

前言

一、什么是程序设计和技巧	(1)
1. 什么是程序设计	(1)
2. 各种情况	(1)
3. 质量标准	(2)
4. 出发点	(3)
5. 基本方法	(4)
6. 基本技巧	(4)
7. 实例分析	(6)
二、程序设计步骤和方法	(18)
1. 了解问题性质	(18)
2. 绘制程序框图	(19)
3. 强调设计思想	(20)
4. 建立数学模型	(23)
5. 选用计算方法	(24)
6. 注意近似处理	(26)
7. 考虑通用合理	(27)
8. 防止程序出错	(28)
9. 划分功能模块	(29)
10. 讲究设计技巧	(31)
三、程序连接的方法	(34)

1. 如何安排总控.....	(34)
2. 单一程序的连接方法.....	(36)
3. 两个不同程序的连接方法.....	(40)
4. 重订行号的方法.....	(45)
四、节省内存和提高速度的方法.....	(48)
1. 程序存贮的基本知识.....	(48)
2. 节省内存的措施.....	(52)
3. 提高速度的方法.....	(58)
4. 使用机器语言子程序.....	(62)
五、编制程序应注意的问题.....	(67)
1. 关于变量的初始化问题.....	(67)
2. 数组元素的下标要恰当.....	(69)
3. 程序的正确性要经得起时间的考验.....	(70)
4. 防止陷入死循环.....	(71)
5. 矩阵输出注意打印语句的安排.....	(72)
6. 妥善处理保留字.....	(73)
7. 重新读数不要忘记 RESTORE.....	(74)
8. 能用简单变量的就不用下标变量.....	(76)
9. 注意程序的清晰.....	(78)
10. 判断循环变量是否改变.....	(80)
11. 数值计算不要超过机器规定范围.....	(81)
12. 注意使用扩展 BASIC 语句.....	(82)
13. 尽量不用 GOTO 语句.....	(84)
14. 尽量少用乘方运算.....	(86)
六、打印输出的技巧.....	(90)
1. 输出空格函数 (SPC函数) 的使用.....	(90)

2. 输出定位函数 (TAB函数) 的使用	(91)
3. 数据的列印输出	(93)
4. 打印机输出控制	(95)
5. 打印输出结果的对齐处理	(98)
6. 字符的放大打印	(104)
七、程序调试技巧	(107)
1. 不同机器的差异	(107)
2. 尽量避开乘方运算	(110)
3. 作适当的精度调整	(111)
4. 如何处理小数步长	(112)
5. 屏幕抄写的方法	(113)
6. 程序编辑举例	(114)
7. 执行过程的自动跟踪	(116)
8. 输入数据的跟踪检查	(118)
9. 几个常用技术	(120)
八、BASIC 语言的编程技巧	(123)
1. 菜单技术	(123)
2. 人机对话方式	(125)
3. 巧用符号函数 SGN(X)	(128)
4. 巧用逻辑表达式	(131)
5. 循环体中判断语句的巧安排	(133)
6. 多用途的 INT(X)函数	(134)
7. 字符串函数的应用技巧	(139)
8. 绝对值函数的一个应用	(142)
9. 多重循环程序的设计技巧	(145)
参考文献	(156)

一、什么是程序设计和技巧

程序设计是以完成预定任务为目的，这是程序设计基本要求。然而常常有这样的情况，一个实际问题，不同的人，不同的思路，编制的程序却是多种多样。这就要讨论程序设计的标准是什么？要依据什么原则来制定这些标准？用什么方法使设计的程序达到标准。在程序设计中究竟有哪些基本方法，又有什么技巧等等。

1. 什么是程序设计

人们根据问题的要求，事先安排好的处理方法和步骤，称作程序。一个程序好比是一篇文章，进行程序设计的过程就是人们按照计算机语言所规定的语法和功能书写文章的过程，这个过程一般来说有以下几个阶段：提出问题，需求分析，确定模型，功能设计，绘制框图，编写程序，动态调试，交付使用。

2. 各种情况

考察人们编写的各种程序，会发现程序的编制情况多种多样，归纳起来有：

- 设计思想简单或复杂；
- 程序冗长或简短；
- 运行速度快或慢；

- 精度高或不够；
- 占用内存多或少；
- 计算方法优或劣；
- 使用方便或相反；
- 阅读修改程序难或易；
- 程序移植性好或差；
- 用计算机协助编程或没有。

从上面情况看出，程序设计灵活性很大，技巧性很高。

3. 质量标准

那么，什么是一个好的程序呢？不同的人有不同的观点，不同的课题要求的标准也不一样，所以很难有一个统一的标准。然而，根据计算机用于数据处理的性质和特点，一个好的程序必须满足以下几个方面。

(1) 正确性 编制程序的目的，就是完成解题任务，这是程序设计的最基本要求。一个正确的程序必须保证在任何情况下，都能得到正确的运行结果。正确性还应包括纠错能力，程序结构精巧，使用灵活方便，结果正确并能达到预定的精度要求。

(2) 通用性 通用性包括几方面内容：一是一个程序对同一类问题的不同情况都可适用；二是不能因人、因时、因地而异；三是一个程序稍作修改后，能完成不同功能或满足不同解题要求。

(3) 可靠性 对一个系统来说，可靠性和正确性同样至关重要。一个程序不能运行一次正确，多次运行就失败；或者调试时可行，但进入反复操作后出问题；或者数据量较少

时正确，一旦大批量输入时出现异常情况，所有这些都体现了程序的可靠性。一个好的程序，应该经得起长时间及大批量数据的考验。

(4) 实用性 程序设计应具有实用价值，讲究经济效益。除此以外，程序设计应考虑到如何使别人使用时感到方便、灵活，愿意用自己设计的程序。程序的易读性要好，又便于修改。因此实用和面向用户，是评价一个程序优劣的重要标准。

4. 出发点

有了程序设计标准，对程序设计的评价就有了依据，一般来说，我们希望编制的程序结构清晰，易读易改，便于移植，结果精确，使用方便。同时在程序设计中还有几点应予注意：

- 解决一个实际问题的方法是多种多样的，设计思路也不尽相同，但我们的立足点还应放在节省机时和减少存贮空间上，只有当两者不能兼顾时，可根据实际需要而偏重一方。

- 不应盲目追求快的响应速度，而损失掉许多别的优点，或者为了提高速度而在其它方面带来不应有的麻烦。要切合实情，量力而行，不要得不偿失。

- 也不要过分追求减少内存空间，使程序易读性很差，带来阅读、调试、修改的困难。我们的出发点是，为确保程序运行的可靠及方便用户，要不惜代价增加一些必要的语句或注释，同时也要尽可能地节省内存。

- 要讲究技巧，注意程序的优化。由于在数据、信息处

理中，是人-机直接交互，因此，应用一些技巧效果会被人直接感受到、体会到，而当人们在实际中切身感到好处时，技巧的成效就越发明显。当然，一个程序也不是越巧越好，拐弯越多越好，严防弄巧成拙，适得其反。

5. 基本方法

程序设计技术性强，也很灵活，有些问题的设计还很复杂，没有统一的模式，但是，根据程序设计的基本思路，性质及过程，总还能总结出一些比较普遍的方法。现简叙如下：

程序设计的基本方法是枚举、归纳和抽象。

建立在抽象原则基础上的一种常用设计方法是逐步求精。

对于规模较大的问题的程序设计，是采用结构模块化。

逐步求精法是一种适应性很强并被广泛采用的程序设计方法，其要点是：

- 对问题全局进行分析；
- 建立抽象模型；
- 确定总体模块图；
- 明确各子模块功能及相互关系；
- 编制、调试每一个子模块；
- 完善整体结构。

6. 基本技巧

程序设计的技巧，包括的内容是十分广泛的，主要有语言编程技巧，程序调试技巧，文件使用技巧，软件加密技

巧，机器使用技巧，功能扩展技巧，数据维护技巧等等。对于这些技巧，将在本书其它章节逐一叙述。

在用BASIC语言进行程序设计时，主要有下面四种技巧：

(1) 简单程序设计 这是初学者程序设计入门的必经之路，它包括数据输入、进行计算、输出结果、结束停机四个内容。由于程序是按行号由小到大顺序执行的，因而比较简单。常用的语句有 LET, INPUT, READ/DATA, PRINT, REM, END 等。

简单程序设计虽然容易，但它能勾画出一个程序的全貌，描述了程序的主要结构，即变量取值、完成计算并保留结果、提供数据、显示结果和结束程序这五个部份。明确这一点，对我们设计一个程序是十分重要的，因为任何事物都是由低级到高级，由简单到复杂。

(2) 分支程序设计 实际问题比较复杂，解决问题的程序常常带有分支，因此，我们十分需要一种具有判断能力的控制转移手段，以便对程序进行有效的控制。这就出现了分支程序设计，而计算机按给定条件进行比较、分析、判断后决定程序的走向。常用的有无条件转向语句 GOTO，条件控制语句 IF/THEN，恢复数据区语句 RESTORE，控制转向语句 ON/GOTO, ON/THEN, ON/GOSUB，以及转子和返回语句 GOSUB/RETURN 等等。

我们可以根据一定的条件，选择不同的语句，控制程序的流向，以解决各种各样的分支问题。必须着重指出，条件语句的使用，在 BASIC 语言中占有很重要的地位，同样，其技巧性也是很强的。

(3) 循环程序的设计 循环程序在我们解决实际问题的

时候，有着极其广泛的应用。循环的技巧可以实现多个数据输入和大量重复的计算。利用循环的方法进行程序设计，既可节省存贮单元又可以提高计算效率，这是程序设计中最主要的和最常用的方法。循环语句比较简单，就是FOR/NEXT，但应用时技巧特强，且十分灵活。

不要忘记，人们利用计算机快速准确且具有逻辑判断的特点，为自己解决那些需要反复进行，而在处理形式上又完全相同的工作时，主要是依靠循环程序来进行。

循环是 BASIC 语言和一切高级语言的核心。

(4) 调用子程序的设计 子程序是程序结构的一种形式，使用十分广泛。子程序在程序设计中常常用来表示那些在功能上具有一定独立性并在程序的不同部位要被经常使用的程序段，用以简化设计并使程序结构清晰。

对于一些重复计算的问题，编制一个相对独立的程序段，在主程序的适当位置安排调用点，GOSUB/RETURN 语句能保证正确调用和自动返回。

对一个规模较大或需要完成多种功能的系统来说，往往需要分段设计，然后连接起来，子程序的技巧可以解决程序段之间的连接问题。使整个程序或系统成为一个有机结合的整体。

程序设计的基本技巧，基本上就是上面四个方面。在这里不拟逐一详述，但须指出一点，即实际问题的设计常常是四种基本技巧的有机组合。

7. 实例分析

一个好的程序应包括下述指标：

- 运行结果正确;
- 有足够的精度;
- 运行速度较快;
- 占用内存较少;
- 简单清晰易懂。

而所有这一切常常取决于好的解题方法。

下面我们通过两个实例的求解，来探讨一下不同的解题方法对程序质量的影响。

例 1，求：

$$Y = 1 - 2^2 + 3^2 - 4^2 + \cdots + 99^2 - 100^2$$

方法 1，改写原式：

$$\begin{aligned} Y &= (1^2 + 3^2 + 5^2 + \cdots + 99^2) - (2^2 + 4^2 + 6^2 + \cdots + 100^2) \\ &= A - B \end{aligned}$$

式中： $A = 1^2 + 3^2 + \cdots + 99^2$

$$B = 2^2 + 4^2 + \cdots + 100^2$$

问题归结为求奇数平方和及偶数平方和的差值。

求和最好用循环语句，循环外设置初值为零，循环体求平方和，即 $S = S + N \wedge 2$ ，最后安排 $Y = A - B$ 并打印，故有程序 No1：

```
10: REM NO. 1
20: A=0
30: FOR N=1 TO 99
    STEP 2
40: A=A+N^2
50: NEXT N
60: B=0
```

```

70:FOR N=2TO 100
  STEP 2
80:B=B+N^2
90:NEXT N
100:Y=A-B
110:LPRINT "Y=";Y
120:END

```

Y=-5050

这个程序结构清楚，层次分明，易读性好。不足之处，占用内存多（120步），速度慢（24秒）。

方法2，将程序 No1 改写成 No1a:

```

10: CLEAR : REM NO.
  1a
20:FOR N=1TO 99
  STEP 2:A=A+N^2
  :NEXT N
30:FOR N=2TO 100
  STEP 2:B=B+N^2
  :NEXT N
40:LPRINT "Y=";A-
  B

```

Y=-5050

程序 No1a 和程序 No1 有以下几点区别:

①两个程序设计思想完全一样，程序 No1a，改用扩展 BASIC 语句，因而比较简短，且节省内存（变成 85 步）。

②运行时间缩短了, 由 24 秒变为 22 秒。

③用 CLEAR 语句, 省去两个循环的初始化条件, 即删去 $A=0, B=0$ 。

④程序 No1 的 100、110 两句, 合并成 No1a 的 40 句。

⑤省去 END 语句, 一般微型计算机都是允许的 (FORTRAN 语言不允许省去结束程序语句 END)。

⑥为了提高速度, 可以把 REM 注释语句放在程序的最后, 因为它不是执行语句, 不影响机器运行速度。

⑦程序 No1a 的第 10 句, 若写成:

10:REM:No1a: CLEAR 则重复运行程序 No1a, 每次结果会不同。因为 REM 是非执行语句, 这样 CLEAR 这一句将不执行, 存储在 Y 中的值没有被清除掉, 重新运行时 Y 值包括了前一次的结果。

由于上述两个程序设计思想完全相同, 后者虽然节省了一些内存, 速度也有微小提高, 但程序质量上并没有实质性的提高。还需要继续改进。

方法 3, 考察原式: $Y = 1 - 2^2 + 3^2 - 4^2 + \cdots + 99^2 - 100^2$

从数值上看, 它们是顺序递增的, 每一项均为 N^2 ($N = 1, 2, 3, \cdots 100$)。

从符号上看, 每一项又是 “+”、“-” 相间的, 其通式为 $(-1)^{n+1}$, n 是奇数时为 +1, n 是偶数时为 -1。

故有以下通项公式:

$$a_n = (-1)^{n+1} \cdot n^2 \quad (n = 1, 2, 3, \cdots 100)$$

$$\therefore \text{原式为: } Y = \sum_{n=1}^{100} (-1)^{n+1} \cdot n^2$$

见程序 No2:

```

10:REM NO.2
20:Y=0
30:FOR N=1TO 100
40:Y=Y+(-1)^(N+1)
   *N^2
50:NEXT N
60:LPRINT "Y=";Y

```

Y=-5049.999999

从上面运行结果可以看出，这个程序除了比较简洁，内存少（72步）外，没有什么优点，相反，缺点倒是明显的，结果不精确，速度慢。同时，本程序中用了负数的乘方运算，有的机器是不执行的，如SHARP PC-1211。

其它机器如SHARP PC-1500，APPLE-Ⅱ，紫金-Ⅱ，中华学习机等均能执行本程序，但精度却不够。正确值为-5050。

方法4，为了避开上述解法中负数的乘方运算，对符号和负数乘方采用下面的方法处理，见程序№3清单：

```

10:REM NO.3
20:Y=0:T=-1
30:FOR N=1TO 100
40:T=(-1)*T:Y=Y+T
   *N^2
50:NEXT N
60:LPRINT "Y=";Y

```

Y=-5049.999999

程序中 $Y=0$ 是为了求和, $T=-1$ 和 40 句对应, n 为奇数 T 为正, n 为偶数 T 为负, 从而解决了 “+”、“-”相间的问题。

但是这个程序也不佳, 速度慢(25秒), 精度不高。唯一的特点是避开了负数的乘方运算, 这样任何计算机均可执行。

顺便说明一下, 在 BASIC 语言中, 求负数的乘方是不允许的, 因为机器做乘方是化成对数来做, 负数的对数无意义。因而一般机器将拒绝执行。有的机器所以会执行(如 PC-1500), 是因为软件程序(BASIC 语言用解释程序)具有更多的功能。

方法 5, 改写原式:

$$Y = (1^2 - 2^2) + (3^2 - 4^2) + \cdots + (99^2 + 100^2)$$

其通项式为: $a_n = n^2 - (n+1)^2$ ($n=1$, 为第一项, $n=3$ 为第二项, \cdots , $n=99$ 为第 50 项)

展开上式: $a_n = -(2n+1)$ ($n=1, 3, 5, \cdots, 99$)

故原式为:

$$Y = \sum_{n=1}^{99} -(2n+1) \quad (n=1, 3, 5, \cdots, 99)$$

\therefore 对上式只要安排一个循环, 步长 STEP 为 2 即可, 见程序 No.4.

```

10:REM NO.4
20:Y=0
30:FOR N=1 TO 99
  STEP 2
40:Y=Y-(2*N+1)
50:NEXT N
60:LPRINT "Y=";Y

Y=-5050
    
```

程序 №4 和前面几个程序相比，有以下几个特点：

①程序较短，占用内存少（67步）。

②运行较快，只用了 2 秒钟，比其它几个程序时间缩短了十多倍。

③精度高，结果正确。

④只有一重循环，简捷明了。

⑤计算方法好，避开了乘方运算，只进行 $Y = Y - (2 * N + 1)$ 的操作。

方法 6，本题还有更好的解法，事实上：

$$\begin{aligned} Y &= 1^2 - 2^2 + 3^2 - 4^2 + \cdots + 99^2 - 100^2 \\ &= (1^2 - 2^2) + (3^2 - 4^2) + \cdots + (99^2 - 100^2) \\ &= (-3) + (-7) + (-11) + \cdots + (-199) \\ &= -N(N+1)/2 \quad (n=2, 4, 6, \cdots, 100) \end{aligned}$$

$N = 2$ 时，是计算原式最初两项的值， $N = 4$ 时，是求前四项的结果， $N = 100$ 时，即为原式的最后答案。

见程序 №5：

```
10:REM NO.5
20:N=100:Y=-N*(N+
  1)/2:LPRINT Y
```

--5050

显然，程序 №5 比上述各程序为优。基本上符合一个优化程序的各项指标要求。

例 2，一个提高程序运行速度的实例：

大家都知道古代印度国王舍罕褒赏他的宰相达依尔（国际象棋发明者）的故事吧！国王问宰相需要什么，达依尔说：“国王只要在国际象棋棋盘的64个格子里放满小麦，我就感恩不尽了。我只要求第一格放一粒，第二格放二粒，以后每格放的粒数是前格的二倍。”国王应允了，结果全印度的粮食全部用完也没有填满棋盘的格子。据说多少年来谁也没有算清过这笔帐，当然更谈不上谁是算得最快的人。

这个问题实际上是求解算式：

$$Y = 1 + 2 + 2^2 + \cdots + 2^{63}$$
$$= \sum_{N=0}^{63} 2^N$$

解法 1，用分支和计数求和的方法计算，见程序№6，

```
10:REM NO.6
20:Y=0:N=0
30:Y=Y+2^N
40:N=N+1
50:IF N<=63THEN 3
   0
60:LPRINT "Y=";Y:
   END
```

Y= 1.844674407E 19

上述程序运行时间11秒（机种PC-1500A）。

解法 2，用循环语句求解，见程序№7，

```

10:REM NO.7
20:Y=1
30:FOR N=1 TO 63
40:Y=Y+2^N
50:NEXT N
60:LPRINT "Y=";Y:
END

```

Y= 1.844674407E 19

本程序运行时间 9 秒。

比较上面两个程序,运行结果相同,设计思想相近,但运行速度各异。一般来说,用FOR/NEXT语句,比用IF/THEN语句运行时间要短。这是由于两种语句所编程序在运行时走的路径不同。用FOR/NEXT语句循环时,运行至NEXT后加一个步长,然后立即转到循环体或执行NEXT下面的一个语句,而IF/THEN语句,运行至THEN $\times \times$ 时,要返回到指定的 $\times \times$ 行号,这时机器要从头顺序查找,一直找到 $\times \times$ 行号,才继续运行,所以它比FOR/NEXT语句,在执行速度上费时较多。

解法 3, 避开乘方运算:

上面两个程序,也可以不用乘方运算。这对提高程序的运行速度是有利的。

考虑原式各项,后一项均为前一项的 2 倍。即

$$a_n = a_{n-1} \times 2$$

这就给我们一个启示:如果在计算前一项 a_{n-1} 后,能将

a_{n-1} 项的值保存好，则在求后一项 a_n 时，只要乘以 2 即可。这样就避开了乘方运算，它可以大大节省计算时间。故有程序 No8:

```

10:REM NO.8
20:A=1
30:Y=A
40:FOR N=1 TO 63
50:A=A*2
60:Y=Y+A
70:NEXT N
80:LPRINT "Y=";Y:
END

```

Y= 1.844674406E 19

上述程序运行时间为 4 秒。

从以上三个程序，还可以看到这样一个事实：解法 1、2 程序较短，但运行速度较慢。解法 3 程序虽长一点，但运行时间较短。由此可知，程序的长短，是和占用内存的多少成比例的，但和程序执行的时间并不完全成正比。常常有这样的情况，程序较长，计算方法较好，运行速度很快，这就是所谓以存贮空间为代价换取了速度的提高。

解法 4，用数学归纳法来解：

考察原式：
$$Y = \sum_{N=0}^{63} 2^N$$

若取一项 则 $N=0$ $Y=2^0=1$
 $=2^{N+1}-1$

$$\begin{aligned}\text{事实上} \quad N=0 \quad Y &= 2^{N+1} - 1 \\ &= 2^1 - 1 \\ &= 1\end{aligned}$$

$$\begin{aligned}\text{若取二项 则 } N=1 \quad Y &= 2^0 + 2^1 \\ &= 1 + 2 \\ &= 3\end{aligned}$$

$$\begin{aligned}\text{事实上} \quad N=1 \text{ 时 } Y &= 2^{N+1} - 1 \\ &= 3\end{aligned}$$

