〔美〕 D·A·利恩 著

BASIC大全

谭浩强 梁延广 薛淑斌 编译

BASIC大全

科学普及出版社

统一书号：7051·1082
定　　价：　4.80元

封面设计：赵一东

BASIC 大全

〔美〕David A. Lien 著

谭浩强 梁延广 薛淑斌 编译

科学普及出版社

BASIC 大全

〔美〕David A. Lien 著

谭浩强 梁延广 薛淑斌 编译

科学普及出版社

内 容 简 介

这是一本供查询用的BASIC手册，它包括了目前在国际上使用较多的二百五十多种计算机上所用的BASIC语言中的语句、命令、函数和运算符近五百个。对它们的含义、作用、使用方法作了简明的介绍，并且通过测试程序及运行结果说明它们的功能，使读者一目了然。此外还介绍了不同的BASIC之间的转换和替代的方法。如果初步学习了BASIC语言，再有这本大全，就可以很容易掌握各种BASIC的使用。解决各种BASIC“方言”间不通用所造成的困难。本书中每一个条目（字或符号）都是相对独立的，按字母顺序排列，可以根据需要单独查阅某一条目，而不必顺序阅读全书。

本书可以作为BASIC语言的参考读物，可供大中学校师生、计算机程序工作人员、计算机站工作人员及其它学习或使用BASIC的读者使用。具有中学文化程度的读者即可看懂本书的基本内容。

David A. Lien

The BASIC Handbook

Encyclopedia of the BASIC Computer Language

（2nd Edition）

COMPUSOFT® PUBLISHING

A Division of CompuSoft, Inc.

1981

BASIC大全

〔美〕David A. Lien 著

谭浩强　梁延广　薛淑斌　编译

科学普及出版社出版

（北京海淀区魏公村白石桥路32号）

新华书店北京发行所发行　各地新华书店经销

沈阳新华印刷厂印刷

开本：787×1092毫米　1/16　印张：21.5　字数：470千字

1985年9月第1版　1986年4月第1次印刷

责任编辑：王曙斌　荆芷萍　封面设计：赵一东

统一书号：7051·1082　本社书号：1113　定价：4.80元

前　言

BASIC语言已经广泛地在国内外各个领域中使用，但是由于过去一个时期中，BASIC的标准化程序较差，许多厂家都各行其是地制定出他们自己的BASIC语言版本，这就形成了各种BASIC“方言”。虽然在1978年美国提出了一个BASIC标准（ANSI BASIC，它包括了BASIC的核心部分）。但是已经形成的各种BASIC“方言”并未因此而消失，各种新的BASIC版本仍然继续出现。尽管各种版本的BASIC的基本核心部分是相同的或接近的，但是有的差别也特别大。有时同一个字或符号在不同的BASIC中有着完全不同的含义，反过来，为了完成某一个操作，在不同的BASIC中所用的字（语句，函数或操作符）又往往是不同的。例如，A$（1，3）在一些计算机BASIC中表示字符串A$中的最左边三个字符，而在另一些计算机中用它表示二维的字符串数组中的一个元素（第一行第三列元素）。从一个字符串P$中取最左边5个字符，有的用P$（1，5），而有的计算机却用LEFT$(P$，3)。为了在打印机上输出数据，有的用PRINT，而有的却用LPRINT，还有的用PRI等等。因此，在一种计算机上调试通过的BASIC程序在另一种计算机上往往不能运行，需要对程序作必要的修改。

由于各种BASIC的不一致性，在教学中不得不指定以某一种计算机的BASIC为基础。然而实际上国内各学校、各单位所用的计算机种类繁多，这就在学习和应用之间出现了一道鸿沟。在学习教科书后还要去学习所用计算机的BASIC说明书，并了解二者间的差别和掌握它们间的转换方法。任何一本教科书都不可能概括所有的（或主要的）BASIC的全部功能、特点和规则的细节。同样，即使一个有丰富经验的程序设计者也不可能对几百种计算机的BASIC了如指掌。这种情况往往使人们困惑。这也是使从事教学工作和实际工作的同志感到头痛的一个问题。

为了适应实际工作的需要，我们根据美国David A. Lien博士的“The BASIC Handbook” — Encyclopedia of the BASIC Computer Language”（第二版）编译了这本书。这是一本百科全书式的计算机BASIC语言手册，它包括了国际上比较流行的二百五十多种计算机所用的BASIC版本中用到的近五百个BASIC字和符号，对每一种BASIC中用到的字（如语句、命令、函数）或符号（如操作符）分门别类地整理出来，对每一个BASIC字和符号在不同的BASIC中的含义和用法都作了说明、分析和比较，同时还指出了不同的BASIC之间的转换方法（即在其它BASIC中用什么方法来实现相同的功能）。

有了这样一本书，就提供了很大的方便。它不象一些计算机说明书那样晦涩难懂而又枯燥无味，在本书中对每一个BASIC字（或符号）都作了扼要而明确的说明，并且提供了“测试程序”和“运行实例”，生动而易懂。一般有BASIC初步知识的人都能很容易地理解和掌握它。可以象查字典一样在几分钟之内就知道一个语句、命令或函数的含义和使用方法。如果想从一个计算机向另一计算机移植BASIC程序，有这本书的帮助，是很容易实现的。

这本书不是BASIC教科书，也不能代替每一种计算机的BASIC说明书，它是一本资

料性的参考书，可以和BASIC教科书配合使用。它可以帮助BASIC语言的使用者解决数百种BASIC方言的存在而引起的程序不相容的难题。读者可以从本书找到各个BASIC字或符号在各计算机上的不同含义和用法，如果你拿到一个BASIC程序，它包含一些其它计算机的BASIC所不具备的功能，你可以从本书中方便地找到怎样修改成适合你的计算机的方法（考虑到不同的计算机的情况，在每一条目中都提出了可能的几种方案，以供读者选择试验）。

关于本书的内容和编写方法有几点要说明一下：

为了节省篇幅，对极个别的计算机所用的BASIC中个别很少用到的字和符号未收入本书。此外，由于新的计算机每年每月不断地问世，本书可能未包括某些最新出现的BASIC中的一些内容。在本书的最后单独印出几种计算机BASIC的有关规则和使用方法。

由于磁盘BASIC用到文件管理，而文件管理在各种不同的计算机间差别很大，因此我们只在书的后部专设一节“关于磁盘BASIC简要介绍”，提供最基本的知识，在前面的BASIC字和符号中不包括磁盘BASIC的语句、命令和操作符。

各种BASIC“方言”间另一个较大的差别是字符串的处理方法。有的BASIC要求在使用字符串前先定义字符串的长度（用DIM语句），并用下标处理子字符串，如C$（3，5），代表C$中第三到第五个字符。而其它的计算机则使用LEET$，MID$，RIGHT$等函数处理子字符串。因此你应当知道你的计算机是属于哪一种方法。如果实在不知道的话，可以使用本书中提供的“测试程序”试验一下便可确定。

为了便于查阅，本书中列出的BASIC字和符号是互相相对独立的，即每一条目都可单独阅读（也便于理解），而不必从本书开头顺序往下读。为了使每一条目都能独立地完整地被使用，有些条目间的少量叙述会有些不可避免的重复。

考虑到本书的对象是已经初步学过或正在学习BASIC的读者，为了不使篇幅过多，在本书中不列入BASIC的最基本的知识和具体的语法规则。它只是从“使用”的角度帮助读者更快地掌握不同BASIC的特点和转换。已经学过或正在使用BASIC的读者（包括计算机程序工作者、教师、学生、科技工作者、管理人员）可以通过本书了解各种BASIC的特点，加深对BASIC的了解，而不致只局限于所用的一种计算机上。这对于开阔眼界、扩大知识面是有好处的。

各种计算机的上机操作步骤和键盘使用方法（包括各功能键的使用）不属于BASIC的范围，没有列入。每一种计算机都有自己特定的操作步骤，无法一一列举。

本书最后有一个“索引和记录卡片”是为读者提供方便而设。它既可以作为读者查阅有关条目的索引（目录），同时也可以记录所用的计算机对每一条目中“测试程序”的反应（成功或失败），以便对所用计算机的BASIC有更多的了解。

参加本书编译工作的还有电子工业部十五所王洪征、暨南大学谭浩邦。谭浩强负责全书的校阅定稿，王曙斌、荆芷萍、秦守雍参加了校阅。我们期望本书的出版能对计算机的使用和普及起一定促进作用。由于我们水平不高，可能有不妥之处，请读者批评指正。

编译者 1984．11

目　　录

使 用 说 明

我们把每个BASIC字的解释说明分为若干个部分。通过下面的例子（SEG$）来说明手册中各个部分的含义。请把下面各点与第7页中的各点对照着看，即①对①，②对②……等等。

① BASIC字本身：它是BASIC程序中的字或是用来控制程序的字。那些用于整个系统监控程序、编辑语言和其它计算机语言中的字不是这本BASIC手册所讨论的部分。

Ⓐ 其它字：字的缩写和可替换的形式。

② ANSI标准记号：如果这里出现“ANSI”这个字，这就意味着这个字是美国国家标准协会制定的最小BASIC字表中的字。

③字的类型：BASIC字可以分成四种类型：

命令：它指示计算机对用程序所要进行的操作，如RUN、LIST等等。某些计算机允许命令放入程序中，因此，也可以起到一个语句的作用。

语句：它是真正出现在程序中并包括一些详细指令的字，计算机根据这些指令做出判断并且执行其功能。例：

```
PRINT A, B, C
```

函数：它是调用外部预先设计好的机器的“微程序”的BASIC字，它们可以执行相对复杂的“函数”，如，求三角值、平方根等等，也可以作为长语句的一部分。例：

```
LET X=TAN(Y)      PRINT LOG(A)
```

操作符：它是在特殊比较或修改功能中起作用的非文字符号，例如：逗号、冒号、等号等等。

在这本《BASIC大全》中，命令、语句和函数按字母顺序排列，那些不以字母开头的操作符放在最后。

④有关的介绍和描述，它告诉我们字是什么，它是干什么的，它还指出这个BASIC字主要是为哪几种计算机使用的，或是为某一计算机专用的。

⑤测试程序：它允许用户把一简短的程序输入到一台计算机中，用来查看它的解释程序或编译程序是否能识别该语句，并且使用它。

⑥运行实例：它表明计算机如何对测试程序作出响应。其结果因机器不同稍有差异，但是，一般情况下不应与运行实例差别太大。

⑦提示：经常有许多程序设计技术，它们大大地简化了程序设计，从而得到高度的简单性和可靠性，如果有这种情况，在这里把它指出来。

⑧替代的形式：当不同的计算机使用不同的形式时，在这里指出可替换的形式。

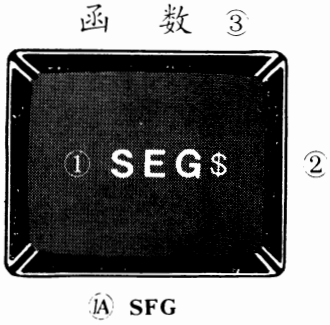
⑨如果你的计算机没有此功能：在可能……和不总是可能时，它给出用其它BASIC字来完成相同目的的可替换的形式。在函数的情况下，通常包括一个子程序，它可弥补由于缺少某个内部函数的缺陷，在语句的情况下可以由用其它字或其它方法的一部分程序来代替，使得在程序执行时得到相同的（或基本相同的）结果。

⑩其它用法：指字在使用中的变化，即字本身在不同的计算机上的不同使用形式

（而不是介绍如何用其它的字来代替这个字来获得所希望的结果的这种“变化”）。

⑪参阅：有些字是相似的或互相关联的，其中有一个字是“中心字”（即以它为主）。为了避免重复，我们只对这个“中心字”进行详细的讨论。而对其它有关的字只介绍其特殊的部分。想要更详细的了解，可以参阅有关的其它字。

④ SEG$函数的作用　是从一个字符串变量中选取一部分。SEG$具有三个自变量：字符串变量；字符串中的起始位置和子符串的字符个数。



Ⓐ SFG

例如：

```
IF A$="COMPUTER", THEN PRINT
SEG$(A$,4,3)
```

打印出 PUT.

⑤ 测试程序

```
10 REM * SEG$ TEST PROGRAM *
20 A$="CONTESTANT"
30 B$=SEG$(A$,4,4)
40 IF B$<>"TEST" THEN 70
50 PRINT "SEG$ PASSED THE ";B$
60 GOTO 99
70 PRINT "SEG$ FAILED THE TEST"
99 END
```

⑥运行实例

```
SEG$ PASSED THE TEST
```

⑦SEG$可以起到LEFT$和RIGHT$的作用。SEG$（A$，1，4)与LEFT$（A$，4）等效，而SEG$（A$，LEN（A$）－3,3）等效于RIGHT$（A$,3)。

⑧替代的形式

少数计算机使用SEG。

⑨如果你的计算机没有此功能

如果你的计算机既不能使用SEG$，也不能使用SEG，可在测试程序中试一下MID$。如果MID$也不能用，那末，对于需要用DIM语句定义所有字符串的计算机（例如，Hewlett—PacRard,）会接受A$(4，7)，以得到A$中从4到7位置的子字符串。

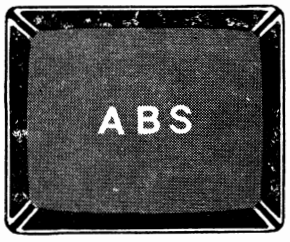
⑩其它用法

没有

⑪参阅

MID$, LEFT$, RIGHT$, DIM

函　　数



A.

ABS函数可以确定一个数值或数值变量的绝对值。一个数的绝对值，就是不带±号的数本身的值。

例：PRINT　ABS（－10）等价于PRINT　10.

ABS函数可以处理在计算机解释程序限度内的任意数（大数或小数）。

测试程序＝1

```
10 REM 'ABS' TEST PROGRAM
20 X=35
30 PRINT "ABS PASSED THE TEST IF";
40 PRINT ABS(-435.28);
50 PRINT ABS(-.03245);
60 PRINT ABS(-X)
70 PRINT "ARE ALL PRINTED AS POSITIVE VALUES."
99 END
```

运行实例

```
ABS PASSED THE TEST IF 435.28 .03245 35
ARE ALL PRINTED AS POSITIVE VALUES.
```

大多数解释程序也允许在算术运算中使用ABS函数。取绝对值这种特性对于需要从数学运算中得到正值的程序是重要的，否则就会产生负值。

ABS后面的整个数学运算，必须放在括号内。

测试程序＝2

```
10 REM 'ABS' MATH OPERATION TEST PROGRAM
20 A=18
30 B=58
40 PRINT "THE ABSOLUTE VALUE OF";(A-B)/2;"IS";ABS((A-B)/2)
99 END
```

运行实例

```
THE ABSOLUTE VALUE OF -20 IS 20
```

替代的形式

某些计算机可以用A.来代替ABS函数。

如果你的计算机没有此功能。

你可以很容易地用下面的子程序来代替。

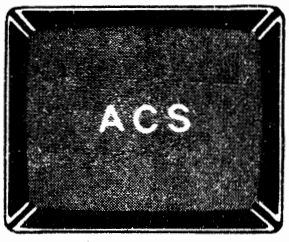
测试程序=3

```
10 REM 'ABS' SUBROUTINE TEST PROGRAM
20 PRINT "ENTER A NEGATIVE NUMBER";
30 INPUT X
40 GOSUB 30010
50 PRINT "THE ABSOLUTE VALUE OF";X;"IS";Y
60 GOTO 20
30010 REM * ABS(X) SUBROUTINE * INPUT X, OUTPUT Y
30012 Y=X
30014 IF X>=0 THEN 30018
30016 Y=-X
30018 RETURN
30999 END
```

运行实例（输入－35.5）

```
ENTER A NEGATIVE NUMBER? -35.5
THE ABSOLUTE VALUE OF -35.5 IS 35.5
ENTER A NEGATIVE NUMBER?
```

函　数



AC.
ACSD
ACSG
ARCOS

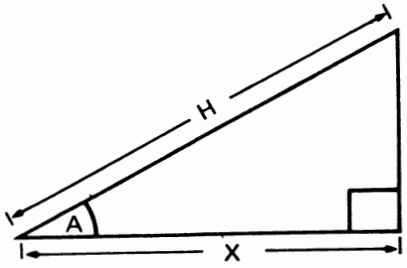
在某些BASIC中，ACS（n）函数用于计算ARCCOS值，角度单位是弧度（而不是度数）。一个弧度的值大约为57度。

Arccos(ACS)定义为直角三角形的角（A），它是斜边为H和直角边为X的夹角。

A = ACS(X/H)

ACS的反函数是COS(COSINE)。一个角（这个角的量度是A弧度）COSINE的值是该角的直角边除以斜边所得的数。

COS(A) = X/H



测试程序

```
10 REM 'ACS' TEST PROGRAM
20 PRINT "ENTER A COSINE VALUE (-1 TO 1)";
30 INPUT C
40 W=ACS(C)
50 PRINT "THE ANGLE WITH THE X/H RATIO OF ";C;" IS ";W;
   "RADIANS"
30999 END
```

运行实例

```
ENTER A COSINE VALUE (-1 TO 1)? 0
THE ANGLE WITH THE X/H RATIO OF 0 IS 1.5708 RADIANS
```

为了把值从弧度转换为度数，可以把这个角的度数(以弧度为单位)乘以57.29578。例如： 40 W = ACS(C) * 57.29578

某些计算机可以用度数或梯度来计算角的值（100梯度 = 90度)。这些计算机如果用度数为单位，则要使用ACSD函数，用梯度为单位则用ACSG函数。在上面这个实例运行中，用 0 代入程序行40中，其结果为90°，也就是100梯度。

替代的形式

Sinclair ZX80使用ARCOS，而SHARP（夏普）1211（TRS—80袖珍计算机）使用 AC.。

如果你的计算机没有此功能

如果你的计算机不接受测试程序中的40行，但是能接受ATN（arctangent，反正切）和SQR（square root，平方根），可以用下面语句代替：

```
40 W=1.5708-2*ATN(C/(1+SQR(1-C*C)))
```

如果你的计算机没有ATN或SQR函数，那么可以用下面的子程序来代替。但一定要把ASN函数下面的程序（见本书12页）和下面这个子程序一起使用。

```
30000 GOTO 30999
30500 REM * ARCCOS SUBROUTINE * INPUT C, OUTPUT W
30502 REM ALSO USES VARIABLES S, X, Y AND Z INTERNALLY
30504 S=C
30506 GOSUB 30520
30508 W=1.570796-W
30510 RETURN
```

这样，测试程序中的40行可以改为：

```
40 GOSUB 30500
```

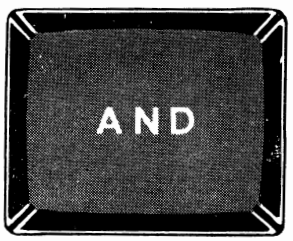
为了使ARCCOS子程序用度数来表示角，需要再加上下面一行：

```
30509 W=W*57.29578
```

参 阅 COS，ASN，ATN，SQR，SIN，TAN

操作符

语　句



A.

AND用于“IF—THEN”语句，作为“逻辑运算”操作符。

例如，IFA=8 ANDB=6 THEN 80 意思是，如果变量A的值等于8和变量B的值等于6，“IF—THEN”条件满足，接着执行80行。

测试程序=1

```
10 REM LOGICAL 'AND' TEST PROGRAM
20 A=8
30 B=6
40 IF A=8 AND B=6 THEN 70
50 PRINT "AND FAILED THE TEST AS LOGICAL OPERATOR"
60 GOTO 99
70 PRINT "AND PASSED THE LOGICAL OPERATOR TEST"
99 END
```

运行实例：

```
AND PASSED THE LOGICAL OPERATOR TEST
```

少数计算机使用AND操作符来对字符串的比较作逻辑运算“与”。

例如，IF A$="A" AND B$="B" THEN 80为，如果字符串变量A$等于（或相同）字母A和字符串变量B$等于B，“IF—THEN”条件得到满足，则接着执行80行。详细介绍参阅操作符“+”和“*”。

测试程序=2

```
10 REM STRING LOGICAL 'AND' TEST PROGRAM
20 A$="A"
30 B$="F"
40 IF A$="A"AND B$ >"B" THEN 70
50 PRINT "'AND' FAILED THE TEST AS A LOGICAL OPERATOR"
60 GOTO 99
70 PRINT "'AND' PASSED THE STRING LOGICAL OPERATOR TEST"
99 END
```

运行实例

```
'AND' PASSED THE STRING LOGICAL OPERATOR TEST
```

某些计算机使用逻辑运算AND来确定两个关系运算的条件是否成立。如果两个运算的条件同时成立，执行AND运算后带回值－1；当AND运算符的条件不成立，执行AND运算则带回值为0。

例如，PRINT A=4 AND B=8，意思是，如果A=4和B=8，则计算机打印一个－1；如果两个条件都不成立，则计算机打印一个0。

测试程序＝3

```
10 REM 'AND' LOGICAL TEST PROGRAM
20 PRINT "ENTER A NUMBER FROM 1 TO 10";
30 INPUT A
40 B=A > 4 AND A < 11
50 IF B=-1 THEN 80
60 PRINT A;"IS NOT GREATER THAN 4 AND LESS THAN 11"
70 GOTO 20
80 PRINT A;"IS GREATER THAN 4 AND LESS THAN 11"
99 END
```

运行实例（典型的例子）

```
ENTER A NUMBER FROM 1 TO 10? 2
 2 IS NOT GREATER THAN 4 AND LESS THAN 11
ENTER A NUMBER FROM 1 TO 10? 8
 8 IS GREATER THAN 4 AND LESS THAN 11
```

少数计算机用AND运算符来表示布尔代数两个二进制数的逻辑“与”。

在此不准备详细介绍布尔代数。它比较两个二进制数的位以确定它们是否都是二进制的“1”。当进行AND的两个位都是二进制的“1”时，其结果为“1”。

例如：

```
1 AND 0 = 0
0 AND 1 = 0
1 AND 1 = 1
```

因此，当计算机把一个数与另一个数进行AND运算时，就是一个数的每一个二进制位的值与另一个数的二进制位的值逻辑上相“与”，其结果产生第三个数。

例如：

十进制		二进制
3	（逻辑）AND	0011
5		0101
＝1		0001

在这个例子中，两个数中只有最右边的那位都是二进制“1”，因此，所得到的结果数是一个十进制的“1”（二进制的0001）。

测试程序 = 4

```
10 REM 'AND' BINARY LOGIC TEST PROGRAM
20 PRINT "ENTER A VALUE FOR X";
30 INPUT X
40 PRINT "ENTER A VALUE FOR Y";
50 INPUT Y
60 A=X AND Y
70 PRINT "THE LOGICAL 'AND' VALUE OF";X;"AND";Y;"IS";A
80 GOTO 20
99 END
```

运行实例（输入6和10）

```
ENTER A VALUE FOR X? 6
ENTER A VALUE FOR Y? 10
THE LOGICAL 'AND' VALUE OF 6 AND 10 IS 2
ENTER A VALUE FOR X?
```

替代的形式

少数计算机（例如，Britains　Acorn）允许用A.替代AND。

其它用法

AND（p，q）语句是王安2200 B计算机用来计算两个十六进制数的值或两个字符串的二进制逻辑“与”。第一个值P必须是一个字符串变量，第二个值是一个字符串变量或者是两位十六进制数。结果形成的值取代两个值中的第一个值。

例如，如果P$等于十六进制数5A，Q$等于十六进制数3C，则AND（P$，Q$）运算后，P$等于十六进制数18。

	十六进制	二进制
	P$ = 5A	01011010
	AND	
	Q$ = 3C	00111100
则	P$ = 18	00011000

当P$是一个字符串，Q是一个十六进制的常数时，P$字符串中的每个字符要转换成相应的ASCII值，然后再与十六进制的常数Q进行逻辑“与”。其结果再从ASCII转换成字符形式，并存放在P$中。例如，字符串“EFG”与十六进制数43相“与”，其结果是字符串“ABC”。

	E	F	G
ASCII	69	70	71
HEX	45	46	47
BINARY	0100 0101	0100 0110	0100 0111
	AND	AND	AND
hex 43	0100 0011	0100 0011	0100 0011
=	0100 0001	0100 0010	0100 0011
ASCII	65	66	67
so P$ =	A	B	C

测试程序 # 5

```
10 REM 'AND' TEST PROGRAM
20 DIM A$3 (Note: this line sets the length of A$ to 3 bytes.)
30 A$ = "CCC"
40 AND(A$,F1)
50 PRINT "AND PASSED THE TEST IF ";A$;"=AAA"
99 END
```

运行实例

```
AND PASSED THE TEST IF AAA=AAA
```

如果你的计算机没有此功能

如果你的计算机没有逻辑AND运算符，它的作用可以用两个IF—THEN语句来代替。可以用下列两行来取代测试程序1中的40行。

```
40 IF A <> 8 THEN 50
45 IF B = 6 THEN 70
```

王安2200 B计算机中AND语句可以用由字符串函数和与逻辑AND运算符组成的一些语句来代替。例如，上述的测试程序中的40行可以用下列语句来取代。

```
40 REM 'AND' REPLACEMENT
41 N = LEN(A$)
42 FOR I = 1 TO N
43 C$ = C$ + CHR$(ASC(MID$(A$, I, 1)) AND 241)
44 NEXT I
45 A$ = C$
```

参阅

OR, XOR, NOT, *, +, =, <, >, <>, <=, >=

命　令

APPEND命令可以把外部存储器（例如，磁盘或磁带）的一个程序与已经存放在内存中的一个程序连接起来。从外设引入的程序的行号必须大于已经存放在内存中的程序的最后一个行号。

例如，APPEND PROG2 使程序PROG2从外设输入，并且附加到内存中已有的程序的末尾。

测试程序

（在本例中，外设假定为盒式磁带）。为了测试APPEND，把下列这段程序作为PROG2存到磁带上。（可参阅CSAVE。）

```
1000 PRINT "THESE LINES ARE"
1010 PRINT "FROM PROG2"
1020 END
```

然后，打入NEW或SCRATCH以便清除该程序，再输入PROG1。

```
10 REM 'APPEND' TEST PROGRAM PROG1
20 PRINT "THESE LINES ARE"
30 PRINT "FROM PROG1"
40 PRINT "    BUT..."
```

然后，打入APPEND PROG2。

在PROG2已输入后，开始运行（用你的计算机规定的有关命名文件名的方法来SAVE PROG2和APPEND PROG2，例如有些计算机规定文件名要用引号括起来，有的则规定文件名只能用一个字母等等）。

运行实例

```
THESE LINES ARE
FROM PROG1
    BUT...
THESE LINES ARE
FROM PROG2
```

APPEND命令常常用于把大的数据文件装入到一个程序中。它也可以用来把一个经常使用的子程序“加”到一个已有的程序中（这就是为什么在这本《BASIC大全》中的所有主要的子程序行号都在30000以上，而且不重叠的原因）。

如果你的计算机没有此功能

如果你的计算机不能顺利地响应APPEND，那么你可以试用TAPPEND、MERGE或WEAVE命令。如果这几个命令中没有一个能工作的话，只要能确定现有

的程序“指针”的位置，某些计算机也可以实现程序的连接（即叠加）。

如果你还是不大清楚的话，可以查阅你的计算机手册。下面是用于TRS—80的过程。

程序“指针”可以定位于内存储器中现有程序的末地址。对于TRS—80Ⅰ型机而言，这个地址存放在16333和16334单元中。用PEEK语句可以取到存放在这两个单元中的值。

从第一个数中减去2（第一个数存放在16333单元中），如果差是负值，则把差加上256，然后从第二个数减去1。

不论发生哪种情况，把这两个数不加变化地存放（POKE）到16548和16549两个单元中。

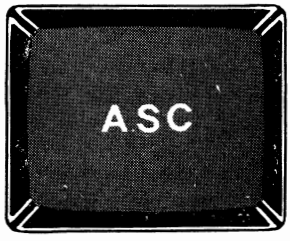
现在，从磁带把程序装入（CLOAD）。恢复16548～16549单元中的值，把233重新存入（POKE）16548单元，把66重新存入16549单元中。

第二段程序现在附加（APPEND）到第一段程序上了。

参阅

TAPPEND, PEEK, POKE, CSAVE, CLOAD, DATA

函　数



ASCII

ASC函数可以把一个字符或字符串变量转换成与它相应的ASCII十进制数。

例如，PRINT ASC（“A”），打印65，65就是字母“A”的ASCII码。

PRINT ASC (A$)　打印出字符串变量A$中第一个字符的ASCII码。

测试程序 1

```
10 REM 'ASC(CHARACTER)' TEST PROGRAM
20 PRINT "THE ASCII CODE FOR LETTER A IS";
30 PRINT ASC("A")
40 IF ASC("A")=65 THEN 70
50 PRINT "ASC FAILED THE TEST"
60 GOTO 99
70 PRINT "ASC PASSED THE TEST"
99 END
```

运行实例

```
THE ASCII CODE FOR LETTER A IS 65
ASC PASSED THE TEST
```

下一个程序使用一个字符串变量来测试ASC函数。

测试程序＃2

```
10 REM 'ASC(STRING VARIABLE)' TEST PROGRAM
20 PRINT "TYPE ANY LETTER, NUMBER, OR CHARACTER";
30 INPUT A$
40 PRINT "THE ASCII CODE FOR ";A$;" IS";ASC(A$)
99 END
```

运行实例（输入H）

```
TYPE ANY LETTER, NUMBER, OR CHARACTER? H
THE ASCII CODE FOR H IS 72
```

某些含有ASC函数的计算机可以接收一个字符以上的字符串，但是只取第一个字符，并把它转换成ASCII码。为了对ASC允许的字符串长度进行测试，我们使用第二个测试程序并逐渐输入（INPUT）较长的字符串，直到出现错误信息为止。

替代的形式

某些计算机（例如，DEC—10）使用ASCII替代ASC。

其它用法

某些解释程序（例如，MAXBASIC）使用ASC（A$，X）的格式，它打印出在A$中的前X字符的ASCII码。

参阅

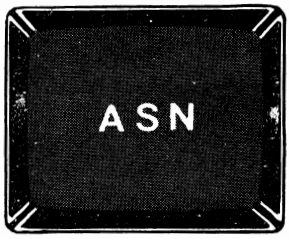
CHR$，CODE

ASN(n)函数是TEKTRONIX4050系列BASIC用来计算ARCSIN(即SIN^{-1})的，它用的角度单位是弧度（而不是角度）。1弧度大约等于57度。

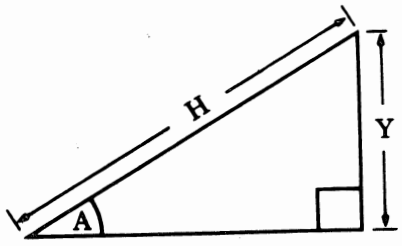
Arcsin (ASN) 的定义是角（A），它是由此角的对角边长度(Y)除以直角三角形的斜边长度(H)取SIN^{-1}而得到的。

A=ASN (Y/H)

函　数



ASNG
ASND
ARCS IN



ASN的反函数是SINE (SIN)。一个角的SINE是这个角的对应直角边与这个直角三角形的斜边之比。

SIN (A) =Y/H

测试程序

```
10 REM 'ATN' TEST PROGRAM
20 PRINT "ENTER A RATIO OR TANGENT VALUE";
30 INPUT N
40 A=ATN(N)
50 PRINT "THE ANGLE WITH THE Y/X RATIO OF";N;"IS";A;
   "RADIANS"
30999 END
```

运行实例（输入0.5）

```
ENTER A RATIO OR SINE VALUE? .5
THE ANGLE WITH THE Y/H RATIO OF .5 IS .52359 RADIANS
```

某些计算机以度数或梯度（100 梯度 = 90度）来计算角的值。这些计算机在用度数时使用ASND函数，而用梯度时使用ASNG。如果上面的程序40行中的ASN换成ASND或ASNG，而在运行中输入0.5，就会得到30度和33.3333梯度的结果。

为了把值从弧度转换成角度，只要把这个角（以弧度为单位）乘以57.29578。

例如：D = ASN (A) * 57.29578

为了把值从度转换成弧度，则把这个角（以度为单位 乘以0.174533。

例如：R = A（角用度数表示）* .0174533。

替代的形式

某些计算机（例如，Sinclair ZX80）用ARCSIN代替ASN。在测试程序的程序行40中试用ARCSIN，以便检查你的计算机是否允许使用它。

如果你的计算机没有此功能

如果你的解释程序具有ATN（tan $^{-1}$）和SQR（平方根）函数，但是没有ASN，可以用Z * ATN (X/ (1 + SQR (1 - X * X)) 来代替ASN。

如果你的解释程序没有ASN或ATN和SQR函数，那么可以用下面的子程序来代替：

```
30000 GOTO 30999
30520 REM * ARCSIN SUBROUTINE * INPUT S, OUTPUT W
30522 REM ALSO USES VARIABLES X AND Z INTERNALLY
30524 X=S
30526 IF ABS(S)<=.5 THEN 30556
30528 X=.5*(1-ABS(S))
30530 IF X>=0 THEN 30536
30532 PRINT S; "IS OUT OF RANGE"
30534 STOP
30536 W=X/2
30538 IF W=0 THEN 30554
30540 Z=0
30542 Y=(X/W-W)/2
30544 IF Y=0 THEN 30554
30546 IF Y=Z THEN 30554
30548 W=W+Y
30550 Z=Y
30552 GOTO 30542
30554 X=W
30556 Y=X*X
30558 W=(4.241734E-2*Y+2.399402E-2) *Y+4.552063E-2
30560 W=(((W*Y+.074947)*Y+1/6)*Y+1)*X
30562 IF ABS(S)<=.5 THEN 30566
30564 W=SGN(S)*(1.570796-2*W)
30566 RETURN
```

为了把这个子程序用在测试程序上，以便用来求比率（Y/H）的ARCSIN（以弧度为单位），可以把测试程序做如下修改：

```
35  S = N
```

```
40  GOSUB 30520
```

为了使ARCSIN子程序能得到以度数为单位的角，要加上以下几行：

```
30566   W = W * 57.29578
30568   RETURN
```

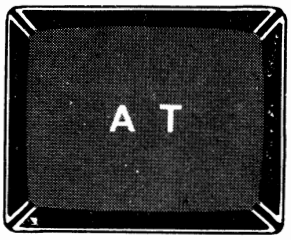
其它用法

没有

参阅

ACS，ATN，COS，SIN，SQR，TAN

函　数



A．

AT函数与PRINT语句（TRS—80Ⅰ型机中的BASIC语句）一起用来确定PRINT语句打印的起始位置。AT函数值可以是数、数值变量或数学表达式。在AT指出的位置和要打印的字符串间要加上逗点或分号。

例如：

```
10 PRINT AT 420, "HELLO"
20 PRINT AT (420); "HELLO"
```

两行都把“HELLO”这个词打印在420这个位置上，括号是可选的（即可有可无）。

测试程序＝ 1

```
10 REM 'AT' TEST PROGRAM
20 PRINT AT 128,"2. IF THIS LINE IS PRINTED AFTER LINE 1."
30 PRINT AT 0, "1. THE 'AT' FUNCTION PASSED THE TEST"
40 GOTO 40
99 END
```

运行实例

```
1. THE 'AT' FUNCTION PASSED THE TEST
2. IF THIS LINE IS PRINTED AFTER LINE 1.
```

TRS—80有1024个PRINT AT的位置（从0到1023）。如果使用一个小于0或大于1023AT值，该计算机就自动地计算超出范围的那个数和1023之间的差，以便求出AT值。

例如，PRINT AT 1034，“HELLO”就是把HELLO这个词打印在位置10中（不要忘了0也是一个位置。）

测试程序＝ 2

```
10 REM 'AT OVERFLOW' TEST PROGRAM
20 PRINT AT 192, "'AT' (OVERFLOW) PASSED THE TEST"
30 PRINT AT 1248, "IF ONLY ONE LINE IS PRINTED."
99 END
```

运行实例

```
'AT' (OVERFLOW) PASSED THE TEST IF ONLY ONE LINE IS
    PRINTED."
```

下面的程序测试了解释程序在AT函数中使用数值、数值变量或数学表达式的能力。

测试程序 = 3

```
10 REM 'AT VALUE' TEST PROGRAM
20 FOR X=1 TO 15
30 PRINT X
40 NEXT X
50 PRINT AT X*28+4, "'AT' PASSED THE TEST IF THIS IS LINE
   #8.";
60 GOTO 60
99 END
```

运行实例

```
1
2
3
4
5
6
7
8 'AT' PASSED THE TEST IF THIS IS LINE #8.
9
10
11
12
13
14
15
```

替代的形式

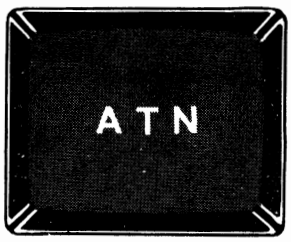
@操作符是某些计算机（例如，TRS—80Ⅱ型、Ⅲ型和磁盘BASIC）用来代替AT函数的。参阅本书中对@操作符的规定。

使用TINY BASIC的计算机也可以用A.来表示AT。

参阅

@, PRINT, TAB, HLIN, VLIN, DRAW, XDRAW

函　数



ANSI

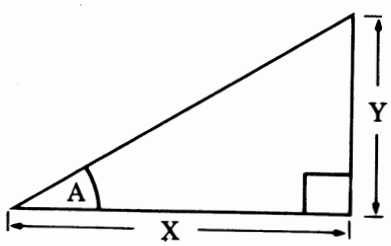
ATN（n）函数用来计算以弧度为单位（而不是角度）的 $\tan^{-1}$。一弧度大约为57度。

ATN（$\tan^{-1}$）可求出角（A），将角A的对角边（Y）除以角的邻边(X)，再取 $\tan^{-1}$即可得角A。ATN是ARCTANGENT的缩写，意思是“反正切”。

A= ATN (Y/X)

ATN的反函数是正切（TAN）。一个角的正切是这角的对边与这个角的邻边（X）之比。

TAN（A）=Y/X



ATAN
ATND
ATNG
ARCTAN

测试程序

```
10 REM 'ATN' TEST PROGRAM
20 PRINT "ENTER A RATIO OR TANGENT VALUE";
30 INPUT N
40 A=ATN(N)
50 PRINT "THE ANGLE WITH THE Y/X RATIO OF";N;"IS";A;
   "RADIANS"
30999 END
```

运行实例（输入 2）

```
ENTER A RATIO OR TANGENT VALUE? 2
THE ANGLE WITH THE Y/X RATIO OF 2 IS 1.10715 RADIANS
```

某些计算机以度或以梯度为单位来计算角的值（100梯度 = 90度）。这些计算机若以度为单位则用ATND函数，若以梯度为单位则使用ATNG。可以用上述两种函数来替换测试程序中的40行，并使它运行。在运行中输入 2，就会得到63.43495度和70.4833梯度的结果。

为了把角的单位由弧度转换成度，将角（以弧度为单位的）乘以57.29578。

如：D= ATN（A）* 57.29578

为了把值从度数转换成弧度，则把这个角（以度数为单位）乘以0.174533。

例：R= A（以度数为单位来表示角）* .0174533。

替代的形式

少数计算机使用ATAN来代替 $\tan^{-1}$函数。为了检查ATAN是否适用于你的计算机，把40行中的ATN用ATAN代替，并让它运行。

某些计算机（例如，Sinclair ZX80）用ARCTAN来代替ATN。在测试程序的40行中，可以试用ARCTAN，以便检查你的计算机是否接受这种形式。

如果你的如果你的计算机没有此功能

如果你的解释程序不具有ATN（$\tan^{-1}$）函数的能力，可以用下面的子程序来代替。在运行此子程序时，必须用到SGN函数（本书225页）下面的子程序（这里没有列出）。

```
30000 GOTO 30999
30570 REM * ARCTANGENT SUBROUTINE * INPUT X, OUTPUT A
      (RADIANS)
30572 REM ALSO USES B, C, D AND T INTERNALLY
30574 GOSUB 30080
30576 D=X
30578 X=ABS(X)
30580 C=0
30582 IF X<=1 THEN 30588
30584 C=1
30586 X=1/X
30588 A=X*X
30590 B=((2.86623E-3*A-1.61657E-2)*A+4.29096E-2)*A
30592 B=(((B-7.5289E-2)*A+.106563)*A-.142089)*A
30594 A=(((B+.199936)*A-.333332)*A+1)*X
30596 IF C<>1 THEN 30600
30598 A=1.570796-A
30600 A=T*A
30602 X=D
30604 RETURN
```

为了把这个子程序和测试程序一起使用，以便求出y/x的$\tan^{-1}$值，我们可以将测试程序修改如下：

```
35 X = N
40 GOSUB 30570
```

为了使求$\tan^{-1}$的子程序用度数来表示角，应加下面一行：

```
30603 A = A * 57.29578
```

其它用法

某些（很少）解释程序可以自动地将其它角度单位转换成度数。

参阅

TAN ASN SIN COS ACS

技巧

ATN函数是非常重要的，大多数计算机只有ATN这样一种“反三角函数”。几乎很少有$\cos^{-1}$（反余弦）和$\sin^{-1}$（反正弦）函数。这样就剩下ATN作为唯一的“窗口”，通过它可以计算所有的角，并且从窗口返回到“外部”去。

显而易见，如果使用ATN，那么就一定要知道TAN，或者能求出TAN（如上面的Y/H就是TAN（A）的值）。当然，说起来容易做起来就要困难些。下面的公式将能从任一个个三角函数求出TAN，并且通过ATN求出该角本身。

$$TAN = 1/COT \qquad TAN = \sqrt{\frac{1 - COS^2}{COS^2}} \qquad TAN = \frac{1}{\sqrt{\frac{1 - SIN^2}{SIN^2}}}$$

$$TAN = \frac{1}{\sqrt{CSC^2 - 1}} \qquad TAN = \sqrt{SEC^2 - 1}$$

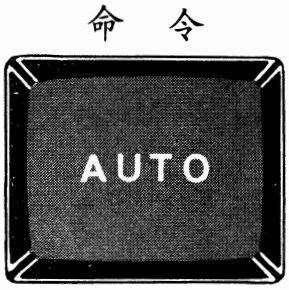
这些公式利用三角函数之间的关系，使我们得到计算每个反函数的方法。例如，为了计算A= ARCSEC (X)，可以使用

A= A T N (SQR (X*X- 1))

每个反函数公式的BASIC语句是：

```
ARCCOS(X) = 1.5708 - 2*ATN(X/(1+SQR(1-X*X)))
ARCCOT(X) = ATN(1/X)
ARCCSC(X) = ATN(1/SQR(X*X-1)
ARCSEC(X) = ATN(SQR(X*X-1))
ARSSIN(X) = 2*ATN(X/(1+SQR(1-X*X)))
```

AUTO命令提供了自动加入程序行号的功能。我们在AUTO 命 令 中指定起始行号和两个行号的间隔。例如，AUTO 100， 5就是把第一个行号定在100，按顺序每二行之间行号增加5。当编写新程序时，这个特点是很方便的。



如果在AUTO命令中没有指定起始行号和行号增量，那么计算机就自动地把第一个程序行号定为10，然后，每行的行号增加10。

如果AUTO命令产生的行号是已存在的，那么在该行号后面显示出一个星号。这就使程序员注意打入计算机中的这一行将删去已有的行。可以中断AUTO命令的执行以防止上述情况发生。为了中断AUTO命令的执行，某些计算机需要按BREAK键，而另一些计算机则需要打入 CONTROL -C（即同时按CONTROL键和C键）。

测试过程

为了测试计算机的AUTO特性，先打入AUTO命令，并且按下ENTER（在某些键盘上是RETURN）。如果行10的后面跟着打印一个提示，那么，计算机就成功地通过AUTO命令的测试。

再按下ENTER键，计算机应该打印下一个行号，它的行号比上一个增加10。

打入命令AUTO 10， 5，然后输入这个程序。

```
10 REM 'AUTO' TEST PROGRAM
15 PRINT "THE NEXT LINE NUMBER SHOULD INCREASED BY 5"
20 PRINT "PRESS THE BREAK KEY TO STOP THE AUTO FEATURE"
99 END
```

在用BREAK键中止了AUTO命令之后，就可以输入该序列之外的行号（如行号99）。

再输入AUTO 10, 5，那么程序行号后面应该跟着打印一个星号，这表示10行中已有信息；如果按下ENTER，原有信息就要被清除。可以打入的新信息以取代原有信息，如果想保存原有信息，则按BREAK键。

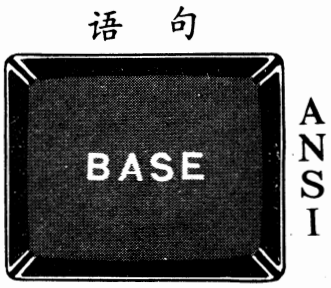
把程序列表来检查每行的内容。

其它用法

没有

参阅

BREAK, LIST, MAN



在某些计算机中（例如，Control Data BASIC第三版），BASE语句用来定义数组的BASE（最低的）元素是0还是1。

例如：

```
10  BASE 0
20  DIM A(5)
```

BASE 0语句规定该数组具有六个元素〔从A（0）到A（5）〕。

许多计算机可以自动地形成0到10（11个元素）的数组元素，而不要预置DIM语句。BASE语句允许这个数组从通常的11个元素（0到10）改为10个元素（1到10），而且反之也有效。

通常，在一段程序中，只用一个BASE语句，它必须在DIM语句和使用数组变量之前执行。

测试程序=1

```
10 REM 'BASE' TEST PROGRAM
20 BASE 0
30 DIM A(5)
40 FOR X=0 TO 5
50 A(X)=X
60 NEXT X
70 FOR X=0 TO 5
80 PRINT A(X);
90 NEXT X
100 PRINT "THE BASE STATEMENT PASSED THE TEST"
999 END
```

运行实例

```
 0  1  2  3  4  5 THE BASE STATEMENT PASSED THE TEST
```

少数计算机（例如，使用MAXBASIC的机型），在一段程序中允许使用一个以上的BASE语句，并允许把BASE的值定义为任意的整数值。

例：

```
10 BASE 5
20 DIM A(10)
```

BASE 5语句规定该数组有6个元素〔A（5）到A（10）〕。

测试程序#2

```
10 REM 'BASE' TEST PROGRAM
20 BASE 3
30 DIM A(5)
40 FOR X=3 TO 5
50 A(X)=X
60 NEXT X
70 BASE 0
80 FOR X=0 TO 2
90 A(X)=X
100 NEXT X
110 FOR X=0 TO 5
120 PRINT A(X);
130 NEXT X
140 PRINT "THE BASE STATEMENT PASSED THE TEST"
999 END
```

运行实例

```
0 1 2 3 4 5 THE BASE STATEMENT PASSED THE TEST
```

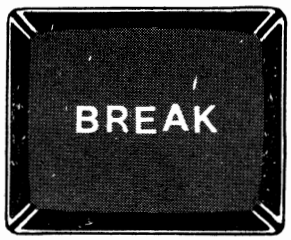
其它用法

ANSI BASIC 包括 OPTION 语句

参阅

DIM，OPTION

语　句



在少数计算机中（例如　Harris BASIC —— V），BREAK语句用来停止一个或多个程序行的执行，并且置计算机于监控方式或立即方式，它类似于STOP语句。

BREAK语句中可以出现一个语句标号或多个语句标号。语句标号间用逗号隔开。BREAK 语句后面的标号可以使程序在该语句处停止执行。

例如，10 BREAK 50，70，100

这个语句的作用是在执行语句50，70和100前使计算机停止运行。在每个 BREAK 之后如果想使程序继续执行，可以打入“CONTINUE”。

在第一个语句标号和最后一个语句标号之间画一个破折号，它表示这个范围内的所有的语句标号。

例如，10 BREAK 50—100

从50到100语句每个语句执行完后都会使计算机停止。

使用BREAK ALL语句可以使程序在执行每一个语句之前都中断一次。这样用户可以一次一行地一步一步地“通过”此程序。在每一次BREAK中断后打入CO(Continue) 命令即可。

BREAK语句与END语句不同，END 语句（在某些计算机中）可以使所有的变量被重新置 0，而执行BREAK语句时，存放在变量中的值是保持不变的。

测试程序

```
10 REM 'BREAK' TEST PROGRAM
20 BREAK 30,50,70-90
30 PRINT "THE COMPUTER SHOULD STOP EXECUTION AT LINE 30."
40 REM TYPE THE COMMAND 'CO' TO CONTINUE
50 PRINT "LINE 50"
60 REM THIS LINE NOT INCLUDED IN THE BREAK STATEMENT
70 PRINT "LINE 70"
80 PRINT "LINE 80"
90 PRINT "AND LINE 90"
99 END
```

运行实例

```
THE COMPUTER SHOULD STOP EXECUTION AT LINE 30
LINE 50
LINE 70
LINE 80
AND LINE 90
```

其它用法

大多数键盘具有BREAK键，用于人工方式使程序中断。

参阅

STOP，CONT，END

命 令
语 句

GOODBYE

BYE是一种用来退出BASIC的命令，大多数大型分时的计算机用BYE作为结束命令，并且终止用户的作业。

有几种小型计算机（例如，SOL和ATARI），当用户打入BYE命令时，就进入监控级或磁盘操作系统。在有的扩展BASIC中，BYE可以作为一个程序语句。

替代的形式

在分时系统中，如果用户要停止计算机工作，常常用GOOD BYE代替（或还有）BYE。要查一下用户手册，看你的计算机中“结束工作”的方法。

其它用法

没有

参阅

SYSTEM，磁盘BASIC简介

语 句

CALL（n）语句使程序控制转移到一个机器语言的子程序，该子程序的入口在内存地址n。该子程序可以是计算机系统软件的一部分，也可以由用户来编写，用户编写的机器语言程序可以用BASIC程序通过POKE语句由键盘输入，或者用“监控/编辑”程序在“系统级”输入。

例： CALL 18624

这个语句将使存放在十进制地址18624中的机器语言程序开始执行。当在机器语言程序中遇到RETURN指令时，返回BASIC，而执行紧跟着CALL语句后的那个语句。为了测试CALL语句，你必须把机器语句程序输入计算机，或确定一个常驻子程序的入口。查阅你的计算机使用手册，就可以了解应该怎样做。

其它用法

某些计算机使用CALL语句转移到一个专门的BASIC子程序。在这些计算机中，CALL语句的用法如同GOSUB语句，不同之处是CALL语句不用语句标号而用子程序名。以这种方法使用CALL语句时，子程序以一个 SUB 语句开始，此语句包含子程序名，子程序以SUBEND 语句结束。

例如，CALL TEST语句可以控制转移到子程序TEST。当遇到SUBEND时，流程返回到CALL语句下一条语句。

测试程序

```
10 REM 'CALL' TEST PROGRAM
20 CALL TEST
30 PRINT "'CALL' PASSED THE ";A$
40 GOTO 99
50 SUB TEST
60 A$="TEST"
70 SUBEND
99 END
```

运行实例

```
'CALL' PASSED THE TEST
```

参阅

USR，POKE，SYSTEM，GOSUB，RETURN

函 数

CDBL函数可以使数和数值变量从“单精度”变成“双精度”。用于CDBL函数中的变量，如果再次使用而不用CDBL函数，则变量又返回到它们原始的“单精度”状态。

双精度变量可以存贮17位数（只打印16位数）。单精度变量精确到6位数。必须十分注意：应保证用于产生双精度结果的数字也必须是双精度。否则，其结果将是一个大的假象。

测试程序

```
10 REM 'CDBL' TEST PROGRAM
20 X=2
30 Y=3
40 PRINT "CDBL CHANGES X/Y FROM";X/Y;"TO";CDBL(X)/CDBL(Y)
50 PRINT "AND BACK TO THE VALUE OF";X/Y;"WHEN REMOVED"
99 END
```

运行实例

```
CDBL CHANGES X/Y FROM .666667 TO .6666666666666667
AND BACK TO THE VALUE OF .666667 WHEN REMOVED
```

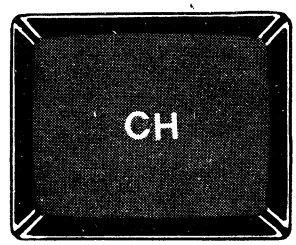
其它用法

没有

参阅

DEFDBL，DEFSNG，DEFINT，CSNG，#，!，%，CINT

函　数



在 ACORN ATOM计算机中，CH函数能得到已给定的字符串第一个字符串的ASCII码。例如，PRINT CH "ACORN"得65(A的ASCII码)，并且在屏幕上显示。CH只识别字符串的第一个字符

要进一步了解请参阅ASC。

测试程序

```
10 REM 'CH' TEST PROGRAM
20 PRINT "THE ASCII CODE FOR LETTER A IS";
30 PRINT CH "A"
40 IF CH"A"=65 THEN 70
50 PRINT "CH FAILED THE TEST"
60 GOTO 99
70 PRINT "CH PASSED THE TEST"
99 END
```

运行实例

```
THE ASCII CODE FOR A IS 65
CH PASSED THE TEST
```

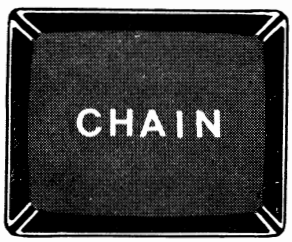
其它用法

没有

参阅

ASC, CODE, CHR$, 附录有关ASCII码

语 句



CHAIN 语句用来把一个新程序从一台外部设备（例如，磁盘或磁带）输入到计算机的内存储器中，并且不用附加RUN命令就执行程序。一个程序可以连接（CHAIN）另外一个程序。一个一个地连接下去，甚至可以返回到开始的那一个程序，该程序是作“菜单”(menu) 用的。

CHAIN的主要优点是能使有关的程序顺序地执行，而不需要把这些程序同时放在计算机内。当有一个共用的文件存放在外部介质上时，CHAIN语句是特别有用的，可以用CHAIN来存取和处理这个文件。在有足够大的磁盘存储器和存取速度快的系统中CHAIN语句得到很好的应用。

如果要将变量的值从一个程序传送到另一个程序中去，则应建立一个单独的文件。在一个程序被CHAIN到另一个程序之前，必须把变量的值存放在这个文件中，这样新程序先将它们读回，然后再执行。

某些BASIC语句能把第一个程序使用的变量值直接地传给第二个程序，例如，Microsoft BASIC 5.0版本可以接收CHAIN“PROG 2”,150,ALL语句，语句“连接”磁盘上的名为“PROG 2”的程序。PROG2在接收到所有变量值后，从150语句处开始执行。这些变量值是由调用程序预先定义了的。

新程序的名称一定要包括在CHAIN之后。一些计算机用跟着该程序名称的一个数来确定新程序开始执行的行号。如果省略了起始行号，那么计算机就自动地从新程序的起点开始执行。

例如，10 CHAIN TEST, 30 告诉计算机要清除目前已在内存中的程序，并从外部设备装入一个名为“TEST”的程序，然后从30行开始执行。在一些计算机（例如，DEC 10 BASIC）中可以指定外部存储设备，方法是在CHAIN语句之后写上设备名称，然后跟着一个冒号（：）。

例如，10 CHAIN PRT: TEST, 70, 它告诉计算机从纸带输入机装入一个名为“TEST”的程序，然后从70行开始执行。

测试程序

将这个程序用“TEST”这个名字存到磁盘或磁带上。

```
10 REM *TEST* PROGRAM
20 PRINT "THE 'TEST' PROGRAM IS NOW RUNNING"
30 FOR X=1 TO 9
40 PRINT X;
50 NEXT X
60 PRINT "THIS PROGRAM SHOULD NOW CHAIN BACK
   TO THE MAIN PROGRAM"
70 CHAIN MAIN, 40
99 END
```

现在，把主程序输入到计算机中，并且用“MAIN”这个名字，把它存放在磁盘上或磁带上。

```
10 REM *MAIN* PROGRAM
20 PRINT "THIS PROGRAM SHOULD LOAD AND RUN THE 'TEST'
   PROGRAM"
30 CHAIN TEST
40 PRINT "CHAIN PASSED THE TEST IF THE 'TEST' PROGRAM"
50 PRINT "PRINTED A SERIES OF NUMBERS"
99 END
```

准备好磁盘或者磁带，然后运行（RUN）。

运行实例

```
THIS PROGRAM SHOULD LOAD AND RUN THE 'TEST' PROGRAM
THE 'TEST' PROGRAM IS NOW RUNNING
 1 2 3 4 5 6 7 8 9
THIS PROGRAM SHOULD NOW CHAIN BACK TO THE MAIN PROGRAM
CHAIN PASSED THE TEST IF THE 'TEST' PROGRAM
PRINTED A SERIES OF NUMBERS
```

如果你的计算机没有此功能。

我们可以在使用软磁盘或盒式磁带的微型机上成功地完成连接。大多数小系统的BASIC从被连接的程序的第一个语句行处开始执行，而不能任意选择从其它地方开始执行。

下面三个程序说明在TRS—80磁盘系统上怎样实现连接的。由RUN代替 CHAIN。注意新程序的名称要放在引号内。

```
10 REM *MAIN* PROGRAM
20 PRINT
30 PRINT "THIS IS THE MAIN CONTROL PROGRAM."
40 INPUT"SHALL WE 'CHAIN' TO PROGRAM #1 OR #2
   (TYPE 1 OR 2)";I
50 PRINT "STAND BY FOR LOADING - - -"
60 ON I GOTO 70, 80
70 RUN "TEST1"
80 RUN "TEST2"
99 END
```

```
10 REM *TEST1* PROGRAM
20 PRINT "TEST PROGRAM NUMBER 1 IS NOW RUNNING"
30 FOR X=1 TO 9
40 PRINT "ONE",
50 NEXT X
60 PRINT
70 PRINT "WE WILL NOW CHAIN BACK TO THE MAIN PROGRAM---"
80 RUN "MAIN"
99 END
```

```
10 REM *TEST2* PROGRAM
20 PRINT "TEST PROGRAM NUMBER 2 IS NOW RUNNING"
30 FOR X=1 TO 9
40 PRINT "TWO",
50 NEXT X
60 PRINT
70 PRINT "WE WILL NOW CHAIN BACK TO THE MAIN PROGRAM---"
80 RUN "MAIN"
99 END
```

打入每个程序，并且用每个程序中10语句指定的名字存放在磁盘上。然后运行主程序。观看屏幕和磁盘驱动器，看程序连接和执行的情况。

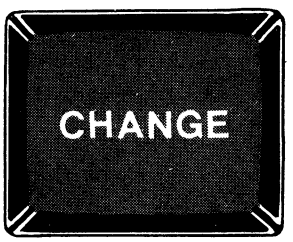
其它用法

没有

参阅

CLOAD，CSAVE，COMMON，RUN

语　句



DEC—10计算机使用CHANGE语句把字符串转换成ASCII码，也可以反过来把ASCII码转换成字符串。

CHANGE X$ TO X就是把字符串X$中的每个字符转换成对应的ASCII码，并把它们存放在数组X中。字符串的长度，即LEN（X$），存放在数组元素X（0）中。

CHANG X TO X$把存放在数组X中的值转换成字符串X$。被转换值的个数放在X（0）中。X的每个值必须在0～255范围中。

测试程序#1

```
10 REM 'CHANGE X$ TO X' TEST
20 DIM X(6)
30 X$ = "SYSTEM"
40 CHANGE X$ TO X
50 PRINT "CHANGE HAS CONVERTED 'SYSTEM' TO"
60 FOR N=1 TO X(0)
70 PRINT X(N);
80 NEXT N
99 END
```

运行实例

```
CHANGE HAS CONVERTED 'SYSTEM' TO
 83 89 83 84 69 77
```

测试程序#2

```
10 REM 'CHANGE X TO X$' TEST
20 DIM X(6)
30 READ X(0)
40 FOR I=1 TO X(0)
50 READ X(I)
60 NEXT I
70 CHANGE X TO X$
80 PRINT "CHANGE ";X$;" THE TEST"
90 DATA 6, 80, 65, 83, 83, 69, 68
99 END
```

运行实例

```
CHANGE PASSED THE TEST
```

如果你的计算机没有此功能

CHANGE语句可以用LEN、ASC、MID$和CHR$语句来模拟。在测试程序1中，可以用下面几行来代替40行：

```
40 X(0) = LEN(X$)
41 FOR I=1 TO X(0)
42 X(I) = ASC(MID$(X$,I,1))
43 NEXT I
```

在测试程序#2中，可以用下列语句取代70行。

```
70 X$ = ""
71 FOR I=1 TO X(0)
72 X$=X$ + CHR$(X(I))
73 NEXT I
```

参阅

LEN、ASC、MID$、CHR$、DIM

函 数

CHR
CHAR
CHARS
CHR兀

CHRS函数可以用来检索用十进制ASCII码表示的单个字符，ASCII码是放在圆括弧中的。例如，PRINT CHRS（75），打印出字母K。

ASCII码可以用ASCII码范围内的一个数或变量来表示（一般是0～127）。许多计算机具有扩展的ASCII码(最多为255），扩展的ASCII码具有专门的功能和作图特性。大多数计算机不考虑某些非标准的ASCII数值（一般是控制行式打印机，清除屏幕等等）。

下面的程序可以测试任一个ASCII码的特性，如果能打印出来，还可以检查ASCII字符的特性。

测试程序

```
10 REM 'CHR$' TEST PROGRAM
20 PRINT "ENTER THE LOWEST ASCII CODE NUMBER";
30 INPUT L
40 PRINT "ENTER THE HIGHEST ASCII CODE NUMBER";
50 INPUT H
60 FOR X=L TO H
70 PRINT "ASCII CODE";X;"= ";
80 PRINT CHR$(X)
90 FOR Y=1 TO 150
100 NEXT Y
110 NEXT X
999 END
```

运行实例（只检查四个数值）

```
ENTER THE LOWEST ASCII CODE NUMBER? 65
ENTER THE HIGHEST ASCII CODE NUMBER? 68
ASCII CODE 65 = A
ASCII CODE 66 = B
ASCII CODE 67 = C
ASCII CODE 68 = D
```

试把这个程序用于你的计算机中所有的ASCII码。

替代的形式

CHRS函数有几种不同的形式，即CHR（SOL和SWTP 4 K），CHAR$（Micropolis BASIC）和CHAR（MAX—80 BASIC）。

其它的用法

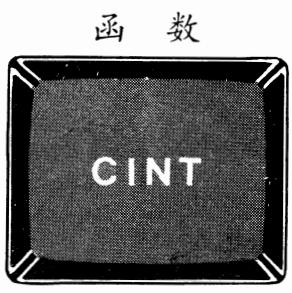
Swedish ABC—80使用CHR兀，可以把不多于四个的ASCII数转换成与它们相对应

的字符。（¤是Swedish通用符号，它代表“字符串”，可以代替$。）

在MAXBASIC中的CHAR（N1，N2）需要两个数值。第一个数值是ASCII码，第二个数值指出可以产生多少个字符。CHAR（73，1）等效于CHR（73）。CHAR（65，4）= AAAA。更详细的叙述可以参阅STRING$。

参阅

ASC、STRING$



CINT函数可以用来把单个数值或者数值变量转换成与它们相对应的整数值。用于CINT函数中的变量，如果它们再使用时不用CINT函数时，则恢复成它们的原来精度。

取整后的数值总是“降低的”，即得到的整数与小数点右边的数无关。一个正数取整，就是将其小数部分截去。一个负数取整时，其结果为舍入到下一个更小的整数。

例如，PRINT CINT（-4.65）将打印数值-5。

大多数计算机不允许在CINT函数中用小于-32768或者大于+32768的数。INT函数与CINT函数很类似，但是，数值不限制在有限的范围内。

测试程序

```
10 REM 'CINT' TEST PROGRAM
20 DEFDBL X
30 X=12345.6789
40 PRINT "CINT CHANGES THE VALUE OF X FROM";X;"TO";
   CINT(X)
50 PRINT "AND BACK TO THE VALUE OF";X;"WHEN REMOVED"
99 END
```

运行实例

```
CINT CHANGES THE VALUE OF X FROM 12345.6789 TO 12345
AND BACK TO THE VALUE OF 12345.6789 WHEN REMOVED
```

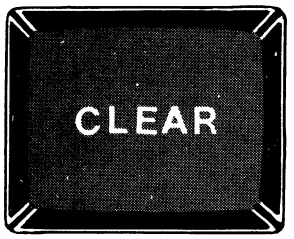
其它用法

没有。

参阅

DEFINT，INT，DEFDBL，DEFSNG，CDBL，CSNG，!，#，%

命　令
语　句



CLR

有一些计算机用CLEAR来清除已在内存中的程序。打入：

10 REM CLEAR TEST PROGRAM

然后打入CLEAR…再打入LIST。如果没有列出任何语句，则CLEAR通过这个测试。

CLEAR语句一般常用于把所有的数值变量置零，并且清除所有保存在字符串中的数据。

测试程序＃1

```
10 REM 'CLEAR' TEST PROGRAM.
20 A=300
30 A$="TEST STRING"
40 PRINT "BEFORE THE 'CLEAR' COMMAND A=";A
50 PRINT "AND STRING VARIABLE A$ = ";A$
60 CLEAR
70 PRINT "AFTER THE 'CLEAR' COMMAND A=";A
80 PRINT "AND STRING VARIABLE A$=";A$
99 END
```

运行实例

```
BEFORE THE 'CLEAR' COMMAND A=300
AND STRING VARIABLE A$=TEST STRING
AFTER THE 'CLEAR' COMMAND A=0
AND STRING VARIABLE A$=
```

某些计算机用CLEAR语句为字符串指定在内存中的字节数目。程序员利用这个特性来保留字符串存储所需要的实际空间总量。

例如，CLEAR 100 是在存储器中划出100个字节给字符串。对于具有CLEAR性能的许多解释程序，常常自动地在存储器中为字符串留出50个字节。其它有的解释程序则最多可保留200个字节。CLEAR ＃＃＃ 允许“省缺”字节数以适应不同的要求。

可以在打印语句中用FRE（A$）来检查内存中的为字符串保留空间中的剩余部分。

测试程序＃2

```
10 REM 'CLEAR X' TEST PROGRAM
20 CLEAR 5
30 PRINT "ENTER FROM 1 TO 5 CHARACTERS";
40 INPUT A$
50 PRINT "STRING "A$;" USED ALL BUT";FRE(A$);"BYTES"
60 PRINT "OF STRING SPACE."
70 GOTO 20
99 END
```

运行实例（输入T和TEST）

```
ENTER FROM 1 TO 5 CHARACTERS? T
STRING T USED ALL BUT 4 BYTES
OF STRING SPACE.
ENTER FROM 1 TO 5 CHARACTERS? TEST
STRING TEST USED ALL BUT 1 BYTES
OF STRING SPACE.
ENTER FROM 1 TO 5 CHARACTERS?
```

某些具有CLEAR功能的计算机允许CLEAR值由一个变量来确定。为了检测这个特性，对第二个测试程序作以下变化：

```
20 A=5
25 CLEAR A
```

如果解释程序接收这个程序变化，那么运行实例的结果应不改变。

替代的形式

Apple Ⅱ 和PET 都使用CRL作为CLEAR的替换形式。

其它用法

NEW

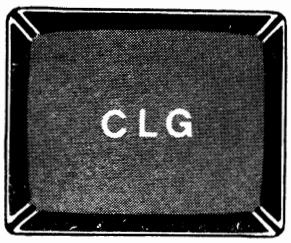
某些计算机用CLEAR作为一个专用语句来清除终端输入或输出的缓冲器中的信息。王安计算机中的CLEAR只作为一个命令。单独使用时，CLEAR清除存储器中所有程序行和变量。CLEAR与其它计算机中的NEW或SCRATCH功能相同。CLEAR P 清除程序行而只留下变量。CLEAR P n_1, n_2 将要清除标号n_1和n_2之间的程序行。如果省去n_2，那么，它将清除从n_1开始的所有行号。CLEARV 只清除存储器中的变量，而CLEAR N只清除非公共的变量。

在Microsoft BASIC5.0版本（BASIC－80）中，CLEAR不保留任何字符串空间，但可以用来保留内存顶部空间。例如，CLEAR，32000可以把所有的数字变量置成“0”，把字符串变量置空，而保留地址为32000以上的内存储单元空间，供机器语言程序之用。

参阅

FRE（A$），COMMON，NEW，SCRATCH

函　数



CLOG

Honeywell（霍尼韦尔）60系列计算机使用CLG(n)函数来计算任一个数(n)的常用对数（以10为底），数(n)的值应大于0。

测试程序

```
10 REM 'CLG' TEST PROGRAM
20 PRINT "ENTER A POSITIVE NUMBER"
30 INPUT X
40 L=CLG(X)
50 PRINT "THE COMMON LOG OF";X; "IS";L
30999 END
```

运行实例（输入100）

```
ENTER A POSITIVE NUMBER? 100
THE COMMON LOG OF 100 IS 2
```

替代的形式

某些计算机用CLOG来代替CLG。

如果你的计算机没有此功能

如果你的计算机在用上述测试程序时失败，你可以在测试程序中用LOG10、LOG。如果这两种测试程序也通不过，可以用LOG那一节中所提供的子程序（本书134页）代替之。为了用它计算常用对数（而不是自然对数），应作如下改变：

```
30150 REM * COMMON LOGARITHM SUBROUTINE * INPUT X,
      OUTPUT L
30197 L=L*.4342945
```