同理取 64 项, 则 $N = 63$

$$\begin{aligned}\therefore \quad Y &= 2^0 + 2^1 + \cdots + 2^{63} \\ &= 2^{N+1} - 1 \\ &= 2^{63+1} - 1 \\ &= 2^{64} - 1\end{aligned}$$

故有程序 No.9:

```
10:REM NO.9
20:Y=2^64-1
30:LPRINT "Y=";Y:
END
```

Y= 1.844674407E 19

运行时间: 1 秒

同上面三个程序相比, 本程序较好, 程序短, 占用内存少, 运行速度快。

最后说明一下, 前面提到了少用乘方的观点, 特别是要求精度较高的计算, 最好不用乘方, 因为它的计算既不够精

确，且速度又慢。为什么解法 4，仍然用了乘方运算，而不失为一个好的程序呢？道理也是不难明白的。

程序 No6, No7 都用了乘方运算，并且都做了 64 次的乘方求和，而程序 No9 只做了一次乘方 2^{64} 然后减 1，所以程序 No9 快。

程序 No8 虽然不做乘方，但反复做乘法和累加，并进行 63 次循环（FOR/NEXT 循环），是需要时间的，而程序 No9，不要循环，一次求乘方解决，因而较快。

通过上述两个实例分析，我们初步勾画了程序设计技巧的部份面貌。可以看出，找到一个好的解题方法，是高质量地进行程序设计的核心。这些实例也可以作为程序设计技巧的引子，希望能起到开阔视野、启迪思维、抛砖引玉的作用。

二、程序设计步骤和方法

为利用计算机来解决实际问题，一般需要经过如下几个步骤。

1. 了解问题性质

这是整个编制过程的基础，也是初学者常常容易忽视的问题。

了解问题的性质，就是分析待编系统要解决什么问题，有什么特殊要求，有多大的信息量，数据量和内存有无矛盾、运行速度是否满足要求，变量够不够用等等。只有对上述诸问题进行仔细分析，才能找到解决问题的办法。

例如，编制一个可以满足任意精度的除法运算程序。

分析：

① 题目要求编制满足任意精度的除法算题程序，这就要求我们认真思考了，因为计算机输出的有效数，其位数是有限的，中华学习机，APPLE-II，紫金-II，PC-1500等机型都是八位字长微机，大都只能输出6—8位有效数，它们都不能满足本题要求。解决满足任意精度的问题，就是本题的难点和关键所在。

② 除数不能为0，否则除法无法执行。

③ 程序中应考虑判别商是正数还是负数的条件语句，否则结果不正确。

④ 打印输出中，必须考虑商的整数部份和小数部份。

⑤ 为了处理小数点以后各位的运算过程，以达到任意精度的要求，必须考虑不够补“0”的问题。

⑥ 为满足任意精度要求，程序中小数点以后商的各位，必须正确安排，要求多少位，就算到多少位。

通过上述分析，我们对本题可以说有了一个清楚的了解，剩下的问题，就是按上述各点分析安排程序结构了。

2. 绘制程序框图

程序框图是一种流程图，它是用各种几何图形及文字说

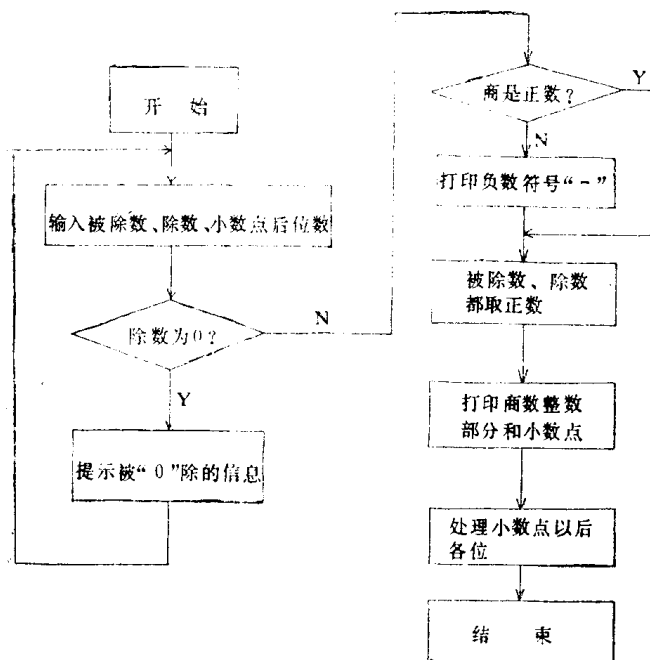


图 1

明来直观地描述计算过程的有向图。它的优点是形象、直观，既能表达程序的设计思想，又能看清程序的编制结构，同时便于阅读、检查、修改程序。学会并习惯画框图，是编制程序的基本功，它被广泛采用在各种高级语言的程序设计中。

例如对上面的高精度除法运算，我们可以绘制成图 1。

我们再次强调，掌握框图的绘制技术是十分重要的，不言自明，一个大型应用程序的设计，不使用框图很难顺利地将程序各语句书写出来。

框图可以使程序设计减少错误，又是从事计算机应用工作者之间一种技术交流的手段和工具。

3. 强调设计思想

这是编制任何一个实际问题的程序都要首先考虑的。在弄清问题的性质、特点、要求以后，如何解决问题，应在头脑中形成一个完整的思索、判断、推理。

对问题分析得愈透彻，考虑得愈周到，出错的机会就愈少，不仅如此，程序设计思想的优劣，关系到程序设计的质量。

例如，上面提及的除法程序，在设计思想上应如何考虑呢？

为了实现高精度除法运算，可以模仿人们手算除法的过程，先求商的最高位，并把它打印出来，再用余数去除以除数，求得商的次高位，如此循环往复，……。这一过程只要商不是一个有限小数，就可以一直进行下去，直到你所希望的精度为止。

变量设置：X, Y, Z 分别为被除数、除数、精度。精度指小数点以后的位数。

M 为商，即 $M = \text{INT}(X/Y)$

安排一个循环，以处理小数点以后各位的运算过程，即余数变成被除数，仍用 X 代表余数； $X = (X - M * Y) * 10$ ，其中乘以 10，完全是模仿人工笔算除法的过程，不够补“0”，循环程序如下：

```
120  FOR J = 1 TO Z
130  X = (X - M * Y) * 10
140  M = INT (X / Y)
150  PRINT M;
160  NEXT J
```

这里 140 句中的 M 是指小数点以后商的各位，Z 为要求的精度，要求多少位，就算到多少位。

对于除数不应为“0”的问题，在程序中只要安排判断语句，看输入是否合理，故有以下程序段：

```
20  INPUT X, Y, Z
25  PRINT "X="; X, "Y="; Y, "Z="; Z
28  PRINT "X/Y="; X / Y
30  IF Y < > 0 THEN 60
40  PRINT "DIVISION BY ZERO ERROR"
50  GOTO 20
```

为判别商是正数还是负数，安排下面二条语句：

```
60  IF X * Y > = 0 THEN 80
70  PRINT "-";
```

当商的正、负问题处理后，被除数和除数均以正数处理，即：

```

80 X = ABS (X)
90 Y = ABS (Y)

```

利用取整函数INT(K),可以方便地求出商的整数部份,
第一次要把商打印出来,并在商M的后面打印一个小数点,
即,

```

100 M = INT (X / Y)
110 PRINT M; ".";

```

至此,全部设计完成,下面给出程序 № 10 的完整清单和一个运算实例:

ULIST

```

10 REM NO.10
20 INPUT X,Y,Z
25 PRINT "X=";X,"Y=";Y,"Z=";Z
28 PRINT "X/Y=";X / Y
30 IF Y < > 0 THEN 60
40 PRINT "DIVISION BY ZERO ERROR"

50 GOTO 20
60 IF X * Y > = 0 THEN 80
70 PRINT "-";
80 X = ABS (X)
90 Y = ABS (Y)
100 M = INT (X / Y)
110 PRINT M; ".";
120 FOR J = 1 TO Z
130 X = (X - M * Y) * 10
140 M = INT (X / Y)
150 PRINT M;
160 NEXT J
170 END

```

```

724513
??119
??60
X=24513          Y=119          Z=60
X/Y=205.991597
205.991596638655462184873949579831932
773109243697478991596638655

```

上面这个程序实例，是计算到小数点以后 60 位，理论和实践证明，可以精确到小数点以后任意位。

28 句，是计算机一般进行除法运算的结果，它只给出小数点以后 6 位有效数字，写到程序中，是为了与高精度除法作为对比。

注意：110 句打印的 M 值，是商的整数部份，而 150 句输出的 M 值，是小数点以后的各位值。

4. 建立数学模型

任何一个实际问题，最后都要用数学表达式来描述，再按照 BASIC 规则书写程序。因此，把实际问题抽象成数学模型，是编制程序的又一重要方法。

例如，某车站托运行李的收费规定，小于 50kg，按 0.15 元/kg 收费，超过 50kg，其超过部份按 0.20 元/kg 收费。

对此问题，可以按下述公式处理：

$$X = \begin{cases} 0.15 \times W & 0 < W \leq 50\text{kg} \\ 0.15 \times 50 + (W - 50) \times 0.20 & W > 50\text{kg} \end{cases}$$

式中： W 为货物重量， X 为收费金额。

这里的计算公式，就是对实际问题的数学处理，也就是说，我们建立了数学模型。

又如求两点间的距离，其数学公式为：

$$D = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

求自然数 1 到 20 的阶乘的和，其数学表达式为：

$$\sum_{n=1}^{20} n!$$

由此可见，不同的问题，其数学模型是不一样的，而一旦正确建立了数学模型，编制程序就容易多了。

5. 选用计算方法

一个计算问题可能在不同的条件下使用不同的公式，而同一个问题即使取相同的公式也可能有不同的计算方法，选用好的解题方法，是高质量的进行程序设计的核心。

例如，求 S 刚刚超过 5 的调和级数的项数。即求：

$$S = \sum_{i=1}^{\infty} \frac{1}{i}$$

下面给出几种解题方法，试比较其优劣：

①

```
10 S = 0
20 FOR I = 1 TO N
30 S = S + 1 / I
40 PRINT I, S
50 NEXT I
60 END
```

②

```

10 S = 0
20 I = 1
30 S = S + 1 / I
40 IF S > 5 THEN 70
50 I = I + 1
60 GOTO 30
70 PRINT S, I
80 END

```

③

```

10 S = 0
20 FOR I = 1 TO 200
30 S = S + 1 / I
40 IF S > 5 THEN 60
50 NEXT I
60 PRINT S, I
70 END

```

④

```

10 S = 0
20 FOR I = 1 TO 200
30 S = S + 1 / I
40 IF S > 5 THEN PRINT S, I: END
50 NEXT I

```

第一种方法不行，因为 N 是未知数，对无赋值的变量，有的机器不接收，有的机器当“0”处理，但都指出语法错误。第二种方法是正确的，但程序编制稍为复杂一些了，用GOTO语句来控制循环，执行速度较慢。第三种方法可以，取 $N = 200$ ，是一个投试的量，对本题可以采用假设一个已

②

知量，投试结果，唯程序不够简练。第四种方法较好，40句的安排就是一个技巧，当得到满足题意的解后，即打印输出，跳出循环，结束停机。

总之，解决一个实际问题的方法是多种多样的。应该选择那些既能节省机时又能节省存贮空间的方法，它需要人们仔细研究、相互借鉴。当两者不能兼顾时，根据实际情况偏重一方。

6. 注意近似处理

运行结果正确并达到预期的精度要求，是进行程序设计时应该考虑的一个标准，但是，也有这样的情况，有些计算并不要求很高的精度，已能满足题意要求，这时我们就要考虑近似处理。

例如，用下述公式：

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{N!}$$

来求 e 值时，一般 N 取 10，已能满足精度要求，见程序 No.11，

```
5  REM  NO.11
10  INPUT N
20  A = 1:B = 1
30  FOR I = 1 TO N
40  A = A / I
50  B = B + A
60  NEXT I
70  PRINT "e=";B
80  END
```

下面是几种运行结果:

$N = 10$ 时, $e = 2.7182818$

$N = 100$ 时, $e = 2.71828183$

$N = 500$ 时, $e = 2.71828183$

而一般 $e = 2.71828$ 已够用了。

7. 考虑通用合理

为保证程序的通用性和逻辑上的合理性, 在程序编制过程中应考虑以下几点:

- 安排判断输入数据是否正确的语句;
- 利用终止标志来终止某一过程或停机;
- 对使用某一程序段来完成重复计算的工作, 应注意正确的调用和返回;
- 对多次重新运行的程序, 应保证任何情况下都正确无误;
- 有时程序中应考虑可能出错的自动处理。

例如, 求任意数的阶乘, 即求 $N!$

分析: N 只能是正整数 (包括 0)

N 不能是负数和小数, 见程序 No.12 (机型 PC-1500):

```
5:REM NO.12
10: CLEAR
20:F=1
30: INPUT N
40: IF N<0 OR N<>
    INT (N) THEN 90
45: IF N=999 THEN 1
    00
```

```

50:FOR I=1 TO N
60:F=F*I
70:NEXT I
80:LPRINT N;"!=";
  F:GOTO 10
90:LPRINT "N=";N;
  " IS DATA ERRO
  R!":GOTO 10
100:END

```

说明：40句就是分辨输入数据正误的判断语句，90句指出输入数据错误并正确返回（这里必须返回到10句，而不是30句），45句是安排的终止语句（ $N = 999$ 是任取的，它应远离要计算的数）。

下面是程序运行的部份实例：

```

      8!= 40320
N=-11 IS DATA ERRO
R!
      0!= 1
N= 1.32 IS DATA ER
ROR!

```

8. 防止程序出错

程序出错时，计算机能够显示错误的性质，指出错误的行号，并自动停机等待操作者处理。这不仅反映了计算机作为“电脑”的本领，而且也体现了BASIC语言能够实现“人机对话”的特点。

如果我们事先预料到程序可能出错，且这种错误并不影响下面程序的计算，而我们又希望程序能继续运行，不要停

机待命,由机器自动处理这种错误并继续工作,这时我们可以使用出错转移语句: ON ERROR GOTO <表达式或行号>。它的功能是,一旦程序出错,计算机自动转到GOTO后面所指向的标号上去。下面是PC-1500机应用的一个实例。

如求: $Y = 1/(I-20)/(I-40)/(I-60)$

$I = 1, 2, 3, \dots, 79, 80$

见程序No.13:

```
5:REM NO.13
10:ON ERROR GOTO
  70
20:FOR I=1 TO 80
30:Y=1/(I-20)/(I-
  40)/(I-60)
40:PAUSE I;"***";
  Y
50:NEXT I
60:END
70:BEEP 3:GOTO 50
```

本题中,当 $I = 20, 40, 60$ 时, $Y \rightarrow \infty$, 计算机会发生溢出而出错。作如上处理后,机器发出信号鸣笛三声,然后转 50 句,继续执行程序,不断给出结果而不会停机待命。

9. 划分功能模块

根据总体需要和系统配置的可能,将程序按功能划分成几个功能模块,这样做好处很多,一是结构清楚,层次分明;二是功能明确,便于移植;三是分头编写,缩短周期。这是在大型程序设计中经常使用的方法。

为便于理解，下面仅通过一个简单例子来说明模块化结构的应用。

例如，有一批学生参加计算机培训，现假定由计算机自动出题并评分，对于经考试成绩及格的，发给结业证书；不及格者给予一次补考机会，若补考仍不及格继续参加下期培训。

我们可以事先编好考试模块，补考模块、发证书模块等，然后按下述框图连接，这样，检查成绩的程序一目了然。这样做，使得整个程序阅读、理解和调试极其方便，而且可以随意修改，如图 2 所示。

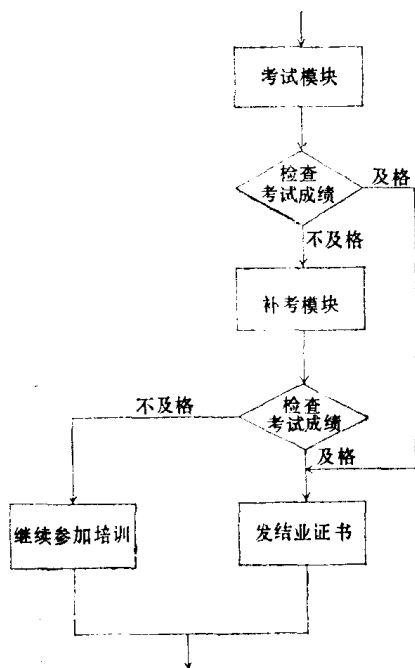


图 2

10. 讲究设计技巧

对初学编程的人来讲，程序设计入门并不难，但是要编制一个质量较高的程序，并非是一件轻而易举的事。因为要想编好一个程序，仅仅靠了解计算机语言的基本语句是很不够的，同时，初编程的人掌握现成的方法太少，技巧不多，再加上不很熟悉规则，往往出错。

程序设计是一项技术性很强的工作，是模仿性和创造性的统一。程序设计又是一项极其复杂而又十分灵活的工作，它需要人们不断探索，不断创造。

这里作为全书的引导，首先向读者介绍程序设计的简单技巧，本书中将有相当大的篇幅详细阐述程序设计技巧的各个方面。

例如，求 10, 20, 30, 40, 50 的平均值。

这是一个非常简单的数值计算问题，编制方法可以有 10 多种，初学者常常用赋值语句，键盘输入语句，读数语句来编，例如：

①

```
10 LET A = 10
20 LET B = 20
30 LET C = 30
40 LET D = 40
50 LET E = 50
60 LET M = (A + B + C + D + E) / 5
70 PRINT M
80 END
```

②

```
10 INPUT A,B,C,D,E
20 M = (A + B + C + D + E) / 5
30 PRINT M
40 END
```

③

```
10 READ A,B,C,D,E
20 M = (A + B + C + D + E) / 5
30 PRINT M
40 DATA 10,20,30,40,50
50 END
```

当然更多的人用循环语句来编，例如：

④

```
10 FOR I = 1 TO 5
20 READ X
30 S = S + X
40 NEXT I
50 PRINT S / 5
60 DATA 10,20,30,40,50
70 END
```

以上程序的正确性是不用怀疑的。但是仔细推敲一下，还有更好的编制方法，今略举一、二，读者可以自行鉴别其优劣。

⑤

```
10 FOR I = 1 TO 5
20 S = S + 10 * I
30 NEXT I
40 PRINT S / 5
50 END
```

③

```
10  FOR I = 10 TO 50 STEP 10
20  S = S + I
30  NEXT I
40  PRINT S / 5
50  END
```

说明：若用 PC-1500 机编制上述循环程序，在循环前要加 $S=0$ 或 CLEAR 语句，而用 APPLE-II 机可以省略。同时 END 语句可以不要，因为机器按行号顺序执行，没有行号及语句，机器会自动结束。此外，NEXT 后面的变量 I 可以省去，运行速度也加快了，这样，下面的一行程序，就是一个优化程序：

```
10  FOR I = 10 TO 50 STEP 10:S =
    S + I: NEXT : PRINT S / 5
```

综上所述，我们从十个方面介绍了编程的一般步骤和方法，实际上也并不一定按上述模式进行，如题目开始就给出一个计算公式，则建立数学模型一步可省去；若问题比较简单，绘制框图的工作也可以不做。

三、程序连接的方法

一个程序中各语句如何连接，两个不同的程序如何合并，主程序和子程序如何搭配，以及几个文件如何联用等等，都属于程序或文件的连接问题。其连接方法也是多种多样的。

1. 如何安排总控

一个程序中，如果在程序的开头安排了总控，那么，人们在操纵机器、支配程序、选择功能方面，就有了更多的灵活与方便。安排总控的方法有以下几种：

(1) 用选择转向语句 ON/GOTO 例如：

```
10 INPUT X
20 ON X GOTO 50,60,95
30 GOTO 10
50 GOSUB 100:GOTO 10
60 GOSUB 200:GOTO 10
95 END
100 REM SUB-(1)
:      :
:      :
:      :
:      :
180 RETURN
200 REM SUB-(2)
```

```

:      :
:      :
:      :
290 RETURN

```

上述程序执行过程如下:

$X = 1$: $20 \rightarrow 50 \rightarrow 100 \rightarrow \dots \rightarrow 180 \rightarrow 10$

$X = 2$: $20 \rightarrow 60 \rightarrow 200 \rightarrow \dots \rightarrow 290 \rightarrow 10$

$X = 3$: $20 \rightarrow 95 \rightarrow \text{END}$

说明: ON/GOTO 是一种控制转向语句, 通常也称开关语句, 事实上它的功能就像一个多路开关, 在程序中用来实现总控。当 $X = i$ 时, 转向第 i 个语句标号, 本题 $i = 1, 2, 3$ 。当 $X < 0$ 或者 $X > N$ (本题 $N = 3$) 时, 相当于控制转向语句不存在, 而执行下一语句, 即 30 句, 又返回总控。

X 可以是常数, 也可以是表达式, 如输入一实数 ($K = -1, 0, 1$) 并求 $F(K)$ 之值, $F(K)$ 定义如下式所示:

$$F(K) = \begin{cases} 2K + 3 & K = -1 \\ 0 & K = 0 \\ 2K - 5 & K = 1 \end{cases}$$

程序:

```

10  INPUT K
20  ON K + 2 GOTO 30,40,50: END
30  F = 2 * K + 3: GOTO 60
40  F = 0: GOTO 60
50  F = 2 * K - 5
60  PRINT "F=";F

```

(2) 用选择转子程序语句 ON/GOSUB 例如:

```
10 INPUT X
15 IF X<=0 THEN END
20 ON X GOSUB 100,200,300,500
30 GOTO 10
100 REM SUB-(1)
:      :
:      :
:      :
:      :
190 RETURN
:      :
:      :
:      :
:      :
500 REM SUB-(4)
:      :
:      :
:      :
:      :
800 RETURN
```

说明: ON/GOSUB 也是开关语句, 它比 ON/GOTO 语句更简捷, 也常用在程序的连接和控制上, 并可省去不少 GOSUB (标号), GOTO(标号)的语句。ON/GOTO 语句, 不具备 ON/GOSUB 语句所具有的记住返回位置的功能。

2. 单一程序的连接方法

(1) 用 INKEY 例如:


```

10: A$=INKEY$
20: IF A$="A" THEN
    "A"
30: IF A$="B" THEN
    "B"
40: IF A$="C" THEN
    80
50: GOTO 10
60: "A":FOR I=1TO
    5:LPRINT I;;
    NEXT I:LPRINT
    :GOTO 10
70: "B":LPRINT "MY
    NAME IS JOHN
    SMITH":GOTO 10
80: END

```

```

1 2 3 4 5
MY NAME IS JOHN SM
ITH

```

说明：以上程序适用于 PC-1500，APPLE-II 没有 INKEY\$ 键，但可以用 GET A\$ 代替。

(2) 开关语句放在循环中 例如：

```

10 FOR I= 1 TO 10
20 ON I GOSUB 100,150,.....,800
30 GOTO 950
100 REM SUB-ONE

```

```

:      :
:      :
:      :
:      :
140 RETURN
:      :
:      :
:      :
:      :
800 REM SUB-TEN
:      :
:      :
:      :
:      :
900 RETURN
950 NEXT I
990 END

```

说明：本程序的连接方法，不用键盘输入语句 INPUT，而是将开关控制语句放在循环程序中，由机器自动完成。

(3) 巧用逻辑相关表达式 逻辑相关符号（如<，>=，…，等等）是表示两个或更多个常量、变量、表达式的大小关系的，而在判断语句中，这样的关系式则表示满足条件是“1”，反之是“0”。

下面是一个成绩分档的例子。

60分以下，D等。

60—76分，C等。

77—89分，B等。

90分以上，A等。

ULIST

```

10 INPUT "SCORE: "; N
20 X = 1 + (N > = 60) + (N > =
    77) + (N > = 90)
30 ON X GOTO 40, 50, 60, 70
40 PRINT "D": END
50 PRINT "C": END
60 PRINT "B": END
70 PRINT "A": END

```

说明：程序中的 20 句是个逻辑关系表达式，使用该句后，程序比较简洁。否则不用 20 句则要加：

```

15 IF N < 60 THEN LET X = 1
20 IF N < = 76 THEN LET X = 2
25 IF N < = 89 THEN LET X = 3
28 IF N < = 100 THEN LET X = 4

```

故程序显得比较冗长，占用内存也多。

(4) 用逻辑运算符号 这里逻辑运算符号主要指 AND、OR、NOT。例如：计算如下函数值

$$f(x) = \begin{cases} 1 & -1 \leq x < 0 \\ x & 0 \leq x < 1 \\ 2x^2 & 1 \leq x < 2 \\ 3x^3 & 2 \leq x < 3 \end{cases}$$

程序：

ULIST

```
10 INPUT X
20 IF X < - 1 OR X > = 3 THEN
    PRINT "WU ZHI": END
30 ON X + 2 GOTO 50,60,70,80
50 PRINT 1: GOTO 10
60 PRINT X: GOTO 10
70 PRINT 2 * X ^ 2: GOTO 10
80 PRINT 3 * X ^ 3: GOTO 10
```

说明：程序中的 20 句，就起着承前启后的作用， X 值在函数的定义域内，转向 30 句控制语句及相应的结果，而 X 值不在定义域内则无解并结束。

3. 两个不同程序的连接方法

(1) 用 EXEC 命令

在使用微机和调试程序的过程中，常发生两个不同 BASIC 程序相互连接的问题，巧用 EXEC 命令，可以方便地得到解决。

设内存中已有一个 BASIC 程序，为简单计，仅有二行：

```
20 PRINT "BBBB"
30 PRINT "CCCC"
```

实际上，它可以是一个我们感兴趣的实用程序，或者是一个通用的子程序段。

为了使上述有用程序，随时和经常调用，以文本文件形式贮存在磁盘上，文件名叫 LISTING。

方法是：键入下述辅助程序，取名为 CAPTURE.

ULIST

```
1  REM  CAPTURE
2  D$ =  CHR$ (4)
3  PRINT D$; "OPEN LISTING"
4  PRINT D$; "WRITE LISTING"
5  POKE 33,30
6  LIST 20,30
7  PRINT D$; "CLOSE LISTING"
8  TEXT : END
20  PRINT "BBBB"
30  PRINT "CCCC"
```

然后将内存中已有的程序（本例是 20，30 行）和 刚键入的文件（CAPTURE），一起存入磁盘，即 SAVE CAPTURE ↵

这样，我们在内存中建立了 CAPTURE 程序、RUN CAPTURE ↵ 后，用 CATALOG（或 CTRL-4）命令检查磁盘目录，立即可以发现增加了一个 LISTING 的文件，其文件类别标志是 T，表示 LISTING 文件内容是文字资料。

注意：一定要将文件名为 CAPTURE 的程序存入磁盘，否则 RUN CAPTURE 后不会建立 LISTING 的 T 文件；同时，存盘后一定要运行，否则磁盘中只有 CAPTURE 的 A 文件，而没有 LISTING 的 T 文件。

为了实现与另一 BASIC 程序连接，可以先清内存（NEW ↵），然后装入被连接的程序，为简单计，设有程序：

```

10 PRINT "AAAA"
40 PRINT "DDDD"
50 PRINT "EEEE"
60 PRINT "END": END

```

以上程序可以是当前从键盘输入，也可以是原来存在磁盘中的程序。

最后，只要执行 EXEC LISTING 命令，即可完成两个程序的连接。

```

EXEC LISTING

```

```

U

```

```

U

```

```

U

```

```

LIST

```

```

10 PRINT "AAAA"
20 PRINT "BBBB"
30 PRINT "CCCC"
40 PRINT "DDDD"
50 PRINT "EEEE"
60 PRINT "END": END

```

U（此时 APPLE-Ⅱ 的控制权又返回到键盘）。

通过上述实例，可以看出 EXEC 命令执行时，把人 对 APPLE-Ⅱ 的控制权转移到一个文字资料文件内，而在此文件被执行后，对 APPLE-Ⅱ 的控制权又返回到控制台（键

盘)。

综上所述, 实现两个 BASIC 程序连接的步骤如下:

- 先装入要连接的程序 (20—30行);
- 键入辅助程序 (1—8行);
- 将 (20—30行) 和 (1—8行) 中的程序存入磁盘 (取文件名为 CAPTURE);
- 运行上述程序, 即生成 LISTING 文本文件 (即要连接的程序 20, 30 行以 T 文件形式存入磁盘);
- 若和其它程序连接, 先清内存, 再键入其它程序 (如 10, 40, 50, 60 行), 或调入已存在磁盘中的程序;
- 执行 EXEC LISTING 命令, 即完成两个 BASIC 程序的连接 (20, 30, 和 10, 40, 50, 60 行)。

最后再说明两点:

① 辅助程序中第 6 行 LIST 后面的行号, 应是要连接的第一个 BASIC 程序的起始行号和最后行号。

② 被连接的两个 BASIC 程序, 各自行号不应重复。

(2) 用 RENUMBER 程序 用户在编制一个较大的系统软件时, 可以有几个人分别编写, 然后再连接成一个完整的程序。不难想象, 用人工合并程序的工作量是很大的, 利用 DOS 3.3 操作系统中的 RENUMBER 公用程序, 将会给程序的合并带来很大方便。

为方便计, 设有两个 BASIC 程序, 它们分别是 №14 和 №15, 且两个程序行号不重合。即程序 №14:

ULIST

```
10 REM NO.14
20 PRINT "ABCD"
30 PRINT "1234"
```

程序No15:

ULIST

```
80 REM NO.15
90 PRINT "BASIC"
95 PRINT "4321"
99 END
```

现在要将上述两个程序合并，具体操作如下：

- 引导 DOS 3.3
- RUN RENUMBER↵
- 取出 DOS 3.3，放入要合并的两个程序（No14和No15）的磁盘；
- 键入 LOAD No14↵
- 键入 &H↵
- 键入 LOAD No15↵
- 键入 &M↵（此时即已实现了两个程序的合并，它们仍在内存中，给它们取一个文件名，如 No16，存入磁盘）；
- 清内存（NEW↵）
- LOAD No16↵

• LIST↵

LIST

```
10 REM NO.14
20 PRINT "ABCD"
30 PRINT "1234"
80 REM NO.15
90 PRINT "BASIC"
95 PRINT "4321"
99 END
```

注意：两个被合并的程序，行号不应重叠，否则按重订行号的方法处理。

4. 重订行号的方法

由于工作上的需要或当程序较长时，可以由几个人分别编制各程序段，然后再将大家的工作合并成一个完整的程序。把多人分头编制的程序合并成一个程序时，应该坚决防止行号重叠的现象，否则就会出现意想不到的后果，使程序无法执行。因此，需要对各人编写的程序各语句的行号重新订正，这是一个很费时的工作。而利用 APPLE- II 微机主磁盘中的 RENUMBER 程序，可以方便地完成重订程序中各语句行号的工作，其操作要领及实例如下：

- 引号 DOS 3.3 主磁盘
- RUN RENUMBER 程序
- 把需要重订行号的程序调入内存
- 打入 &F <xxx> , I<xxx>↵

即可完成行号的重订工作。

例如：有一个文件名为 №17 的程序：

ULIST

```
1  REM  NO.17
2  D$ =  CHR$ (4)
3  PRINT D$;"OPEN LISTING"
4  PRINT D$;"WRITE LISTING"
5  POKE 33,30
6  LIST 20,30
7  PRINT D$;"CLOSE LISTING"
8  TEXT : END
20  PRINT "BBBB"
30  PRINT "CCCC"
```

现在需要重订行号，从 100 号开始，相邻语句的行号间隔为 50。步骤如下：

- SAVE №17 ↵
- 引导 DOS 3.3
- RUN RENUMBER ↵
- LOAD №17 ↵
- 键入 & F100, 150 ↵
- LIST ↵

从屏幕上可以看到以下结果：

ULIST

```
100 REM NO.17
150 D$ = CHR$ (4)
200 PRINT D$;"OPEN LISTING"
250 PRINT D$;"WRITE LISTING"
300 POKE 33,30
350 LIST 20,30
400 PRINT D$;"CLOSE LISTING"
450 TEXT : END
500 PRINT "BBBB"
550 PRINT "CCCC"
```

四、节省内存和提高速度的方法

我们曾经指出，一个好的程序除了要求结果正确，还希望占用内存少和运行速度快。在选择各种各样解题方法时，应该选择那些既能节省机时又能节省存贮空间的办法，因为它们是进行高质量程序设计的重要指标。

1. 程序存贮的基本知识

在具体讨论节约内存的措施前，必须了解一下语句和程序的存贮情况。

程序是由程序行组成的，每一行语句存贮情况如下：1个语句的行号占用2个字节，语句长度占用2个字节，回车占用1个字节。每一句行号和回车之间称之为文本，文本中每个字符占用1个字节，每个命令也占用1个字节。注意这里仅指语句中各字符的代码所占用的字节数，并不包括变量所占的字节数。

按照上述存贮分配，用几种方法来检查语句或程序的存贮情况。

例如，两个简单的赋值语句：

```
10 LET A = 32
20 LET D = PEEK (105) + 256 * PEEK
    (106)
```

试检查一下它们占用的字节数。

(1) 根据 1 个行号占用 2 个字节 语句长度占用 2 个字节, 1 个回车占用 1 个字节, 因此上述两行不包括从命令开始的文本部份, 每行各占 5 个字节, 另外 1 个指令(如 LET, PEEK) 占 1 个字节, 1 个字符(如 A, =, 3, 2, * 等) 占 1 个字节, 所以 10 句的文本部份占 5 个字节, 20 句的文本部份占 20 个字节, 这样, 两个赋值语句共占 35 个字节。

(2) 用 FRE 语句 检查方法如下:

```
UPRINT FRE(1)
-29188
```

```
U10 LET A=2
```

```
U20 LET D=PEEK(105)+256*PEEK(106)
```

```
UPRINT FRE(1)
-29223
```

可以看出, 在没有键入两个赋值语句前, 用户的可用字节数为 $65536 - 29188 = 36348$ 。

在键入两个赋值语句后, 用户的可用字节数就减少到
 $65536 - 29223 = 36313$

利用这一检查, 就可以了解到两条赋值语句所占用的字节数为

$$36348 - 36313 = 35$$

这个结果和方法(1)一致。

注意: 语句 FRE 后面括号中的数字是无关紧要的。

当然，也不一定要用 65536 处理，可用

$$|-29223| - |-29188| = 35$$

(3) 用一个程序来检查 由于 APPLE-II 机的程序存储区是从 \$0800 地址开始的 (即 10 进制的 2048)，所以下面的程序可以用来检查 10 句和 20 句在存储区中 语句 的 存储方式，并可了解它们占用内存的字节数。

ULIST

```
10 LET A = 32
20 LET D = PEEK (105) + 256 * PEEK
   (106)
30 REM STATEMENT STORAGE
40 FOR K = 2048 TO 2083
50 LET X = PEEK (K)
60 PRINT X; " ";
70 NEXT K
80 END
```

运行上述程序，得到如下结果：

URUN

```
0 11 8 10 0 170 65 208 51 50 0 36 8 20 0
170 68 208 226 40 49 48 53 41 200 50 53
54 202 226 40 49 48 54 41 0
```

这一连串的数字都是 10 进制数，一下不容易看清它们的含义，实际上包括 5 个方面的内容。

• 浮点 BASIC 保留字的内部代码，如 170 就是 16 进制 AA，它是 LET 的编码。又如 226 就是 16 进制 E2，它

是 PEEK 的编码。而 202 对应 \$CA，是保留字 * 的代码。

- 机内 ASCII 码，如 65 (10 进制的 ASCII 码)，对应的 ASCII 字符是 A。而 41 对应的 ASCII 字符是)等等。

- 行号，如 10 就是行号 10，20 就是行号 20 等。

- 行号前的两个单元存贮着下句地址，地址的低位在前，地址的高位在后。如上述结果中 10 前面的二组数字 11 和 8，是地址。如语句 10 的下句地址为：

$$256 * 8 + 11 = 2059$$

即下一语句 20 从 2059 开始。

同理，地址 2059 和 2060 就是语句 20 句的下句地址，即

$$256 * 8 + 36 = 2084$$

换句话说，语句 10 和 20 占用了从地址 2048 (实际上是 2049) 到 2083 这个存贮区共 36 个字节 (实际上是 35 个字节)。

这个结果和上面分析的一致。

- 语句结尾，每个语句以全 0 字节结尾。

顺便说明的是，上述程序可以了解程序或语句的存贮方式及语句长度，同时，也可用该程序了解机器所用的关键字代码。方法是将行号 10 和 20 改成其它语句。

对照上述程序运行结果，以及浮点 BASIC 保留字代码和机内 ASCII 码，不难将上述结果按存贮地址排列起来，今给出部份结果，其余部份请有兴趣的读者自行完成，见表 4.1。

表 4.1

2048	0	
	11	下句地址为
2050	8	2059
	10	行号 10
2052	0	
	170	LET
2054	65	A
	208	=
2056	51	3
	50	2
2058	0	
	36	下句地址为
2060	8	2084
	20	行号 20

2. 节省内存的措施

了解上述存贮概念后，就很容易找到节省内存的方法，具体措施有：

(1) 采用组合语句 前面已经指出，每个语句行必需有 5 个字节的说明性存贮。它们是 2 个字节的行号、2 个字节的下句地址和 1 个字节的结束符。因此，很容易想到在 1 个行号下安排多个语句。行号少了，省出了占用的字节，也省出了处理行号的时间，从而节省了内存。

例如，有一段程序如下：

```
10 LET A = 7
20 LET B = 5
30 LET C = 3
```


该程序共三行，占用内存 27 个字节，现在用一条组合语句来代替。

```
10 LET A = 7: LET B = 5: LET C = 3
```

这里增加了 2 个语句分隔符“:”，所以比上面三行程序节约 8 个字节存贮量。

按照扩展 BASIC 规定，还可以这样写：

```
10 A = 7: B = 5: C = 3
```

这样又省去了 3 个字节。

因此，在一个长程序中，使用多语句行可以节约不少存贮量。但是，应该注意，一个程序行允许的字符数是有限的，APPLE-Ⅱ机最多不超过 239 个字符。这样做的不足之处是，程序变得不够清晰，难以阅读和理解，还会造成编辑和修改的困难。

(2) 删掉所有的 REM 语句 注释语句 REM 往往包含很多字符，它常常是告诉读者怎样使用程序，各部份程序内容提示、操作、数据输入、打印输出等信息。有 1 个字符就要 1 个字节来存贮。因此，这个语句是很浪费存贮空间的。REM 是非执行语句，在程序中不起作用，将它取消不影响程序功能，但却节约了很多存贮量。

例如，删除前

```
ULIST
```

```
10 LET A = 32  
20 LET D = PEEK (105) + 256 * PEEK  
    (106)  
30 REM STATEMENT STORAGE
```

U?FRE(1)
-29247

取消后

ULIST

```
10 LET A = 32
20 LET D = PEEK (105) + 256 * PEEK
      (106)
```

U?FRE(1)
-29223

共节省 24 个字节的存储量 (REM语句后每一个空格也
占用 1 个字节)。

又如有以下程序:

```
10 REM                      定义时间矩阵
20 DIM T(40, 40)
```

可改为:

```
10 DIM T(40, 40):REM      定义时间矩阵
```

或改为

```
10 DIM T(40,40)          定义时间矩阵
```

前者适用APPLE-Ⅱ, 后者适用 IBM PC, 二者都是使
用简要的注释, 并采用组合语句方式, 以节省内存。

但若改为

```
10 REM      定义时间矩阵:DIM T(40,40)
```

则语法上虽没有错误, 但数组 T 并没有开辟, 因为

REM 是非执行语句，冒号后面的内容计算机并未访问。

同采用组合语句一样，取消 REM 语句，使程序阅读、使用变得困难。

(3) 省去程序最后的 END 语句 这一点也是比较自然的，当 END 放在程序结尾时，有或无均一样，机器扫描到没有行号时，会自动结束。这一点和 BASIC 规定中有关 END 的说明不一样，但实际上却是可行的。

(4) 不要随意扩大数组 为了保证程序的正确执行和各种信息的正确存贮，有时候需要开辟一定的数组，太少了不够存贮，机器指出 BAD SUBSCRIPT 的出错信息；太大了浪费内存空间。每多用 1 个数组元素（下标变量），就多用 5 个字节内存。

另外，在数组中，尽可能用整数代替实数，前者每个元素要用 2 个字节，后者却要用 5 个字节，因此，对 100 个元素的数组，用整数就可以节约 300 个字节，这是很可观的。

(5) 用单个字母作变量名 用单个字母作变量名，名字越短越好，易读易改内存少，程序运行也快。

例如，

```
10 NUMBER = 1
```

改为：

```
10 N = 1
```

则可省用 5 个字节的存贮量。

(6) 尽量使用变量代替常数 变量所占的内存比常数省(变量的名字要短)。因此要尽量使用变量而少用常数。对于多次用到的常数可以先赋值给某一简单变量，而后再调用，这样所占内存少。

例如，在程序中要使用 10 次 3.1415926 这个常数，那么，可以使用语句

```
10 P = 3.1415926
```

然后，每一次使用该常数时，用 P 而不用 3.1415926，这样每次就节省了好几个字节，调用 10 次就节省了几十个字节。

另外，常用的变量，最好放在程序的最前面，以减少每次搜索时间，提高运行速度。

(7) 用 ON/GOTO 语句代替 IF/THEN 语句 在解决比较复杂的问题时，常常遇到多分支的情况，这样需要多次使用 IF/THEN 语句，这显然是繁琐的。而使用 ON/GOTO 语句，可以解决多个出口，既可节省内存，也为用户提供方便。

例如，

```
10 IF X = 1 THEN 100  
20 IF X = 2 THEN 200  
30 IF X = 3 THEN 300
```

可改为：

```
10 ON X GOTO 100,200,300
```

(8) 用 INPUT 语句代替 DATA 语句 在允许的情况下，尽量用 INPUT 语句代替 DATA 语句。

例如，

```

10  FOR I = 1 TO 5: READ A(I): NEXT
    I
20  DATA 4,3,2,7,1

```

可改为:

```

10  FOR I = 1 TO 5: INPUT A(I): NEXT

```

这一改动可以减少不少内存, 不过由于 INPUT 是键盘输入语句, 逐个送数, 时间要慢一点, 再次运行本程序, 又要一个一个送入数据。

另外, 对于改动后的语句, 还有两点需要说明, 其一是 NEXT 后面省去了循环变量 I , 在紫金-Ⅱ, APPLE-Ⅱ 机上都是允许的, 它可以提高运行速度 (原因见下面怎样提高程序的运行速度); 其二是对于下标变量不到 10 的, 可以不要数组说明语句 DIM, 这不仅符合 BASIC 规则, 而且在 APPLE-Ⅱ 等机器上是可执行的。但在 PC-1500 机上就不能执行, 只要用到下标变量, 不任其值大小, 一定要在程序的前头安排 DIM 语句。

(9) 不用重复给变量初始化 在用计数器 ($N = N + 1$) 和累加和 ($S = S + A$) 时, 常常要首先给变量赋以初值 0, 但对不同的机器, 处理方法是有所区别的。

例如, 对 APPLE-Ⅱ 机

```

10  X = 0: M = 0: N = 0

```

但这一语句可以不要, 实际上在执行 RUN 命令前, 对

未赋值的变量 X ， M ， N 均以 0 处理，而在执行 RUN 命令后，内存变量已置空，不用再重置，多次运行不会影响程序的正确性。

对 PC-1500 机，上述程序行则是必要的。否则，重复运行程序每次结果都不一样。另外，对 PC-1500 机来说，也可以用 CLEAR 语句来代替给某些变量（可以是多个变量）初始化，用以代替给变量一个一个的赋零值，这样处理，程序更加简洁，且节省了内存。