为了和测试程序一起使用这个子程序，把测试程序40语句改为：

40 GOSUB 30150

换算系数

为了把常用对数转换成自然对数，把常用对数的值乘以2.302585倍。

例如，X = CLG (N) * 2.302585

为了把自然对数转换成常用对数，把自然对数的值乘以0.4342945。

参阅

LOG10，LOG

函 数

CLK

在DEC公司的BASIC—PLUS—2 和德克萨斯仪器公司的990机BASIC中，CLK$与PRINT语句一起用来指示以小时（0～24）、分和秒（hh：mm：ss）为单位的时刻。计算机自动地在小时和分的数值后面加入一个冒号（：），然后作为一个字符串把时间打印出来。

例如，PRINT CLKS打印出一个22：19：15这样的形式的结果，它表示现在的时间是下午10点19分15秒。

测试程序

```
10 REM 'CLK$' TEST PROGRAM
20 PRINT "THE CURRENT TIME IS ";
30 PRINT CLK$
40 PRINT "'CLK$' PASSED THE TEST IF A SIX DIGIT NUMBER
   IS PRINTED"
99 END
```

运行实例（典型的例子）

```
THE CURRENT TIME IS 10:28:45
'CLK$' PASSED THE TEST IF A SIX DIGIT NUMBER IS PRINTED
```

替代的形式

Sperry Univac 系统9 型机的BASIC使用CLK(n) 函数来表示时间（hhmmss）。要求在CLK后面写上一个数学表达式（写在括号中），尽管它对CLK函数的结果没有影响。把测试程序的30语句改成30 PRINT CLK（0），并运行该程序，以检查你的计算机能否接受CLK。

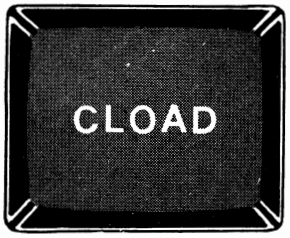
其它用法

没有

参阅

TIME，TIME$

命 令
语 句



CLOAD是某些解释程序（例如Micorsoft BASIC）使用的专用命令，它可以把程序从盒式磁带装入计算机。

测试程序

```
10 REM  CLOAD' TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE CLOAD FEATURE"
99 END
```

把上面的程序从键盘敲入到计算机中，再把该程序存储到盒式磁带上。（参阅CSAVE的的详细叙述）。

一旦该程序记录到盒式磁带上后，NEW或SCRACH，或者相应的语句来清除计算机的存储器。

反绕该磁带，然后把录音机置成“Play”状态，打入CLOAD命令。

盒式录音机的马达是由计算机来控制的，并在“装入”之前把马达接通，在“装入”之后把马达断开。盒式磁带将“重现”该程序，并把该程序装入计算机。

列程序清单来验证计算机内存中的程序和原来输入的程序是否完全一样。如果一切正常，就运行该程序。

运行实例

```
THIS PROGRAM TESTS THE CLOAD FEATURE
```

某些具有CLOAD功能的计算机使用CLOAD“程序名”仅将盒式磁带中具有该名字的程序装入。用程序名来识别一个特定的程序，它可以含有一个以上字母或数字，但是计算机只识别第一个字符。

用CSAVE“A”把测试程序记录在盒式磁带上（参阅CSAVE），清除计算机内存，然后用CLOAD“A”装回计算机中。把程序清单打印出来以检查可能发生的错误。

某些具有CLOAD功能的计算机使用CLOAD？“程序名”把存储在计算机中的程序与存放在盒式磁带上以该名字标识的程序进行比较。计算机逐位地把两个程序进行比较，如果有差异，则打印一个错误信息。这样，可以使磁带上的内容与存储器中的内容进行比较，以便在清除两者中任何一个之前验证一下你是否成功地执行了CSAVE或CLOAD。例如，如果将程序成功地（无误地）存在磁带上之后，就可以清除内存中的程序。

使用CLOAD？“A”命令检查在盒式磁带上的测试程序（用程序名“A”来存储）。如果没有打印一个错误信息，那么两个程序是相同的。

把这一行加到存放在计算机中的测试程序上。

30 REM EXTRA LINE

再使用命令CLOAD？“A”再来检查存放在盒式磁带上的“A”程序。此时会打印出一个错误信息，它指出存放在计算机中的程序和存放在磁带上的程序之间有差异。

某些使用CLOAD的计算机用CLOAD＊（数组名）作为一个命令，该命令用来将存放在盒式磁带上的以该数组名命名的一个数组装入计算机。例如:CLOAD*A意思是“装入数组A”。

CLOAD*（数组名）也可以作为程序语句，因此，数组数据可以在程序执行过程中装入。

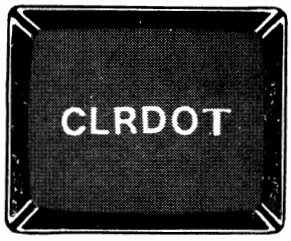
其它用法

没有

参阅

CSAVE, LIST, CHAIN, RECALL, APPEND

语　句



Sweden's ABC 80使用CLRDOT语句作为一个图形特性来“关掉”显示屏幕上的一个图形块。

被“关掉”的块由CLRDOT语句后面的L，C坐标指定。L指定行（在图形方式中为0～71），而C指定列(在图形方式中为2～79）。

例如，CLRDOT 9，15　使计算机关掉屏幕左上角的第十行,第十六列的块。要打开图形块,可参阅SETDOT。

测试程序

```
10 REM 'CLRDOT' TEST PROGRAM
20 PRINT CHR$(12)     'CLEARS SCREEN
30 PRINT "CLRDOT PASSED THE TEST IF A LINE APPEARS"
40 PRINT "AND THEN DISAPPEARS."
50 FOR T=1 TO 2000 : NEXT T
60 PRINT CLR$ (12)
70 FOR R=0 TO 23
80 PRINT CUR(R,0); CHR$ (151);
90 NEXT R
100 R=5
110 FOR C=2 TO 35
120 SETDOT R,C
130 NEXT C
140 FOR T=1 TO 2000 : NEXT T
150 FOR C=2 TO 35
160 CLRDOT R,C
170 NEXT C
180 FOR T=1 TO 2000 : NEXT T
999 END
```

运行实例

```
CLRDOT PASSED THE TEST IF A LINE APPEARS
AND THEN DISAPPEARS.
```

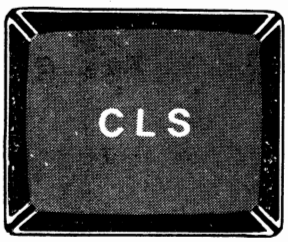
其它用法

没有。

参阅

RESET，SETDOT，SET，::

命 令
语 句



CLS（清除屏幕）命令用来执行与许多键盘上的CLEAR键相同的功能。它可以立即清除整个屏幕而不用人工干扰程序。CLS也可以作为一个程序语句来使用，在开始一个图形显示或者一个新的页之前清除屏幕。

测试程序

```
10 REM 'CLS' TEST PROGRAM
20 FOR X=1 TO 15
30 PRINT "THIS LINE SHOULD DISAPPEAR"
40 NEXT X
50 CLS
60 PRINT "IF THIS IS ALL THAT'S ON THE SCREEN"
70 PRINT "THE CLS STATEMENT PASSED THE TEST"
99 END
```

运行实例

```
IF THIS IS ALL THAT'S ON THE SCREEN
THE CLS STATEMENT PASSED THE TEST
```

如果你的计算机无此功能

许多显示屏可以用一个ASCII字符来“清除屏幕”。

试把这个语句代替测试程序中的50语句。

```
50  PRINT CHR$(24)
```

如果用CHR$(24)不成功(由于与某些生产家使用的ASCII码不相符)，试用下面这个程序来寻找能实现屏幕清除的ASCII码。

测试程序

```
10 REM ASCII CLEAR SCREEN SEARCH
20 FOR X=0 TO 128
30 PRINT "ASCII CODE";X;
40 PRINT CHR$(X)
50 FOR Y=1 TO 200
60 NEXT Y
70 NEXT X
99 END
```

其它用法

TRS—80 COLOR COMPUTER用CLS(n)来清除屏幕并设置背景颜色。下面0～8个数指出哪个数对应“接通”哪种颜色。如果使用CLS时没有给出一个数，那么使用的颜色就是当前背景的颜色。

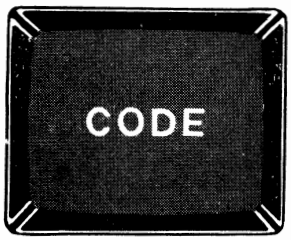
0 = 黑色	3 = 蓝色	6 = 深蓝色
1 = 绿色	4 = 红色	7 = 深红色
2 = 黄色	5 = 浅黄色	8 = 桔色

要注意CLS和CLEAR语句是完全不同的。

参阅

CHR$(X)

函 数



Sinclair ZX80使用CODE函数来把一个字符转换成“数字码”。Sinclair不使用所有其它计算机都使用的ASCII码。

例如，PRINT CODE (“A”) 打印38。PRINT CODE (A$)打印出存放在字符串变量A$中字符串第一个字符的代码。

测试程序

```
10 REM 'CODE' TEST PROGRAM
20 PRINT "THE NUMERIC CODE FOR THE LETTER A IS";
30 PRINT CODE("A")
40 PRINT
50 PRINT "TYPE ANY LETTER, NUMBER OR CHARACTER";
60 INPUT A$
70 PRINT "THE NUMERIC CODE FOR ";A$" IS";CODE(A$)
99 END
```

运行实例

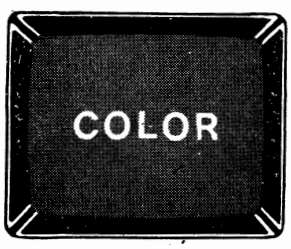
```
THE NUMERIC CODE FOR THE LETTER A IS 38

TYPE ANY LETTER, NUMBER OR CHARACTER? *
THE NUMERIC CODE FOR * IS 20
```

参阅

ASC, CHR$

命 令
语 句



COLOR语句在APPLE Ⅱ BASIC中作为一个专门的特性,用它来指定作图语句PLOT、HLIN-AT和VLIN-AT。执行时，显示屏幕上的颜色,每次执行一个图形语句时,显示同一种颜色。为了改变颜色,必须由COLOR语句来指定新的颜色。

计算机显示16种不同的颜色,而每种颜色都分配一个数字(从0～15)。它们是：

0	黑色	8	褐色
1	深红色	9	桔色
2	深蓝色	10	灰色
3	紫色	11	粉红色
4	深绿色	12	绿色
5	灰色	13	黄色
6	中蓝色	14	无色
7	浅蓝色	15	白色

在COLOR和COLOR值之间必须有一个等号（=）。颜色值可以是数，也可以是数值变量。

例如，COLOR = 13 为下一个作图语句选择黄色。COLOR既可以作为一个命令，又可以作为一个程序语句。

测试程序

```
10 REM 'COLOR' TEST PROGRAM
20 GR
30 FOR X=0 TO 15
40 COLOR = X
50 Y=X*2
60 HLIN 0,39 AT Y
70 NEXT X
99 END
```

运行实例

如果你的计算机接收此测试程序，那么将会出现用16种颜色中的每一种显示出贯穿屏幕的垂直线。

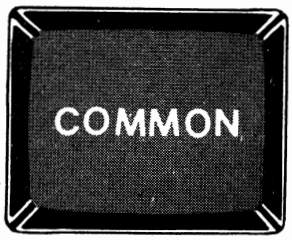
其它用法

参阅ATARI和TRS—80 COLOR BASIC简介

参阅

GR，PLOT，HLIN-AT，VLIN-AT

语　句



COM

在某些计算机中，COMMON语句用来把一个数值从一个程序送到另一个程序。如果两个（或更多）程序中的每一个包含类似COMMON语句，当第二个程序链接到第一个程序时，存放在COMMON命名的变量中的现行值对第二个程序也有效。（参阅CHAIN）。

例如：如果第一个程序含有COMMON语句：

10　COMMON A, B, C, I, J

而第二个程序也含有COMMON语句：

30　COMMON X, Y, Z, T, U

那末，A的最终值变成X的初始值，B的最终值变成Y的初始值，如此等等。

替代形式

某些计算机（如王安2200）用COM作为COMMON的缩写形式。

测试程序

用“TEST”为名，将这段程序存放在磁盘或磁带上。

```
10 REM * TEST * PROGRAM
20 COMMON A
30 PRINT "THE 'TEST' PROGRAM RECEIVED A";A
40 A=A*2
50 CHAIN MAIN, 50
99 END
```

现在，把下面的程序输入到计算机，然后，以“MAIN”为名字把该程序存放在同一个磁带上TEST之后，或者存放在磁盘上。

```
10 REM * MAIN * PROGRAM
20 COMMON A
30 A = 5
40 CHAIN TEST
50 PRINT "AND RETURNED A";A
99 END
```

准备好将要被程序命令读取的磁盘或磁带，然后运行。

运行实例

```
THE 'TEST' PROGRAM RECEIVED A 5
AND RETURNED A 10
```

其它用法

COM语句（在王安机上）也可以确定字符串变量长度，字符串最多可有64个字符（它的隐含长度为16个字符）。

例如，COM A$（100）1，B$8建立一个字符串数组A$（n），它包含100个元素，每个元素包含一个字符串变量B$的长度为8个字符串。

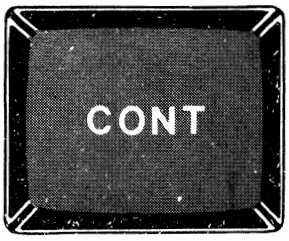
参阅

CHAIN, DIM

由于执行一个STOP语句，或者使用键盘上的BREAK键而导致程序中断后，使用CONT命令可以重新启动使程序执行。不象RUN命令，使程序从起点开始执行，CONT命令从中断的行的后面一行开始再继续执行，而变量不重置为零。

CONT不能作为程序语句使用，因为只有当程序STOP时才使用它。

命　令



CON
CO
C.

测试程序

```
10 REM 'CONT' TEST PROGRAM
20 PRINT "ENTER THE 'CONT' COMMAND"
30 STOP
40 PRINT "THE CONT COMMAND PASSED THE TEST"
99 END
```

运行实例

```
ENTER THE 'CONT' COMMAND
BREAK AT 30
CONT
THE CONT COMMAND PASSED THE TEST
```

替代形式

CONT命令的几种缩写形式是CON，CO和C.。试一下把CONT的每种缩写用于测试程序，找出你的计算机所能接收的那一个。

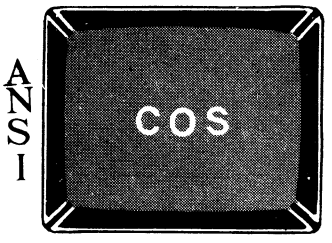
其它用法

没有

参阅

STOP，END，RUN

函　数

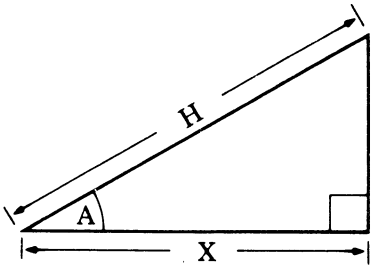


COSD
COSG

COS(A)函数可以计算角A的COSINE（余弦）值，该值是以弧度为单位（而不是以角度为单位）来表示的。1 弧度≈57 度。

在直角三角形中，一个角的 cosine(cos)定义为该角的邻边的长度与斜边长度之比。

COS(A)=X/H



COS的反三角函数为ARCCOS即COS^{-1}。当已知一个角的COS，或者已知两边之比（X/H）时，用ARCCOS（缩写为ACS）可求出该角的值。

测试程序

```
10 REM 'COS' TEST PROGRAM
20 PRINT "ENTER AN ANGLE (EXPRESSED IN RADIANS)";
30 INPUT R
40 Y=COS(R)
50 PRINT "THE COSINE OF A";R;"RADIAN ANGLE IS";Y
30999 END
```

运行实例（输入1 ）

```
ENTER AN ANGLE (EXPRESSED IN RADIANS)? 1
THE COSINE OF A 1 RADIAN ANGLE IS .540302
```

为了从角度转换成弧度，可以把以角度为单位的值乘以0.0174533倍。

例如：R=COS(A * .0174533)

为了从弧度转换成角度，可以把弧度乘以57.29578倍。

某些计算机是以角度或梯度为度量单位（100梯度=90度）的。这些计算机用度数单位时使用COSD，用梯度为单位时使用COSG。把这两个函数的每一个，替换到测试程序的行40中，并且让它运行。在运行中输入1，在用COSD时应该产生0.999848，用COSG时应产生0.999877。

如果你的计算机没有此功能

如果你的解释程序不具有COS函数，你可以用下面的子程序来代替。

必须将SIN一节中的子程序（本书226页）和下面的程序一起运行（为了节约篇幅，此处不重复印出）。

```
30000 GOTO 30999
30330 REM * COSINE SUBROUTINE * INPUT X IN RADIANS,
      OUTPUT Y
30332 REM ALSO USES C, D, W AND Z INTERNALLY
30334 X=X*57.29578
30336 W=ABS(X)/X
30338 D=X
30340 X=X+90
30342 GOSUB 30366
30344 X=D/57.29578
30346 IF Z<>-1 THEN 30352
30348 IF W<>1 THEN 30352
30350 Y=-Y
30352 RETURN
```

为了把这个子程序与测试程序一起用来求一个角(以弧度为单位来表示)的COSine值，将测试程序作如下改变：

```
35 X=R
40 GOSUB 30330
```

为了求一个角（以角度为单位来表示的)的COSine的值,或者是删除行号30334,或者把行号40改成：

```
40 GOSUB 30336
```

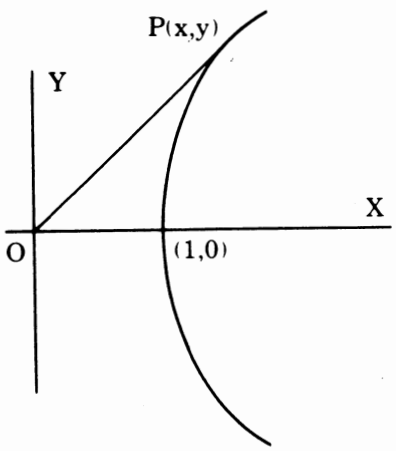
其它用法

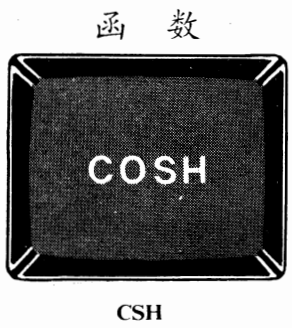
某些（很少见）解释程序可以自动地转换成角度。

参阅

SIN, ASN, ATN, TAN, ACS, COSH, SINH, TANH

COSH函数可以用来计算一个数的双曲余弦。双曲函数表示以双曲线为基础的各种关系，它类似于在一个圆上表示三角函数的方法。





如果在一段双曲线上（即X*X- Y*Y= 1 的图形）从原点到曲线一点P作一直线（见图），形成一个具有N/ 2 面积的一个区域。COSH(N)将给出交点X坐标的值。〔SINH(N)将给出Y的坐标值。〕

与三角函数不同，N不是一个角的度量单位，因此它不是以角度或弧度为单位的。N可以是任意一个实数，可为正，也可为负，但是COSH(N)总是一大于或等于1 的数。

测试程序

```
10 REM 'COSH' TEST PROGRAM
20 PRINT "ENTER A VALUE";
30 INPUT N
40 C=COSH(N)
50 PRINT "THE HYPERBOLIC COSINE OF";N;"IS";C
30999 END
```

运行实例（输入1 ）

```
ENTER A VALUE? 1
THE HYPERBOLIC COSINE OF 1 IS 1.54308
```

如果你的计算机没有此功能

如果你的计算机不接受COSH语句，你可以用EXP函数来代替，如下所示：

```
40 C=.5 *(EXP(N)+EXP(-N))
```

如果你的计算机也没有EXP函数的话，可用下面的子程序来代替。该子程序也必须和EXP下面的子程序一起使用。

```
30000 GOTO 30999
30430 REM * COSH SUBROUTINE * INPUT N, OUTPUT C
30432 REM ALSO USES A, B, E, L AND X INTERNALLY
30434 X=N
30436 GOSUB 30200
30438 C=E
30440 X=-N
30442 GOSUB 30200
30444 C=.5*(C+E)
30446 RETURN
```

为了使用这个子程序，在测试程序中可以做如下变化：

40 GOSUB 30430

替代的形式

Harris BASIC—V用CSH来代替COSH函数。

其它用法

没有

参阅

SINH， ANH， EXP

函 数

COUNT是ACORN ATOM计算机中的一种函数，它是统计从前一次输出回车后打印字符个数的“计数”。COUNT类似于POS（参阅POS的详细资料）。

测试程序

```
10 REM 'COUNT' TEST PROGRAM
20 PRINT "THIS LINE HAS A CHARACTER COUNT OF";
30 K = COUNT
40 PRINT K+3
99 END
```

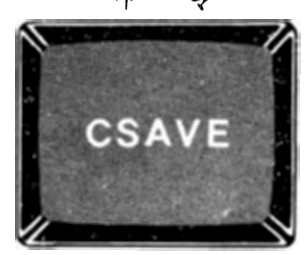
运行实例

```
THIS LINE HAS A CHARACTER COUNT OF 37
```

参阅

POS

命 令



某些计算机（例如，Microsoft解释程序）使用CSAVE命令把程序从计算机的内存记录到盒式磁带上。

测试程序

```
10 REM 'CSAVE  TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE
   CSAVE FEATURE"
99 END
```

为了记录准备好盒式磁带录音机，打入CSAVE命令。计算机将控制录音机的操作，在记录周期的开始前先启动录音机马达，在记录周期结束后关闭马达。

一旦把程序记录在盒式磁带上，打入NEW（或任何其它需要的命令）来清除内存的程序。把这个程序从磁带上再装回计算机（参阅CLOAD）。打印程序清单以验证存放在计算机内存中的程序和原来敲入的那个程序是否相同。

运行实例

```
THIS PROGRAM TESTS THE CSAVE FEATURE
```

在某些计算机中用CSAVE（程序名）来将一个特定的名称分配给正在记录在盒式磁带上的那个程序。文件名可以含有一个或多个字母、数字，或者可选择的ASCII符号，但是计算机只识别第一个字符。程序名标识一个程序，以后通过CLOAD（程序名）命令来检索此程序。

使用CSAVE “A” 把测试程序记录在盒式磁带上，清除内存的内容，然后，用CLOAD “A” 命令把程序装回计算机。

把程序列表以便检查可能出现的错误。

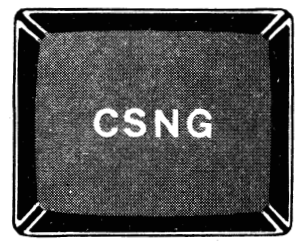
其它用法

在Microsoft, BASIC 5.0版本中使用CSAVE* 命令，把一个数值数组的值存放在磁带上。

参阅

CLOAD, LIST, STORE

函 数



CSNG用来将已定义为“双精度”的数或数值变量又还原为“单精度”。在CSNG函数中使用的变量，如果再使用它们时不用CSNG函数，则仍为它们原来的双精度状态。

单精度变量能够存储含有不多于7位数字的数（仅打印6位数字）。双精度意味着精确到17位数字。如果CSNG与一个包含多于6位数字的双精度一起使用，那末，这个数要舍入为6位有效数。

测试程序

```
10 REM  CSNG  TEST PROGRAM
20 DEFDBL X
30 X=1234567890123456
40 PRINT "CSNG CHANGES THE VALUE OF X FROM";X;
   "TO";CSNG(X)
50 PRINT "AND BACK TO THE VALUE OF";X;"WHEN REMOVED"
99 END
```

运行实例

```
CSNG CHANGES THE VALUE OF X FROM 1234567890123456 TO
  1.23457E+15
AND BACK TO THE VALUE OF 1234567890123456 WHEN REMOVED
```

其它用法

没有

参阅

DEFSNG，DEFDBL，DEFINT，CDBL，!，#，%，CINT

函 数

Sweden's ABC—80计算机把CUR函数和PRINT语句一起使用。它把下一个打印字符的位置确定在指定的位置L, C上。PRINT CUR(L, C)产生的结果类似于

```
PRINT AT (64*L+C) or PRINT @ (64*L+C)
```

测试程序

```
10 REM 'CUR' PROGRAM
20 PRINT CHR$(12) 'CLEARS SCREEN ON ABC 80
30 PRINT CUR(12,16); "MIDDLE"
40 PRINT "CUR PASSED IF MIDDLE IS IN THE CENTER"
99 END
```

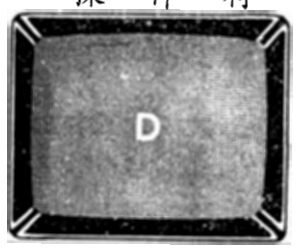
运行实例

```
                MIDDLE
CUR PASSED IF MIDDLE IS IN THE CENTER
```

参阅

PRINT AT, @, LOCATE

操 作 符



D操作符表示“指数”或“标准科学记数法”中的“双精度”。

例如：1.23456789D+20。

以单精度表示的数在指数记数法中用字母“E”来书写。

例如：1.234E+20

测试程序

```
10 REM 'D' DOUBLE PRECISION EXPONENT TEST PROGRAM
20 A#=12345678901234567890
30 PRINT "EXPONENTIAL NOTATION 'D' PASSED THE TEST IF"
40 PRINT A#; "CONTAINS THE LETTER 'D'"
99 END
```

运行实例

```
EXPONENTIAL NOTATION 'D' PASSED THE TEST IF
 1.2345678901234567D+18 CONTAINS THE LETTER 'D'
```

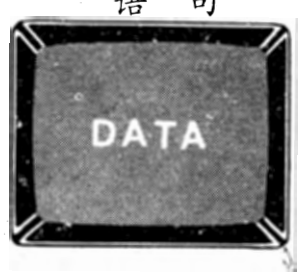
其他用法

字母“D”如同字母表中所有的字母一样，可被所有计算机用作代表一个数值变量。

参阅

E，#，!，DEFDBL，DEFSNG

语 句



ANSI

DAT

D.

DATA语句含有被READ语句读的数据。DATA语句中的各项必须由逗号分隔，可以为正数，也可以为负数。

测试程序# 1

```
10 REM 'DATA' TEST PROGRAM
20 DATA 20,-10,,5
30 READ A,B,C
40 D=A+B+C
50 PRINT "D =";D
60 PRINT "DATA PASSED THE TEST IF D
   = 10.5"
99 END
```

运行实例

```
D = 10.5
DATA PASSED THE TEST IF D = 10.5
```

大多数计算机允许在DATA语句中有字符串。有些计算机要求把字符串放在引号里，而其它一些计算机仅当字符串最前面或字符串最后面包括有一个空格、逗号或冒号时才需要用引号括起来。

测试程序=2

```
10 REM 'DATA' TEST PROGRAM USING STRINGS
20 DATA "LINE NUMBER",20,"PASSED"
30 READ A$,A,B$
40 PRINT "DATA STATEMENT IN ";A$;A;B$;" THE TEST"
99 END
```

运行实例

把行20中的字符串的引号消去，然后再运行，查看在你的解释程序中是否需要引号。

可以把DATA语句放在一个程序中任一个位置上。

替代的形式

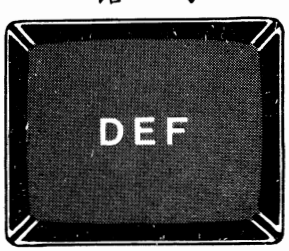
DAT（由PDP—8 E使用的）和D.（Tiny—BASIC使用的）都可以作为DATA的缩写。

参阅

READ，RESTORE

语 句

ANSI



DEF语句可以使用户定义（建立）一些新函数（大多数计算机都有一些内部函数）。新函数可以和任一种内部函数一样地使用。

例如：DEF FNA（R）= R * R * 3.14159。在这个例子里把表达式R * R * 3.14159定义成函数FNA。（公式是用来求圆的面积的，通常写作 πr^2）。FN是Functicn（函数）一词的缩写。在DEF语句中，FN的后面一定要跟着任一个合法的数字变量。这个例子里用“A”来标识作为圆面积的函数FNA，然而也可以使用其它任一个变量。一旦定义了一个函数，在同一个程序中一般不能对它再定义。

放在括号里的变量〔上面的（R）〕必须和语句中等号右边的变量一致。通常把这些变量称为“虚变量”（或形式变量）。

用DEF语句定义FN（变量）函数中这一操作，可以用来处理任一个数或数值变量。

例如：

```
10 X=2
20 DEF FNA(N)=3*N-1
30 PRINT FNA(X)
```

把这个例子里的FN函数命名为“A”（FNA），在20行中则把等式3＊ N－1赋给FNA。每当执行FNA时，跟着FNA后面的数值变量（X）代替了DEF语句中的“虚变量”（N）。

测试程序# 1

```
10 REM 'DEF' TEST PROGRAM
20 PRINT "ENTER THE RADIUS OF A CIRCLE (IN INCHES)";
30 INPUT R
40 DEF FNC(X)=2*3.14159*X
50 PRINT "THE CIRCUMFERENCE OF A CIRCLE"
60 PRINT "WITH A RADIUS OF";R;"INCHES IS";FNC(R);
   "INCHES"
99 END
```

运行实例（输入4）

```
ENTER THE RADIUS OF A CIRCLE (IN INCHES)? 4
THE CIRCUMFERENCE OF A CIRCLE
WITH A RADIUS OF 4 INCHES IS 25.1327 INCHES
```

有些计算机允许在DEF表达式中有多个（多于1个）变量。这些变量中的每一个都必须在FN（变量）函数的后面列出。

测试程序# 2

```
10 REM 'DEF' MULTIPLE VARIABLE TEST PROGRAM
20 DEF FNA(X,Y)=(X+Y)/2
30 PRINT "ENTER ANY TWO NUMBERS";
40 INPUT X,Y
50 A=FNA(X,Y)
60 PRINT "THE AVERAGE OF";X;"AND";Y;"IS";A
999 END
```

运行实例（输入20和40）

```
ENTER ANY TWO NUMBERS? 20,40
THE AVERAGE OF 20 AND 40 IS 30
```

有些计算机允许在几行中定义同一个函数。在下面的测试程序中，如果变量X的值小于10，则把函数FNA定义成X＊2，或如果X的值大于或等于10，则把函数FNA 定义X/2。

测试程序＃3

```
10 REM 'DEF' REQUIRING MORE THAN ONE LINE
20 PRINT "ENTER A VALUE FOR X THAT IS GREATER
   OR LESS THAN 10";
30 INPUT X
40 DEF FNA(X)
50 FNA=X*2
60 IF X <10 THEN 80
70 FNA=X/2
80 FNEND
90 PRINT "THE VALUE OF THE FUNCTION IS";FNA(X)
999 END
```

运行实例（输入12）

```
ENTER A VALUE FOR X THAT IS GREATER OR LESS THAN 10? 12
THE VALUE OF THE FUNCTION IS 6
```

测试程序3中的FNEND语句告诉计算机停止定义函数FNA。多行的DEF语句必须用FNEND语句来结束，计算机不允许转移到多行DEF语句（本例40—80行），也不允许从多行DEF语句转出去。更详细的资料可参阅FNEND。

如果你的计算机没有此功能

如果你的计算机没有DEF的功能，可以用一个含有相同等式的子程序来代替FN。

例如：测试程序2中的DEF语句可用下面的子程序来代替：

```
100 A=(X+Y)/2
110 RETURN
```

测试程序做如下的变化：

删去程序行20后再加上：

```
50 GOSUB 100
70 GOTO 999
```

“虚”变量不能和GOSUB语句一起使用，因此，子程序中的实际变量在每次调用前必须是给定的。

有些BASIC允许用DEF语句定义字符串函数。例如：10DEF FNLS(A$)=LEFT$(A$,1)函数取某个字符串A$的第一个字符（这可用于检查键盘的输入）。

参阅

FN，FNEND，GOSUB，RETURN

语 句

DEFDBL语句用来定义（说明）一个或多个变量为“双精度”。双精度变量可以存储精确到17位的数（只打印16位数字）。单精度的变量一般精确到6位数字。

注意：DEFDBL语句只应该用在单精度不能满足要求的地方，因为双精度变量需要更多的内存空间，而且处理双精度变量也需要更多的时间。在大多数计算机中，DEFDBL语句必须在DEFDBL语句中列出的变量被赋值之前执行。

测试程序＃1

```
10 REM 'DEFDBL' TEST PROGRAM
20 A=1.23456789012345G
30 PRINT "DEFDBL IN LINE 50 CHANGED THE VALUE OF
   VARIABLE 'A'"
40 PRINT "FROM";A;"TO";
50 DEFDBL A
60 A=1.23456789012345G
70 PRINT A
99 END
```

运行实例

```
DEFDBL IN LINE 50 CHANGED THE VALUE OF VARIABLE 'A'
FROM 1.23457 TO 1.23456789012345G
```

具有DEFDBL功能的大多数计算机允许在一个DEFDBL语句中指定多个“双精度”变量。例如：DEFDBL A,F,M把变量A、F和M定义成双精度的变量，而DEFDBL A—M则是把A到M的字母打头的全部变量定义成双精度的变量。

测试程序＃2

```
10 REM 'DEFDBL' (WITH MULTIPLE VARIABLES) TEST PROGRAM
20 DEFDBL A,G,L-N
30 A=1/3
40 G=2/3
50 L=1/9
60 M=1.23456789012345G7D+38
70 N=-1.23456789012345G7D+38
80 PRINT "DEFDBL PASSED THE TEST IF THE FOLLOWING"
90 PRINT "NUMBERS CONTAIN MORE THAN 7 DIGITS:"
100 PRINT A;G;L;M;N
999 END
```

运行实例

```
DEFDBL PASSED THE TEST IF THE FOLLOWING
NUMBERS CONTAIN MORE THAN 7 DIGITS:
  .3333333333333333  6666666666666667  .1111111111111111
  1.2345678901234567D+38 -1.2345678901234567D+38
```

“+38”之前的D是与指数记数法中的E含义相同，但它表示这个数是“双精度”的。

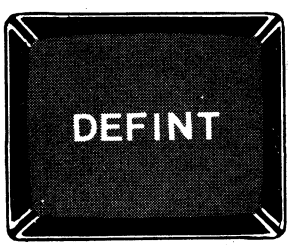
有些计算机可能在打印运行实例中的前三个数值时不象上面表示的那样，因为它们的计算是用单精度进行的。在具有双精度说明符号（即# 符号）的计算机中，这个问题可以得到解决。在30、40和50行中，在每个除数的后面加一个# 号就是产生的正确结果。

```
30 A=1/3#
40 G=2/3#
50 L=1/9#
```

参阅

DEFSNG，DEFINT，#，%，!，CDBL，CSNG，CINT，D and E

语　句



DEFINT 语句用来定义（说明）在DEFINT语句中列出的变量是整型的。定义成整型的变量存放赋给它的整数值。由于存储整数值要比存储非整数值所需要的内存单元要少，所以这在大的程序中是特别有用的。

使用DEFINT语句的潜在缺点是很多解释程序不能处理比INT函数所允许的数值（一般是从－32767到＋32767）大的数值。

在DEFINF语句中列出的变量被赋值之前，计算机必须先执行DEFINT语句。

测试程序＝1

```
10 REM 'DEFINT' TEST PROGRAM
20 DEFINT A
30 A=12.68
40 B=12.68
50 IF A=12 THEN 70
60 GOTO 80
70 IF B=12.68 THEN 100
80 PRINT "DEFINT FAILED THE TEST LINE 20"
90 GOTO 999
100 PRINT "THE DEFINT STATEMENT PASSED THE TEST
    IN LINE 20 BY"
110 PRINT "CHANGING THE VALUE OF VARIABLE A FROM";B;
    "TO";A
999 END
```

运行实例

```
THE DEFINT STATEMENT PASSED THE TEST IN LINE 20 BY
CHANGING THE VALUE OF VARIABLE A FROM 12.68 TO 12
```

具有DEFINT功能的大多数计算机也允许在一个DEFINT语句里指定多个变量(用逗号分隔)。例如：DEFINT A,F,M把变量A,F,M定义为整型。DEFINT A—M则把从字母A到字母M开头的所有变量定义成整型。

测试程序＝2

```
10 REM 'DEFINT' (WITH MULTIPLE VARIABLES) TEST PROGRAM
20 DEFINT A,G,L-N
30 A=6.25
40 B=21.42
50 G=-6.19
60 L=4.001
70 M=32000.999
80 N=14.8
90 PRINT "IF THE NUMBERS";A;G;L;M;N;" ARE INTEGERS,"
100 PRINT "AND THE NUMBER";B;"IS A DECIMAL, THEN DEFINT"
110 PRINT "PASSED THE MULTIPLE VARIABLE TEST IN LINE 20."
999 END
```

运行实例

```
IF THE NUMBERS 6 -7 4 32000 14 ARE INTEGERS,
AND THE NUMBER 21.42 IS A DECIMAL, THEN DEFINT
PASSED THE MULTIPLE VARIABLE TEST IN LINE 20.
```

如果解释程序有一个双精度的说明符(即Microsoft BASIC中的＃号)和(或)单精度说明符（在Microsoft BASIC中是!)号，这两个说明符中之一分配给在DEFINT语句中已列出的变量，该变量作为双精度（或单精度），因为说明符号优先于DEFINT语句。详见＃、！和％操作符。

测试程序＝3

```
10 REM 'DEFINT' TEST PROGRAM
20 REM USES DOUBLE-PRECISION TYPE DECLARATION
   CHARACTER'#'
30 DEFINT A,B
40 A=9.123456789012345
50 B#=9.123456789012345
60 IF A=B# THEN 110
70 PRINT "A =";A
80 PRINT "B# =";B#
90 PRINT "THE TEST PASSED, SHOWING # OVER-RIDING DEFINT"
100 GOTO 999
110 PRINT "THE # CHARACTER OVER-RIDE FEATURE FAILED
    THE TEST"
999 END
```

运行实例

```
A = 9
B# = 9.123456789012345
THE TEST PASSED, SHOWING # OVER-RIDING DEFINT
```

参阅

INT，#，DEFSNG，DEFDBL，CINT，CSNG，CDBL，% !

语 句

DEFSNG语句用来定义（说明）指定的变量为“单精度”。单精度的变量可以储存不多于7位数字的数（只打印6位数字）。双精度意味着有16位数字的精度。

因为大多数解释程序自动将变量作为单精度处理，所以在程序中DEFSNG语句用于在DEFDBL语句定义双精度变量或 DEFINT 语句定义整型变量之后，把变量重新定义为单精度。

在大多数计算机中，DEFSNG语句中列出的变量被赋值之前，必须执行DEFSNG语句。下面的20行说明X和Y都是双精度的。

测试程序# 1

```
10 REM 'DEFSNG' TEST PROGRAM
20 DEFDBL X,Y
30 X=1.23456789012345G
40 Y=X
50 PRINT "DOUBLE PRECISION VALUE OF Y=";Y
60 DEFSNG Y
70 Y=X
80 PRINT "SINGLE PRECISION VALUE OF Y=";Y
99 END
```

运行实例

```
DOUBLE PRECISION VALUE OF Y= 1.23456789012345G
SINGLE PRECISION VALUE OF Y= 1.23457
```

具有DEFSNG功能的大多数计算机允许在一个DEFSNG语句中指定多个变量（由逗号分隔）。例如：DEFSNG A, F, M把变量A, F和M定义成单精度，而DEFSNG A—M则是把从字母A到字母M开头的所有变量定义成单精度。

测试程序# 2

```
10 REM 'DEFSNG' (WITH MULTIPLE VARIABLES) TEST PROGRAM
20 DEFDBL A,G,L-N
30 GOSUB 200
40 PRINT "THE DOUBLE PRECISION VALUES OF A,G,L,M AND N
   ARE"
50 PRINT A;G;L;M;N
60 DEFSNG A,G,L-N
70 GOSUB 200
80 PRINT "THE SINGLE PRECISION VALUES OF A,G,L,M AND N
   ARE"
90 PRINT A;G;L;M;N
100 GOTO 999
200 REM SUBROUTINE
210 A=1234.567890
220 G=A/10
230 L=G/10
240 M=L/10
250 N=M/10
260 RETURN
999 END
```

运行实例

```
THE DOUBLE PRECISION VALUES OF A,G,L,M AND N ARE
 1234.56789 123.456789 12.3456789 1.23456789 .123456789
THE SINGLE PRECISION VALUES OF A,G,L,M AND N ARE
 1234.57 123.457 12.3457 1.23457 .123457
```

如果解释程序有一个双精度的说明符(在Microsoft BASIC中是# 号)和/ 或一个整型说明符(在Microsoft BASIC中是% 号),并且这两个说明符中之一分配给在DEFSNG语句中已列出的变量，该变量作为双精度（或整型）变量，因为说明符号优先于DEFSNG语句。并见# 、! 和% 操作符。

测试程序# 3

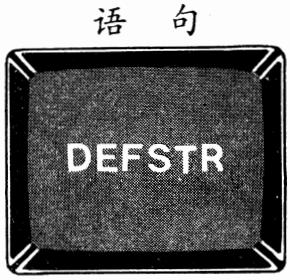
```
10 REM 'DEFSNG' TEST PROGRAM
20 REM USES DOUBLE PRECISION DECLARATION CHARACTER '#'
30 DEFSNG A,B
40 A=1.2345678901234567
50 B#=1.2345678901234567
60 IF A=B# THEN 110
70 PRINT "A =";A
80 PRINT "B# =";B#
90 PRINT "THE TEST PASSED WITH # OVER-RIDING DEFSNG"
100 GOTO 999
110 PRINT "THE # CHARACTER OVER-RIDE FEATURE FAILED THE
    TEST"
999 END
```

运行实例

```
A = 1.23457
B# = 1.2345678901234567
THE TEST PASSED WITH # OVER-RIDING DEFSNG
```

参阅

DEFINT, #, DEFDBL, !, CSNG, CDBL, CINT, %



DEFSTR语句用来指定字符串变量，在DEFSTR语句中列出的变量和用$（字符串）符号定义的字符串变量的含义相同。

在大的程序中，只确定那些需要存储字符串的变量是重要的，因为字符串变量需要的内存空间比数值变量需要的内存空间多。

在被定义为字符串变量的变量被赋于一个字符串值之前，必须先执行DEFSTR语句。

测试程序# 1

```
10 REM 'DEFSTR' TEST PROGRAM
20 A=25
30 PRINT "NUMERIC VARIABLE A =";A
40 DEFSTR A
50 A="TEST STRING"
60 PRINT "STRING VARIABLE A = ";A
99 END
```

运行实例

```
NUMERIC VARIABLE A = 25
STRING VARIABLE A = TEST STRING
```

具有DEFSTR功能的大多数计算机允许用一个DEFSTR语句来定义多个变量（用逗号来分隔）。例如：DEFSTR A, F, M把变量A, F和M定义成字符串变量。DEFSTR A—M把从字母A到字母M开头的所有变量都定义成字符串变量。

测试程序# 2

```
10 REM DEFSTR (WITH MULTIPLE VARIABLES) TEST PROGRAM
20 DEFSTR A,G,L-N
30 A="DEFSTR "
40 G="PASSED THE "
50 L="MULTIPLE VARIABLE "
60 M="TEST "
70 N="IN LINE 20."
80 PRINT A;G;L;M;N
99 END
```

运行实例

```
DEFSTR PASSED THE MULTIPLE VARIABLE TEST IN LINE 20.
```

有些解释程序要求用DIM语句或CLEAR语句指定分配给字符串的内存空间。

对于具有说明符(即%,# 或!)的解释程序，当把这些符号加在DEFSTR语句中已定义的变量上时，优先于DEFSTR函数。将第二个测试程序作如下变化，就可以测试这个特性。

```
70 N="IN LINE"
80 PRINT A;G;L;M;N;
85 A!=20
90 PRINT A!
```

加在85和90行的单精度说明符（！）优先于20行中的DEFSTR语句，你可以打印出运行结果来。

其它用法

没有

参阅

DEFDBL，DEFINT，DEFSNG，DIM，CLEAR，$，D（指数符号），E（指数符号），%（整型符号），#（双精度符号）和！（单精度符号）.

命　令
函　数
语　句

DEGREE

有些计算机（例如:Cromemco 16K扩展BASIC）把DEG当作一个命令来使用，该命令可以使计算机执行三角函数时以度数为单位（而不是以弧度为单位）。1度≈0.02弧度。

测试程序＃1

```
10 REM 'DEG COMMAND' TEST PROGRAM
20 A=SIN(1.4)
30 PRINT "THE SINE OF 1.4 RADIANS IS";A
99 END
```

运行实例

如上所示，计算机将执行此程序并计算1.4 弧度角的正弦值。

```
THE SINE OF 1.4 RADIANS IS .98545
```

打入DEG命令，然后运行。计算机将输出以1.4 度（度量单位为度）的角的正弦值。

```
THE SINE OF 1.4 RADIANS IS.024432
```

为了使计算机返回到弧度方式，我们可以打入RAD或SCR。(SCR还会清除整个程序。)

如果你的计算机没有此功能

如果你的计算机没有DEG命令，可以在程序中把度数乘以0.0174533 倍的方法 来模拟。为了把这个变换用在第一个测试程序中，可以使程序改变如下：

```
20 A=SIN(1.4*.0174533)
```

其它用法

在TRS—80袖珍计算机中 DEG把角度的度量单位从度、分、秒的形式转换成度数和小数形式。

例如：输入DEG33.4025

DEG把33° 40′ 25″ 转换成33.6736 度。

在袖珍计算机上用DMS可以反 过 来 把 十进制的度数转换成度、分、秒的形式。

例如：输入DMS 33.6736

DMS把33.6736 度转换成30° 40′ 25″ 。

少数计算机（例如那些使用MAX BASIC 的）具有作为内部函数的DEG (n)，它可以把以弧度为单位表示的值 (n) 转换成以度数为单位的值。

测试程序＝ 2

```
10 REM 'DEG FUNCTION' TEST PROGRAM
20 PRINT "ENTER AN ANGLE (EXPRESSED IN RADIANS)";
30 INPUT A
40 B=DEG(A)
50 PRINT "THE RADIAN ANGLE OF";A;"IS EQUAL TO";B;"DEGREES
99 END
```

运行实例（输入1.4 ）

```
ENTER AN ANGLE (EXPRESSED IN RADIANS)? 1.4
THE RADIAN ANGLE OF 1.4 IS EQUAL TO 80.2141 DEGREES
```

替代的形式

有些计算机（例如Sharp和TRS—80袖珍机)把DEGREE当作一个语句来使用，该语句使计算机在进行三角计算时采用度为单位。

假如你的计算机没有此功能

假如你的计算机没有DEG函数，可以把弧度值乘以57.29578。为了在第二个测试程序中实现这个转换，使程序作以下变化：

```
40 B=A*57.29578
```

参阅

SIN，COS，TAN，ATN，RAD，ASN，ACS

命　令

DEL

DELETE 命令用来把指定的程序行从计算机的内存中“删去”。

测试程序

```
10 REM 'DELETE' TEST PROGRAM
20 PRINT "LINE 20"
30 PRINT "LINE 30"
40 PRINT "LINE 40"
50 PRINT "LINE 50"
60 PRINT "LINE 60"
70 PRINT "LINE 70 - END OF DELETE TEST"
99 END
```

运行该程序以确保已把所有的行正确地输入到计算机中了。

运行实例

```
LINE 20
LINE 30
LINE 40
LINE 50
LINE 60
LINE 70 - END OF DELETE TEST
```

用DELETE（行号）命令可以把一个程序行从计算机的内存中删去。为了测试这个特性，我们试用命令DELETE 50，然后运行这个程序。这个命令应该删去了“50行”的打印语句。用列清单和运行的方法进行检查。

一些计算机使用DELETE(行号=—行号=)命令可以把更多的程序从计算机的内存中删去。由这个命令确定的行号范围内的所有行都被删去。为了测试这一特性，试用命令DELETE30—40，然后运行该程序。30和40行都应被删去。有些计算机要求第一个和/或最后一个行号是实际上存在的。在其它计算机中，即使最后的行号没有用到，也删去该范围内的所有行。

一些计算机用DELETE—（行号）删去该程序中从第一个行号开始到DELETE命令中指定的行号为止的所有行。为了测试该特性，试用命令DELETE—60然后运行程序。除了70和99行外的所有行都应该删去。

具有DELETE特性的一些计算机，可以列出成组的行号和个别的行号，用逗号分隔。用它删去相应的行。

例如：DELETE 20，40—50，从该程序中删去了20，40，50行。为了测试这一特性，重新输入测试程序并试用命令DELETE 20,40—60。列程序清单以验证除了10，30，70和99外的所有行都已被删去。

有些计算机使用DELETE（行号）—来删去在DELETE命令中指定的那一行号开始到末尾的所有的行。为了测试该特性，试用命令DELETE 30—。列程序清单以验证只剩下10语句行。

替代的形式

有些计算机（例如:DEC—10和APPLE）把DEL作为DELETE命令来使用。在DEC—10机上DEL的作用如上所述。

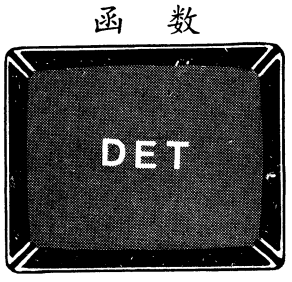
苹果机的DEL规定在DELETE使用连字符“—”的地方使用逗号。为了删去20到50行，打入DEL 20，50。删去单个一行需要DEL 30，30（或只打入该行号，然后按RETURN 键）。

如果你的计算机没有此功能

如果你的计算机没有DELETE命令，用单独打入每个行号接着按ENTER 键或RETURN 键的方法也能达到同样的目的。为了删去一个程序中的所有行，可以使用NEW命令或SCRATCH 命令

参阅

NEW，LIST，SCRATCH



DET是一个行列式函数，该函数带回与方阵（即一个行数和列数相同的二维数组）有关的单个数值。如果D=DET(A)，在这里A是一个2 × 2的数组，那么D=A（1，1）*A（2，2）-A（1，2）*A（2，1）。如果A是一个3 × 3的数组，那么行列式应该由六个乘积来组成。4 × 4的矩阵是由24个乘积的“和”以及差组成。

测试程序# 1

```
10 REM * DET * TEST PROGRAM
20 DIM A(3,3)
30 FOR I=1 TO 3
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70 NEXT I
80 D=DET(A)
90 PRINT "THE DETERMINANT OF ARRAY A IS ";D
100 DATA 1,1,1, 1,2,3, 1,4,9
30999 END
```

运行实例

```
THE DETERMINANT OF ARRAY A IS 2
```

有些解释程序仅当MAT INV 首先用在数组上时才计算DET函数。(详见 MAT INV)。一旦用MAT B = INV(A) 来求过矩阵的“逆”后，就可以求出矩阵A的行列式之值。如果DET = 0，那么矩阵A没有逆，数组B的值是无效的。

测试程序# 2

```
10 REM * DET WITH MAT INV * TEST PROGRAM
20 DIM A(3,3), B(3,3)
30 FOR I=1 TO 3
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70 NEXT I
80 MAT B=INV(A)
90 PRINT "THE DETERMINANT OF MATRIX A IS ";DET
100 DATA 1,1,1 1,2,3, 1,4,9
30999 END
```

运行实例

```
THE DETERMINANT OF MATRIX A IS 2
```

如果你的计算机没有此功能

如果你的计算机不允许使用任何一种形式的DET函数，那么可用下面子程序来代替。

```
30000 GOTO 30999
30940 REM * DET SUBROUTINE * INPUT N, A( , ), OUTPUT D
30942 REM ALSO USES I, J, K, L AND R INTERNALLY
30944 REM >> VALUES OF ARRAY A ARE ALTERED BY THIS
      ROUTINE <<
30946 D=1
30948 FOR K=2 TO N
30950  L=K-1
30952  IF A(L,L)<>0 THEN 30976
30954  FOR I=K TO N
30956   IF A(I,L)<>0 THEN 30964
30958  NEXT I
30960  D=0
30962  GOTO 30996
30964  FOR J=1 TO N
30966   R=A(J,L)
30968   A(J,L)=A(J,I)
30970   A(J,I)=R
30972   D=-D
30974  NEXT J
30976  FOR I=K TO N
30978   R=A(I,L)/A(L,L)
30980   FOR J=K TO N
30982    A(I,J)=A(I,J)-R*A(L,J)
30984   NEXT J
30986  NEXT I
30988 NEXT K
30990 FOR L=1 TO N
30992  D=D*A(L,L)
30994 NEXT L
30996 RETURN
```

为了使这个子程序与测试程序 1 一起使用，可作以下变化：

```
75 N=3
80 GOSUB 30940
```

其它用法

没有

参阅

MAT INV, DIM

语　句

把DIGITS 语句用在TSC 扩展BASIC中，以指定由一个PRINT 语句打印的数值的最大位数。例如：20 DIGITS 8，2可以用在一个程序中，在该程序中所有被打印的值都代表美元和美分。第一个数（8）指定被打印的数的总位数，第二个数（2）指定小数点右边的位数。第二个数必须不大于打印的数的总位数。

如果一个数的实际值大得超过了允许打印的位数，就把这个数打印成指数形式。该数的小数部分被四舍五入到所要求位数，最右边的数字被截去，不显示出来。

测试程序

```
10 REM DIGITS TEST PROGRAM
20 DIGITS 6,4
30 X = 0.1234567
40 PRINT X
50 PRINT "DIGITS PASSED THE TEST IF 0.1235 WAS PRINTED"
99 END
```

运行实例

```
 0.1235
DIGITS PASSED THE TEST IF 0.1235 WAS PRINTED
```

PERCOM SuperBASIC用DIGITS 语句指定要打印的小数点后面的位数。例如：20 DIGITS= 4 限制所有要打印的值都打印到小数点后面四位。

如果你的计算机没有此功能

如果DIGITS语句不适用于你的计算机，可在20行试用PRECISON 语句。

也可以删去20行和用下面语句来代替40行的方法来控制小数点后面的最大位数。

```
40 PRINT USING "**.****";X
```

如果PRINT USING 也不适用于你的计算机，请不要失望！可用下面的语句来代替：

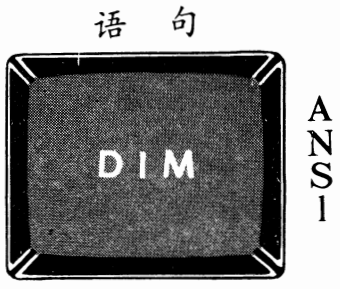
```
40 PRINT INT(X*10000 + .5)/10000
```

其它用法

没有

参阅

PRECISION，PRINT USING，IMAGE，FMT，INT



DIM语句用来规定数值或字符串型数组中所允许的元素个数。

为一个数组定维的方法是：把数组变量放在DIM语句中，后面在括号中的放入数组的大小。

例如：DIM A（20）允许数组变量 A 使用从A(0)开始到A（20）为止的21个数组元素。（有些计算机的数组元素从A（1）开始。另外很少的计算机(例如那些遵守ANSI BASIC的机器)使用BASE语句可以定义最低的数组元素0或1。详见BASE。

当执行DIM语句时，计算机把存放在每个指定的数组元素中的值置成零。

测试程序＃1

```
10 REM 'DIM' NUMERIC ARRAY TEST PROGRAM
20 DIM A(10)
30 PRINT "THESE NUMBERS ARE STORED IN AND PRINTED"
40 PRINT "FROM A SINGLE DIMENSION NUMERIC ARRAY."
50 FOR X=1 TO 10
60 A(X)=X
70 PRINT A(X);
80 NEXT X
99 END
```

运行实例

```
THESE NUMBERS ARE STORED IN AND PRINTED
FROM A SINGLE DIMENSION NUMERIC ARRAY.
 1  2  3  4  5  6  7  8  9  10
```

为了检查你的解释程序使用数组元素是否从0开始，使测试程序作如下变化：

```
50 FOR X=0 TO 10
```

如果你的解释程序接收数组元素A（0），那么运行结果应该打印从0到10的各个数。

大多数计算机不需要用DIM语句就允许每个数组使用从0(或1）到10的各个元素。从测试程序中删去20行以便对这个特性进行检查。

如果能运行，就对50行作以下变化：

```
50 FOR X=1 TO 15
```

然后再运行。有些计算机(例如：TRS—80 Level 1）不要求任何定维数组大小仅仅自动地受未使用的内存总数限制。TRS—80 Level 1只允许有一个命名为A(n)的数组。大多数计算机允许全部范围的字母变量，还有很多计算机允许数组具有字母/数字的数组名〔例如：A 3 (n)〕。

假设50行做了上述变化而没有成功，那么将20行改变为：

```
20 DIM A(15)
```

然后运行。

运行实例

```
THESE NUMBERS ARE STORED IN AND PRINTED
FROM A SINGLE DIMENSION NUMERIC ARRAY.
 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
```

下一个程序检测计算机为字符串数组定维的能力。有些计算机（例如：Hew lett—Packard ）要求所有的字符串（包括字符串数组）都要说明长度，没有DIM，就没有开辟字符串空间。

测试程序 = 2

```
10 REM 'DIM' STRING ARRAY TEST PROGRAM
20 DIM A$(4)
30 FOR X=1 TO 4
40 READ A$(X)
50 NEXT X
60 PRINT "THE 'DIM' STATEMENT PASSED THE ";
70 FOR X=1 TO 4
80 PRINT A$(X);
90 NEXT X
100 DATA T,E,S,T
999 END
```

运行实例

```
THE 'DIM' STATEMENT PASSED THE TEST
```

在一些计算机中用DIM语句来设定数值数组或字符串数组的最大元素,这些数组可以是两维的（或更多维的）。

例如：DIM A（20，25）指定数组A的第一维最大元素为20，而第二维最大元素为25。

大多数具有两维和三维数组能力的计算机自动地为每一维中保留十个元素的空间。很多较小的计算机(例如：Microsoft 解释程序)只为第一维和第二维保留空间。

测试程序＝ 3

```
10 REM 'DIM' TWO DIMENSION ARRAY TEST PROGRAM
20 DIM A(3,4)
30 PRINT "THESE NUMBERS ARE STORED IN AND PRINTED"
40 PRINT "FROM A TWO DIMENSION NUMERIC ARRAY."
50 FOR I=1 TO 3
60 FOR J=1 TO 4
70 A(I,J)=I
80 NEXT J
90 NEXT I
100 FOR I=1 TO 3
110 FOR J=1 TO 4
120 PRINT A(I,J),
130 NEXT J
140 PRINT
150 NEXT I
999 END
```

运行实例

```
THESE NUMBERS ARE STORED IN AND PRINTED
FROM A TWO DIMENSION NUMERIC ARRAY.
 1             1             1             1
 2             2             2             2
 3             3             3             3
```

测试程序＝ 4

这个程序测试计算机定义三维数值数组变量（即定维）的能力。

```
10 REM 'DIM' THREE DIMENSION ARRAY TEST PROGRAM
20 DIM A(3,4,2)
30 PRINT "THESE NUMBERS ARE STORED IN AND PRINTED"
40 PRINT "FROM A THREE DIMENSION NUMERIC ARRAY."
50 FOR K=1 TO 2
60 FOR I=1 TO 3
70 FOR J=1 TO 4
80 A(I,J,K)=I
90 NEXT J
100 NEXT I
110 NEXT K
120 FOR K=1 TO 2
130 FOR I=1 TO 3
140 FOR J=1 TO 4
150 PRINT A(I,J,K),
160 NEXT J
170 NEXT I
180 PRINT
190 NEXT K
999 END
```

运行实例

```
THESE NUMBERS ARE STORED IN AND PRINTED
FROM A THREE DIMENSION NUMERIC ARRAY.
 1             1             1             1
 2             2             2             2
 3             3             3             3

 1             1             1             1
 2             2             2             2
 3             3             3             3
```

如果你的计算机没有此功能

如果你的计算机不允许有多维数组，那么用单维数组进行模拟也不是困难的。为了使用一个两维的数组，例如：A（3，4），可把这个数组定成A（12），然后用A（（I—1）＊4＋J）来代替每一个二维数组元素A（I，J）。如果可以使用零下标，那么数组就可以定为A(19),即（3＋1）＊（4＋1）－1＝19。然后在用A(I，J)的地方用A（I＊4＋J）。

三维的数组也是类似的，为了定义数组A（3，4，2），可以使用DIMA（24）［或DIM A（59）—如果用零作下标］，然后用A（（（I－1）＊4＋（J－1）＊2＋K）［或用A（（I＊4＋J）＊2＋K），如果用零作下标代替A(I,J, K)］通常：

```
对M×N数组：        DIM A(M*N) 和用    A((I-1)*N+J)
   不用零下标：     DIM A((M+1)*(N+1)-1) 和用
      用零下标：    A(I*N+J)
对L×M×N数组：
   不用零下标：     DIM A(L*M*N) 和用    A(((I-1)
                   *M+(J-1))*N+K)
      用零下标：    DIM A((L+1)*(M+1)*(N+1)-1)
                   和用    A((I*M+J)*N+K)
```

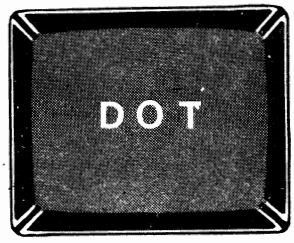
其它用法

没有

参阅

CLEAR，MAT INPUT，MAT PRINT，MAT READ

函　数



Sweden的ABC—80计算机用DOT函数指出屏幕上的一个特定图形块是否“打开”。该图形块是由DOT函数后面的L, C坐标确定的，这里L确定行(在图形方式是从0到71)，而C确定列（在图形方式是从2到79）。

例如：IF DOT（9，15）THEN 950,如果这个位于左上角的第10行第16列的块是“开”的，则转移到950行。

为了“打开”图形块参阅SETDOT。

测试程序

```
10 REM 'DOT' TEST PROGRAM
20 PRINT CHR$ (12)     'CLEARS SCREEN
30 R=5
40 PRINT CUR(R,0);CHR$ (151);  'SETS GRAPHICS MODE
50 FOR C=1 TO 35
60 SETDOT R,C
70 NEXT C
80 PRINT
90 IF DOT (5,12) THEN 120
100 PRINT "THE BLOCK IS OFF"
110 GOTO 130
120 PRINT "THE BLOCK IS ON"
130 FOR T=1 TO 2000 : NEXT T
140 PRINT CHR$ (12)
999 END
```

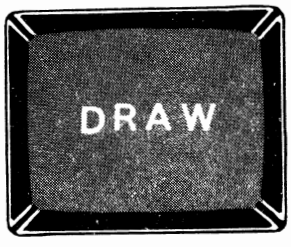
运行实例

```
THE BLOCK IS ON
```

参　阅

POINT, SETDOT, CLRDOT, $

语　句



DRAWTO

几种计算机（例如：APPLE Ⅱ机）用DRAW语句来画出一个从X,Y位置开始的预先确定的形状（编号为N）。

例如：DRAW　N　AT X, Y

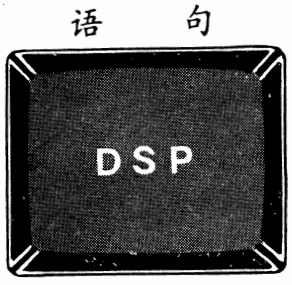
另一个版本,DRAW　X,Y用在SINclair ZX 80机上，从现行位置(H,K)开始到新位置(X+H,Y+K)划一条线。

Atari计算机使用画线语句DRAW TO　X,Y,从现行位置开始到位置（X,Y）止画一条线。

有些计算机用与那些使用DRAW的计算机相同的方法来使用PLOT。详见PLOT。

参 阅

PLOT ，XDRAW



DSP语句用在APPLE Ⅱ的BASIC中作为一个分析手段，它显示一个指定的变量和它的值，每当给该变量赋值时就显示这个变量的值。与该变量有关的行号（前面有一个＃号）也显示出来。在一个程序中允许有多个DSP语句。例如：

```
10 DSP X
20 DSP Y
```

命令计算机每当变量X和Y被赋值或重新赋值时显示（打印）变量X和Y和它们的值，以及行号。

测试程序

```
10 REM 'DSP' TEST PROGRAM
20 DSP A
30 DSP B
40 A=5
50 B=10
60 C=A*B
70 A=A+C
80 PRINT "THE DSP STATEMENT PASSED THE TEST"
99 END
```

运行实例

```
#40 A=5
#50 B=10
#70 A=55
THE DSP STATEMENT PASSED THE TEST
```

如果你的计算机没有此功能

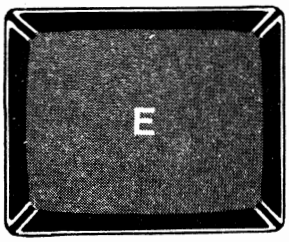
在变量值变化的每一点加入一个临时的测试语句行，就可以模拟这个非常方便的查找故障的特性。例如：

```
10 REM DSP SIMULATION
40 A=5
41 PRINT "*40 A=";A
50 B=10
51 PRINT "*50 B=";B
60 C=A*B
70 A=A+C
71 PRINT "*70 A=";A
80 PRINT "END OF THE DSP SIMULATION"
99 END
```

参 阅

TRON, TRACE

操作符



用操作E来表示“指数记数法”或“标准的科学记数法”。

例如：1.23E +12意味着123后面有10个零。

以双精度表示的数书写成指数符号的形式时，用字母“D”来表示。

例如：1.23456789D+20

测试程序

```
10 REM 'E' SINGLE PRECISION EXPONENT TEST PROGRAM
20 A=123456789
30 PRINT "EXPONENTIAL NOTATION 'E' PASSED THE TEST IF"
40 PRINT A;"CONTAINS THE LETTER 'E'"
99 END
```

运行实例

```
EXPONENTIAL NOTATION 'E' PASSED THE TEST IF
 1.23457E+08 CONTAINS THE LETTER 'E'
```

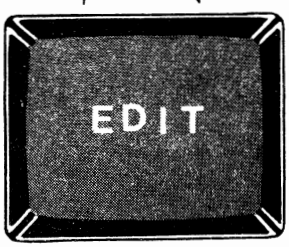
其它用法

在所有的计算机中，字母“E”象字母表中所有其它的字母一样，可以用来表示数值变量。

参 阅

D, !, #, DEFSNG, DEFDBL, CSNG, CDBL

命 令



EDIT是一些计算机（例如那些使用Microsoft BASIC的机器）使用的专门命令，该命令允许编辑由EDIT命令确定的程序行。在一些计算机中，它类似于RUN和LIST命令，如果它的后面有行号，则隐含第一个程序行。

测试程序

```
10 REM 'EDIT' TEST PROGRAM
20 PRINT "CAN THIS PROGRAM BE
   MODIFIED"
30 PRINT "BY THE EDIT COMMAND?"
99 END
```

在装入这个程序之后，打入EDIT 20来决定计算机是否有EDIT功能。计算机应该打印出数20，后面可能跟着一个光标。这表示计算机处于EDIT方式，并准备修改20行。

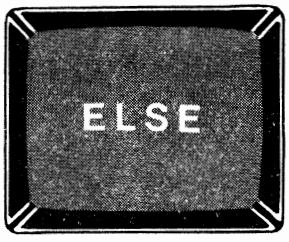
EDIT命令可以调用你的编辑程序，但是你应该检查计算机手册以便了解如何实现编辑功能和又如何返回BASIC状态。有时它象按“回车”那样容易，但有时（特别是在大的、多种语言的、分时的计算机）它有一整套命令使进入和脱离“编辑程序”。

其它用法

有许多版本的文本、字符和行的编辑程序，每一种都使用它自己的“语言”，而且不是BASIC语言。因此这本BASIC大全将不讨论编辑语言。

ELSE

语　　句



当IF—THEN语句的条件得不到满足时，用ELSE语句来执行另一个替代的语句。例如：IF X = 3 THEN 100 ELSE STOP命令计算机：若X等于3就转移到100行，如果X不等于3就暂停。

测试程序

```
10 REM 'ELSE' TEST PROGRAM
20 X=1
30 IF X < 5 THEN 60 ELSE GOTO 90
40 PRINT "ELSE FAILED THE TEST"
50 GOTO 99
60 PRINT X;
70 X=X+1
80 GOTO 30
90 PRINT "'ELSE' PASSED THE TEST"
99 END
```

运行实例

```
1  2  3  4  'ELSE' PASSED THE TEST
```

如果你的计算机没有此功能

如果你的计算机没有 ELSE语句，在测试程序中用改变30行的方法来模拟 ELSE 语句。把30行变成下面形式：

```
30 IF X < 5 THEN 60
```

再加上下面的新行：

```
35  GOTO  90
```

参 阅

IF-THEN，GOTO

语 句

ANSI

E.