(10) 尽量采用子程序 对执行同样操作和重复使用的程序段，要编成子程序的形式，以减少程序行数，达到节省内存的目的。

(11) 经常注意使用 $X = FRE(0)$ 语句 清理存贮空间，把可用的存贮区还给用户。

(12) 不要任意提高变量精度 整型变量、单精度变量、双精度变量占用的字节数分别为 2、4、8。因此，不要任意提高精度，而应尽量采用整型变量。

3. 提高速度的方法

一个程序的优劣，其运行速度是个很重要的指标，下面是提高程序执行速度的几个常用方法。

(1) 尽量少用 GOTO 语句 如 GOTO 10000，机器对 GOTO 以后的标号，是逐一搜索整个程序，并从最低行号开始，一直到 10000 时才执行下续语句，并不是“立即”跳到 10000 句执行，这一点应特别注意。

(2) 尽量使用简单运算 计算机执行运算有所谓“优先权”问题，例如执行加法比做乘法快，而乘法又比除法和取

幂要快。因此，对以下语句：

```
10 A = B / 2
20 A = B * 3
30 A = B ^ 4
```

应改为：

```
10 A = B * 0.5
20 A = B + B + B
30 A = B * B * B * B
```

这里有二点要作说明，一是经上述改动后，内存增加了，可见在本例中，提高了速度，但增加了内存，它们是互相矛盾的；二是改乘方为连乘运算，不仅提高了速度，而且提高了精度，因为计算机做乘方运算是通过对数与指数运算来完成的（详见后面章节说明）。

（3）避免重复对表达式求值 对变量赋一个常量，要比变量赋另一个变量要快。

对以下语句：

```
10 A = 3 * C + E - Y
20 B = 3 * C + E + Z
30 D = (3 * C + E) / W
```

可改为：

```
10 X = 3 * C + E
20 A = X - Y
30 B = X + Z
40 D = X / W
```

细心的读者一定会发现，改动以后的语句，不仅速度加快，而且还节省了两个字节内存。

(4) 采用扩展 BASIC 写程序 例如：

```
100 IF X=-1 THEN 150
    :      :
    :      :
    :      :
    :      :
150 PRINT S
160 END
```

可改为：

```
100 IF X = - 1 THEN PRINT S: END
```

(5) 提高循环程序的执行速度 关于这一点的详细说明，我们将在有关章节阐述，这里仅从原则上提出如下措施：

- 减少循环终值和初值之差；
- 增大步长减少循环次数；
- 减少循环层数；
- 把计算放在循环体外执行；
- 减少或巧安排循环体内语句；
- 简化表达式，采用低级运算；
- 调整循环体语句次序，减少无用循环；
- 巧用字符串函数或逻辑运算符号。

(6) 其它提高速度的措施有：

- 尽量重复使用同一个变量名
- 重复使用的数据用变量代替
- 常用的变量在程序中应尽早出现
- 尽量避免用乘方运算
- 省去 NEXT 后面的变量

说明：上面仅就节省内存和提高速度两个方面，提出了一些简单可行的方法，其中还介绍了不少使用语句编程的技巧，但是，必须说明，上述各种方法绝不是最有效和最根本的方法。

事实上，上述各种方法，没有必要也不可能同时采用和兼顾，因为这些方法有时是相互矛盾相互制约的。有的提高了速度，却增加了内存；有的节省了存贮空间，但影响了程序的清晰程度，破坏了可读性，增加了阅读和修改的困难；有的虽减少了内存，但延长了程序的执行时间。所以，我们应该根据实际情况区别对待。最好选择那些既能节省机时又能少用存贮空间的办法，而当两者不能兼顾时，应根据实际情况偏重一方，如经过估算，内存比较紧张，那么就多采用一些节省内存的措施。如果是速度太慢，则应从提高速度上多下功夫。

最后，强调两点：

- 最好的程序设计，取决于最佳的解题方法和设计思想。
- 解决节省内存，提高速度的根本措施，是使用机器语言子程序。事实上，不少有用的软件大都用机器语言来编制。

4. 使用机器语言子程序

(1) 节省内存的一个实例 前面谈到的节省内存的措施, 实际上都是很有限的, 要更进一步节省存储量, 最有效的措施是使用机器语言子程序。为了说明机器语言子程序的这一突出优点, 请看一个实例。

假若要求在显示屏上打出 26 个英文字母, 可采用下述我们比较熟悉的 BASIC 语言编程方法。

见程序 No.18:

```
5  REM  NO.18
10  REM  TYPE 26 CHAREACTERS
15  REM  ASCII CODE OF A IS 65
20  LET A = 65
30  PRINT CHR$(A);
40  LET A = A + 1
45  REM  END TYPE AT CHAREACTER Z
    (CODE 90)
50  IF A < = 90 THEN GOTO 30
60  END
```

U?FRE(1)

-29347

这样一个简单的 BASIC 程序, 很容易阅读。先取 A 的 ASCII 码 65 赋于变量 A, 然后打印出以 A 值为代码的字符 A; 40 句是个计数器, 用来控制循环, 以后每次循环使 A 加 1, 并打印出下一个字符, 只要计数的 A 值不大于 90, 一直重复下去, 最终打印出最后一个字符 Z (它的 ASCII 码值是 90)。

用 FRE 语句, 可以检查出这个简单的 BASIC 程序, 占用了 157 个字节的存储量 (未键入上述程序时 PRINT FRE(1) 为 -29188)。

如果删除所有的 REM 语句, 那么, 就可以少用 100 个左右字节内存, 而减少到 53 个字节的存储量, 例如:

```
20 LET A = 65
30 PRINT CHR$(A);
40 LET A = A + 1
50 IF A < = 90 THEN GOTO 30
60 END
```

```
U?FRE(1)
-29241
```

同样这个程序, 如果用机器代码来编, 只要用 13 个字节的存储量。存储在首地址为 300 的地址中。

UCALL-151

*300.30C

0300- A9 C1 20 ED FD 18 69 01

0308- C9 D8 D0 F6 60

对应的汇编语言是:

***300.30CL**

0300-	A9 C1	LDA	##C1
0302-	20 ED FD	JSR	\$FDED
0305-	18	CLC	
0306-	69 01	ADC	##01
0308-	C9 D8	CMP	##D8
030A-	D0 F6	BNE	\$0302
030C-	60	RTS	

如果在 BASIC 状态, 键入 CALL 768↵, 立即可以在屏幕上看到以下信息:

ÜCALL768

ABCDEFGHIJKLMNOPQRSTUVWXYZ

这个结果, 和上述 BASIC 程序运行的结果是完全一致的。

CALL 后面的 768 是十进制数, 它等效于十六进制 300。

从这个例子明显看到, 将子程序编成机器语言形式, 可以有效地节省存贮空间。

(2) 提高速度的一个实例 例如, 两种语言排序的比较, 用汇编语言编制的程序比用 BASIC 语言编写的程序, 在执行速度上要快的多。

下面我们通过枚举法排序来说明这个问题。

程序 No.19 是用 BASIC 语言写的, 数组 A(N) 是用来存放被排序后的数据, 数据的个数 N 由键盘输入, 50 句用来

产生随机数并打印, 100 句就是大家比较熟悉的枚举法排序, 它和 110 句一起构成排序并打印输出。

```
5  REM  NO.19
10  INPUT N: DIM A(N)
50  FOR I = 1 TO N:A(I) = INT ( RND
    (1) * 100): PRINT A(I);" ";
    : NEXT I
80  PRINT : PRINT
100 FOR I = 1 TO N - 1: FOR J =
    I + 1 TO N: IF A(I) > A(J) THEN
    T = A(I):A(I) = A(J):A(J) =
    T
110 NEXT : NEXT : PRINT : FOR I =
    1 TO N: PRINT A(I);" ";: NEXT
```

以上程序运行后, 如果 N 取 100, 则在屏幕上将要运行 1 分钟左右。

如果用汇编语言来编制枚举法排序程序, 将发现其运行速度是很快的, 同样 100 个数从产生到排序完毕仅几秒钟。

机器语言的子程序为:

#0300.032D.

```
0300- A5 06 B5 07 C6 07 A2 00
0308- E8 8A A8 C8 BD 00 60 D9
0310- 00 60 B0 0B 98 C5 06 D0
0318- F2 8A C5 07 D0 EA 60 48
0320- B9 00 60 9D 00 60 68 99
0328- 00 60 4C 14 03 00
```

以上程序取名 №20，把它存入磁盘备用。

此外，我们再编写一个 BASIC 程序，用来调用机器语言排序的子程序，取名为 №21：

ULIST

```
5 REM NO.21
10 POKE 6,100: FOR I = 1 TO 100:
    A = INT ( RND (1) * 255): PRINT
    A;" "; POKE 24576 + I,A: NEXT
    I: PRINT
40 PRINT : PRINT : PRINT
50 CALL 768
80 FOR I = 1 TO 100: PRINT PEEK
    (24576 + I);" ";: NEXT I
```

现在可以在屏幕上比较两种语言编制的排序程序，在速度上究竟有多大差别，操作如下：

- 装入机器语言的排序程序，BLOAD №20 ↵

- RUN №19 ↵

出现？后，键入 100 ↵

观察排序的执行速度

- 清内存，NEW ↵

- RUN №21 ↵

再观察排序速度。应该注意的是，上述操作必须先装入机器语言子程序 №20，这样在程序 №21 中，CALL 768 才能正确调用，读者可以思考这是为什么？如果不调入程序 №20，又会是什么结果？不言而喻，CALL 后面的 768，应是程序 №20 的入口地址 \$0300。

五、编制程序应注意的问题

为了使初学者尽快地学会程序编制方法，在这一章中重点介绍编制程序时应注意的若干问题。在写法上，通过简单易懂且有一定典型性的程序实例，简要叙述编程方法和技巧的不同侧面。还列举编程时普遍发生的一些错误，让读者在比较中得到鉴别。另外，尽量选择一些好的例题，因为它们会给读者一些有益的启迪。

1. 关于变量的初始化问题

在用计数 ($N = N + 1$)，求和 ($S = S + A$)，及求阶乘 ($K = K * I$) 时，应注意给变量赋初值。

考察下面三个程序，指出有无错误。

① 求 $5!$

CLIST

```
10 FOR I = 1 TO 5
20 A = 1
30 A = A * I
40 NEXT A
50 PRINT "5!=";A
60 END
```

② 求 $T = \sum_{n=0}^{20} N$

ULIST

```
10 LET T = 1
20 FOR N = 0 TO 20
40 T = T + N
45 PRINT T
50 NEXT N
55 PRINT : PRINT
60 PRINT T
70 END
```

③ 求10, 20, 30, 40, 50的平均值

ULIST

```
20 S = 0
30 IF N > = 5 THEN PRINT S / N
   : END
40 INPUT A
50 S = S + A
60 N = N + 1
70 GOTO 30
```

①中有两个错误:

• FOR/NEXT不对应或不配对, 将40句NEXT后面的A改为I, 或干脆省去I更好。这样做可以提高速度, 因为NEXT I每循环一次, 都要从程序开头往下顺序查找, 一直找到与之对应的FOR语句才能执行, 而无I的NEXT语句, 则不需要进行上述寻找, 即直接与FOR语句构成循环。这种不带变量的NEXT语句, 在DATA语句大量读入数据时, 能明显提高速度。

• 程序的运行结果不是5!而是5, 应删去20句而改成5
A = 1。应该注意求阶乘的初始条件必须是1, 而且应该放在
循环之前, 否则出错。

②中的错误是10句, T = 1。求和, 求乘方和的初始化
条件一定是0, 或不给变量赋初值, 而改用10 CLEAR。对
于 APPLE- II 机本题的T变量可以不要初始化, 因为在 RUN
命令执行时, 内存变量已置空, 多次运行不会影响结果的正
确性。

③中的错误, 一下不容易看出来, 用 APPLE- II 机执行
每次结果均一样; 用PC-1500机执行, 各次结果不同, 原因
是N未清除, 不断累加, 计数愈来愈大; 用PC-1211机则不
执行, 因为60句中第一次N未赋值。所以本题最好作如下修
改, 20句变为: S = 0, N = 0, 这就给变量N也赋了初值。

2. 数组元素的下标要恰当

例如, 试分析下述程序的执行结果:

ULIST

```
10 DIM C(20)
20 FOR I = 11 TO 30
30 C(I) = I
40 PRINT C(I); " ";
50 NEXT I
60 PRINT
70 END
```

现给出两组答案, 你认为哪一个对?

① 11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30

② 11 12 13 14 15 16 17 18 19 20

第一种答案认为循环20次，每次把I值赋给C(I)，所以打印20个值。

第二种答案认为循环虽然进行20次，但因C数组元素的最大下标值是20，I值只能赋给C(11)，C(12)，…，C(19)，C(20)之中，计算机并没有开出C(21)，C(22)，…，C(30)的内存。

显然，第二种答案是正确的，请看运行结果：

RUN

11 12 13 14 15 16 17 18 19 20

3. 程序的正确性要经得起时间的考验

如果设计的程序用一次就失效了，这个程序的价值就很低。下面的程序十分有趣，它的正确性是随机的，即有时正确，有时错误，试分析产生错误的原因。

题目是这样的：

产生0—9的10个随机整数共三组，并能找出每组中的最大值。

下面给出用PC-1500机编制的一个程序：

```
5: CLEAR
10: FOR J=0 TO 2
12: COLOR J
15: COSUB 50
20: NEXT J
25: END
```

```

50:FOR I=1TO 10
55:X=INT (10*RND
    (0))
60:LPRINT X;" ";
65:IF XM<XLET XM=
    X
70:NEXT I
72:COLOR 3
74:LPRINT
75:LPRINT "XM=";X
    M
80:RETURN

```

请读者仔细分析程序，是否能找到其中的错误，并请键入机器，多次观察运行结果，检查程序的正确性。

最后，将发现这个程序的正确性确实是随机的。

原因在于当某次运行后XM中放了最大值9时，再次运行如果10个随机数均小于9，到75句打印出的最大值仍为9，这就出现错误。而当每组出现的最大值一个比一个大时，打印的结果又是正确的。

改正的办法是在75句和80句之间，加一句，如 78: XM = 0，则这个程序的正确性就能经得起时间的考验。

4. 防止陷入死循环

请分析以下两个程序，运行后出现什么情况。

①

```

10 A = 30
20 FOR I = 1 TO 50 STEP 2
30 A = A + 1
40 IF I = 9 THEN GOTO 10

```

```

50 NEXT I
60 PRINT "A=";A
70 END

```

```

② 10 GOSUB 100
   20 END
   100 PRINT A;" ";
   110 GOSUB 200
   120 PRINT A,B
   130 RETURN
   200 B = 7
   210 PRINT B;" ";
   220 GOSUB 100
   240 RETURN

```

上述两个程序都属于死循环，第一个程序永远跳不到60句，因而无打印结果，机器一直循环不息。第二个程序虽能打印出0 7 0 7 0 7……的结果，但毫无意义，机器也不会停止，除非强迫中断。

5. 矩阵输出注意打印语句的安排

例如，用双下标变量写一个程序，要求建立并打印除对角线（从左到右）元素为9，其余元素为0的 3×3 方阵。

下面给出一个程序，试分析打印格式是否满足题意要求。

```

10 DIM C(3,3)
20 FOR I = 1 TO 3
30 FOR J = 1 TO 3
40 IF I = J THEN 70
50 C(I,J) = 0
60 GOTO 80

```

```

70 C(I,J) = 9
80 PRINT C(I,J)
90 NEXT J
110 NEXT I
120 END

```

这个程序结构基本正确，唯打印输出不合要求，正确的是作两处改动，80句后加；即80 PRINT C(I,J)；添 100句 PRINT。两者缺一不可，否则打印出来的结果不是一行就是一列。

6. 妥善处理保留字

例如，试分析下述条件语句的正确性。

```
50 IF B>A THEN A=B
```

这条语句表面上看不出什么错误，逻辑关系也是正确的。但它确实是错误的，对不同微机来说，错误的情况又不一样。

当我们将上述语句键入 APPLE-II 或紫金-II 时，一经列表，屏幕上将显示出：

```
50 IF B > AT HENA = B
```

```
URUN
```

```
?SYNTAX ERROR IN 50
```

这个AT是BASIC语言保留字，尽管在输入程序时A与T之间以空格符间隔，但在APPLE SOFT BASIC语言中，并不把空格视为分隔符。也就是说，若THEN字左边的第一个

字母（不管中间是否有空格）是A的话，系统就会把A与THEN的T字母合并成AT，而造成错误。出现这种情况可用括号将A括起来。

而在PC-1500机中，上述情况则处理为……ATN……，ATN是PC-1500机的保留字。解决方法之一可用括号处理。

有趣的是，若将THEN，一个字符一个字符的键入机器，则可避免上述错误发生，反之若将THEN在键入PC-1500时，用T.输入，则出现上述错误。

此外，对PC-1500机来说，经上述处理后仍不能正确运行，并出现ERROR 1 IN 50的错误信息，问题是THEN后面不能省去LET，即应为：

```
50 IF B>(A) THEN LET A = B
```

或改为：

```
50 IF B>(A) LET A = B
```

7. 重新读数不要忘记 RESTORE

例如，求一组无规律的已知数的平均值和均方值（10个数放在DATA区中）。

计算公式为：

$$\text{平均值: } M = \left(\sum_{i=1}^n x_i \right) / N$$

$$\text{均方值: } C = \sqrt{\sum_{i=1}^n (M - x_i)^2} / N$$

程序：

```

5 M = 0: C = 0: N = 10
10 FOR I = 1 TO N
20 READ X
30 M = M + X
40 NEXT I
50 M = M / N: PRINT "M="; M
60 RESTORE
70 FOR I = 1 TO N
80 READ X
90 C = C + (M - X) * (M - X)
100 NEXT I
110 C = SQR (C) / N: PRINT "C=";
    C
120 DATA 1, -3, 2, 7, 13, 5, 16, 23, 0,
    -12
130 END

```

```

URUN
M=5.2
C=3.0258883

```

说明:

① $M = 0$, $C = 0$, 安排在循环外, 是为了求和、求平方和、求平均值、求均方值;

② $N = 10$, 也安排在循环之前, 这里是尽量使用变量来代替常量, 因为程序中有四次要用常数10, 这样做有二个好处, 既可节省内存, 又可加快执行速度;

③ 90句为什么用 $C = C + (M - X) * (M - X)$, 而不用 $C = C + (M - X) \wedge 2$, 这基于两方面的考虑, 首先计算机做

乘方是转换成对数和指数来运算的，这样速度慢，精度也不能保证，改用连乘会提高程序的运行速度，且比较精确；其次，由于 M 是平均值，若 X 值较大，则会出现负数的乘方，机器要出错，因为负数的乘方无意义，例如， $M=9$ ， $X=13$ ，则 $(M-X)\wedge 2=(9-13)\wedge 2=(-4)\wedge 2$ ，机器就不执行，改用连乘后全是正数，那么执行 110 句开平方也不会出现负数开方的问题；

④ 最后谈一下本程序中 60 句 RESTORE 的作用。如果没有 60 句，本程序的 80 句将无数可读，程序出错。

8. 能用简单变量的就不用下标变量

例如，设计一个方阵(5×5)，要求：

- 任意一个正方形对角线顶点元素的乘积均相等；
- 任意一个矩形对角线顶点元素的乘积均相等。

解法 1：用二维数组比较方便

```

10  DIM A(5,5)
20  FOR I = 1 TO 5
30  FOR J = 1 TO 5
40  A(I,J) = I * J
50  PRINT A(I,J); " ";
60  NEXT
70  PRINT
80  NEXT
90  END

```

```

RUN
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15

```



```
4 8 12 16 20
5 10 15 20 25
```

```
U?FRE(1)
-29481
```

解法2：用简单变量比较巧妙

```
ULIST
```

```
10 FOR I = 1 TO 5
20 FOR J = I TO 5 * I STEP I
30 PRINT J;" ";
40 NEXT
50 PRINT
60 NEXT
70 END
```

```
URUN
```

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

```
U?FRE(1)
-29264
```

两种解法结果一致。而解法2，不用开辟数组，程序简洁且节省内存。用PRINT FRE (1) 检查，后者比前者少用

217 个字节的存储空间，这是一个不小的量。因此，能用简单变量解题的，就不用下标变量。

9. 注意程序的清晰

例如，有 10 个数，已知前 5 个数分别为 1, 2, 3, 4, 5，而后五个数分别是它们前面 5 个数之和，试编程计算并打印输出这 10 个数。

解法 1：先用简单变量

ULIST

```
10 DATA 1,2,3,4,5
20 READ A,B,C,D,E
30 PRINT A;" ";B;" ";C;" ";D;" "
   ;E
40 FOR I = 6 TO 10
50 S = A + B + C + D + E
60 PRINT S;" ";
70 A = B
80 B = C
90 C = D
100 D = E
110 E = S
120 NEXT
```

URUN

```
1 2 3 4 5
15 29 56 109 214
```

U?FRE(1)

-29384

解法 2：再用下标变量

```
10 DIM X(10)
20 DATA 1,2,3,4,5
30 FOR I = 1 TO 5
40 READ X(I)
50 PRINT X(I); " ";
60 NEXT : PRINT
70 FOR I = 6 TO 10
80 X(I) = X(I - 1) + X(I - 2) + X
      (I - 3) + X(I - 4) + X(I - 5
      )
90 PRINT X(I); " ";
100 NEXT
```

ORUN

1 2 3 4 5

15 29 56 109 214

U?FRE(1)

-29404

比较上述两个程序，解法 2 所编程序简洁、清晰、易读性好。当然占用内存是多了一些。

此外，两个程序占用内存的情况，说明了这样一个问题，程序长（解法 1）内存不一定大；程序短（解法 2）内存不一定小。这主要看程序采用什么结构。一般讲程序愈长，占用内存愈多。本题是个例外，因为解法 2 用了数组，存储量增加。

10. 判断循环变量是否改变

下面是两道容易出错的循环程序例题，试写出它们的运行结果。

①

```
10 L = 3
20 FOR A = 1 TO L
30 PRINT A; " "
40 L = 5
50 NEXT A
60 PRINT L
70 END
```

② ULIST

```
10 FOR I = 1 TO 5
20 I = I + 1
30 PRINT TAB( 6); I; " "
40 NEXT I
50 END
```

程序①有两组结果，试判断哪一个对？

a. 1, 2, 3, 4, 5, 5

b. 1, 2, 3, 5

程序②有三种结果，哪一个对？

a. 2, 4, 6

b. 2, 4, 6, 8, 10

c. 2, 3, 4, 5, 6

显然，程序①的 b 结果是对的，因为循环体内的其它语句并没有改变循环变量 A，循环次数仍是 3 次，60 句打印的是 40 句的 L 之值不能改变循环终值。此题初学者

最易混淆。

程序②的④结果是正确的，因为循环体中循环变量 I 的值被改变了，循环次数不再是 5 次，而应该是 3 次。

11. 数值运算不要超过机器规定范围

例如，试问将数 300003 的百位数和万位数上的零换成什么数后，所得的数是 13 的倍数（只要求换的数相同）。

给出以下程序：

```
20:A=300003
30:FOR I=10100TO
    90900STEP 1010
    0
40:S=A+I
50:IF INT (S/13)<
    >S/13THEN 70
60:LPRINT S
70:NEXT I
80:END
```

这个程序从理论上分析是正确的，语法规则也没有错误，可是运行本程序没有结果，在 PC-1500 上给出 ERROR 19 IN 30 错误。

原因是循环变量的终值 90900 超过机器的规定范围，PC-1500 机的终值、步长的取值范围是 - 32768—32768。

若将上述程序稍加改动，即可完成本题要求，即：

```
20:A=300003
30:FOR I=1TO 9
40:S=A+I*10000+I*
    100
```

```

50: IF INT (S/13)=
    S/13 THEN
    LPRINT S
60: NEXT I
70: END

```

320203

12. 注意使用扩展BASIC语句

描述定义在区间 $(-10, 10)$ 函数 y 的值, 见图 3。

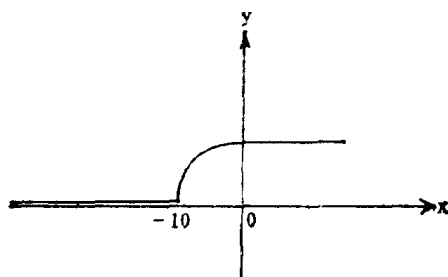


图 3

对这样的问题, 首先要建立数学模型。显然, 下述方程能够描述图 3。

$$y = \begin{cases} 0 & x \leq -10 \\ \sqrt{100 - x^2} & -10 < x < 0 \\ 10 & x \geq 0 \end{cases}$$

见程序№22:

```
5  REM  NO.22
10  FOR X = - 12 TO 10
20  IF X > = 0 THEN Y = 10: GOTO
    60
30  IF X < - 10 THEN Y = 0: GOTO
    60
40  Y = SQR (100 - X * X)
60  PRINT "X=";X,"Y=";Y
70  NEXT
```

本程序的特点是比 №23 程序行几乎少一半, 节省内存 20 个字节, 但二者运行速度无大的差异。

ULIST

```
5  REM  NO.23
10  FOR X = - 12 TO 10
20  IF X > = 0 THEN 50
30  IF X < - 10 THEN 60
40  Y = SQR ( ABS (100 - X * X))
45  GOTO 70
50  Y = 10
55  GOTO 70
60  Y = 0
70  PRINT "X=";X,"Y=";Y
80  NEXT X
90  END
```

ORUN	
X=-12	Y=0
X=-11	Y=0
X=-10	Y=0
X=-9	Y=4.35889894
X=-8	Y=6
X=-7	Y=7.14142843
X=-6	Y=8
X=-5	Y=8.66025404
X=-4	Y=9.16515139
X=-3	Y=9.53939202
X=-2	Y=9.79795897
X=-1	Y=9.94987438
X=0	Y=10
X=1	Y=10
X=2	Y=10
X=3	Y=10
X=4	Y=10
X=5	Y=10
X=6	Y=10
X=7	Y=10
X=8	Y=10
X=9	Y=10
X=10	Y=10

两个程序都不能用 $X \wedge 2$ ，否则机器不执行，虽然从形式上看 $Y = \text{SQR}(100 - X \wedge 2)$ 与 $Y = \text{SQR}(100 - X * X)$ 都符合 BASIC 规则。

13. 尽量不用GOTO语句

例如，求全班学生某一门课的平均成绩。

解法 1：


```

5: S=0
10: READ X
15: IF X=-1 THEN 40
20: S=S+X
30: GOTO 10
40: LPRINT S/20
60: DATA 87, 65, 98,
      75, 100, 92, 86, 7
      7, 90, 98
70: DATA 96, 68, 69,
      85, 74, 95, 86, 87
      , 85, 99, -1

```

85.6

解法 2:

```

5: S=0
10: FOR I=1 TO 20
20: READ X
30: S=S+X
40: NEXT I
50: LPRINT S/20
60: DATA 87, 65, 98,
      75, 100, 92, 86, 7
      7, 90, 98
70: DATA 96, 68, 69,
      85, 74, 95, 86, 87
      , 85, 99

```

85.6

说明:

① 解法 2 不用 GOTO 语句, 速度比解法 1 提高一倍, 如果数据更多, 或者程序中省去多个 GOTO 语句, 则提高运行速度更为明显;

② 解法 2 的程序中, 如果 50 句改成 LPRINT(S/I), 则结果就不对了, 变成 81.52380952, 原因是循环结束, 终值加步长 1, $I = 21$, 而不是 20, 这是循环的基本概念, 初学者常常弄错。

14. 尽量少用乘方运算

有三位整数, 等于每位整数的立方和。

例如, $153 = 1^3 + 5^3 + 3^3$ 称水仙花数, 试求 100 到 1000 之间的水仙花数。

解法 1: 设所求的数为 I , 这是一个三位数, 设百位数、十位数和个位数分别为 X , Y , Z , 显然, 满足 $I = X^3 + Y^3 + Z^3$ 的数才是所求的水仙花数。

由于要找 100 到 1000 之间的水仙花数, 可以安排一个循环, 让其逐个“扫描”。

百位数: $X = \text{INT}(I/100)$

十位数: $Y = \text{INT}((I - X*100)/10)$

个位数: $Z = I - X*100 - Y*10$

见程序: No.24:

```
5  REM  NO.24
10  FOR I = 100 TO 999
20  X =  INT (I / 100)
30  Y =  INT ((I - X * 100) / 10)
```

```

40 Z = I - X * 100 - Y * 10
50 IF I < > X ^ 3 + Y ^ 3 + Z ^
   3 THEN 70
60 PRINT I
70 NEXT I
80 END

```

运行上述程序后，并没有任何结果。在PC-1500机上运行了3分20秒，在APPLE-Ⅱ机上运行了2分10秒，均以提示符告终。

再仔细检查原程序和键入机器的程序均没有错误，程序设计和语法规则也都是正确的，那么是什么原因呢？

原来，在BASIC语言中，计算机做乘方是通过 \ln 与 e 的指数运算来完成的。

如求： $X = A^B$

实际上是先做 $\ln X = B \ln A$ 的运算，再得到 $X = e^{B \ln A}$

即机器求 X 值，是先求 A 的对数，再乘 B ，然后进行指数运算，这就解释了机器进行乘方运算，速度所以慢的原因。

同时，上述运算要求 A 既不为零，也不能为负数，国产机DTS-100系列以及PC-1500，APPLE-Ⅱ，紫金-Ⅱ都有这个约定。

另外一个很重要的原因，就是机器做乘方运算精度不够。例如 $3^4 = 81$ ，而机器计算得出的结果是80.9998；又如， $6^3 = 216$ ，而计算机显示的结果为215.9999，这就解释了为什么上述程序运行后得不到正确结果的原因。

例如，153这个数是一个水仙花数，因为

$$1^3 + 5^3 + 3^3 = 153$$

但机器执行上述程序时，会出现什么情况呢？如：

$$I = 153$$

执行 20句 $X = 1$

执行 30句 $Y = 5$

执行 40句 $Z = 3$

执行 50句 $I \approx 1 \wedge 3 + 5 \wedge 3 + 3 \wedge 3$

从而跳到 70 句，再循环。出现 50 句不等式成立的原因是 $1 \wedge 3 + 5 \wedge 3 + 3 \wedge 3 \approx 153$ (这是计算机精度不够造成的)。

通过这个例题，可以得出这样的结论，以后编程时，尽量少用乘方，特别是要求精度较高的问题，更不要用乘方运算，因为它的运算既不精确，而且速度也慢。

那么，上面的题目是否无解呢？显然不是的，它不仅无解，而是有多组解，方法之一是把所有立方数改成连乘。程序及运行结果如下：

```
5  REM    NO.25
10  FOR I = 100 TO 999
20  X =  INT (I / 100)
30  Y =  INT ((I - X * 100) / 10)
40  Z =  I - X * 100 - Y * 10
50  IF I < > X * X * X + Y * Y *
    Y + Z * Z * Z THEN 70
60  PRINT X,Y,Z
70  NEXT I
80  END
```

ORUN

1	5	3
3	7	0
3	7	1
4	0	7

上述程序在PC-1500机上运行1分40秒，在APPLE-Ⅱ机上运行约40秒。这也进一步说明了不用乘方，速度快多了。

上述运行结果表明 $I > 407$ 以后就没有水仙花可找了，所以10句改成FOR I = 150 TO 410，则运行更快，因为省去了600多次无效循环。

说明：上面我们通过14个方面，近22个程序，对程序设计中注意的问题进行了实例分析和初步探讨。这些问题一般来说比较简单，仅仅勾画了程序设计技巧的一些侧面。实际上，编程的方法和技巧，注意的问题和方面，远远不止这些。我们将在以后的内容中不断补充、不断扩展。同时，也希望广大读者，在实践中不断总结提高、有所发现有所创造，编制出更多更好的程序来。

六、打印输出的技巧

人们在利用计算机进行计算的时候，总是需要把计算的结果通过输出设备打印出来。在BASIC语言中，具有两种打印语句，一种是PRINT语句，它使输出的数据在显示器的屏幕上显示出来；另一种是LPRINT语句，它使输出的信息在打印纸上打印出来。PC-1500机同时具备上述两种打印语句，而APPLE II只有PRINT语句。

在这一章，我们不想重复讲述有关打印语句的功能、规则和输出格式等内容，因为这些内容对已学过BASIC语言的读者来说，显得多余和烦琐。而对没有学过BASIC语言的同志来说，可以从任一本教材和书籍中查阅。

这里要介绍的，是打印输出方面的一些技巧和实例，而这些可能是一般书籍中很少见到的。目的在于提供使用。

1. 输出空格函数（SPC函数）的使用

如果某些问题需要按照某种特定格式输出时，仅用PRINT语句，显然是不够的，虽然它们比较方便，但过于死板。使用SPC函数和TAB函数（输出定位函数），将使输出格式更加自由灵便。

SPC（<算术表达式>）只能与PRINT语句连用，其功能是生成若干空格。

例如，显示如下设计的工资表头

```

*****
*           *
*  WAGES LIST  *
*           *
*****

```

则程序设计为No.26:

ULIST

```

5  REM NO.26
10 A$ = "*****"
20 PRINT SPC( 12);A$
30 PRINT SPC( 12);"*"; SPC( 12)
   ; "*"
40 PRINT SPC( 12);"*"; SPC( 1);
   "WAGES"; SPC( 1);"LIST"; SPC(
   1); "*"
50 PRINT SPC( 12);"*"; SPC( 12)
   ; "*"
60 PRINT SPC( 12);A$

```

程序执行后,即可显示如上形式的表头。

2. 输出定位函数 (TAB函数) 的使用

(1) 下面是用TAB函数来改写前面那个列印工资表头的程序, 见程序No.27:

ULIST

```
5 REM NO.27
10 A$ = "*****"
20 PRINT TAB( 13);A$
30 PRINT TAB( 13);"*"; TAB( 26)
   ;"*"
40 PRINT TAB( 13);"*"; TAB( 15)
   ;"WAGES"; TAB( 21);"LIST"; TAB(
   26);"*"
50 PRINT TAB( 13);"*"; TAB( 26)
   ;"*"
60 PRINT TAB( 13);A$
```

读者可上机运行此程序，观察其结果。

(2) 打印图案例题，见程序No.28:

```
5:REM NO.28
7:CSIZE 1
8:FOR K=1TO 5
9:COLOR 3
10:LPRINT TAB (18
   ); "*"
20:X=18
30:Y=X
40:FOR I=1TO 5
50:X=X-1
60:Y=Y+1
70:LPRINT TAB (X)
   ; "*" ; TAB (Y); "
   *"
80:NEXT I
85:COLOR 2
90:FOR I=1TO 4
```



```

100:X=X+1
110:Y=Y-1
120:LPRINT TAB (X)
      ;"*";TAB (Y);"
      *
130:NEXT I
140:LPRINT TAB (18
      );"*"
150:NEXT K
160:END

```

3. 数据的列印输出

这是指将有关的数据列印在纸上。

(1) 一般常用的方法,打开打印机电源,键入PR#1↵,即可启动打印机,LIST↵,所有在屏幕上的程序在打印纸上打印出来。RUN↵打印输出结果。不用打印机时,按RESET键或键入PR#0↵。

若要打印80个字符,可在键盘上键入POKE 1657, 80↵

若要打印高分辨率图象,可键入PRINT CHR\$(17)↵和POKE 1913, 1等。

(2) 在程序中用打印机命令,见程序No.29;

LIST

```

5  REM. NO.29
10  PR# 1
20  PRINT "CTRL I BON"
30  PRINT "PTINTER"
40  PRINT "CTRL I I"
50  PRINT "SCREEN AND PRINTER"

```

```

55 PRINT "QWERTYUIOPASDNJHBVGFCX
    DRDSEYGHU"
60 PRINT "ZXCVBNMKJHGFD"
70 PR# 0
80 PRINT "SCREEN ONLY"

```

```

URUN
CTRL I BON
PTINTER
CTRL I I
SCREEN AND PRINTER
QWERTYUIOPASDNJHBVGFCXDRDSEYGHU
ZXCVBNMKJHGFD

```

(3) DOS不在内存中的列印方法, 见程序%30:

ULIST

```

3 REM NO.30
5 PR# 1
10 A$ = "*****"
20 PRINT TAB( 13);A$
30 PRINT TAB( 13);"*"; TAB( 26)
    ; "*"
40 PRINT TAB( 13);"*"; TAB( 15)
    ; "WAGES"; TAB( 21); "LIST"; TAB(
    26); "*"
50 PRINT TAB( 13);"*"; TAB( 26)
    ; "*"
60 PRINT TAB( 13);A$
70 PR# 0

```

RUN↵, 程序执行到5句时系统自动接通打印机, 然后打印工资表头在纸上, 至70句, 输出设备又恢复为终端。

(4) DOS存在于内存之中的列印方法, 见程序No.31.

ULIST

```
0 REM NO.31
1 D$ = CHR$(4)
2 PRINT D$;"PR#1"
10 A$ = "*****"
20 PRINT SPC(12);A$
30 PRINT SPC(12);"*"; SPC(12)
   ; "*"
40 PRINT SPC(12);"*"; SPC(1);
   "WAGES"; SPC(1);"LIST"; SPC(
   1); "*"
50 PRINT SPC(12);"*"; SPC(12)
   ; "*"
60 PRINT SPC(12);A$
70 PRINT D$;"PR#0"
```

即在有关打印语句前加入: <行号>D\$=CHR\$(4), <行号>PRINT D\$; "PR#1", 返回终端用: <行号>PRINT D\$; "PR#0".

PRINT可以用?代替。

4. 打印机输出控制

(1) PRINT CHR\$(12)语句——跳新页

如果输出资料需要打印在新页时, 可在打印语句前加一条语句: PRINT CHR\$(12), 例如, 程序No.32,

ULIST

```
3  REM  NO.32
5  PR# 1
7  PRINT  CHR$ (12);
10 REM  PRO-5
20 PRINT "THIS IS A BASIC PROGRA
    M"
30 PRINT "DO YOU WANT PRINT(Y/N)
    ?"
40 PRINT "THIS IS A AUTO APPLE!"

50 PRINT "USE EXEC INSTRUCTION"
60 END
```

运行程序后，下述结果打印在新页上。

```
THIS IS A BASIC PROGRAM
DO YOU WANT PRINT(Y/N)?
THIS IS A AUTO APPLE!
USE EXEC INSTRUCTION
```

(2) PRINT CHR\$(10)语句——跳行控制 例如，程序No.33，

ULIST

```
5  REM  NO.33
10 PR# 1
20 PRINT "THIS IS A BASIC PROGRA
    M"
30 PRINT  CHR$ (10)
```

```
40 PRINT "DO YOU WANT PRINT(Y/N)
   ?"
```

```
50 PRINT CHR$ (10)
```

```
60 PRINT "USE EXEC INSTRUCTION"
```

```
GRUN
```

```
THIS IS A BASIC PROGRAM
```

```
DO YOU WANT PRINT(Y/N)?
```

```
USE EXEC INSTRUCTION
```

(3) 字体放大、缩小与还原

打印机在输出时可将字形横和竖方向各放大2倍或缩小为二分之一，以达到放大或缩小字体的目的。

PRINT CHR\$(14)——字体放大2倍

PRINT CHR\$(15)——字体缩小为1/2

PRINT CHR\$(13)——字体还原

下面是一个打印实例，见程序№34：

```
10 REM NO.34
20 PRINT CHR$ (4); "PR#1"
30 PRINT
40 A$ = "BASIC"
50 B$ = "PROGRAM"
60 PRINT A$, B$
70 PRINT CHR$ (14); A$; " "; B$; CHR$
   (31), A$
80 PRINT CHR$ (14); A$, B$
90 PRINT A$
```

```

95 PRINT CHR$ (15);A$,B$
100 PRINT A$,B$
110 PRINT CHR$ (15);A$;"    ";B$
    ;"    "; CHR$ (18);A$ + " " +
    B$
120 PRINT CHR$ (15);A$;"    ";B$
    ;"    "; CHR$ (14),A$
130 PRINT
140 PRINT CHR$ (4);"PR#0"
150 END

```

URUN

```

BASIC          PROGRAM
BASIC PROGRAM BASIC
BASIC          PROGRAM
BASIC          PROGRAM
BASIC PROGRAM BASIC PROGRAM
BASIC PROGRAM BASIC

```

5. 打印输出结果的对齐处理

在打印输出计算结果时，总希望输出格式统一美观，如字符输出前面对齐、输出结果后面对齐、或者打印成简单的报表形式，那么，在程序中如何实现呢？

(1) 字符输出的前面对齐 请看程序№35，该程序十分简单，循环读数并打印输出，为了使输出的结果前面对齐，主要加了50句，下面是程序清单和运行结果。

```

10  REM NO.35
15  N = 10
20  DIM X(N),Y(N),Z(N),W(N)
30  FOR I = 1 TO N
40  READ X(I),Y(I),Z(I),W(I)
50  PRINT TAB( 1);X(I); SPC( 10 -
      LEN ( STR$ (X(I))));Y(I); SPC(
      12 - LEN ( STR$ (Y(I))));Z(
      I); SPC( 13 - LEN ( STR$ (Z
      (I))));W(I)
60  NEXT I
70  DATA    0.2,0.14,0.432,1.00
80  DATA    1.0,0.43,1.734,0.994
90  DATA    2.0,0.68,2.782,0.982
100 DATA    3.0,0.9,3.660,0.970

110 DATA    4.0,1.09,4.254,0.956
120 DATA    5.0,1.27,4.845,0.939
130 DATA    6.0,1.43,5.242,0.921
140 DATA    8.0,1.74,5.965,0.881
150 DATA    10,2.02,6.530,0.835
160 DATA    12,2.28,7.057,0.794

```

URUN

.2	.14	.432	1
1	.43	1.734	.994
2	.68	2.782	.982
3	.9	3.66	.97
4	1.09	4.254	.956
5	1.27	4.845	.939
6	1.43	5.242	.921
8	1.74	5.965	.881
10	2.02	6.53	.835
12	2.28	7.057	.794

(2) 输出字符的后面对齐 和上面程序基本一样, 只需要改动50句为:

```
50 PRINT SPC( 2 - LEN ( STR$ (
    X(I))) );X(I); SPC( 10 - LEN
    ( STR$ (Y(I))) );Y(I); SPC( 1
    2 - LEN ( STR$ (Z(I))) );Z(I
    ); SPC( 13 - LEN ( STR$ (W(
    I))) );W(I)
```

输出结果如下:

URUN			
.2	.14	.432	1
1	.43	1.734	.994
2	.68	2.782	.982
3	.9	3.66	.97
4	1.09	4.254	.956
5	1.27	4.845	.939
6	1.43	5.242	.921
8	1.74	5.965	.881
10	2.02	6.53	.835
12	2.28	7.057	.794