END语句用来终止程序的执行。很多计算机要求把 END语句放在该程序中的最高的行号处。而其它一些计算机允许 END语句放在任何地方。

对很多计算机（大部分是微机）而言，END语句是可选的（即可以不用）。

测试程序

```
10 REM 'END' TEST PROGRAM
20 PRINT "THE FIRST END STATEMENT FOLLOWS"
30 END
40 PRINT "THE SECOND END STATEMENT FOLLOWS"
99 END
```

运行实例

```
THE FIRST END STATEMENT FOLLOWS
```

如果你的计算机不能通过这个测试程序并且不允许在30行出现 END语句，那么删去30行然后再运行该程序。

接着删去99行以观察在你的计算机中END语句是否是可选的。

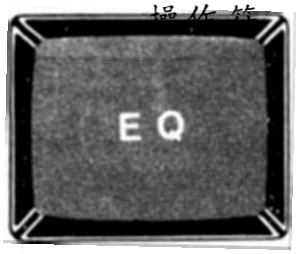
替代的形式

TRS—80 Level Ⅰ和使用Tiny BASIC的计算机用E.作为END的缩写。

参　阅

STOP（在同一个程序中使用END语句和STOP语句有时会遇到一些问题）。

在几种计算机中（例如T I 990），EQ操作符作为一个关系操作符使用时，可作为等号的一个可选字（见30行）。不能用它来对变量赋值。这就是为什么20行要使用一个等号(而不能用EQ)的原因。详见等号=。



测试程序

```
10 REM 'EQ (EQUAL)' TEST PROGRAM
20 A=10
30 IF A EQ 10 THEN 60
40 PRINT "THE EQ OPERATOR FAILED THE TEST"
50 GOTO 99
60 PRINT "THE EQ OPERATOR PASSED THE TEST"
99 END
```

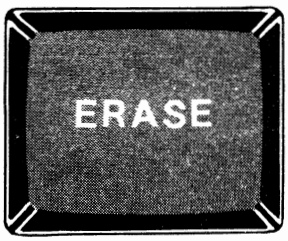
运行实例

```
THE EQ OPERATOR PASSED THE TEST
```

参　阅

=, <>, IF-THEN, GE, GT, LE, LT, NE, <, >, =<, =>

命 令
语 句



ERASE 是一个命令，用它来删除内存中的一个程序。ERASE 等效于其它一些计算机所使用的NEW或SCRATCH。为了检测你的计算机中的ERASE命令，你可以往计算机里输入一个短程序，例如：

```
10  REM  THIS A  SHORT  PROGRAM
99  END
```

打入LIST来检查已有的程序。

打入ERASE，然后再打入LIST。如果ERASE有效，那么程序已被删掉了。

有些解释程序（例如，BASIC—80）则把ERASE当作一个语句来使用，用它可以把一个数组从一个程序中除去，这样，就释放了该数组所使用的存储空间。由于在一个程序内使用ERASE，在该程序运行时，就可以重新给这个数组定维，而这是为了大多数解释程序和编译程序所禁止的过程。

某些计算机不用ERASE这个语句也可以重新给数组定维，但是大多数计算机在同一个数组名出现在两个DIM语句中时，会给出一个错误信息。

测试程序

```
10 REM 'ERASE' TEST PROGRAM
20 DIM A(15)
30 FOR I=1 TO 15
40 A(I) =I
50 NEXT I
60 ERASE A
70 PRINT "ERASE PASSED THE TEST IF 0 =";A(1)
80 DIM A(5,5)
90 A(5,5)=2
100 PRINT "ERASE PASSED";A(5,5);" TESTS."
999 END
```

运行实例

```
ERASE PASSED THE TEST IF 0 = 0
ERASE PASSED 2 TESTS.
```

参阅

NEW，SCRATCH，DIM，LIST

函 数

ERRL

ERL 函数与ON—ERROR语句一起用来识别已经出现错误的最后的行号。

ERL函数的初始值为行号65535（最大的两字节值）。当出现错误时，ERL函数转到出现错误的那个行号。每当错误发生在不同的行号时，包含在ERL函数中的那个行号就发生变化。

由于把ERL函数用于“错误捕捉”子程序，因而识别错误行和采取适当的行动是可能的。

测试程序

```
10 REM 'ERL' TEST PROGRAM
20 ON ERROR GOTO 100
30 PRINT "ENTER THE NUMBER 10, 20, THEN 30";
40 INPUT N
50 A=10/(N-10)
60 A=10/(N-20)
70 A=10/(N-30)
80 PRINT "THE NUMBER ";N;"DID NOT CAUSE AN ERROR"
90 GOTO 30
100 PRINT "AN ERROR HAS JUST OCCURRED IN LINE"; ERL
110 RESUME 30
999 END
```

运行实例

```
ENTER THE NUMBER 10, 20, THEN 30? 10
AN ERROR HAS JUST OCCURRED IN LINE 50
ENTER THE NUMBER 10, 20, THEN 30? 20
AN ERROR HAS JUST OCCURRED IN LINE 60
ENTER THE NUMBER 10, 20, THEN 30? 30
AN ERROR HAS JUST OCCURRED IN LINE 70
ENTER THE NUMBER 10, 20, THEN 30?
```

替代的形式

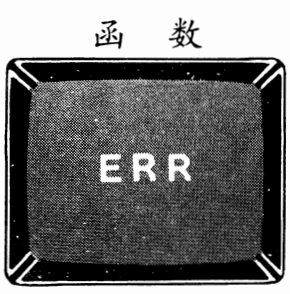
Hewlett—Packard的35和45以及85计算机使用ERRL。

其它用法

没有。

参阅

ERROR, ON-ERROR-GOTO, RESUME



ERRN

在某些计算机（例如，那些具有Microsoft BASIC的机器）中用ERR函数以标识发生在程序中的最后一个错误的的代码。每当出现不同的错误时，包含在ERR函数中的错误码就发生变化。由于把ERR函数用在“错误捕捉”子程序中，因而识别发生的错误类型和采取适当的行动是可能的。可参阅计算机手册关于特殊错误码的表。

测试程序

```
10 REM 'ERR' TEST PROGRAM
20 DIM A(5)
30 CLEAR
40 ON ERROR GOTO 100
50 PRINT "ENTER A SAMPLE NUMBER";
60 INPUT N
70 A(N)=10/N
80 PRINT "THE NUMBER";N;"DID NOT CAUSE AN ERROR"
90 GOTO 50
100 IF ERR = 9 THEN 130
110 IF ERR = 11 THEN 160
120 GOTO 180
130 PRINT "THE NUMBER";N;"IS TOO LARGE"
140 PRINT "USE A NUMBER BETWEEN 1 AND 5"
150 RESUME 30
160 PRINT "THE SMALLEST NUMBER ALLOWED IS 1"
170 RESUME 30
180 PRINT "THE NUMBER";N;"CAUSED AN ERROR CODE OF";ERR
999 END
```

运行实例（典型值）

```
ENTER A SAMPLE NUMBER? 12
THE NUMBER 12 IS TOO LARGE
USE A NUMBER BETWEEN 1 AND 5
ENTER A SAMPLE NUMBER? 0
THE SMALLEST NUMBER ALLOWED IS 1
THE NUMBER 1 DID NOT CAUSE AN ERROR
ENTER A SAMPLE NUMBER?
```

替代的形式

Hewlett—Packard的35、45以及85计算机使用ERRN。

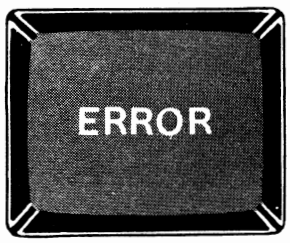
其它用法

TRS-80 Level Ⅱ BASIC 把一个值存放在ERR函数中，该值不等于真正的错误码。为了把存放在ERR函数中的那个值转换成真正的错误码，则要把ERR的值除以2后再加上1。

例如：PRINT ERR/2 + 1。

参阅 命 令 语 句

ERL, ON-ERROR, RESUME, DIM, CLEAR



ERROR ## 用来有意地使计算机报告一个错误(ERROR)。错误性质是由ERROR语句中的一个错误码来确定的。ERROR语句一般用在很多程序中来执行“错误捕捉子程序”或打印一个指定的错误信息。

测试程序=1（对一个Microsoft解释程序而言）

```
10 INPUT N
20 IF N > 32000 THEN ERROR 7
99 END
```

当一个大于32000的值赋给变量N时，20行中的IF—THEN语句的条件能得到满足，那么计算机就产生错误信息。

OM ERROR IN 20

（OM是Out of Memory的缩写，上面信息表示在20行超出内存），即使计算机实际上并没有超出内存。

不能把变量作为ERROR码使用。每个代码必须由一个真正的整数的错误码来确定。如果这个确定的错误码不为计算机的解释程序承认的话，那么大多数计算机打印错误信息“UNPRINTABLE ERROR”（不能打印出来的错误）。

也可以把ERROR作为一个命令输入到计算机中以测试特殊的错误码。参阅你的计算机手册中关于错误信息的表。

测试程序#2

```
10 REM 'ERROR' TEST PROGRAM
20 PRINT "ERROR PASSED THE TEST IF ERROR MESSAGE 'OS' OR"
30 PRINT "'OUT OF STRING SPACE' IS PRINTED."
40 ERROR 14
99 END
```

运行实例（典型值）

```
ERROR PASSED THE TEST IF ERROR MESSAGE 'OS' OR
'OUT OF STRING SPACE' IS PRINTED.
?OS ERROR IN 40
```

参阅

ON-ERROR-GOTO, RESUME, ERR, ERL

函　数

某些计算机（例如，Digital Group MAXI—BASIC，北极星BASIC以及Processor Technology 8K BASIC）用EXAM(n)函数来读出计算机内存中指定地址的内容。

例如：X=EXAM（200）把存放在内存地址200中的值赋给变量X。

EXAM函数把内存地址的内容告诉我们，该地址是0到255之间的一个十进制数（这个值的范围可以保存在一个8位的内存字节中）。EXAM函数可以与FILL语句一起用来读出FILL语句已经存放在存储器中的内容。(某些计算机使用POKE或STUFF。）当然，能被检验（EXAM）的最高的编号地址取决于该计算机的内存容量。

在执行下面这个测试程序之前，检查你的计算机手册，以确定从18368到18380的内存地址是否是“自由”的（即未用的）。这样就可以避免把数据装入（FILL）到留给其它的计算机操作用的内存地址中。如果从18368到18380的地址不是“自由”的，选择一组12个相邻的内存地址，然后相应地改变测试程序中的20行和60行。

测试程序

```
10 REM 'EXAM' TEST PROGRAM
20 FOR X=18368 TO 18380
30 READ Y
40 FILL X,Y
50 NEXT X
60 FOR X=18368 TO 18380
70 Y=EXAM(X)
80 PRINT CHR$(Y);
90 NEXT X
100 DATA 84,69,83,84,128,67,79,77,80,76,69,84,69
999 END
```

运行实例

```
TEST COMPLETE
```

参阅

FILL，POKE，PEEK，USR，SYSTEM，STUFF，FETCH

语　句

EXCHANGE是适用于某些BASIC（例如：TDLBASIC）的语句，该语句交换两个变量的值或数组元素的值。例如，EXCHANGE A, B 结果是：原来A 的值存放在B 中，而原来B的值存放在A中。EXCHANGE语句对于以递增次序或递减次序排列一个数组的值是非常有用的。

测试程序

```
10 REM 'EXCHANGE' TEST PROGRAM
20 PRINT "ENTER TWO VALUES (SEPARATED BY COMMAS)"
30 INPUT A,B
40 IF A<=B THEN 60
50 EXCHANGE A,B
60 PRINT A;" IS LESS THAN OR EQUAL TO ";B
70 GOTO 20
99 END
```

运行实例

```
ENTER TWO VALUES (SEPARATED BY COMMAS)
? 3,7
3 IS LESS THAN OR EQUAL TO 7
ENTER TWO VALUES (SEPARATED BY COMMAS)
? 9,1
1 IS LESS THAN OR EQUAL TO 9
ENTER TWO VALUES (SEPARATED BY COMMAS)
?
```

MAXBASIC 使用一个双等号（= =）来交换同一类型的两个变量的内容。例如：50A = B 等效于测试程序中的50行。

如果你的计算机没有此功能

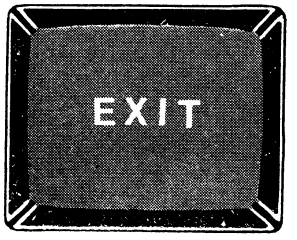
如果你的计算机不能使用EXCHANGE语句的话，你可在50行试用SWAP。如果EXCHANGE和SWAP没有一个适用于你的计算机，那么用下列几行来代替50行就能交换二个值。：

```
48 T=A
50 A=B
52 B=T
```

参阅

SWAP

语　句



EXIT是由某些BASIC（例如，北极星BASIC）使用的一个语句，在完成指定的循环次数之前用EXIT语句可以从FOR——NEXT循环中退出。

EXIT语句把程序控制转移到指定的行号，然后脱离FOR——NEXT循环。脱离循环时的循环计数器的值（即循环变量的值）在程序的其它部分继续可以使用。

测试程序

```
10 REM 'EXIT' TEST PROGRAM
20 PRINT "ENTER A WORD - TYPE 'DONE' TO QUIT"
30 FOR I=1 TO 500
40 INPUT A$
50 IF A$="DONE" THEN EXIT 100
60 PRINT "ANOTHER";
70 NEXT I
80 PRINT "'EXIT' FAILED THE TEST."
90 GOTO 999
100 PRINT "'EXIT' PASSED THE TEST. 'DONE' WAS WORD
    NUMBER", I
999 END
```

运行实例（典型值）

```
ENTER A WORD - TYPE 'DONE' TO QUIT
?START
ANOTHER ?CHECK
ANOTHER ?EXIT
ANOTHER ?HERE
ANOTHER ?DONE
'EXIT' PASSED THE TEST. 'DONE' WAS WORD NUMBER 5
```

如果你的计算机没有此功能

在大多数场合下，用GOTO语句可代替EXIT语句。在一些计算机上，如果用这种方法脱离循环后，接着又执行另一个循环FOR I =…，计算机不能判定哪一个循环是有效的。它要求从“正常出口”脱离循环。在这些计算机中，可用下列语句代替50行。

```
50 IF A$<>"DONE" THEN 60
52 J = I    'J STORES THE CURRENT VALUE OF THE LOOP
   COUNTER
54 I = 999 'SET I TO A VALUE ABOVE THE LIMIT OF THE
   LOOP
56 GOTO 70
```

再加上行75和95

```
75 IF I=999 THEN
95 I=J
```

在转到程序其它部分之前，使循环“正常地”终止。

函　数

参阅

FOR，NEXT，GOTO

EXP（n）函数计算自然对数的底数e （2.718282…）的n次方。

EXP刚好和LOG函数相反。

例如，A = EXP （3）与A=2.718282＊2.718282＊2.718282的结果相同。

可以把（n）的值写成一个数值或一个数值变量。

测试程序

```
10 REM 'EXP' TEST PROGRAM
20 N=4.60517
30 E=EXP(N)
40 PRINT "IF THE NATURAL EXPONENTIAL OF";N;"IS";E
50 PRINT "THEN THE EXP FUNCTION PASSED THE TEST."
30999 END
```

运行实例

```
IF THE NATURAL EXPONENTIAL OF 4.60517 IS 100
THEN THE EXP FUNCTION PASSED THE TEST.
```

如果你的计算机没有此功能

如果你的解释程序不接受EXP函数的话，可用下面的子程序来代替EXP。

```
30000 GOTO 30999
30200 REM * EXPONENTIAL SUBROUTINE * INPUT X, OUTPUT E
30202 REM ALSO USES A, B AND L INTERNALLY
30204 L=INT (1.4427*X)+1
30206 IF ABS(L)<127 THEN 30218
30208 IF X<=0 THEN 30214
30210 PRINTX; "IS OUT OF RANGE"
30212 STOP
30214 E=0
30216 RETURN
30218 E=.693147*L-X
30220 B=X
30222 A=1.32988E-3-1.41316E-4*E
30224 A=((A*E-8.30136E-3)*E+4.16574E-2)*E
30226 E=(((A-.166665)*E+.5)*E-1)*E+1
30228 A=2
30230 IF L>0 THEN 30238
30232 A=.5
30234 L=-L
30236 IF L=0 THEN 30244
```

```
30238 FOR X=1 TO L
30240 E=A*E
30242 NEXT X
30244 X=B
30246 RETURN
```

为了把这个子程序与测试程序一起使用，把测试程序做如下改变：

```
 25 X=N
 30 GOSUB 30204
```

参阅

LOG，LOG10，CLG

函　数

FETCH（n）函数在Digital Group Opusl和Opus 2 BASIC中用来读出计算机内存中的地址的内容。

例如，X= FETCH（3000）把存放在内存地址3000中的十进制数赋给变量X。

该十进制的数值可以是0到255（该值的范围可以被存放在一个8位的字节中）之间的一个数。当然，能取出的最高的编码地址取决于这个计算机的内存容量。

FETCH函数可以与STUFF语句一起用来检查STUFF语句已存放在内存中的内容（一些计算机用POKE或FILL来代替STUFF）。

在执行这个测试程序之前，要检查你的计算机手册以确定从18368到18377的内存地址是否为“自由”空间。这样就可以避免把数据装进（STUFF）留给专门目的使用的内存单元中。如果从18368到18377的地址在你的计算机中不是留作“自由”空间的，那么选择10个自由的相邻的内存地址，然后相应地改变测试程序中的30和70行。

测试程序

```
10 REM 'FETCH' TEST PROGRAM
20 Y=1
30 FOR X=18368 TO 18377
40 STUFF X,Y
50 Y=Y+1
60 NEXT X
70 FOR X=18368 TO 18377
80 Y=FETCH(X)
90 PRINT Y;
100 NEXT X
110 PRINT
120 PRINT "'FETCH' PASSED THE TEST IF #1 THRU #10
    ARE PRINTED"
999 END
```

运行实例

```
 1  2  3  4  5  6  7  8  9  10
'FETCH' PASSED THE TEST IF #1 THRU #10 ARE PRINTED
```

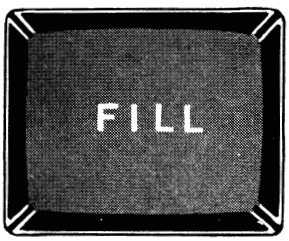
其它用法

没有

参阅

STUFF，POKE，PEEK，FILL，USR，SYSTEM，EXAM

语　句



FILL语句由一些解释程序（例如，北极星的BASIC和Digital Group MAXI ——BASIC）用来将一个值赋给计算机内存中的一个指定的字节，该值在0到255（即8位的最大值）之间的一个整数值。

例如FILL 3000, 15把十进制的数15“装入”内存地址3000中。

EXAM函数可以用来检查FILL语句已经“装入”到内存中的内容（某些计算机使用PEEK或FETCH来代替EXAM）。

计算机用FILL“装入”内容时不应清除掉原来为其它目的用的内存单元的内容，这部分（自由的）空间的总数以及它们的内存地址是不断在变化的。在运行下面测试程序之前，查你的计算机手册，以确定18368到18380这段地址是否为“危险的”单元（即已有专用）。

测试程序

```
10 REM 'FILL' TEST PROGRAM
20 FOR X=18368 TO 18380
30 READ Y
40 FILL X,Y
50 NEXT X
60 FOR X=18368 TO 18380
70 Y=EXAM(X)
80 PRINT CHR$(Y);
90 NEXT X
100 DATA 84,69,83,84,128,67,79,77,80,76,69,84,69
999 END
```

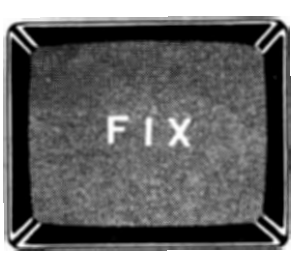
运行实例

```
TEST COMPLETE
```

参阅

POKE, STUFF, EXAM, PEEK, FETCH, USR, SYSTEM

函　数



FIX函数用来把小数点右边的数截去。它的操作类似于INT函数，但FIX函数对负数“降低”数的值。

例如：

```
10 PRINT FIX(3.6)
20 PRINT FIX(-3.6)
```

打印数3和－3．而：

```
10 PRINT INT(3.6)
20 PRINT INT(-3.6)
```

打印数 3 和 - 4。

FIX函数可以处理计算机的解释程序允许范围内的任意一个数，不管是大数还是小数。

测试程序

```
10 REM 'FIX' TEST PROGRAM
20 N=-12.3456
30 A=FIX(N)
40 PRINT "FIX PASSED THE TEST IF ";N;"IS CHANGED TO ";A
99 END
```

运行实例

```
FIX PASSED THE TEST IF -12.3456 IS CHANGED TO -12
```

如果你的计算机没有此功能

如果你的解释程序没有FIX函数功能，但是具有ABS、INT以及SGN函数，那么测试程序中的30行可以用下面的语句行来代替：

```
30 A=SGN(N)*INT(ABS(N))
```

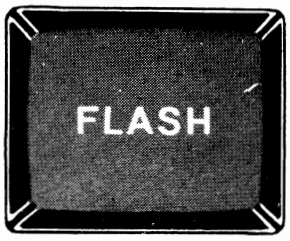
其它用法

没有

参阅

INT，ABS，SGN

语　句
命　令



APPLEⅡ机把FLASH作为命令或语句使用，FLASH可把屏幕置于FLASH（闪动）状态。在FLASH状态时，计算机的所有输出都一闪一闪地显示，即一会儿在黑色的背景上显示白色字符，过一会又在白色的背景上显示黑色字符，这两种显示是交替进行的。

为了把显示恢复成不闪动的状态（即正常状态），可打入NORMAL。

测试程序

```
10 REM 'FLASH' TEST PROGRAM
20 FLASH
30 PRINT "THIS IS A FLASHY MESSAGE."
99 END
```

为了运行这个程序，要清除屏幕，然后打入RUN。

运行实例

```
THIS IS A FLASHY MESSAGE. (屏幕应当是闪动的)
```

其它用法

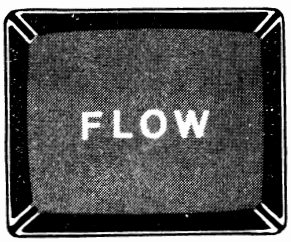
没有

参阅

NORMAL, INVERSE

语 句

命 令



FLOW是Microplis BASIC使用的命令，该命令可以把计算机执行的每一个程序行的行号显示在屏幕上。FLOW命令可作为程序故障查找的辅助手段，使用NOFLOW语句可以关闭FLOW。

在一个程序中可以把FLOW和NOFLOW结合在一起使用,以便仅跟踪程序中所要求跟踪的那一部分。

测试程序

```
10 REM 'FLOW' TEST PROGRAM
20 PRINT "'FLOW' TRACES EACH LINE"
30 FLOW
40 GOTO 90
50 PRINT "UNTIL TURNED OFF BY"
60 NOFLOW
70 PRINT "THE 'NOFLOW' STATEMENT"
80 GOTO 110
90 PRINT "THAT FOLLOWS THE 'FLOW' STATEMENT"
100 GOTO 50
110 PRINT "AS ILLUSTRATED BY THIS LINE"
999 END
```

运行实例

```
'FLOW  TRACES EACH LINE
<40> <90> THAT FOLLOWS THE 'FLOW' STATEMENT
<100> <50> UNTIL TURNED OFF BY
<60> THE 'NOFLOW' STATEMENT
AS ILLUSTRATED BY THIS LINE
```

其它用法

没有

参阅

NOFLOW, TRACE, TRACE ON, TRON

函 数
语 句

FMT函数用在某些BASIC中（例如，Micropolis BASIC）用它可以把PRINT语句的输出格式化。FMT和PRINT USING有些类似，也和在FORTARN语言中使用FORMAT格式输出类似。

FMT以字符串表达式的形式表示要打印的数值的格式。字符串表达式必须用引号括起来。

下面是字符串表达式的有效字符：

9　每个9可以确定输出字段中一个数字的位置。如果要打印一个5位数字的数，在字符串表达式中可以使用5个9（“9 9 9 9 9”）。如果一个数的位数少于FMT所指定的位数的话，那么其左边位置都打印成零。

Z　把Z用来代替上面的9。如果用Z，前导零都不打印出来（用空格来代替零）。

V　V可以定位小数点。然而不把小数点打印出来，它不单独占据一个打印位置。

$　如果把$用到Z或9的左边，那么在一个数的前面打印一个$。两个或多个$可以给出一个“浮动”的$，也就是把这个$打印在该数的第一位数最左边的位置上。

*　只要前面出现一个前导零，就打印一个*。如把*和$一起使用，那么$打印在*的左边（刚好与PRINT USING相反）。

.（句号）如果一个句号出现在字符串表达式中，就打印一个句号。

,（逗号）逗号可以用来表明在打印值中包含逗号。在第一位数字前面的逗号不打印出来。

当使用不包含上述任一个字符的字符串表达式时数值被打印成“文本”，即原样照印。数值可以用FMT函数使之格式化。

如果指定的字段太小，不能放下要打印的数值，那么在该字段的每一个位置上，都打印出问号（？），包括“文本”的每一位置。

FMT删去格式表达式所有不能容纳下的后面几位小数。

测试程序

```
10 REM 'FMT' TEST PROGRAM
20 PRINT FMT(2401,"999999")
99 END
```

运行实例

```
002401
```

其它一些例子：

语句	结果
`20 PRINT FMT(-2401,"999999")`	`0-2401`
`20 PRINT FMT(2401,"ZZZZ99")`	`2401`
`20 PRINT FMT(123.456,"ZZZZV99")`	`12345`
`20 PRINT FMT(123.456,"ZZZZV.99")`	`123.45`
`20 PRINT FMT(12345,"Z,ZZZ,ZZZ")`	`12,345`
`20 PRINT FMT(1000000,"THIS BOOK IS WORTH $$$,$$$,$$$")`	`THIS BOOK IS WORTH $1,000,000`
`20 PRINT FMT(123,"*****99")`	`****123`

其它用法

FMT作为一个语句（在Honeywell）可用来把一个打印行格式化。它类似于IMAGE语句。使用这个格式化语句的PRINT语句必须包括FMT的行号，该行号应放在PRINT后面的第一项。

例如：

```
180 PRINT, 190, X, T
190 FMT F6.2, X6, E12.5
```

F 6.2, X6和E 12.5告诉计算机如何打印以及在什么地方打印X和T的值。

F　表示打印的值是带小数点的。第一个数（6）表示字段的宽度（即被打印字符的最多的个数，包括符号，如果有的话，还有小数点），而第二个（2）表示小数点右面的位数。

E　指数形式的符号（例如：1.385E +05）。第一个数也表示字段的宽度，可是第二个数表示被打印数的有效位数（包括整数和小数）。

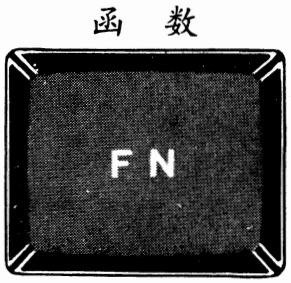
Xn　表示由数n 指明的空格数。在我们这个例子里，在打印X和T值之间有六个空格。

如果你的计算机没有此功能

如果你的计算机不含有任何形式的FMT函数的话，你可以试用PRINT USING。PRINT USING中的“####.##”与FMT中的“ZZZZV.99”等效。E12.5可以改写成“##.#####”等等。

参阅

PRINT USING, IMAGE



FN是一个函数。它允许“用户定义”的函数作为一个内部函数那样来使用。这个由用户定义的函数是由跟着FN的一个字母来命名的，它的后面是括号中的一个或 多个值。例如：FNA(X, N)。

DEF语句定义一个过程，在以后用FN函数时就要执行这个过程。

例如：

```
10 DEF FNA(X)=1/X
20 PRINT FNA(N)
```

在这个例子里，FN函数命名为“A”(FNA)，在10行中把FN定义为函数1 /X。在这里用FNA来计算任一个数值表达式的倒数（当然，除了0 值以外）。

每执行一次FNA跟着FNA后面的数值变量(N)就代替DEF语句中的“虚变量”(在这个例子中的X)。任意一个有效数值变量或表达式都可以代替N。

测试程序

```
10 REM 'FN' TEST PROGRAM
20 DEF FNX(A) = (A-32)*.5555555
30 PRINT "ENTER A TEMPERATURE IN FAHRENHEIT DEGREES";
40 INPUT F
50 C = FNX(F)
60 PRINT F;"DEGREES FAHRENHEIT =";C;"DEGREES CELSIUS."
99 END
```

运行实例（输入70）

```
ENTER A TEMPERATURE IN FAHRENHEIT DEGREES? 70
 70 DEGREES FAHRENHEIT = 21.1111 DEGREES CELSIUS.
```

其它用法

某些BASIC（例如：DEC BASIC—PLUS）可以把FN定义为能用字符串的函数。

例如：

```
DEF FNP$(A$,N) = RIGHT$(STRING$(N," ")+A$,N)
```

这个函数（FNP$）的作用是：对一个长度不足N的字符串A$的前面加上若干空格来填充字符串，以使字符串具有长度N。

如果你的计算机没有此功能

如果你的计算机不允许你用这种方法来定义函数的话，你就得在每一个需要用该函数的程序行处具体地写出你所要求的函数。

参阅

DEF，FNEND

语 句

FNEND语句用在一些计算机中,这些计算机允许在整个程序中的不同处DEF（定义)和重新定义一个函数。FNEND语句终止该函数的定义过程。

多行的DEF语句(即定义和再定义的过程)必须用FNEND语句来结束,在执行FNEND语句之前不能转出或进入这组DEF语句。

测试程序

```
10 REM 'FNEND' TEST PROGRAM
20 PRINT "ENTER A VALUE FOR X THAT IS
   GREATER OR LESS THAN 10";
30 INPUT X
40 DEF FNA(X)
50 FNA=X*2
60 IF X<10 THEN 80
70 FNA=X/2
80 FNEND
90 PRINT "THE NEW VALUE FOR X IS";FNA(X)
999 END
```

运行实例（输入6）

```
ENTER A VALUE FOR X THAT IS GREATER OR LESS THAN 10? 6
THE NEW VALUE FOR X IS 12
```

其它用法

没有

参阅

DEF，FN

语 句

ANSI

FOR语句是一个FOR—TO—NEXT语句的一部分，FOR语句用来把数赋给一个数值变量，这个变量是由FOR—TO语句指定变化范围的值。

紧跟着FOR的变量开始时取一初值（等号后第一个数），每当执行到与FOR的对应的NEXT语句时它就递增1。当它超过跟着TO的那个数时，程序就接着执行跟着相应的NEXT语句后面的那一行。

测试程序

```
10 REM 'FOR' TEST PROGRAM
20 FOR X=1 TO 5
30 PRINT X;
40 NEXT X
50 PRINT "THE 'FOR' STATEMENT PASSED THE TEST"
99 END
```

运行实例

```
 1  2  3  4  5 THE 'FOR' STATEMENT PASSED THE TEST
```

有些计算机使用STEP来指定FOR—TO—NEXT语句的增量，这个增量可以不等于1，也可以是负数即递减（以渐减的顺序来改变增量的值）。

更详细的资料可参阅STEP。

替代的形式

有些计算机（例如，ACORN和TRS—80 LevelI)也可以用F.来代替FOR语句。

如果你的计算机没有此功能

如果你的计算机不接受FOR语句的话，你可以用一个计数循环来代替FOR。20和40行可用下面的行来代替：

```
20 X=1
35 X=X+1
40 IF X<=5 THEN 30
```

其它用法

有些计算机（例如，DEC BASIC—PLUS—2），在特定的条件下允许使用一个不带NEXT的FOR—TO语句，NEXT是蕴涵的，而不真正地把NEXT写出来。

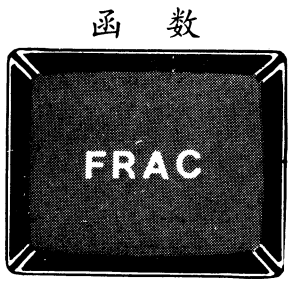
例如：

```
10 PRINT X,SQR(X) FOR X=1 TO 12
```

它打印出从1 到12的数和这些数的平方根表。

参阅

NEXT, STEP



一些计算机（例如，Micropolis BASIC）用FRAC函数来分离出一个数的小数部分以及它们的符号。例如，30 F= FRAC（－12.345）把值－0.345赋给F。

测试程序

```
10 REM 'FRAC' TEST PROGRAM
20 N = -12.5
30 F = FRAC(N)
40 PRINT "FRAC PASSED THE TEST IF -0.5 ="; F
99 END
```

运行实例

```
FRAC PASSED THE TEST IF -0.5 = -0.5
```

如果你的计算机没有此功能

如果FRAC函数不能通过测试的话，可用下面一行来代替30行：

```
30 F = N - SGN(N)*INT(ABS(N))
```

然后再运行测试程序。

参阅

INT，FIX

FREE

FRE（字符串）函数用来指出计算机存储器中已分配但没有使用的字符串空间的总字节数。任一个字符（放在引号中的）或字符串变量可以和FRE函数一起使用。下面50行中的B$完全是任意的。

大多数具有FRE函数能力的计算机在打开计算机时，自动地保留50个字节的字符串空间。

测试程序#1

```
10 REM 'FRE(STRING)' TEST PROGRAM
20 PRINT "ENTER ANY COMBINATION OF LETTERS AND NUMBERS";
30 INPUT A$
40 PRINT "THE AMOUNT OF UNUSED STRING SPACE=";
50 PRINT FRE(B$)
99 END
```

运行实例（典型值，输入COMPUTER）

```
ENTER ANY COMBINATION OF LETTERS AND NUMBERS? COMPUTER
THE AMOUNT OF UNUSED STRING SPACE = 42
```

在测试程序试用字母和数的各种不同的组合来证明FRE函数的作用。

某些计算机把数或数字变量用在FRE函数中来指出剩余的内存总数（而不是留给字符串专用的那一部分），这类似于MEM语句。

测试程序#2

```
10 REM 'FRE(MEMORY)' TEST PROGRAM
20 PRINT FRE(N);"BYTES OF MEMORY REMAIN."
99 END
```

运行实例

```
 13504 BYTES OF MEMORY REMAIN.
```

剩余的内存的总数将取决于你的计算机内存的容量。

通常打入NEW把所有的字符串（和数值）变量重新置空（和零），这样，整个的内存又变成可以利用的了。

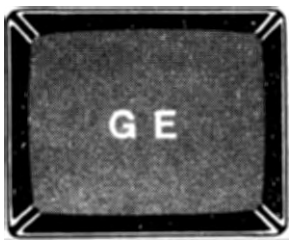
替代的形式

许多BASIC（其中包括北极星，Processor Technology和Digital Group MAXI-BASIC）使用FREE(0)来得到剩余的内存总数。在测试程序2中的20行使用FREE(0)就可以看出你的计算机是否接受它。

参阅

MEM，CLEAR，$，NEW

操作符



一些计算机（例如，TI990）用GE操作符作为“大于或等于”符号（>=）的缩写。

要想更详细地了解请参看>=。

测试程序

```
10 REM 'GE' (GREATER THAN OR EQUAL TO) TEST PROGRAM
20 IF 20 GE 10 THEN 50
30 PRINT "THE GE OPERATOR FAILED THE TEST IN LINE 20"
40 GOTO 99
50 IF 20 GE 20 THEN 80
60 PRINT "THE GE OPERATOR FAILED THE TEST IN LINE 50"
70 GOTO 99
80 PRINT "THE GE OPERATOR PASSED THE TEST"
99 END
```

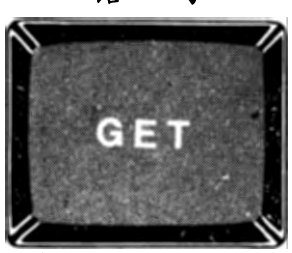
运行实例

```
THE GE OPERATOR PASSED THE TEST
```

参阅

>=, IF-THEN

语　句



GET是一个由一些计算机（例如，PET和APPLEⅡ）使用的一个语句，它从键盘接收一个字符，此字符不显示在屏幕上，也不需要等待按RETURN键。GET语句的使用类似于INKEYS语句。

如果用的是一个数字变量，例如GET A，则GET语句仅接收数值输入。如果用一个字符串变量（例如，GETAS），则可以接收任何一个键（STOP键除外）的输入。

在APPLEⅡ中，GET语句可以使程序的执行暂停直到按下任一个键时为止。在PET中GET语句搜索键盘，如果GET语句没有发现被按下的键，它就存入一个空字符，并且继续进行这个程序的其余部分。（更详细的资料可参阅INKEY$）。

测试程序

```
10 REM 'GET' TEST PROGRAM
20 PRINT "TYPE IN ANY CHARACTER"
30 GET A$
40 IF A$="" THEN 30
50 PRINT "YOU JUST PRESSED THE ";A$;" KEY."
60 PRINT "PRESS ";A$;" AGAIN TO CONTINUE."
70 GET B$
80 IF B$=A$ THEN 20
90 GOTO 70
99 END
```

运行实例（输入X）

```
TYPE IN ANY CHARACTER.
YOU JUST PRESSED THE X KEY.
PRESS X AGAIN TO CONTINUE.
```

其它用法

很多计算机（例如，DEC PDP—11）使用GET＃来从磁盘或磁带读一个个记录。

例如：GET＃2，REC%

从文件＃2中读取存放在记录号REC%中的信息。

一些BASIC（例如，NEC的N—BASIC, TRS—80 Extended color BASIC和Micro Soft Level Ⅲ BASIC）提供了GET@，它存储显示在屏幕上的一部分的信息。必须指定长方形对角的X、Y的位置，还要加上将要存放信息的那个数组的名字。

例如：

```
GET@(10,8)-(25,14),A%
```

在数组A%中把字符和图形符号存放在10～25和8～14行的地方。用一个诸如PUT@语句又可以把这个信息显示在屏幕上。

Hewlett—Packard BASIC 用 GET语句来从磁盘或磁带装入一个程序或数据文件。

例如：

```
GET "PROG"
```

清除内存中的任何程序，然后再“装入”PROG。

用GET也可以在一个现存程序的末尾附加（APPEND）一个程序段，其方法是指定一个分配给被“装入”的程序的第一个行号。

```
GET "SUB1", 500
```

当程序SUB1 装入到内存中时，从行号500开始重新给程序SUB1编行号。行号小于500的现有程序中所有行号（小于500的行）都保留，然后接着就是附加在末尾的从行号500开始的SUB 1 的程序行。

如果你的计算机没有此功能

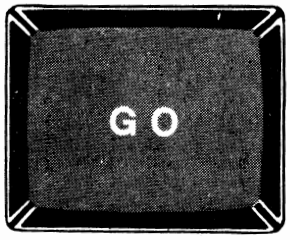
如果你的计算机不能运行上面的测试程序，那么计算机可能是使用另一个关键字来接收来自于键盘的一个字符。在30行和70行中用下列字之一：INKEY$、KEY$、INCH、

INCHAR、KEYIN再试运行测试程序。

参　阅

INKEY$, KEY$, PUT, APPEND, #

语　句
命　令



GO是作为GOTO语句和GOSUB语句的一部分来使用的。通常只有当GO与另一个BASIC字组合在一起时，才有意义。大多数计算机不计较在GO后面是否有空格，而是自动地转换成GOTO或GOSUB语句。其它的计算机（例如，TRS—80Level Ⅰ）则不允许在GO后面有空格。

下面这个程序在GOTO语句中使用GO。

更详细的资料可参看GOTO。

测试程序＃1

```
10 REM  GO' TEST PROGRAM
20 PRINT "THE GO STATEMENT";
30 GO TO 60
40 PRINT "FAILED THE TEST"
50 GOTO 99
60 PRINT "PASSED THE TEST."
99 END
```

运行实例

```
THE GO STATEMENT PASSED THE TEST.
```

下面这个程序在GOSUB语句中使用GO。更详细的资料可参看GOSUB。

测试程序＃2

```
10 REM 'GO' (USED WITH SUB) TEST PROGRAM
20 GO SUB 100
30 PRINT "PASSED THE TEST WHEN USED WITH SUB."
40 GO TO 999
100 REM SUBROUTINE
110 PRINT "THE GO STATEMENT";
120 RETURN
999 END
```

运行实例

```
THE GO STATEMENT PASSED THE TEST WHEN USED WITH SUB.
```

其它用法

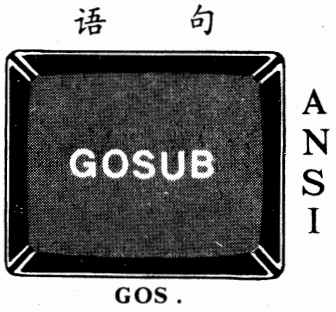
DATAPONT把GO作为GOTO的一个缩写。它可作为一个语句和一个直接命令。作为一个命令，GOn在n行重新启动该程序。

参　阅

GOTO，GOSUB，IF-GOTO，ON-GOTO，GOTO-OF，ON-GOSUB，GOSUB-OF，CONT

GOSUB语句的作用是使程序离开主程序而转去执行一个子程序。GOSUB语句后面必须紧跟着一个行号，该行号指明要去执行的那个子程序的第1行。

在子程序执行的末尾必须使用一个RETURN语句以便控制从子程序返回到主程序。



测试程序

```
10 REM 'GOSUB' TEST PROGRAM
20 GOSUB 100
30 PRINT "PASSED THE TEST AT LINE 20"
40 GOTO 999
100 REM SUBROUTINE
110 PRINT "THE GOSUB STATEMENT ";
120 RETURN
999 END
```

运行实例

```
THE GOSUB STATEMENT PASSED THE TEST AT LINE 20
```

某些计算机（例如，SinclairZX－80和ACORN ATOM）允许在GOSUB语句中用变量或表达式。在这类计算机中GOSUB N以及GOSUB N＊10＋100都是可以接收的语句。在执行该程序的过程中，必须给N一个合适的值。如果N是一个小的整数，那么GOSUB N＊10产生的结果类似于ON N GO SUB10，20，30……。把下面这些行插入到测试程序中，以便对此特性进行测试：

```
15 N=100
20 GOSUB N
```

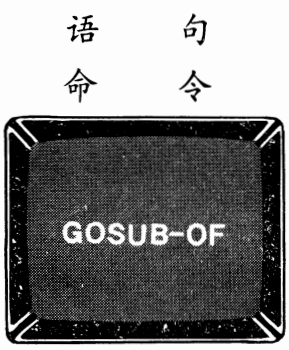
然后再运行测试程序。

替代的形式

在各种Tiny BASIC中把GOS.作GOSUB的一个缩写。

参 阅

RETURN, ON-GOSUB, IF-GOSUB



GOSUB—OF语句是一个多分支转子程序语句，它的转移方式类似由H－P和Tektronix计算机使用的ON－GOSUB语句。

例如：GOSUB X OF1000,2000使计算机转移到以1000行开头的那个子程序(若X = 1),也可以转移到以2000行开头的那个子程序（若X = 2）。

测试程序

```
10 REM 'GOSUB-OF' TEST PROGRAM
20 PRINT "ENTER THE NUMBER 1, 2, OR 3";
30 INPUT X
40 PRINT "THE GOSUB-OF STATEMENT ";
50 GOSUB X OF 100, 200, 300
60 GOTO 20
100 REM SUBROUTINE #1
110 PRINT "BRANCHED TO SUBROUTINE #1"
120 RETURN
200 REM SUBROUTINE #2
210 PRINT "BRANCHED TO SUBROUTINE #2"
220 RETURN
300 REM SUBROUTINE #3
310 PRINT "BRANCHED TO SUBROUTINE #3"
320 RETURN
999 END
```

运行实例

```
ENTER THE NUMBER 1, 2, OR 3? 1
THE GOSUB-OF STATEMENT BRANCHED TO SUBROUTINE #1
ENTER THE NUMBER 1, 2, OR 3? 2
THE GOSUB-OF STATEMENT BRANCHED TO SUBROUTINE #2
ENTER THE NUMBER 1, 2, OR 3? 3
THE GOSUB-OF STATEMENT BRANCHED TO SUBROUTINE #3
ENTER THE NUMBER 1, 2, OR 3?
```

参 阅

ON-GOSUB, GOTO-OF, ON-GOTO

GOTO语句使程序的执行“转移”到某个指定的行号。很多计算机也可以按两个字GOTO来接收这个语句。

语　句

GO TO
GOT
G.

测试程序

```
10 REM 'GOTO' STATEMENT TEST PROGRAM
20 PRINT "THE GOTO STATEMENT ";
30 GOTO 60
40 PRINT "FAILED."
50 STOP
60 PRINT "HAS PASSED THE TEST."
99 END
```

运行实例

```
THE GOTO STATEMENT HAS PASSED THE TEST.
```

一些计算机（例如：SinclairZX－80，ACORN ATOM以及那些使用Micropolis BASIC的机器）允许在GOTO语句中用变量或表达式。在这些计算机中类似GOTO N甚至GOTO N＊10＋100 都是可以接收的语句。在该程序执行中，必须给N一个适当的值以便引用一个现有的行号。如果N是一个小的整数，GOTO N＊10产生的结果类似于ON N GOTO 10，20，30……。把下列两行插入到测试程序中来检测这一特性：

```
25 N=60
30 GOTO N
```

然后再运行测试程序。

其它用法

GOTO语句常和其它一些关键字一起使用。

替代的形式

在测试程序的30行中试用GOT和G. 以观察你的计算机是否接收这些缩写。

参　阅

IF-GOTO，ON-GOTO，and GOTO-OF

语　句

GOTO—OF语句是被某些计算机（例如，Hewlett Packard和Tektronix）用作一个多转移的工具，它把一系列IF—THEN测试放在一个语句中。

例如：GOTO X OF100,200,300，如果X的整数值分别等于1，2或3的话，那么就指示计算机转移到100行，200和300行。如果INT X的值小于1或大于3的话，那么在这个例子中的测试完全失败，这样，程序执行下一个程序行。X的INT值不能超过这个语句中可能分支的个数。

大多数计算机既可以接收GO TO语句（两个字）又可以接收GOTO（一个字），而有几种计算机（例如：VARIN620）只能把GO TO作为两个字来接收。

测试程序

```
10 REM 'GOTO OF' TEST PROGRAM
20 X=2
30 GOTO X OF 40,60
40 PRINT "'GOTO-OF' FAILED THE TEST"
50 GOTO 99
60 PRINT "'GOTO-OF' PASSED THE TEST"
99 END
```

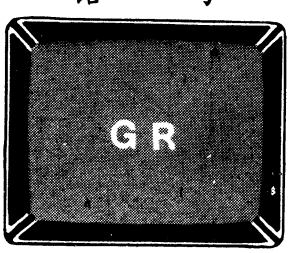
运行实例

```
'GOTO-OF' PASSED THE TEST
```

参　阅

ON-GOTO，ON-GOSUB，IF-THEN，INT，GOSUB-OF

命　令
语　句



在APPLEⅡ BASIC中，GR既可作为一个命令又可作为一个语句使用。用它把计算机的操作从文本（TEXT）方式转变成图形方式。在使用专门的作图语句PLOT，HLIN—AT和VLIN—AT之前必须先执行GR语句。

在开始一个新图形显示之前，也可用GR清除屏幕。每执行一次GR语句，计算机就要清除整个屏幕。

测试程序

```
10 REM 'GR' TEST PROGRAM
20 GR
30 COLOR=6
40 HLIN 0,39 AT 20
50 END
```

运行实例

如果计算机接收了GR语句，那么就会有一条横跨屏幕的蓝色的水平线出现。

参 阅

TEXT，COLOR，HLIN-AT，VLIN-AT，PLOT，CLS

在一些计算机中（例如，TI990）把GT用来作为“大于”符号（>）的一个替换形式。

更详细的资料可参看>。

操作符

测试程序

```
10 REM 'GT (GREATER THAN)' TEST
   PROGRAM
20 IF 10 GT 5 THEN 50
30 PRINT "THE GT OPERATOR FAILED
   THE TEST"
40 GOTO 99
50 PRINT "THE GT OPERATOR PASSED THE TEST"
99 END
```

运行实例

```
THE GT OPERATOR PASSED THE TEST
```

参 阅

>，IF-THEN，>=，<，<=，=，<>，EQ，GE，LE，LT，NE

语　句
命　令

有几种计算机中（例如，Sharp/TRS—80 袖珍计算机和 Tektronix4050 系列机）以 GRAD（梯度）为单位进行计算，而不是以弧度为单位。（100 梯度 = 90 度）。大多数计算机在接通时处于弧度方式，但有些计算机也具有以度数为单位来计算三角函数的能力，少数计算机则以梯度进行计算。

测试程序

```
10 REM 'GRAD' TEST PROGRAM
20 R = SIN(40)
30 PRINT "THE SINE OF 40 RADIANS IS";R
40 GRAD
50 G = SIN(40)
60 PRINT "THE SINE OF 40 GRADS IS";G
99 END
```

运行实例

```
THE SINE OF 40 RADIANS IS 0.745113
THE SINE OF 40 GRADS IS 0.587785
```

参　阅

RAD，DEG，ACS，ASN，ATN，COS，SIN，TAN

在APPLEⅡ BASIC 中HILT－AT 语句作为一个专门的特性，用它能在屏幕指定行上显示一条水平线。

水平线的长度是由跟着HILT 后面的两个数确定的。这些数指出线的界限。线的长度可以在 0 到39之间。

跟在A T 后面的那个数代表要画出的那根线所在的行号。这个数可以在 0 到39的范围内。

例如：HLIN 10，30 A T 20 告诉计算机在20行的第10列第30列画一根水平线。

在计算机接收HLIN －A T 语句之前必须先执行G R 语句（参看G R）。线的颜色是由COLOR 语句确定的(参看COLOR)。

测试程序

```
10 REM 'HLIN-AT' TEST PROGRAM
20 GR
30 Y=0
40 FOR X=0 TO 39
50 COLOR = Y
60 HLIN 0,39 AT X
70 Y=Y+1
80 IF Y < 16 THEN 100
90 Y=0
100 NEXT X
999 END
```

运行实例

如果计算机接受了HLIN－AT语句，那么39条各种颜色的水平线会充满屏幕。

APF BASIC 在HLIN语句中不用AT。它用的是一个逗号。所使用的形状和颜色由SHAPE和COLOR 语句说明，它们应在HLIN使用之前出现。

参阅

GR，COLOR，PLOT，VLIN-AT，TEXT

HOME 是一个用来清除屏幕并把光标定位在左上角的命令。HOME 命令类似于其它计算机上的CLS命令。

HOME 也可以作为一个程序语句以便在程序产生一个图形显示之前清除屏幕。

测试程序

```
10 REM 'HOME' TEST PROGRAM
20 FOR I=1 TO 12
30 PRINT "THIS NEEDS TO BE ERASED"
40 NEXT I
50 HOME
60 PRINT "HOME PASSED IF THIS IS ALL THAT IS DISPLAYED"
99 END
```

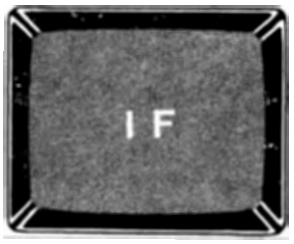
运行实例

```
 HOME PASSED IF THIS IS ALL THAT IS DISPLAYED
```

参阅

CLS

语 句
修饰词



ANSI

IF 语句是条件转移语句：IF－THEN, IF－GOTO, IF－GOSUB, IF－LET 等语句的一部分。IF 语句用来指出用一个关系操作符（参看 =，<，>，<=，>=，<>）测试的结果。

例如：IF X=3 THEN 100 当X=3时计算机就分支或“转移”到100行。如果条件不满足（即X≠3），那么测试就归于“失败”，程序将继续执行下一行。

这些条件的IF—THEN测试必须安排在多语句行的最后，因为计算机或是分支到指定的行号（如果测试结果为“真”），或是失败,程序转移到下一个程序行(如果测试结果为“假”）。

例如：30 IF X=3 THEN 100：PRINT “X=3”

PRINT 语句永远都不会被执行。

测试程序

```
10 REM 'IF' TEST PROGRAM
20 X=10
30 IF X=10 THEN 60
40 PRINT "'IF' FAILED THE TEST"
50 GOTO 99
60 PRINT "'IF' PASSED THE TEST"
99 END
```

运行实例

```
'IF' PASSED THE TEST
```

为了更进一步检查你的计算机的IF功能，可以参看后几页IF－GOTO 语句和IF－LET 语句下面的测试程序。

其它的用法

少数BASIC （例如，DEC BACSIC－PLUS－2 ）把IF当作大多数其它语句的修饰词来使用。

例如：X=Y IF X<Y

赋值语句X=Y只有当X的当前值小于Y的值时才被执行。

参阅

IF-THEN，IF-GOTO，IF-LET，ELSE

IF—G.
IF—GOT

IF—GOTO是使用一个关系操作符（参看=，<，>，<=，>=，<>）的条件转移语句。

当IF—GOTO语句的条件得到满足时，计算机就执行转移语句GOTO。

例如：IF X= 3 GOTO 100 告诉计算机如果X等于3就“分支”或“跳”到100行。如果条件得不到满足（即X≠3），测试“失败”，程序将继续执行下一行。

测试程序

```
10 REM 'IF-GOTO' TEST PROGRAM
20 X=30
30 IF X=30 GOTO 60
40 PRINT "THE IF-GOTO STATEMENT FAILED THE TEST"
50 GOTO 99
60 PRINT "THE IF-GOTO STATEMENT PASSED THE TEST"
99 END
```

运行实例

```
THE IF-GOTO STATEMENT PASSED THE TEST
```

替代的形式

有些计算机允许用IF—GOT（例如，PDP—8E）或IF—G.（例如，TRS—80 Level I）作为IF—GOTO语句的一个缩写形式。

其它用法

一些解释程序允许把THEN用来代替GOTO。参看IF—THEN。

参阅

GOTO，GOSUB，ELSE，IF-THEN，IF

IF—LET语句是使用一个关系操作符（参看=，<，>，<=，>=，<>）的条件LET语句。

当IF—LET语句的条件得到满足时，计算机把某个值赋给跟在LET后面的那个变量。

测试程序

```
10 REM 'IF-LET' TEST PROGRAM
20 X=30
30 IF X>20 LET X=10
40 PRINT "X =";X
50 PRINT " 'IF-LET' PASSED THE TEST IF THE VALUE OF
   X IS 10"
99 END
```

运行实例

```
 X = 10
 'IF-LET' PASSED THE TEST IF THE VALUE OF X IS 10
```

其它用法

LET 语句的使用在很多计算机中是不一致的，大多数机器允许把LET 省略掉，而其它一些机器则允许用THEN 代替LET 。

大多数允许使用IF－LET 语句的计算机也允许使用任一个可执行的语句，例如，PRINT, READ, INPUT, GOSUB, 以及另一个IF 语句等等。

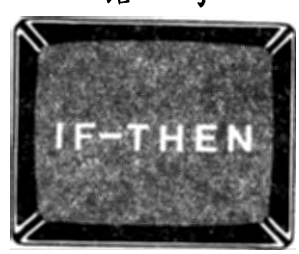
例如：

改变30 IF X>20 IF X>40 X=10

参阅

IF-THEN, LET, IF

语　句



ANSI

IF—THEN语句是使用一个关系操作符（参看=，<，>，<=，>=，<>）的条件转移语句。

当IF—THEN语句的条件得到满足时，计算机就转移到THEN后面的那个行号。例如：IF X=3 THEN 100告诉计算机，如果X= 3计算机就转移到100行。如果条件得不到满足（即X≠3），则测试“失败”，程序继续执行下一行。

IF—THE
IF—T .

测试程序#1

```
10 REM 'IF-THEN' TEST PROGRAM
20 X=30
30 IF X=30 THEN 60
40 PRINT "THE IF-THEN STATEMENT FAILED THE TEST"
50 GOTO 99
60 PRINT "THE IF-THEN STATEMENT PASSED THE TEST IN
   LINE 30"
99 END
```

运行实例

```
THE IF-THEN STATEMENT PASSED THE TEST IN LINE 30
```

某些计算机允许当IF－THEN语句的条件满足时，执行数学运算。例如：IF A= 3 THEN X= 2 ＊（A+ 6）/3，如果变量A的值等于3，就计算X的值。如果A的值不等于3，那么测试失败，程序续继执行下一行。

为了测试你的计算机的这一特性，加上以下的程序行：

```
70 IF X=30 THEN X=X+90
80 PRINT "X=";X
90 PRINT"IF-THEN PASSED THE TEST IN LINE 70 IF X=120"
```

运行实例

```
THE IF-THEN STATEMENT PASSED THE TEST IN LINE 30
X=120
IF-THEN PASSED THE TEST IN LINE 70 IF X=120
```

一些解释程序允许当IF—THEN的条件得到满足时，执行任一个操作语句。例如：IF X= 3 THEN END，当X的值等于3时，就会终止程序的执行。

把下面一行加到测试程序中来检查这个功能：

```
100 IF X=120 THEN PRINT "IF-THEN PASSED THE TEST IN
    LINE 100"
```

运行实例

```
THE IF-THEN STATEMENT PASSED THE TEST IN LINE 30
X=120
IF-THEN PASSED THE TEST IN LINE 70 IF X=120
IF-THEN PASSED THE TEST IN LINE 100
```

在很多计算机中THEN语句的使用是不一致的。当IF后面直接跟着一个数学运算或操作语句，或转移语句时，很多计算机允许把THEN省略。

测试程序#2

这个程序对三个不同的隐含（即不使用）THEN进行测试。

```
10 REM TEST PROGRAM WITH IMPLIED 'THEN' IN LINES 30,
   60 AND 999
20 X=30
30 IF X=30 GOTO 60
40 PRINT "LINE 30 FAILED THE TEST"
50 GOTO 999
60 IF X=30 GOSUB 100
70 GOTO 999
100 REM SUBROUTINE
110 PRINT "LINES 30 AND 60 PASSED THE TEST. DOES LINE
    999?"
120 RETURN
999 IF X=30 END
```

运行实例

```
LINES 30 AND 60 PASSED THE TEST. DOES LINE 999?
```

在使用允许多语句行的解释程序时，必须要特别小心。如果IF—THEN的测试“失败”,程序转移到**下一行**，而不是转移到同一行上的下一个语句。为此，在同一行内，IF—THEN语句测试的后面通常不再跟以其它的语句。因为这是一个不好程序设计风格。

例如：IF X = 5 THEN X=X+Y: PRINT X。在大多数计算机中,如果X的值不等于5，那么就不执行PRINT语句。

有些计算机接受象50 IF X THEN 120这样的语句。当X是一个非零的值时，这类计算机可以使程序控制转移到120行，当X = 0时,计算机就要执行50行后面的那一行。

测试程序#3

```
10 REM 'IF X THEN' TEST PROGRAM
20 PRINT "TYPE IN A NUMBER"
30 INPUT X
40 IF X THEN 70
50 PRINT "YOUR NUMBER WAS ZERO THAT TIME."
60 GOTO 20
70 PRINT "YOU ENTERED A NON-ZERO VALUE OF";X
80 GOTO 20
99 END
```

运行实例

```
TYPE IN A NUMBER ? 4
YOU ENTERED A NON-ZERO VALUE OF 4
TYPE IN A NUMBER ? 0
YOUR NUMBER WAS ZERO THAT TIME.
TYPE IN A NUMBER ?
```

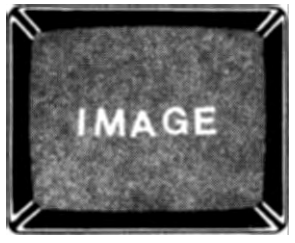
替代的形式

有些计算机允许把IF—THE（例如：PDP—8E）或IF—T.（例如：TRS—80 Level I）作为IF—THEN的缩写形式。

参阅

IF，IF-GOTO，IF-LET，ELSE，GOTO，GOSUB，STOP，END

语　句



一些计算机（例如：Hewlett—Packard）用IMAGE语句和PRINT USING语句一起指定打印格式，例如：

```
110 IMAGE 6A, 2D, ":", 2D, ":", 2D
.
.
190 PRINT USING 110, "TIME: ", H, M, S
```

产生象下面形式的结果：

```
TIME: 12:55:31
```

可以用在HP IMAGE 语句中的格式字符是：

A　每个要打印的字母用一次A(可以把一个数放在A的前面，例如：6A。数字6表示要打印的字符数目)

D　每一位要打印的数字用一个D（也可把一个数放在D的前面）

E　用来表示指数格式（例如：用D.4DE的格式打印12345.67应该打印出1.2346E+04)

S　用来表示一个要打印的正（+）号或负（-）号

X　每个要打印的空格用一个X（也可把一个数放在X的前面）。

.（句号)用来指出小数点打印的位置。

,（逗号)和/（斜杠）作为格式分隔符。/（斜杠）还可以产生一个新行。

( )　用来重复一组格式描述（例如：2（5D.2D, 3X, 4D) 与5D.2D,3X,4D,5D.2D, 3X, 4D是相同的）

+　抑制在打印行的末尾换行，使下一个PRINT语句也从本行的开头起打印。

-　抑制回车，使下一个PRINT语句打印在下一行的下一个位置上。

#　既抑制回车又抑制换行，使下一个PRINT语句从当前行结束的地方接着打印。

在IMAGE行中，引号中的字符串常数可以用于任何地方。

参阅

FMT，:，PRINT USING

函　数

INDEX函数告诉我们一个短字符串的第一个字符在长字符串中的起始位置，这个短字符串是长字符串的一部分。字符的位置是从左边开始计算的。

例如：

```
INDEX("ABADABA","DAB") = 4
```

如果证明短字符串不是长字符串的一部分，那么INDEX=0。

INDEX类似于其它计算机中的INSTR。

测试程序

```
10 REM 'INDEX' TEST PROGRAM
20 A$ = "KEYBOARD"
30 B$ = "OAR"
40 K = INDEX(A$,B$)
50 IF K<>5 THEN 110
60 B$ = "ORE"
70 K = INDEX(A$,B$)
80 IF K<>0 THEN 110
90 PRINT "'INDEX' PASSED THE TEST"
100 GOTO 30999
110 PRINT "'INDEX' FAILED THE TEST"
30999 END
```

运行实例

```
'INDEX' PASSED THE TEST
```

如果你的计算机没有此功能

如果测试程序通不过的话，可以试用INSTR或POS代替INDEX。如果INSTR和POS在你的计算机上也通不过，就用印在INSTR下面的那个子程序（本书121页）（为节约篇幅，在此不印出）。

为了使这个子程序与测试程序一起使用，对程序做些修改：

```
35 N = 1
40 GOSUB 30060
70 GOSUB 30060
```

参阅

INSTR，POS

函　数

INKEY$函数用来在每执行一次INKEY$时,从键盘上读一个字符。和INPUT语句不同，INKEY$并不停止执行程序以等待按ENTER键。计算机只等待很短的时间直到收到从键盘来的一个信息。如果未按键盘上的任一键，INKEY$仅读入一个“空”字符串（ASCII码为0）。

因为INKEY$函数不等待你从键盘输入的字符和按“ENTER”键，所以INKEY$通常放在程序循环中以便反复地扫描键盘来寻找一个被按的键。

例如：

```
10 IF INKEY$="X" GOTO 100
20 GOTO 10
100 PRINT "YOU HIT 'X', DIDN'T YOU?"
110 GOTO 10
```

INKEY$函数反复地查找是否从键盘输入字母X，如果发现，则IF—THEN语句的条件得到满足。当输入字母X时，程序流程就转移到100行。

测试程序

```
10 REM 'INKEY$' TEST PROGRAM
20 CLS
30 PRINT "PRESS ANY KEY ON THE KEYBOARD"
40 A$=INKEY$
50 IF A$="" GOTO 40
60 PRINT "YOU HAVE JUST PRESSED THE ";A$;" KEY"
70 PRINT: PRINT "PRESS THE ";A$;" KEY AGAIN TO START
   OVER"
80 IF INKEY$=A$ GOTO 20
90 GOTO 80
99 END
```

运行实例（输入R）

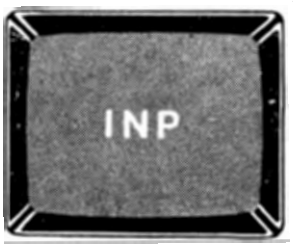
```
PRESS ANY KEY ON THE KEYBOARD
YOU HAVE JUST PRESSED THE R KEY

PRESS THE R KEY AGAIN TO START OVER
```

参阅

INPUT，IF-THEN，INPUT$，GET

语 句



INP 语句代表“从端口的输入”。

INP 语句用来将指定的计算机端口上的一个字节的十进制数值信息读入计算机。该字节的值可以是从 0 到 255 的任一个正整数。

例如：PRINT INP（X）打印X端口的那个字节的十进制数值。

对于一个特定的条件而言，INP 语句是到监控端口的一个非常有用的工具。例如可以从一个远程外部设备输入信息。其它的应用包括从太阳能热水加热系统上的远程传感器上读入温度等。

测试程序

```
10 REM 'INP' TEST PROGRAM
20 FOR X=0 TO 255
30 PRINT "THE DECIMAL VALUE OF THE BYTE AT PORT#";
40 PRINT X;"IS";INP(X)
50 NEXT X
99 END
```

运行实例（典型的）

```
THE DECIMAL VALUE OF THE BYTE AT PORT# 0 IS 255
                    ⋮
THE DECIMAL VALUE OF THE BYTE AT PORT# 255 IS 127
```

其它用法

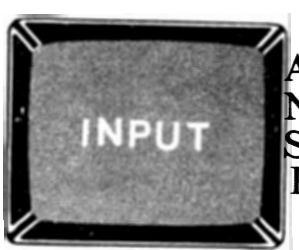
在PDP—8 版本中INP 语句作为INPUT 语句的一个缩写。

更详细的资料可参阅INPUT。

参阅

OUT，PEEK，POKE，INPUT，PIN

语 句



ANSI

IN.

I.

INPUT 语句允许用户从键盘上将数据赋给变量。当计算机执行一个INPUT 语句时，计算机打印一个问号，表示机器正在等待你把数据赋给某个变量。计算机继续等待，直到按下ENTER（或RETURN）键为止。

测试程序#1

```
10 REM 'INPUT' STATEMENT TEST PROGRAM
20 PRINT "ASSIGN A VALUE TO THE VARIABLE X"
30 INPUT X
40 PRINT "THE VALUE OF X IS";X
99 END
```

运行实例（输入10）

```
ASSIGN A VALUE TO THE VARIABLE X
? 10
THE VALUE OF X IS 10
```

替代的形式

大多数Tiny BASIC以及ACORN计算机允许使用IN.作为INPUT的缩写。TRS-80 Level I可以接受I.，而PDP-8的一些版本允许使用INP。

其它用法

在微型机较新的解释程序版本中，INPUT语句兼有PRINT和INPUT二者的功能（这样可节省空间）。

测试程序#2

```
10 REM 'INPUT/PRINT' STATEMENT TEST PROGRAM
20 INPUT "ASSIGN A VALUE TO THE VARIABLE 'X'"; X
30 PRINT "THE VALUE OF X IS";X
99 END
```

运行实例（输入10）

```
ASSIGN A VALUE TO THE VARIABLE 'X' ? 10
THE VALUE OF X IS 10
```

注意在INPUT语句之前没有PRINT语句，PRINT和INPUT二者的功能是由20行的INPUT组合在一起的。

参阅

INPUT1，INKEY$，INP，INPUT$，PIN，LINEINPUT，INPUTLINE

INPUTLINE 语句

INPUTLINE 语句类似于INPUT。INPUTLINE 用于TSC 扩展BASIC 和PRIME BASIC/VM中以接收一整行的输入，并把它赋给一个字符串变量。INPUTLINE 语句与INPUT 语句一样都要“提醒”用户（INPUTLINE 用一个！）。INPUTLINE 语句所“输入”的字符串可以包括逗号，冒号和其它一些专用字符，它们不需要放在引号中。

INPUTLINE 语句类似于LINEINPUT 语句。（详见LINEINPUT 语句。）

测试程序

```
10 REM INPUTLINE TEST PROGRAM
20 PRINT "TYPE YOUR NAME - LAST, FIRST";
30 INPUTLINE N$
40 PRINT "HELLO, ";N$;". I'M COMPUTER, MICRO."
99 END
```

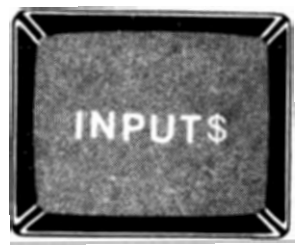
运行实例

```
TYPE YOUR NAME - LAST, FIRST ! DOE, JOHN
HELLO, DOE, JOHN. I'M COMPUTER, MICRO.
```

参阅

LINEINPUT, INPUT

INPUT$ 函数



INPUT$（n）是一个Microsoft BASIC 函数，用它可以读来自于键盘的指定数目的字符（不显示在屏幕上），在输入最后一个字符以后，不需按ENTER 或RETURN 键，但是在程序往下进行之前，必须打入正确的字符个数。

例如：Z$ = INPUT$（5）使程序暂停，用户输入 5 个字符。

测试程序

```
10 REM INPUT$() TEST PROGRAM
20 PRINT "TYPE IN ANY COMBINATION OF 3 CHARACTERS."
30 A$ = INPUT$(3)
40 PRINT "YOU JUST TYPED ";A$
99 END
```

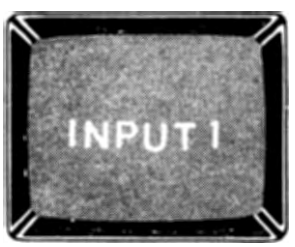
运行实例（输入Q W E ）

```
TYPE IN ANY COMBINATION OF 3 CHARACTERS.
YOU JUST TYPED QWE
```

参阅

```
INKEY$, KEY$, GET, INPUT
```

语　句



几种计算机（例如：北极星和使用Maxi—BASIC 的Digital Group）以类似于INPUT 语句的方式使用INPUT 1 语句，但是INPUT1语句在输入数据变量以后，不回车和换行。

详见INPUT语句。

测试程序

```
10 REM 'INPUT1' TEST PROGRAM
20 PRINT "ENTER A VALUE FOR THE VARIABLE X"
30 INPUT1 X
40 PRINT " VARIABLE X=";X
50 PRINT "INPUT1 PASSED THE TEST IF THE WORDS
   VARIABLE X ="; X
60 PRINT "ARE PRINTED ON THE SAME LINE AS THE ? SIGN"
99 END
```

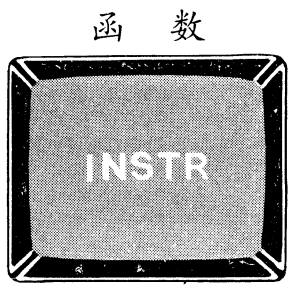
运行实例（输入1 0 ）

```
ENTER A VALUE FOR THE VARIABLE X
? 10 VARIABLE X = 10
INPUT1 PASSED THE TEST IF THE WORDS VARIABLE X = 10
ARE PRINTED ON THE SAME LINE AS THE ? SIGN
```

参阅

```
INPUT, INP, INPUTLINE, LINEINPUT, PIN
```

INSTR（N，A$，B$）是一个字符串函数，该函数给出字符串B$在字符串A$中第一次出现的起始位置。如果B$不在A$里面，那么INSTR=0。N是可选的，用来指定在A$的第N个的字符处开始寻找。详见POS。



测试程序#1

```
10 REM 'INSTR' TEST PROGRAM
20 A$ = "PROGRAM"
30 B$ = "RAM"
40 K = INSTR(A$, B$)
50 IF K<>5 THEN 110
60 B$ = "ROM"
70 K = INSTR(A$,B$)
80 IF K<>0 THEN 110
90 PRINT "'INSTR(A$,B$)' PASSED THE TEST"
100 GOTO 30999
110 PRINT "'INSTR(A$,B$)' FAILED THE TEST"
30999 END
```

运行实例

```
'INSTR(A$,B$)' PASSED THE TEST
```

测试程序#2

```
10 REM 'INSTR(N,A$,B$)' TEST PROGRAM
20 A$ = "COMPUSOFT"
30 B$ = "O"
40 K = INSTR(4,A$,B$)
50 IF K=7 THEN 80
60 PRINT "'INSTR(N,A$,B$)' FAILED THE TEST"
70 GOTO 30999
80 PRINT "'INSTR(N,A$,B$)' PASSED THE TEST"
30999 END
```

运行实例

```
'INSTR(N,A$,B$)' PASSED THE TEST
```

如果你的计算机没有此功能

如果两个测试都不成功，试用POS和INDEX函数。如果你的计算机可以使用MID$和LEN函数，则可使用下面这个子程序：

```
30000 GOTO 30999
30060 REM * INSTR SUBROUTINE * INPUT N, A$, B$, OUTPUT K
30062 REM ALSO USES L INTERNALLY
30064 L = LEN(B$)
30066 FOR K=N TO LEN(A$)-L+1
30068 IF B$=MID$(A$,K,L) THEN 30074
30070 NEXT K
30072 K=0
30074 RETURN
```

为了把这个子程序与测试程序#1一起使用，做下面一些改变：

```
35 N = 1
40 GOSUB 30060
70 GOSUB 30060
```

对测试程序 # 2 也可以做如下的改变：

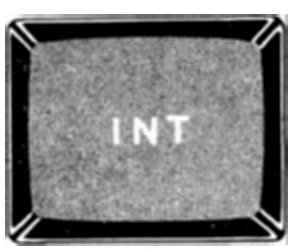
```
35 N = 4
40 GOSUB 30060
```

参阅

POS，INDEX，MID$，LEN

函　数

ANSI



I.

INT（取整）函数用来把数舍入到整数值（整数）。在BASIC 中，取整后的数总是比原来的数小。整数的值与被截去的小数值无关。只是当负数取整时，结果是舍入到下一个更小的整数。例如：INT（－4.65）就变成－5。

一些计算机用INT 函数进行处理时，数的大小是有限制的。有些微型机不接收小于－32768或大于+32767的数。

测试程序

```
10 REM 'INT' TEST PROGRAM
20 READ X
30 PRINT "THE INTEGER VALUE OF"; X;
40 X = INT(X)
50 PRINT "IS"; X
60 IF X=999 THEN 999
70 GOTO 20
80 DATA 3.33,2.864,.35,-3.15,32766.853,-32766.853,999.99
999 END
```

运行实例

```
THE INTEGER VALUE OF 3.33 IS 3
THE INTEGER VALUE OF 2.864 IS 2
THE INTEGER VALUE OF .35 IS 0
THE INTEGER VALUE OF -3.15 IS -4
THE INTEGER VALUE OF 32766.853 IS 32766
THE INTEGER VALUE OF -32766.853 IS -32767
THE INTEGER VALUE OF 999.99 IS 999
```

替代的形式

大多数Ting BASIC把I. 作为INT函数的缩写。

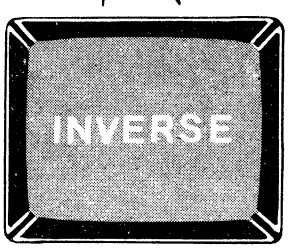
其它用法

除了指出的限制之外，没有其它用法。

参阅

CINT, %, FIX

语 句
命 令



APPLEⅡ把INVERSE作为一个命令或语句，用来将屏幕输出按相反（INVERSE）的方式显示。在INVERSE方式中，计算机的输出是以黑字符显示在白色的背景上。为了使显示恢复到它的正常方式，要打入NORMAL。

测试程序

```
10 REM INVERSE TEST PROGRAM
20 PRINT "THIS DEMONSTRATES ";
30 INVERSE
40 PRINT "INVERSE PRINTING."
99 END
```

为了运行这个程序，要清除屏幕并打入RUN。

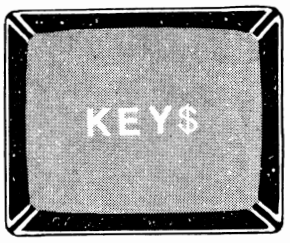
运行实例

```
THIS DEMONSTRATES INVERSE PRINTING.
```

参阅

NORMAL, FLASH

函　数



KEY

KEY$函数用于APF虚拟机器BASIC中，来输入一个字符,它不显示在屏幕上。KEY$（n）只接收三个不同的n值中的一个。

KEY$（0）读键盘，KEY$（1）读右方的游戏控制器，而KEY$（2）读左方的游戏控制器。

例如：A$ =KEY$（0）扫描键盘寻找任一个被按的键。如果没有键被按，那么A$ =“ ”(空)，否则A$“读”被打入的那个字符。KEY$（0)等效于INKEY$。（详见INKEY$。）

测试程序#1

```
10 REM KEY$(0) TEST PROGRAM
20 CALL 17046 'CLEARS THE SCREEN IN APF BASIC
30 PRINT "TYPE ANY CHARACTER"
40 A$ = KEY$(0)
50 IF A$="" THEN 40
60 PRINT "YOU JUST TYPED A ";A$
70 PRINT "TO REPEAT, TYPE ";A$;" AGAIN."
80 IF KEY(0)=A$ THEN 20
90 GOTO 80
99 END
```

运行实例（输入#）

```
TYPE ANY CHARACTER
YOU JUST TYPED A #
TO REPEAT, TYPE # AGAIN.
```

R$ =KEY$（1）检查右方的游戏控制器。如果没有键被按的话，R$ =“ ”。如果0～9的数字键中有一个被按，那么R$就变成那个被按键的数字字符。

当按CL键时，一个“？”就被存放在R$中。当按EN或FIRE键时,R$ =KEY$（1）将把R$置成“！”。

最后，R$接收N（北），S（南），E（东）或W（西），以指出哪一个方向的按键被按下。L$ =KEY$（2）以类似的方法读左方的游戏控制器。

测试程序#2

```
10 REM KEY$(1 & 2) TEST PROGRAM
20 R$ = KEY$(1)
30 IF R$="" THEN 20
40 L$ = KEY$(2)
50 IF L$="" THEN 40
60 PRINT "THE RIGHT HAND CONTROL GENERATED ";R$
70 PRINT "THE LEFT HAND CONTROL GENERATED ";L$
99 END
```

运行实例（典型的）

```
THE RIGHT HAND CONTROL GENERATED !
THE LEFT HAND CONTROL GENERATED N
```

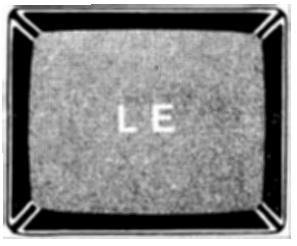
替代的形式

Texas Instruments′ TI 99/4把KEY用在CALL 语句中(即CALL KEY)来完成与KEY$（0）相同的工作。

参阅

INKEY$, PDL, GET, INPUT

操作符



某些计算机（例如：TI990）把LE操作符作为“小于或等于”符号（＜＝）的替换字。

详见＜＝操作符。

测试程序

```
10 REM 'LE (LESS THAN OR EQUAL TO)' TEST PROGRAM
20 IF 10 LE 20 THEN 50
30 PRINT "THE LE OPERATOR FAILED THE TEST IN LINE 20"
40 GOTO 99
50 IF 20 LE 20 THEN 80
60 PRINT "THE LE OPERATOR FAILED THE TEST IN LINE 50"
70 GOTO 99
80 PRINT "THE LE OPERATOR PASSED THE TEST"
99 END
```

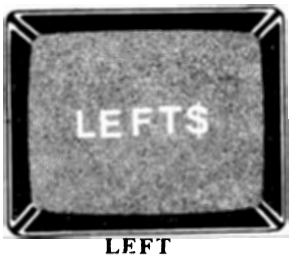
运行实例

```
THE LE OPERATOR PASSED THE TEST
```

参阅

＜＝，＜，＞＝，＞，＝，＜＞，EQ，GE，GT，LT，NE，IF-THEN

函　数



LEFT

LEFT$（串，n）函数用来把从字符串最左边开始的指定数目（n）的字符提取出来。

例如：PRINT LEFT$（“RUNNING”，3）打印字母RUN，RUN是字符串RUNNING中最左边的三个字符。

必须把字符串放在引号中或列出一个字符串变量。字符的个数（n）可以表达成一个变量、数或算术运算式。必须用逗号把字符串与数隔开。

如果（n）的值是十进制小数，那么计算机自动地求出其整数值。

测试程序

```
10 REM 'LEFT$' TEST PROGRAM
20 A$="THEATER"
30 B$=LEFT$("TESTING",4)
40 PRINT LEFT$(A$ ,3);"'LEFT$' FUNCTION PASSED THE";B$
99 END
```

运行实例

```
THE 'LEFT$' FUNCTION PASSED THE TEST
```

替代的形式

某些BASIC（例如：MAX BASIC和DEC BASIC—PLUS）用LEFT（串，n）来替代LEFT$。

参阅

```
PRINT, RIGHT$, MID$, CHR$, SPACE$, STR$, STRING$,
INKEY$, INSTR, POS, SEG$, DEFSTR
```

LEN函数是用来测量字符串长度的，它统计放在引号中的或分配给字符串变量的字符的个数。

例如：10 PRINT LEN（“TEST”）应该打印4，“TEST”这个词中的字母数目为4。

函 数

测试程序

```
10 REM 'LEN' TEST PROGRAM
20 PRINT "TYPE ANY COMBINATION OF LETTERS AND NUMBERS"
30 INPUT A$
40 PRINT "YOU ENTERED ";A$" WHICH CONTAINS";
50 PRINT LEN(A$); "CHARACTERS"
99 END
```

运行实例（输入ABC 1 2 3）

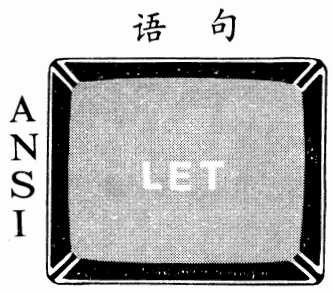
```
TYPE ANY COMBINATION OF LETTERS AND NUMBERS
? ABC123
YOU ENTERED ABC123 WHICH CONTAINS 6 CHARACTERS
```

替代的形式

British ACORN ATOM计算机把L.作为LEN的一个缩写。

参阅

```
ASC, FRE, LEFT$, MID$, RIGHT$, STR$, VAL, SEG$
```



LET语句用来给变量赋值（例如：LET X=20）。有些计算机要求写“LET”，而大多数计算机把“LET”作为可选的。当不要求LET时，有时就把它作为“标志”变量的一种方法，这些变量是要求被赋予新值的，或者该处要求专门的标志。

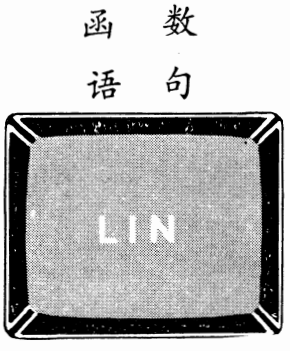
测试程序

```
10 REM 'LET' TEST PROGRAM
20 LET X=20
30 PRINT "THE LET STATEMENT PASSED THE TEST IN LINE";X
99 END
```

运行实例

```
THE LET STATEMENT PASSED THE TEST IN LINE 20
```

为了确定你的计算机是否要求写LET，可以从20行删去LET这三个字母，然后再试一下。



LIN（n）语句（用在Hewlett-Packard 2000的BASIC和Digital Group Opus 1和Opus 2的BASIC中）在打印下一行之前在打印机或CRT（屏幕）上跳过指定的行数（n）。

测试程序

```
10 REM 'LIN' TEST PROGRAM
20 PRINT "THE LIN STATEMENT PASSED THE TEST"
30 LIN(5)
40 PRINT "IF 5 LINES ARE SKIPPED BEFORE THIS LINE IS
   PRINTED"
99 END
```

运行实例

```
THE LIN STATEMENT PASSED THE TEST


IF 5 LINES ARE SKIPPED BEFORE THIS LINE IS PRINTED
```

有些计算机把LIN当作PRINT语句中的一个函数。如果上面测试程序不能运行，试用：30 PRINT LIN（5）

如果你的计算机没有此功能

如果你的计算机没有LIN（n），可以很容易地用若干（n）个PRINT语句来代替LIN（n）。

例如：用下列行代替测试程序中的30行：

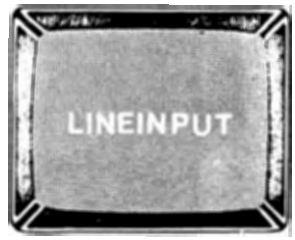
```
28 FOR X=1 TO 5
30 PRINT
32 NEXT X
```

因为每个PRINT语句产生一个换行，所以这些行将使计算机执行与LIN（5）相同的操作。

参阅

PRINT，VTAB

语　句



LINPUT

LINEINPUT语句类似于INPUT语句。LINEINPUT语句可接收一整行（LINE）的输入，一行输入最多可达254个字符，把它们赋给一个字符串变量。

LINEINPUT语句不给出“提醒”的符号（例如一个问号）。“输入”的字符串可以包括逗号、冒号和它其的一些专用字符，这些字符不必放在引号中。除了回车以外的任何字符都可被接收。

测试程序

```
10 REM LINEINPUT TEST PROGRAM
20 PRINT "TYPE YOUR NAME - LAST, FIRST";
30 LINEINPUT N$
40 PRINT "HELLO, ";N$;". I'M -80, TRS."
99 END
```

运行实例

```
TYPE YOUR NAME - LAST, FIRST DOE, JOHN
HELLO, DOE, JOHN. I'M -80, TRS.
```

LINEINPUT语句可以打印它自己的“提醒”信息。删去20行,再把30行变成下列形式:

```
30 LINEINPUT "TYPE YOUR NAME - LAST, FIRST ";N$
```

结果应该与前面的相同。

替代的形式

一些计算机可以用短的形式LIN　PUT 代替LINEINPUT 。

参阅

INPUTLINE, INPUT

命　令

语　句

LIS
LI
L.

LIST 命令用来按行号顺序显示每一个程序行，这些行是以此顺序出现在程序中的。一些计算机（或终端)以“滚动”的方式输出程序清单（即一行接一行地往上“顶”），除非用一个指定的功能键（Control C, Control S, SHIFT@,等等)使之停止。其它一些计算机则在显示前面12行,16行,24行或更多行以后暂停止,然后,按一次“上箭头”,“下箭头”或其它适当的键时,前进一行或几行后继续列表。

LIST 命令也可以与行号配合使用，以指定列清单的起始点（可以不是程序的开头）。很多计算机还可以接受一个起始和结束的行号。例如：LIST 10-40或LIST 10-40仅仅列出行号为10到40的行。

测试程序

```
10 REM 'LIST' COMMAND TEST PROGRAM
20 REM THIS COMMAND
30 REM WILL DISPLAY EACH LINE
40 REM AS HELD BY THE COMPUTER
50 PRINT "LIST TEST COMPLETE"
99 END
```

运行实例

打入LIST 20 - 30, 然后你的计算机应该打印:

```
20 REM THIS COMMAND
30 REM WILL DISPLAY EACH LINE
```

如果你的计算机不接受行号范围，试输入LIST 20。

应该打印出标号为20的行。如果这个测试不成功，试输入没有行号的LIST。

当输入LIST 20时，有些计算机将显示20行和20行以后的所有各行。如果你的计算机是这样的话，那么使用LIST 20－20或LIST 20，20以便把20行单独打印出来。

试用下面的列表命令：

```
LIST-
LIST30-
LIST-30
```

如果你的计算机接受这些LIST命令，那么LIST －应该把整个的程序都列表。LIST 30－表示从程序30行开始列表，而LIST － 30表示从程序第一行开始到30行结束。如果这些命令都不能使用的话，试用逗号（，）来代替连接符（－）。

一些计算机把LIST命令作为一个程序语句。为了在你的计算机上测试这个语句，把下列一行加到测试程序中：

60 LIST

如果LIST是作为一个程序语句接收的话，那么应打印：

```
LIST TEST COMPLETE
10 REM 'LIST' COMMAND TEST PROGRAM
20 REM THIS COMMAND
30 REM WILL DISPLAY EACH LINE
40 REM AS HELD BY THE COMPUTER
50 PRINT "LIST TEST COMPLETE"
60 LIST
99 END
```

替代的形式

LIST可以使用几种缩写形式：其中有LIS（PDP－8E），LI(Texas Instruments 990）和L.（Tiny BASIC）。

参阅

LLIST

命 令
语 句

Microsoft BASIC用LLIST命令来把当前内存中的程序在打印机上打印列表，而不是在屏幕上列表。经常把LLIST作为一个命令来使用，但是也可以把LLIST作为程序中一个语句来使用。

适用于LIST的所有功能都可以用在LLIST中。因为是在纸上列表而不是在屏幕上列表，所以不需要在打印12、16或24行后暂停（象LIST那样），而是一直继续进行到程序的末尾或者按下BREAK，RESET、Control C等键为止。

注意：在你打入LLIST 命令之前，必须把打印机连接到计算机上，而且要为输出做好准备,否则计算机可能要被“挂起”。在TRS—80Ⅲ型机上，按BREAK 键或类似的键，可以使计算机脱离“挂起”状态。

测试程序

```
10 REM LLIST COMMAND TEST PROGRAM
20 REM THE LLIST COMMAND
30 REM WILL PRINT THIS PROGRAM
40 REM ON THE PRINTER
99 END
```

运行实例

打入LLIST 命令，在你的打印机上应该产生上述的列表。打印LLIST 20－30应该打印出：

```
20 REM THE LLIST COMMAND
30 REM WILL PRINT THIS PROGRAM
```

打入LLIST —20应该只打印前面两行而LLIST 40—应该只打印最后两行。

为了检查LLIST 是否能作为一个程序语句来使用，在测试程序中加上以下一行：

```
50 LLIST
```

然后打入RUN。如果机器接收了LLIST，则打印机应该打印：

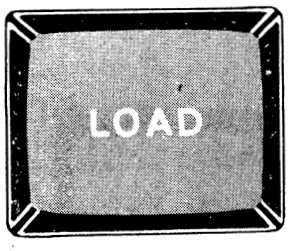
```
10 REM LLIST COMMAND TEST PROGRAM
20 REM THE LLIST COMMAND
30 REM WILL PRINT THIS PROGRAM
40 REM ON THE PRINTER
50 LLIST
99 END
```

参阅

LIST，LPRINT

命　令

LOAD命令用来把一个程序从盒式磁带或磁盘上装入计算机中。



测试程序

把这段程序从键盘上输入到计算机中，然后再存放到盒式磁带中（详见SAVE）。

```
10 REM 'LOAD' TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE LOAD FEATURE"
99 END
```

一旦把测试程序记录在盒式磁带上后，用NEW,SCRATCH或任何适当的命令来清除计算机的内存。

倒回盒式磁带，并把录音机置于Play状态，然后再打入LOAD命令。

计算机控制盒式录音机的马达，在LOAD周期之前把录音机的马达接通,而在LOAD周期之后把录音机的马达断开。盒式磁带应该“重现”测试程序，并把该程序装入计算机。

把测试程序列表，以便检验存放在计算机内存中的测试程序与原来输入的测试程序是否相同（参看LIST）。如果一切都正常，就运行该程序。

如果需要，在把程序存入（SAVE）磁带时，可以给这个程序起个名字。这样，在LOAD时，其它的一些程序都被忽略，而当发现以此名字命名的程序时，就把指定的这个程序装入计算机。例如：LOAD“TEST”。

运行实例

```
THIS PROGRAM TESTS THE LOAD FEATURE
```

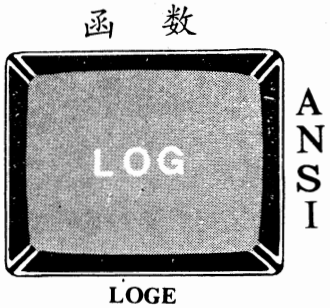
其它用法

LOAD“文件名”一般用来装入一个先前已存放在磁盘上的程序。需要一个文件名。

参阅

CLOAD，SAVE，CSAVE，LIST，NEW

函数LOG（n）可以计算任一个大于零的数（n）的自然对数。对于常用对数（以10为底）参看LOG 10或CLG。



测试程序

```
10 REM 'LOG' TEST PROGRAM
20 PRINT "ENTER A POSITIVE NUMBER";
30 INPUT N
40 L=LOG(N)
50 PRINT "THE NATURAL LOG OF";N;"IS";
30999 END
```

运行实例（输入100）

```
ENTER A POSITIVE NUMBER? 100
THE NATURAL LOG OF 100 IS 4.60517
```

替代的形式

有些BASIC 使用LN（例如：Micropolis BASIC），另外有些BASIC 使用LOGE，还有少数（例如：DEC —10）可以使用自然对数的三种形式（即LN, LOGE, LOG）

如果你的计算机没有此功能

如果自然对数的三种形式在你的计算机上都不能使用的话，你可以用下列子程序来代替：

```
30000 GOTO 30999
30150 REM * NATURAL LOGARITHM SUBROUTINE * INPUT X,
      OUTPUT L
30152 REM USES A, B, C, D AND E INTERNALLY
30154 IF X>0 THEN 30160
30156 PRINT "LOG UNDEFINED AT";X
30158 STOP
30160 A=1
30162 B=2
30164 C=.5
30166 D=X
30168 E=0
30170 IF X<A THEN 30178
30172 X=C*X
30174 E=E+A
30176 GOTO 30170
30178 IF X>=C THEN 30186
30180 X=B*X
30182 E=E-A
30184 GOTO 30178
30186 X=(X-.707107)/(X+.707107)
30188 L=X*X
30190 L=(((.598979*L+.961471)*L+2.88539)*X+E-.5)*.693147
30192 IF L>1E-6 THEN 30196

30194 L=0
30196 X=D
30198 RETURN
```

为了把这段子程序用在测试程序中，对程序修改如下：

```
35 X=N
40 GOSUB 30150
```

转换系数

为了把自然对数转换成常用对数，把自然对数乘以·4342945 即可。例如：

X = LOG（N） *.4342945

为了把常用对数转换成自然对数，把常用对数乘以2.302585。

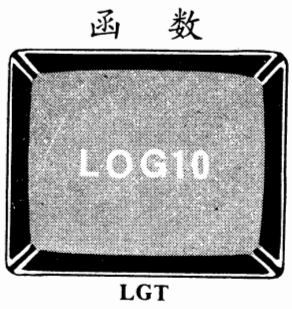
其它用法

少数计算机（例如：IMSAI 4K）使用LOG来计算常用对数，而不是自然对数（但这是极少见的例外）。

参阅

LOG10，CLG

LOG10（n）函数计算任一个数（n）的常用对数（以10为底）值，数（n）的值大于零。



LGT

测试程序

```
10 REM 'LOG10' TEST PROGRAM
20 PRINT "ENTER A POSITIVE NUMBER";
30 INPUT N
40 L=LOG10(N)
50 PRINT "THE COMMON LOG OF";N;"IS";L
30999 END
```

运行实例（输入100）

```
ENTER A POSITIVE NUMBER? 100
THE COMMON LOG OF 100 IS 2
```

替代的形式

有的BASIC使用其它几个字。有些BASIC使用LGT（例如Tektronix 4050系列的BASIC和MAX BASIC）。其它一些BASIC使用CLG和CLOG。

如果你的计算机没有此功能

如果你的计算机运行测试程序不成功，试用LOG下面的测试程序（见133页）。如果可以接受LOG，那么加上下面这个语句行，就可以用LOG来计算常用对数的值：

```
40 L=LOG(N)*.4342945
```

如果LOG的测试程序也不成功，用LOG下面的子程序（见134页）来代替，并作下列改变：

```
30150 REM * COMMON LOGARITHM SUBROUTINE * INPUT X,
      OUTPUT L
30197 L=L*.4342945
```

测试程序变化如下：

```
35 X=N
40 GOSUB 30150
```

转换系数

为了把常用对数转换成自然对数，只要把常用对数的值乘以2.302585即可。

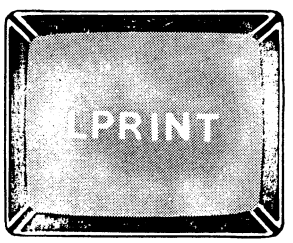
例如X = LOG 10（N）＊2.302585

为了把自然对数转换成常用对数，只要把自然对数的值乘以.4342945 即可。

参阅

CLG；LOG

命 令
话 句



Microsoft BASIC 用LPRINT 来把一个PRINT 语句传送到打印机，而不是送到显示屏幕上。LPRINT可用作直接命令或程序语句。

PRINT 语句中的所有可选项除了@之外的都适用于LPRINT。包括T A B，带逗号的“打印区域”，带分号的紧凑打印和LPRINT USING。详见PRINT 和PRINT USING。

注意：在使用LPRINT 运行一个程序之前，必须把打印机联接到计算机上，并为输出做好准备。否则，计算机就要被“挂起来”，程序就会丢失。Microsoft BASIC 的最新版本（例如：TRS—80Ⅲ型）用简单地按一下BREAK键的方法就可以脱离“挂起”状态。

测试程序

```
10 REM LPRINT TEST PROGRAM
20 LPRINT "YOU HAVE ONE FINE PRINTER HERE!"
30 LPRINT
40 A = 30
50 LPRINT TAB(A); A; A+4; A+8
60 LPRINT "JUST MEASURING THE LENGTH OF A LINE."
70 LPRINT 1,2,3
80 LPRINT " AND CHECKING ZONE PRINTING."
90 LPRINT "IT ALL LOOKS GOOD TO ME."
100 LPRINT USING "I'LL GIVE YOU $#.## FOR IT.",8/3
999 END
```

运行实例

```
YOU HAVE ONE FINE PRINTER HERE!
                              30  34  38
JUST MEASURING THE LENGTH OF A LINE.
 1                2                3
AND CHECKING ZONE PRINTING.
IT ALL LOOKS GOOD TO ME.
I'LL GIVE YOU $2.67 FOR IT.
```

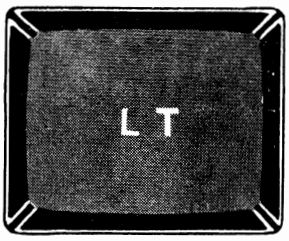
参阅

PRINT，PRINT USING，TAB，，（逗号），.；（分号），LLIST

在一些计算机中（例如：TI990）把L T操作符作为“小于”号（<）的替换符号。

详见<。

操作符



测试程序

```
10 REM  LT(LESS-THAN) TEST PROGRAM
20 IF 5 LT 10 THEN 50
30 PRINT "THE LT OPERATOR FAILED THE TEST"
40 GOTO 99
50 PRINT "THE LT OPERATOR PASSED THE TEST"
99 END
```

运行实例

```
THE LT OPERATOR PASSED THE TEST
```

参阅

<，=，>，>=，=，<>，EQ，GE，GT，LE，NE，IF-THEN

命 令

APPLE Ⅱ BASIC，用MAN命令进行人工插入程序行号。

如果计算机是处在自动行号方式，那么计算机在接收MAN 命 令 之前必须先打入Control X。

测试过程

为了测试计算机的MAN特性，打入AUTO命令并按RETURN键。使计算机处于自动编行号方式，如果打印出行号10，那么计算机就已成功地进入自动方式。现 打 入Control X和MAN命令。输入一些测试程序行来验证计算机是否通过了MAN命令的测试。

参 阅

AUTO

语 句

一些计算机用MAT CON语句来把一个数组中的每个元素都置成某个常数(CONstant)——一般地是置成1。

也可以用CON重新定义数组，在重新定义的数 组中提供的元素数目要小于或等于由DIM语句原来保留的单元数目。

例如，如果已把数组A 定义为一个 3 × 5 的数组，那么MAT A=CON在 3 行 (每行五列) 里存放15个 1 而MAT A=CON(2,6)在 2 行(每行 6 列) 里 存放12个 1 。

测试程序 #1

```
10 REM * MAT CON * TEST PROGRAM
20 DIM A(3,5)
30 MAT A=CON
40 FOR I=1 TO 3
50   FOR J=1 TO 5
60     PRINT A(I,J);
70   NEXT J
80   PRINT
90 NEXT I
100 PRINT " 'MAT CON' PASSED THE TEST IF A"
110 PRINT "3X5 ARRAY OF ONES WAS PRINTED."
999 END
```

运行实例

```
 1  1  1  1  1
 1  1  1  1  1
 1  1  1  1  1
'MAT CON' PASSED THE TEST IF A
3X5 ARRAY OF ONES WAS PRINTED.
```

测试程序=2

```
10 REM * MAT CON * REDIMENSION TEST PROGRAM
20 DIM A(3,5)
30 MAT A=CON(2,6)
40 FOR I=1 TO 2
50   FOR J=1 TO 6
60     PRINT A(I,J);
70   NEXT J
80   PRINT
90 NEXT I
100 PRINT "'MAT CON' PASSED THE REDIMENSION TEST IF A"
110 PRINT "2X6 ARRAY OF ONES WAS PRINTED"
999 END
```

运行实例

```
 1  1  1  1  1  1
 1  1  1  1  1  1
'MAT CON' PASSED THE REDIMENSION TEST IF A
2X6 ARRAY OF ONES WAS PRINTED
```

如果你的计算机没有此功能

如果你的计算机不接受MAT CON，那么可以用FOR—NEXT循环来代替30行：

```
22 FOR I=1 TO 3
26   FOR J=1 TO 5
30     A(I,J)=1
34   NEXT J
38 NEXT I
```

参 阅

MAT ZER，MAT IDN，DIM，FOR，NEXT

语 句

MAT IDN语句用在方阵上（每维元素个数相同的二维数组）来形成单位矩阵，这个矩阵的主对角线元素上为1，而其它元素为0。

例如：如果把矩阵A定义为4×4的数组，那么MAT A=IDN产生下列矩阵：

```
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
```

测试程序＝1

```
10 REM 'MAT IDN' TEST PROGRAM
20 DIM A(4,4)
30 MAT A= IDN
40 FOR I=1 TO 4
50  FOR J=1 TO 4
60   PRINT A(I,J)
70  NEXT I
80  PRINT
90 NEXT J
100 PRINT "'MAT IDN' PASSED THE TEST IF A 4X4 IDENTITY"
110 PRINT "MATRIX WAS PRINTED"
999 END
```

运行实例

```
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
'MAT IDN' PASSED THE TEST IF A 4X4 IDENTITY
MATRIX WAS PRINTED
```

也可以用IDN来重新定矩阵的大小，在重定大小的矩阵中提供的元素数目要小于或等于由DIM语句原来保留的元素数目。

例如：如果A是一个 2 × 5 的数组，那么，MAT A = IDN（3，3）就重新定义了一个 3 × 3 的单位矩阵。

测试程序＝2

```
10 REM 'MAT IDN' REDIMENSION TEST PROGRAM
20 DIM A(2,5)
30 MAT A=IDN(3,3)
40 FOR I=1 TO 3
50  FOR J=1 TO 3
60   PRINT A(I,J);
70  NEXT J
80  PRINT
90 NEXT I
100 PRINT "'MAT IDN' PASSED THE REDIMENSION TEST IF A
    3X3 IDENTITY"
110 PRINT "MATRIX WAS PRINTED"
999 END
```

运行实例

```
 1  0  0
 0  1  0
 0  0  1
'MAT IDN' PASSED THE REDIMENSION TEST IF A 3X3 IDENTITY
MATRIX WAS PRINTED
```

如果你的计算机没有此功能

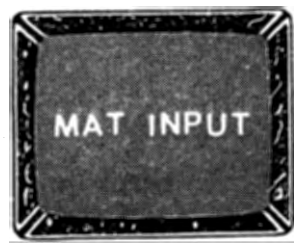
如果你的计算机不能通过IDN测试，那么可以将下面这些行插入测试程序中。

```
26 FOR I=1 TO 4
28  FOR J=1 TO 4
30   A(I,J)=0
32  NEXT J
34  A(I,I)=1
36 NEXT I
```

参 阅

DIM，FOR，NEXT 及MAT的其他语句。

语 句



用MAT INPUT语句可以由键盘给一个数组中的每个元素赋值。由DIM语句确定被赋值的数组元素的数目。

例如：

```
10 DIM A(5)
20 MAT INPUT A
```

DIM语句允许数组变量A使用被命名为A(0)到A(5)的数组元素。(更详细的资料可参阅 DIM 。)MAT INPUT给A(1)到A(5)的元素赋值。

当执行MAT INPUT语句的时候，计算机打印一个问号（?）表示计算机准备好接收数组中第一个元素的值。如果输入的数据都在一行上，则在打入每个值之后必须打入一个逗号，最后按RETURN或ENTER键。如果输入的一行中提供的值不足以使数组中每个元素都能接收到一个值，那么计算机就会打印一个双问号（? ?）表示还需要输入其它的值。正如原先介绍的INPUT语句的情况一样，可以每一次输入一个值然后按RETURN。

MAT INPUT语句先给二维数组的第一行中的每列元素赋值，然后再给第二行的各列元素赋值……。

例如：

```
10 DIM A(2,3)
20 MAT INPUT A
```

计算机在给A（2，1），A（2，2），A（2，3）赋值之前先给数组变量元素A（1，1），A（1，2），A（1，3）赋值。

如果使用的数组元素不多于10的话，那么大多数具有MAT INPUT功能的计算机可以用MAT INPUT语句来定义数组的大小。例如：MAT INPUT A(2,3)。如果一个数组需要的元素多于10个，那么它就必须用DIM语句定义。

例如：

```
10 DIM B(20,20)
20 MAT INPUT A(3,5)
30 MAT INPUT B(15,11)
```

测试程序

```
10 REM 'MAT INPUT' TEST PROGRAM
20 DIM A(3,4)
30 PRINT "ENTER 12 NUMBERS (SEPARATED BY COMMAS)"
40 MAT INPUT A
50 FOR I=1 TO 3
60  FOR J=1 TO 4
70   PRINT A(I,J);
80  NEXT J
90  PRINT
100 NEXT I
110 PRINT "THE MAT INPUT STATEMENT PASSED THE TEST"
120 PRINT "IF THE INPUT VALUES ARE PRINTED IN A"
130 PRINT "MATRIX HAVING THREE ROWS OF FOUR COLUMNS."
999 END
```

运行实例（典型的）

```
ENTER 12 NUMBERS (SEPARATED BY COMMAS)
?1,2,3,4,5,6,7,8,9,10,11,12
 1   2   3   4
 5   6   7   8
 9   10   11   12
THE MAT INPUT STATEMENT PASSED THE TEST
IF THE INPUT VALUES ARE PRINTED IN A
MATRIX HAVING THREE ROWS OF FOUR COLUMNS.
```

如果你的计算机没有此功能

如果你的计算机没有MAT INPUT功能，那么可以用FOR—NEXT和INPUT语句来代替MAT INPUT。

把下面的几行代入测试程序：

```
33 FOR I=1 TO 3
36  FOR J=1 TO 4
40   INPUT A(I,J)
43  NEXT J
46 NEXT I
```

这个替换与MAT INPUT语句稍有区别：在MAT INPUT语句中，在打入每个值之后不需要按RETURN或ENTER键。在本替换中每输入一个值就要按一个RETURN或ENTER键。

参 阅

MAT PRINT, MAT READ, FOR, INPUT, DIM, NUM

语 句

MAT INV用在一个方阵中(每维元素数相同的二维数组)来产生一个矩阵,这个矩阵是原来矩阵的逆矩阵。

不是所有的矩阵都有它的逆矩阵的。如果A是一个有逆矩阵的矩阵的话,当执行MAT B = INV(A)时,B也是一个矩阵;若把B矩阵乘以A矩阵,结果应该为一单位矩阵(参看MAT IDN。)

有些计算机计算逆矩阵和计算矩阵的行列式是一起使用的(参看DET)。这时,可以由测试DET来检查是否存在一个真正的逆矩阵。如果DET = 0,那么矩阵就没有逆矩阵,而B中的值是无效的。

测试程序

```
10 REM * MAT INV * TEST PROGRAM
20 DIM A(3,3), B(3,3)
30 FOR I=1 TO 3
40   FOR J=1 TO 3
50     READ A(I,J)
60   NEXT J
70 NEXT I
80 MAT B=INV(A)
90 FOR I=1 TO 3
100   FOR J=1 TO 3
110     PRINT B(I,J);
120   NEXT J
130   PRINT
140 NEXT I
150 PRINT "'MAT INV' PASSED THE TEST IF 3 -.5 -.5"
160 PRINT "                              -1   0   1"
170 PRINT "                              -1   .5 -.5 WAS
    PRINTED"
180 DATA 1,1,1,  3,4,5,  1,2,1
30999 END
```

运行实例

```
 3 -.5 -.5
-1   0   1
-1   .5 -.5
'MAT INV' PASSED THE TEST IF 3 -.5 -.5
                             -1   0   1
                             -1   .5 -.5 WAS PRINTED
```

如果你的计算机没有此功能

如果MAT INV不适用于你的计算机,可用下面的子程序来代替:

```
30000 GOTO 30999
30850 REM * MAT INV SUBROUTINE * INPUT N, A( , )  OUTPUT B( , )
30852 REM  ALSO USES 1, J, K, P AND R INTERNALLY
30853 REM DIMENSION ARRAY J HERE IF N>10
30854 FOR I=1 TO N
30856  FOR J=1 TO N
30858   B(I,J)=A(I,J)
30860  NEXT J
30862 NEXT I
30864 M=N-1
30866 FOR K=1 TO N
30867  J(K)=0
30868  P=B(K,1)
30870  IF P<>0 THEN 30894
30872  FOR I=K+1 TO N
30873   J(K)=I
30874   IF B(I,1)=0 THEN 30888
30876   FOR J=1 TO N
30878    R=B(K,J)
30880    B(K,J)=B(I,J)
30882    B(I,J)=R
30884   NEXT J
30886   GOTO 30868
30888  NEXT I
30890  PRINT"*** NO INVERSE EXISTS ***"
30892  GOTO 30999
30894  FOR J=1 TO M
30896   B(K,J)=B(K,J+1)/P
30898  NEXT J
30900  B(K,N)=1/P
30902  FOR I=1 TO N
30904   IF I=K THEN 30916
30906   R=B(I,1)
30908   FOR J=1 TO M
30910    B(I,J)=B(I,J+1)-R*B(K,J)
30912   NEXT J
30914   B(I,N)=-R*B(K,N)
30916  NEXT I
30918 NEXT K
30920 FOR K=M TO 1 STEP -1
30922  J=J(K)
30924  IF J=0 THEN 30936
30926  FOR I=1 TO N
30928   R=B(I,K)
30930   B(I,K)=B(I,J)
30932   B(I,J)=R
30934  NEXT I
30936 NEXT K
30938 RETURN
30999 END
```

为了把这个子程序与测试程序一起使用，将测试程序做如下的变化：

```
75 N=3
80 GOSUB 30850
```

参 阅

MAT IDN, MAT *, DET, FOR , NEXT, DIM, MAT READ, MAT PRINT

语　句

用MAT PRINT语句来打印存放在指定的数组元素中的值。要打印的元素的数目是由DIM语句来确定的。详见DIM语句。

例如：

```
10 DIM A(3)
20 MAT PRINT A
```

打印赋给“A”数组的三个值，这三个值是A（1）到A（3）。

测试程序＝1

```
10 REM 'MAT PRINT' TEST PROGRAM
20 DIM A(5)
30 FOR X=1 TO 5
40 A(X)=X
50 NEXT X
60 MAT PRINT A
70 PRINT "END OF MAT PRINT TEST"
99 END
```

运行实例

```
 1
 2
 3
 4
 5
END OF MAT PRINT TEST
```

大多数具有MAT PRINT语句功能的计算机允许在MAT PRINT语句后跟一个逗号，这样可以使打印的数组值位于确定的水平打印区中（参看逗号）。为了检验测试程序中的这个特性，把60行变成如下形式：

60　MAT PRINT A,

然后运行。

运行实例（80列的屏幕）

```
 1             2             3             4             5
END OF MAT PRINT TEST
```

可以在MAT PRINT语句后跟一个分号，使打印的数组值按紧凑格式（即只用一个或两个空格来分隔这些值），（参看分号）。为了检测这一特性，把60行变成下列形式。

60　MAT PRINT A;

然后运行。

运行实例

```
 1   2   3   4   5
END OF MAT PRINT TEST
```

MAT PRINT语句可以打印一维以上的数组的内容。第一维中的元素数确定要打印的行数，而第二维中的元素数确定列数。

例如：

```
DIM A(2,3)
MAT PRINT A
```

DIM语句定义A为两维数组，它是由用MAT PRINT语句打印出具有2行和3列的矩阵。

可以在一个MAT PRINT语句中依次列出需要打印的几个数组，当中以 逗号 或分号分隔。结果由下面的测试程序示出。

测试程序 = 2

```
10 REM 'MAT PRINT' WITH MULTIPLE ARRAY VARIABLES TEST
   PROGRAM
20 DIM A(3),B(3,5)
30  FOR I=1 TO 3
40   FOR J=1 TO 5
50    B(I,J)=J
60   NEXT J
70  A(I)=1
80  NEXT I
90 MAT PRINT A;B,
100 PRINT "END OF MAT PRINT TEST"
999 END
```

运行实例

```
 1   2   3
 1             2             3             4             5
 1             2             3             4             5
 1             2             3             4             5
END OF MAT PRINT TEST
```

一些计算机允许用MAT PRINT USING语句使数组按格式打印。详见PRINT USING。

如果你的计算机没有此功能

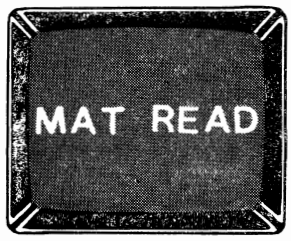
如果你的计算机没有MAT PRINT的功能，那么MAT PRINT语句可以用FOR—NEXT和PRINT语句来模拟。把下列行代入测试程序 = 2 中：

```
81    FOR X=1 TO 3
82      PRINT A(X);
84    NEXT X
86  PRINT
88    FOR I=1 TO 3
90      FOR J=1 TO 5
92        PRINT B(I,J),
94      NEXT J
95      PRINT
96    NEXT I
98  PRINT
```

参 阅

MAT INPUT, MAT READ, , (逗号), ; (分号), FOR, PRINT, DIM, PRINT USING

语 句



用MAT READ语句来读取DATA语句中的值,并把这些值赋给一个数组。DIM语句确定这个数组的大小。

例如:

```
10 DIM A(5)
20 MAT READ A
```

DIM语句允许数组A使用以A（0）到A（5）命名的六个数组元素。详见DIM 。MAT READ语句只将数据读入到A（1）到A（5）元素中。

MAT READ语句先对二维数组中第一行中的每一列元素赋值，然后再给第二行各列元素赋值。

例如:

```
10 DIM A(2,3)
20 MAT READ A
```

计算机从DATA 语句读入六个值，并把这六个值先赋给数组变量元素A（1，1），A（1，2）和A（1，3），然后再赋给A（2，1），A（2，2），A（2，3）。

测试程序

```
10 REM 'MAT READ' TEST PROGRAM
20 DIM A(3,4)
```

```
30 MAT READ A
40 FOR I=1 TO 3
50  FOR J=1 TO 4
60   PRINT A(I,J);
70  NEXT J
80  PRINT
90 NEXT I
100 DATA 1,2,3,4,5,6,7,8,9,10,11,12
110 PRINT "THE MAT READ STATEMENT PASSED THE TEST"
120 PRINT "IF A MATRIX IS PRINTED HAVING 3 ROWS OF
    4 COLUMNS"
999 END
```

运行实例

```
 1   2   3   4
 5   6   7   8
 9   10  11   12
THE MAT READ STATEMENT PASSED THE TEST
IF A MATRIX IS PRINTED HAVING 3 ROWS OF 4 COLUMNS
```

如果使用的数组元素不多于10的话，那么大多数具有MAT READ功能的计算机允许由MAT READ语句来确定数组大小。如果一个数组中所需要的元素大于10的话，那么必须用DIM语句定其大小。

例如：

```
110 DIM B(20,20)
120 MAT READ A(3,5)
130 MAT READ B(15,11)
```

为了检测你的计算机中的这一特性，将测试程序中的20行删去，并把30行变成下列形式：

30 MAT READ A(3, 4)

如果你的计算机接受这一特性，那么运行结果应不变化。

如果你的计算机没有此功能

如果你的计算机没有MAT READ功能，那么 MAT READ 语句可以 用 FOR—NEXT 和READ语句来代替。

把下面各行代入测试程序：

```
23 FOR I=1 TO 3
26  FOR J=1 TO 4
30   READ A(I,J)
33  NEXT J
36 NEXT I
```

参 阅

MAT PRINT, MAT INPUT, READ, DATA, DIM, FOR

语　句

用MAT TRN语句来产生一个转置矩阵（二维数组）。如果A是一个M×N的数组，B是一个N×M的数组，那么MAT B = TRN（A）用下面的方法产生矩阵B；矩阵A的第一行变成矩阵B的第一列，矩阵A的第二行变成矩阵B的第二列，依此类推。

测试程序

```
10 REM * MAT TRN * TEST PROGRAM
20 DIM A(3,5), B(5,3)
30 FOR I=1 TO 3
40  FOR J=1 TO 5
50   A(I,J)=10*I+J
60  NEXT J
70 NEXT I
80 MAT B=TRN(A)
90 FOR J=1 TO 5
100  FOR I=1 TO 3
110   PRINT B(J,I);
120  NEXT I
130  PRINT
140 NEXT J
150 PRINT "'MAT TRN' PASSED THE TEST IF THE NUMBERS 11-15,"
160 PRINT "21-25, AND 31-35 ARE PRINTED IN 3 COLUMNS."
999 END
```

运行实例

```
 11  21  31
 12  22  32
 13  23  33
 14  24  34
 15  25  35
'MAT TRN' PASSED THE TEST IF THE NUMBERS 11-15,
21-25, AND 31-35 ARE PRINTED IN 3 COLUMNS.
```

语句MAT A = TRN(A)是合法的，条件是仅当A的二维元素数相同(即A是一方阵)。

其它用法

有些计算机允许把MAT语句作为可选字。在那些程序中，80行可用下面一行来代替：

80 B = TRN（A）

如果你的计算机没有此功能

如果你的计算机不接受MAT TRN语句，你可以用下列行来产生一个M×N转置矩阵：

```
76 FOR I=1 TO M
78  FOR J=1 TO N
80   B(J,I)=A(I,J)
82  NEXT J
84 NEXT I
```

参 阅

DIM，FOR，NEXT ，及其它MAT语句。

语 句

一些计算机使用MAT ZER语句来把一个数组的每个元素置成0。也可以使用MAT ZER语句重新定义数组的大小，在重新定义的数组中提供的元素数目应小于或等于由DIM语句原来定义的元素数目。

例如，如果数组A已定义成3 × 5的数组，那么MAT A=ZER在3行（每行5列）中存放15个零，而MAT A=ZER（2，6）则在2行（每行6列）中存放12个零。

测试程序 # 1

```
10 REM 'MAT ZER' TEST PROGRAM
20 DIM A(3,5)
22 REM START WITH SOME NON-ZERO VALUES
24 FOR I=1 TO 3
26  A(I,I)=I
28 NEXT I
30 MAT A=ZER
40 FOR I=1 TO 3
50  FOR J=1 TO 5
60   PRINT A(I,J);
70  NEXT J
80  PRINT
90 NEXT I
100 PRINT "'MAT ZER' PASSED THE TEST IF A"
110 PRINT "3X5 ARRAY OF ZEROS WAS PRINTED."
999 END
```

运行实例

```
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0
'MAT ZER' PASSED THE TEST IF A
3X5 ARRAY OF ZEROS WAS PRINTED.
```

测试程序 # 2

```
10 REM 'MAT ZER' REDIMENSION TEST PROGRAM
20 DIM A(3,5)
30 MAT A=ZER(2,6)
40 FOR I=1 TO 2
50  FOR J=1 TO 6
60   PRINT A(I,J);
70  NEXT J
80  PRINT
90 NEXT I
100 PRINT "'MAT ZER' PASSED THE REDIMENSION TEST IF A"
110 PRINT "2X6 ARRAY OF ZEROS WAS PRINTED"
999 END
```

运行实例

```
 0  0  0  0  0  0
 0  0  0  0  0  0
'MAT ZER' PASSED THE REDIMENSION TEST IF A
2X6 ARRAY OF ZEROS WAS PRINTED
```

如果你的计算机没有此功能

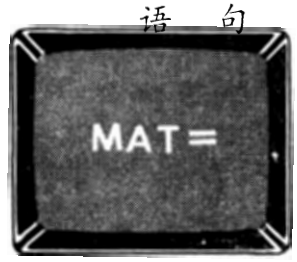
如果你的计算机不允许使用MAT ZER语句，可以用嵌套的FOR—NEXT 循环来代替30行：

```
22 FOR I=1 TO 3
26  FOR J=1 TO 5
30   A(I,J)=0
34  NEXT J
38 NEXT I
```

参　阅

MAT CON，MAT IDN，DIM，FOR，NEXT

语　句



MAT = 语句把存放在一个数组中的各个值赋给另一个数组的相对应的元素。例如：MAT　B = A把数组A的值“拷贝”到数组B中。大多数接受MAT语句的计算机需要把数组B的大小定得与数组A相同，这是由一个 DIM语句来完成的。

测试程序 # 2

```
10 REM * MAT ASSIGNMENT * TEST PROGRAM
20 DIM A(2,3), B(2,3)
30 FOR I=1 TO 2
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70 NEXT I
80 MAT B=A
90 FOR I=1 TO 2
100  FOR J=1 TO 3
110   IF B(I,J) <> A(I,J) THEN 160
120  NEXT J
130 NEXT I
140 PRINT "THE MAT ASSIGNMENT PASSED THE TEST"
150 GOTO 999
160 PRINT "THE MAT ASSIGNMENT FAILED THE TEST"
170 DATA 1, 2, 3, 4, 5, 6
999 END
```

运行实例

```
THE MAT ASSIGNMENT PASSED THE TEST
```

其它用法

有些计算机允许MAT为可选字。在那些程序中，80行可用下面的形式来代替：

```
80 B=A
```

如果你的计算机没有此功能

如果你的计算机通不过MAT赋值测试，你可用嵌套循环把一个数组的值“拷贝”到另一个数组中。用下面各行来代替测试程序中的80行：

```
76 FOR I=1 TO 2
78  FOR J=1 TO 3
80   B(I,J) = A(I,J)
82  NEXT J
84 NEXT I
```

参 阅

DIM, FOR, NEXT, READ, DATA, PRINT, MAT READ, MAT PRINT

语 句

使用MAT + 语句把大小相同的两个数组的相应的元素加起来，然后把结果存放在相同大小的第三个数组中。例如：MAT C = A + B使C (1,1)等于A (1,1) +B (1,1)，C (1,2) 等于A (1,2) +B (1,2)等等。

测试程序

```
10 REM 'MAT +' TEST PROGRAM
20 DIM A(2,3), B(2,3), C(2,3)
30 FOR I=1 TO 2
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70  FOR K=1 TO 3
80   READ B(I,J)
90  NEXT K
100 NEXT I
110 MAT C=A+B
120 FOR I=1 TO 2
130  FOR J=1 TO 3
140   PRINT C(I,J);
150  NEXT J
160  PRINT
170 NEXT I
180 PRINT "'MAT +' PASSED THE TEST IF  3  6  9"
190 PRINT "                              12  15  18
    WAS PRINTED"
200 DATA 2, 4, 6,  1, 2, 3
210 DATA 8,10,12,  4, 5, 6
999 END
```

运行实例

```
 3  6  9
 12  15  8
'MAT +' PASSED THE TEST IF  3  6  9
                            12  15  18  WAS PRINTED
```

其它用法

象在大多数计算机中LET字可省写一样，一些计算机也允许省写MAT。在这些程序中，110行可用110 C = A + B来代替，其结果相同。

如果你的计算机没有此功能

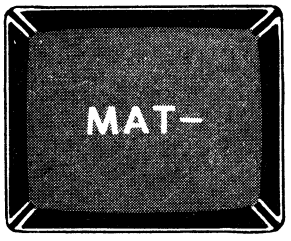
如果你的计算机通不过MAT + 语句的测试程序，可以用下面各行代替110行来实现矩阵加法：

```
106 FOR I=1 TO 2
108  FOR J=1 TO 3
110   C(I,J) = A(I,J) + B(I,J)
112  NEXT J
114 NEXT I
```

参 阅

MAT -, MAT *, DIM, FOR, NEXT, MAT READ, MAT PRINT

语　句



使用MAT－语句把大小相同的两个数组相减，然后把结果存放在相同大小的第三个数组中。例如：MAT C＝A－B使C（1，1）等于A（1,1）－B（1,1），C（1,2）等于A（1,2）－B（1,2）等等。

测试程序

```
10 REM 'MAT -' TEST PROGRAM
20 DIM A(2,3), B(2,3), C(2,3)
30 FOR I=1 TO 2
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70  FOR K=1 TO 3
80   READ B(I,J)
90  NEXT K
100 NEXT I
110 MAT C=A-B
120 FOR I=1 TO 2
130  FOR J=1 TO 3
140   PRINT C(I,J);
150  NEXT J
160  PRINT
170 NEXT I
180 PRINT "'MAT -' PASSED THE TEST IF  1  2  3"
190 PRINT "                            4  5  6
    WAS PRINTED"
200 DATA 2, 4, 6,  1, 2, 3
210 DATA 8,10,12,  4, 5, 6
999 END
```

运行实例

```
 1   2   3
 4   5   6
'MAT -' PASSED THE TEST IF  1  2  3
                            4  5  6  WAS PRINTED
```

其它用法

象在很多计算机中可以省写LET字一样，一些计算机允许省写MAT，在这些程序中，110行可用110 C＝A－B来代替，其结果相同。

如果你的计算机没有此功能

如果你的计算机通不过MAT－语句的测试程序，可用下面各行代替110行来实现矩阵减法：

```
106 FOR I=1 TO 2
108  FOR J=1 TO 3
110   C(I,J) = A(I,J) - B(I,J)
112  NEXT J
114 NEXT I
```

参 阅

```
MAT +, MAT *, DIM, FOR, NEXT, MAT READ, MAT PRINT
```

语 句

MAT * 语句用于两种不同类型的乘法。

在MAT B = (K) * A的形式中（称为纯量乘法），数组A的每一个元素被乘以K值，然后把结果存放在数组B中。必须把K的值放在括号里，并且数组B的定维必须与数组A相同。

MAT A = (K) * A的形式是合法的，并把乘积再存放在数组A中。

测试程序 # 1

```
10 REM *  SCALAR MULTIPLICATION TEST PROGRAM *
20 DIM A(2,3), B(2,3)
30 FOR I=1 TO 2
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70 NEXT I
80 K=10
90 MAT B=(K)*A
100 FOR I=1 TO 2
110  FOR J=1 TO 3
120   PRINT B(I,J);
130  NEXT J
140  PRINT
150 NEXT I
160 PRINT "'MAT *' PASSED THE TEST IF 10 20 30"
170 PRINT "                              40 50 60  WAS PRINTED"
180 DATA 1, 2, 3, 4, 5, 6
999 END
```

运行实例

```
 10  20  30
 40  50  60
'MAT *' PASSED THE TEST IF 10 20 30
                           40 50 60 WAS PRINTED
```

也可以用MAT * 语句把一个矩阵A与另一个矩阵B相乘，条件是矩阵A的大小为

M×N，而B的大小为N×P（即A的第二个维数必须等于B的第一个维数）。这种乘法称为矩阵乘法。

矩阵乘法的结果存放在大小为M*N的矩阵中。MAT A = A * B是非法的。然而，若A是一个方阵，那么MAT B = A * A是正确的。（注意：MAT C = B * A产生的结果与MAT C = A * B产生的结果是不同的。）

测试程序 # 2

```
10 REM * MATRIX MULTIPLICATION TEST PROGRAM *
20 DIM A(2,3), B(3,2), C(2,2)
30 FOR I=1 TO 2
40  FOR J=1 TO 3
50   READ A(I,J)
60  NEXT J
70 NEXT I
80 FOR I=1 TO 3
90  FOR J=1 TO 2
100   READ B(I,J)
110  NEXT J
120 NEXT I
130 MAT C=A*B
140 FOR I=1 TO 2
150  FOR J=1 TO 2
160   PRINT C(I,J);
170  NEXT J
180  PRINT
190 NEXT I
200 PRINT "'MAT *' PASSED THE TEST IF 10  14
210 PRINT "                                4  41  WAS PRINTED"
220 DATA 1,2,3,  4,5,0
230 DATA 1,4,  0,5,  3,0
999 END
```

运行实例

```
10  14
4  41
'MAT *' PASSED THE TEST IF 10  14
                           4  41  WAS PRINTED
```

其它用法

某些计算机可以省写MAT。在那些程序中，测试程序中的90行可用下面的形 式 来代替：

90 B = (K) * A

其结果相同。

测试程序# 2 中的130行也可以等效于：

130 C = A * B

如果你的计算机没有此功能

如果MAT *语句在你的计算机上不适用，那么你可以用嵌套的FOR—NEXT循环

来实现以上两种乘法。对于纯量乘法，可用下面的语句代替测试程序#1中的90行：

```
86 FOR I=1 TO 2
88  FOR J=1 TO 3
90   B(I,J)=K*A(I,J)
92  NEXT J
94 NEXT I
```

对矩阵乘法，需要用三个嵌套的循环来代替130行：

```
122 FOR I=1 TO 2
124  FOR J=1 TO 2
126   C(I,J)=0
128   FOR K=1 TO 3
130    C(I,J) = C(I,J) + A(I,K) * B(K,J)
132   NEXT K
134  NEXT J
136 NEXT I
```

参　阅

MAT +, MAT -, DIM, FOR, NEXT, MAT READ, MAT PRINT

函　数

用MAX函数来确定两个值中哪一个较大。

例如：Y = A MAX 5，如果A的值大于5，那么就把A的值赋给Y。否则，Y就等于5。

测试程序 = 1

```
10 REM 'MAX' TEST PROGRAM
20 A=12
30 Y=A MAX 5
40 IF Y=12 THEN 70
50 PRINT "'MAX' FAILED THE TEST"
60 GOTO 99
70 PRINT "'MAX' PASSED THE TEST"
99 END
```

运行实例

```
'MAX' PASSED THE TEST
```

Micropolis BASIC使用MAX来把两个字符串进行比较，并得到字母表中位置在后

面的那一个字符串。

例如：MAX（“ABC”，“XYZ”），即得到函数值为“XYZ”。

其它用法

一些计算机使用MAX（A，B）的形式来达到同样的目的。还有少数计算机用MAX函数可以求出一个数组中的最大值。例如：

```
30  M = MAX (A)
```

如果你的计算机没有此功能

用下面的公式可以确定两个数的最大值（Y）：

$$Y = (A + B + ABS(A - B)) / 2$$

或用下面的行代入测试程序：

```
32 Y=A
32 IF A>=B THEN 40
34 Y=B
```

用下面这个程序可以确定一个数组中的最大值。

测试程序 =2

```
10 DIM A(6)
20 FOR I=1 TO 6
30  READ A(I)
40 NEXT I
50 M=A(1)
60 FOR I=2 TO 6
70  IF M>=A(I) THEN 90
80  M=A(I)
90 NEXT I
100 PRINT "THE MAXIMUM VALUE IS";M
110 DATA 3,5,13,1,8,-3
999 END
```

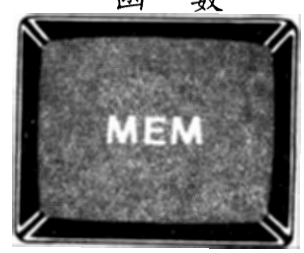
运行实例

```
THE MAXIMUM VALUE IS 13
```

参　阅

ABS，MIN，　=

函 数



M.

通常把MEM函数用在一个打印命令中来显示计算机内存中还没有使用的字节的总数。也可以把MEM用作程序语句。

测试程序

```
10 REM 'MEM' TEST PROGRAM
20 PRINT MEM; "BYTES OF MEMORY ARE
   REMAINING"
30 END
```

运行实例（典型的）

```
13504 BYTES OF MEMORY REMAINING
```

（当然，可用的内存总数取决于你的计算机的内存容量。）

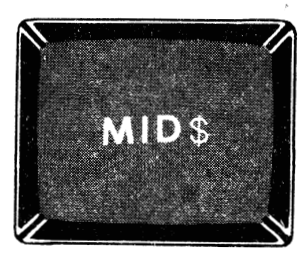
替代的形式

TRS—80 Level I用M.作为MEM的缩写。

参 阅

FRE, CLEAR

函 数



MID

MID$（串，n1，n2）用来分离出一个字符串，该字符串含有从左端第n1个字符开始的n2个字符。

例如，PRINT MID$（“COMPUTER”，4，3）打印出字母PUT，它是字符串中最左端第4个字符开始的三个“中间的”（MIDdle）字符。

字符串必须用引号括起来或者把它赋给一个字符串变量。字符的个数（n2）和起始位置（n1）可以用变量、常数或算术表达式表示。在MID$函数中的各项必须用逗号分隔。

如果n1或n2的值是一个带小数点的数，计算机自动地把它转换成整数值。

测试程序＃1

```
10 REM 'MID$' TEST PROGRAM
20 A$="CONTESTANT"
30 B$=MID$(A$,4,4)
40 PRINT MID$("ATHENA",2,3)" 'MID$' FUNCTION PASSED
   THE ";B$
99 END
```

运行实例

```
THE 'MID$' FUNCTION PASSED THE TEST
```

假如省略长度（n2），大多数BASIC将从指定的位置开始，把后面所有的字符打印出来。然而，有一些（例如，TDL BASIC）只把第n1个字符打印出来。其它的BASIC则不允许省略长度值n2。

测试程序＃2

```
10 REM 'MID$(A$,N)' TEST PROGRAM
20 PRINT MID$("RODENT",3)
99 END
```

请你想想，此程序是打印出DENT,D，还是打印出错误信息呢？

一些解释系统（例如，BASIC—80、TDL BASIC等等），可以把MID$写在等号的左边，用来改变字符串变量的内容。例如，30 MID$（A$，3，5）=B$，用B$中前五个字符取代A$中从左端第三个字符开始的五个字符。假如B$不足五个字符，则在A$中插入空格。如果长度省略不写（在此例中指的是5），则A$中所有余下的字符都被B$所取代。

测试程序＃3

```
10 REM 'MID$=' TEST PROGRAM
20 A$="CORPORATION"
30 MID$(A$,3,4)="MPUT"
40 PRINT "A COMPUTER WOULD SURE HELP THIS ";A$
99 END
```

运行实例

```
A COMPUTER WOULD SURE HELP THIS COMPUTATION
```

替代的形式

某些解释系统（例如，DEC的BASIC－PLUS和Harris BASIC－V）允许用MID代替MID$来分离子串。

如果你的计算机没有此功能

即使没有MID$,函数，大多数计算机也有办法来分离出子串。有的用SEG$,而有的用SUBSTR 。

有的计算机要求对字符串变量定义长度（例如，北极星（North Star），Hewlett—Packard等等），它们用下标来分离字符串字符。例如：

```
10 DIM A$(8)
20 A$="ABCDEFGH"
30 PRINT A$(3,5)
99 END
```

打印出CDE。

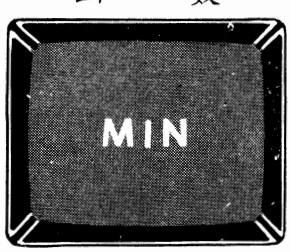
Sinclair ZX80 8K BASIC要求30语句写成：

```
30 PRINT A$(3 TO 5)
```

参　阅

PRINT，RIGHT$，LEFT$，CHR$，SPACE$，STR$，STRING$，INKEY$，SEG$

函　　数



MIN用来确定二个值中较小的一个值。

例如，Y = A MIN 5，如果A的值小于5，则把A赋给Y。否则，Y = 5 。

测试程序＃1

```
10 REM 'MIN' TEST PROGRAM
20 A=2
30 Y=A MIN 5
40 IF Y=2 THEN 70
50 PRINT "'MIN' FAILED THE TEST"
60 GOTO 99
70 PRINT "'MIN' PASSED THE TEST"
99 END
```

运行实例

```
'MIN' PASSED THE TEST
```

其它用法

有些计算机使用的形式是MIN(A, B)，其作用如前所述。少数计算机使用MIN可以找到数组中最小的值。例如：

```
30  M = MIN( A )
```

Micropolis BASIC使用MIN进行两个字符串的比较，然后得到按字母表顺序 排 在前面的字符串。

例如：MIN（“ABC”，“XYZ”）得到“ABC”。

如果你的计算机没有此功能

可以用以下公式来确定两个数中较小的值（Y）：

```
Y= (A+B-ABS(A-B))/2
```

或者用这三个语句来确定：

```
30 Y=A
32 IF A<=B THEN 40
34 Y=B
```

用下面的程序能够确定数组中的最小值：

测试程序＃2

```
10 DIM A(6)
20 FOR I=1 TO 6
30  READ A(I)
40 NEXT I
50 M=A(1)
60 FOR I=2 TO 6
70  IF M<=A(I) THEN 90
80  M=A(I)
90 NEXT I
100 PRINT "THE MINIMUM VALUE IS ";M
110 DATA 3,5,13,1,8,-3
999 END
```

运行实例

```
THE MINIMUM VALUE IS -3
```

参 阅

ABS，MAX，<=

在某些计算机上（例如，H. P. 3000，COMPAL，Harris 计算机系统以及 Apple）使用X MOD Y 来计算X被Y除后的余数（MODulo）。

例如，PRINT 8 MOD 5打印出 8 被 5 除后的余数 3 。

少数计算机自动地化整余数值。

例如，PRINT 10.5 MOD 4 打印出 2 （余数2.5的整数值）。

测试程序

```
10 REM 'MOD' TEST PROGRAM
20 A = 13 MOD 5
30 IF A = 3 THEN 60
40 PRINT "THE MOD FUNCTION FAILED THE TEST"
50 GOTO 99
60 PRINT "THE MOD FUNCTION PASSED THE TEST"
99 END
```

运行实例

```
THE MOD FUNCTION PASSED THE TEST
```

如果你的计算机没有此功能

MOD用起来是很方便的，但并不是不能替换的。下面是一步一步实现此功能的方法。

```
20 A = 13/5
22 A = A-INT(A)
24 A = INT(A*5)
```

一般的形式可以写成：

```
20 A = INT(X - Y*INT(X/Y))
```

你可将13代入X， 5 代入Y，用它代替测试程序中的20语句试试。

其它用法

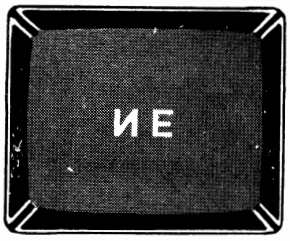
少数计算机（例如Harris BASIC—V）使用MOD(X, Y) 的形式计算X被Y除后的余数。

参 阅

INT , FIX

命　令

运算符



在个别计算机中（例如 T．I．990）用 NE 作为NEW命令和关系运算符“不等于”（not-equal）（<>）的缩写形式。当在命令方式使用NE时，认为它是NEW，而作为一个程序语句使用时，把它看作不等号< >。

程序=1使用NE作为命令NEW。详见NEW。

测试程序 = 1

```
10 REM 'NE(NEW)' TEST PROGRAM
20 PRINT "HELLO THERE"
99 END
```

运行实例

用LIST命令列出程序清单，以确保上面表示的程序已送入计算机中，打入NE以清除测试程序，然后再一次打入LIST证实程序已被“清除”。

程序=2使用NE作为“不等于”关系运算符。详见< >。

测试程序 = 2

```
10 REM 'NE (<>)' TEST PROGRAM
20 A=10
30 IF A NE 20 THEN 60
40 PRINT "THE NE OPERATOR FAILED THE TEST"
50 GOTO 99
60 PRINT "THE NE OPERATOR PASSED THE TEST"
99 END
```

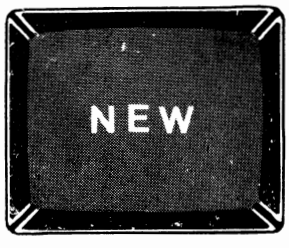
运行实例

```
THE NE OPERATOR PASSED THE TEST
```

参　阅

NEW，<>，<，>，<=，>=，=，EQ，GE，GT，LE，LT，IF-THE.