(3) 打印成报表形式 用调子程序的方法, 在需要画线的地方就调画线子程序 (500句), 应该指出的是, 子程序的最后, 除了加510句 RETURN以外, 还应加505句判断语句, IF I > 10 THEN END, 否则在打印结束后, 出现? RETURN WITHOUT GOSUB ERROR IN 510。转子程序放在循环中, 即46 GOSUB 500。下面是程序№36的清单和结果, 字符输出采用后面对齐的方式。


```

10  REM    NO.36
15  N = 10
20  DIM X(N),Y(N),Z(N),W(N)
30  FOR I = 1 TO N
40  READ X(I),Y(I),Z(I),W(I)
46  GOSUB 500
50  PRINT SPC( 2 - LEN ( STR$ (
      X(I) ) ) );X(I); SPC( 10 - LEN
      ( STR$ (Y(I) ) ) );Y(I); SPC( 1
      2 - LEN ( STR$ (Z(I) ) ) );Z(I
      ); SPC( 13 - LEN ( STR$ (W(
      I) ) ) );W(I)
60  NEXT I
70  DATA 0.2,0.14,0.432,1.00
80  DATA 1.0,0.43,1.734,0.994
90  DATA 2.0,0.68,2.782,0.982
100 DATA 3.0,0.9,3.660,0.970

110 DATA 4.0,1.09,4.254,0.956
120 DATA 5.0,1.27,4.845,0.939
130 DATA 6.0,1.43,5.242,0.921
140 DATA 8.0,1.74,5.965,0.881
150 DATA 10,2.02,6.530,0.835
160 DATA 12,2.28,7.057,0.794
500 PRINT "-----"
      "-----"
505 IF I > 10 THEN END
510 RETURN

```

URUN

.2	.14	.432	1
1	.43	1.734	.994
2	.68	2.782	.982
3	.9	3.66	.97
4	1.09	4.254	.956
5	1.27	4.845	.939
6	1.43	5.242	.921
8	1.74	5.965	.881
10	2.02	6.53	.835
12	2.28	7.057	.794

(4) 小数点对齐的打印输出 在打印输出数据时，常常希望小数点能够对齐，下面采用字符串函数来处理这类问题。

例如，有10组数据已存放在DATA区中，现需要将每组中的第2项，第3项数据打印出来，且要求小数点对齐。

以小数点后取两位为例：

首先让 $B = X * 100$ ，这就是将每组的第2项数值扩大100倍，如0.14就变成14，1.09则变成109。

其次将B值变成字符, 如T\$ = "□□□□□□" + STR\$(B)。

再使T\$ = RIGHT\$(T\$, 8), 即从右边取T\$的8个字符(空格也算一个字符), 如□□□□□□14; □□□□□109等。

最后安排 PRINT LEFT\$(T\$, 6) + "." + RIGHT\$(T\$, 2), 这样, 14和1.09的小数位就对齐了。

其它各位由循环自动完成, 而对于小数点后三位的处理方法同上, 但必须用新的字符串变量, 本例中用Q\$。

下面给出程序№37和结果, 注意体会取字符个数及空格数的安排。

```
10 REM NO.37
30 FOR I = 1 TO 10
40 READ Z,X,Y,W
45 B = X * 100:C = Y * 1000
47 T$ = " " + STR$(B):Q$ =
   " " + STR$(C)
50 T$ = RIGHT$(T$,8):Q$ = RIGHT$(Q$,11)
52 PRINT LEFT$(T$,6) + "." + RIGHT$(T$,2);
54 PRINT TAB(12); LEFT$(Q$,8)
   + "." + RIGHT$(Q$,3)
60 NEXT I
70 DATA 0.2,0.14,0.432,1.00
80 DATA 1.0,0.43,1.734,0.994
90 DATA 2.0,0.68,2.782,0.982
100 DATA 3.0,0.9,3.660,0.970
```

110	DATA	4.0, 1.09, 4.254, 0.956
120	DATA	5.0, 1.27, 4.845, 0.939
130	DATA	6.0, 1.43, 5.242, 0.921
140	DATA	8.0, 1.74, 5.965, 0.881
150	DATA	10, 2.02, 6.530, 0.835
160	DATA	12, 2.28, 7.057, 0.794

URUN

.14	.432
.43	1.734
.68	2.782
.90	3.660
1.09	4.254
1.27	4.845
1.43	5.242
1.74	5.965
2.02	6.530
2.28	7.057

8. 字符的放大打印

在APPLE-Ⅱ微型计算机的外围设备EPSON, CP80等打印机中, 有一个控制符CHR\$(14), 其功能是设置放大字符打印方式, 但这个放大打印命令只对一行有效, 即它只能放大打印此控制符后面的一行字符, 虽然字体美观醒目, 但一遇回车字符(CR)后, 其放大打印方式将被自动解除。如果要打印输出多行的放大的程序清单或数据等, 则要在每一行前加PRINT CHR\$(14); 实在感到过于烦琐, 若改用下面的机器语言程序, 便可较好地解决字符的放大打印输出问题。

机器语言的起止地址为\$0300—\$0323, 在CALL-151 监

控状态下，键入：

***300.323**

```
0300- A9 0E 85 36 A9 03 85 37
0308- 20 EA 03 4C 00 C1 85 1C
0310- 20 00 C1 A5 1C C9 8D D0
0318- 0A A9 1B 20 00 C1 A9 0E
0320- 20 00 C1 CD
```

将上述机器语言子程序存入磁盘备用，取文件名为BIG。
存盘方法：BSAVE BIG, A\$ 300, L\$23。

放大打印的操作步骤及实例如下：

- NEW↵
- BLOAD BIG↵
- LOAD No.26↵
- PR#1↵
- CALL768↵
- LIST↵
- RUN↵
- PR#0↵

说明：

• BLOAD BIG↵是指调已存于盘中的文件名为BIG的
机器语言子程序。

• LOAD后面可以是任意一个存于盘中的程序或文件，
本例是调No.26文件。

• CALL后面的768，是10进制数，对应的16进制数是
300，它正是机器语言子程序的入口地址。

ULIST

```

5  REM NO.26
10  A#="*****"
20  PRINT SPC( 12);A#
30  PRINT SPC( 12);" "; SPC( 12)
   ;***:
40  PRINT SPC( 12);" "; SPC( 12)
   "WAGES"; SPC( 12)"LIST"; SPC(
   12);***:
50  PRINT SPC( 12);" "; SPC( 12)
   ;***:
60  PRINT SPC( 12);A#

```

URUN

```

*****
**
**  WAGES LIST
**
*****

```

七、程序调试技巧

这一章主要谈程序调试问题，但是，很显然程序调试不是一个孤立的技术，它也涉及微机应用和操作的实际本领，以及语言的编程方法和技巧问题，因此，我们尽量谈及解决问题的各个方面，但主要偏重于调试技巧。

1. 不同机器的差异

当我们编制好程序，上机调试的时候，会发现这样的情况，同一个程序在不同的机器上执行结果相差很大。

这里有两个实例：

例 1，写出程序 No.38 的运行结果：

```
10:REM NO.38
20:FOR X=1 TO 10
    STEP 0
30:LPRINT X;
40:NEXT X
50:END
```

我们先不忙上机执行，而是判断一下可能是什么结果，
今给出以下三种答案，你认为哪一个对？

- 10 个 1；
- 无穷多个 1；
- 没有结果。

现来分析一下上面三种解答，第一种解法认为循环次数

共 10 次，第一次循环初值 1 赋给 X，未超过终值，故执行循环体后打印输出为 1，因步长为 0，再循环 10 次，所以输出 10 个 1。持第二种解法的观点似乎更妥当些，因无步长可加，永远达不到终值，所以认为有无数个 1。第三种解法认为没有结果，理由是有的微机不执行 0 步长的程序。

那么，究竟哪一个答案对呢？

现把程序 №38 键入 PC-1500 机，RUN↵，于是有以下显示：

```
ERROR 19 IN 20
```

结果表明：PC-1500 机不执行步长为零的循环程序。

应该指出的是，上述程序是正确的，第二种解答的正确性也是不容怀疑的，在有的微机（如 APPLE-II）上，解为无数个 1，有兴趣的读者不妨试一下。

例 2，试问将数 300003 的百位数和万位数上的零换成什么数后，所得结果是 13 的倍数（只要求写出正确的程序，换的数相同）。

有人给出程序 №39：

```
10:REM NO.39
20:A=300003
30:FOR I=10100TO
    90900STEP 1010
    0
40:S=A+I
50:IF INT (S/13)<
    >S/13THEN 70
60:LPRINT S
70:NEXT I
80:END
```


这个程序从理论上分析是没有错误的，语法规则似乎也对，可是运行本程序却没有结果。在 PC-1500 机上同样给出 19 类错误，原因是循环变量的终值 90900 超过规定范围。但是，同样这个程序，在 APPLE-II 机上，却可以得到正确的结果：320203。

请看：程序 No.40：

```
10 REM NO.40
20 A = 300003
30 FOR I = 10100 TO 90900 STEP 1
    0100
40 S = A + I
50 IF INT (S / 13) < > S / 13 THEN
    70
60 PRINT S
70 NEXT I
80 END
```

```
URUN
320203
```

为了能在 PC-1500 机上得到本题的答案，可作如下改动，见程序 No.41：

```

10: REM NO. 41
20: A=300003
30: FOR I=1 TO 9
40: S=A+I*10000+I*
    100
50: IF S/13=INT (S
    /13) THEN
    LPRINT S
60: NEXT I
70: END
80: END
320223

```

2. 尽量避开乘方运算

例如，已知四位数字的数 3025 有一个特殊性质：它的前两位数字(30)和后两位数字(25)的和平方就是该数本身 $((30+25)^2 = 3025)$ 。试列出具有这种性质的所有4位数字的数（1986年四川省青少年计算机程序设计竞赛高中组第五题）。

```

10: REM NO. 42
20: FOR N=1000 TO 9
    999
30: F=INT (N/100)
40: L=N-100*F
50: IF (F+L)^2<>N
    THEN 70
60: LPRINT N;" ";F
    +L
70: NEXT N
80: END

```

从程序的正确性和语法规则来看，程序 No.42 是无可指责的。但它在 PC-1500 机上运行了 16 分 30 秒，无任何结果。在紫金-Ⅱ上运行 10 分钟左右，最后以提示符] 告终。

可见两种机种均不执行本题的乘方运算。

错误在 50 句的乘方上，原因同前所述。这样，本题中 50 句 $(F+L) \wedge 2 < N$ 永远成立而转向 70 句，故本题以提示符告终而无结果。

为了得到正确结果，只要将乘幂运算改成连乘即可：

```
50 IF(F+L)*(F+L)< N THEN 70
```

运行结果为：

2025	45
3025	55
9801	99

3. 作适当的精度调整

前面谈到的求水仙花问题，除了将乘方改成连乘可以求解外，还可以用下述程序 No.43 处理：

```
10 REM NO.43
20 FOR I = 100 TO 999
30 X = INT (I / 100)
40 Y = INT ((I - X * 100) / 10)
50 Z = I - X * 100 - Y * 10
60 IF I < > INT (X ^ 3 + Y ^ 3
    + Z ^ 3 + 0.5) THEN 80
70 PRINT X,Y,Z
80 NEXT I
90 END
```

QRUN

1	5	3
3	7	0
3	7	1
4	0	7

由此可见，不是微机都不可以做乘方，只是要考虑机器的精度，作必要的处理，本例中采用多加0.5的方法。

4. 如何处理小数步长

有的机器没有处理小数步长的功能，例如PC-1500，因而程序 No.44 无法运行：

```
5:REM NO.44
10:FOR I=1TO 10
    STEP 0.5
20:LPRINT I;
30:NEXT I
40:NEXT
```

解决此问题的方法有多种，今给出两种解法。程序主要思路是成比例地扩大循环变量的初值、终值和步长，然后在循环体中再缩小相应的倍数。

①

```
10:FOR I=10TO 100
    STEP 5
20:LPRINT I/10;
30:NEXT I
40:END
```

```
1 1.5 2 2.5 3 3.5
4 4.5 5 5.5 6 6.5
7 7.5 8 8.5 9 9.5
10
```

②

```
10:FOR I=0TO 18
20:T=1+0.5*I
30:LPRINT T;
40:NEXT I
50:END
```

```
1 1.5 2 2.5 3 3.5
4 4.5 5 5.5 6 6.5
7 7.5 8 8.5 9 9.5
10
```

第一种解法比较简单，第二种解法比较巧妙。

5. 屏幕抄写的方法

当程序较大时，输入程序花费的时间较长，但是有些语句彼此基本相同，如采用屏幕抄写的方法，可使工作量大大减轻，并加快程序的输入速度。

例如要打入如下程序：

```
10 FOR I=1 TO LEN(A$):B$=MID$(A$,I,1)
20 E=ASC(B$)+A:IF E > 255 THEN E=E-255
30 N$=N$+CHR$(E):NEXT I
:
:
:
:
80 FOR I=1 TO LEN(A$):B$=MID$(A$,I,1)
90 E=ASC(B$)-A:IF E < 0 THEN E=E+255
100 N$=N$+CHR$(E):NEXT I
:
:
:
```

在程序中第 10、80 句以及第 30、100 句全部相同，此时不用每句都同样打一遍，可以先键入第 10 句，然后将光标移到第 10 句的行号处，将行号改成 80 后再将光标移至本行的程序尾，回车即可。这样第 10 句和第 80 句同时存在。30 句和 100 句按同样方法处理。

20 句和 90 句基本相同，可先键入 20 句，然后将光标移至第 20 句的行号处，改成 90，再将光标移至“+”下面，改成“-”；移至“>”下面，改成“<”；移至“255”处，改成“0”，最后再移至 E 后面的“-”处，改成“+”，“→”移至程序尾，回车即可。

上述方法对于有大量相同字符，特别是有相同汉字的语句来讲，效果极为明显。

6. 程序编辑举例

提高键盘输入效率，增强程序删除、修改、增补、插入的能力，是加快调试速度，节省机时的重要手段。

由于程序编辑的指令较多，涉及的内容也广泛，本节不想全面阐述，仅就初学者感到不太熟悉，比较麻烦的操作，举一、二个实例。

(1) 程序的删除 初学者对清除某一程序行，通常是键入该行号，按回车完成。若删除多个程序行，用此法就显得过于烦琐，应改用 DEL 命令。

如 DEL 90, 400

则可删除 50 行至 400 行的全部语句行，加快了删除的速度。

若删除磁盘中的文件，则用 DELETE 命令。如 DELETE

START↵, 即将文件名为 START 的文件取消, 从而空出原占有的磁区, 用来存放新的文件。

(2) 字符的插入 对 APPLE-II 型机及兼容型紫金-II 机, 都不能将原来一串字符从中推开而腾出空位让其他字符插入, 因而在语句行中插入字符的操作比较麻烦, 这里仅举一例说明插入过程。

例如: 有一语句行

```
10 PRINT "THIS IS A STRING CONVERSION PROGRAM"
```

现要求在 10 句 PRINT 的后面, 插入 TAB(5), 插入操作步骤如下:

① 按 **ESC** 键, 进入编辑;

② 用 **I** 键, **J** 键把光标移动到上述语句的最左边字符;

③ 用 **→** 键将光标右移到要插入地方 (此时屏幕上显示为:

```
10 PRINT " THIS IS A STARING CON-  
VERSION PROGRAM" );
```

④ 再用 **ESC** **I** 键将光标往上移动一格, 接着按 **RETURN**, 退出编辑, 键入 TAB(5)

(现在屏幕显示为:

```
TAB(5); █
```

```
10 PRINT "THIS IS A STARING CONVER-  
SION PROGRAM" );
```

⑤ 按 **ESC** **M** 键, 把光标下移一行, 移到 10 语句上;

⑥ 再用 **J** 键将光标左移到第一双引号处, 从这里开

始,再用 \rightarrow 键,将光标移到语句的最后,按 $\boxed{\text{RETURN}}$ 键退出编辑;

⑦ 键入LIST 10,检查,如果你希望能避免多余的空白,用 $\boxed{\text{ESC}}$ $\boxed{\text{A}}$ 键会将显示点向右移,但不超字节。

7. 执行过程的自动跟踪

程序设计往往不会一次成功,错误难免。在调试程序、检查错误的过程中,常常希望知道程序是否按我们设计的要求顺序执行,以便跟踪程序的执行过程进行检查,从而加速纠错的速度,这就要用TRACE语句(IBM-PC机用TRON)。

它的功能是指在程序执行过程中,每执行一步都把与之相关的语句行号列出来。

在调试程序前,先键入跟踪命令TRACE,然后再执行RUN命令。解除跟踪的语句是NOTRACE,例如:

把100(含100)以内的所有被7整除或被5整除的正整数一一打印出来。

有人编写程序No.45:

```
5  REM NO.45
10  FOR N = 1 TO 100
20  IF INT (N / 5) < > N / 5 AND
    INT (N / 7) < > N / 7 THEN
    10
30  PRINT N;" ";
40  NEXT N
50  PRINT
60  END
```


RUN↵后,什么结果也没有。可见这个程序是有错误的。

为了检查上述程序错误的原因,键入TRACE,再运行,结果:

```
URUN70
TRACE
#5 #10 #20 #10 #20 #10 ....
```

说明程序进入死循环,永远得不到答案,若在上述程序№45中加上一句:

```
15 PRINT N;
```

程序会没完没了地打印出 1。

```
URUN
1111111.....11111.....
```

注意:在程序的适当位置,安排打印语句,判断结果的正确与否,也是调试程序常用的方法。

总之,利用跟踪语句,可以迅速发现错误原因,事实上,上述错误是程序设计不允许的。请看:第一次进入循环体时, N 为 1, 它既不能为 5 整除, 又不能被 7 整除, 因而转回 10 行。第二次进入循环, N 仍为 1, 经 20 句又返回, 如此往复, 以至无穷, 所以得不到结果。正确程序设计和结果, 见程序№46:

```
ULIST

5  REM  NO.46
10  FOR N = 1 TO 100
20  IF INT (N / 5) < > N / 5 AND
    INT (N / 7) < > N / 7 THEN
    40
30  PRINT N; " ";
40  NEXT N
50  END
```

5 7 10 14 15 20 21 25

28 30 35 40 42 45 49

50 55 56 60 63 65 70

75 77 80 84 85 90 91

95 98 100

8. 输入数据的跟踪检查

用 INPUT 语句大量输入数据时，常常由于数据量大，时间长，操作失误是很难避免的，输入的数据由于无法更改只好作废，再重新输入又会花费不少时间，同时也很难保证不再出错。

如果在输入数据的过程中，随时可以跟踪检查，则可以节省不少工作量。

程序 №47 可以帮助解决上述困难。

LIST

```
5  REM NO.47
10 N = 10: DIM R(N): FOR I = 1 TO
    N
20  PRINT I,: INPUT A$:B$ = LEFT$(
    (A$,1): IF B$ = "A" OR B$ =
    "M" THEN A$ = A$ + ".":Y = -SGN
```

```

      (ASC (B$) - 66):T = - VAL
      (MID$ (A$,2)) * Y:I = I + T
      - (T = 0) * Y:I = (I > N) *
      N + (I < 1) + (I < = N AND
      I > = 1) * I: GOTO 20
30 R(I) = VAL (A$): NEXT I
40 FOR I = 1 TO N: PRINT R(I);"
      "": NEXT I

```

如果输入过程中出错，可以用 M 或 A 后面跟一个数字来修改错误。例如在第五次发现第三次数据有错，则键入 M2（表示退二个序号），屏上立即出现第三次的序号，此时键入要改的数据。而用 A 后面带一个数字，则表示前进几个序号，下面是一组实例。

ORUN

1	?12
2	?54
3	?67
4	?33
5	?M2
3	?44
4	?79
5	?80
6	?74
7	?33
8	?21
9	?99
10	?M3
7	?65
8	?A
9	?89
10	?30
12	54 44 79 80 74 65 21 89 30

9. 几个常用技术

(1) 暂停技术 为了调试的需要,有时要让程序运行到某一语句后暂停下来,这可以在运行前事先插入一些 STOP 语句。这样程序运行到该语句时会自动停下来。用这种方法特别适用大型程序的调试,因为大型程序经常采用模块化结构,每个模块分别完成某种功能,用 STOP 语句放在适当的模块后的相应位置,可以逐一检查每个模块功能的正确性,待全部调试完成后,再删去所有 STOP 语句。

(2) 插入查错技术 在程序中插入一些专门用以检查和提示的语句,以判断计算结果的正确性,数据存贮的正误,程序的流向以及执行过程。这种技术还适用于检查某一循环是否被执行,某一支是否正确转移等等。

插入语句可以是显示语句、打印语句或者用音响提示语句。

例如:

```
80 ON X GOSUB 400, 600
```

如果在 400, 600 这些子程序中,没有明显的对屏幕显示信息,没有标志或提示,这样就不知道程序的确切流向。插入 PRINT 语句,就能了解整个程序流向和执行过程。

```
例如: 78 PRINT N
      401 PRINT "NETER 400"
      601 PRINT "ENTER 600"
```

又如可以从键盘上直接输入有关语句,检查数组存贮情况,某个计算结果是否正确等。

```
PRINT N(7)
PRINT A+B
```

屏幕上立即显示N(7)中的存贮情况和表达式 $A+B$ 的结果。当然这种方法适用于静态调试,即输入程序后,没有运行程序前,相应的上面提到的方法就是指动态调试了。

显然,上述两种方法,对一些程序较大,运行时间较长的系统来说,插入技术是行之有效的方法。

(3) 限制技术 对于一些一时无法迅速排除错误的程序来说,为了使调试工作继续进行下去,可采用限制技术,方法除了用 ERROR 语句外,还可以增加一些限制条件和范围的语句。

例如:

```
      :  
      :  
      :  
      :  
100  X=Y/W  
110  IF X >= 47 THEN 330  
      :  
      :  
      :  
      :
```

运行中如果 $W=0$, 则出错, 程序被迫停下来。为保证让程序先通过这一句, 可加一条限制:

```
99  IF W=0 THEN 500
```

这样此段程序就可顺利通过, 至于 500 句的内容, 视程序的要求去写。

(4) 循环延迟技术 对图象显示, 有时我们希望看得清楚些, 不希望一幅图象还没有停留一段时间, 另一幅图象随即接踵而来, 这时可以用延迟技术, 方法比较简单, 在显示两