命　令



NEW命令清除存放在内存中的BASIC程序。然而，它不清除解释程序本身。在将一个新的程序输入计算机，并清除已在内存中的程序时使用NEW。

测试程序

```
10 REM 'NEW' COMMAND TEST PROGRAM
20 PRINT "HELLO THERE."
99 END
```

NE

N．

运行实例

列出程序清单以确保上面的程序已经输入。用PRINT MEM命令（或者用PRINT FRE（0），或者用其它相应的命令）检查剩余的内存空间。

打入NEW命令以清除测试程序，然后再检查一次内存空间。我们能够看到可用内存有了相应的增加。

为了确证程序已被“清除”，打入LIST命令再进行一次检查。

有些计算机可使用SCRATCH或SCR代替。

替代的形式

少数计算机接受NE或N．作为NEW的缩写形式。

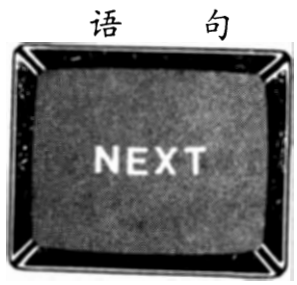
其它用法

有些计算机（例如，Sinclair ZX80）使用NEW n清除程序，并且同时给BASIC建立一定数量的可用内存。如果机器语言程序和BASIC程序同时出现在内存，切实可行的（甚至是必要的）办法是给机器语言程序留出内存的顶部。n是十进制数。

参　阅

CLEAR，SCRATCH

NEXT语句用来使程序返回到循环体的开头，即与NEXT 相应的语句FOR的下一语句。当超出FOR语句的循环范围时，计算机转移到NEXT语句后面的行继续执行程序。



NEX
N．

例如：

```
10 FOR X=1 TO 3
20 NEXT X
99 END
```

NEXT语句执行三次，X值增长到4，超出了FOR语句的循环范围3，计算机执行99行。

测试程序＝1

```
10 REM 'NEXT' TEST PROGRAM
20 FOR X=1 TO 4
30 PRINT X,
40 NEXT X
50 PRINT
60 PRINT "THE 'NEXT' STATEMENT PASSED THE TEST."
99 END
```

运行实例

```
 1             2             3             4
THE 'NEXT' STATEMENT PASSED THE TEST.
```

因为NEXT语句只返回到前面的循环体（即使用同一个循环变量的FOR语句的下一语句），所以大部分机器可以使用“嵌套”FOR—NEXT语句。详见FOR。

测试程序＝2

```
10 REM TEST PROGRAM WITH NESTED 'NEXT' STATEMENTS
20 FOR A=1 TO 3
30 FOR B=1 TO 4
40 PRINT A;B,
50 NEXT B
60 PRINT
70 NEXT A
80 PRINT "THE 'NEXT' STATEMENT PASSED THE TEST WHEN
   NESTED"
99 END
```

运行实例

```
 1  1          1  2          1  3          1  4
 2  1          2  2          2  3          2  4
 3  1          3  2          3  3          3  4
THE 'NEXT' STATEMENT PASSED THE TEST WHEN NESTED
```

许多计算机允许NEXT不带变量。在这种情况下，只要不超出该循环规定的范围，计算机返回到前面的FOR语句（而不按同一循环变量名来找FOR语句）。

为了测试这个性能，我们从测试程序2中消去第50行中的循环变量B和第70行的循环变量A，然后再将程序运行一次，结果应同前。

有些计算机允许NEXT指定一个以上的循环变量。为了结束三重嵌套循环，可以把NEXT语句写成这种形式：NEXT K, J, I（循环变量的次序与相应的FOR语句相反）。

替代的形式

少数计算机允许用NEX和N.作为NEXT的缩写形式。

其它用法

某些计算机（例如DEC BASIC－PLUS－2）在一定情况下允许隐含NEXT。即写FOR但不写 NEXT。

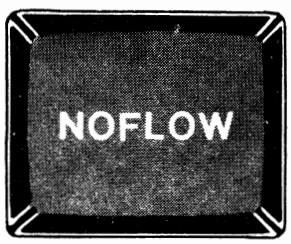
例如：

```
30 PRINT X,X*X FOR X=1 TO 5
```

参 阅

FOR

命 令
语 句



Micropolis BASIC 用NOFLOW命令来终止它的跟踪功能（见FLOW）。NOFLOW也可以作为一个程序语句，在程序中指定的地方关闭跟踪功能。

测试程序

```
10 REM 'NOFLOW' TEST PROGRAM
20 PRINT "THE FIRST THREE LINES OF
   THIS PROGRAM"
30 NOFLOW
40 PRINT "ARE PRINTED WITH THE TRACE TURNED ON."
50 PRINT "THIS LINE IS PRINTED WITH THE TRACE TURNED OFF."
99 END
```

运行实例

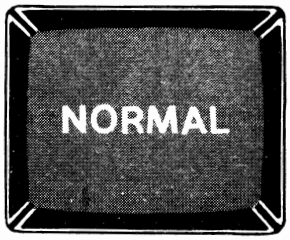
在打入FLOW命令后运行测试程序

```
<10> <20> THE FIRST THREE LINES OF THIS PROGRAM
<30> ARE PRINTED WITH THE TRACE TURNED ON.
THIS LINE IS PRINTED WITH THE TRACE TURNED OFF.
```

参 阅

FLOW, NOTRACE, TRACE OFF, TROFF

命令语句



APPLE Ⅱ 用NORMAL作为命令或语句，使显示返回到正常方式。在这种方式中，来自计算机的所有输出在黑色背景上显示出白色字符。NORMAL是在用了FLASH或者INVERSE之后使用的，这二者产生特殊的显示效果。

测试程序

```
10 REM 'NORMAL' TEST PROGRAM
20 INVERSE
30 PRINT "THIS IS INVERSE PRINTING."
40 NORMAL
50 PRINT "BACK TO NORMAL"
99 END
```

为了运行这个程序，清除屏幕并打入RUN。

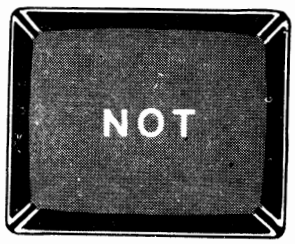
运行实例

```
THIS IS INVERSE PRINTING.
BACK TO NORMAL
```

参 阅

INVERSE，FLASH

操作符



在IF—THEN语句中，用NOT作为取反条件的逻辑运算符。

例如，IF NOT（A＞5） THEN 60，意思是："如果存放在A中的值不大于5，转到60行。"

测试程序＃1

```
10 REM LOGICAL 'NOT' TEST PROGRAM
20 A=3
30 IF NOT(A>5) THEN 60
40 PRINT "'NOT' FAILED THE LOGICAL OPERATOR TEST"
50 GOTO 99
60 PRINT "'NOT' PASSED THE LOGICAL OPERATOR TEST"
99 END
```

运行实例

```
'NOT' PASSED THE LOGICAL OPERATOR TEST
```

少数计算机在字符串的比较中使用NOT。例如，IF NOT（A$ = “YES” OR A$ = “NO”）THEN 60；意思是“如果存放在A$中的字符串既不是YES也不是NO，程序控制转到60行”。

测试程序 # 2

```
10 REM STRING LOGICAL 'NOT' TEST PROGRAM
20 PRINT "TYPE A YES OR A NO";
30 INPUT A$
40 IF NOT(A$="YES" OR A$="NO") THEN 70
50 PRINT "THANK YOU"
60 GOTO 99
70 PRINT A$;" IS NEITHER YES NOR NO!"
80 GOTO 20
99 END
```

运行实例

```
TYPE A YES OR A NO? OK
OK IS NEITHER YES NOR NO!
TYPE A YES OR A NO? NO
THANK YOU
```

有些计算机使用NOT算符求一个数的二进制反码（即改变用二进制数表示的每一位。所有的0变成1，所有的1变成0。）

测试程序 # 3

```
10 REM 'NOT' COMPLEMENT TEST PROGRAM
20 PRINT "ENTER A NUMBER BETWEEN -32768 AND 32767";
30 INPUT A
40 B=NOT(A)
50 PRINT "THE BINARY COMPLEMENT OF";A;"IS";B
60 GOTO 20
99 END
```

运行实例（输入5）

```
ENTER A NUMBER BETWEEN -32768 AND 32767? 5
THE BINARY COMPLEMENT OF 5 IS -6
ENTER A NUMBER BETWEEN -32768 AND 32767?
```

如果你的计算机没有此功能

通过改变规定的条件，能够写出不用NOT而作用相同的语句。

例如，NOT（A$ = “YES” OR A$ = “NO”）与A$ <> “YES” AND A$ <> “NO” 作用相同。

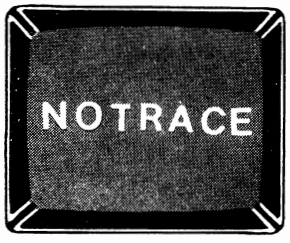
一个数A的二进制反码可以由B = -（A + 1）得到（A和B都是十进制数），例如5的反码得 -6。

参 阅

```
AND, OR, XOR, IF-THEN
```

命 令

语 句



APPLE Ⅱ BASIC 用NOTRACE 终止跟踪方式(见(TRACE)。NOTRACE 也可作为一个语句，在程序中指定的地方关闭跟踪功能。

测试程序

```
10 REM 'NOTRACE' TEST PROGRAM
20 TRACE
30 PRINT "EACH LINE SHOULD BE TRACED"
40 NOTRACE
50 PRINT "BY THE 'TRACE' STATEMENT"
60 PRINT "UNTIL TURNED OFF BY THE 'NOTRACE' STATEMENT"
99 END
```

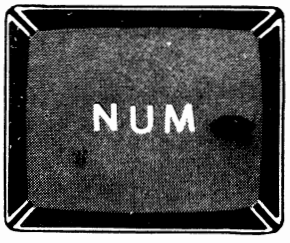
运行实例

```
#30 EACH LINE SHOULD BE TRACED
#40 BY THE 'TRACE' STATEMENT
UNTIL TURNED OFF BY THE 'NOTRACE' STATEMENT
```

参阅

```
TRACE, TRACE OFF, TROFF, NOFLOW
```

函 数



某些 BASIC（例如，Digital Group BASIC）用NUM函数把数字字符串转换成数值。也就是说：把一个数字字符串（包括小数点）转换成它所表示的数值。

例如

```
30 X = NUM("5.2")
40 PRINT X, X/2
```

把5.2赋给X 并且打印5.2和2.6。“5.2”是“字符串”形式，所以它不能用来计算。把字符串转换成数值形式，它就可以出现在计算中了，例如X/2。NUM类似于VAL函数。

测试程序=1

```
10 REM 'NUM' TEST PROGRAM
20 A$="45.12"
30 A=NUM(A$)
40 PRINT "IF THE STRING ";A$;" IS CONVERTED TO THE
   NUMBER";A
50 PRINT "THEN THE NUM FUNCTION PASSED THE TEST."
99 END
```

运行实例

```
IF THE STRING 45.12 IS CONVERTED TO THE NUMBER 45.12
THEN THE NUM FUNCTION PASSED THE TEST.
```

一些有MAT INPUT 的计算机（例如,DEC PDP—11）使用NUM 得到输入数据的个数。

测试程序# 2

```
10 REM 'NUM (MAT INPUT)' TEST PROGRAM
20 DIM N(20)
30 PRINT "TYPE A FEW NUMBERS SEPARATED BY COMMAS"
40 MAT INPUT N
50 S = 0
60 FOR I=1 TO NUM
70 S = S+N(I)
80 NEXT I
90 PRINT "THE AVERAGE OF";NUM;"VALUES IS";S/NUM
```

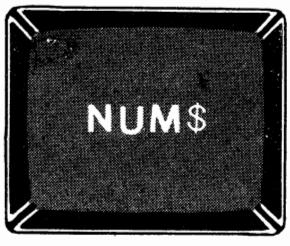
运行实例（典型的）

```
TYPE A FEW NUMBERS SEPARATED BY COMMAS
? 15, 34, 2, 8, 54, 19
THE AVERAGE OF 6 VALUES IS 22
```

参阅

VAL, MAT INPUT

函　数



NUM$ 是一个类似于STR$ 的函数。它把数值表达式转换成一个数字字符串。

例如，NUM$ （25.6）把值 25.6变成一个字符串“25.6”。它们的不同在于数值形式可参加算术运算,而字符串形式不能用于算术运算。

测试程序

```
10 REM 'NUM$' TEST PROGRAM
20 A = 123456
30 A$ = NUM$(A)
40 PRINT "IF THE NUMBER";A;"IS CONVERTED TO THE
   STRING ";A$
50 PRINT "THEN THE NUM$ FUNCTION PASSED THE TEST."
99 END
```

运行实例

```
IF THE NUMBER 123456 IS CONVERTED TO THE STRING 123456
THEN THE NUM$ FUNCTION PASSED THE TEST.
```

参阅

STR$, VAL, ASC, CHR$

ON ERROR GOTO 语句的作用是，当程序发生错误时转移到“出错子程序”，而不停止程序的执行。ON ERROR GOTO 语句在程序中必须出现在预料的错误之前。在ON ERROR GOTO 语句之后遇到的任一错误，都会使计算机去执行 ON ERROR GOTO语句中列出行号的行。

ON ERR GOTO

测试程序

```
10 REM 'ON-ERROR-GOTO' TEST PROGRAM
20 ON ERROR GOTO 100
30 PRINT "ENTER A NUMBER AND ITS INVERSE WILL BE
   COMPUTED";
40 INPUT N
50 A=1/N
60 PRINT "THE INVERSE OF";N;"IS";A
70 GOTO 30
100 PRINT "THE INVERSE OF 0 CANNOT BE COMPUTED -
    TRY AGAIN"
110 RESUME 30
999 END
```

运行实例（输入 4 和 0）

```
ENTER A NUMBER AND ITS INVERSE WILL BE COMPUTED?4
THE INVERSE OF 4 IS .25
ENTER A NUMBER AND ITS INVERSE WILL BE COMPUTED? 0
THE INVERSE OF 0 CANNOT BE COMPUTED - TRY AGAIN
ENTER A NUMBER AND ITS INVERSE WILL BE COMPUTED?
```

在此有一个**被 0 除**的错误。

如果在执行 ON ERROR GOTO 子程序时执行 ON ERROR GOTO 0，则打印出错误信息并停止程序的执行。在测试程序中增加下面语句来测试这个特性：

105 ON ERROR GOTO 0

某些计算机遇到语法错误时，在执行 ON ERROR GOTO 语句和停止程序执行之后，由编辑功能打印出有错误的行。然后计算机进入编辑工作方式。把测试程序中的50行改成 50 ILLEGAL LINE以测试此特性。

一般使用 RESUME 语句来从一个 ON ERROR GOTO 子程序返回到主程序。

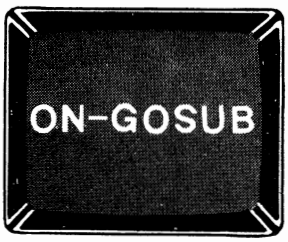
替代的形式

少数计算机（例如 APPLESOFT）可用 ON ERR GOTO 代替 ON ERROR GOTO。

参阅

ERROR，RESUME，ERR，ERL

语　　句



ON—GOS.

ON—GOSUB 是一个多子程序分支结构。它相当于把若干个IF—GOSUB 测试组成一个语句。

例如，ON X GOSUB 100，200，300，如果X的整数值分别为1、2或3，则指示计算机转去执行以行号100、200或300开始的子程序。如果此例中的INT X 小于1或大于3，则ON—GOSUB 无效。在某些计算机中，如果ON—GOSUB 指令无效，执行程序的下一行；在另外一些计算机中，则以程序“失败”告终，并打印出错误信息。

测试程序

```
10 REM 'ON-GOSUB' TEST PROGRAM
20 PRINT "ENTER THE NUMBER 1, 2 OR 3";
30 INPUT X
40 PRINT "THE ON-GOSUB STATEMENT";
50 ON X GOSUB 100,200,300
60 GOTO 20
100 REM SUBROUTINE #1
110 PRINT "BRANCHED TO SUBROUTINE #1"
120 RETURN
200 REM SUBROUTINE #2
210 PRINT "BRANCHED TO SUBROUTINE #2"
220 RETURN
300 REM SUBROUTINE #3
310 PRINT "BRANCHED TO SUBROUTINE #3"
320 RETURN
999 END
```

运行实例

```
ENTER THE NUMBER 1, 2 OR 3? 1
THE ON-GOSUB STATEMENT BRANCHED TO SUBROUTINE #1
ENTER THE NUMBER 1, 2 OR 3? 2
THE ON-GOSUB STATEMENT BRANCHED TO SUBROUTINE #2
ENTER THE NUMBER 1, 2 OR 3? 3
THE ON-GOSUB STATEMENT BRANCHED TO SUBROUTINE #3
ENTER THE NUMBER 1, 2 OR 3?
```

利用同一个测试程序，输入大于1小于4的小数。

输入小于1大于4的值又如何呢？请运行一下程序试试看。

如果你的计算机没有此功能

如果你的计算机不允许使用 ON—GOSOB，可用下面的语句代替。

```
45 IF X=1 GOSUB 100
50 IF X=2 GOSUB 200
55 IF X=3 GOSUB 300
```

（请注意，在被调用的子程序中X的值是不改变的。）在执行以上语句时，实际上是按以下语句实现的。

```
45 IF INT(X)=1 THEN 100
50 IF INT(X)=2 THEN 200
55 IF INT(X)=3 THEN 300
```

涉及ON—GOSOB 语句的其它技巧请见ON—GOTO。

替代的形式

TRS—80 LevelI 允许用ON—GOS。

参阅

ON-GOTO，ON-ERROR-GOTO，GOTO-OF，GOSUB-OF，GOSUB

语　句

ON—GOT

ON—G

ON—GOTO 是一个多分支结构。它相当于把若干个IF—THEN 测试组成一个语句。例如，ON X GOTO 100，200，300，如果X的值分别为1、2或3，则指示计算机转去执行行号为100，200 或300 的行。如果X的值小于1或大于3.999，测试失败，并继续执行下一个程序行。

在这个语句中X的整数值不能超过可能分支的数。如果X的值是一个小数，计算机自动地求出它的整数值，并且选择对应的分支行号。

测试程序

```
10 REM 'ON(X)GOTO' TEST PROGRAM
20 PRINT "ENTER THE NUMBER 1, 2 OR 3"
30 INPUT X
40 PRINT "THE ON-GOTO STATEMENT";
50 ON X GOTO 100,200,300
60 PRINT "FAILED THE TEST"
70 GOTO 999
100 PRINT "BRANCHED TO LINE 100"
110 GOTO 20
200 PRINT "BRANCHED TO LINE 200"
210 GOTO 20
300 PRINT "BRANCHED TO LINE 300"
310 GOTO 20
999 END
```

运行实例（输入1， 2和3）

```
ENTER THE NUMBER 1,2 OR 3
? 1
THE ON-GOTO STATEMENT BRANCHED TO LINE 100
ENTER THE NUMBER 1, 2, OR 3
? 2
THE ON-GOTO STATEMENT BRANCHED TO LINE 200
? 3
THE ON-GOTO STATEMENT BRANCHED TO LINE 300
ENTER THE NUMBER 1, 2 OR 3
```

使用同一个测试程序，输入小于1的值，然后输入大于3.999的值，再运行一下。

替代的形式

少数计算机允许在ON—GOTO 语句中对 GOTO简写。PDP—8 E 写成ON X GOT的形式，而Tiny BASIC则允许ON X G． 的形式。

如果你的计算机没有此功能

如果计算机不允许使用ON—GOTO 测试，可用下面的语句代替：

```
45 IF X=1 THEN 100
50 IF X=2 THEN 200
55 IF X=3 THEN 300
```

在执行以上语句时，实际上是按以下语句实现的。

```
45 IF INT(X)=1 GOSUB 100
50 IF INT(X)=2 GOSUB 200
55 IF INT(X)=3 GOSUB 300
```

技巧

由于截断的原因，如果所产生的X值比所要求的稍微低些，会产生错误。为了避免这个错误，我们对X 的 值 稍加提高，以保证ON—GOTO语句的正确运行。例如：

```
ON X+.1 GOTO 100, 200, 300
```

在这种情况下，如果X的值经过舍入为1.99,而不是所要求的值2.0,加上0.1使X的值大于2（2.09）。然后再通过内部取整函数使 2.09 变成所要求的2.0。如果不出现舍入误差，这样做也不会出现错误。

改换基数

当X的值不是1、2或3时,为了使ON—GOTO语句有效,我们可以采用表达式形式。例如：

```
ON X-50 GOTO 100, 200, 300
```

当X的值分别为51、52或53时，分支到100、200或300行。

其它用法

不同的解释系统对分支选择的个数可能有不同的限制（例中只用了3个）。

也可以使用带有某些关键字的ON—GOTO语句,使指定的条件发生时,分支到程序的另一部分去。

当程序企图读入的数据个数比存放在磁盘或磁带上的数据个数多时,ON END GOTO 2000使程序控制转移到2000行。为了同一个目的,有的计算机可使用这种形式：ON EOF（1）GOTO 2000。

如果检查出任意一个错误，ON ERROR GOTO 2000 将与ON-ERR GOTO 2000 语句一样，转到2000行。详见ON—ERROR GOTO。

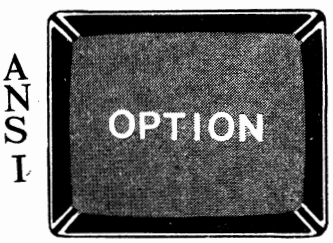
ON—GOTO 的另一种形式是ON X RESTORE 语句。它将把数据指针恢复到ON语句所指示的DATA 行的开始。

例如：ON X RESTORE 200，210，220，如果 X 是1，把DATA指针恢复到200行，如果X是2恢复到210行，如果X是3恢复到220行。详见RESTORE。

参阅

ON-GOSUB，ON-ERROR，GOTO-OF，GOSUB-OF，GOTO

语　句



在Harris BASIC—V 中 OPTION 与BASE 语句一起用来定义可变的数组下限（0到10之间的一个任意整数），即定义下标值最小的数组元素。例如：

```
10 OPTION BASE=5
20 DIM A(10)
```

OPTION BASE 语句定义这个数组具有6个元素〔从A（5）到A（10）〕。

如果不指定OPTION BASE 的值，计算机默认 BASE 的值为0。　详见BASE。

测试程序

```
10 REM 'OPTION' TEST PROGRAM
20 OPTION BASE=3
30 DIM A(5)
40 FOR X=3 TO 5
50 A(X)=X
60 NEXT X
70 OPTION BASE=0
80 FOR X=0 TO 2
90 A(X)=X
100 NEXT X
110 FOR X=0 TO 5
120 PRINT A(X);
130 NEXT X
140 PRINT "THE OPTION STATEMENT DID NOT CRASH"
999 END
```

运行实例

```
 0  1  2  3  4  5 THE OPTION STATEMENT DID NOT CRASH
```

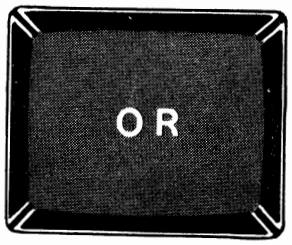
其它用法

ANSI 标准BASIC 只能指定OPTION BASE 的值为0和1，并且BASE 后面不需要等号。

参阅

BASE

语 句



OR 是与 IF–THEN 语句一起使用的，OR 用来建立“逻辑运算”操作，以对多条件进行测试。

例如： IF A = 2 OR B = 6 THEN 70 ，意思是“如果变量A的值等于2，或者变量B 的值等于6，或者二个条件均 满 足，则执行“跳到70行”。

测试程序＃ 1

```
10 REM LOGICAL 'OR' TEST PROGRAM
20 A=8
30 B=6
40 IF A=2 OR B=6 THEN 70
50 PRINT "OR FAILED THE TEST AS A LOGICAL OPERATOR"
60 GOTO 99
70 PRINT "OR PASSED THE LOGICAL OPERATOR TEST"
99 END
```

运行实例

```
OR PASSED THE LOGICAL OPERATOR TEST
```

少数计算机允许使用OR运算符对文字串进行复合测试。

例如， IF A$= “Y” OR A$= “YES”THEN 80，意思是“如果串变量A$包含字母Y或者包含字“YES”,则满足IF–THEN条件，并转到80行继续执行。”某些计算机允许用“＋”号代替OR。

测试程序＃ 2

```
10 REM STRING LOGICAL 'OR' TEST PROGRAM
20 A$="A"
30 B$="F"
40 IF A$="A" OR B$="B" THEN 70
50 PRINT "OR FAILED THE STRING LOGICAL OPERATOR TEST"
60 GOTO 99
70 PRINT" "OR PASSED THE STRING LOGICAL OPERATOR TEST"
99 END
```

运行实例

```
OR PASSED THE STRING LOGICAL OPERATOR TEST
```

有些计算机使用逻辑运算符OR来测定两个逻辑操作中是否有一个是满足条件的。如果至少有一个条件满足，则用一个TRUE值来响应(大部分计算机用－1；具体情况请查阅你所使用的计算机手册）。当条件都不满足时,用一个FALSE值来响应（大部分计算机用0）。

例如，PRINT A= 4 OR B＞5，意思是：“如果A的值为4,或者B的值大于5，或者二者都满足，计算机将打印出－1。”

测试程序＃ 3

```
10 REM 'OR' LOGICAL TEST PROGRAM
20 PRINT "ENTER A NUMBER FROM 1 TO 10";
30 INPUT A
40 B= A<1 OR A>10
50 IF B<>0 THEN 80
60 PRINT A; "IS A NUMBER BETWEEN 1 AND 10"
70 GOTO 20
80 PRINT A; "IS NOT GREATER THAN 0 AND LESS THAN 11"
90 GOTO 20
99 END
```

运行实例（典型的）

```
ENTER A NUMBER FROM 1 TO 10? 6
 6 IS A NUMBER BETWEEN 1 AND 10
ENTER A NUMBER FROM 1 TO 10? 13
 13 IS NOT GREATER THAN 0 AND LESS THAN 11
```

少数计算机使用OR算符按照布尔代数的规则进行两个数的二进制逻辑“OR”运算。OR按位比较二个数的二进制形式。如果进行“或”运算的位中有一个是1，计算机输出1。

例如：

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

因此，当计算机将一个数“或”另一个数时，对二个数的相应位进行“或”的运算，产生第三个数。

例如：

	十进制		二进制
	3		0011
逻辑		OR	
	5		0101
=	7		0111

在这个例子中，只有当相应位都是 0 时，相应的结果位上才为 0 。

测试程序＃ 4

```
10 REM 'OR' BINARY LOGIC TEST PROGRAM
20 PRINT "ENTER A VALUE FOR X";
30 INPUT X
40 PRINT "ENTER A VALUE FOR Y";
50 INPUT Y
60 A = X OR Y
70 PRINT "THE LOGICAL 'OR' VALUE OF ";X;" AND ";Y;
   "IS ";A
80 GOTO 20
99 END
```

运行实例（输入 6 和 10）

```
ENTER A VALUE FOR X? 6
ENTER A VALUE FOR Y? 10
THE LOGICAL 'OR' VALUE OF 6 AND 10 IS 14
ENTER A VALUE FOR X?
```

其它用法

有时把OR算符用在另一种形式中。如OR（P$，Q$）用来修改字符串。如果P$ 和Q$ 是字符串，它们的内容将按字符进行“或”运算，结果放在P$ 中。如果P$ 的字符比Q$ 的少，Q$ 中多余的字符被忽略。 如 Q$ 的字符少，P$ 中多余的字符保持不变。

王安（WANG）2200B还接受OR（A$、B），此处A$ 是一字符串，B是一个16进制常数，A$ 的每一个字符与16进制数B进行“或”运算，结果放在A$ 中。

例如：如果P$ = “ABC”，Q$ = “DEF”，OR（P$，Q$）的结果“EGG”放在P$ 中。如果A$ = “DOD”，OR（A$，02）的结果“FFF”放在A$ 中。

测试程序＃ 5

```
10 REM 'OR' STRING MODIFIER TEST PROGRAM
20 A$="ABC"
30 B$="LMN"
40 OR (A$,B$)
50 PRINT "OR PASSED THE TEST IF ";A$;"=MOO"
99 END
```

运行实例

```
OR PASSED THE TEST IF MOO=MOO
```

如果你的计算机没有此功能

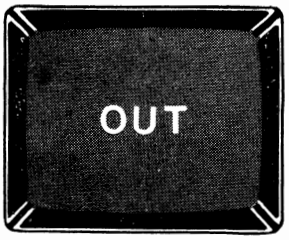
如果你的计算机没有逻辑算符OR，可用减法和乘法来模拟它。请用下面的语句替换测试程序＃1 的40行：

```
40 IF (A-2) * (B-6) = 0 THEN 70
```

参阅

AND，XOR，NOT，+，*

语　句



OUT语句的作用是把一个数（字节值）送到指定的计算机输出端口。

OUT语句的形式是：OUT（端口，字节值）。

字节值和端口值必须是正整数，或是值的范围在0到255之间的变量。例如：OUT 255，4　把与4(十进制)等效的二进制数送到255号端口。

按下盒式录音机的Play键并试一下测试程序＃1。

测试程序＃1（在TRS—80机上）

```
10 REM 'OUT' TEST PROGRAM
20 PRINT "ENTER '4' TO TURN ON THE CASSETTE RECORDER
   MOTOR"
30 INPUT X
40 OUT 255,X
50 PRINT "ENTER '0' TO TURN THE MOTOR OFF"
60 INPUT X
70 OUT 255,X
99 END
```

运行实例

```
ENTER '4' TO TURN ON THE CASSETTE RECORDER MOTOR
? 4
ENTER '0' TO TURN THE MOTOR OFF
? 0
```

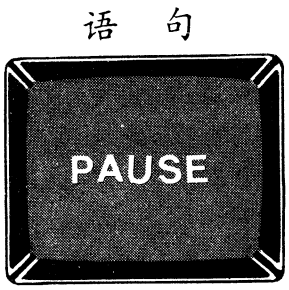
如果盒式录音机打不开，试运行下面这个程序，去查找能使你的计算机工作的端口和字节值。

测试程序＃2

```
10 REM 'OUT' SEARCH PROGRAM
20 FOR P=0 TO 255
30 FOR B=0 TO 255
40 PRINT "PORT#";P,
50 PRINT "BYTE#";B
60 OUT P,B
70 NEXT B
80 NEXT P
99 END
```

参阅

INP，PIN，PEEK，POKE



Sharp/ TRS—80袖珍计算机使用语句 PAUSE 在屏幕上显示数据。PAUSE语句除了在使程序重新运行之前中止0.85秒以外，它很象PRINT语句（在袖珍计算机机上PAUSE语句用来中止程序，直到用户敲入ENTER键重新开始。）

例如：PAUSE A$；F显示A$ 的内容并且跟着把F的值显示出来。

PAUSE J，K在显示器的左边显示 J的值，并且从24字符宽的屏幕的第13列开始显示K的值。

用USING 可进行格式化输出（见PRINT USING。）

例如：PAUSE USING“####.##” ； C 将C 的值显示出来，并在二位小数之后截断（不舍入）。USING子句指定的格式对所有的PRINT和PAUSE语句都是有效的，直到另一个USING子句改变或撤消它为止。例如可用 PAUSE USING; N，在打印N的值之前撤消任何现有的格式。

其它使用方法：

少数BASIC（例如Processor Tech）使用PAUSE n语句把程序中止n/ 10秒后再继续执行。

测试程序#1

```
10 REM 'PAUSE' TEST PROGRAM
20 PRINT "PAUSE PASSED THE TEST IF THIS"
30 PRINT "MESSAGE STAYS ON THE SCREEN FOR"
40 PRINT "FIVE SECONDS...."
50 PAUSE 50
60 PRINT "BEFORE THIS PART IS PRINTED."
99 END
```

运行实例

```
PAUSE PASSED THE TEST IF THIS
MESSAGE STAYS ON THE SCREEN FOR
FIVE SECONDS....  (pause)
BEFORE THIS PART IS PRINTED.
```

如果你的计算机没有此功能

如果你使用的计算机没有这种功能，可以用FOR—NEXT循环来达到这种暂停的目的。根据经验确定适当的循环次数。然后用下面几行代替50行。

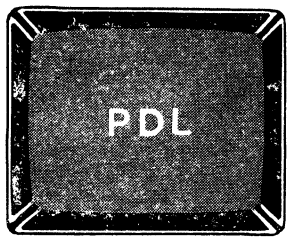
```
50 FOR X=1 TO 1800
55 NEXT X
```

改变50语句中循环终值，使程序在你所用的计算机运行时暂停 5 秒钟。

参阅

PRINT，PRINT USING

函 数



PDL是在APPLE Ⅱ BASIC 中使用的专门函数，用来指出二个游戏控制器的当前值。这二个控制器以PDL（0）和PDL（1）来标识。(PDL是“开关”(Paddle)的简写形式，并用来指出控制游戏的“开关”状态。)

测试程序

```
10 REM 'PDL' TEST PROGRAM
20 A=PDL(0)
30 B=PDL(1)
40 PRINT "THE VALUE OF PDL(0) IS";A
50 PRINT "THE VALUE OF PDL(1) IS";B
60 PRINT "CHANGE THE CONTROL UNIT SETTINGS AND (RUN)
   AGAIN"
99 END
```

运行实例（典型的）

```
THE VALUE OF PDL(0) IS 13
THE VALUE OF PDL(1) IS 146
CHANGE THE CONTROL UNIT SETTINGS AND (RUN) AGAIN
```

参阅

GR，PLOT，COLOR，KEY$

语 句

PEEK语句用来检查计算机内存中指定地址的内容。

例如：X=PEEK（18370）把存放在内存地址18370单元中的数值赋给变量X。PEEK语句指出内存地址的内容，它是0到255之间的一个数（在8位的存贮单元中可以存放这个范围的值）。PEEK与POKE语句经常一起使用，用PEEK来读出由POKE存入内存的信息。当然，所能取（PEEK）的最高地址取决于计算机内存的大小。

在运行下面的测试程序之前，请查阅一下你的计算机手册，看看内存地址18368到18380的单元是否保留下来做为自由内存区。这样可以避免把POKE数据输入到为计算机的正常操作所保留的内存地址中去。如果在你的计算机上没有把地址18368到18380的单元留下来做为自由内存区，选择13个连续的自由区地址并且适当地改变测试程序中的20行和60行。

测试程序（在TRS—80机上）

```
10 REM 'PEEK' TEST PROGRAM
20 FOR X=18368 TO 18380
30 READ Y
40 POKE X,Y
50 NEXT X
60 FOR X=18368 TO 18380
70 Y=PEEK(X)
80 PRINT CHR$(Y);
90 NEXT X
100 DATA 84,69,83,84,128,67,79,77,80,76,69,84,69
999 END
```

运行实例

```
TEST COMPLETE
```

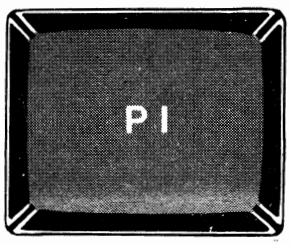
PEEK和POKE语句也和USR（X）语句一起使用，以运行机器语言子程序。

参阅

POKE，USR(X)，SYSTEM，EXAM，FETCH，STUFF，FILL

函　数

用PI代表π值（3.14159265）。



测试程序

```
10 REM 'PI' TEST PROGRAM
20 R=6
30 C=2*PI*R
40 PRINT "THE CIRCUMFERENCE OF A
   CIRCLE"
50 PRINT "WITH A RADIUS OF 6 FEET
   IS";C;"FEET"
99 END
```

运行实例

```
THE CIRCUMFERENCE OF A CIRCLE
WITH A RADIUS OF 6 FEET IS 37.6991 FEET
```

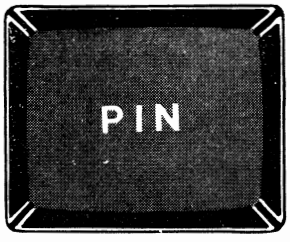
如果你的计算机没有此功能

如果你使用的计算机没有PI功能，可以用3.14159265这个值来代替它。

其它用法

在Harris BASIC—V中用PI（X）计算PI * X的值。

函　数



少数解释系统（例如，Heath Benton Harbor BASIC）使用PIN从指定的计算机端口上读一个字节的十进制值信息。这个字节值可以是任意的从0到255的正整数。

例如，PRINT PIN（X）打印端口X的字节的十进制值。

测试程序

```
10 REM 'PIN' TEST PROGRAM
20 FOR X=0 TO 255
30 PRINT "THE DECIMAL VALUE OF THE BYTE AT PORT#";
   X;"IS";PIN(X)
40 NEXT X
99 END
```

运行实例（典型的）

```
THE DECIMAL VALUE OF THE BYTE AT PORT# 0 IS 255
                                       .
                                       .
                                       .
                                       .
                                       .
                                       .
                                       .
                                       .
THE DECIMAL VALUE OF THE BYTE AT PORT# 255 IS 127
```

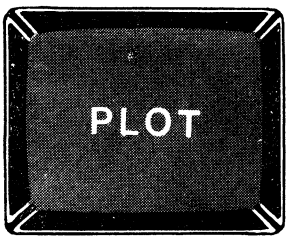
代替的字

请见INP。

参阅

INP，OUT，PEEK，USR，

语 句



在APPLEⅡ BASIC中使用PLOT(n1,n2)作为一个特殊功能。用来“打开”或“点亮”屏幕上指定格子上彩色的作图块。颜色由COLOR语句来确定(见COLOR)。

格子块的位置通过PLOT语句后面的二个参数来指定。第一个参数（n 1）指定列，第二个参数（n 2）指定行。

例如，PLOT 10,25通知计算机在作图坐标的第10列第25行显示出有色彩的块。

要“关闭”各个作图块，对每个块必须选择颜色为0（黑色）。执行GR语句请除整个屏幕（见GR）。

列号（n 1）的范围可从0到39，行号是从0到47，但只有0到39是在作图区域内。屏幕上作图区域的底下8行是留给TEXT(文本)的。每个文本行需要2行（作图行），在作图区下面可以安排4个文本行。

测试程序

```
10 REM 'PLOT' TEST PROGRAM
20 GR
30 COLOR = 4
40 PLOT 0,0
50 PLOT 39,0
60 PLOT 39,39
70 PLOT 0,39
99 END
```

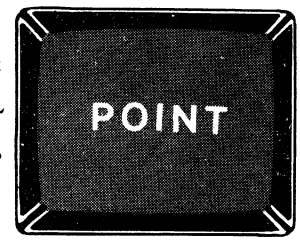
运行实例

如果计算机接受了PLOT语句，在屏幕上的每一个角上将出现一个绿色的点。

参阅

GR, COLOR, TEXT, HLIN-AT, VLIN-AT, SET, RESET, POINT, DRAW

函 数



在TRS—80上POINT函数和IF—THEN语句一起使用来指示指定的作图块是否“打开着”。

作图块由跟在POINT函数后面的括号中的X，Y坐标值来指定。当块亮着时，LEVEL Ⅰ BASIC带回的值是1，LEVEL Ⅱ BASIC带回的值是－1。当块没亮时，二者带回的值都为0。

测试程序

```
10 REM 'POINT' TEST PROGRAM
20 CLS
30 FOR X=20 TO 30 STEP 2
40 SET(X,8)
50 NEXT X
60 PRINT "POINT PASSED THE TEST IF NUMBERS 1010101010
   APPEAR"
70 FOR X=20 TO 30
80 A=0
90 IF POINT(X,8)=1 THEN A=1
100 PRINT A;
110 NEXT X
120 GOTO 120
999 END
```

运行实例（LEVEL Ⅰ BASIC）

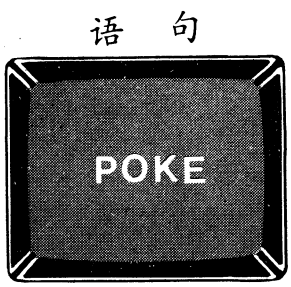
```
POINT PASSED THE TEST IF NUMBERS 1010101010 APPEAR
 1  0  1  0  1  0  1  0  1  0  1  0  1
```

为了使LEVELⅡ获得同样的结果，把90行修改如下：

```
90 IF POINT(X,8)=-1 THEN A=1
```

参阅

SET，RESET，CLS，DOT



POKE语句用来在指定的内存单元中存放0到255（十进制）的整数值。例如，POKE 15360，65 把ASCII码数65（代表字母‘A’）存放到内存地址为15360的单元中去。

在运行下面的测试程序之前，检查你的计算机手册，以确定15360到16383的内存地址在计算机的CRT内存区中，并在存入的时候不致抹去作为其它用途的内存的内容。

测试程序

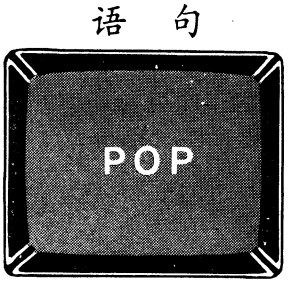
```
10 REM 'POKE' TEST PROGRAM
20 REM USES CRT MEMORY ADDRESSES 15360 TO 16383
30 FOR Y=65 TO 90
40 FOR X=15360 TO 16383
50 POKE X,Y
60 NEXT X
70 NEXT Y
99 END
```

运行实例

如果屏幕上填满了A ～Z 的字母，计算机对POKE的测试就通过了。

参阅

PEEK，FILL，STUFF，EXAM，FETCH



POP是Apple Ⅱ BASIC的一种性能。它把GOSUB行号的地址“上托”，使之顶出存放它的内存栈顶。当程序遇到语句RETURN时，它从栈取地址以决定返回执行的地方。但这时得到的地址并不是最近执行的GOSUB的地址，而是上一个GOSUB语句的地址。下面再说明一下。

每次执行GOSUB时，它的机器语言地址存放在叫做“下推”栈的特殊内存区中。存放在这个栈中的最后一个值是第一个读出并使用的值。RETURN语句读出“在栈上的”顶部地址以便在它的GOSUB完成之后确定控制程序“返回”到什么地方。

在测试程序中，当程序GOSUB到50行时，20行的机器语言地址存放在栈顶。当50行的GOSUB 80被执行时，50行的地址进入栈中并在20行的地址之上。

80行的POP语句把50行的地址“上托”并使其出栈，即把它去掉了。当90行的RETURN语句到栈中找重新恢复执行的地方时，它找到的是20行的地址而不是50行的地址。执行从20行的尾端重新恢复，并且接着执行30行。

当计算结果或者错误条件需要分支到程序的另一个地方时，可使用POP语句，不返回到最近的GOSUB。

测试程序

```
10 REM 'POP' TEST PROGRAM
20 GOSUB 50
30 PRINT"'POP' PASSED THE TEST"
40 GOTO 99
50 GOSUB 80
60 PRINT "'POP' FAILED THE TEST"
70 GOTO 99
80 POP
90 RETURN
99 END
```

运行实例

```
'POP' PASSED THE TEST
```

如果你的计算机没有此功能

如果你的计算机使用POP失败，可利用“标记”来产生类似的结果。

```
10 REM 'POOR MAN'S POP'
15 F=0
20 GOSUB 50
30 PRINT "RETURN TO HERE"
40 GOTO 99
50 GOSUB 80
55 IF F=1 THEN 70
60 PRINT "DON'T PRINT THIS"
70 RETURN
80 F=1
90 RETURN
99 END
```

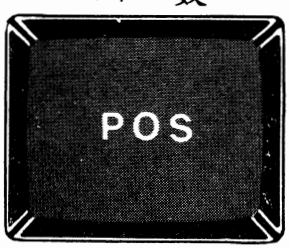
运行实例

```
RETURN TO HERE
```

参阅

GOSUB，RETURN，ON GOSUB，GOSUB OF

函　数



POS（n，A$，B$）是一个串函数，其作用是在A$中把B$第一次出现的起始位置找出来。如果在A$中没有找到B$，POS = 0。

n是可选的，它指示在A$中从第n个字符开始搜索。如果不用n，搜索从A$的第一个字符或最左边的字符开始。

测试程序 # 1

```
10 REM 'POS' TEST PROGRAM
20 A$ = "PROGRAM"
30 B$ = "RAM"
40 K = POS(A$,B$)
50 IF K<>5 THEN 110
60 B$ = "ROM"
70 K = POS(A$,B$)
80 IF K<>0 THEN 110
90 PRINT " POS(A$,B$)  PASSED THE TEST
100 GOTO 30999
110 PRINT "'POS(A$,B$)' FAILED THE TEST"
30999 END
```

运行实例

```
'POS(A$,B$)' PASSED THE TEST
```

测试程序 # 2

```
10 REM 'POS(N,A$,B$)' TEST PROGRAM
20 A$ = "COMPUSOFT"
30 B$ = "O"
40 K = POS(4,A$,B$)
50 IF K=7 THEN 80
60 PRINT "'POS(N,A$,B$)' FAILED THE TEST"
70 GOTO 30999
80 PRINT "'POS(N,A$,B$)' PASSED THE TEST
30999 END
```

运行实例

```
POS(N,A$,B$)' PASSED THE TEST
```

其它用法

某些解释系统（例如，Microsoft BASIC）使用POS(n)函数给出当前打印行中指针的位置。n 的值是无关要紧的——它仅仅是一个“虚”数。

测试程序 # 3

```
10 REM 'POS(N)' TEST PROGRAM
20 PRINT "THIS LINE HAS A CHARACTER COUNT OF";
30 K = POS(N)
40 PRINT K+3
99 END
```

运行实例

```
THIS LINE HAS A CHARACTER COUNT OF 37
```

如果你的计算机没有此功能

如果测试程序# 1 和 # 2 在机器上都通不过，试一下INSTR和INDEX函数，或者使用印在INSTR下面的子程序(见126页)。为了和测试程序一起使用这个子程序，在测试程序 # 1 中做以下修改：

```
35 N = 1
40 GOSUB 30060
70 GOSUB 30060
```

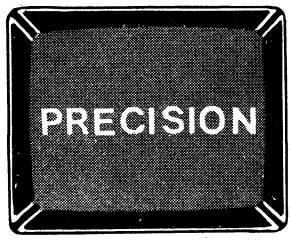
对测试程序 # 2 做以下修改：

```
35 N = 4
40 GOSUB 3 060
```

参阅

INSTR, INDEX

语　句



在TDL BASIC中使用 PRECISION 语句指定用PRINT打印的小数点右侧的最大位数。例如,20PRECISION 2可以在一个打印值代表元和分的程序中使用。如果实际的值比指定的位数长，就在你所要的位数那里进行四舍五入，其右的各位不打印出来。

测试程序

```
10 REM PRECISION TEST PROGRAM
20 PRECISION 4
30 X = 0.1234567
40 PRINT X
50 PRINT "PRECISION PASSED THE TEST IF 0.1235 WAS
   PRINTED"
99 END
```

运行实例

```
 0.1235
PRECISION PASSED THE TEST IF 0.1235 WAS PRINTED
```

如果你的计算机没有此功能

如果在你的计算机上PRECISION语句不能用，请把测试程序中的20行改成DIGITS语句试一下。

把测试程序中的20行删除，把40行修改如下，也可以对小数点后的最大位数进行控制：

```
40 PRINT USING "##.####";X
```

如果PRINT USING也不能用,不要失望! 可用下面的语句代替。

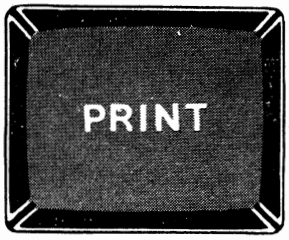
```
40 PRINT INT(X*10000 + .5)/10000
```

参阅

DIGITS, PRINT USING, IMAGE, FMT, INT

命 令

语 句



ANSI

PRI
P.

PRINT的使用范围很广。最普遍的用法是在程序中作为语句，用来打印变量的值或者引号中括起来的任何内容。例如，PRINT X打印变量X的数值，而PRINT“X”则打印字母X。

大部分计算机用PRINT既作为命令（正如你在标准计算器中使用的那样）又作为程序语句。

例如：作为命令，PRINT 4 * 12/(2 + 6) 打印出答案数值6。

测试程序 # 1

```
10 REM 'PRINT' TEST PROGRAM
20 PRINT "THE PRINT STATEMENT WORKS"
99 END
```

运行实例

```
THE PRINT STATEMENT WORKS
```

在PRINT语句中，如果各项用逗号分隔，则在事先指定的水平区内打印，每个打印区占16列宽。打印区的实际宽度因机器而异。

例如，PRINT 1，2，3，4用类似于下面的格式打印：

```
1               2               3               4
```

详见 ，(逗号)。

测试程序 # 2

```
10 REM 'PRINT' WITH COMMA TEST PROGRAM
20 PRINT "THE COMMA WORKED IN THE PRINT STATEMENT
30 PRINT "IF THESE NUMBERS ARE PRINTED IN 4 ZONES
40 PRINT 1,2,3,4
99 END
```

运行实例

```
THE COMMA WORKED IN THE PRINT STATEMENT
IF THESE NUMBERS ARE PRINTED IN 4 ZONES
 1               2               3
```

分号很象逗号，但是用分号分隔各项时，打印出来的值是互相紧挨着的，而不是在预先指定的区域内打印。

将40行修改如下：

```
40 PRINT 1;2;3;4
```

再把此程序运行一下，注意现在打印出来的各项的距离。

在PRINT语句中常常用分号（；）把在一行上的字或句子的各部分连在一起。

例如；PRINT “H”；“I ”，打印出“HI”。

详见 ；(分号)。

测试程序 # 3

```
10 REM  PRINT WITH SEMICOLON TEST PROGRAM
20 PRINT "IS THIS PRINTED ";
30 PRINT "ON ONE LINE?"
99 END
```

运行实例

```
IS THIS PRINTED ON ONE LINE?
```

TAB（n）与PRINT语句一起使用，它类似于打字机上的tab键。在指定了括号中的值后，在打印前插入(n个)空格。详见TAB。

在TRS—80 Level I BASIC中AT函数（在TRS—80 Level Ⅱ BASIC中使用@操作符）与 PRINT语句一起用来指定PRINT语句的起始位置。详见PRINT AT和@。

某些计算机使用PRINT USING作为特殊的PRINT功能，它允许使用指定的格式打印数或串。

例如：PRINT USING “**####，##”；12.5打印出***12.50

详见PRINT USING。

某些BASIC（例如 North Star）在PRINT后面直接跟格式说明信息。例如；执行

PRINT %C10F2, P 时打印出P 的值共占10列宽度。其中有一小数点，后有二位小数（10F 2）。

C使得每三位插入一个逗号。如果P = 12345.678，打印出来的P的形式为12,345.68。

其它的格式选择如下：

$（打印前导的美元符号），

Z（抑制尾随的零），

wEd（表示常有d 位小数的指数形式的数，w至少为（d + 8），

nI（最大为n 位的整型数）。

MAT PRINT打印存放在数组变量中的值。

例如：

```
10 DIM A(3)
20 MAT PRINT A
```

将打印出赋给数组元素 A（1），A（2）和 A（3）的值。详见MAT PRTNT。

在TRS—80 LEVEL I BASIC中使用PRINT#把数据存放到盒式磁带上。为了用一个PRINT#语句存放一个以上的值，使用下面的格式：

PRINT#A；“，”；B；“，”；C 等等。

为了测试这个性能，让盒式录音机处于录音状态并运行下面的程序。

测试程序# 4

```
10 REM 'PRINT#' TEST PROGRAM
20 PRINT "DATA SHOULD BE RECORDING ON CASSETTE TAPE"
30 A$="TEST"
40 PRINT# A$;",";1;",";2;",";3
50 PRINT "PRINT# HAS COMPLETED THE DATA TRANSFER"
99 END
```

运行实例

```
DATA SHOULD BE RECORDING ON CASSETTE TAPE
PRINT# HAS COMPLETED THE DATA TRANSFER
```

当使用一个录音机时，改进的 TRS—80 BASIC 需要在PRINT#后面跟一个 -1。如果使用第二个录音机，则通过PRINT#-2 来存取（等等）。

例如；PRINT#-1,A,B,C$把变量A，B 和C$的值存放在设备号为# 1的磁带上。

测试程序# 5

让盒式录音机处于录音状态并运行这个程序。

```
10 REM 'PRINT#' TEST PROGRAM
20 PRINT "DATA SHOULD BE RECORDING ON CASSETTE TAPE"
30 PRINT#-1,"TEST",1,2,3
40 PRINT "PRINT#-1 HAS COMPLETED THE DATA TRANSFER"
99 END
```

运行实例

```
DATA SHOULD BE RECORDING ON CASSETTE TAPE
PRINT#-1 HAS COMPLETED THE DATA TRANSFER
```

为了检查已存放好的数据，将磁带倒回，使录音机处于PLAY状态并运行以下程序。

```
10 REM * INPUT DATA FROM CASSETTE*
20 PRINT "THE COMPUTER SHOULD BE READING DATA FROM
   CASSETTE"
30 INPUT#-1,A$,A,B,C
40 PRINT "THE FOLLOWING DATA WAS READ FROM THE CASSETTE"
50 PRINT A$,A,B,C
99 END
```

运行实例

```
THE COMPUTER SHOULD BE READING DATA FROM CASSETTE
THE FOLLOWING DATA WAS READ FROM THE CASSETTE
TEST                1                2                3
```

通常带有文件处理功能的微型计算机和大型计算机使用PRINT#，将数据存贮在诸如磁盘或盒式磁带这样的外部设备“文件”上。每个数据文件由一个数字(文件名)来标识。

这个数字应写在 PRINT#语句中，它用来指定把数据存放到哪个设备上。数据可由数值或字符串组成。

例如： PRIN# 3 ;A ,B ,“TESTING”在名为#3的文件上存放变量A、变量B 的内容以及字 “TESTING” 。使用FICE#, I N PUT#和READ#被用来指定文件名和数据存贮空间，并从文件存贮器读回数据。

替代的形式

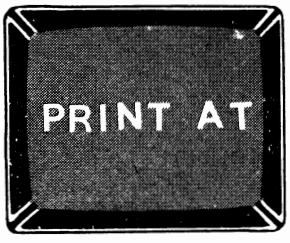
少数计算机允许关键字PRINT用简写形式。PDP—8E使用PRI, Britain's Acorn ATOM和TRS—80 LEVEL I 以及 Tiny BASIC 接受P.。

此外，若干BASIC允许用一个字符代替PRIN T。Microsoft BASIC使用 ?，North Star BASIC使用 !，DEC BASIC—PLUS 使用&，Sweden ABC—80使用 ;、Digital Group's Maxi BASIC 使用 # 。

参阅

TAB，AT，@，PRINT USING，MAT PRINT，#，，(逗号)，；(分号)，CUR，LIN，LPRINT，%，?，&，!

语 句



PRI NT@
P. A.

TRS—80 LEVEL I BASIC 使用PRINT AT 指出PRINT语句的起始位置。AT值可以是数值，数值变量或者算术表达式。在AT值和字符串之间必须用逗号或分号分隔。屏幕上有1024个位置，分为16行，每水平行上有64个地址。

例如：

```
10 PRINT AT 420, "HELLO"
20 PRINT AT (420);"HELLO"
```

二行都在420的位置上打印出字HELLO。括号是可选的。详见AT。

测试程序

```
10 REM 'PRINT AT' TEST PROGRAM
20 PRINT AT 128,"2. IF THIS LINE IS PRINTED AFTER LINE 1."
30 PRINT AT 0,"1. THE 'PRINT AT' STATEMENT PASSED THE TEST"
40 GOTO 40
99 END
```

运行实例

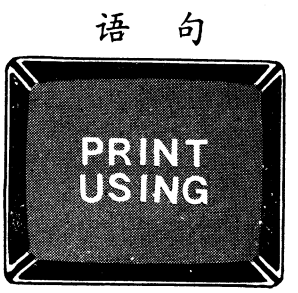
```
1. THE 'PRINT AT' STATEMENT PASSED THE TEST
2. IF THIS LINE IS PRINTED AFTER LINE 1.
```

替代的形式

Microsoft BASIC用PRINT@来代替PRINT AT，并要求在它的后面有一个逗号。Tiny BASIC 接受P.A.。

参阅

PRINT, AT, @, TAB



所有大小不同的计算机都使用PRINT USING作为PRINT的特殊功能，它允许使用可变格式打印数或字符串。PRINT USING语句是在BASIC中可使用的功能最强的(和最复杂的)打印语句，所以我们在这里一一介绍它的功能，一次介绍一个。并不是每个性能在每种计算机上都有的，但是测试程序将很快让你识别出你的计算机上能做些什么。你的计算机还具有什么其它性能可参看你的计算机使用手册。

用英磅符号（#）为一个数或数值变量中的小数点左边和右边的数值保留位置。

如果小数点的右侧没有任何数，自动地插入零。这对打印财务上的报表是很重要的。#号总是在每一个数中的同一个位置上打印小数点，这样就比较容易检查各行的数。详见#。

测试程序# 1

```
10 REM 'PRINT USING' TEST PROGRAM
20 PRINT "THE # OPERATOR PASSED THE PRINT USING TEST"
30 PRINT "IF THE FOLLOWING NUMBERS ARE PRINTED"
40 FOR X=1 TO 5
50 READ N
60 PRINT USING "########.##";N
70 NEXT X
80 DATA 1.2,400,2400000,82450.5,-.25
99 END
```

运行实例

```
THE # OPERATOR PASSED THE PRINT USING TEST
IF THE FOLLOWING NUMBERS ARE PRINTED
       1.20
     400.00
2400000.00
  82450.50
     -0.25
```

在#号前面放二个星号（**），在指定数值的小数点左边的所有没有用到的位置上都打印出星号(*)。它的主要目的是防止有人在用计算机打印出的支票上加大数值。

例如，PRINT USING "**####, ##"; 234.25将打印出***234.25。对测试程序#1做如下修改来测试这个性能。

```
20 PRINT "THE ** OPERATOR PASSED THE PRINT USING TEST"
60 PRINT USING "*********.##";N
```

运行实例

```
THE ** OPERATOR PASSED THE PRINT USING TEST
IF THE FOLLOWING NUMBERS ARE PRINTED
*******1.20
*****400.00
*2400000.00
***82450.50
******-0.25
```

在#号前插入二个美元符号（$$），能在用PRINT USING语句打印出来的数前打印一个$符号。

例如,PRINT USINSG"$$###.##";1.25将打印出$1.25。为了能在你的计算机上测试这个性能，对测试程序# 1做以下修改：

```
20 PRINT "THE $$ OPERATOR PASSED THE PRINT USING TEST"
60 PRINT USING "$$########.##";N
```

运行实例

```
THE $$ OPERATOR PASSED THE PRINT USING TEST
IF THE FOLLOWING NUMBERS ARE PRINTED
       $1.20
     $400.00
$2400000.00
   $82450.50
      -$0.25
```

如果在小数点左边的#号之间的任意位置加入一个或几个逗号，可以使小数点左边的数每三位加一个逗号。PRINT USING语句中逗号的位置对所打印出来的逗号的位置不起作用。

例如：

```
PRINT USING "#,####.##";12000
PRINT USING "####,#.##";12000
PRINT USING "#,##,#,#.##";12000
```

每一个语句打印出来的数都为12,000.00 。

为了测试这个特性，对测试程序= 1作如下修改。

```
20 PRINT "PRINT USING 'COMMA' PASSED THE TEST"
60 PRINT USING "#,########.##";N
```

运行实例

```
PRINT USING 'COMMA' PASSED THE TEST
IF THE FOLLOWING NUMBERS ARE PRINTED
         1.20
       400.00
2,400,000.00
   82,450.50
      -0.25
```

在#号的最左边放一个+号,可以在每个正数前面打印出一个“+”号，在每个负数前面打印出一个“-”号。如果+号放在#号的右面，凡遇负数在它的最后打印出一个“-”号，凡遇正数在它的最后插入一个空格。

例如：

```
PRINT USING "+####";123
PRINT USING "####+";-123
```

将打印出

```
+123

123-
```

对测试程序＃1作如下修改，以便测试此性能。

```
20 PRINT "THE + OPERATOR PASSED THE PRINT USING TEST"
60 PRINT USING "+########.##";N
```

运行实例

```
THE + OPERATOR PASSED THE PRINT USING TEST
IF THE FOLLOWING NUMBERS ARE PRINTED
      +1.20
     +400.00
+2400000.00
  +82450.50
      -0.25
```

在#号后面可以跟四个指数符号(^^^^)，使打印出来的数用指数或科学记数法的形式表示。少数计算机（例如TRS—80）用↑↑↑↑来代替。

例如：PRINT USING“##^^^^”；100打印出来的数表示成1E+02。

测试程序＃2

```
10 REM 'PRINT USING EXPONENTIATION' TEST PROGRAM
20 PRINT "PRINT USING '^^^^^' PASSED THE TEST"
30 PRINT "IF THE NUMBER";123456
40 PRINT "IS PRINTED USING SCIENTIFIC NOTATION"
50 PRINT USING "##^^^^";123456
99 END
```

运行实例

```
PRINT USING '^^^^^' PASSED THE TEST
IF THE NUMBER 123456
IS PRINTED USING SCIENTIFIC NOTATION
 1E+05
```

有些计算机（例如，使用Microsoft BASIC各种版本的机器）使用!(用引号括起来)，它把PRINT USING语句中所列出的串变量或字符串中最左边的一个字符打印出来。

例如：

PRINT USING“!”；“WORD”打印出字母W。

测试程序# 3

```
10 REM 'PRINT USING !' TEST PROGRAM
20 PRINT "ENTER A SAMPLE WORD";
30 INPUT A$
40 PRINT "THE PRINT USING STATEMENT AND THE ! OPERATOR"
50 PRINT "PASSED THE TEST IF THE FIRST LETTER IN ";A$;" IS ";
60 PRINT USING "!";A$
99 END
```

运行实例（输入HANDBOOK）

```
ENTER A SAMPLE WORD? HANDBOOK
THE PRINT USING STATEMENT AND THE ! OPERATOR
PASSED THE TEST IF THE FIRST LETTER IN HANDBOOK IS H
```

使用\\(反斜杠)只把串中最左边的字符打印出来。打印的字符数由两个＼号中间的空格数来决定。计算机把两个＼号也算作字符位置，　因此，通过 \\至少指定二个字符。

例如：PRINT USING“\ \”;“COMPUSOFT”打印出前三个字符COM,因为两个＼符号中包含一个空格（1空格＋2反斜杠＝3字符）。TRS—80使用%号来代替＼符号。

测试程序# 4

```
10 REM 'PRINT USING \' TEST PROGRAM
20 A$ = "TESTIFIED"
30 PRINT "THE PRINT USING STATEMENT ";
40 PRINT "AND THE \ OPERATOR PASSED THE ";
50 PRINT USING "\   \";A$
99 END
```

运行实例

```
THE PRINT USING STATEMENT AND THE \ OPERATOR
  PASSED THE TEST
```

大部分计算机允许将PRINT USING中的操作符,数和串指定为变量。如下例中10行包含了打印格式，称为映象行。

例如：

```
10 A$="!"
20 B$="ABCD"
30 PRINT USING A$;B$
```

将把串变量B$中最左边的一个字母A打印出来。

测试程序#5表示三种不同的PRINT格式可以通过分号连在一起。

测试程序 # 5

```
10 REM 'PRINT USING VARIABLES' TEST PROGRAM
20 A$="**$#####.##"
30 B$="\   \"
40 C$="TESTIMONIAL"
50 A=19.95
60 PRINT "THE BASIC HANDBOOK PASSED THE ";
70 PRINT USING A$;A;
80 PRINT USING B$;C$
99 END
```

运行实例

```
THE BASIC HANDBOOK PASSED THE ****$19.95 TEST
```

把上例中的整个格式放在一个映象行中,把要输出的所有变量放在另一行中,可以产生同样的结果。把测试程序#5 中的 30行和80行删掉，并修改20行和70行如下：

```
20 A$="**$#####.## \   \"
70 PRINT USING A$;A,C$
```

或者更好一点的话，可删除60行和80行，並作如下修改：

```
20 A$="\                                    \**$#####.## \   \"
30 B$="THE BASIC HANDBOOK PASSED THE "
70 PRINT USING A$;B$,A,C$
```

（注意,除了PRINT USING的指定的格式之外，所有项的后面的分号可用逗号来代替。）当然，串映象行必须在它相应的PRINT USING 行之前执行。

其它用法

某些计算机（例如：DEC—10 和Sperry / Unirac VS/ 9）要求：当在一行上输出多个变量并且PRINT USING 格式在一个映象行中时，必须给映象行指定一个行号，并且以一个冒号开始。然后在PRINTUSING 语句中引用这个行号作为格式。

例如：

```
60 A = 12.34
70 B = 56.78
80 C$ = "MAIN FRAME"
90 PRINT USING 100,A,B,C$
100 :###.## $$$$.## 'CCCCCCCCC
```

将打印出：

```
12.34  $56.78 MAIN FRAME
```

测试程序＃6

```
10 REM 'PRINT USING LINE NUMBER' TEST PROGRAM
20 PRINT "THE PRINT USING STATEMENT PASSES THE TEST"
30 PRINT "IF THE NUMBER 125.50 IS PRINTED NEXT"
40 PRINT USING 50,125.5
50 :###.##
99 END
```

运行实例

```
THE PRINT USING STATEMENT PASSES THE TEST
IF THE NUMBER 125.50 IS PRINTED NEXT
125.50
```

如果你的计算机没有此功能

在没有PRINT USING的BASIC中，仍可使用其它方法来实现PRINT USING的某些功能。例如：

```
90 PRINT USING "######.###";X
```

打印X的值，在三位小数之后进行四舍五入。可以用DIGITS来代替。（见DIGITS。）

如果DIGITS不能用，通过下面的方法可获得同样的结果

```
90 PRINT INT (X*1000+.5)/1000
```

要确定小数点在数的哪一列上是比较困难的。为了重现测试程序＃1 中PRINT USING语句的效果，我们要确定在小数点之前打印出多少位。自然对数在这里可帮助我们确定要打印的位数（见LOG10）。LOG10（10）＝1，LOG 10（100）＝2，LOG 10（1000）＝3…。因此，为了产生与下面语句同样的结果，60 PRINT USING“########.##”；N 可以用下面语句来代替。

```
56 T=0
57 K=ABS(N)
58 IF K<.1 THEN 60
59 T=LOG10(K)
60 PRINT TAB(8-T); SGN(N)*INT(K*100+.5)/100
```

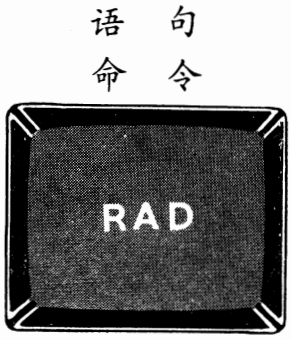
甚至浮动的 $ 符号也可以包括进来。

```
60 PRINT TAB(7-T); "$"; SGN(N)*INT(K*100+.5)/100
```

参阅

PRINT，＃，＃＃，！，↑，+，_，％，IMAGE，FMT，DIGITS，\（反斜杠），$，&，：

在少数计算机上(例如,Cromemco 16K 扩展BASIC)使用RAD以弧度代替角度进行三角计算。在接通电源时大部分计算机在弧度方式下工作，但是有些机器也有用角度计算三角函数的功能。如果在程序中已使用了DEG，需要用RAD使计算机恢复到“正常”方式。1 弧度近似等于57度。



RADIAN

测试程序#1

```
10 REM 'RAD' TEST PROGRAM
20 DEG
30 D = SIN(30)
40 PRINT "THE SINE OF 30 DEGREES IS";D
50 RAD
60 R = SIN(30)
70 PRINT "THE SINE OF 30 RADIANS IS";R
99 END
```

运行实例

```
THE SINE OF 30 DEGREES IS 0.5
THE SINE OF 30 RADIANS IS -0.988032
```

替代的形式

某些计算机（例如，Sharp /TRS—80 Pocket）用RADIAN语句使计算机以弧度方式进行三角计算。

其它用法

少数BASIC（例如MAX BASIC)使用RAD(n)作为把值(n)从角度化成弧度的函数。

测试程序#2

```
10 REM 'RAD FUNCTION' TEST PROGRAM
20 PRINT "ENTER AN ANGLE MEASURE (IN DEGREES)";
30 INPUT D
40 R = RAD(D)
50 PRINT "A MEASURE OF";D;" DEGREES IS EQUAL TO";R;"
   RADIANS"
99 END
```

运行实例

```
ENTER AN ANGLE MEASURE (IN DEGREES) ?45
A MEASURE OF 45 DEGREES IS EQUAL TO 0.785398 RADIANS
```

如果你的计算机没有此功能

如果你的计算机没有RAD函数，可以用0.0174533乘上角度值来代替。为了在测试程序2中使用这个转换系数，用下面的语句代替40行。

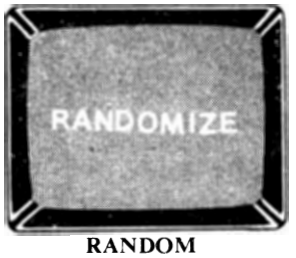
```
40 R = D * 0.0174533
```

参阅

DEG，GRAD，ACS，ASN，ATN，COS，SIN，TAN

语　句

ANSI



RANDOM
RAN

使用RANDOMIZE来使计算机产生的随机数序列随机化。建立这个序列是为了使用RND函数时的需要。

程序中使用RND函数之前用RANDOMIZE，以保证在每次运行程序时，为RND函数产生一组新的随机数序列。

测试程序

```
10 REM 'RANDOMIZE' TEST PROGRAM
20 RANDOMIZE
30 FOR X=1 TO 8
40 PRINT RND,
50 NEXT X
99 END
```

运行实例（典型的）

```
 .250186        .975707        .775985        .544615
 .890564        .227299        .406976        .771341
```

每运行一次测试程序，就打印出一组新的随机数。在试运行有20行语句的测试程序之前，要搞清楚你所用的版本中RND的情况。

某些BASIC（例如，BASIC—80）要求在RANDOMIZE语句中包含一个“种子”值。例如，RANDOMIZE 49817。如果不包括“种子”值，程序中止并提醒用户从键盘敲入一个“种子”值。

替代的形式

若干计算机用关键字RANDOM（例如，TRS—80）代替RANDOMIZE。还有一些（例如，PDP—8 E）则接受RAN作为RANDOMIZE的缩写形式。

如果你的计算机没有此功能

如果你的计算机上没有RANDOMIZE，可以在RND中使用一个负数来重新设置“种子”或产生一个新的随机序列（见RND）。如果不是这种情况，可使用一个简单的过程这个过程对每个程序的每次运行都能产生新的随机序列。

用下面的程序代替测试程序中的20行：

```
20 PRINT "ENTER A WHOLE NUMBER 'SEED' FOR RND";
22 INPUT S
24 FOR I=1 TO S
26 X=RND          'or RND(0)
28 NEXT I
```

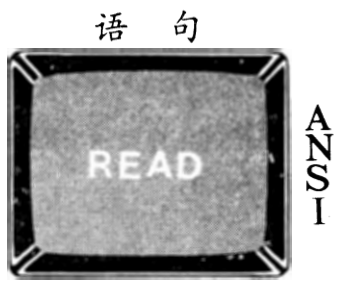
使程序运行若干次，每次输入不同的数。原因是计算机开机时每次产生同一个随机序列（在某些计算机上每次打入RUN时产生同一个随机序列）。实际上，每次运行程序时，RND从随机数表的顶部“读”数。我们的过程使得程序把表的顶部若干数“扔掉”，并开始从第（S+1）项开始读数。这样在每一次输入不同的S值时，就可以得到不同的起始点。如果想在每次运行时都得到同一个随机序列，可以在每次运行时使用相同的种子值S。

参阅

RND

READ语句的作用是：从DATA行中读数据并将这些数据赋给READ语句中的变量。

每次执行READ语句时，从DATA行中读数据。然后指针移到DATA行中下一个数据项，等待另一个READ语句去读。当所有的DATA语句中的全部数据读完时，在执行另外的READ语句之前，必须把指针重新设置在DATA表的开始。（见RESTORE）。



REA.
REA

测试程序 # 1

```
10 REM 'READ' STATEMENT TEST PROGRAM
20 READ A
30 PRINT "THE READ STATEMENT WORKED IN LINE";A
40 DATA 20
99 END
```

运行实例

```
THE READ STATEMENT WORKED IN LINE 20
```

因为计算机允许在一个READ语句中放置一个以上的变量，所以必须用逗号把这些变量分隔开，并且DATA的元素个数必须大于或等于READ表列中变量的个数。

测试程序 # 2

```
10 REM 'MULTIPLE READ' STATEMENT TEST PROGRAM
20 READ A,B,C
30 D=A+B+C
40 PRINT "D=";D
50 PRINT "THE READ STATEMENT PASSED THE TEST IF D = 60"
60 DATA 10,20,30
99 END
```

运行实例

```
D = 60
THE READ STATEMENT PASSED THE TEST IF D = 60
```

大部分计算机也允许从DATA语句中读字符串。每次从DATA语句中读一个串，在READ语句中必须有一个相应的串变量。

测试程序 # 3

```
10 REM 'READ STRINGS' TEST PROGRAM
20 READ D$
30 PRINT "THE READ STATEMENT PASSED THE ";D$
40 DATA TEST
99 END
```

运行实例

```
THE READ STATEMENT PASSED THE TEST
```

许多计算机允许同一个READ语句既读数值又读字符串，这些不同类型的数据可在同一个DATA行中。

测试程序 # 4

```
10 REM 'MULTIPLE READ' STATEMENT TEST PROGRAM
20 READ A,B,C,D$
30 D=A+B+C
40 PRINT "THE READ STATEMENT PASSED THE TEST IN ";D$;D
50 DATA 2,8,10,LINE
99 END
```

运行实例

```
THE READ STATEMENT PASSED THE TEST IN LINE 20
```

替代的形式

某些计算机允许READ的缩写形式。使用Palo Alto Tiny BASIC各种版本的计算机接受REA.作为READ的缩写。而另外一些（例如，PDP—8 E）则接受REA。

参阅

DATA，RESTORE，,(逗号).