幅图象之间程序段，加入：

```
〈行号〉 FOR K=1 TO 1000: NEXT K
```

终值的大小决定了延迟时间的长短，可由程序自行调整。

(5) 强制技术 是指将特定的数值人为地输入程序中，强制程序按预定的方式去运行，以考察程序是否产生预期的结果。这种方法常用来判断逻辑错误或算法错误。

例如：

```
30 INPUT K$  
40 IF 50 < = ASC (K$) AND 80 >  
    = ASC (K$) THEN 60  
50 GOTO 30
```

为了检查第 40 句逻辑功能是否正确，可以给 K\$ 送 0, 60, 90 等数，看其是否能分别转到第 30 句（经 50 句）、第 60 句、第 30 句（经 50 句）。

由于上述方法是带强制性的，人为地指定程序的运行路线，所以，它适用于全面调试一个程序。

八、BASIC 语言的编程技巧

程序设计是一项技术性很强而又十分灵活的工作，是模仿和创造的统一。模仿是对别人的程序设计进行学习、研究、试算、比较和借鉴的过程；而创造是指在符合程序设计语言规则的前提下，对程序进行提炼、取精、优化、创新和发挥的过程。

程序设计离不开语言，编程技巧离不开语句。而要进行高水平的程序设计，又归结于找到好的解题方法。这是一个包罗万象的课题，差不多有上百本书和大量文章来讨论解题的最好技术。这里仅从语句和函数的使用、程序设计思想和结构的安排上，作一些粗浅的介绍。

1. 菜单技术

我们曾经指出模块化程序结构是一种先进的程序设计方法。它的主要思想是将一个大的程序划分为若干功能模块，并供用户选择，但是究竟选用哪一个功能，完全由操作者自定。因此设计一个选择程序，供操作者使用，就显得十分必要。方法是在程序开始，计算机自动显示出各功能名称，用户根据需要键入希望完成的功能号，则程序立即转入相应的模块，由于这种方法酷似餐馆中的点菜方式，故称“菜单”技术。

下面是一个程序实例（见程序№48），采用多择一结

构:

```
5  REM NO.48
10  REM  CATALOG
20  PRINT "1. ADD A RECORD"
30  PRINT "2. DELETE A RECORD"
40  PRINT "3. CHANGE A RECORD"
50  PRINT "4. MERGN A RECORD"
60  PRINT "5. SORT RECORDS"
70  PRINT "6. SEARCHING RECORDS"
80  PRINT "7. DISPLAY A RECORD"
90  PRINT "8. LIST THE FILE"
100 PRINT
110 INPUT K$
120 IF K$ = "Q" THEN  END
130 MO = VAL (K$)
140 IF MO < 1 OR MO > 8 THEN 110

150 ON MO GOSUB 200,400,600,800,
    1000,1200,1400,1600
160 GOTO 110
```

说明:

程序一经运行, 立即显示各功能项:

URUN

1. ADD A RECORD
2. DELETE A RECORD
3. CHANGE A RECORD
4. MERGN A RECORD
5. SORT RECORDS
6. SEARCHING RECORDS
7. DISPLAY A RECORD
8. LIST THE FILE

INPUT K3 是等待用户响应，由操作者输入希望完成的功能代号，在 1 到 8 之间。

140 句是防止程序乱跑(BOMB PROOFING)。上述 8 种功能是用 1 到 8 这八个数字来响应的，若使用者可能按错数字键，则程序可能会“爆炸”，所以安排了“防爆”措施。增加了是否在 1 到 8 以内的逻辑检查，如出错，自动返回重新输入。它并不是文件管理上提出的要求，但是它属于提高程序质量的范畴。

150 句是程序在分析用户要求后，所作的分支处理，显然用开关控制语句 ON/GOSUB 最为合适。而 150 句后面的行号，是上述程序中相应 8 个功能模块的入口地址。每一独立的程序段，分别完成注释中的每种功能（8 个模块功能段此处从略，请不要忘记每段结尾处应有 RETURN）。

2. 人机对话方式

人机对话是计算机的重要功能，又是 BASIC 语言的一个显著特点。利用键盘进行人机对话，可以辅助教学工作，进行电化演示，既省力又省时，直观教学效果也好，日益受到普遍的欢迎，如有程序 No.49:

ULIST

```
5 REM NO.49
10 PRINT "HELLO! LET US LEARN ARITHMETIC"
20 S = 0
30 FOR I = 1 TO 10
40 A = INT (10 * RND (1))
50 B = INT (100 * RND (1))
```

```

60 PRINT "No: "; I
70 PRINT A; "*" ; B; "=" ;
80 INPUT C
85 PRINT C
90 IF C < > A * B THEN 110
100 S = S + 10
110 NEXT I
120 PRINT "YOUR SCORE IS: "; S
130 IF S > = 90 THEN PRINT "VE
RY GOOD!"
140 PRINT "I HOPE YOU WILL SUCCE
ED"
150 PRINT "DO YOU WANT TO CUNTIN
UE (YES OR NO)?"
160 INPUT "X$=" ; X$
170 IF X$ = "Y" THEN 20
180 PRINT "SEE YOU LATER!"
190 END

```

```

URUN
HELLO! LET US LEARN ARITHMETIC
No:1
4*57=?228
228
No:2
5*83=?413
413
No:3
8*57=?456
456
No:4
1*87=?87

```

```

87
No: 5
1*78=?78
78
No: 6
8*8=?64
64
No: 7
7*27=?169
169
No: 8
1*55=?55
55
No: 9
2*91=?182
182
No: 10
0*31=?31
31
YOUR SCORE IS: 70
I HOPE YOU WILL SUCCEED
DO YOU WANT TO CUNTINUE (YES OR NO)?
X$=N
SEE YOU LATER!

```

关于上述程序，不再一一说明了，请初学的读者耐心地一步一步地看懂它。简单地说是这样的：计算机随机地出 1 位整数和 2 位整数相乘的 10 个题目，请你回答，对一条自动加 10 分，不对不加分，最后给出你的成绩，如果你的成绩在 90 分以上，则给你一个好评语 (VERY GOOD!)，否则机器打印出：I HOPE YOU WILL SUCCEED (希望你成功) 的信息。并询问你是否再做试题，如你从键盘键

入 Y(Yes)，则再重复上述过程，反之，若你键入除 Y 这个字符以外的任何字符，则计算机打印出：SEE YOU LATER（下回见）的信息。简单地说这个程序是：计算机自动出题并判分。

读者可以仿照上述程序的模式，自己来编制一些不同内容的教学程序，比如说四则运算、三角函数学习、收音机调试、外语教学等等。

3. 巧用符号函数SGN(X)

对于统计一系列数字中正数、负数及零的个数问题，常用的方法是用条件语句来判断，GOTO 语句来循环，见程序 No.50。

```
5:REM NO.50
7:CLEAR
10:READ X
20:IF X=999THEN 6
   0
30:IF X>0LET R3=R
   3+1:GOTO 10
40:IF X=0LET R2=R
   2+1:GOTO 10
50:R1=R1+1:GOTO 1
   0
60:LPRINT R1;R2;R
   3
70:DATA 2, -3, 0, 4,
   -5, 8, 0, -6, 3, 7,
   999
```

3 2 5

巧用符号函数 $\text{SGN}(X)$ 能决定 X 符号的特点，容易写出程序No.51。R(3)、R(2)、R(1)记录正数、零和负数的个数，E为数值 X 的符号函数值。用E+2作为统计数 X 的正、负或零的数R的下标。程序的主要特点是省去了所有的条件判断和GOTO循环，巧妙地运用了数组存贮，见程序No.51:

```

5:REM NO.51
10: CLEAR :DIM R(3
)
20:FOR I=1TO 10
40:READ X
50:E=SGN (X)
60:R(E+2)=R(E+2)+
1
70:NEXT I
80:LPRINT R(1);R(
2);R(3)
90:DATA 2, -3, 0, 4,
-5, 8, 0, -3, 6, 7

```

3 2 5

实际上 $\text{SGN}(X)$ 函数形式虽然简单，功能却很特殊，它的应用方法和技巧很多，现再举几种用法如下。

① 代替逻辑运算

设A, B为逻辑变量，它们的取值不是0，就是1，那么A OR B，可以用 $\text{SGN}(A+B)$ 来代替。

② 代替多个分支语句

例如: 40 IF $X > 0$ THEN 100

```
50 IF X=0 THEN 150
```

```
60 IF X<0 THEN 200
```

可用一句代替:

```
40 ON SGN(X)+2 GOTO 200, 150, 100
```

③ 变复合条件为简单条件

例如: $A > B$ 且 $B > C$ 化为

$$\text{SGN}(A - B) + \text{SGN}(B - C) = 2$$

④ 求两数的大者或小者

例如, 从键盘输入A, B两数, 要求把大数放在H中, 小数放在L中, 一般采用比较和交换的方法再转移, 见程序No.52, 也可用赋值方法直接求得, 如程序No.53。

```
10:REM NO.52
20:INPUT A,B
25:LPRINT "A,B=";
   A;" ";B
30:IF A>BLET H=A:
   L=B:GOTO 50
40:H=B:L=A
50:LPRINT H;" ";L
```

```
10:REM NO.53
20:INPUT A,B
25:LPRINT "A,B=";
   A;" ";B
30:H=(A+B)/2+SGN
   (A-B)*(A-B)/2
40:L=(A+B)/2-SGN
   (A-B)*(A-B)/2
50:LPRINT H;" ";L
```

```
A,B= 125 397
      397 125
```

```
A,B= 62 97
      97 62
```

```
A,B= 24 -6
      24 -6
```

```
A,B= 51 -37
      51 -37
```

4. 巧用逻辑表达式

有四个数 A, B, C, D , 已知 $A > B, B > C, C > D$, 从而作出 A 是最大值这个逻辑判断时, 只要写出一行语句即可:

```
10 IF A>B AND B>C AND C>D THEN  
PRINT "MAXIMUM VALUE IS: "; A
```

因此, 用逻辑表达式写出来的程序行或段, 条理清楚, 结构合理, 层次分明, 清晰易懂。

下面是一个较复杂一点的例子。有6位同学, 他们姓名、性别(男生为 M , 女生为 F)、年龄(Y)、身长(L)、体重(W)的数据均已放在 $DATA$ 语句中, 试把符合下述所有条件的学生找出来:

- 是男生;
- 年龄在18—20岁之间;
- 身长在175—180cm之间;
- 体重在60—70kg之间。

程序 No.54 可以这样编制:

```
5: REM NO. 54  
10: C$="M"  
20: FOR I=1 TO 6  
30: READ A$, B$, Y, L  
    , W  
40: IF B$=C$ AND Y>  
    =18 AND Y<=20  
    AND L>=175 AND  
    L<=180 AND W>=60  
    AND W<=70 THEN  
    LPRINT A$; " ";
```

```

      B$:Y:L;W
50: NEXT I
70: END
75: DATA "BAI YUN"
      , "M", 16, 178, 61
      , "LUI WU", "F",
      18, 175, 63
80: DATA "HO JH", "
      F", 20, 180, 60, "
      LI HUO", "M", 21
      , 185, 70
90: DATA "LI MING"
      , "M", 19, 177, 65
95: DATA "LI HUA",
      "M", 18, 176, 66

```

```

LI MING M 19 177 6
5
LI HUA M 18 176 66

```

象这种从若干个数据中寻找所需要信息的方法，叫做“查找”，又称“检索”，它是数据处理中一项最基本的工作。

关于这个程序，我们不想过多的说明，但有一点必须提及，上面程序中的第 40 行，相当于下列程序段：

```

40 IF B# < > C# THEN 50
41 IF Y < 18 THEN 50
42 IF Y > 20 THEN 50
43 IF L < 175 THEN 50
44 IF L > 180 THEN 50
45 IF W < 60 THEN 50
46 IF W > 70 THEN 50
47 LPRINT A$; B$; Y; L; W

```


可见一条逻辑表达式代替了多个条件语句，从而节省了内存空间和执行时间。

对于判别条件同时依赖于多种关系时，我们的结论是：尽量采用逻辑表达式，以利程序的简化。

5. 循环体中判断语句的巧安排

减少无用循环和避免重复计算，是提高循环程序质量的最主要方法，其措施之一是及时与有效地安排判断控制语句，以便迅速脱离无效循环，加快程序运行速度。

请看下例：

为奖励计算机程序设计优胜者，共买了四种奖品，分别奖给一、二、三等奖和鼓励奖获得者，奖品金额分别为1.39元、1.04元、0.83元和0.59元，总共花去12元。已知奖励等级越高，得奖人数越少，试问不同等级得奖人数各是几个？

解：这是一个特殊的不定方程求解问题。设不同等级得奖的人数为 A, B, C, D ，据题意：

$$139A + 104B + 83C + 59D = 1200 \quad \langle 1 \rangle$$

$$A \leq B \leq C \leq D$$

分析题意：

由于每个得奖人数至少取1人，则当 $B = C = D = 1$ 时，方程 $\langle 1 \rangle$ 变为 $139A = 1200 - 104 - 83 - 59$ ，解之 $A \approx 7.16$ ，又因人数应为正整数，故 A 取7，同理 B 取8， C 取12， D 取14。

故有程序No.55：

```

5 REM N9.55
10 FOR A = 1 TO 7
20 FOR B = A TO 8
30 FOR C = B TO 12
40 D=(1200-193*A-104*B-83*C)/59
50 IF D<C AND D<1 AND D<>INT(D) THEN 70
60 LPRINT A;B;C;D
70 NEXT C
80 NEXT B
90 NEXT A
100 END

```

RUN

1, 1, 3, 12

上述程序最大的一个缺点是，在打印出结果后并没有立即停止，而是继续进行毫无意义的循环，一直到又经过32秒钟才停机（机种PC-1500），原因是程序中忽略了得奖人数只有且仅有一组解，因此，没有在得出正确结果后，及时跳出循环并停机。

明确上述事实，60句可改为：

```

60: LPRINT A, " , " ; B, " , " ; C, " , " ; D:
END 并删去 100 句。

```

这样，原程序执行速度加快 32 秒。

总之，在已知实际问题只有一种（或一组）解答时，当得到满意的解答后，应有效地安排控制转向语句，立即停机。这样做，极大地缩短了循环的范围，避免了大量无意义的工作，是提高速度的有效方法。

6. 多用途的 INT(X) 函数

取整函数 INT(X)，是把数值 X 取整，自动给出一个不

大于 X 的最大整数。定义简单，但应用技巧很强。现一一列举如下：

(1) 对一个数取整数部份 例如，

$\text{INT}(17.34)$ 其值为 17

$\text{INT}(-8.55)$ 其值为 -9

对取整函数，不能简单地理解“截去小数部份”。这种说法在 X 值为正数时是正确的，而在 X 为负值时不成立。这一点初学者常常弄错。

(2) 对小数部份四舍五入

$X > 0$ 时，按 $\text{INT}(X + 0.5)$ 处理

$X < 0$ 时，按 $-\text{INT}(\text{ABS}(X) + 0.5)$ 处理

如 $X = 12.5$ ，则 $\text{INT}(12.5 + 0.5) = 13$

如 $X = -12.4$ ，则 $-\text{INT}(\text{ABS}(-12.4) + 0.5) = -12$

(3) 保留或精确到小数点后第 N 位

$X > 0$ 时： $\text{INT}(X * 1E + N) / (1E + N)$ (保留 N 位)

$\text{INT}(X * 1E + N + 0.5) / (1E + N)$ (精确 N 位)

$X < 0$ 时： $-\text{INT}(\text{ABS}(X) * 1E + N) / (1E + N)$ (保留 N 位)

$-\text{INT}(\text{ABS}(X) * 1E + N + 0.5) / (1E + N)$

(精确 N 位)

注意：保留到小数点后第 N 位与精确到小数点后第 N 位在概念上是不同的，前者仅取小数点后第 N 位，不需要进行四舍五入处理；后者要进行 $+ 0.5$ 处理，才能保证要求的精度。

(4) 保留到整数部分的某一位 处理方法和保留到 X 值的小数点后第 N 位，十分类同，只要把 $1E + N$ 改为 $1E - N$ 即可。

(5) 判断整数 X 能否被整数 Y 整除 一般有: 若 $\text{INT}(X/Y) = X/Y$ 成立, 则 X 能被 Y 整除, 反之若 $\text{INT}(X/Y) < X/Y$, 则 X 不能被整除。

如有一组数 5, 9, 12, 14, 5, 17, 21, 问能否被 3 整除, 见程序 No.56:

ÜLIST

```
5  REM NO.56
10  FOR I = 1 TO 7: READ X: IF X /
    3 = INT (X / 3) THEN PRINT
    X
15  NEXT I
20  DATA 5,9,12,14,15,17,21
30  END
```

ÜRUN

9
12
15
21

(6) 求余数 求整数 X 除以整数 Y 的余数 T , 可用:

$$T = X - \text{INT}(X/Y) * Y$$

例如: 若 X, Y 均为正整数, 求 X/Y 的余数 T , 见程序 No.57:

ÜLIST

```
5  REM NO.57
10  INPUT X,Y
15  IF X = - 1 AND Y = - 1 THEN
    END
```

```

20 T = X - INT (X / Y) * Y
25 PRINT T
30 GOTO 10

```

(7) 求整数 X 是整数 Y 和 Y 的倍数 当 $X = \text{INT}(X/Y) * Y$ 成立时, 则 X 必为 Y 和 Y 的倍数。

例如, 求 $Y = (1+2+3+4) + (6+7+8+9) + \dots + (96+97+98+99)$

见程序№58:

LIST

```

5  REM    NO.58
10 Y = 0
20 FOR I = 1 TO 99
30 IF INT (I / 5) * 5 = I THEN
    50
40 Y = Y + I
50 NEXT I
60 PRINT Y
70 END

```

因原式中没有 5 和 5 的倍数, 所以在程序中的 30 句, 安排一个条件语句, 跳过 5 和 5 的倍数不加。

当然, 本题还有好的解法, 时间更短:

LIST

```

10 FOR I = 10 TO 390 STEP 20: Y =
    Y + I: NEXT I: PRINT Y

```

(8) 判断整数 X 的奇、偶性 若 $X/2 = \text{INT}(X/2)$, 则 X

为偶数，若 $X/2 < \text{INT}(X/2)$ ，则 X 为奇数。

下面是一个能同时判别数的奇、偶性的程序№59:

ULIST

```
5  REM NO.59
10  FOR I = 1 TO 10
20  READ X
30  IF X / 2 = INT (X / 2) THEN
    60
40  PRINT X;" IS A ODD NUMBER"
50  GOTO 70
60  PRINT X;" IS A EVEN NUMBER"
70  NEXT I
80  DATA 44,27,31,5,8,0,3,12,9,2
90  END
```

URUN

```
44 IS A EVEN NUMBER
27 IS A ODD NUMBER
31 IS A ODD NUMBER
5 IS A ODD NUMBER
8 IS A EVEN NUMBER
0 IS A EVEN NUMBER
3 IS A ODD NUMBER
12 IS A EVEN NUMBER
9 IS A ODD NUMBER
2 IS A EVEN NUMBER
```

(9) 找素数 素数又称质数。最易理解的一种方法是，把它除以2到 $(X-1)$ 的每一个整数，若均除不尽，则 X 必为素数。

例如，求3到52之间的所有素数，见程序№60:

ULIST

```
5  REM NO.60
10  FOR X = 3 TO 52
20  FOR I = 2 TO X - 1
30  IF INT (X / I) = X / I THEN
    60
40  NEXT I
50  PRINT X;" ";
60  NEXT X
70  END
```

URUN

3 5 7 11 13 17 19 23 29 31 37 41 43 47

(10) 产生某一范围内的随机整数 如要产生A到B之间的随机整数可用:

$$\text{INT}(\text{RND}(1) * (B - A) + A)$$

有关 INT 函数的用法和技巧, 就介绍到这里。实际上它的应用还有不少, 读者可以在实际编程中不断探索。

7. 字符串函数的应用技巧

对字符串的处理, 可以大大增强计算机的非数值处理的功能, 而且也可以用字符串函数来处理数值计算问题。

下面是几个实例。

例 1, 求一个四位数, 要求:

- 是一个完全平方数;

- 第一、二位数字，第三、四位两两相同
- 每一位数字是整数。

本题的解法很多，传统的方法都是用数值计算、判断来解题，这也比较自然，因为题意要求的就是求一个数值。而我们这里则采用一种新的解法，首先把数值变成字符串，然后用字符串函数来处理，程序No.61及结果如下：

ULIST

```

5  REM NO.61
10  FOR I = 33 TO 99 STEP 11
20  N = I * I
30  A$ = STR$ (N)
40  B$ = RIGHT$ (A$,1)
50  C$ = MID$ (A$,3,1)
60  D$ = LEFT$ (A$,1)
70  E$ = MID$ (A$,2,1)
80  IF B$ = C$ AND D$ = E$ THEN PRINT
    "N=";N;"=";I;"*";I: END
90  NEXT I

```

URUN

N=7744=88*88

上述程序比较容易理解，只需要弄清楚各字符串函数的意义。应该提出的是该程序技巧性很高，请注意第10句的安排和第80句的处理。笔者曾用多种方法（主要是用数值计算编制的程序），与上述程序对比，结果发现它是各种解法中最快的一种。

例 2，找出不超过六位数的回文数中同时又是完全平方

数（一个两位以上的数，如果左、右数字对称，就称做回文数，例如 121，12321，698896 等都是回文数）。以下是程序№62的清单和运行结果。

ULIST

```
5  REM  NO.62
10  FOR I = 11 TO 999
20  N = I * I
30  A$ = STR$ (N)
40  B$ = RIGHT$ (A$,3)
50  C$ = MID$ (A$,3,1) + MID$ (A
    $,2,1) + MID$ (A$,1,1)
60  IF B$ = C$ THEN PRINT N;" =
    ";I;" * ";I
70  NEXT I
80  END
```

URUN

```
121 = 11 * 11
484 = 22 * 22
676 = 26 * 26
10201 = 101 * 101
12321 = 111 * 111
14641 = 121 * 121
40804 = 202 * 202
44944 = 212 * 212
69696 = 264 * 264
94249 = 307 * 307
698896 = 836 * 836
```