语 句
命 令

APPLEⅡ计算机用RECALL作为命令和语句。其作用是从盒式磁带上输入一个数值型数组。在程序控制下，可以把一个大数组存放在磁带上，然后再通过同一个程序或者别的程序调用（RECALL）它。详见STORE。

例如：

```
10 DIM A(3,3,3)
.
200 RECALL A
```

把原先存放在磁带上的64个值（4 × 4 × 4）读出来，以便在后面使用（注意，每维有四个值，下标为0，1，2，3）。

测试程序

```
10 REM 'RECALL' TEST PROGRAM
20 DIM A(25),B(25)
30 FOR I=1 TO 25
40 A(I)=I
50 NEXT I
60 STORE A
70 PRINT "REWIND TAPE AND SET TO PLAY - PRESS RETURN"
80 INPUT A$
90 RECALL B
100 FOR I=1 TO 25
110 PRINT B(I),
120 NEXT I
999 END
```

运行实例

```
REWIND TAPE AND SET TO PLAY - PRESS RETURN
?
 1              2              3
 4              5              6
 7              8              9
 10             11             12
 13             14             15
 16             17             18
 19             20             21
 22             23             24
 25
```

参阅

STORE, CLOAD, DIM

语　句

ANSI

REMARK

REM语句用来表达一个程序或部分程序行的内容，如同一个“备忘录”或“便笺”一样。REM是一个非执行语句。用REM开始的行被计算机忽略(即不被执行)。

如果在多语句行中使用，REM语句前的那些语句句将被执行，但是它后面的语句被忽略。如果需要的注释行多于一个程序行，每一个这样行都要用REM开始。

测试程序

```
10 PRINT "'REM' TEST PROGRAM"
20 REM PRINT "REM FAILED THE TEST"
30 REM * REM FAILED THE TEST IF LINE 20 IS PRINTED*
40 PRINT "REM PASSED THE TEST"
99 END
```

运行实例

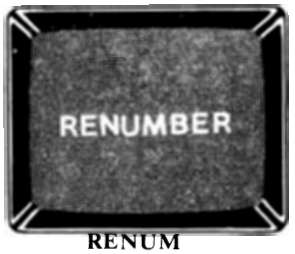
```
'REM' TEST PROGRAM
REM PASSED THE TEST
```

有些计算机允许REM或REM　ARK二种写法，而其它计算机只允许其中一种。

参阅

(撇号)，!

命　令



RENUM

REN

一些计算机（例如，Cromemco 16K扩展BASIC）使用RENUMBER改变程序的行号。在同一程序中的GOTO, GOSUB, IF-THEN, ON-GOTO和ON-GOSUB语句中使用的行号也相应自动改变，以便和未重新编行号前的分支转移效果相同。

如果在RENUMBER语句中不写数值，计算机自动地以10作为起始行号，以10为增量值为每个程序行重新编号。

测试程序

```
2 REM "RENUMBER" TEST PROGRAM
3 X=1
4 PRINT "IF EACH PROGRAM LINE ";
5 GOTO 10
6 PRINT "THE RENUMBER COMMAND ";
7 X=2
8 GOTO 12
10 PRINT "IS RENUMBERED"
12 ON X GOTO 6,14
14 PRINT "PASSED THE TEST."
16 END
```

运行一下，以了解它是怎样工作的。

打入RENUMBER命令，并使之再一次运行。

运行实例

```
IF EACH PROGRAM LINE IS RENUMBERED
THE RENUMBER COMMAND PASSED THE TEST.
```

为了证实程序被重新编排行号，列出的程序清单，应如下所示：

```
10 REM "RENUMBER" TEST PROGRAM
20 X=1
30 PRINT "IF EACH PROGRAM LINE ";
40 GOTO 80
50 PRINT "THE RENUMBER COMMAND ";
60 X=2
70 GOTO 90
80 PRINT "IS RENUMBERED"
90 ON X GOTO 50,100
100 PRINT "PASSED THE TEST."
110 END
```

如果使用RENUMBER n，指明被重新编号的程序的开始行号为n，行号的增值为10。为了在测试程序上测试这个性能，打入RENUMBER 20，并列出程序清单如下：

```
20 REM "RENUMBER" TEST PROGRAM
30 X=1
40 PRINT "IF EACH PROGRAM LINE ";
50 GOTO 90
60 PRINT "THE RENUMBER COMMAND ";
70 X=2
80 GOTO 100
90 PRINT "IS RENUMBERED"
100 ON X GOTO 60,110
110 PRINT "PASSED THE TEST."
120 END
```

用RENUMBER n1,n2 指明被重新编号的程序的开始行号为n1，行号的增量为n2。为了在测试程序上测试这个性能，打入RENUMBER50,20，列出程序清单。现在它应如下所示：

```
50 REM 'RENUMBER' TEST PROGRAM
70 X=1
90 PRINT "IF EACH PROGRAM LINE ";
110 GOTO 190
130 PRINT "THE RENUMBER COMMAND ";
150 X=2
170 GOTO 210
190 PRINT "IS RENUMBERED"
210 ON X GOTO 130,230
230 PRINT "PASSED THE TEST."
250 END
```

用RENUMBER n1, n2, n3指明从老行号n3开始对程序重新编号，行号n3重新指定为n1，行号增值为n2。为了在测试程序上测试这个性能，从键盘上打入RENUMBER 500，10，90并列出程序清单。应如下面这个清单所示：

```
50 REM 'RENUMBER' TEST PROGRAM
60 X=1
65 PRINT "IF EACH PROGRAM LINE ";
70 GOTO 550
520 PRINT "THE RENUMBER COMMAND ";
530 X=2
540 GOTO 560
550 PRINT "IS RENUMBERED"
560 ON X GOTO 520,570
570 PRINT "PASSED THE TEST."
580 END
```

少数计算机使用RENUMBER n1, n2, n3, n4，对当前程序中n3到n4的行重新编号。新的行号从n1开始，增量为n2。打入RENUMBER 60，5，70，510来测试这个性能，打印出程序清单如下：

```
50 REM 'RENUMBER' TEST PROGRAM
60 X=1
65 PRINT "IF EACH PROGRAM LINE ";
70 GOTO 550
520 PRINT "THE RENUMBER COMMAND ";
530 X=2
540 GOTO 560
550 PRINT "IS RENUMBERED"
560 ON X GOTO 520,570
570 PRINT "PASSED THE TEST."
580 END
```

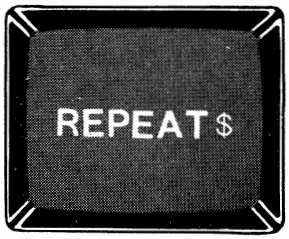
替代的形式

某些计算机接受RENUMBER的缩写形式，例如RENUM和REN。TRS—80 Disk BASIC使用NAME。DEC计算机使用RESEQUENCE。

参阅

GOTO，GOSUB，IF-THEN，ON-GOTO，ON-GOSUB，LIST

函 数



REPEAT$函数用来建立一个字符串，它由一个给定的字符组重复若干次构成。例如，REPEAT$("MICRO", 4）产生的字符串为MICROMICROMICROMICRO，而REPEAT$（"*"，16）则产生16个*号：**********。

测试程序

```
10 REM 'REPEAT$' TEST PROGRAM
20 PRINT "TYPE YOUR FIRST NAME";
30 INPUT N$
40 PRINT "NOW PICK A NUMBER FROM 2 TO 5";
50 INPUT M
60 A$ = REPEAT$(N$,M)
70 PRINT A$
99 END
```

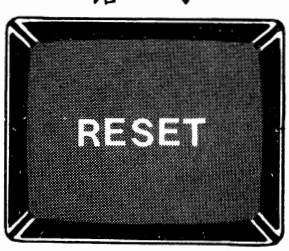
运行实例

```
TYPE YOUR FIRST NAME ? DAVID
NOW PICK A NUMBER FROM 2 TO 5 ? 3
DAVIDDAVIDDAVID
```

参阅

STRING$

语 句



TRS—80用RESET语句"关闭"屏幕上原先已打开的作图块。被"关闭"的作图块是由RESET语句后面的括号内的X，Y坐标来指定的。例如，RESET（5，8）通知计算机关闭第5列第8行位置上的作图块。

如果要打开作图块，请看SET。

R.

测试程序

```
10 REM 'RESET' TEST PROGRAM
20 CLS
30 Y=1
40 FOR X=1 TO 100
50 SET (X,Y)
60 NEXT X
70 PRINT
80 PRINT "RESET PASSED THE TEST IF THE LINE DISAPPEARS"
90 FOR X=1 TO 100
100 RESET (X,Y)
110 NEXT X
999 END
```

运行实例

```
RESET PASSED THE TEST IF THE LINE DISAPPEARS
```

替代的形式

在LEVEL I和Tiny BASIC中使用R. 作为RESET的缩写形式。

参阅

SET, CLS, POINT, CLRDOT

语 句

ANSI

REST.
RES

RESTORE语句的执行使得DATA指针“重新置位”到第一个DATA行中第一个数据前。这样计算机就能通过此语句多次调用存放在DATA语句中的数据。

测试程序 # 1

```
10 REM 'RESTORE' TEST PROGRAM
20 READ X
30 IF X=3 THEN 50
40 GOTO 20
50 RESTORE
60 READ X
70 IF X=1 THEN 100
80 PRINT "RESTORE FAILED THE TEST"
90 GOTO 999
100 PRINT "RESTORE PASSED THE TEST"
110 DATA 1,2,3
999 END
```

运行实例

```
RESTORE$ PASSED THE TEST
```

替代的形式

少数计算机（例如TRS—80LEVEL i BASIC）允许把RESTORE写成REST的形式。其它的（例如DEC PDP— 8 E）接受RES 作为RESTORE的缩写形式。

其它用法

有些解释系统允许在RESTORE语句后面加一个DATA语句的行号，只把指定的DATA行的指针重新置位在该行的第一个数据前。请看测试程序# 2 中的100行。

测试程序# 2

```
10 REM 'RESTORE (LINE#)' TEST PROGRAM
20 READ X
30 PRINT X;
40 IF X=3 THEN 60
50 GOTO 20
60 READ X
70 PRINT X;
80 IF X=6 THEN 100
90 GOTO 60
100 RESTORE 180
110 READ X
120 IF X=4 THEN 150
130 PRINT "RESTORE FAILED THE TEST"
140 STOP
150 PRINT "RESTORE PASSED THE TEST"
160 GOTO 999
170 DATA 1,2,3
180 DATA 4,5,6
999 END
```

运行实例

```
 1  2  3  4  5  6 RESTORE PASSED THE TEST
```

有些计算机（例如DEC—10）能够分别恢复数值型数据和字符串数据。嵌入在程序中的RESTORE$只能恢复字符串数据。在重新使用数值型数据时用RESTORE *。

测试程序# 3

```
10 REM RESTORE$ TEST PROGRAM
20 READ X$
30 READ N
40 RESTORE$
50 READ T$
60 READ N
70 IF N=2 THEN 100
80 PRINT "IT DIDN'T WORK"
90 GOTO 999
100 PRINT "RESTORE$ PASSED THE ";T$
110 DATA TEST, 1, 2
999 END
```

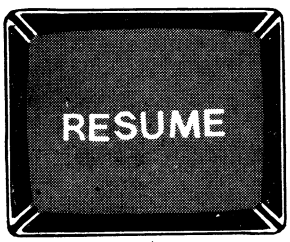
运行实例

```
RESTORE$ PASSED THE TEST
```

参阅

DATA，READ

语　句



使用RESUME语句作为ON—ERROR—GOTO子程序中的最后一个语句，通知计算机在指定的行号重新开始程序的执行。如果在RESUME语句之前没有ON—ERROR—GOTO语句，计算机不允许执行RESUME语句。对于测试程序使用RESUME（行号）的情况请见ON—ERROR—GOTO。(为了节省篇幅在此不重复了。)

使用RESUME NEXT语句转移到出错的下一行并继续程序的执行。为了在你的机器上测试RESUME NEXT的性能，在ON—ERROR—GOTO测试程序中把110行修改如下：

```
110 RESUME NEXT
```

运行实例(ON－ERROR－GOTO测试程序 使用RESUME NEXT）（输入0）

```
ENTER A NUMBER AND IT'S INVERSE WILL BE COMPUTED? 0
THE INVERSE OF 0 CANNOT BE COMPUTED - TRY AGAIN
THE INVERSE OF 0 IS 0
?
```

使用RESUME 0 和RESUME（没有行号或NEXT）转移到出错语句。

测试程序

```
10 REM 'RESUME' TEST PROGRAM
20 ON ERROR GOTO 100
30 PRINT "ENTER A POSITIVE NUMBER";
40 INPUT N
50 A=LOG(N)
60 PRINT "THE LOG OF";N;"IS";A
70 GOTO 30
100 PRINT "A NEGATIVE NUMBER IS NOT ALLOWED"
110 N=N*-1
120 RESUME 0
999 END
```

运行实例（输入－4）

```
ENTER A POSITIVE NUMBER? -4
A NEGATIVE NUMBER IS NOT ALLOWED
THE LOG OF 4 IS 1.38629
ENTER A POSITIVE NUMBER?
```

为了在你的计算机上测试RESUME（没有行号或没有NEXT）的性能，把上面的测试程序中120行修改如下：

```
120 RESUME
```

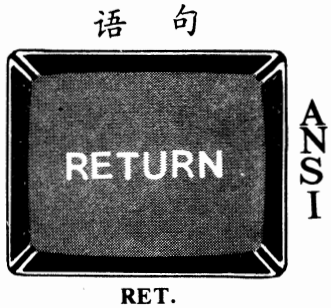
运行此测试程序，运行结果应没有变化。

参阅

ON-ERROR-GOTO，ERL，ERR

RETURN语句与GOSUB语句是联合使用。它作为子程序中最后一个语句，通知计算机返回到含有 GOSUB语句的行，并且从此处继续程序的执行。

如果在RETURN语句之前没有GOSUB语句，计算机不执行RETURN语句。



RET.
RET
R.

测试程序

```
10 REM 'RETURN' STATEMENT TEST PROGRAM
20 GOSUB 50
30 PRINT "WAS ACCEPTED."
40 GOTO 99
50 PRINT "THE RETURN STATEMENT ";
60 RETURN
70 PRINT "WAS NOT ACCEPTED."
99 END
```

运行实例

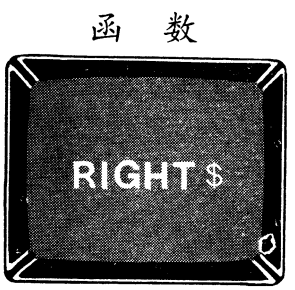
```
THE RETURN STATEMENT WAS ACCEPTED.
```

替代的形式

有几种缩写形式，它们是RET.（TRS—80 LEVEL Ⅰ），RET（DEC PDP－8 E）和R.（Acorn ATOM）。

参阅

GOSUB，ON-GOSUB，IF-GOSUB，GOSUB-OF



RIGHT

RIGHT$（串，n）函数的作用是把串中最右面的n个字符分离出来。

例如，PRINT RIGHT$(“COMPUSOFT”，4）打印出COMPUSOFT串的最右侧4个字母SOFT。

这个串必须用引号括起来或者赋给一个串变量。字符数(n)可以表示成一个变量、常数或算术表达式。必须用逗号分隔串与数。

如果(n)的值是小数，计算机自动求出它的整数值。

测试程序

```
10 REM 'RIGHT$' TEST PROGRAM
20 A$="CONTEST"
30 B$=RIGHT$(A$,4)
40 PRINT "THE ";RIGHT$("ALRIGHT",5);" FUNCTION PASSED
   THE ";B$
99 END
```

运行实例

```
THE RIGHT$ FUNCTION PASSED THE TEST
```

替代的形式

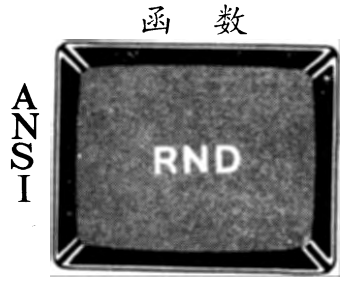
有些计算机（例如，使用MAX BASIC的机器）用RIGHT而不是用RIGHT$。

其它用法

少数BASIC使用RIGHT$（A$，N）从A$串的第N个位置开始分离出子串。注意包括第N个字符右边的所有字符。例如，RIGHT$（“CHOCOLATE”，6）得到LATE，因为L是在第六个位置上。

参阅

PRINT，LEFT$，MID$，CHR$，SPACE$，STR$，STRING$，INKEY$，INSTR，SEG$



几乎所有的计算机都使用RND函数来产生随机数。在不同的计算机之间RND函数用法很不相同。

大部分计算机使用RND(n)通知计算机建立0到1之间的随机小数。极少数计算机（例如DEC和APF）遵照ANSI规则只用RND不用(n)。ANSI还要求每次使用RND重复它自身建立的同一个序列（为了调试目的），除非程序中包括BASIC字RANDOMIZE，使之“重新

随机化”

某些计算机（例如Apple整型和TRS—80 LEVEL Ⅰ BASIC）有一个有限的“整型BASIC”。它们使用RND(n)产生随机整数，这些整数当然是在指定机器上允许的最小整数和最大整数之间的数。一般为－32768到＋32767 。

通常，每使用一次RND，计算机将“产生”一个不同的数。但是，常常需要确定N的值，以指定RND函数将怎样进行操作。

例如，如果给出的N是负值，它可以“重置”随机数发生器。也就是说，它指定每改变一次这个负数时就建立一个新的序列。如果重复用同一个负数，则重复产生相同的序列。产生同一个序列对调试程序来说是很有价值的。

其它计算机使用负的N产生重复的一个随机数。在这种情况下，为了使RND“正常”工作N必须是正的。

如果N的值大于1，少数计算机（例如，Sinclair ZX80和TRS—80）得到一个1和N之间的一个正整数。

正确地决定在程序中使用哪一种RND的用法，将使你避免许多无谓的“虚功”。在你的计算机上如何使用RND，你的计算机手册会提供最好的说明。

测试程序＃1

```
10 REM 'RND' TEST PROGRAM
20 FOR X=1 TO 8
30 PRINT RND,
40 NEXT X
99 END
```

运行实例（典型的）

```
 .862675        .735285        .476059        .55141
 .245708        .242171        .968336        .721014
```

某些计算机使用RND(0)，与RND作用相同。

测试程序＃2

```
10 REM 'RND(0)' TEST PROGRAM
20 FOR X=1 TO 8
30 PRINT RND(0),
40 NEXT X
99 END
```

运行实例（典型的）

```
 .627633        .358479        .137551        .127641
 .125054        .809923        .888076        .787762
```

当RND(n)产生了一个随机数后，某些计算机用RND(0)重复随机数发生器产生的最后一个随机数。

测试程序# 3

```
10 REM 'RND(0) AS A REPEAT' TEST PROGRAM
20 PRINT "RND(1)"
30 FOR X=1 TO 4
40 PRINT RND(1),
50 NEXT X
60 PRINT "RND(0)"
70 FOR Y=1 TO 4
80 PRINT RND(0),
90 NEXT Y
99 END
```

运行实例（典型的）

```
RND(1)
 .592453        .245804        .118263        .961308
RND(0)
 .961308        .961308        .961308        .961308
```

当n大于1 时，少数计算机建立1 到n之间的随机整数（例如TRS—80）。RND(n) 自动对n值取整。

测试程序 # 4

```
10 REM 'RND' TEST PROGRAM
20 N=10
30 FOR X=1 TO 4
40 PRINT RND(N),
50 NEXT X
99 END
```

运行实例（典型的）

```
 8              7              2              9
```

技巧

如果你的计算机产生的随机数大于0 而小于1 ，而你又需要一个0 到9 间的随机整数，可用下面的技巧实现。

```
PRINT INT(10*RND)
```

用下面这个技巧可以打印1 到10之间的随机数。

```
PRINT INT(10*RND+1)
```

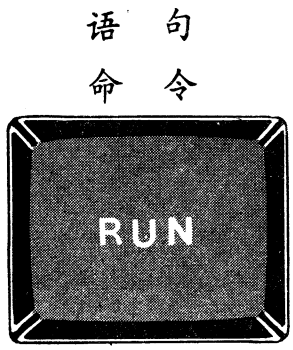
产生从A到B的随机整数的一般格式为：

```
INT(RND*(B-A+1)+A)
```

参阅

RANDOMIZE

RUN命令使计算机从程序的最小行号开始执行程序或者执行在存贮器中的程序。许多计算机可以在RUN后面跟一个行号，用来指定开始执行的行号，而不是从程序的第一行开始（例如RUN40）。



RU
R.

测试程序

```
10 REM 'RUN' TEST PROGRAM
20 PRINT "THIS PRINTING STARTED AT LINE 20."
30 GOTO 99
40 PRINT "THIS PRINTING STARTED AT LINE 40."
99 END
```

运行实例

打入RUN命令后，计算机将显示出：

```
THIS PRINTING STARTED AT LINE 20.
```

如果在RUN命令之后加上行号40，RUN40或RUN40，计算机将从程序的40行开始执行并打印出以下信息：

```
THIS PRINTING STARTED AT LINE 40.
```

大部分计算机把RUN严格地作为监控级的命令，而少数计算机将RUN作为程序语句。那些使用Microsoft Disk BASIC的计算机把RUN作为CHAIN语句的一种形式来接受（见CHAIN）。例如，570 RUN“PROG:1”从磁盘#1上把名为PROG的程序装入并运行。

替代的形式

某些计算机（例如TI990）使用缩写RU.。使用Tiny BASIC的计算机接受R.。

参阅

CHAIN

命 令

在有些计算机中（例如APPLE Ⅱ BASIC和Commodore PET）使用SAVE把程序从计算机内存记录到盒式磁带或磁盘上。详见CSAVE。

测试程序

```
10 REM 'SAVE' TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE SAVE FEATURE"
99 END
```

为了记录，准备好盒式录音机并打入SAVE命令。计算机将通过打开和关闭录音机马达（在存入的开始和结尾）来控制盒式录音机的操作。

程序一旦录制到盒式磁带上，打入NEW（或者其它适当命令）从内存中删去程序。把程序从磁带装入计算机（见LOAD）。为了检验最初打入的程序与存在计算机内存中的程序是否一致，用LIST列出程序清单。

运行实例

```
THIS PROGRAM TESTS THE SAVE FEATURE
```

其它用法

一些带有磁盘存贮能力的计算机使用SAVE把程序从计算机内存“拷贝”到磁盘存贮器。这样做需要文件名。

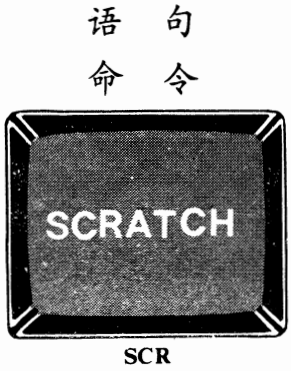
例如：

```
SAVE "TEST"
```

参阅

LOAD，CSAVE，CLOAD，LIST

SCRATCH作为命令的作用是，从内存用户区删除程序并且对所有变量重新置零。这与其它机器上使用的NEW是等效的（详见NEW）。



SCR

测试程序

```
10 REM 'SCRATCH' COMMAND TEST PROGRAM
20 PRINT "THIS IS A TEST"
99 END
```

运行实例

为了弄清楚程序是否已经输入到内存，用LIST列出程序清单。打入SCRATCH命令，然后再列一次清单。程序已不复存在。检查一下打入SCRATCH命令前后的可用内存空间，当程序已不复在内存中时，可以看到可用空间增大了。

替代的形式

SCR往往作为SCRATCH的缩写形式。

其它用法

某些系统（例如DEC—10）使用SCRATCH作为语句，其作用是准备好一个磁盘文件以便接收从计算机输出的信息，SCRATCH语句指定的磁盘文件中原有的内容被清除，这样就可以向该磁盘文件输出新的数据。

例如：

```
100 SCRATCH #1
```

清除在设备号#1上已经打开的文件的内容，并将指针设置到它的第一个记录上。下一个PRINT#1语句（或等价的其它语句）将输出数据到文件的第一个记录中。

其它BASIC（例如，Micropolis BASIC）使用SCRATCH作为命令从磁盘目录中删除文件。在这种应用中，SCRATCH后面必须跟有文件说明。

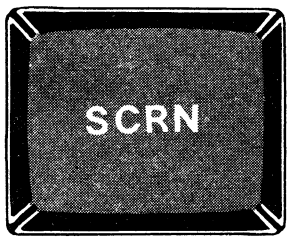
例如：

```
SCRATCH "1:FILE.GROUP"
```

参阅

NEW，CLEAR，ERASE

函 数



SCRN是APPLEⅡ BASIC特有的功能，其作用是指出屏幕上作图块的颜色。APPLE计算机可以在屏幕上显示16种颜色（编号为0 到15）。显示的全部颜色表见COLOR。

作图块是通过SCRN后面括号中的X、Y坐标来指定的。X值代表列号，Y值代表行号。X值的范围从0 到39，Y值从0 到47。

测试程序

```
10 REM 'SCRN' TEST PROGRAM
20 GR
30 COLOR=11
40 PLOT 20,10
50 IF SCRN(20,10)=11 THEN 80
60 PRINT "THE SCRN FUNCTION FAILED THE TEST"
70 GOTO 99
80 PRINT "THE SCRN FUNCTION PASSED THE TEST"
99 END
```

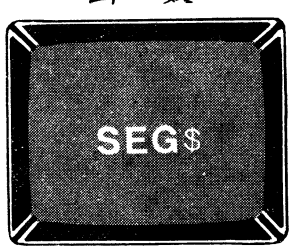
运行实例

```
THE SCRN FUNCTION PASSED THE TEST
```

参阅

COLOR，PLOT，GR，POINT

函 数



SEG

SEG$函数的作用是，从一个字符串变量中选取一部分。SEG$有三个自变量：字符串变量，字符串中的起始位置和子字符串的字符个数。

例如：IF A$= “COMPUTER”，THEN PRINT SEG$（A$，4，3）打印出PUT。

测试程序

```
10 REM * SEG$ TEST PROGRAM *
20 A$="CONTESTANT"
30 B$=SEG$(A$,4,4)
40 IF B$<>"TEST" THEN 70
50 PRINT "SEG$ PASSED THE ";B$
60 GOTO 99
70 PRINT "SEG$ FAILED THE TEST"
99 END
```

运行实例

```
SEG$ PASSED THE TEST
```

SEG$可以起到LEFT$和RIGHT$的作用。SEG$（A$，1，4）与LEFT$（A$，4）等效，而SEG$（A$，LEN（A$）－3，3）等效于RIGHT$（A$，3）。

替代的形式

少数计算机使用SEG。

如果你的计算机没有此功能

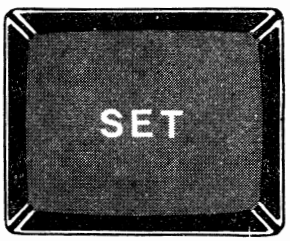
如果你的计算机既不能使用SEG$也不能使用SEG，可在测试程序中试一下MID$。如果MID$也不能用，那么，对于需用DIM语句定义字符串的计算机（例如，Hewlett—Packard），会接受A$(4，7），以得到A$中从位置4到位置7的子字符串。

参阅

MID$，LEFT$，RIGHT$，DIM

SET

语　句



TRS—80使用SET语句“打开”或“点亮”屏幕上指定坐标方格的作图块。在坐标内点亮的块，是由SET语句后面的括号内的X，Y坐标来指定的。例如，SET（5，8）命令计算机“打开”位于第5列第8行的作图块。

关于“关闭”彩色块，请见RESET。

S.

测试程序

```
10 REM 'SET' TEST PROGRAM
20 PRINT "ENTER X COORDINATE";
30 INPUT X
40 PRINT "ENTER Y COORDINATE";
50 INPUT Y
60 SET(X,Y)
70 PRINT "SET PASSED THE TEST"
80 PRINT "IF A LIGHT APPEARED AT (X,Y) COORDINATE
   (";X;",";Y;")."
99 END
```

运行实例（输入65和40）

```
ENTER X COORDINATE? 65
ENTER Y COORDINATE? 40
SET PASSED THE TEST
IF A LIGHT APPEARED AT (X,Y) COORDINATE (65,40).
```

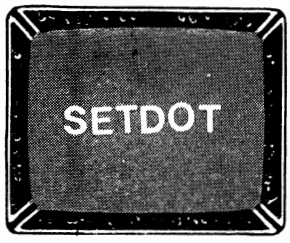
替代的形式

某些计算机（例如TRS—80LEVEL Ⅰ）允许用S. 作为缩写形式。

参阅

RESET，SETDOT

语　句



Sweden的ABC－80用SETDOT语句“打开”屏幕上的作图块。“打开”的块由SETDOT语句后面的L，C坐标指定，L确定行（在作图方式中从0到71），C确定列（在作图方式中从2到79）。

例如，SETDOT 9，15使计算机把从左上角开始的第10行第16列上的块打开。关于如何关闭作图块请见CLRDOT。

测试程序

```
10 REM 'SETDOT' TEST PROGRAM
20 PRINT CHR$(12)                    'CLEARS SCREEN
30 PRINT "SETDOT PASSED THE TEST IF A LINE APPEARS"
40 FOR T=1 TO 2000 : NEXT T
50 PRINT CHR$(12)
60 FOR R=0 TO 23                     'LINES 60,,70 AND
70 PRINT CUR(R,0);CHR$(151);         '80 SET THE SCREEN
80 NEXT R                            'IN GRAPHICS MODE
90 R=5
100 FOR C=2 TO 35
110 SETDOT R,C
120 NEXT C
130 FOR T=1 TO 2000 : NEXT T
140 PRINT CHR$(12)
999 END
```

运行实例

```
SETDOT PASSED THE TEST IF A LINE APPEARS
```

参阅

SET，CLRDOT，RESET，¤

函 数

SGN通知我们一个数的符号。如果数的符号为负我们得到－1，如果是0得0，如果数为正得到1。

例如，PRINT SGN（－8），SGN(0)，SGN(4)打印出

－1　　0　　1

测试程序

```
10 REM 'SGN' FUNCTION TEST PROGRAM
20 X = 5
30 T = SGN(X)
40 IF T=1 THEN 70
50 PRINT "'SGN' FAILED THE TEST"
60 GOTO 99
70 PRINT "'SGN' PASSED THE TEST"
30999 END
```

运行实例

```
'SGN' PASSED THE TEST
```

如果你的计算机没有此功能

如果你的计算机对SGN的测试失败了，可用下面的子程序代替：

```
30000 GOTO 30999
30080 REM * SGN FUNCTION SUBROUTINE * INPUT X, OUTPUT T
30082 T = 0
30084 IF X=0 THEN 30092
30086 T = 1
30088 IF X>0 THEN 30092
30090 T = -1
30092 RETURN
```

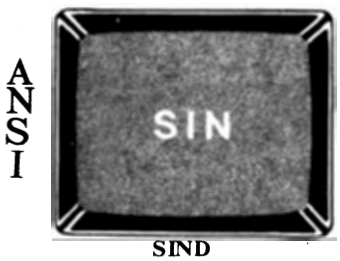
并将测试程序中30行改成：

```
30 GOSUB 30080
```

参阅

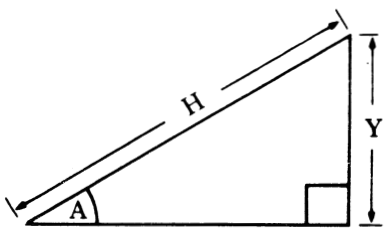
ABS

函 数



SIND
SING

当以弧度（不是度）表示一个角时，SIN（A)函数计算角A的正弦值。1 弧度≈57度。



正弦（SIN）定义为角的对边长度与斜边长度之比。此公式只用在直角三角形中：

SIN（A）= Y/H

SIN的反函数是ARCSIN（Sin^{-1}）。当一个角的正弦或者边比（Y/H）已知时，用ARCSIN可求出角的值。详见ASN。

测试程序

```
10 REM 'SINE' TEST PROGRAM
20 PRINT "ENTER AN ANGLE (EXPRESSED IN RADIANS)";
30 INPUT R
40 Y=SIN(R)
50 PRINT "THE SINE OF A";R;"RADIAN ANGLE IS";Y
30999 END
```

运行实例（输入 1）

```
ENTER AN ANGLE (EXPRESSED IN RADIANS)? 1
THE SINE OF A 1 RADIAN ANGLE IS .841471
```

为了把度化为弧度，用已知的度数乘以0.0174533即可。

例如，R = SIN（A＊. 0174533）

要把弧度化成度，用已知的弧度去乘57.29578。

某些计算机也允许以度或梯度(grads)为单位输入角度(100梯度 = 90度)。在这些机器上使用度时用SIND函数，使用梯度（grads）时用SING函数。修改40行，如果用SIND将产生.0174524，用SING将产生.0157073。

如果你的计算机没有此功能

如果你的解释系统没有SIN功能，可用下面的子程序来代替，以计算用弧度为单位的正弦值。

```
30000 GOTO 30999
30360 REM * SINE SUBROUTINE * INPUT X IN RADIANS,
      OUTPUT Y
30362 REM ALSO USES C AND Z INTERNALLY
30364 X=X*57.29578
30366 IF X=0 THEN 30408
30368 Z=ABS(X)/X
30370 C=X
30372 X=Z*X
30374 IF X<360 THEN 30378
30376 X=X-INT(X/360)*360
30378 IF X<=90 THEN 30398
30380 X=X/90
30382 Y=INT(X)
30384 X=(X-Y)*90
30386 ON Y GOTO 30388,30392,30396
30388 X=90-X
30390 GOTO 30398
30392 X=-X
30394 GOTO 30398
30396 X=X-90
30398 X=Z*X/57.29578
30400 IF ABS(X)<2.48616E-4 THEN 30408
30402 Y=X*X
30404 Y=((((Y/72-1)*Y/42+1)*Y/20-1)*Y/6+1)*X
30406 GOTO 30410
30408 Y=X
30410 X=C/57.29578
30412 RETURN
```

再对测试程序修改如下：

```
35 X=R
40 GOSUB 30360
```

为了求一个角（以度表示）的正弦，可以删除30364行或者修改40行成为：

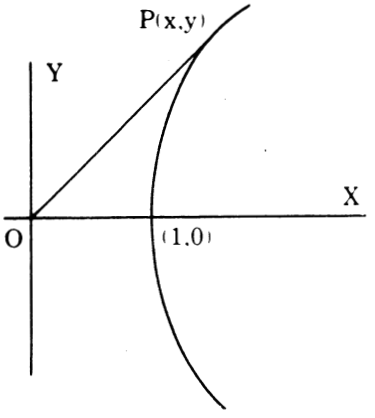
```
40 GOSUB 30366
```

其它用法

一些（很少）解释系统自动地把角度的其它表示单位均转成以度来表示。

参阅

TAN，COS，ATN，ACS，ASN，SINH



SINH(N)是计算一个数的双曲正弦的函数。双曲函数表示以双曲线为基础的关系，与在圆上定义三角函数的方法相似。

如果在单位双曲线上（例如，X＊X－Y＊Y＝1的图），从原点出发画一条直线到曲线（见图）上的点P，形成的区域的面积为N/2。SINH(N)将给出相交点（如图上的P点）的Y坐标值。（COSH(N)将给出其X值。）

SINH和三角函数不同，N不是角度单位，因此，不必指定度或弧度作为单位。N可以是任意的正负实数。

测试程序

```
10 REM 'SINH' TEST PROGRAM
20 PRINT "ENTER A VALUE";
30 INPUT N
40 S=SINH(N)
50 PRINT "THE HYPERBOLIC SINE OF";N;"IS";S
30999 END
```

运行实例（输入值为1）

```
ENTER A VALUE? 1
THE HYPERBOLIC SINE OF 1 IS 1.1752
```

如果你的计算机没有此功能

如果你的计算机不接受SINH，可以用EXP函数来计算此值，例如：

```
40 S=.5 * (EXP(N)-EXP(-N))
```

如果你的计算机也没有EXP函数，请用下面的子程序来代替。在EXP函数一节中下面(85页)列出了EXP函数求值的替代方法（30200开始的子程序），利用它和下面的子程序可以求出SINH函数。

```
30000 GOTO 30999
30450 REM * SINH SUBROUTINE * INPUT N, OUTPUT S
30452 REM ALSO USES A, B, E, L AND X INTERNALLY
30454 X=N
30456 GOSUB 30200
30458 S=E
30460 X=-N
30462 GOSUB 30200
30464 S=.5*(S-E)
30466 RETURN
```

为了使用这个子程序，在测试程序中作以下修改：

```
40 GOSUB 30450
```

替代的形式

Harris BASIC—V用SNH来表示SINH函数。

参阅

```
COSH, TANH, EXP
```

语　句

HARRIS BASIC－V用SLEEP程序的执行停止你所指定的时间，以十分之一秒为一单位。

例如，SLEEP 300使得计算机在继续执行程序之前停止30秒，因为300×0.1秒＝30秒。

测试程序

```
10 REM 'SLEEP' TEST PROGRAM
20 PRINT "THE COMPUTER SHOULD PRINT THE FOLLOWING LINE"
30 SLEEP 150
40 PRINT "AFTER SLEEPING 15 SECONDS"
99 END
```

运行实例

```
THE COMPUTER SHOULD PRINT THE FOLLOWING LINE
                          (停止15秒)
AFTER SLEEPING 15 SECONDS
```

如果你的计算机没有此功能

在程序中插入FOR—NEXT循环来“消耗”计算机的时间。测试你的计算机看它每秒执行多少次循环。微型计算机每秒只能执行几百次循环，而大型机能执行50,000或更多次。

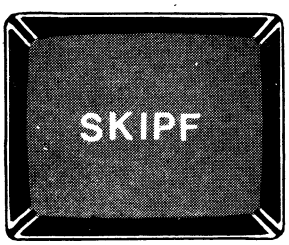
在测试程序中用下面的语句代替30行（假如你的计算机每秒执行1000次循环）：

```
30 FOR L=1 TO 15000
35 NEXT L
```

参阅

```
WAIT
```

命 令



少数计算机（例如袖珍Sharp/TRS—80）利用SKIPF使盒式磁带跳到文件尾。换言之，用SKIPF使磁带跳过（SKIP）文件。

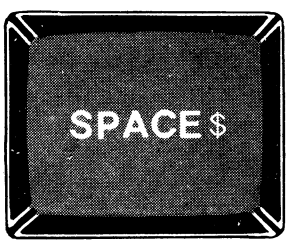
如果给出的SKIPF命令没有文件名，就跳过当前文件并且磁带停在此文件的尾部。如果指定了文件名，例如SKIPF“A”，计算机搜索磁带中的文件“A”并停在“A”文件尾。如果文件“A”没有找到，在屏幕上输出I/O ERROR（输入/输出错）。

如果在 SKIPF 命令的后面指定一个不存在的文件名可以显示磁带上的程序目录。在搜索中遇到的每个文件名都在屏幕上显示出来，直到磁带结束并显示出I/O ERROR（错误）信息为止。

参阅

CLOAD，CSAVE。

函 数



使用SPACE$(n)函数插入指定数目(n)的空格。

例如，PRINT SPACE$（20）；“HELLO”在HELLO前面打印20个空格。

带有SPACE$(n)功能的大部分计算机要求的值(n)要大于0小于256 。

SPACE
SPA
SPC

测试程序

```
10 REM 'SPACES' TEST PROGRAM
20 A$=SPACE$(10)
30 PRINT "IF THE FOLLOWING LINE CONTAINS 10 LEADING SPACES"
40 PRINT A$;"THE SPACE$ FUNCTION PASSED THE TEST"
99 END
```

运行实例

```
IF THE FOLLOWING LINE CONTAINS 10 LEADING SPACES
          THE SPACE$ FUNCTION PASSED THE TEST
```

如果你的计算机没有此功能

如果你的计算机使用STRING$，在测试程序中试一下20 A$=STRING$(10,″ ″)。然后试再试一下20 A$=STRING$（10，32）第二个数（32）是空格的ASCII码。

在大多数情况下，可通过小心地使用TAB函数插入空格。记住，SPACE$是从当前

指针位置开始计数，而TAB总是从最左边开始计数。

例如用40 PRINT TAB（10）；“THE SPACE$ FUNCTION PASSED THE TEST”可以达到与上例同一个目的，还可以用一个变量代替数值10。也可以直接将空格放到引号中。

例如：

```
40 PRINT"          THE SPACE$ FUNCTION PASSED  THE TEST"
```

替代的形式

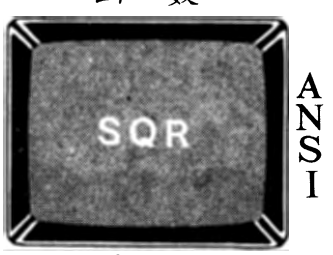
也可以用其它几个字来表示SPACE$函数。它们之中有SPACE（MAX BASIC使用），SPA(Hewlett—Packard 2000)和SPC(Benton Harbor BASIC)。

参阅

TAB，STRING$

SQR(n)函数计算任何正数(n)的平方根。

函 数



SQRT

测试程序

```
10 REM 'SQR' TEST PROGRAM
20 PRINT "THE SQUARE ROOT OF 225
   IS";
30 PRINT SQR(225)
40 PRINT "'SQR' PASSED THE TEST IF
   THE RESULT IS 15"
30999 END
```

运行实例

```
THE SQUARE ROOT OF 225 IS 15
'SQR' PASSED THE TEST IF THE RESULT IS 15
```

替代的形式

一些计算机例如DEC－10和NORTH STAR使用SQRT表示平方根函数。

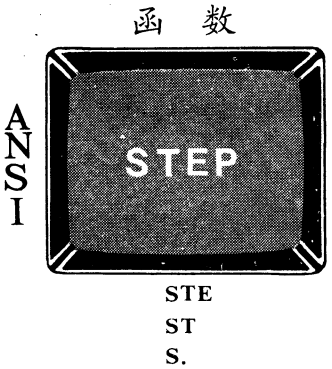
如果你的计算机没有此功能

如果你的计算机不能使用平方根函数，可用下面的子程序代替：

```
30000 GOTO 30999
30020 REM * SQUARE ROOT SUBROUTINE * INPUT X, OUTPUT Y
30022 REM ALSO USES VARIABLES W AND Z INTERNALLY
30024 IF X=0 THEN 30048
30026 IF X>0 THEN 30032
30028 PRINT "ROOT OF NEGATIVE NUMBER?"
30030 STOP
30032 Y=X/4
30034 Z=0
30036 W=(X/Y-Y)/2
30038 IF W=0 THEN 30050
30040 IF W=Z THEN 30050
30042 Y=Y+W
30044 Z=W
30046 GOTO 30036
30048 Y=0
30050 RETURN
```

为了在测试程序中使用这个子程序，要作以下修改：

```
25 X=225
30 GOSUB 30020
35 PRINT Y
```



使用STEP函数指定FOR—NEXT语句中的步长。步长值可以是正、负、甚至可以是非整型的小数。当不指定步长时，步长值为1。

测试程序#1

```
10 REM 'STEP' TEST PROGRAM
20 PRINT "WHEN THE STEP VALUE IS 2,
   X=";
30 FOR X=1 TO 10 STEP 2
40 PRINT X;
50 NEXT X
99 END
```

运行实例

```
WHEN THE STEP VALUE IS 2, X= 1  3  5  7  9
```

以下程序测试解释系统处理负步长值的能力。

测试程序#2

```
10 REM 'NEGATIVE STEP' TEST PROGRAM
20 PRINT "WHEN THE STEP VALUE IS -2, X=";
30 FOR X=10 TO 1 STEP -2
40 PRINT X;
50 NEXT X
99 END
```

运行实例

```
WHEN THE STEP VALUE IS -2, X= 10  8  6  4  2
```

测试程序#3 检查解释系统处理非整型小数步长值的能力。

测试程序#3

```
10 REM 'NON-INTEGER STEP' TEST PROGRAM
20 PRINT "WHEN THE STEP VALUE IS .5, X=";
30 FOR X=1 TO 5 STEP .5
40 PRINT X;
50 NEXT X
99 END
```

运行实例

```
WHEN THE STEP VALUE IS .5, X= 1  1.5  2  2.5  3  3.5  4
  4.5  5
```

有些解释系统接受变量作为步长值。例如FOR X= 1 TO 30 STEP A，在每次执行相应的NEXT语句时，X按A变量的值进行增值。

测试程序#4

```
10 REM 'VARIABLE STEP' TEST PROGRAM
20 PRINT "ENTER A STEP VALUE (BETWEEN 1 AND 10)"
30 INPUT S
40 PRINT "THE VALUE OF X=";
50 FOR X=1 TO 10 STEP S
60 PRINT X;
70 NEXT X
99 END
```

运行实例（输入3）

```
ENTER A STEP VALUE (BETWEEN 1 AND 10)
? 3
THE VALUE OF X= 1  4  7  10
```

替代的形式

PDP—8 E使用STE，TI 990使用ST，TRS—80 LEVEL I BASIC使用S. 。

如果你的计算机没有此功能

如果没有STEP的功能，或者功能不强，可以很容易地用下面的方法来代替递增的（步长为正）FOR—NEXT语句的功能。在测试程序#4 的50行中略去STEP S，并加入以下几行：

```
45 Y=1
60 PRINT Y;
65 Y=Y+S
67 IF Y>10 GOTO 99
```

在相应的NEXT语句之前插入这几行语句，就可以使X按所需的任意整型值或小数值进行增值。

参阅

FOR，NEXT

语　句

ANSI

STO
ST.
S.

STOP语句用来使程序停止执行并使计算机处于命令或即时方式。STOP可放在程序中任意一处，但是在应写END语句的地方通常是不用STOP的。

某些计算机在出现STOP语句的地方停止程序的执行，而其它计算机则跳到包含END语句的那一行上去。

许多用解释系统(而不是编译系统)的计算机会打印出程序停止时的行号，并且可以通过CONTINUE命令（见CONT）继续执行程序。

测试程序

```
10 REM 'STOP' TEST PROGRAM
20 PRINT "SEE THE STOP STATEMENT IN ACTION"
30 STOP
40 PRINT "THE STOP STATEMENT FAILED THE TEST"
99 END
```

运行实例

```
SEE THE STOP STATEMENT IN ACTION
BREAK AT LINE 30
```

替代的形式

在PDP—8 E和Tektronix 4050系统上用STO，在TRS—80 LEVEL Ⅰ BASIC 上用ST. 和S. 。

其它用法

在同一个程序中试着同时使用STOP和END，可能会失败，除非你知道你的计算机性能。有些机器（例如Varian）在运行中，遇到程序中的STOP停止之后，若使其继续运行需要物理干预（按一个按钮）。

其它的（大部分大型机）对使用STOP的个数无限制，但只能有一个END。另外一些BASIC则允许不限制使用END的次数，但没有STOP语句。大部分微型机允许STOP和END任意地使用。

注意，关于STOP/END的问题几乎都能够妥善解决，并且程序很容易转换。

参阅

CONT，END，GO

语　句
命　令

在APPLEⅡ计算机上使用STORE作为命令和语句，把数值型数组存放到盒式磁带上。在程序控制下，可以把一个大数组存放到磁带上，然后由同一个程序或其它程序重新调用。详见RECALL。

例如：

```
10 DIM A(3,3,3)
.
200 STORE A
```

将在磁带上存放64个值（4 * 4 * 4）供以后使用。（注意，每组四个值，下标是0，1，2，3）

测试程序

```
10 REM 'STORE' TEST PROGRAM
20 DIM A(25),B(25)
30 FOR I=1 TO 25
40 A(I)=I
50 NEXT I
60 STORE A
70 PRINT "REWIND TAPE AND SET TO PLAY - PRESS RETURN"
80 INPUT A$
90 RECALL B
100 FOR I=1 TO 25
110 PRINT B(I),
120 NEXT I
999 END
```

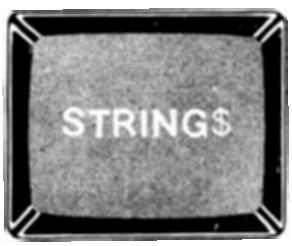
运行实例

```
REWIND TAPE AND SET TO PLAY - PRESS RETURN
?
 1              2              3
 4              5              6
 7              8              9
 10             11             12
 13             14             15
 16             17             18
 19             20             21
 22             23             24
 25
```

参阅

RECALL, CSAVE, DIM

函 数



STRING
STR

STRING$(n, ASCII码) 函数是用在PRINT语句中的。它用来打印n次ASCII字符。

例如，**PRINT STRING$** (10，65) 打印ASCII字符A (ASCII码是65)10次。

测试程序#1

```
10 REM 'STRING$' TEST PROGRAM
20 PRINT STRING$(18,42);
30 PRINT "STRING$ FUNCTION";
40 PRINT STRING$(18,42)
99 END
```

运行实例

```
**********STRING$ FUNCTION*********
```

其它使用方法

某些计算机（例如TRS—80）允许在STRING$函数中用字符串（用引号括起来）或字符串变量。

例如，10 **PRINT STRING$** (10，“A”) 字母A打印10次。又如：

```
10 A$="B"
20 PRINT STRING$(5,A$)
```

打印字母B 5 次。

测试程序#2

```
10 REM 'STRING$' TEST PROGRAM
20 PRINT "ENTER ANY LETTER, NUMBER OR SYMBOL";
30 INPUT A$
40 PRINT STRING$(20,".");
50 PRINT STRING$(20,A$)
99 END
```

运行实例

```
ENTER ANY LETTER, NUMBER OR SYMBOL? X
....................XXXXXXXXXXXXXXXXXXXX
```

替代的形式

某些计算机用STRING或STR来代替STRIN$ 。

如果你的计算机没有此功能

如果你的计算机不允许使用STRING$函数，可以用下面方法代替：找出与STRING$函数中所列ASCII码相应的ASCII字符(见附录)。然后把它放在PRINT语句中，所放的个数由STRING$函数中第一个数来决定。

例如，用

```
10 PRINT "------------"
```

或者

```
10 FOR N=1 TO 12
15 PRINT "-";
20 NEXT N
```

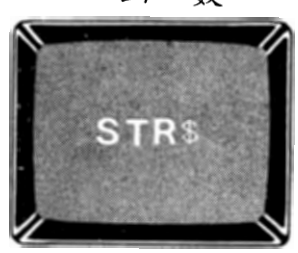
来代替

```
10 PRINT STRING$(12,45)
```

参阅

PRINT, ASC, CHR$, LEN, MID$, LEFT$, RIGHT$, STR$, VAL

函 数



STR$(n)用来把数值(n)转换成字符串。值(n)可以用一个数或数值型变量表示。

例如：

```
10 A$=STR$(35)
20 PRINT A$
```

打印字符串35（它不再是数值，已通过STR$函数转换成字符串)。计算机自动地在数之前给它留出一符号位，如果数是正的，此位置是一左空格。

用STR$函数把数值转换成字符串后，就可以用字符串函数（例如，LEFT$，RIGHT$，MID$，ASC等等）对其内容进行部分的修改，这是很方便的。

测试程序

```
10 REM 'STR$' TEST PROGRAM
20 A = 123
30 A$ = STR$(A)
40 PRINT "IF THE NUMBER";A;"IS CONVERTED TO THE
   STRING";A$
50 PRINT "THEN THE STR$ FUNCTION PASSED THE TEST."
99 END
```

运行实例

```
IF THE NUMBER 123 IS CONVERTED TO THE STRING 123
THEN THE STR$ FUNCTION PASSED THE TEST.
```

参阅

ASC, CHR$, LEN, LEFT$, MID$, RIGHT$, STRING$, VAL NUM$

语　句

在Digital Group Opus 1 和Opus 2 BASIC 中用STUFF把 0 到255之间的一个整数值送入到指定的内存单元中去。

例如，STUFF 3000，65 把十进制数65装入到内存地址为3000的单元中去。

FETCH函数能与STUFF一起使用，检查已经用STUFF装入到内存的内容（某些计算机用 PEEK 或EXAM来代替）。

计算机可利用的内存总量和能被装入（STUFFed）而不清除掉为其它目的而设的内存地址是在变化的。在运行这个测试程序之前，请查阅你的计算机手册，以确定地址15001到15010是不是“无危险”的内存单元。如果它们已经被使用，选择10个其它的连续地址。

测试程序

```
10 REM 'STUFF' TEST PROGRAM
20 FOR X=1 TO 10
30 STUFF 15000+X,X
40 NEXT X
50 FOR X=15001 TO 15010
60 Y=FETCH(X)
70 PRINT Y;
80 NEXT X
90 PRINT
100 PRINT "'STUFF' PASSED THE TEST IF #1 THRU #10 ARE
    PRINTED"
999 END
```

运行实例

```
 1  2  3  4  5  6  7  8  9  10
'STUFF' PASSED THE TEST IF #1 THRU #10 ARE PRINTED
```

如果你的计算机没有此功能

如果在你的计算机上测试程序没有通过，请用POKE和FILL中的测试程序试一下。

参阅

POKE，FILL，PEEK，FETCH，EXAM

语　句

若干计算机（例如INTELLEC，SWTP，COMPUCORP）使用SWAP交换两个变量或数组元素的值。

例如，SWAP（A, B）结果是A原来的值存放在B中，而B原来的值存放在A中。SWAP在按递增或递减的顺序排列数组元素的值时是非常有用的。

测试程序

```
10 REM 'SWAP' TEST PROGRAM
20 PRINT "ENTER TWO VALUES (SEPARATED BY COMMAS)"
30 INPUT A,B
40 IF A<=B THEN 60
50 SWAP (A,B)
60 PRINT A;" IS LESS THAN OR EQUAL TO ";B
70 GOTO 20
99 END
```

运行实例

```
ENTER TWO VALUES (SEPARATED BY COMMAS)
? 3,7
3 IS LESS THAN OR EQUAL TO 7
ENTER TWO VALUES (SEPARATED BY COMMAS)
? 9,1
1 IS LESS THAN OR EQUAL TO 9
ENTER TWO VALUES (SEPARATED BY COMMAS)
?
```

如果你的计算机没有此功能

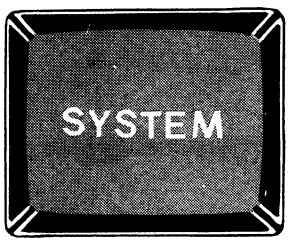
如果SWAP在你的计算机上不能用，请在50行改用EXCHANGE。如果二个都不行，可用以下语句代替50行达到交换值的目的：

```
48 T=A
50 A=B
52 B=T
```

参阅

EXCHANGE

语　句
命　令



SYS

某些计算机使用SYSTEM命令，把机器语言数据（目标代码）由盒式磁带或磁盘装入计算机。这些计算机也可用SYSTEM作为程序语句。

当计算机执行包含有SYSTEM语句的行或从终端上敲入SYSTEM时，计算机转到监控方式并打印一个星号后面跟一个问号（＊？），或者其它的提示符号。这个符号表示计算机已准备好从磁盘或磁带接受目标文件。

在盒式录音机中放入一盘目标代码带，然后使录音机置到播放（PLAY）方式。打入目标文件名并按回车(RETURN)键。录音机的马达是由计算机控制的，在装入(LOAD)开始之前，自动打开录音机的马达，在装入结束后自动关闭它。盒式磁带机将数据重新送回计算机。当数据已装入计算机后，显示出另一个＊？符号。

为了执行目标文件子程序，打入一个斜杠（/）后面跟着一个内存起始地址。如果只输入斜杠（/），后面没有起始地址，则从目标文件中指定的地址开始。

替代的形式

Commodore PET, DEC—10和Sperry Univac System/ 9 使用SYS作为 SYSTEM 的缩写形式。

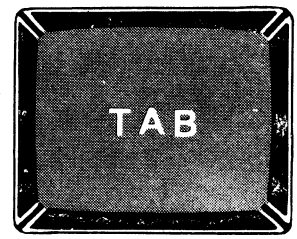
其它用法

SYSTEM命令类似于许多键盘上的ESC(Escape)键。这二者都可以用作系统执行状态或监控状态下的命令。

参阅

PEEK, POKE, MON

函　数



ANSI

T·

TAB函数与PRINT语句一起使用，类似于打字机上的TAB键。当PRINT语句后跟TAB（ ）时，计算机在打印的语句之前插入若干空格，空格的个数由括在括号内的数决定。TAB值必须是正值，并且其值应小于一行中允许打印的字符数。

如果在一行上用了一个以上的TAB，TAB中自变量的数值必须是逐渐增大的，并且应当留出打印项之间的空间。假如TAB间留的空间不足，就会出现“超限”（overrun），正如同在打印机上那样。

TAB值可以是一个数，如PRINT TAB（5）；一个变量，如PRINT TAB（X）；或一个表达式，如 PRINT（2X＋Y） TAB（ ）后面跟一个逗号或分号，但这取决于解释系统。

测试程序

```
10 REM 'TAB' FUNCTION TEST PROGRAM
20 PRINT TAB(5); "TAB 5"
30 X = 10
40 PRINT TAB(X); "TAB 10"
50 PRINT TAB(6*X/5+8); "TAB 20"
999 END
```

运行实例

```
TAB 5
     TAB 10
                TAB 20
```

对测试程序增加以下语句，可以很快确定出你的计算机允许的TAB的最大值：

```
60 PRINT "TYPE IN A TAB VALUE";
70 INPUT T
80 PRINT TAB(T); "TAB";T
90 GOTO 60
```

在70行输入T值，然后在80行输出个数等于T的空格，跟着打印空格个数T。

替代的形式

使用各种Tiny BASIC版本的计算机接受T.作为TAB的缩写。

如果你的计算机没有此功能

没有一个完整的东西能代替TAB，但是有几种方法可以获得适当的输出。假设原来的PRINT字符串为：

```
200 PRINT TAB(10);"THE";TAB(20);"QUICK";TAB(30);"BROWN";
210 PRINT TAB(40);"FOX"
```

TAB值是简单的数并可用以下内容来代替：

```
200 PRINT"          THE        QUICK      BROWN      FOX"
```

或者不那么精确：

```
200 PRINT "THE","QUICK","BROWN","FOX"
```

或者用以上两种方法（插入若干空格以及自动地按打印区输出）的结合。

第三种（一般是不大被满意的）方法是把分号（；），逗号（，）和插入空格组合起来。但是在某些解释系统（或编译程序）中这种做法可能会出现十分糟糕的情况。

参　阅

PRINT、PRINT USING、PRINT AT、,(逗号)、;(分号)、SPACE$

函　数

ANSI

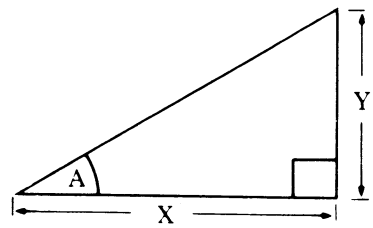
TAND

TANG

当一个角用弧度（不是度）表示时，TAN(A) 计算角A的正切。1弧度≈57度。

正切定义为角的对边与角的邻边之比。

TAN(A) = Y/X



TAN的反函数是ARCTAN(ATN)。当一个角的TAN值或斜率（Y/X）已知时，可用ARCTAN可求出此角的值。

测试程序

```
10 REM 'TAN' TEST PROGRAM
20 PRINT "ENTER AN ANGLE (EXPRESSED IN RADIANS)";
30 INPUT R
40 Y=TAN(R)
50 PRINT "THE TANGENT OF A";R;"RADIAN ANGLE IS";Y
30999 END
```

运行实例（输入1）

```
ENTER AN ANGLE (EXPRESSED IN RADIANS)? 1
THE TANGENT OF A 1 RADIAN ANGLE IS 1.55741
```

为了把值从度化成弧度，用度数去乘.0174533。例如，R = TAN (A*.0174533)。要把弧度化为度，用已知的弧度去乘57.29578。

某些计算机接受以度或梯度（grads）（100梯度 = 90度）为角的单位。在这些机器中如果用度为单位，使用TAND函数，如果用梯度单位，使用TANG函数。把测试程序

中的40行分别改用TAND和TANG，将分别输出.0174451和.0157092。

如果你的计算机没有此功能

如果你的解释系统有SIN和COS，但是没有TAN，可用SIN（A）/COS（A）来代替TAN（A）。

如果你的解释系统没有SIN，COS或TAN，可用以下子程序来计算以弧度为单位的TAN。将SIN和COS两节中列出的子程序加到下面的子程序中，以使其工作。

```
30000 GOTO 30999
30300 REM *TANGENT SUBROUTINE * INPUT X IN RADIANS,
      OUTPUT Y
30302 REM ALSO USES A, B, C, D, W AND Z INTERNALLY
30304 X=X*57.29578
30306 A=X
30308 GOSUB 30336
30310 IF ABS(Y)>1E-8 THEN 30316
30312 PRINT "TANGENT UNDEFINED"
30314 STOP
30316 B=Y
30318 X=A
30320 GOSUB 30366
30322 Y=Y/B
30324 X=A/57.29578
30326 RETURN
```

对测试程序做以下修改：

```
35 X=R
40 GOSUB 30300
```

为了求一个角（以度表示）的正切，可以删去30304行，或者把测试程序中的40行改成：

```
40 GOSUB 30306
```

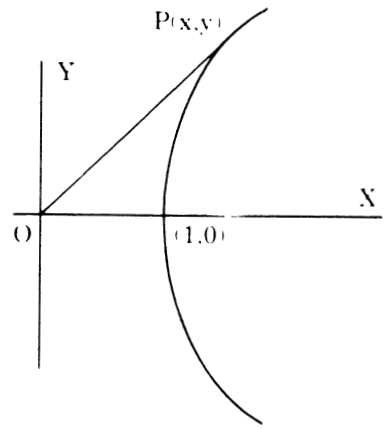
其它用法

一些（很少的）解释系统把角的各种单位都自动地化成度。

参 阅

SIN，COS，ATN，ASN，ACS，TANH

TNH



TANH（N）是计算一个数的双曲正切的函数。双曲函数表示以双曲线为基础的关系，它与在圆上定义三角函数的方法相似。

如果在单位双曲线上（例如，X * X - Y * Y = 1的图形），从原点到曲线上的一点P引一条直线（见图），在曲线上围成一块面积为N/2的区域。TANH（N）给出原点到P点这条线的斜率。（COSH（N）给出交点P的X坐标值，SINH（N）给出Y值。）

不象三角函数，N不需要指定角的单位，因此不必用度或弧度。N可以是任意的正负实数，但是TANH（N）总是在-1和1之间。

测试程序

```
10 REM  TANH  TEST PROGRAM
20 PRINT "ENTER A VALUE";
30 INPUT N
40 T=TANH(N)
50 PRINT "THE HYPERBOLIC TANGENT OF ";N;"IS";T
30999 END
```

运行实例（输入0.5）

```
ENTER A VALUE? .5
THE HYPERBOLIC TANGENT OF .5 IS .462117
```

如果你的计算机没有此功能

如果你的计算机不接受TANH，你可以用EXP函数代替来计算它的值。例如：

```
40 T=1-2*EXP(-N)/(E P(N)+EXP(-N))
```

如果也没有EXP函数，可用下面的子程序来代替。在EXP函数一节中(89页)列出了求EXP函数的替代方法（30200开始的子程序），利用下面的子程序和求EXP函数的子程序可以求得TANH函数值。

```
30000 GOTO 30999
30470 REM * TANH SUBROUTINE * INPUT N, OUTPUT T
30472 REM ALSO USES A, B, E, L AND X INTERNALLY
30474 X=N
30476 GOSUB 30500
30478 T=E
30480 X=-N
30482 GOSUB 30500
30484 T=1-2*E/(T+E)
30486 RETURN
```

为了使用这个子程序，对测试程序做以下修改：

```
40 GOSUB 30470
```

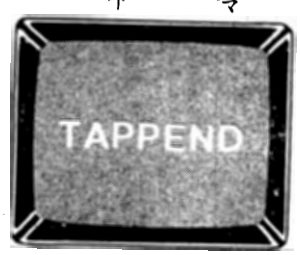
替代的形式

Harris BASIC—V使用TNH代替TANH函数。

参　阅

SINH, COSH, EXP

命　令



某些解释系统（例如 Percom）使用TAPPEND 把一个已经在内存的程序和盒式磁带上的程序连在一起。它可用做磁带文件的扩充（Tape APPEND）。

被送入内存的程序的行号应大于已在内存的最后的行号。

测试程序

为了测试TAPPEND，将下面的短程序以PROG 2为名存放到磁带上。

```
1000 PRINT "THESE LINES ARE"
1010 PRINT "FROM PROG2"
1020 END
```

然后打入NEW或SCRATCH清除此程序，并打入下面这个程序PROG 1：

```
10 REM 'TAPPEND' TEST PROGRAM PROG1
20 PRINT "THESE LINES ARE"
30 PRINT "FROM PROG1"
40 PRINT "      BUT..."
```

最后打入TAPPEND PROG 2并运行。

运行实例

```
THESE LINES ARE
FROM PROG1
     BUT...
THESE LINES ARE
FROM PROG2
```

参　阅

APPEND，CLOAD，CSAVE，TLOAD，TSAVE

语　句
命　令

在APPLE Ⅱ BASIC中TEXT作为命令和程序语句，其作用是使计算机的操作由作图方式改变到正常的文本方式。

测试程序

```
10 REM  TEXT  TEXT PROGRAM
20 TEXT
30 PRINT "THE 'TEXT' STATEMENT DID NOT CRASH"
40 END
```

运行实例

```
THE  TEXT  STATEMENT DID NOT CRASH
```

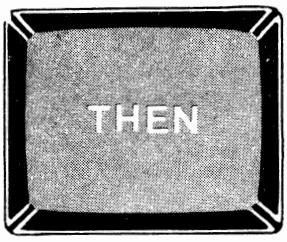
其它用法

在带有MAXBASIC的计算机中用TEXT来指定某些变量为串变量。例如，TEXT A，F，M定义变量A，F和M作为串变量。

参　阅

GR，DEFSTR，$

语　句



ANSI

THE
T.