本程序的设计思想和例1基本相同，唯一要注意的是50句是本程序的核心语句，请注意体会字符串表达式的写法和执

行顺序。

若 50 句写成,

```
50 C$ = MID$(A$,1,1) + MID$(A$,2,1) + MID$(A$,3,1)
```

会出现什么问题, 请有兴趣的读者不妨上机试一下。

8. 绝对值函数的一个应用

我们知道一些简单的数学运算, 可以代替逻辑运算。

例如, $A*B$ 代替 $A \text{ AND } B$

$\text{SGN}(A+B)$ 代替 $A \text{ OR } B$

$1-A$ 代替 $\text{NOT } A$

当然, 这里的 A 、 B 都是指逻辑变量, 其取值只有两种, 1 和 0。AND, OR, NOT 都是逻辑运算, 分别叫逻辑“与”, 逻辑“或”, 逻辑“非”。

事实上, 还有一种逻辑运算, 叫异或运算 XOR。它也可以用简单函数来代替。

异或运算有四种情况:

$1 \text{ XOR } 1 = 0$

$1 \text{ XOR } 0 = 1$

$0 \text{ XOR } 0 = 0$

$0 \text{ XOR } 1 = 1$

可以看出上述规律是变量相同, 结果为 0, 变量不同, 结果为 1。

绝对值函数也可以代替异或运算, 事实上

$$\text{ABS } (1-1) = 0$$

$$\text{ABS } (1-0) = 1$$

$$\text{ABS } (0-0) = 0$$

$$\text{ABS } (0-1) = 1$$

可见上面两组等式一一对应。

题目：A, B, C 三人在一起，A 指责 B 正在撒谎，B 指责 C 正在撒谎，而 C 指责 A 和 B 都在撒谎。试编一程序，判断究竟谁撒谎。

分析：对这个具体问题，因为只有撒谎和不撒谎两种，所以可以作一个逻辑上的假定：撒谎叫“1”，说实话叫“0”。

对每一个人来说，都有撒谎和讲真话两种可能，三个人（三个变量）就有 8 种情况，即

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

A 指责 B 撒谎，也是两种情况，若 A 说真话，那么 B 说谎；若 A 说假话，则 B 是老实人。即

$$A = 0, B = 1$$

$$A = 1, B = 0$$

把这种情况可以用 $\text{ABS}(A - B) = 1$ 来综合。

同理： B 指责 C 撒谎，可以表示为： $ABS(B - C) = 1$ ，而 C 指责 A 和 B 都在撒谎，则可以表示为： $ABS(C - A * B) = 1$

现在，可以安排程序了，由于 A, B, C 都有撒谎、不撒谎两种可能，八种情况，所以容易想到用三重循环来解决。满足题意的三个条件用条件语句来判断，为使程序更加简洁，用逻辑表达式来表示这三个条件。程序No.63清单和运行结果如下：

ULIST

```
5  REM NO.63
10  FOR A = 0 TO 1: FOR B = 0 TO
    1: FOR C = 0 TO 1
20  IF ABS (A - B) = 0 OR ABS (
    B - C) = 0 OR ABS (C - A *
    B) = 0 THEN 60
30  IF A = 1 THEN PRINT "A"
40  IF B = 1 THEN PRINT "B"
50  IF C = 1 THEN PRINT "C"
60  NEXT C: NEXT B: NEXT A
```

URUN

A
C

可见真正的说谎者是 A 和 C 。

本程序主要讲了 $ABS(X)$ 的一个特殊应用，它可以代替逻辑上的异或 XOR 运算，为在有逻辑判断的程序中，提供了方便。同时，我们还应看到，上述问题用人来进行判断还

是比较困难的，可是利用计算机来代替人进行逻辑判断和推理却是可以的。事实上，计算机具有一定的逻辑思维能力，也许这就是“电脑”的由来吧！因此，在以后的编程中，可以根据题目的要求，在有可能利用计算机协助我们编程时，尽可能巧用计算机，这会减少许多计算工作和一些复杂问题的判断及分析。特别是实际问题往往是很复杂的，倘有多种变量，多个相互制约的关系，人的智力不可能一下子涉及到问题的全部细节，常常难于做出判断，所以遇到这类问题，最好求助于计算机帮忙。

9. 多重循环程序的设计技巧

众所周知，由于计算机具有快速准确及逻辑判断的特点，几乎所有的高级程序设计语言都设置了循环语句，用以有效地解决大量的在处理形式又完全相同的重复性计算问题。对多重循环来说，这种重复计算的工作量是十分巨大的，多重循环最内层循环体的执行次数，一般来说是各层循环次数连乘积之值，由于这个数字往往非常庞大，所以对多重循环来说，十分注意减少不必要的循环次数，尽一切可能避免毫无意义的重复工作，特别是如有可能尽量减少循环嵌套的层次，就是一个有实际意义的工作。这样，我们就可能使计算机执行较少的命令，干出更多的工作，从而达到节约时间、少占存贮、加快速度、提高效率的目的。

对于嵌套的循环来说，外层循环的控制变量取一定数值后，内层循环则要整整转上一圈，只要内层循环不结束，就谈不上外层循环控制变量的增值和判断。这种循环语句的执行过程，和时钟的转动规律有十分相似的地方，以三重循环

为例，三层循环的关系好象手表的时针、分针与秒针，时针移动一格，分针转动一圈，分针移动一格，秒针走动一周。但二者也有差异，时钟上足发条，指针的移动都是做的有用的工作，它在一秒一秒地计时；而程序的循环呢？并非都是执行有效的计算，事实上，常常发生不可能出现我们所希望的结果时，计算机还在继续进行毫无意义的循环。

这里从一个具体实例出发，谈谈多重循环的设计技巧。

用100元买100只鸡，已知每只小鸡、公鸡、母鸡分别为0.5元、2元、3元，问各种鸡至少买一只一共有多少种买法。

解：设 A, B, C 分别为购买小鸡、公鸡和母鸡的只数，据题意有以下方程：

$$A + B + C = 100 \quad \langle 1 \rangle$$

$$0.5A + 2B + 3C = 100 \quad \langle 2 \rangle$$

而约束条件为： $1 \leq A, B, C \leq 98$

我们可以用循环的方法，取 A, B, C 的各种可能取值，去投试上述方程，只有全部满足方程 $\langle 1 \rangle$ 和 $\langle 2 \rangle$ 的 A, B, C 值，才是问题的解，故容易写出程序No.64：

ULIST

```

10  REM NO.64
20  FOR A = 1 TO 100
30  FOR B = 1 TO 100
40  FOR C = 1 TO 100
50  IF A + B + C < > 100 THEN 80

60  IF A + 4 * B + 6 * C < > 200
    THEN 80

```

```

70 PRINT A;B;C
80 NEXT C
90 NEXT B
100 NEXT A
110 END

```

程序 №64 总的循环次数高达 100 万次 ($100 \times 100 \times 100 = 10^6$)，所以运行时间是很长的，在 PC-1500 机上要运行 13 个多小时，在 APPLE- II 机上要运行 2 个多小时，而在 WANG PCS- II 和 WANG 2200T (均为美国计算机)，运行了 4 个多小时。

程序 №64 并没有原则性错误，因为它确实可以得到正确的结果，为了更快地得到答案，显然还有值得探讨的问题。

第一个是 A, B, C 的终值是否都是 100?

程序 №64 中的三个循环变量 A, B, C 的终值均为 100，是没有考虑每只鸡至少买一只这个约束条件编写的。事实上，各种鸡至少各买一只，每种鸡最多只能买 98 只。所以三个循环变量的终值都只要取 98 即可。对于每一个单循环来说，终值仅仅减小了 2，好象微不足道，但对多重循环来说，它的意义就非同小可。因为各重循环终值按 98 计算，总的循环次数变为 $98 \times 98 \times 98 = 941192$ 次，比程序 №64 循环次数减少了 58808 次。也就是说，减少了近 6 万次的无效循环。

第二个是改写一下循环变量的终值：

循环变量的初值、终值和步长可以是具体的数值，也可以是已赋值的变量，或者是表达式。从这个基本约定出发，

把三重循环的终值改写一下:

```
20 FOR A = 1 TO 98
30 FOR B = 1 TO 99 - A
40 FOR C = 1 TO 100 - A - B
```

显然, 上述程序段, 和三重循环变量都从 1 变化到 98, 是等价的。

注意: 本程序段写法比较特殊, 它是考虑到 A, B, C 三个变量均应满足 $A+B+C=100$ 和 $1 \leq A, B, C \leq 98$ 这两个条件的。但在执行的效果上, 却有很大的不同。

为了说明这个问题, 我们先考察一下程序 No. 65 和 No. 66:

LIST

```
5 REM NO.65
10 N = 0
20 FOR A = 1 TO 5
30 FOR B = 1 TO 5
40 FOR C = 1 TO 5
50 N = N + 1
60 NEXT C
70 NEXT B
80 NEXT A
90 PRINT "N=";N
```

LIST

```
5 REM NO.66
10 N = 0
20 FOR A = 1 TO 5
30 FOR B = 1 TO 6 - A
40 FOR C = 1 TO 7 - A - B
50 N = N + 1
```



```

60  NEXT C
70  NEXT B
80  NEXT A
90  PRINT "N=";N

```

上述两个程序假设都满足：

$$A + B + C = 7$$

$$1 \leq A, B, C \leq 5$$

这两个条件。

由于多重循环最内层循环体的执行次数，是各层循环次数连乘积之值，所以程序№65，总共执行循环 $5 \times 5 \times 5 = 125$ 次，程序的 90 行是统计打印循环总次数的。

对于程序№66，根据多重循环的执行过程，不难计算出总的循环次数只有 35 次。它仅为程序№65 循环次数的 28%。

由此可知，对本程序来说，循环终值的写法是大有讲究的。

现在我们再回到本题上来，改用程序№67，求解本题，

LIST

```

10  REM  NO.67
20  FOR A = 1 TO 98
30  FOR B = 1 TO 99 - A
40  FOR C = 1 TO 100 - A - B
50  IF A + B + C = 100 AND A + 4 *
    B + 6 * C = 200 THEN PRINT
    A;B;C
80  NEXT C: NEXT B: NEXT A

```

程序 №67 比程序 №64 写得更为简洁，但最主要的是由

于把循环终值形式上改写了一下，总循环次数已由 №64 的 100 万次，改变到只有 161700 次，它仅为程序 №64 循环次数的 16.17%，显然，这是一个大的飞跃，它省去了 838300 次无用循环。

从上述事实，可以看到多重循环的工作量是非常巨大的，而缩短初值和终值之间的差距是减少循环次数的一个有效方法。当然，这种减少，必须由题意分析确定，不能任意减少。不同的题目，可以用增大初值、减少终值或两者兼而有之，来达到减少循环次数的目的，上例是用减少终值的方法来实现的。至于本例改写终值的形式，是受本题不定方程求解这个特殊条件制约的。

第三个是三个循环的终值也不可能都是 98，

这个问题也是非常明显的，若买 98 只小鸡则共花去 $98 \times 0.50 = 49$ 元，再买母鸡、公鸡各一只，又分别花去 3 元和 2 元，虽然买了 100 只鸡，但只花了 $49 + 3 + 2 = 54$ 元，不满足“百钱买百鸡”的题意，同理母鸡、公鸡也不可能买到 98 只，否则 100 元钱不够花。

让我们进一步思考，看看还有什么方法，来减少循环次数。

改写方程〈1〉和〈2〉

$$A + B = 100 - C \quad \langle 3 \rangle$$

$$A + 4B = 200 - 6C \quad \langle 4 \rangle$$

〈3〉式 $\times 6$ - 〈4〉式得：

$$5A + 2B = 400 \quad \langle 5 \rangle$$

〈4〉式 - 〈3〉式得：

$$5C + 3B = 100 \quad \langle 6 \rangle$$

〈3〉式 $\times 4$ - 〈4〉式得:

$$3A - 2C = 200 \quad \langle 7 \rangle$$

改写〈5〉、〈6〉、〈7〉三式则有:

$$\left\{ \begin{array}{l} B = \frac{400 - 5A}{2} \end{array} \right. \quad \langle 8 \rangle$$

$$\left\{ \begin{array}{l} C = \frac{100 - 3B}{5} \end{array} \right. \quad \langle 9 \rangle$$

$$\left\{ \begin{array}{l} A = \frac{200 + 2C}{3} \end{array} \right. \quad \langle 10 \rangle$$

表面上原来两个方程〈1〉、〈2〉,变成了三个方程〈8〉、〈9〉、〈10〉,多了一个方程,但经过这样处理,却使问题简单多了,因为每个方程只是一个变量依赖于另一个变量的单一关系。

由〈8〉式, B 最少取 1 时 (根据题意), $A = 79.6$, 取整数 79;

由〈10〉式, C 最少取 1 时, $A = 67.3$, 取整数 67。

故可知 A 的变化范围: $67 \leq A \leq 79$

再考察方程〈2〉, 当 A 为奇数时, 不满足方程的解, 所以 A 必为偶数, 这样, 可知 A 的取值范围是 68 到 78, 且步长为 2。

这个事实告诉我们, A 的终值不是 98, A 的初值也不是 1, 并且 A 的变化也是有规律可寻的, 它的间隔是 2。

读者不难想到, B 和 C 的初、终值又是什么? 它们是否也按一定的规律变化呢? 这个问题, 还是让计算机帮助我们判断吧!

第四个是计算机可以帮助找到变量的取值范围和变化规

律:

A 的变化范围确定了, 代入(8)式, 则可确定 B 的变化范围; 同理 B 的数值确定了, 代入(9)式, 也可确定 C 的变化范围。

现在, 编两个小程序№.68和№.69, 让计算机自动求解:

ÜLIST

```
5 REM NO.68
7 PRINT "B=";
10 FOR A = 68 TO 78 STEP 2
20 B = (400 - 5 * A) / 2
30 PRINT B; ", ";
40 NEXT A
```

ÜRUN

B=30, 25, 20, 15, 10, 5,

由此可知, B 的变化范围是从 5 到 30, 且步长为 5。

ÜLIST

```
5 REM NO.69
7 PRINT "C=";
10 FOR B = 5 TO 30 STEP 5
20 C = (100 - 3 * B) / 5
30 PRINT C; ", ";
40 NEXT B
```

ÜRUN

C=17, 14, 11, 8, 5, 2,

程序结果表明, C 的取值范围是从 2 到 17 而步长为 3。

综合 A 、 B 、 C 的取值范围和步长变化规律, 可以写出程序№70:

ULIST

```
10  REM NO.70
20  FOR A = 68 TO 78 STEP 2
30  FOR B = 5 TO 30 STEP 5
40  FOR C = 2 TO 17 STEP 3
50  IF A + B + C = 100 AND A + 4 *
    B + 6 * C = 200 THEN PRINT
    A,B,C
60  NEXT C: NEXT B: NEXT A
```

读者不难发现, 程序№70循环次数共有 125 次, 仅为程序№64 循环次数的万分之一点二五。显然, 本程序的运行速度是很快的, 在 PC-1500 机上仅运行 20 秒就得出全部结果。

由此可知, 同时增大初值和缩小终值, 并且加大步长值, 会使程序效率大大提高, 它们是加快程序循环部分执行速度的又一个好办法。

同时, 我们还应看到, 如果没有计算机的“帮助”, 很快地作出上述三个变量的取值范围及变化规律的判断, 还是比较困难的。在程序设计中, 利用计算机协助人们编程, 也是一个值得研究的课题。

第五个是循环嵌套的重数可以减少:

减少循环的层次, 使循环嵌套愈少愈好, 这样, 便可以进一步提高程序的运行速度。

分析程序No.70, 用 $C = 100 - A - B$ 来代替40句, 显然是可以的, 因为当 A 和 B 的值确定后, C 的值也就确定下来, 故可以省去一重循环, 这样总的循环次数只有25次了。同样, 我们从方程〈8〉、〈9〉、〈10〉中还看到这样一个事实, 每个变量只依赖一个变量, A 定 B 也就定了, 而 B 定 C 也定了, 因而完全可以再省去一重循环, 即删去30行的一重循环, 而改用 $B = (400 - 5 * A) / 2$ 的赋值语句。省去二重循环后, 循环次数只有5次, 它仅为No.64循环次数的百万分之五, 可见运行时间会进一步缩短, 而速度则会大大提高, 见程序No.71:

ULIST

```
10  REM  NO.71
20  FOR A = 68 TO 78 STEP 2
30  B = (400 - 5 * A) / 2
40  C = 100 - A - B
50  IF A + B + C = 100 AND A + 4 *
    B + 6 * C = 200 THEN PRINT
    A,B,C
60  NEXT A
```

第六个是循环体的判断语句可以省去:

经过上面好几步的简化, 我们得到程序No.71, 好象该做的我们都做了——循环初值增大、终值减小、步长增加、循环层次又减少了, 但作为循环程序的设计技巧, 就本题来说, 循环体中的内容还可以斟酌。由于 A 、 B 、 C 之间关系完全由方程〈8〉、〈9〉、〈10〉所制约, 并且在处理上述工作时, 又已经考虑了约束条件, 所以, 程序No.71 中循环体内的判断语

句完全可以省去。因而有程序No.72:

ULIST

```
10 REM NO.72
20 FOR A = 68 TO 78 STEP 2: B = (
    400 - 5 * A) / 2: C = 100 - A
    - B: PRINT A, B, C: NEXT A
```

URUN

68	30	2
70	25	5
72	20	8
74	15	11
76	10	14
78	5	17

这个程序不仅简短节省内存，而且运行速度又是所有程序中最快的一个。在PC-1500机上只运行了几秒钟，而在APPLE-Ⅱ机上执行时间仅为1秒钟左右。

综上所述，我们围绕一个不定方程的求解过程，探讨了循环程序设计技巧的各个方面。虽然本题是从一个具体例子出发，但引出的结论却具有普遍的意义。循环是BASIC语言中的精华和核心，编程序几乎离不开循环，因此，掌握循环程序的设计技巧，无论对学习那一种高级语言都是重要的。循环程序的设计技巧可以归纳以下五个方面：在步长不变的情况下，减小初值和终值的差距是减少循环次数的有效方法；在初、终值一定的情况下，加大步长可以大大提高程序的效率；减少循环嵌套的重数，是提高程序运行速度的关键；调整和精简循环体的语句，是提高程序质量的有力措施；合理

安排控制和转向语句,是及时脱离无效循环避免重复计算的前提。而所有这一切,又以尽量减少无用循环次数,努力避免大量无意义的重复工作为根本,这是高质量进行循环程序设计的核心。

参 考 文 献

- [1] 张世英编,《苹果-II BASIC 程序设计》,北京师范大学出版社,(1985)。
- [2] 谭浩强、田淑清、谢锡迎编著,《BASIC 语言》,科学普及出版社,(1985)。
- [3] 吴志洪、吴洪森编著,《程序设计实用技巧》,浙江大学出版社,(1986)。
- [4] 王飞龙主编,《苹果-I 微型计算机和结构化 BASIC 语言编程》,湖北科学技术出版社,(1984)。
- [5] 那莫西主编,《BASIC 语言简明教程》,南开大学出版社,(1984)。
- [6] 李冠英、张毅忠、郑存陆编,《APPLE-II 微型计算机 BASIC 语言与磁盘操作系统》,广东科技出版社,(1984)。
- [7] LEARNING BASIC FAST, Claude J. De. Rossi, (1974)。



海淀走读 0023490

011411

前言

目录

前言

一、什么是程序设计和技巧

- 1．什么是程序设计
- 2．各种情况
- 3．质量标准
- 4．出发点
- 5．基本方法
- 6．基本技巧
- 7．实例分析

二、程序设计步骤和方法

- 1．了解问题性质
- 2．绘制程序框图
- 3．强调设计思想
- 4．建立数学模型
- 5．选用计算方法
- 6．注意近似处理
- 7．考虑通用合理
- 8．防止程序出错
- 9．划分功能模块
- 10．讲究设计技巧

三、程序连接的方法

- 1．如何安排总控
- 2．单一程序的连接方法
- 3．两个不同程序的连接方法
- 4．重订行号的方法

四、节省内存和提高速度的方法

- 1．程序存贮的基本知识
- 2．节省内存的措施
- 3．提高速度的方法
- 4．使用机器语言子程序

五、编制程序应注意的问题

- 1．关于变量的初始化问题
- 2．数组元素的下标要恰当
- 3．程序的正确性要经得起时间的考验
- 4．防止陷入死循环
- 5．矩阵输出注意打印语句的安排
- 6．妥善处理保留字
- 7．重新读数不要忘记RESTORE

- 8 . 能用简单变量的就不用下标变量
- 9 . 注意程序的清晰
- 1 0 . 判断循环变量是否改变
- 1 1 . 数值计算不要超过机器规定范围
- 1 2 . 注意使用扩展 B A S I C 语句
- 1 3 . 尽量不用 G O T O 语句
- 1 4 . 尽量少用乘方运算

六、打印输出的技巧

- 1 . 输出空格函数 (S P C 函数) 的使用
- 2 . 输出定位函数 (T A B 函数) 的使用
- 3 . 数据的列印输出
- 4 . 打印机输出控制
- 5 . 打印输出结果的对齐处理
- 6 . 字符的放大打印

七、程序调试技巧

- 1 . 不同机器的差异
- 2 . 尽量避开乘方运算
- 3 . 作适当的精度调整
- 4 . 如何处理小数步长
- 5 . 屏幕抄写的方法
- 6 . 程序编辑举例
- 7 . 执行过程的自动跟踪
- 8 . 输入数据的跟踪检查
- 9 . 几个常用技术

八、B A S I C 语言的编程技巧

- 1 . 菜单技术
- 2 . 人机对话方式
- 3 . 巧用符号函数 S G N (X)
- 4 . 巧用逻辑表达式
- 5 . 循环体中判断语句的巧安排
- 6 . 多用途的 I N T (X) 函数
- 7 . 字符串函数的应用技巧
- 8 . 绝对值函数的一个应用
- 9 . 多重循环程序的设计技巧

参考文献