THEN与IF语句一起使用，它指示当IF语句的条件满足时，计算机执行它后面的操作。

详见IF—THEN。

测试程序

```
10 REM 'THEN' TEST PROGRAM
20 X=10
30 IF X=10 THEN 60
40 PRINT "'THEN' FAILED THE TEST"
50 GOTO 99
60 PRINT "'THEN' PASSED THE TEST"
99 END
```

运行实例

```
'THEN' PASSED THE TEST
```

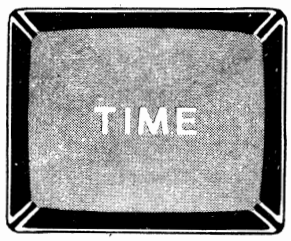
替代的形式

POP－8E计算机允许使用THE，而TRS－80LEVEL Ⅰ BASIC使用T.。

参　阅

IF-THEN

语　句
命　令



TIM
TI

在某些计算机中TIME用来完成一种专门功能，指出从已知时间的参考点开始的所经过的时间（以秒或若干分之几秒为单位）。

大部分“分时”机器统计的时间是从午夜12点到下一个午夜12点，而“独占的”机器统计的时间是从计算机开机时刻开始直到关机。

例如，PRINT TIME 能打印出类似于017230形式的数，表示某些运行单位的总的计算机运行时间。

计算机的时间测量单位是不同的。Commodore PET以每秒60次的频率增加TIME的值，而使用MAXBASIC的那些计算机以每秒1000次的频率增加它的值，DEC BASIC－PLUS－2以每秒一次的频率增加其值。

某些计算机（例如Commodore PET）以六位数指示已经过的时间，这个值是不能改变或重新置零的，除非关闭计算机。

测试程序

```
10 REM 'TIME' TEST PROGRAM
20 A=TIME
30 PRINT "TIME IS MARCHING ON"
40 FOR X=1 TO 2000
50 NEXT X
60 B=TIME
70 IF B<A THEN 100
80 PRINT "THE TIME FUNCTION FAILED THE TEST"
90 GOTO 999
100 PRINT "'TIME' PASSED THE TEST - ELAPSED TIME =";B-A
999 END
```

运行实例（典型的）

```
TIME IS MARCHING ON
'TIME' PASSED THE TEST - ELAPSED TIME = 270
```

替代的形式

某些计算机（例如DEC—10）使用TIM，而另外一些计算机（例如Commodore PET）使用TI 。

其它用法

DEC BASIC —PLUS— 2 使用以下TIME 形式：

TIME（0）指出从午夜开始所经过的总的时间（以秒为单位）。

例如，100 PRINT TIME（0）打印出类似于25128的值，指出从午夜开始已经过25.128秒。

TIME（1 %）指出程序经过的总的时间（以十分之一秒为单位）。

例如，100 PRINT TIME（1 %）打印出类似于85这样的值,它表示在打印TIME（1 %）之前，运行经过了8.5秒。

TIME（2 %）指出从终端连接到分时系统开 始 已经过的时间（以分钟为单位）。

例如，10 PRINT TIME（2 %）打印出一个类似于130这样的值，指出终端连到分时系统后已经过了130分钟。

Hewlett Packard 2000F 分时BASIC用TIME 作为一个命令，打印出终端连到系统上后所经过的时间，以及总的累积统计时间。

例如，如果打入命令TIME，它将打印出类似下面的信息：

```
CONSOLE TIME = 5 MINUTES. TOTAL TIME = 2045 MINUTES.
```

参 阅

```
TIME$, CLK$
```

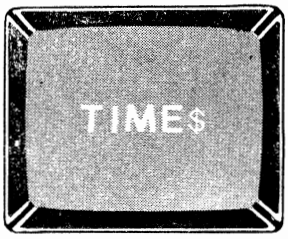
某些计算机（例如Commodore PET和DEC BASIC—PLUS—2）使用TIME$指示本日的时间。

PET计算机用一个六位数来表示时（0—24），分，秒（hhmmss）。TIME$的初值以六位数（用引号括起来）赋给TIME$。

例如，TIME$ = “144500”设置时间在144500（即下午2：45）。计算机从开机（TIME$初始值为000000)或从赋于新值开始以秒为增量累计时间。

命　令

函　数



TI$

测试程序

```
10 REM 'TIME$' TEST PROGRAM
20 PRINT "THE CURRENT TIME IS";TIME$
30 PRINT "THE TIME$ FUNCTION PASSED THE TEST"
40 PRINT "IF A SIX DIGIT NUMBER IS PRINTED"
99 END
```

运行实例（典型的）

```
THE CURRENT TIME IS 012536
THE TIME$ FUNCTION PASSED THE TEST
IF A SIX DIGIT NUMBER IS PRINTED
```

替代的形式

Commodore PET用TI$作为TIME$的缩写形式。

其它用法

DEC BASIC—PLUS—2使用TIME$（0%）以小时和分指示出本日的时间，不表示秒。

例如，PRINT TIMES（0%）将以14:32的形式打印出时间。计算机在小时和分之间自动地插入冒号。此外，DEC BASIC－PLUS－2使用TIME$（n）指示午夜前n分钟的时间。例如，PRINT TIME$（61）打印出22：59。

参　阅

TIME，CLK$

命　令

PERCOM BASIC通过使用TLOAD命令把程序从盒式磁带装入计算机。

测试程序

从键盘把下面程序输入计算机，并且把它存放到盒式磁带上。（详见TSAVE。）

```
10 REM 'TLOAD' TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE TLOAD FEATURE"
99 END
```

一旦把程序记录到盒式磁带上，用NEW（或者用SCRATCH或者用别的适当的命令）清除计算机内存。

倒回磁带，然后将录音机设置到播放（PLAY）方式并且打入TLOAD命令。

当录音机停止时，列出程序清单以证实上面的程序（见清单）已装入到机器上。如果一切没有问题，运行程序。

运行实例

```
THIS PROGRAM TESTS THE TLOAD FEATURE
```

参　阅

CLOAD，LOAD，TSAVE，LIST，NEW，SCRATCH

函　数

在ACORN ATOM计算机中TOP函数的作用是指出没有使用的内存的第一个字节的地址。

例如，PRINT TOP将打印出可用内存的开头地址（用十六进制）。如果你知道你的程序的开始地址（SA），然后打印TOP－SA,将告诉用户该程序的大小。

测试程序

```
10 REM 'TOP' TEST PROGRAM
20 IF TOP=0 THEN 50
30 PRINT "FREE MEMORY STARTS AT ";TOP
40 GOTO 99
50 PRINT "TOP FAILED THE TEST"
99 END
```

运行实例

```
FREE MEMORY STARTS AT 285A
```

确实的地址将取决于计算机的大小和当前已使用的内存总量。

参 阅

```
FRE(0), MEM
```

语 句
命 令

在APPLE Ⅱ BASIC中TRACE用来打印出程序中计算机执行过的每一个行号。此命令用来作为调试手段。用NOTRACE命令有终止TRACE命令的作用。

TRACE也可以用作一个程序语句。允许只跟踪程序中某一个指定的部分。

测试程序

```
10 REM 'TRACE' TEST PROGRAM
20 PRINT"'TRACE' TRACES EACH LINE"
30 TRACE
40 GOTO 90
50 PRINT "UNTIL TURNED OFF BY"
60 NOTRACE
70 PRINT "THE 'NOTRACE' STATEMENT"
80 GOTO 110
90 PRINT "THAT FOLLOWS THE 'TRACE' STATEMENT"
100 GOTO 50
110 PRINT "AS ILLUSTRATED BY THIS LINE"
999 END
```

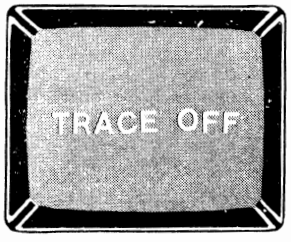
运行实例

```
'TRACE' TRACES EACH LINE
#40#90 THAT FOLLOWS THE 'TRACE' STATEMENT
#100#50 UNTIL TURNED OFF BY
#60 THE 'NOTRACE' STATEMENT
AS ILLUSTRATED BY THIS LINE
```

参 阅

NOTRACE, TRON, TRACE ON, FLOW

语 句
命 令



在Motorola BASIC中用TRACE OFF命令关闭跟踪功能（请见TRACE ON）。TRACE OFF也可以用作程序语句，关闭对程序中指定区域的跟踪。

测试程序

```
10 REM 'TRACE OFF' TEST PROGRAM
20 TRACE ON
30 PRINT "EACH LINE SHOULD BE TRACED"
40 TRACE OFF
50 PRINT "BY THE 'TRACE ON' STATEMENT"
60 PRINT "UNTIL TURNED OFF BY THE 'TRACE OFF' STATEMEN
99 END
```

运行实例

```
 30  EACH LINE SHOULD BE TRACED
 40  BY THE 'TRACE ON' STATEMENT
UNTIL TURNED OFF BY THE 'TRACE OFF' STATEMENT
```

参 阅

TRACE ON, NOTRACE, TROFF, NOFLOW

语　句
命　令

在Motorola BASIC中TRACE ON命令的作用是打印程序中计算机执行过的每一个行号。它用作调试手段。TRACE OFF命令使这个作用终止。

TRACE ON也可以作为一个语句，用来跟踪程序中某一个指定的部分。

测试程序

```
10 REM 'TRACE ON' TEST PROGRAM
20 PRINT "'TRACE ON' TRACES EACH LINE"
30 TRACE ON
40 GOTO 90
50 PRINT "UNTIL TURNED OFF BY"
60 TRACE OFF
70 PRINT "THE 'TRACE OFF' STATEMENT"
80 GOTO 110
90 PRINT "THAT FOLLOWS THE 'TRACE ON' STATEMENT"
100 GOTO 50
110 PRINT "AS ILLUSTRATED BY THIS LINE"
999 END
```

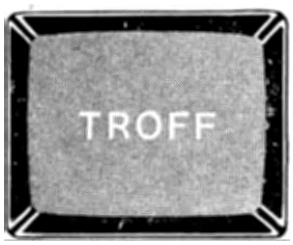
运行实例

```
'TRACE ON' TRACES EACH LINE
<40> <90> THAT FOLLOWS THE 'TRACE ON' STATEMENT
<100> <50> UNTIL TURNED OFF BY
<60> THE 'TRACE OFF' STATEMENT
AS ILLUSTRATED BY THIS LINE
```

参　阅

TRACE OFF, TRACE, TRON, FLOW

语　句
命　令



TROFF（trace off）是在许多解释系统（例如，TRS—80）中用来终止跟踪功能的命令。TROFF也可作为程序语句，用来关闭程序中指定的区域的跟踪。

测试程序

打入TRON命令，然后运行这个程序

```
10 REM 'TROFF' TEST PROGRAM
20 PRINT "THE FIRST THREE LINES OF THIS PROGRAM"
30 TROFF
40 PRINT "ARE PRINTED WITH THE TRACE TURNED ON."
50 PRINT "THIS LINE IS PRINTED WITH THE TRACE TURNED OFF."
99 END
```

运行实例

```
 10  20  THE FIRST THREE LINES OF THIS PROGRAM
<30> ARE PRINTED WITH THE TRACE TURNED ON.
THIS LINE IS PRINTED WITH THE TRACE TURNED OFF.
```

参　阅

TRON，NOTRACE，NOFLOW，TRACE OFF

语　句
命　令

TRON（trace on）命令可以用来作为分析的手段，它打印出程序中计算机执行过的每一个行号。执行TROFF或者NEW命令后，跟踪功能就终止了。TRON用来作为程序跟踪和调试的手段。

测试程序

```
10 REM 'TRON' TEST PROGRAM
20 GOTO 50
30 PRINT "OF THIS TEST PROGRAM."
40 GOTO 70
50 PRINT "TRON TRACES EACH LINE"
60 GOTO 30
70 PRINT "END OF TEST PROGRAM."
99 END
```

运行实例

在运行这个程序之前打入TRON命令。

```
<10><20><50> TRON TRACES EACH LINE
<60><30>OF THIS TEST PROGRAM.
<40><70>END OF TEST PROGRAM.
<99>
```

TRON也可以作为程序语句，用来跟踪程序中某一部分。为了验证这个功能，打入TROFF以保证“跟踪”作用已终止，然后在上述测试程序中增加下面一行并使之运行。

35　TRON

运行实例

```
TRON TRACES EACH LINE
OF THIS TEST PROGRAM.
<40><70>END OF TEST PROGRAM.
<99>
```

参　阅

TROFF，NEW，FLOW，TRACE，TRACE ON

TSAVE　　命　令

PERCOM BASIC用TSAVE命令把计算机内存中的程序存放到盒式录音带中。详见CSAVE。

测试程序

```
10 REM  TSAVE' TEST PROGRAM
20 PRINT "THIS PROGRAM TESTS THE TSAVE FEATURE"
99 END
```

将盒式录音机设置到录音状态并打入TSAVE命令。计算机控制录音机的操作，在存（Save）之前和之后自动地打开和关闭录音机马达。

一旦程序已记录在盒式磁带上，打入NEW（或SCRATCH，或其它适当的命令）从内存中清除程序。把程序从磁带上装入到计算机（见TLOAD）。列出程序清单以证实刚装入计算机内存的程序与原来输入的测试程序相同。

运行实例

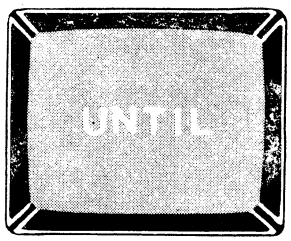
```
THIS PROGRAM TESTS THE TSAVE FEATURE
```

参　阅

CSAVE，SAVE，TLOAD，LIST，NEW，SCRATCH

语　句

修饰符



UNTIL既是修饰符又是语句。当作为修饰符时，某些计算机(如DEC PDP—11）使用UNTIL产生一个条件。

例如，60 X=X+Y UNTIL X>100 Y与X的值相加送入X，直到X的值大于100为止。就象具有它自己的内部FOR—NEXT循环一样。当然，如果X和Y有值，而且这些值使得X的和总是达不到100（例如X=1，Y=-5），计算机就总是执行60行。

测试程序

```
10 REM 'UNTIL' TEST PROGRAM
20 PRINT "TYPE 4 OR 5 NUMBERS (ONE AT A TIME)"
30 PRINT "THE LAST ONE BEING ZERO"
40 INPUT X
50 S=S+X
60 GOTO 40 UNTIL X=0
70 PRINT "THE SUM OF THE NUMBERS TYPED IN IS";S
99 END
```

运行实例

```
TYPE 4 OR 5 NUMBERS (ONE AT A TIME)
THE LAST ONE BEING ZERO
? 23
? 82
? 47
? 125
? 6
? 0
THE SUM OF THE NUMBERS TYPED IN IS 283
```

某些计算机（例如DEC PDP-11）在特殊形式的FOR语句中使用UNTIL。例如

```
50 FOR X=1 STEP 3 UNTIL X*X>100*X+35
```

注意，语句中没有循环终值（例如FOR X=1 TO20），当指定的条件达到时循环就结束了。

少数计算机（例如Acorn ATOM）用UNTIL作为语句，结束用DO语句建立的循环。例如：

```
10 DO X=X+1
20 PRINT X, X*X, SQR(X)
30 UNTIL X=12
```

参　阅

WHILE, FOR

函　数

USER

USR函数执行存放在计算机内存中的机器语言子程序。这个机器语言程序可以用POKE语句从键盘输入或用SYSTEM命令从磁带或磁盘输入到内存。

在程序中可以象用任何其它“内部”函数那样使用USR函数。

例如10PRINT USR（N），将打印出由用户的机器语言子程序计算出来的值。

如果有一个计算N的平方根的机器语言子程序存放在内存中，计算机将打印数N的平方根。

为了测试USR函数，你必须使用POKE语句或SYSTEM命令，把一个机器语言子程序装入到计算机（适当的地址上）。请查阅你的计算机手册以了解在你的计算机上使用这个专门函数的专门的指令。

替代的形式

某些计算机用USER代替USR。

参　阅

POKE，SYSTEM

函　数

VAL函数的作用是：把作为字符串的数字转换成数值。VAL有脱掉引号的作用。例如：

```
10 A$="35"
20 PRINT VAL(A$)
```

打印出作为数值的数35。

测试程序 = 1

```
10 REM 'VAL' TEST PROGRAM
20 A$="45.12"
30 A=VAL(A$)
40 PRINT "IF THE STRING ";A$;" IS CONVERTED TO THE
   NUMBER";A
50 PRINT "THEN THE VAL FUNCTION PASSED THE TEST."
99 END
```

运行实例

```
IF THE STRING 45.12 IS CONVERTED TO THE NUMBER 45.12
THEN THE VAL FUNCTION PASSED THE TEST.
```

注意：某些BASIC不能转换象“ -45”(+号或 -号之前有空格)这样的字符串。假如在你的计算机上有此问题，当执行PRINT VAL（“ -45”）时显示出0。在有这种问题的程序中请写入以下几行语句：

```
200 IF LEFT$(A$,1)<>" " THEN 230
210 A$ = RIGHT$(A$,LEN(A$)-1)
220 GOTO 200
230 A = VAL (A$)
```

其它用法

某些BASIC（例如Microsoft 的各种版本）允许在VAL函数中使用数字和字母的组合，但数字必须在字母的前面。如果不是这样的次序（即数字不在字母之前），VAL产生一个0，表示在第一个字符位置上没有找到数字。

例如，PRINT VAL（“123 ABC”）

打印出123。

测试程序 = 2

```
10 REM 'VAL WITH MIXED STRING' TEST PROGRAM
20 A$="12 O'CLOCK"
30 A=VAL(A$)
40 PRINT "IF THE STRING ";A$;" IS CONVERTED TO THE
   NUMBER";A
50 PRINT "THE VAL FUNCTION ACCEPTED NUMBERS MIXED
   WITH LETTERS."
99 END
```

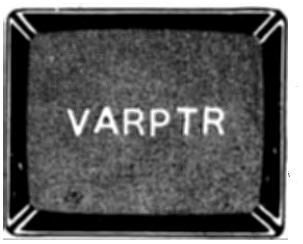
运行实例

```
IF THE STRING 12 O'CLOCK IS CONVERTED TO THE NUMBER 12
THE VAL FUNCTION ACCEPTED NUMBERS MIXED WITH LETTERS.
```

参 阅

STR$, ASC, CHR$, LEN, LEFT$, MID$, RIGHT$, STRING$

函 数



某些BASIC（例如Microsoft BASIC)用VARPTR函数来找出一个变量的内存地址。它可代表变量指针（VARiable PoinTeR）。例如：

A = VARPTR（X）

把存放变量X的值的第一个内存单元的地址赋给A。为了看看在A单元中存放了什么，打入PRINT PEEK（A）。我们所得到的结果的含义取决于变量X的类型。

如果X是整型（见DEFINF和%），在A单元中是X值的最低有效字节（LSB：Least Significant Byte)。A + 1单元中是X值的最高有效字节（MSB：Most Significant Byte）。如果MSB*256 + LSB = X，说明VARPTR操作是正确的。

测试程序 =1

```
10 REM ' VARPTR' TEST PROGRAM
20 X% = 32737
30 A = VARPTR(X%)
40 PRINT "'VARPTR' PASSED THE TEST IF";
50 PRINT PEEK(A+1)*256+PEEK(A);"EQUALS";X%
99 END
```

运行实例

```
'VARPTR' PASSED THE TEST IF 32737 EQUALS 32737
```

当VARPTR与串变量一起使用时，在A单元中存放字符串的长度。接下来的二个单元A + 1和A + 2存放字符串起始地址的LSB和MSB。

VARPTR可以与串变量一起使用，使得能够在USR子程序中使用字符串。例如：T = USR(VARPTR（A$)）把三个地址中的第一个传送到机器语言子程序。

VARPTR也可用来交换二个串变量的内容。认真研究下面测试程序：

测试程序＝2

```
10 REM  VARPTR(A$)  TEST PROGRAM
20 READ F$,L$
30 DATA FIRST,LAST
40 PRINT "THE ";F$;" SHALL BE ";L$
50 A=VARPTR(F$)
60 B=VARPTR(L$)
70 F1=PEEK(A)
80 F2=PEEK(A+1)
90 F3=PEEK(A+2)
100 L1=PEEK(B)
110 L2=PEEK(B+1)
120 L3=PEEK(B+2)
130 POKE B,F1
140 POKE B+1,F2
150 POKE B+2,F3
160 POKE A,L1
170 POKE A+1,L2
180 POKE A+2,L3
190 PRINT "AND THE ";F$;", ";L$
999 END
```

运行实例

```
THE FIRST SHALL BE LAST
AND THE LAST, FIRST
```

VARPTR也可以和单精度、双精度变量一起使用。带回的数是存贮单精度值的四个存贮单元中的第一个单元的地址。（双精度需要八个存贮单元。）由于这些数通常是以“标准指数形式”存放的，所以用测试程序#1中的方法来“再现”它们的值是不切实际的。

VARPTR常常用来简单地找出变量的地址，所以，地址能传送到机器语言子程序。

如果你的计算机没有此功能

如果VARPTR在你的机器上通不过，请试用PTR或ADR。

参　阅

PEEK，POKE，USR，DEFINT，%

语　句

在APPLE Ⅱ BASIC中VLIN－AT用来在屏幕上的指定的列上显示一条垂直线（VLIN－AT是Vertical LINe AT的缩写）。

垂直线的长度由VLIN后面的二个数来确定。这二个数指示垂直线二端的距离。线可以是在0到39行之间的任意一个长度。

AT后面的数是线所在的列数。这个数在0到39之间。

例如，VLIN 10，30 AT 20通知计算机在第20列上，在第10行到第30行之间画一条垂直线。

在计算机接受VLIN—AT语句之前，必须先执行GR语句（见GR）。线的颜色由COLOR语句（见COLOR）确定。

测试程序

```
10 REM 'VLIN-AT' TEST PROGRAM
20 GR
30 Y=0
40 FOR X=0 TO 39
50 COLOR = Y
60 VLIN 0,39 AT X
70 Y=Y+1
80 IF Y<16 THEN 100
90 Y=0
100 NEXT X
999 END
```

运行实例

如果计算机接受了VLIN—AT 语句，屏幕上将充满了39条各种颜色的垂直线。

APF BASIC在它的VLIN语句中不使用AT。例如可用VLIN 10，30，20它与上面的例子等效。使用的颜色在VLIN的执行之前由COLOR语句指定。

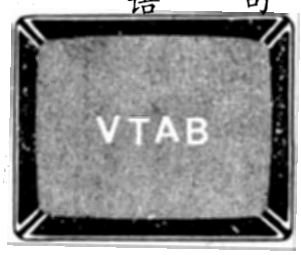
参 阅

GR，COLOR，PLOT，HLIN-AT，TEXT

VTAB 语句

APPLE Ⅱ BASIC使用VTAB（Vertical tab）为PRINT语句指定屏幕上的垂直位置。VTAB 的值是从 1 到24，它代表屏幕的24行。

例如，VTAB 12指定PRINT的起点定在屏幕上从上往下数第12行。



测试程序

```
10 REM 'VTAB' TEST PROGRAM
20 PRINT "ENTER A VTAB VALUE FROM 1 TO 24";
30 INPUT N
40 VTAB N
50 PRINT "VTAB PASSED THE TEST IF THIS IS PRINTED ON
   LINE";N
99 END
```

运行实例（输入5）

```
ENTER A VTAB VALUE FROM 1 TO 24? 5


VTAB PASSED THE TEST IF THIS IS PRINTED ON LINE 5
```

如果你的计算机没有此功能

要使打印从屏幕的（自上而下数）某一指定的行开始，最容易的方法是：首先清屏幕（可以用一连串的PRINT语句，或者一系列的ASCII“换行”符，或用CLS）。查阅手册以确定如果在你的计算机上用PRINT CHR$（N）代表换行的话，N的值应是多少。

然后，再用若干个PRINT语句或ASCII字符，在执行打印之前使光标下移所需的行数。

参 阅

TAB，PRINT-AT，PRINT，ASC，CHR$，HOME，LIN

语　句
命　令

某些计算机（例如那些使用MAX—BASIC的机器）使用WAIT把程序执行挂起等待一定的时间。

例如，WAIT 30通知计算机在执行下一个语句前等待30秒。

有几种计算机按给定时间的若干分之一（例如1/10或者1/1000）来处理。

例如，WAIT 1000，用ADDS BASIC的计算机需要等待1000秒，而VARIAN 620则需要等待10秒钟。

下面的程序允许用来检查你的计算机的WAIT性能。

测试程序＝1

```
10 REM 'WAIT TIME PERIOD' TEST PROGRAM
20 PRINT "ENTER A UNIT OF TIME FOR THE COMPUTER TO WAIT";
30 INPUT T
40 PRINT "THE COMPUTER IS WAITING FOR";T;"UNITS OF TIME"
50 WAIT T
60 PRINT "THE WAIT STATEMENT PASSED THE TEST"
99 END
```

运行实例（输入60）

```
ENTER A UNIT OF TIME FOR THE COMPUTER TO WAIT? 60
THE COMPUTER IS WAITING FOR 60 UNITS OF TIME

THE WAIT STATEMENT PASSED THE TEST
```

在测试程序#1中的WAIT可用下面的FOR—NEXT来取代，同样达到延迟时间的目的。为了在你的计算机上产生与WAIT语句相同的时间延迟量，需要校正T的值。

```
50 FOR X=1 TO T
55 NEXT X
```

一些其它的计算机（例如用Microsoft BASIC的那些机器）使用WAIT挂起程序执行，直到所指定的计算机端口的字节值满足给定的条件为止，此条件是由WAIT后面所列出的二个二进制值建立的。

例如，WAIT 30，2，5通知计算机“等待”，直到端口30的二进制值与5的二进制值进行或运算，结果值再与2的二进制值进行“逻辑与”运算，产生一个非零值为止。当此条件满足时，从下一个语句继续执行程序。如果此条件不满足，按键盘上的BREAK，MONITOR，ESCAPE（或类似功能的）键，可以脱离WAIT状态。

在WAIT语句中列出的每一个值必须在0到255之间（这是可放在8位的内存单元中的值的范围）。当最后一个字节值（如上例中的5）在WAIT语句中省略掉时，计算机假设它的值为零。

在上述例中，在计算机继续执行程序之前，端口30的二进值的第二位必须是“开”

的（即为1）。如下表所示。

	端口值	第三个二进制值		第二个二进制值	
1	0	1	1	0	0
2	1	0	1	1	1
4	0	1	1	0	0
8	0	0	0	0	0
16	0	0	0	0	0
32	0	0	0	0	0
64	0	0	0	0	0
128	0	0	0	0	0
	2 XOR	5 =	7 AND	2 =	2

某些计算机（例如Processor Technology Extended Cassette BASIC）“等待”直到在指定的计算机端口的二进制值和第二个二进制值进行“与”，等于第三个二进制值为止。

例如，WAIT 120，255，6 计算机等待直到接口120的二进制值等于6为止，如表中所示。

	端口值	第二个二进值	第三个二进值
1	0	1	0
2	1	1	1
4	1	1	1
8	0	1	0
16	0	1	0
32	0	1	0
64	0	1	0
128	0	1	0
	6 AND	255 =	6

测试程序＝2

```
10 REM 'WAIT FOR PORT CONDITION' TEST PROGRAM
20 PRINT "THE COMPUTER IS WAITING FOR ONLY BIT 1 TO
   BE SET"
30 PRINT "IN PORT 20 (THE DECIMAL VALUE OF 2)"
40 WAIT 20, 255, 2
50 PRINT "BIT 1 IN PORT 20 IS SET"
99 END
```

运行实例

```
THE COMPUTER IS WAITING FOR ONLY BIT 1 TO BE SET
IN PORT 20 (THE DECIMAL VALUE OF 2)
BIT 1 IN PORT 20 IS SET
```

如果你不能在端口20的第二位设置1，按键盘上的BREAK（或其它类似功能的键），从这个状态中脱离出来。

某些计算机可用WAIT作为命令。

如果你的计算机没有此功能

如果你的计算机有INP功能，但没有WAIT，在测试程序2中用INP代替WAIT，修改如下：

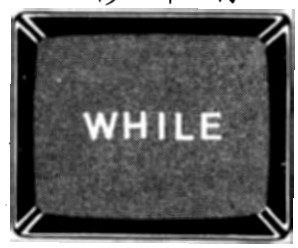
```
40 IF INP(20)=2 THEN 50
45 GOTO 40
```

参　阅

INP，FOR，NEXT，XOR，AND

语　句

修饰符



WHILE是一组语句中的开始的语句，这组语句是被反复执行的直到某些条件为“假”为止。用WHILE语句开始的循环必须用WEND,ENDLOOP或者THEN结束。例如：

```
100 WHILE X<>0
110    INPUT X
120    S=S+X
130 WEND
140 PRINT "SUM =";S
```

只要条件（X＜70）为真，就一次又一次地执110，120行。当输入一个0时，循环中止并转到140行。

测试程序

```
10 REM 'WHILE' TEST PROGRAM
20 L$ = "FIRST"
30 WHILE L$<>"LAST"
40 READ X, Y, L$
50 PRINT "X*Y =";X*Y
60 WEND
70 PRINT "WHILE PASSED THE TEST"
80 DATA 3, 20, MORE
95 DATA 16, 40, MORE
100 DATA 12, 32, LAST
999 END
```

运行实例

```
X*Y = 60
X*Y = 640
X*Y = 384
WHILE PASSED THE TEST
```

某些BASIC把将DO与WHILE合在一起使用来形成条件循环。30行可写成 30 DO WHILE L$< >“LAST”，同时60行用DOEND代替WEND。

另外一些BASIC（例如DEC BASIC—PLUS）使用WHILE作为其它语句的修饰词，形成一个条件。例如，求小于10000的二次方的最大值：

```
10 X = 1
20 X = X*2 WHILE X<10000
30 PRINT X
99 END
```

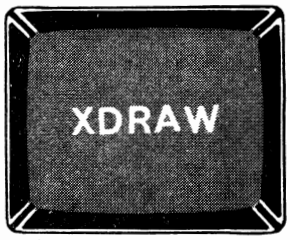
如果你的计算机没有此功能

如果你使用的计算机没有WHILE语句，可以用IF语句建立条件循环。删除测试程序中的20行和30行，并将60行改成60 IF L$< >“LAST”THEN40，将会产生和上述“运行实例”同样的结果。

参 阅

UNTIL，FOR，IF

语 句



XDRAW语句在APPLE Ⅱ中用来清除屏幕上预先画好的图形。

XDRAW也可以用来画图形。如果已用XDRAW画出一个图形，再用XDRAW则会擦掉图形而不擦去背景。

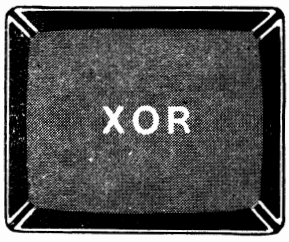
例如：XDRAW 2 AT 135，78将在坐标135（水平）和78（垂直）的位置开始画出号码为2的形状（由用户图形表中选用）。

XDRAW以当前背景颜色的补色画出所需图形。除此之外，XDRAW与DRAW有相同的作用，详见DRAW。

参阅

DRAW

操作符



在IF——THEN语句中使用XOR作为“异或”逻辑运算符。例如，IF A=3 XOR B=3 THEN 80，意思是“假如A和B之中有一个值为3，但二者不同时为3，IF—THEN条件就为真，转去执行80行”。

详见OR。除了以下的一点不同之外，XOR与OR作用相同。如果二个条件都满足，XOR认为测试“失败”，而OR认为“成功”，通常给我们带回一个值为0。

XRA

测试程序#1

```
10 REM LOGICAL 'XOR' TEST PROGRAM
20 A=6
30 B=8
40 IF A=3 XOR B=8 THEN 70
50 PRINT "'XOR' FAILED THE TEST AS A LOGICAL OPERATOR"
60 GOTO 99
70 IF A=6 XOR B=8 THEN 90
80 PRINT "'XOR' PASSED THE LOGICAL OPERATOR TEST"
85 GOTO 99
90 PRINT "'XOR' FAILED THE 'EXCLUSIVE OR' TEST"
99 END
```

运行实例

```
'XOR' PASSED THE LOGICAL OPERATOR TEST
```

替代的形式

NORTH STAR IBASIC使用XRA作为异或运算符。

如果你的计算机没有此功能

如果你的计算机允许用OR、AND和NOT，但是没有XOR和XRA，用下面的行替换测试程序1中的40行：

```
40 IF (A=3 OR B=8) AND NOT (A=3 AND B=8) THEN 70
```

以下测试程序对没有逻辑算符的计算机提供了一个代替XOR的方法。

测试程序#2

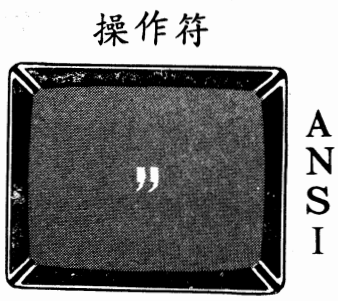
```
10 REM 'XOR' SIMULATION
20 PRINT "TYPE IN TWO NUMBERS BETWEEN 1 AND 10"
30 INPUT A,B
40 IF (A-3)*(B-8)<>0 THEN 90
50 IF A-3<>0 THEN 70
60 IF B-8=0 THEN 90
70 PRINT "EITHER A=3 OR B=8, BUT NOT BOTH"
80 GOTO 20
90 PRINT "BOTH CONDITIONS ARE TRUE OR BOTH ARE FALSE"
100 GOTO 20
999 END
```

运行实例

```
TYPE IN TWO NUMBERS BETWEEN 1 AND 10
? 3,5
EITHER A=3 OR B=8, BUT NOT BOTH
TYPE IN TWO NUMBERS BETWEEN 1 AND 10
? 3,8
BOTH CONDITIONS ARE TRUE OR BOTH ARE FALSE
```

参阅

OR, AND, NOT, +, *



在PRINT 语句中使用双引号（“）把需要打印的字母、数字或字符括起来。如果忽略引号，计算机就把字母认作变量，并将赋给它们的值打印出来。

例如：PRINT “A”打印字母A。而PRINT A则打印赋给变量A的值。

可以使用引号打印出一个数，但在这个数的前面没有通常留给＋或－符号的空格。如果要打印空格，可以把它们也放在引号中。

例如：

```
10 PRINT "       THE NUMBER";
20 PRINT "10"
```

将打印出

```
 THE NUMBER10
```

不能在引号内再“嵌套”引号。计算机不能识别PRINT 语句中哪个引号是终止引号。

例如， PRINT “I SAID “HELLO”TO HIM” 将无法运行。在这种情况下，内层的引号可用撇号来代替。例如，

PRINT “I SAID ‘HELLO’ TO HIM”

测试程序#1

```
10 REM 'QUOTED (")' PRINT STATEMENT TEST PROGRAM
20 A=5
30 B=10
40 PRINT "A+B=";A+B
50 PRINT "THE QUOTATION MARKS PASSED THE PRINT TEST."
99 END
```

运行实例

```
A+B= 15
THE QUOTATION MARKS PASSED THE PRINT TEST.
```

在最近的计算机中，在INPUT 语句中可以用引号。INPUT 语句可以同时起PRINT 语句和INPUT 语句的作用。

测试程序#2

```
10 REM 'QUOTED (")'INPUT STATEMENT TEST PROGRAM
20 INPUT "ASSIGN A VALUE TO VARIABLE X";X
30 PRINT "THE VALUE OF X IS";X
99 END
```

运行实例（输入5）

```
ASSIGN A VALUE TO VARIABLE X? 5
THE VALUE OF X IS 5
```

某些计算机要求把DATA语句中的字符串用引号括起来，而其它机器则仅仅在字符串之前或之后有空格，逗号或分号，或者字符串中包含空格、逗号或分号的情况下要求用引号括起来。

测试程序#3

```
10 REM 'QUOTED (")'DATA STATEMENT TEST PROGRAM
20 DATA " DATA STATEMENT "
30 READ A$
40 PRINT "QUOTES IN";A$;"PASSED THE TEST"
99 END
```

运行实例

```
QUOTES IN DATA STATEMENT PASSED THE TEST
```

在MICROSOFT BASIC中引号可以用于CSAVE和CLOAD，为存放在盒式磁带上的程序指定一个特定的名字。例如，

```
CSAVE "A"
CLOAD "A"
```

将在盒式磁带上存放一个程序，对它命名为“A”，然后只把命名为“A”的程序装回到计算机中。有关详细的内容和过程请看CLOAD和CSAVE。

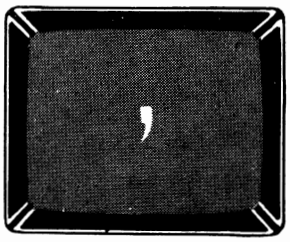
TRS—80 LEVEL I BASIC在PRINT #语句中使用引号把数据存贮到盒式磁带上去。

例如，`PRINT#A;",";B;",";C` 将把变量A、B、C的值存放到盒式磁带上去。磁带上三个值之间插入了二个逗号，这样在以后输入时能将各个值分隔开。详见PRINT。

参阅

PRINT，TAB，;(分号)，,(逗号)，DATA，READ，CSAVE，CLOAD

操作符



ANSI

逗号是一个使用范围广泛的操作符。最通常的一种用法是用于PRINT 语句中，它使各个打印项打印在事先规定好的水平的区域中。例如，PRINT 1， 2， 3，4每一个数打印在一个分隔的区中。

每一区通常允许最多为16个字符。每一行中打印区的数目是 4 到 8 个，取决于显示屏(或打印机)的行宽。

测试程序#1

```
10 REM TEST PROGRAM USING 'COMMA' FOR ZONING
20 PRINT "THE FOLLOWING LINE WILL PRINT IN 4 ZONES"
30 PRINT 1,2,3,4
40 PRINT "THE FOLLOWING LINES SHOW YOUR AVAILABLE ZONES"
50 PRINT 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
99 END
```

运行实例（每行的显示为 4 个区，最多 64 个字符）

```
THE FOLLOWING LINE WILL PRINT IN 4 ZONES
 1             2             3             4
THE FOLLOWING LINES WILL SHOW YOUR AVAILABLE ZONES
 1             2             3             4
 5             6             7             8
 9             10            11            12
 13            14            15            16
```

逗号也用于分隔数组各维的元素。例如，A（I,J，K）。逗号将I,J和K分隔开，使之成为三维数组中独立的元素。

测试程序#2

```
10 REM TEST PROGRAM USING 'COMMA' IN 2 DIMENSION ARRAY
20 A(1,1)=5
30 PRINT "A(1,1) =";A(1,1)
40 PRINT "LINE 20 PASSED THE TEST IF A(1,1) = 5."
99 END
```

运行实例

```
A(1,1)= 5
LINE 20 PASSED THE TEST IF A(1,1) = 5.
```

逗号在DATA，DIM，INPUT，ON— GOTO 和READ语句中用法是相似的，用来分隔各数据项。

下面的程序测试INPUT 和PRINT 语句中逗号的作用。

测试程序#3

```
10 REM 'COMMA' TEST PROGRAM
50 PRINT "ENTER THREE NUMBERS";
60 INPUT A,B,C
100 PRINT "NUMBER 1 =";A,2;"=";B,3;"=";C
999 END
```

运行实例（输入11、12、13）

```
ENTER THREE NUMBERS? 11,12,13
NUMBER 1 = 11   2 = 12          3 = 13
```

为了测试READ语句和DATA语句中逗号的作用，在上面最后一个测试程序（即测试程序#3）中加上下面这些行：

```
80 READ D,E,F
100 PRINT "NUMBER";D;"=";A,E;"=";B,F;"=";C
110 DATA 1,2,3
```

运行这个程序，运行结果与上面“运行实例”的相同。

为了测试ON—GOTO语句中逗号的作用，加上以下各行：

```
30 FOR X=1 TO 3
40 ON X GOTO 50,80,100
70 GOTO 90
90 NEXT X
```

运行这个程序，仍然得到上述“运行实例”的结果。

DIM语句中的逗号的作用可以由增加下面这一行来检查。

20DIM A（1），B（2），C（3），

增加了这一行不会改变运行结果。

逗号的其他应用请看PRINT USING，AT和@。

其它用法

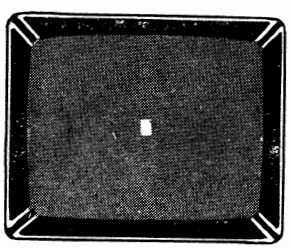
有些计算机（如带Palo Alto Tiny BASIC的）在LET语句中使用逗号，它的作用和大多数计算机使用冒号相同。在使用逗号的PRINT和INPUT语句中，PRINT和INPUT可以用#和—运算符加以修饰，如PRINT #－1，1，3，5或INPUT #－1，A$，B，C。

参阅

DATA，DIM，INPUT，ON-GOTO，AT，@，PRINT USING，READ.

句点在几乎所有的计算机中（除了只带整数BASIC的以外）都用来表示小数点。

在MICROSOFT BASIC（和其它一些BASIC）中，在LIST和EDIT后面加一个句点，使计算机LIST（列表）或EDIT（编辑）最后一个被送入的程序行，或在程序执行时引起错误的程序行。



测试程序

```
10 REM '.(PERIOD)' TEST PROGRAM
20 PRINT "THE PERIOD FOLLOWING THE LIST COMMAND"
30 PRINT "SHOULD LIST THE LAST LINE YOU ENTER"
99 END
```

运行实例

打入命令：LIST．（如果在LIST后面没有句点，则当然会LIST整个程序），计算机打印出：

END

把下面一行增加到测试程序中：

40 PRINT "THE PERIOD PASSED THE TEST"

打入命令：EDIT.（包括句点）。如果你的计算机有这个EDIT功能，计算机将打印出数字40后面跟一个光标。它表示计算机处于EDIT方式之中，已准备好修改40行（最后送入的行）。

其它用法

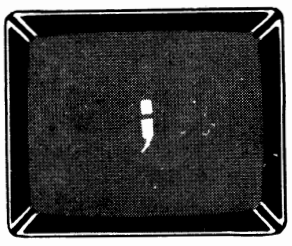
有几种计算机（如TRS—80 LEVEL Ⅰ 和 Ting BASIC的其它版本，句点用来作为缩写字的一部分。

例如，字母I在正常的情况下作为一个变量，但I.可以用作INPUT或INT(INTeger)的缩写，它取决于解释程序。此外，P.=PRINT，R.=RUN，L.=LIST，等等。

参阅

EDIT, LIST, INPUT, INT

操作符



在PRINT语句中,分号使几个打印的部分紧接在一行上,例如，PRINT “H”；“I”打印出HI。

测试程序#1

```
10 REM 'SEMICOLON' STRING TEST PROGRAM
20 PRINT "IF THIS SENTENCE IS PRINTED ";
30 PRINT "ON ONE LINE, THE TEST PASSED."
99 END
```

运行实例

```
IF THIS SENTENCE IS PRINTED ON ONE LINE, THE TEST PASSED.
```

在用分号分隔打印的数值或数值型变量时，会在数之前自动插入一个空格用作表示该数＋或－的符号。在数的后面又自动地插入一个空格（因为这样的空格经常是需要的)。当程序设计者企图得到一个有特殊要求的打印格式时，分号的这种特性可能会造成一定的困难。(APPLEⅡ和DEC—10不采用这种自动插入空格特性）。

例如，PRINT 1； 2； 3会在打印的各数间空两个空格。

测试程序#2

```
10 REM 'SEMICOLON' TEST PROGRAM WITH NUMERICS
20 A=5
30 PRINT "STUDY THE SPACING BETWEEN EACH OF THE NUMBERS."
40 PRINT 1;"2";"3";4;A;"6"; -7
50 PRINT "1234567890123456789Ø"
99 END
```

运行实例

```
STUDY THE SPACING BETWEEN EACH OF THE NUMBERS.
 1 23 4  5 6-7
1234567890123456789Ø
```

其它用法

有一些解释系统在连接的各字符串之间插入一个空格。这种特性（是比较罕见的特性）使得测试程序#1的20行中字母“D”后面的空格成为不需要的了。

参阅

COMMA, PRINT USING, TAB

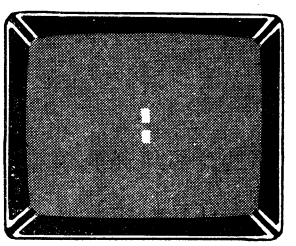
使用冒号可以使多个语句出现在同一行中。

例如：10 PRINT “SAMPLE LINE”:

LET A=10: GOTO 99

使三个独立的语句—PRINT，LET和GOTO同在行号为10的一行上。

操作符



测试程序#1

```
10 REM 'COLON (:) OPERATOR' TEST PROGRAM
20 PRINT "THIS TEST";:FOR X=1 TO 5000:
   NEXT X: PRINT" IS COMPLETE"
99 END
```

运行实例

THIS TEST（稍停）IS COMPLETE

GOTO, IF—GOTO, IF—THEN, ON—GOTO和其它引起分支的语句应当作为多语句行中的最后一个语句，以防止在未执行完整个行之前就分支转移出去了。

例如，在下面一行中：

```
10 FOR X=1 TO 10:NEXT X:GOTO 100:PRINT "THE LOST WORDS"
```

计算机在执行PRINT语句前就转到100行去了。没有办法打印出“THE LOST WORDS”。

大多数计算机不允许在多语句行中出现DATA语句。在其它的（如IMSAI）计算机中，如果在GOSUB语句后面跟有其它语句，也不执行这些语句（尽管由RETURN语句指出返回该行）。

在多语句行中使用IF—THEN语句应当特别小心。假如在IF——THEN同一行的后面跟有一些语句，某些BASIC（如Microsoft BASIC）只在条件为真时执行它们。其它的（如North Star BASIC）则在执行完IF—THEN语句后执行后面的语句，除非THEN后面出现分支转移。用下面的短程序可以测试你的计算机是属于哪一种。

测试程序#2

```
10 A = 1
20 IF A =2 THEN 30:PRINT "EXECUTES THE REST":GOTO99
30 PRINT "IGNORED THE REST"
99 END
```

冒号也被某些计算机（如DEC—10）用作PRINT USING语句中引用的“映象行”（image line）的第一个字符。

例如：

```
120 :$$******.**
130 PRINT USING 120, C
```

它产生的结果相同于：

```
130 PRINT USING "$$******.**", C
```

如果你的计算机无此功能

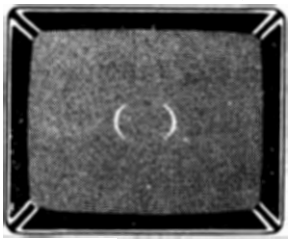
许多计算机不具备在一行中写多个程序行的功能。其它有些计算机有此功能，但是不用冒号而用反斜杠(\)来分隔各语句。个别的计算机则用分号。

参阅

\, ;, GOTO, IF-THEN, IF-GOTO, ON-GOTO, PRINT USING, IMAGE

操作符

ANSI



[ ]

在算术运算中用括号来决定执行算术运算的次序。在括号内的算术运算比括号外的先执行。假如一个算术运算被包在一对括号内，而这对括号又被包在另一对括号内（如此下去），则计算机首先执行“最内层”的那些运算。当在同一层的括号内时，先执行最左面的运算。

例如，A = 5 + (((2 * 4) - 2) * 3)。

计算机按下面的顺序执行算术运算：

A = 5 + ((8 - 2) * 3) = 5 + (6 * 3)

= 5 + 18 = 23

测试程序#1

```
10 REM '() PARENTHESES' TEST PROGRAM
20 A=(10*(5-3))/2
30 PRINT "A =";A
40 PRINT "THE PARENTHESES PASSED THE TEST IF A = 10"
99 END
```

运算实例

```
A = 10
THE PARENTHESES PASSED THE TEST IF A = 10
```

在某些逻辑运算中要求用括号来标识两个相比较的量。在TRS—80LEVEL I和其它Ting BASIC 中这是必要的。例如：

IF （A= 8） *（B= 6） TNEN 80

更详细的知识请参阅 * 和A N D。

括号亦用来在DIM语句和数组变量中把各元素括起来。更详细的知识请参阅DIM。

测试程序#2

```
10 REM '() PARENTHESES' TEST PROGRAM USING DIM AND ARRAYS
20 DIM A(5,5)
30 A(1,1)=20
40 PRINT "() PASSED THE TEST IN LINES";A(1,1);"AND";A(1,1)+10
99 END
```

运行实例

```
() PASSED THE TEST IN LINES 20 AND 30
```

大多数计算机有内部函数，用括弧将所用的数或字母（即自变量）括起来。例如：L O G （10）。

大多数计算机是用小括弧而不用中括弧（方括弧）的，用小括弧来代替方括弧。

例如：

```
[(A*B)/C]
```

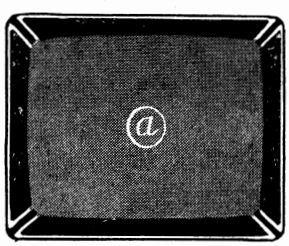
可以写成：

```
((A*B)/C)
```

参阅

```
*, +, AND, OR, DIM
```

操作符



@运算符被几种计算机（如TRS—80 LEVEL Ⅱ）用来对PRINT语句指定显示屏上的起止位置。它的值可以从 0 到1023，在它之后必须有一个逗号。例如：

PRINT @ 475，“HELLO”

在CRT（显示器）上从坐标475位置处开始打印HEL LO这个字。

测试程序

```
10 REM "@" PRINT MODIFIER TEST PROGRAM
20 PRINT @ 128, "2. IF THIS LINE IS PRINTED AFTER LINE 1."
30 PRINT @ 0, "1. THE @ OPERATOR PASSED THE TEST"
40 GOTO 40
99 END
```

运行实例

```
1. THE @ OPERATOR PASSED THE TEST
2. IF THIS LINE IS PRINTED AFTER LINE 1.
```

如果你的计算机无此功能

如果你的计算机不用@操作符作为PRINT的“修饰符”，则@的作用可以用下面的方法的来代替：用一些PRINT语句（以实现换行）和一些空格或TAB函数以达到CRT上同一个位置。

其它用法

@（AT）操作符被某些计算机（如North Star）用来抹掉显示屏上显示出来的最后一行，并执行一个“回车”。例如，打入：10 REM LINE DELETION TEST（但不按入ENTER或RETURN键），并按@键，这行就被抹去，光标回到左边顶端。

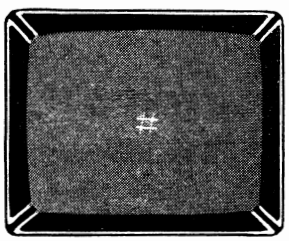
在某些计算机上，可以用下面的办法来完成上述功能：按RUB,（Crub out）、SCR（scratch）、←（左箭头），或SND（send）键，或在打入需要删去的行的行号之前和之后按ENTER（或RETURN）键。

Exatron Stringy Floppy系统用的每一个命令用一个@符号作前缀。例如：@NEW，@SAVE，@LOAD。

参阅

PRINT, AT, PRINT AT, DELETE

操作符



#用来指定变量作为“双精度”。双精度变量能够存贮包含17位的数（只打印出16位）。单精度变量只能精确到6位。每一次程序中用到双精度变量时，都必须在变量名的后面加#符号，以定义它为双精度。如果一个带#符号的变量在同一程序的DEFSNG或DEFINT语句中的变量表中出现，双精度符号（#）将优先于这些语句中的说明，将变量指定为双精度。

测试程序#1

```
10 REM '#' DOUBLE PRECISION OPERATOR TEST PROGRAM
20 DEFSNG A,B
30 A=1.2345678901234567
40 B#=1.2345678901234567
50 IF A=B# THEN 100
60 PRINT "A =";A
70 PRINT "B# =";B#
80 PRINT "THE # SIGN PASSED THE DOUBLE PRECISION TEST"
90 GOTO 999
100 PRINT "THE # SIGN FAILED THE DOUBLE PRECISION TEST"
999 END
```

运行实例

```
A = 1.23457
B# = 1.234567890123456
THE # SIGN PASSED THE DOUBLE PRECISION TEST
```

有几种计算机将#作为PRINT 语句的缩写符。详见PRINT。

测试程序#2

```
10 REM '#' TEST PROGRAM
20 #"THE # SIGN PASSED THE PRINT TEST"
99 END
```

运行实例

```
THE # SIGN PASSED THE PRINT TEST
```

有几种计算机用#操作符作为关系运算符“不等于”（<>）。

例如，IF A#B THEN 100告诉计算机当变量A的值不等于变量B 的值时分支转移到100行。

大多数计算机（用Microsoft BASIC 的)在PRINT USING语句中用#操作符来指出一个数或一个数值变量的打印位置。假如PRINT USING语句包含的#号比数的位数多，在小数点左边的未用到的#号处打印一个空格，而在小数点右边的未用到的#号处打印一个零。

例如，10 PRINT USING“#####.###”；12.5将打印出：

```
 12.500
```

在“1”之前有三个空格，它取代了三个未用的#符号。

详见PRINT USING。

测试程序#3

```
10 REM '#' PRINT USING TEST PROGRAM
20 PRINT "ENTER A VALUE FOR VARIABLE N";
30 INPUT N
40 PRINT "THE NUMBER";N;"IS PRINTED AS";
50 PRINT USING "####.##" ;N
99 END
```

运行实例

```
ENTER A VALUE FOR VARIABLE N? 12.5
THE NUMBER 12.5 IS PRINTED AS  12.50
```

具有文件处理功能的计算机在下面这样一些语句中使用#操作符，如 INPUT #，PRINT #，READ #，CLOSE #以及其它要指出从外部设备（如磁盘和盒式磁带）存取数据的设备号的那些语句。

测试程序#4（在 TRS —80 LEVEL Ⅱ 盒式磁带上存贮数据）

将盒式录音机置RECORD方式并运行这个程序：

```
10 REM 'PRINT#' TEST PROGRAM
20 PRINT "DATA SHOULD BE RECORDING ON CASGETTE TAPE"
30 PRINT#-1, "TEST" ,1,2,3
40 PRINT "PRINT# HAS COMPLETED THE DATA TRANSFER"
99 END
```

运行实例

```
DATA SHOULD BE RECORDING ON CASSETTE TAPE
PRINT# HAS COMPLETED THE DATA TRANSFER
```

为了测试计算机的READ #的功能，将磁带倒回，将录音机置于PLAY方式，清除内存，运行下面一个测试程序。

测试程序#5（从磁带读数据到计算机）

```
10 REM 'INPUT#' TEST PROGRAM
20 PRINT "THE COMPUTER SHOULD NOW READ DATA FROM
   CASSETTE"
30 INPUT#-1,A$,A,B,C
40 PRINT "THE INPUT# STATEMENT PASSED THE TEST IF"
50 PRINT A$;A;B;C;"IS PRINTED"
99 END
```

运行实例

```
THE COMPUTER SHOULD NOW READ DATA FROM CASSETTE
THE INPUT# STATEMENT PASSED THE TEST IF
TEST 1   2   3 IS PRINTED
```

在大的分时系统中（如 DEC —10），一个程序可以存取许多个不同的数据文件，每一个文件给定一个名字并且存贮在磁盘上。程序中有一个语句对每一个要用到的文件都给定一个号码，用这个文件号来调用文件（而不是用文件名）。#号理解为“号”——文件号（名），从（向）这个文件读（READ）、输入（INPUT）、打印（PRINT）数据或作其它的处理。

例如：

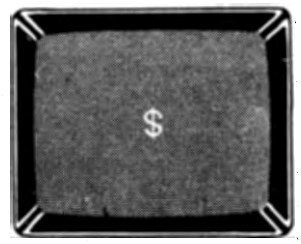
```
30 FILE #1,"TESTING"
80 READ #1,A,B,C,D,E
```

等等。

参阅

DEFDBL，DEFSNG，DEFINT，!，%，PRINT，REM，PRINT USING，READ#，<>

操作符



ANSI

在一个字母或字母和数字后面跟一个$符号用来说明变量是字符串变量。

在一个赋值语句中，向一个字符串变量赋值的字符串必须用引号括起来。例如，A$ = “THE BASIC HAND BOOK．”假如用INPUT语句向一个字符串变量输入信息，可以不必用引号括起来（见INPUT和READ）。

测试程序#1

```
10 REM '$' TEST PROGRAM WITH STRING STATEMENT
20 A$="LINE 20"
30 PRINT "THIS COMPUTER PASSED THE '$' TEST IN";A$
99 END
```

运行实例

```
THIS COMPUTER PASSED THE '$' TEST IN LINE 20
```

能够赋给一个字符串变量的字符个数是受计算机解释系统限制的。大多数具有字符串处理功能的计算机至少可以接收16个字符，有些可以达255个字符。

一些计算机（如Hewlett — Packard）要求用DIM语句为每一个字符串保留存贮空间〔如10 DIM A$（50）〕。见DIM和CLEAR。

下面的程序可以验证对字符串变量A$（读作A串）的赋值。

测试程序#2

```
10 REM '$' INPUT STRING WITH LENGTH TEST PROGRAM
20 PRINT "ENTER A KNOWN QUANTITY OF CHARACTERS"
30 INPUT A$
40 PRINT "COUNT THE NUMBER OF CHARACTERS PRINTED BELOW"
50 PRINT A$
99 END
```

运行实例（典型的）

在本次运行中输入的字符串为10个字符长。

```
ENTER A KNOWN QUANTITY OF CHARACTERS
? 1234567890
COUNT THE NUMBER OF CHARACTERS PRINTED BELOW
1234567890
```

假如这些字符都被打印出来而且没有出现错误信息，可以再运行一次，再增加输入10个字符。如果打印出来，再继续此过程直到字符串的末尾被截断，或者出现错误信息。

大多数具有处理字符串功能的计算机允许所有的字母作为字符串变量的标识（即名字）。有几种计算机则只允许用几个字母（如Radio Shack TRS — 80 LEVEL Ⅰ只允

许两个字符串，A$和B$，而且这两个串变量不能互相作比较）。

下面程序可以测定你的计算机允许使用的字母的范围。

测试程序#3

```
10 REM '$' (STRING) VARIABLE TEST PROGRAM
20 A$="LINE 20,"
30 PRINT "A$ PASSED THE TEST IN ";A$;
40 Z$=" AND Z$ IN LINE 40"
50 PRINT Z$
99 END
```

运行实例

```
A$ PASSED THE TEST IN LINE 20, AND Z$ IN LINE 40
```

许多具有字符串处理功能的计算机允许用字母、数字和其它符号的组合来指定字符串变量和数值变量。每一个变量必须以一个字母开头。通常解释程序只接收和处理前面几个字母数字（一般是两个）。例如，AB34K$和ABYN8 $（假如接收的话），一般是把它们作为同一个字符串变量（AB$）处理。因为它们的前两个字母是相同的。做下面一个实验将很快地表示出你的计算机的能力。

测试程序#4

```
10 REM '$' (STRING NAME) TEST PROGRAM
20 ABCDE$="TEST STRING"
30 PRINT "ABXYZ$ = ";ABXYZ$
40 PRINT "AB123$ = ";AB123$
50 PRINT "ONLY THE FIRST TWO LETTERS OF THE STRING
   NAME"
60 PRINT "WERE RECOGNIZED IF THE TWO STRINGS ARE
  . IDENTICAL"
99 END
```

运行实例

```
ABXYZ$ = TEST STRING
AB123$ = TEST STRING
ONLY THE FIRST TWO LETTERS OF THE STRING NAME
WERE RECOGNIZED IF THE TWO STRINGS ARE IDENTICAL
```

BASIC语句或函数所用的字不能用作字符串变量或数值变量名。如：SPRINTS$是一个非法的字符串名，因为它包含“PRINT”这个字。请查阅你所用的手册中“保留字”表，这些保留字是不能用于你的计算机程序中的（即不能用作变量名或数组名）。

大多数具有字符串功能的计算机允许字符串的比较。这就是：用关系运算符可以将一个字符串或字符串变量与另一个字符串或字符串变量逐个字符地进行比较。当字符串与字符串变量比较时，字符串要用引号括起来。

测试程序#5

```
10 REM '$' (STRING) COMPARISON TEST PROGRAM
20 READ A$
30 IF A$="WHOA" THEN 60
40 PRINT A$,
50 GOTO 20
60 PRINT "STRINGS CAN BE COMPARED."
70 DATA ONE, TWO, WHOA
99 END
```

运行实例

```
ONE            TWO            STRINGS CAN BE COMPARED.
```

其它用法

英国的Acorn ATOM将 $ 作为字符串变量的前缀（用$A而不用A$）

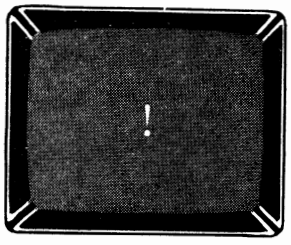
有一些计算机（如Apple Ⅱ）用$作为表示机器地址和（或）十六进数的前缀。如：$8 A = 138（十进数）。

参阅

DEFSTR, CHR$, FRE(string), INKEY$, LEN, LEFT$, MID$, RIGHT$, STR$, STRING$, VAL, LET, DATA, READ, DIM, CLEAR, TEXT, &, %

！（惊叹号）

操作符



！（惊叹号）用来指定变量为“单精度”。单精度变量可以存贮不超过7位的数（只打印出6位）。双精度为有17位的精度。

由于各变量是自动地取单精度，因此，！操作符在程序中被用来对已用DEFDBL 语句或#操作符说明为双精度的变量改变回单精度。

测试程序#1

```
10 REM '!' SINGLE PRECISION OPERATOR TEST PROGRAM
20 DEFDBL X,N
30 N=1234.56789012345
40 X=N
50 PRINT "DOUBLE PRECISION VARIABLE X=";X
60 X!=N
70 PRINT "SINGLE PRECISION VARIABLE X! =";X!
80 PRINT "THE '!' SINGLE PRECISION OPERATOR PASSED
   THE TEST"
99 END
```

运行实例

```
DOUBLE PRECISION VARIABLE X = 1234.56789012345
SINGLE PRECISION VARIABLE X! = 1234.57
THE '!' SINGLE PRECISION OPERATOR PASSED THE TEST
```

! 操作符亦被某些计算机（如使用Microsoft BASIC的计算机）用在PRINT USING语句中,它只打印出字符串中最左的一个字符。

例如：PRINT USING“! ”；“COMPUSOFT”打印出字母“C”。

详见PRINT USING和下一个测试程序。

测试程序#2

```
10 REM '! STRING SPECIFIER' TEST PROGRAM
20 PRINT "ENTER A SAMPLE WORD";
30 INPUT A$
40 PRINT "THE FIRST LETTER IN THE WORD ";A$;" IS ";
50 PRINT USING "!";A$
99 END
```

运行实例（输入HANDBOOK）

```
ENTER A SAMPLE WORD? HANDBOOK
THE FIRST LETTER IN THE WORD HANDBOOK IS H
```

其它用法

某些解释程序（如：COMPUMAX BASIC）用！作为REM的缩写符。

测试程序#3

```
10 PRINT "'! (REMARK) ' TEST PROGRAM"
20 ! PRINT " THE ! SIGN FAILED THE REM TEST"
30 REM THE ! SIGN FAILED THE REMARK TEST IF LINE 20 IS
   PRINTED
40 PRINT "THE ! SIGN PASSED THE TEST"
99 END
```

运行实例

```
'! (REMARK)' TEST PROGRAM
THE ! SIGN PASSED THE TEST
```

NORTH STAR用！作为PRINT的代用符。

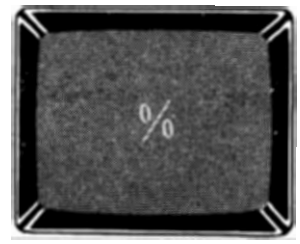
参阅

DEFDBL，DEFSNG，#，PRINT USING，DEFINT，CSNG，CDBL，PRINT，CINT，%

%被某些计算机（如那些使用Microsoft BASIC的，用来定义变量作为整型。当在变量的右边放一个%符号时，变量只能存贮整数值。

关于取整函数（INT函数）的详细知识参阅INT。

操作符



测试程序#1

```
10 REM '% INTEGER OPERATOR' TEST PROGRAM
20 I%=2.864
30 IF I%=2 THEN 60
40 PRINT "THE %. INTEGER OPERATOR FAILED THE TEST"
50 GOTO 99
60 PRINT "THE % INTEGER OPERATOR PASSED THE TEST"
99 END
```

运行实例

```
THE % INTEGER OPERATOR PASSED THE TEST
```

某些计算机（如使用Microsoft BASIC的）在PRINT USING语句中使用%操作符。它将字符串中最左边的若干字符打印出来，打印字符的个数等于两个%符号之间的格数（列数）。计算机将两个%号也计算在内，因此至少指定打印两个字符（指定字符串中一个字符，见！操作符）。

例如，PRINT USING "%　　%";"ABCDEFGHI"将打印出前四个字母"ABCD"，因为在两个%之间有两个空格（两个空格＋2个%＝4个字母）。详见PRINT USING。

测试程序#2

```
10 REM '%' STRING SPECIFIER TEST PROGRAM
20 A$"TESTIMONIAL"
30 PRINT "THE % OPERATOR PASSED THE STRING SPECIFIER ";
40 PRINT USING "%  %";A$
99 END
```

运行实例

```
THE % OPERATOR PASSED THE STRING SPECIFIER TEST
```

某些计算机在执行PRINT USING语句时用%符号来标志已经超过给定的"字段指示符"（#）范围的数。

例如，PRING USING "###.#";1234.56将打印出:%1234.6。如果小数

点左面的全部数字超过了给定的相应的“字段指示符”＃的个数，它仍完整地打印出这些数字。如果小数点右面的数字超过了“字段指示符”，则产生截断舍入。详见PRINT USING。

测试程序#3

```
10 REM '%' PRINT USING OVERFLOW TEST PROGRAM
20 A=123.45
30 PRINT "THE PRINT USING STATEMENT CHANGED *";A;"TO ";
40 PRINT USING "**.*" ;A
99 END
```

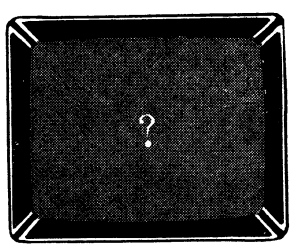
运行实例

```
THE PRINT USING STATEMENT CHANGED * 123.45 TO %123.5
```

参阅

INT, CINT, DEFINT, CSNG, CDBL, DEFSNG, DEFDBL, PRINT USING, IMAGE, FORMAT, !, #, &, \

操作符



许多计算机（如使用Microsoft BASIC的）用？（问号）来作为PRINT的缩写符。大多数（但非全部）计算机在列表打印（LIST）程序清单时会自动地将？号改为“PRINT”。

详见PRINT。

测试程序#1

```
10 REM '? (PRINT)' TEST PROGRAM
20 ? "THE ? SIGN PASSED THE PRINT TEST"
99 END
```

运行实例

```
THE ? SIGN PASSED THE PRINT TEST
```

计算机用？作为INPUT的提示符，表示它在等待你打入某些数据或回答。在按ENTER或RETURN键后，程序继续执行下去。

详见INPUT。

测试程序#2

```
10 REM '?' (INPUT REQUEST) TEST PROGRAM
20 PRINT "THE ? SIGN PASSED THE TEST"
30 PRINT "IF THE FOLLOWING LINE CONTAINS THE ? SIGN"
40 INPUT A
99 END
```

运行实例

```
THE ? SIGN PASSED THE TEST
IF THE FOLLOWING LINE CONTAINS THE ? SIGN
?
```

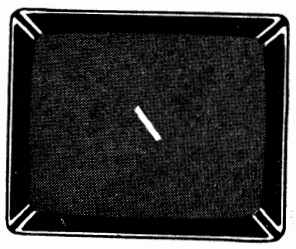
某些计算机（如使用Microsoft BASIC的）用？和CLOAD命令来把一个存贮在计算机内存中的程序和一个存贮在盒式录音带上的程序相比较。

为了测试这个特性，参考CLOAD中的测试过程。

参阅

PRINT，#，INPUT，CLOAD，LIST，!

操作符



有几种计算机用\符号使一个程序行中容纳多个语句。

例如，10 A=10\B=5\C=A-B\PRINT C把四个语句组合在一行上。

详见：（冒号）。

测试程序

```
10 REM '\ OPERATOR' TEST PROGRAM
20 PRINT "THIS TEST ";\FOR X=1 TO 500\
   NEXT X\PRINT "IS COMPLETE"
99 END
```

运行实例

```
THIS TEST IS COMPLETE
```

\（反斜杠）操作符被某些计算机（例如使用BASIC—80的计算机）用于PRINT USING语句中。它将打印出字符串中若干个字符，字符个数比用\括起来的空格的个数多2。计算机将\也计算在内，因此不能指定少于两个字符（指定字符串中一个字符的方法

请见! 操作符）。

例如，PRINT USING "\ \";"ABCDEFGHI" 将打印前四个字母 "ABCD"，因为在两个\号之间有两个空格（2个空格+2个\= 4个字符）。详见PRINT USING和%操作符。

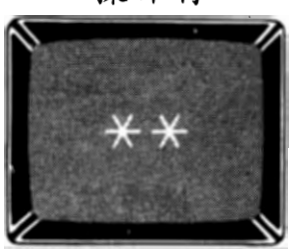
其它用法

有时在某些终端上用\把被用DEL或RUB键删除的字符包起来，以与其它字符分隔开（例如：ABC\DE\F表示DE两字符已被删掉，有效字符为ABCF。这在修改已打入的错字符时会常遇到的。

参阅

PRINT USING，%

操作符



在某些计算机（如DEC—10，DEC—BASIC—PLUS—2，HP 3000和那些使用MAXBASIC的）中用**（双星号）作为算术指数符号，用来计算一个基数的指定方次的值。

例如，2**3就是2的三次方或2^3。详见↑。

测试程序

```
10 REM '** (EXPONENTIATION)' TEST PROGRAM
20 PRINT "ENTER A BASE NUMBER";
30 INPUT B
40 PRINT "NEXT, ENTER THE EXPONENT";
50 INPUT E
60 A=B**E
70 PRINT "THE NUMBER";B;"TO THE";E;"POWER IS";A
30999 END
```

运行实例

```
ENTER A BASE NUMBER? 4
NEXT, ENTER THE EXPONENT? 3
THE NUMBER 4 TO THE 3 POWER IS 64
```

某些计算机（如使用Microsoft BASIC的）在PRINT USING语句中也使用**（双星号）。在指定打印的数的小数点的左边如有未用到的位置就打印出*号。这样做的最初的目的是为了防止某些人企图私自增加由计算机打印出来的支票中的数。

例如，PRINT USING "**######.##"；456.25将打印出***456.25。

#符号代表打印的数值的位置，未用到的位置由＊充满之。

详见PRINT USING。

参阅

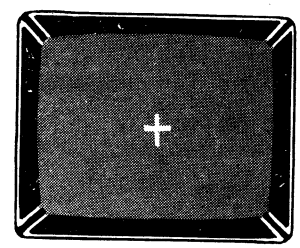
PRINT USING, ↑, ↓

+符号最通常的使用是用于算术加法。例如，

PRINT A + B

打印变量A和B值之和。

操作符



ANSI

测试程序#1

```
10 REM '+' MATH OPERATOR TEST
   PROGRAM
20 PRINT "ENTER A VALUE FOR
   VARIABLE A";
30 INPUT A
40 PRINT "ENTER A VALUE FOR
   VARIABLE B";
50 INPUT B
60 C=A+B
70 PRINT "THE SUM OF";A;"+";B;"IS";C
99 END
```

运行实例（输入6和14）

```
ENTER A VALUE FOR VARIABLE A? 6
ENTER A VALUE FOR VARIABLE B? 14
THE SUM OF 6 + 14 IS 20
```

某些计算机用+号作为IF—THEN语句中的逻辑“OR”操作符。

例如，10 IF（A = 8）+（B = 6）THEN 80读作：假如A的值等于8或（OR）B的值等于6则IF—THEN的条件满足，转到80行继续执行。

注意：（A = 8）和（B = 6）都要用括号括起来。由于在括号中括起来的是简单的等式，不可能有其它理由解释，此处+号作为逻辑OR来使用。

测试程序#2

```
10 REM '+' LOGICAL OPERATOR TEST PROGRAM
20 PRINT "ENTER A VALUE FOR VARIABLE A";
30 INPUT A
40 PRINT "ENTER A VALUE FOR VARIABLE B";
50 INPUT B
60 PRINT "A =";A,"B =";B
70 IF (A=8)+(B=6) THEN 100
80 PRINT "NEITHER A = 8 NOR B = 6"
90 GOTO 999
100 PRINT "EITHER A = 8 OR B = 6"
999 END
```

运行实例

```
ENTER A VALUE FOR VARIABLE A? 4
ENTER A VALUE FOR VARIABLE B? 6
A = 4              B = 6
EITHER A = 8 OR B = 6
```

其它用法

许多计算机用+号来将两个字符串连接起来成为一个字符串。例如，PRINT“H”+“I”将字符串“H”和字符串“I”连接起来形成“HI”这样一个字。

测试程序#3

```
10 REM '+' CONCATENATION TEST PROGRAM
20 A$="PASSED THE CON"
30 B$="CATENATION TEST"
40 PRINT "THE + SIGN ";
50 PRINT A$+B$
99 END
```

运行实例

```
THE + SIGN PASSED THE CONCATENATION TEST
```

某些计算机在PRINT USING语句中用+号使打印的数的前面自动加上+或－符号。

参阅

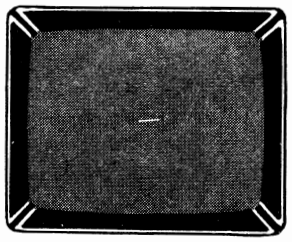
AND, *, $, PRINT USING, OR , &

用－号作为算术减法的符号，以求出两个数或数值变量值的算术差。例如，PRINT A －B 打印出变量A减去变量B的值。

－号亦用来作为算术运算中的负号。负号的简单的解释是："将它原来的值改变为相反的符号"。

例如，PRINT －（3－8）将3减去8，它的结果是－5。第一个－号（负号）使括弧内的符号变成相反的符号，打印出5（即＋5，＋号不印出）。

操作符



ANSI

测试程序

```
10 REM '- SIGN' TEST PROGRAM
20 A=3
30 B=6
40 C=B-A-(B-A)
50 PRINT "C =";C
60 PRINT "THE - SIGN PASSED THE TEST IF C = 0"
99 END
```

运行实例

```
C = 0
THE - SIGN PASSED THE TEST IF C = 0
```

其它用法

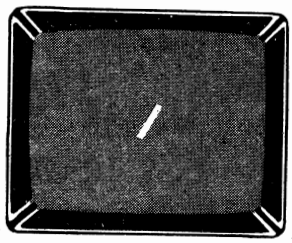
某些计算机在PRINT USING语句中用－号在打印的负数之前自动地加上前导号。

参阅

PRINT USING，+

/号用作算术除法符号，以求出两个算术变量的商。例如，8/4就是8÷4。

操作符



ANSI

测试程序

```
10 REM '/ DIVISION SIGN' TEST PROGRAM
20 A=8
30 B=4
40 C=A/B
50 PRINT "C =";C
60 PRINT "THE / SIGN PASSED THE TEST IF C = 2"
99 END
```

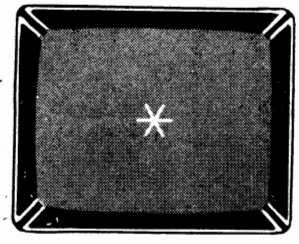
运行实例

```
C = 2
THE / SIGN PASSED THE TEST IF C = 2
```

某些解释系统（如Palo Alto Tiny BASIC）只取商的整数值，因此PRINT 15/4得到的商为3。

操作符

ANSI



*号（星号）用来作算术乘法的符号（代替×号），以求出两个数或数值变量之积。

测试程序#1

```
10 REM '*' MATH OPERATOR TEST
   PROGRAM
20 A=5
30 B=A*6
40 PRINT "* PASSED THE TEST IN LINE";B
99 END
```

运行实例

```
* PASSED THE TEST IN LINE 30
```

其它用法

某些计算机也用*号作为“逻辑运算”的操作符，表示AND（与）。例如：

IF （A = 8）*（B = 6）THEN 80 读作：“如果A的值等于8和（AND）B的值等于6，则IF —THEN的条件成立，转去80行继续执行。”

注意，（A = 8）和（B = 6）都要用括弧括起来。这便于判断所用的*是作为乘号还是作为逻辑AND。

测试程序#2

```
10 REM '*' LOGICAL 'AND' TEST PROGRAM
20 A=8
30 B=6
40 IF (A=8) * (B=6) THEN 70
50 PRINT "* FAILED THE TEST AS AND OPERATOR"
60 GOTO 99
70 PRINT "* PASSED THE AND OPERATOR TEST"
99 END
```

运行实例

```
* PASSED THE AND OPERATOR TEST
```

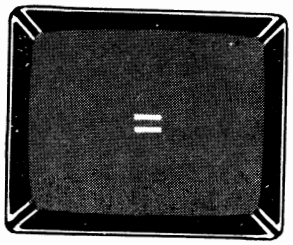
某些计算机在PRINT USING语句中用＊号来指定所打印的数值或字符串的格式。见PRINT USING中细节。

参阅

AND, PRINT USING, OR, +

=号可以用作赋值操作符。例如，A = 3 + 5，将值8赋给变量A。

操作符



测试程序#1

```
10 REM TEST PROGRAM USING = AS
   ASSIGNMENT OPERATOR
20 A=4
30 B=6
40 C=A+B
50 PRINT "C =";C
60 PRINT "THE = SIGN PASSED THE TEST IF C = 10"
99 END
```

```
C = 10
THE = SIGN PASSED THE TEST IF C = 10
```

大多数计算机也用=号作为关系运算符，用来比较两个数值是否相等。例如， IF A=B THEN100告诉计算机当数值变量A等于数值变量B时转到100行。如果等号条件不成立（即A≠B），测试“失败”，程序执行下一行并继续下去。

大多数计算机也用=号来作字符串的比较符。用这个特性可以使一个字符串或字符串变量和另一个字符串或字符串变量逐个字符相比较。例如， IF A$="ABCD" THEN100解释程序将内存中的字符串变量A$和引号中括起来的字符（由左至右）逐个地比较它们的ASCII代码。如果所有字符的ASCII码都相等，则计算机会使流程分支转移或“跳”到100行去。如果不是全部字符的ASCII码相等，则测试“失败”，接着执行程序的下一行。

测试程序#2

```
10 REM TEST OF = SIGN AS NUMERIC COMPARISON OPERATOR
20 A=5
30 IF A=5 THEN 60
40 PRINT "= SIGN FAILED NUMERIC COMPARISON TEST"
50 GOTO 99
60 PRINT "= SIGN PASSED NUMERIC COMPARISON TEST"
99 END
```

运行实例

```
=SIGN PASSED NUMERIC COMPARISON TEST
```

测试程序#3

```
10 REM TEST PROGRAM USING = FOR STRING COMPARISON
20 A$ = "ABCDE"
30 IF A$ ="ABCDE" THEN 60
40 PRINT "THE = SIGN FAILED THE STRING COMPARISON TEST"
50 GOTO 99
60 PRINT "THE = SIGN PASSED THE STRING COMPARISON TEST"
99 END
```

运行实例

```
THE = SIGN PASSED THE STRING COMPARISON TEST
```

其它用法

不同的解释系统允许所比较的字符串的长度是不同的。有的只允许一个字符与另一个字符相比较，而另外一些则允许任意个数的字符进行比较，例如一个人的名字和地址（或其它）等等。

将＞或＜和=组合在一起是非常普遍的。有时它们只能用来对数值量进行比较，但在大多数计算机中可以比较二个字符串，首先自动地将字符转换为ASCII码，然后按它们的ASCII码进行比较。

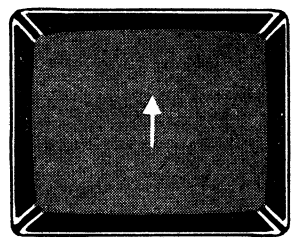
参阅

>, <, $, <=, >=, EQ, GE, GT, LE, LT, NE, <>

↑（上箭头）用来作为算术指数符号，以计算一个基数的指定的方次。某些计算机用 ^ 代替↑。

例如，2 ↑ 3就是2的立方或2^3。

操作符



ANSI

测试程序#1

```
10 REM ' ↑(EXPONENTIATION)' TEST PROGRAM
20 PRINT "ENTER A BASE NUMBER";
30 INPUT D
40 PRINT "NEXT ENTER A POWER NUMBER";
50 INPUT F
60 P=D↑F
70 PRINT "THE NUMBER";D; "TO THE";F;"POWER IS";P
30999 END
```

运行实例（输入4和3）

```
ENTER A BASE NUMBER? 4
NEXT ENTER A POWER NUMBER? 3
THE NUMBER 4 TO THE 3 POWER IS 64
```

↑符号也用来计算一个数的根的值，求指定的数n的倒数次方（1/n次方，即n次方根）。

例如，8 ↑ (1/3) 就是8的立方根，或 $\sqrt[3]{8}$。

测试程序#2

```
10 REM ' ↑(USED AS A RADICAL SIGN)' TEST PROGRAM
20 PRINT "ENTER A BASE NUMBER";
30 INPUT B
40 PRINT "NEXT ENTER A ROOT NUMBER";
50 INPUT N
60 R=B↑(1/N)
70 PRINT "THE";N;"ROOT OF";B;"IS";R
30999 END
```

运行实例

```
ENTER A BASE NUMBER? 64
NEXT ENTER A ROOT NUMBER? 3
THE 3 ROOT OF 64 IS 4
```

↑

大多数具有PRINT USING的计算机在格式字符串中如果有4个↑则打印出一个数的指数或‘E’形式。例如，PRINT USING“##.####↑↑↑↑”，12345打印出12.345E+03。

↑的其它表示方法

有些计算机用**。其它有的用^表示幂的计算。

如果你的计算机无此功能

如果上面这些在你的计算机上试验都失败，可以用下面的子程序代替。

把求LOG和EXP的子程序加到下面的子程序中才能使它工作（为节约篇幅在此不再印出）。（下面GOSUB 30150 和GOSUB 30200 是调用LOG子程序和EXP子程序的。见LOG和EXP一节。）

```
30000 GOTO 30999
30100 REM * EXPONENTIATION SUBROUTINE * INPUT X,Y;
      OUTPUT P
30102 REM ALSO USES A, B, C, D, E, F AND L INTERNALLY
30104 P=0
30106 IF X<>0 THEN 30112
30108 IF Y<0 THEN 30122
30110 RETURN
30112 P=1
30114 IF Y=0 THEN 30140
30116 F=X
30118 IF X>0 THEN 30130
30120 IF Y=INT(Y) THEN 30126
30122 PRINT "***";X;"TO THE";Y;"POWER IS UNDEFINED ***"
30124 STOP
30126 P=1-2*Y+4*INT(Y/2)
30128 X=-X
30130 GOSUB 30150
30132 X=Y*L
30134 GOSUB 30200
30136 P=P*E
30138 X=F
30140 RETURN
```

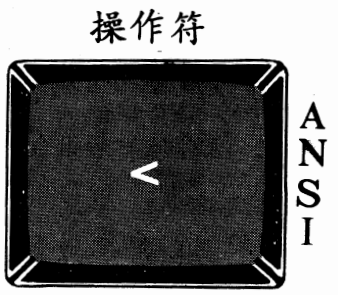
为了将此子程序用于上面的测试程序中，要作以下一些改变：

```
35  X = D
55  Y = F
60  GOSUB   30100
```

参阅

EXP，LOG，**，PRINT USING

在IF—THEN语句中用<符号作为关系运算符“小于”以比较两个数值。例如，IF A <B THEN100 告诉计算机当变量A的值小于变量B的值时转移到100行去。



测试程序#1

```
10 REM '< RELATIONAL OPERATOR' TEST PROGRAM
20 A=10
30 IF A 20 THEN 60
40 PRINT "THE < SIGN FAILED THE TEST"
50 GOTO 99
60 PRINT "THE < SIGN PASSED THE TEST"
99 END
```

运行实例

```
THE < SIGN PASSED THE TEST
```

其它用法

大多数计算机可以用<符号来比较字符串。<符号将两个字符串中的字符自左而右地比较它们的ASCII码。当出现第一个不同的字符时就可以确定它们的大小关系了。

例如，字符串“A B C D E F”小于字符串“A B D”，尽管第一个字符串的字符多。这是由于第一个字符串中的C的ASCII码（十进制67）小于第二个字符串中的D的ASCII码（十进制68），“A B C D E F”<“A B D”为真。

如果各字符串有相同的字符序列，则长的字符串为“大”。例如，字符串“ABCD”大于字符串“A B C”。

某些解释系统对参加比较的字符串的字符个数有一定限制。

测试程序#2

```
10 REM '< STRING OPERATOR' TEST PROGRAM
20 A$="ABC"
30 B$="ABCD"
40 C$="B"
50 IF A$<B$ THEN 80
60 PRINT "THE < SIGN FAILED THE TEST IN LINE 50"
70 GOTO 999
80 IF B$<C$ THEN 110
90 PRINT "THE < SIGN FAILED THE TEST IN LINE 80"
100 GOTO 999
110 PRINT "THE < SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE < SIGN PASSED THE TEST
```

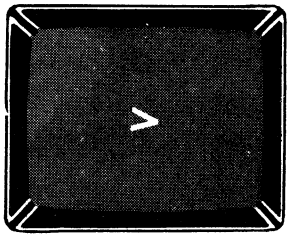
<常常与=号组合在一起成为<=运算符，以及与>组成“不等于”运算符<>或><。

参阅

>，<>，=，$，IF-THEN，<=，>=，EQ，GE，GT，LE，LT，NE

操作符

ANSI



在IF—THEN语句中用>符号作为关系运算符“大于”以比较两个数值。例如，IF A>B THEN 100 告诉计算机：当变量A的值大于变量B的值时转移到100行去。

测试程序#1

```
10 REM '> RELATIONAL OPERATOR' TEST PROGRAM
20 A=20
30 IF A > 10 THEN 60
40 PRINT "THE > SIGN FAILED THE TEST"
50 GOTO 99
60 PRINT "THE > SIGN PASSED THE TEST"
99 END
```

运行实例

```
THE > SIGN PASSED THE TEST
```

其它用法

大多数计算机可以用>符号来比较字符串。>符将两个字符串中的字符自左而右地逐个比较它们的ASCII码。当出现第一个不同的字符时就可以确定它们的大小关系了。

例如，字符串“A B D”大于字符串“A B C D E F”，尽管第一个字符串的字符个数少。这是由于在第一个字符串中的D的ASCII码（十进制数68）大于（或68在67之后），第二个字符串中的C的ASCII码（十进数67）“A B D”>“A B C D E F”为真。

如果各字符串有相同的字符序列，则长的字符串为“大”。例如，字符串“A B C D”大于字符串“A B C”。

某些解释系统对参加比较的字符串的字符个数有一定限制。

测试程序#2

```
10 REM '> STRING OPERATOR' TEST PROGRAM
20 A$="ABCD"
30 B$="ABC"
40 C$="B"
50 IF A$ > B$ THEN 80
60 PRINT "THE > SIGN FAILED THE TEST IN LINE 50"
70 GOTO 999
80 IF C$ > B$ THEN 110
90 PRINT "THE > SIGN FAILED THE TEST IN LINE 80"
100 GOTO 999
110 PRINT "THE > SIGN PASSED THE TEST"
999 END
```

运行实例

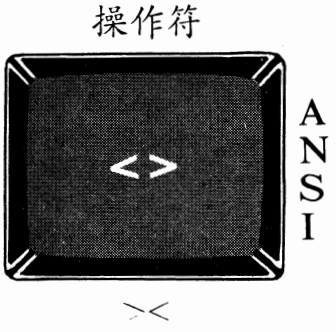
```
THE > SIGN PASSED THE TEST
```

>常常与=组合成>=运算符，以及和<组成“不等于”运算符<>或><。

参阅

<, <>, =, GT, LT, NE, $, IF-THEN, >=, <=, EQ, GE, LE

在IF—THEN语句中用<>符号作为关系运算符“不等于”以比较两个数值。例如，IF A <>B THEN100 告 诉计算机：当变量A的值不等于变量B的值时转移到100行去。



测试程序#1

```
10 REM ' <> RELATIONAL OPERATOR'
   TEST PROGRAM
20 A=10
30 IF A <> 20 THEN 60
40 PRINT "THE <> SIGN FAILED THE TEST"
50 GOTO 99
60 PRINT "THE <> SIGN PASSED THE TEST"
99 END
```

运行实例

```
THE <> SIGN PASSED THE TEST
```

其它用法

大多数计算机可以用<>符号来比较字符串。<>符将两个字符串中的字符自左而右地比较它们的ASCII码。当出现第一个不同的字符时就可以确定它们的关系了。

例如，IF A$<>“A B C ”THEN 100解释系统将存贮在字符串变量A$中的每一个字符和引号中的各字符相比较。如果出现有一个字符不同,或一个字符串长于另一个，则<>的条件成立，计算机使流程转移到100行。

某些解释系统对参加比较的字符串的字符个数有一定限制。

测试程序#2

```
10 REM <> STRING OPERATOR TEST PROGR
20 A$="ABCDE"
30 IF A$ <> "ABCD" THEN 60
40 PRINT "THE    SIGN FAILED THE TEST IN LINE 30"
50 GOTO 999
60 IF A$ <> "ABCDE" THEN 80
70 GOTO 100
80 PRINT "THE    SIGN FAILED THE TEST IN LINE 60"
90 GOTO 999
100 PRINT "THE <> SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE <> SIGN PASSED THE TEST
```

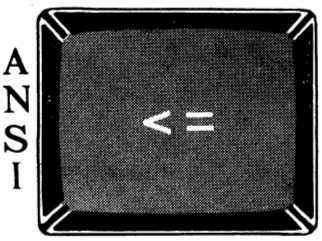
替代的形式

某些计算机使用操作符><或≠。

参阅

#, <, >, IF-THEN, $, =, <=, >=, EQ, GE, GT, LE, LT, NE

操作符



=<

≤

在IF—THEN语句中用<=符号作为关系运算符“小于或等于”以比较两个数值。例如，IF A<=B THEN 100 告诉计算机：当变量A的值小于或等于变量B的值时转移到100行去。

测试程序#1

```
10 REM ' <= RELATIONAL OPERATOR
   TEST PROGRAM
20 A=10
30 IF A <= 20 THEN 60
40 PRINT "THE <= SIGN FAILED THE TEST IN LINE 30"
50 GOTO 999
60 IF A<=10 THEN 90
70 PRINT "THE <= SIGN FAILED THE TEST IN LINE 60"
80 GOTO 999
90 PRINT "THE <= SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE <= SIGN PASSED THE TEST
```

其它用法

大多数计算机可以用＜＝符号来比较字符串。＜＝符将两个字符串中的字符自左而右地比较它们的ASCII码。当出现第一个不同的字符时就可以确定它们的关系了。

例如，字符串“A B C D E F”小于字符串“A B D”，尽管第一个字符串的字符多。这是由于在第一个字符串中的C的ASCII码（十进制数67）小于第二个字符串中的D的ASCII码（十进数68）（或67在68前面），“A B C D E F”＜＝“A B D”为真。同样，如果两个字符串有相同的字符而且长度相等，则满足＜＝关系。

如果各字符串有相同的字符序列，则长的字符串为“大”。例如，字符串“A B C D”大于字符串“A B C”。

某些解释系统对参加比较的字符串的字符个数有一定限制。

测试程序#2

```
10 REM '<= STRING OPERATOR' TEST PROGRAM
20 A$="ABC"
30 B$="ABCD"
40 C$="B"
50 IF A$ <=B$ THEN 80
60 PRINT "THE <= SIGN FAILED THE TEST IN LINE 50"
70 GOTO 999
80 IF A$ <="ABC" THEN 110
90 PRINT "THE <= SIGN FAILED THE TEST IN LINE 80"
100 GOTO 999
110 IF B$ <= C$ THEN 140
120 PRINT "THE <= SIGN FAILED THE TEST IN LINE 110"
130 GOTO 999
140 PRINT " THE <= SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE <= SIGN PASSED THE TEST
```

替代的符号

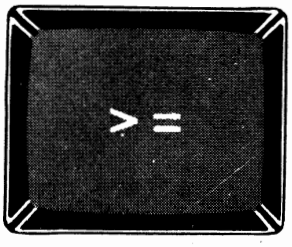
某些计算机用 =＜或≤符号来代替之。其它一些计算机允许用=＜作为可选用的形式。

参阅

IF-THEN, <, =, >, =, <>, $, LE, LT, EQ, GT, GE, NE

操作符

ANSI



=>

>

在IF—THEN语句中用>=符号作为“大于或等于”比较符，以比较两个数值（或字符串，如果允许的话）。

例如，IF A >= B THEN 100 告诉计算机：如果变量A的值大于或等于变量B的值时，计算机就使流程转移到100行去。

测试程序#1

```
10 REM ' >= RELATIONAL OPERATOR' TEST PROGRAM
20 A=20
30 IF A >= 10 THEN 60
40 PRINT "THE >= SIGN FAILED THE TEST IN LINE 30"
50 GOTO 999
60 IF A >= 20 THEN 90
70 PRINT "THE >= SIGN FAILED THE TEST IN LINE 60"
80 GOTO 999
90 PRINT "THE >= SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE >= SIGN PASSED THE TEST
```

其它用法

大多数计算机允许用>=运算符来比较两个字符串。它自左而右地比较两个字符串中的各个字符。当出现第一个不同的字符时就可以确定它们的关系了。

例如，字符串“A B D”大于字符串“A B C D E F”，尽管第一个字符串短。这是由于第一个字符串中的D的ASCII码（十进数68）大于第二个字符串中C的ASCII码（十进数67）。“A B D”>“A B C D E F”为真。同样，如果两个字符串有相同的字符而且它们的长度相等，则>=关系满足。

如果各字符串的字符序列相同，则长的字符串为“大”。字符串“A B C D”大于字符串“A B C”。

有些计算机对参加比较的字符串中字符的个数有一定限制。

测试程序#2

```
10 REM   = STRING OPERATOR  TEST PROGRAM
20 A$="ABCD"
30 B$="ABC"
40 C$="B"
50 IF A$   = B$ THEN 80
60 PRINT "THE   = SIGN FAILED THE TEST IN LINE 50"
70 GOTO 999
80 IF A$  ="ABCD" THEN 110
90 PRINT "THE  = SIGN FAILED THE TEST IN LINE 80"
100 GOTO 999
110 IF C$  = B$ THEN 140
120 PRINT "THE   = SIGN FAILED THE TEST IN LINE 110"
130 GOTO 999
140 PRINT "THE  = SIGN PASSED THE TEST"
999 END
```

运行实例

```
THE  = SIGN PASSED THE TEST
```

替代的符号

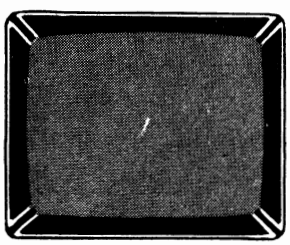
有些计算机用运算符 = >或≥来代替。

参阅

IF-THEN, >, <, <=, <>, EQ, GE, LE, LT, NE

许多计算机用撇号 ' 作为REM（注释）语句的缩写。有关这方面的规定请参阅REM。

语 句



测试程序#1

```
10 REM '(APOSTROPHE) TEST PROGRAM
20 'PRINT "THE APOSTROPHE FAILED THE REM TEST"
30 REM THE APOSTROPHE FAILED THE TEST IF LINE 20 IS PRINTED
40 PRINT "THE APOSTROPHE PASSED THE REM TEST"
99 END
```

运行实例#1

```
THE APOSTROPHE PASSED THE REM TEST
```

其它用法

有些计算机在PRINT 语句中用撇号代替引号将字符串括起来。

测试程序#2

```
10 REM '(APOSTROPHE) * USED AS QUOTES * TEST PROGRAM
20 PRINT 'THE APOSTROPHE PASSED THE QUOTATION MARK TEST'
99 END
```

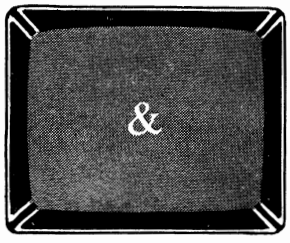
运行实例#2

```
THE APOSTROPHE PASSED THE QUOTATION MARK TEST
```

参阅

REM, PRINT, !, "

操作符



有些计算机用& 符号作为“连接”操作符,将两个字符串连接在一起并把它作为一个字符串来存贮。

例如，假如A$ = “SORT ”和B$ = “WARE ”，则C $ = A$ & B$ 将“SOFTWARE ”存放在C$中（参阅 + 。）

某些计算机（如DEC —10）用& 来指出：PRINT USING语句中预设的位置放不下要打印的数字。例如：PRINT USING “# # # ”; - 1 2 3 打印& - 1 2 3（参阅%）。

小心：Applesoft BASIC 在内部使用& 但用户是不能使用的。在这种BASIC 中如果用下面列出的例子（使用& 的），会导致执行地址$3F 5，并且要求改正措施以使计 算机恢复正常控制。

有少数计算机（如PDP—11）用& 来标明：在一行内容纳不下一个长语句,在下一行继续之。

例如：

```
IF (5 - X) * (X - 1) > 0 THEN &
    PRINT "THE VALUE OF X LINES BETWEEN 1 AND 5"
```

某些具有MAT INPUT 功能的计算机（如DEC—10）用& 来指出：将在下一行上打入其它要输入的值（见MA T INPUT）。

如，当执行下列程序语句：

```
DIM C(25)
.
.
PRINT  PLEASE ENTER THE VALUES OF ALL CHECKS OUTSTANDING'
MAT INPUT C
```

用户可以在打入完一行的数值后，打入& 符号并按回车键，然后在下一行继续输入其余的数值如：

```
?12.50, 55.00, 37.84, 163.00, 43.00, 100.00, 5.198 &
?18.80, 25.00 (回车)
```

而MAX BASIC 都是用& 来标志MA T INPUT 语句所要求输入的值“至此结束”。

有的计算机在一个数的前面加& 前缀，以表示一个机器地址或一个十六进制的数。如B 5是一个变量名，而&B 5是一个十六进制数，它等于十进数181（见$ ）

有少数计算机（如PDP—11的BASIC —PLUS）可用& 代替PRINT。

```
10 & "THIS IS A PRINT LINE"
```

打印出：

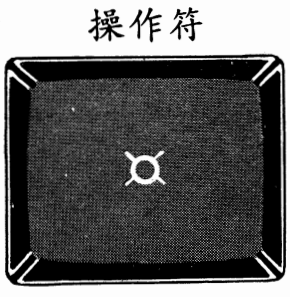
```
THIS IS A PRINT LINE
```

参阅

PRINT，PRINT USING，MAT INPUT，+，$，%

¤

¤（发音为SOL，形似太阳）在计算机中它等价于$ 符号。



在世界上绝大多数计算机使用$ 符号，而Swedish ABC 80计算机用¤代替$ 。

在合法的变量后跟一个¤形成字符串变量（如A¤代替A$）。CHR¤与CHR $等价；MID¤与MID $等价，SPACE¤与SPACE$等价，等等。

若一个程序包含带¤的任何变量，把它作为后面跟一个$ 处理，认为是字符串。

参阅

$，CHR$，LEFT$，MID$，RIGHT$，SPACE$，STRING$，NUM$

典型的计算机BASIC简介

（1）IBM－PC BASIC

1．命令一览表

命　　令	功　　能
AUTO	自动产生行号
BLOAD	将二进制数据（例如机器语言程序）装入内存
BSAVE	将二进制数据送到磁盘保存
CLEAR	清除程序变量，释放内存区
CONT	继续运行程序
DELETE	删除指定的程序行
EDIT	显示出需要修改的程序行
FILES	列出软盘文件目录
KILL	删除软盘上的文件
LIST	显示程序清单
LLIST	打印程序清单
LOAD	装入一个程序
MERGE	将保存在软盘上的程序和在内存中的程序拼接起来
NAME…AS…	将软盘文件改名
NEW	清除当前程序和变量
RENUM	重编程序行号
RESET	初始化软盘信息
RUN	运行程序
SAVE	将内存中当前程序保存在软盘上
SYSTEM	BASIC结束，关闭所有文件，返回DOS
TRON	置跟踪状态
TROFF	脱离跟踪状态

2．语句一览表

A．非输入/输出语句

语　　句	功　　能
CALL	调用机器语言程序

语　　句	功　　能
CHAIN	链接程序。调用一个程序并将变量传递给它
COM(n)ON/OFF/STOP	通讯功能的开启和停止
COMMON	给链接程序传递变量
DATE$	置日期
DEF FN	定义数值或字符中函数
DEF＃type	定义变量类型，type为INT，SNG，DBL，STR之一
DEF SEG	定义内存用户可用区间
DEFUSR	定义机器语言子程序的起始地址
DIM	说明数组可用的最大下标值并分配给空间
END	结束程序
ERASE	清除数组空间
ERROR	显示某一错误代码的信息
FOR	设置并执行循环
GOSUB	转子程序
GOTO	无条件转移
IF…THEN…ELSE	条件判断
KEY　ON/OFF/LIST	显示软按键（功能键）或关闭显示软按键
KEY	设置软按键
KEY(n)ON/OFF/STOP	启停功能键或光标控制键
LET	赋值
MID$	将一个字符串中的一部分（子串）用另一个字符串来代替
NOTOR	启停磁带机马达
NEXT	循环出口
ON COM(n)GOSUB	根据通信缓冲区的信息转子程序
ON ERROR GOTO	出错转移
ON…GOSUB	控制转移（执行子程序）
ON…GOTO	控制转移
ON KEY(n)GOSUB	根据功能键或光标控制键转子程序
ON PE NGOSUB	根据光笔状态转子程序
ON STRIG(n)GOSUB	根据游戏操纵杆状态转子程序
OPTION BASE	定义数组下标下限
PEN　ON/OFF/STOP	启停光笔功能
POKE	向内存指定地址写入数据
RANDOMIZE	重置启随机数发生器
REM	注释

语　　句	功　　能
OPEN “COM…”	打开通讯文件
OUT	送数据到端口
PAINT	屏幕着色
PLAY	放音乐
PRINT	屏幕显示
PRINT USING	格式显示
PRINT #	向文件写数据
PRINT # USING	按格式输出给文件
PRESET	在屏幕上用背景色画一个点
PSET	在屏幕上画一个点，如未指定颜色则用前景色画点
PUT #	向随机文件写数据
PUT	给屏幕送绘图信息
READ	从数据区读数
RSET	给随机文件缓冲区串变量按右对齐赋值
SCREEN	置屏幕显示类型
SOUND	喇叭发声
WIDTH	置屏幕宽
WRITE	在屏幕上输出
WRITE #	输出数据到文件

3．函数

A．算术函数

函　　数	功　　能
ABS（X）	取 x 的绝对值
ATN（X）	反正切
CDBL（X）	转换成双精度
CINT（X）	转换成整形
COS（X）	余弦
CSNG（X）	转换成单精度
EXP（X）	e^x
FIX（X）	X 截尾取整
INT（X）	不大于X的最大整数
LOG（X）	lnX

语　　句	功　　能
RND（X）	产生一个随机数
SGN(X)	取X的符号
SIN（X）	$\sin x$
SQR（X）	$\sqrt{x}$
TAN（X）	$\tan x$

B．与串有关的数值函数

函　　数	功　　能
ASC（X$）	取X$中第一个字符的ASCII码
CVI(X$),CVS(X$), CVD(X$)	将随机文件缓冲区串变量变为整形、单精度型、双精度型数
INSTR(n, X$, Y$)	求出子串X$在Y$中的位置
LEN（X$）	X$的长度
VAL（X$）	将X$变成数值量

C．输入、输出和其它用途的函数

函　　数	功　　能
CSRLIN	得到光标的垂直位置
EOF（f）	指示文件f的文件结束状态
ERL	取最后产生错误的行号
ERR	取最后一次错误的错误代码
FRE（X$）	内存中BASIC不使用的空间
INP（n）	从端口n读一个字节
LOC（f）	末次读写记录的位置
LOF（f）	文件长度
LPOS（n）	打印机的打印头位置
PEEK（n）	读内存地址n的一个字节
PEN（n）	读光笔
POINT（X，Y）	取点（X，Y）的颜色
POS（n）	光标列位置
SCREEN（行、列）	得到指定位置上的字符的ASCⅡ码

函　　数	功　　能
STICK（n）	取游戏操纵杆座标
STRIG（n）	取游戏操纵杆状态
USR（X）	调用机器语言子程序，自变量为*x*
VARPTR（X）	取变量*x*在内存中地址
VARPTR（#f）	取文件控制块地址

D．串函数（返回串值）

函　　数	功　　能
CHR$（n）	求ASCII码为*n*的字符
LEFT$（X$, n）	取X$左端*n*个字符
MID$（X$, n, m）	取X$中第*n*字符开始的*m*个字符
RIGHT$（X$, n）	取X$中右端*n*个字符
SPACE$（n）	得*n*个空格的串
STRING$（n, m）	取ASCII码为*m*的字符*n*次
STRING$（n, X$）	取X$的第一个字符*n*次

E．输入、输出和其它类型

函　　数	功　　能
DATE$	取系统日期
HEX$（n）	把*n*转换成十六进制的字符
INKEY$	从键盘读一个字符
INPUT$（n, #f）	从文件*f*读*n*个字符
MKI$(*x*), MKS$(*x*), MKD$(X)	将整型、单精度型、双精度型数转换成随机文件缓冲区串变量
OCT$（n）	将*n*转换为八进制字串
SPC$（n）	打印*n*个空格
STR$（x）	将*x*转换成字串
TAB（n）	在第*n*位置上开始打印
TIME$	取系统时间
VARPTP$（v）	取变量类型、地址

(2) APPLE(苹果)Ⅱ BASIC

在APPLE(苹果)Ⅱ微型计算机上配备有浮点BASIC和整形BASIC两种。现按功能把在这两种BASIC中可用的语句、键盘命令和函数整理如下。除了注明只用作语句或只能用作键盘命令外，其余都既可用作语句又可用作命令。凡是在整形BASIC中不能使用的语句、命令和函数都标以＊号。

1．简单变量和数组元素

类　　型	名　　字
实　　型	AB, AB（3，12，7）在整型BASIC中不允许
整　　型	AB%，AB%（3，12，7）在整型BASIC中不用%号
字 符 串	AB$, AB$（3，12，7）

以上A代表字母，B代表字母或数字。名字可多于两个字符。但在浮点BASIC中只识别前两个字符，在整型BASIC中可识别全部字符。

实型数的范围为±9.99999999E＋37，整型数的范围为±32767。字符串的字符个数从0到255。

数组元素最多用到88维。而在整型BASIC中只允许一维。数组可定义的大小取决于内存的大小，数组元素下标的最小值为0。

2．运算符

算术运算符有^、＊、/、＋、－

关系和逻辑运算符有＝、＜＞（在整型BASIC中用#）、＜、＞、＜＝、＞＝、NOT、AND、OR。

如果关系表达式或逻辑表达式的值为“真”，则得数值1；若为“假”，则得数值0。关系运算符也可用于字符串的比较。

3．程序行的形式

一个语句行号后面可以跟若干个语句，它们之间用：号分开。

4．系统和实用命令、语句以及函数

LOAD	从盒式磁带上取一个程序到内存中。
LOAD AAA	从磁盘上取一个文件名为AAA的程序到内存中。
SAVE	把内存中当前程序存到盒式磁带上。
SAVE AAA	在磁盘上存入一个文件名为AAA的BASIC程序。
RENAME AAA, BBB	把磁盘上AAA的文件名改成BBB。
DELETE AAA	删除磁盘上名为AAA的文件。

NEW　　清除当前在内存中的程序。

RUN　　执行内存中当前的BASIC程序。

RUN 477　　在语句行号477上开始执行程序。

RUN AAA　　运行磁盘上文件名为AAA的程序。

*STOP　　停止程序执行，并在终端上显示执行终止的行号。

END　　停止程序执行，但并不显示停止的行号。在整型 BASIC 中END必须具有程序中最大的行号。

同时按CTRL键和C 键　　只能作为键盘命令。用来使程序停止运行或中止列程序清单。

同时按CTRL键和S 键　　恢复由CTRL - C中止的列清单。

RESET键　　无条件返回到监控程序。使用CTRL - C或0 G进入到BASIC。

CONT　　继续执行由STOP、END或CTRL - C停止的程序，整型BASIC中使用CON。

TRACE　　辅助的调试手段。列出执行时每个语句的行号，在整型BASIC中只列出一行中第一条语句的行号。

NOTRACE　　停止TRACE功能。

PEEK (X)　　函数。得到X存贮单元中的内容。

POKE X, 13　　把X存贮单元中的内容换成13。

*WAIT X, Y, Z　　其中“, Z”是可选项。使程序执行暂停，当X存贮单元中的内容与Z进行“逻辑异”运算，然后和Y进行“逻辑与”运算之后，如果结果为非零值时，恢复运行。

CALL X　　调用由存贮单元X开始的机器语言子程序。

*USR (X)　　函数。把X值传送给机器语言子程序。

HIMEM: X　　设置可为BASIC程序所用的最高内存地址X。

*LOMEM: X　　设置可为BASIC程序所用的最低内存地址X。

CATALOG　　显示磁盘上文件名的目录。

AUTO　　只有整型BASIC可用此命令。用来自动为打入的语句编语句号。

MAN　　在CTRL - C或CTRL - X之后，清除AUTO自动编号方式，改成人工编号。只有整型BASIC可用。

DSP X　　辅助的调试手段，当变量发生变化时，连同编号同时显示出来（RUN命令清除DSP的功能，因此，只在使用CON或GOTO继续执行程序时，DSP命令才有效）。只有在整型BASIC下才有此命令。

NO DSP X　　终止对X的DSP方式。只有整型BASIC才有此命令。

5．与编辑及格式有关的命令、语句和函数

LIST　　列出全部程序清单。

LIST X—Y　　列出X行到Y行的程序清单。

DEL X, Y　　从X行删到Y行。

REM XYZ　　程序中用作说明语句，执行时略去。

VTAB Y　　Y值为 1 到24，把光标移到Y行。

HTAB X　X值为1到40，把光标移到X列。

TAB（X）　函数。只在PRINT语句中使用，把光标移到X列。X值为1到40。

*POS（0）　函数。求出光标当前所在行的位置。得到的值为0到39。

*SPC（X）　函数。只能在PRINT语句中使用，在上一个打印项目和下一个打印项目之间置X个空格。

HOME　清除荧光屏上画面，并把光标移到顶端。在整型BASIC中用CALL-936来代替。

CLEAR　把所有变量置成零。在整型BASIC中用CLR来代替。

*FRE（0）　函数。求出尚可使用的内存量。

*FLASH　使计算机在荧光屏上的输出呈闪光形式。

INVERSE　使计算机在荧光屏上的输出呈白底黑字的形式。整型BASIC用POKE 50，127来代替。

NORMAL　停止闪光形式或白底黑字形式。整型BASIC中用POKE 50，255来代替。

*SPEED=X　设置字符输出的速率（X为0到255）。

先按esc键，后按A键或I键　光标向右移动一格。

先按esc键，后按B键或J键　光标向左移动一格。

先按esc键，后按C键或K键　光标向上移动一格。

先按esc键，后按D键或M键　光标向下移动一格。

先按esc键，后按E键　从光标开始清除到本行末尾。

先按esc键，后按F键　从光标开始清除整个屏幕。

按“右箭头”键　光标向右移动，荧屏上光标经过的字符送到内存。

按“左箭头”键　光标向右移动，荧屏上光标经过的字符从内存中删除。

同时按CTRL键和X键　取消当前打印的这一行。

同时按REPT键和某个字符键　使该字符重复。

6．与数组和字符串有关的语句、命令和函数

DIM A(X，Y，Z)　数组定义语句。在整型BASIC中，字符串变量和数组都用DIM语句定义。

LEN（A$）　函数。得A$中的字符。

*STR$（X）　函数。求出X的数值，并将它转换成字符串。

*VAL（A$）　函数。求出A$，把串中前面的数字字符转成数值，直到遇到第一个非数字字符为止。

*CHR$（X）　函数。得出代码为X的ASCⅡ字符。

ASC（A$）　得出A$串中第一个字符的ASCⅡ代码。

*LEFT$（A$，X）　得到A$串中最左边的X个字符。

*RIGHT$（A$，X）　得到A$串中最右边的X个字符。

*MID$（A$，X，Y）　得到A$中从第X字符开始的Y个字符的子串。

+　可用作连接字符串的操作符。

*STORE A　把数组A存放到盒式磁带上。A只能是数值数组而不能是字符数组。

*RECALL B　从盒式磁带上取回数组，数组B必须经过DIM语句正确定义过。

7．与输入输出有关的语句、命令或函数

INPUT A$　只能用作语句。执行时，在终端上显示一个？号，等待用户输入一个字符串给A$。而整型BASIC在等待输入字符串时不打印出？号。

INPUT "XYZ"；A　只能用作语句。执行时，在终端上只显示XYZ而不显示？号，等待用户输入一个实数给A。而整型BASIC在等待输入字数值时打印一个？号。

*GET A$　只能用作语句。执行时，等待用户在键盘上输入一个字符给AS，不需要按RETURN键，并且不在荧光屏上显示字符。

*DATA X，"Y"，Z　只能用作语句。用来建立一个可为READ语句使用的数据项表。

*READ A　把DATA语句中下一个数据项分配给A。

*RESTORE　再从DATA语句中的第一个数据项开始读数据。

PRINT "X＝"；X　在终端上打印字符串X＝和变量X的值。分号使打印项连在一起，逗号把打印项分开显示在三个区域内。可以用？号来代替PRINT。

PRINT CTRL — D；"磁盘操作命令"　在程序执行时执行一个磁盘操作命令。

IN＃6　从通道号6的外部设备上取入数值。键盘通道号为0，其它外部设备号从1到7。

PR＃6　把输出信息送到通道号为6的外部设备上，行打印机的通道号通常为1。

LET X＝Y　LET可以省略。把Y的值赋给X。

*DEF FN A（X）＝X＋23/X　由用户定义一个函数FNA。

8．控制转移语句和命令

GOTO 347　转移到语句号为347的语句。整型BASIC允许GOTO后面跟一个算术表达式。

IF X＜＝3 THEN STOP　如果关系表达式X＜＝3的值为"真"（非零）于是继续执行THEN后同一行上的语句。如果关系表达式X＜＝3为"假"，使得THEN后同一行中的语句全都忽略，跳到下一个行号的语句继续执行。而整型BASIC只是忽略THEN中的语句部分。

FOR X＝1 TO 20STEP 4…NEXT X　执行对应的FOR和NEXT语句之间的所有语句。NEXT语句中X可省略，而在整型BASIC中不允许省略。

GOSUB 33　转向语句标号为33的子程序。在整型BASIC中GOSUB后面可以是算术表达式。

RETURN　标志子程序结束，返回到调用本子程序的GOSUB语句后的一条语句。

POP　从RETURN地址栈中取消一个地址。

*ON X GOTO397，12，458　转移到行号表中第X个行号。

*ON X GOSUB 397，12，458　转移到行号表中第X个行号上的子程序。

*ONERR GOTO 5400　在执行此语句后，凡出现错误时，转移到以5400行号开始的出错处理子程序段，而不打印出错信息并且程序继续执行而不中止。

*RESUME　从出错处理子程序中返回到发生错误的语句重新执行。

9．和绘图、游戏控制有关的语句、命令和函数

TEXT　使荧光屏由低分辨率绘图方式或高分辨率绘图方式恢复到通常的文本方式（24行，每行40个字符）。

·低分辨率绘图

GR　使荧光屏处于低分辨率绘图方式。把荧光屏从上到下，从左到右分成40行40列个区域，全部都呈黑色。最底下还剩4行作为文本区。

COLOR = X　X的值由0到15，它们分别代表黑、靛、深蓝、紫红、深绿、灰色、中蓝、浅蓝、棕、橙、灰、粉红、绿、黄、水、白等十六种颜色。此语句为下一个PLOT语句所指定的位置设置颜色。

PLOT X，Y　使Y行、X列的位置上呈现一种由上一条COLOR语句指定的颜色。X、Y的值的范围为0到39。PLOT 0，0表示左上角第一个点的位置上出现某种颜色。

HLIN X1，X2 AT Y　在Y行上，从第X1列到第X2列为止画出一条水平线。

VLIN Y1，Y2 AT X　在X列上，从第Y1行开始到第Y2行为止，画出一垂直线。

*SCRN（X，Y）　函数。得出荧光屏上在X，Y位置上的颜色。

·*高分辨绘图。整型BASIC无此功能。

HGR　使荧光屏上绘图的第一页处于高分辨率绘图方式。荧光屏从左到右，自上到下分成280×160个区域（160行280列）。全都呈黑色。最底下剩4行作为文本区。

HGR2　使荧光屏上绘图的第二页处于高分辨率绘图方式。荧光屏上分成280×192个区域。全部呈黑色。

HCOLOR = X　语句。为下一个PLOT语句确定颜色。X的值为0到7。它们分别代表黑1、绿、紫、白1、黑2、橙、黄、白2等8种颜色。

HPLOT X，Y　在荧光屏Y行X列的位置上放一个色点。X从0到279，Y从0到159（HGR）或191（HGR 2）。

HPLOT X1，Y1 TO X2，Y2 …TO XN，YN　从X1，Y1点到X2，Y2的点上画一条线，并继续画线条，直到到达XN，YN点。

SHLOAD　从盒式带上取出一个形象表。

DRAW 3 AT X，Y　用由HCOLOR设置的颜色，从X，Y位置开始，画一个在预先取出的形状表中形状定义为# 3的形状。

XPRAW 3 AT X，Y　画一个形状表中形状定义为3的形状。只是此形状上每一点的颜色是荧光屏上该点颜色的补色。

ROT = X　使由DRAW或XDRAW建立的图形转一个角度。ROT = 0是不转；ROT = 16是顺时针方向转90°；ROT = 32是顺时针方向转180°；以此类推。

SCALE = X　为DRAW或XDRAW的图形设置比例（1到255）。

·游戏控制

PDL（X）　函数。得到一个由游戏控制器的当前值（0到255）。X值在0到3的范围内。

PEEK（X－16287）　如果得到值＞127，游戏控制器X（0到2）上的按钮正被按着。

PEEK（－16336）　扬声器发一次“咔嗒”声音。

10．其余函数

*SIN（X）、COS（X）、TAN（X）　分别求出X弧度的正弦、余弦、正切值。

*ATN（X）　求出X的反正切值。单位为弧度。

*INT（X）　得到小于或等于X值的最大整数。

RND（1）　每用一次得到一个0到0.999999999之间的实型随机数。

RND（0）　再次得到上一次的那个随机数。

RND（－3）　得到4.48217179E 08。对于每个不同的负自变量，得到一个固定的数。在此之后，RND用正自变量，将跟着一个固定的数列。

SGN（X）　如果X＜0得－1；X＝0得0；X＞0，得1。

ABS（X）　得X的绝对值。

*SQR（X）、EXP（X）、LOG（X）　分别求得X的平方根值，e（2.718289）的X次方值、X的自然对数值。

MOD（X1，X2）　只有整型BASIC有此函数，求X1被X2除后的余数。

(3)TRS-80 COLOR BASIC

本节的目的是向BASIC手册的读者提供一个TRS—80特殊功能的简要说明。

扩展的TRS—80彩色BASIC有一些关键字是此计算机特有的或者是和通常的用法很不相同的。

AUDIO　使计算机通过电视机（TV）扬声器发出声音

格式：AUDIO ON

AUDIO OFF

CIRCLE　以指定的圆心、指定的半径、指定的颜色画一个圆或圆的一部分。给出了起始和终止点，就能显示出圆的任何部分。

还能够把圆压成不大圆的形状。

格式：CIRCLE (X, Y) R, C, H, S, E

CIRCLE (128, 96), 35, 6

此处（X, Y）是圆心的位置，R是半径，C指定几种可用的颜色（0—8）中的一种，H指出高/宽之比（1表示是真正的圆，H＞1表示高大于宽，H＜1表示宽大于高）。

S是0—1间的一个数，它指出圆弧的起始位置，0代表三点钟，0.25代表六点钟（半圆），等等。E也是0～1间的数，它指出圆弧终止的位置，画出的部分是一圆弧，从起始点到终止点沿顺时针方向画出弧线。

CLOADM　从磁带上装入一个机器语言程序。

格式：CLOADM "NAME", A

此处NAME是程序名，A（可选项）代表加到起始地址的数，它使程序送到你所希望的内存位置。

CSAVEM　将一个机器语言程序保存到磁带上。

格式：CSAVEM X, 4 E, 6 F, 5 F

COLOR　对显示选择"前景"和"背景"的颜色。颜色的代码从0到8。根据PMODE的设定，在一次使用中可以用两种颜色或四种颜色。SCREEN语句选择用哪两种或四种颜色。

如PMODE设定0，2和4，可以用两种颜色。用SCREEN语句选择颜色组（Color set）0时给出黑色（0）和绿色（1），而选颜色组1时，可用黑色和淡黄色（5）。PMODE设定1和3时，可以从四种颜色中选择。颜色组0有绿、黄（2），蓝（3）和红(4)，而颜色组1包括淡黄、青蓝（6）、深红（7），和橙色（8）。

在文本方式（也是由SCREEN选择的），可用的颜色也是黑色和绿色（颜色组0）或红和橙（颜色组1）。

格式：COLOR 5, 7

当画图时，在深红的背景上显示浅黄的线。如果不用COLOR语句，在当前的颜色组的最高号码的颜色是前景颜色，最低号码的颜色是背景的颜色。

DLOADM　用来从另一计算机装入一个机器语言程序。

格式：DLOADM "NAME", B

此处B是0或1指定波特率300或1200。"波特率"是传送数据的速度（位/秒）。

DRAW　画出一条或多条直线（甚至一整个图形），在DRAW之后跟着以引号括起来的指令或者以字符串变量代表的指令。

格式：DRAW "字符串"。

DRAW "S2；BM128, 96；E50；L100；D100；R100；H50"

此处"字符串"是由下列指令构成的组合。

B　空格——不画图

M　移向指定的位置（隐含128, 96或最后一个DRAW语句的终点）。M指令后面跟一个加号（+）或减号（-），意思是：把这数加到当前位置以设定下一条线的起始点

U　向上画出指定的数（对V, D, R, L, E, F, G和H, 隐含值为1）

D　向下

R　向右

L　向左

E　45°（右上）

F　135°（右下）

G　225°

H　315°

X　执行另一个指令串并返回。例如：XAS

C　颜色（隐含为前景颜色）

A　旋转角

0 = 不旋转（隐含）

1 = 顺时针90°

2 = 顺时针180°

3 = 顺时针270°

S　比例因子，这里因子为1意思是$\frac{1}{4}$比例尺，2代表$\frac{1}{4}$或$\frac{1}{2}$比例尺，等等。（隐含为$\frac{4}{4}$或全比例尺）

N　不修正，也就是从原先的起始点画下一条线。

EXEC　将程序控制转给一个机器语言子程序（在指定的地址），见USR。

格式：EXEC 24623

JOYSTK　是一个函数，它带回一个0到63之间的值，R指出操纵杆的位置。

格式：JOYSTK(n)

此处n是一个0到3之间的数，它决定受检查的是哪一操纵杆。

0 = 右操纵杆的垂直方向

1 = 右操纵杆的水平方向

2 = 左操纵杆的垂直方向

3 = 左操纵杆的水平方向

JOYSTK（0）必须是用在任何程序中第一个JOYSTK函数。

LINE　在指定的点之间画一条线，一个匣子，或一个填满（实心）的匣子。指定的点是线的二个终点或匣子的对角。

格式：LINE (X1 ,Y1) - (X2 ,Y2), A, B

LINE (20, 10) - (150,100), PSET, BF

此处（X1 ,Y1 ）是直线的起始点，（X2 ,Y2 ）是直线的终点或匣子的对角点。

A可以是PSET，它使直线（或匣子）选“前景”的颜色，也可以用PRESET，它选择“背景”颜色（也就是使图形与背景颜色相同，从而使图形“抹掉”）。B是可选的，它可以是“B”，表示画一空匣子，也可以是“BF”，表示为实心的匣子。假如省略起始点，则从先前的终止点开始画（如果没有用过上一个LINE语句则从显示屏的中心点开始画）。但连接符“-”不能省略。

MOTOR　打开或关闭盒式录音机。

格式：MOTOR ON

MOTOR OFF

PAINT　用选定的颜色涂满从一个指定的点到指定颜色的分界点间的区域。

格式：PAINT (X,Y) , C, B

此处（X, Y）是要涂色的区域中的一个点，C选择一种可能的颜色（见COLOR），B指出已有的分界线的颜色。

PCLEAR　保留用于作图的内存总数。作图内存分为8页，每页有1536个存贮单元，如果程序中无PCLEAR语句则计算机自动保留内存中四页为作图用。

格式：PCLEAR 2

只保留两页，以使节余的部分可以作为大程序利用。

PCLS　清除作图屏幕显示，与在文本方式下用CLS清除屏幕的用法类似。

格式：PCLS n

此处n是可用作背景的一个颜色代码。如n省略，则作为当前的背景颜色。

PCOPY　将一个作图页“复抄”到另一页。

格式：PCOPY 3 TO 5

PLAY　用一个指令串来产生音乐，这些指令指定音调、拍子、音量、休止符等等。

格式：PLAY “音乐指令”

PLAY “T2；L4；O3；A;L8；A; A; B－；A; G; F; L2，3A; P4”

此处“音乐指令”是一个字符串，它可以包括以下一些指令符号。

音调：A到G间一个字母，或1到12的一个数，用－表示降半音，+或#表示升半音

O　指定五个音阶（1—5）之一。

L　表示音的长（L1 = 全音符，L2 = 半音符，L4 = 四分之一音符，L4· = 带点的四分之一音符，等等）。

T　设定速度（拍子）（1～255）

V　设定音量（1—31）

P　决定休止符或停唱的长度（1—255）

X　执行另一个指令串（见DRAW）

PMODE　设定画图屏幕的方案，即可用多少种颜色，以及先使用哪一种画图“页”。

格式：PMODE R, P

此处R是由0到4之间的一个数，它产生下面的方案：

0 给出128×96个格子，用两种颜色

1 给出128×96个格子，用四种颜色

2 给出128×192个格子，用两种颜色

3 给出128×192个格子，用四种颜色

4 给出256×192个格子，用两种颜色

P的值从1到在PCLEAR中指定的值，它指定从哪一个画图页开始。

PPOINT　检查画图屏幕中指定的点的颜色。如果该点是“开”状况（“ON”）则PPOINT带回颜色代码。

格式：PPOINT (X, Y)

PRESET　将一个点设置为当前的背景颜色。

格式：PRESET (X, Y)

PSET　将一个点设置为当前的前景颜色。

格式：PSET (X, Y)

SCREEN　使计算机处于文本方式或作图方式以及指出用哪一种颜色组（见COLOR）

格式：SCREEN M, C

此处M为0（对文本方式）或1（作图方式），C指定COLOR（颜色）组0或COLOR（颜色）组1。

SOUND　在给定的时间长度内按指定的音调发音。

格式：SOUND P, T

此处P是一个数，它决定音调（1为低音，225为最高音，89接近于钢琴上的C中音），T设定持续时间（10 = 0.60秒，25 = 1.50秒，等等）

TIMER　是一个函数，它按“瞬间”（1/60秒）来测量时间。TIMER从电源接通开始工作并在每次到达65535时重新置零。TIMER能够测量出某一事件的时间，先使：

```
TIMER = 0
```

（可以给TIMER赋以0—65535之间的任何一个数），可以用下面语句“读”出TIMER的值：

```
PRINT TIMER
```

（4）ACORN ATOM BASIC

本节的目的是向BASIC手册的读者提供一个ACORN ATOM BASIC 特殊功能的简要说明。

英国Acorn ATOM系统所支持的BASIC是很不标准化的，它和本书所介绍的有很多不同，它提供的特征在其它的机器上是不适用的。由于Acorn的迅速推广流行，有可能在国际上的计算机的应用中可以看到有许多程序是为Acorn写的。为了帮助读者了解它们，我们提供了这个Acorn BASIC的最主要的、独特的性能的简要介绍。

Acorn BASIC的最主要的特殊点是它的变量的命名。整型变量可以用字母A—Z。数组必须命名为AA—ZZ。因此DIM AA（10）为AA保留11个存贮单元。

DIM A（10）则完全是又一事。它把BASIC程序中用到的最后一个字节的地址存入A，并使此地址增加11（程序此时长了11个字节）。

浮点变量（实型变量）的形式是在变量或数组名的前面放一个%符号。%A, %B, …… %Z和%AA, %BB, ……%ZZ以及%@和%@@都是有效的变量名和数组名。字符串名的形式是在数值变量前放一个$，即$A, $B, …$Z和$AA, $BB, …$ZZ。在不致产生含混（二义性）的地方，可以不必用$。

下面的语句在最前面加了一个字母F，R表示它们是用于实型的关键字。它们的使用和在这本BASIC手册中有关部分中介绍的相应的关键字是类似的（即使不完全相同的话）。用这些关键字时变量必须用前缀%说明为实型（如%A）。

关键字	由此变化来的	含　义
FDIM	DIM	数组名必须为%AA到%ZZ形式
FGET	GET	只能从一个文件读数值
FIF	IF	AND和OR不允许用FIF
FINPUT	INPUT	不允许输入字符串
FPRINT	PPINT	不允许出现字符串表示式
FPUT	PUT	只能向一文件输出数值
FUNTIL	UNTIL	AND和OR不允许用FUNTIL

更详细的知识可参考：DIM, GET, IF, INPUT, PRINT, PUT, UNTIL, %

用！操作符来指出需要寻找的或存贮的一个值的地址。！A=123456将123456最后两个有效位的十六进数形式存到A地址中，然后接着存到A+1，A+2地址中，直到整个数值存贮完为止。用A!6来代替！A则从A后面6个字节开始存贮。

Acorn用的其它操作符和关键字如下：

'（撇号）在PRINT语句中产生一个新行。如果不用'号则下一个PRINT将在同一行上继续打印

@指定数值输出的字段宽度

? 类似于！，但一次只存贮和调用一个字节

十六进制常数的前缀

& 在PRINT语句中的十六进制的前缀，也作为逻辑AND运算符（即按位相加两个数，如 6 &12 = 4，因为0110 AND 1100 = 0100）

 （“黑底白字”的反斜杠）是逻辑OR运算符

: 逻辑“异或”运算符

% 给出除法的余数。如13 % 5 = 3。%也用来将一个表达式转换为整数形式。

BGET 从一个随机文件中取一个字节

BPUT 向一个随机文件存一个字节

EXT 给出一个文件的长度（字节数）

FIN 为准备输入或修改而初始化一个随机文件，并得到一个引用此文件的数值。

FLT 将整型表达式转换成实型

FOUT 为准备输出而初始化一个随机文件，并且得到一个引用此文件的数值。

LINK 调用一个机器语言子程序

OLD 在打入NEW之后（但在使内存出现任何变化之前）重新装入一个程序

PTR 允许随机文件的指针控制

SGET 从一个随机文件读一个字符串

SHUT 在使用完后关闭一个文件

SPUT 向一随机文件写一个字符串

使用ATOM的所有用户都必须细心地阅读它的使用手册，使自己成为使用上面那些规定的能手。其它的关键字是和其它的BASIC类似的，它们已在本书的其它页中作了介绍。

关于“磁盘BASIC”的简要介绍

许多计算机具有在硬磁盘上或软磁盘上存贮程序和数据的能力。磁盘是作为“大容量外部存贮”的介质而使用的，因为它能存贮的信息总量比计算机内能存贮的要大好多倍。

关于DOS

磁盘操作系统（Disk Operating System，简称DOS）是一个“主控”软件程序。DOS是操作系统，它主要是与计算机和其外部设备的控制与操作有关的。它使CPU（计算机的心脏），BASIC解释程序（或编译程序），和存贮器，以及输入/输出设备协同工作如同一个整体。

它也能告诉CPU当前的程序或数据存贮在什么地方以及什么地方是空的可以存贮其它的程序或数据。有些系统把DOS看作“执行程序（Executive）”或“监控程序（Monitor）”。

由于DOS的文件处理功能，我们往往对它是有所了解的。它允许我们能打印出存贮在磁盘上的程序和数据文件的目录（一般用DIR命令）。DOS命令常常会和磁盘BASIC的语句与命令弄混，因为DOS或BASIC可能有一些相同的命令。

我们可以用DOS命令删除一个文件（用KILL或DELETE）；重新命名一个文件（用RENAME）；将一个磁盘上的程序“复抄”到另一个磁盘上（用COPY）；它也允许我们装入BASIC解释程序或编译程序（它也是一个软件程序）。“实用程序（Utility）”常常是和DOS一起使用来完成某些功能的，如对一个数据文件排序（SORT），对程序行重新排列行号（RENUMBER）和帮助你调试一个程序（DEBUG）等等。

磁盘BASIC不是完全相同的

磁盘BASIC是通常的BASIC加上一些附加的功能，例如向磁盘存贮和从磁盘检索程序。

为了能控制大的数据文件，“简单的、老的”BASIC必须经历某些小的变化（主要是增加）。磁盘BASIC包括“通常的”BASIC语言的语句、函数、命令和操作符，再加上与存贮在磁盘上文件有关的语句和命令。

所增加的磁盘BASIC特性是不完全相同的，在此不可能对所有的磁盘BASIC的功能和其使用作详细的论述。但是，为了使你能了解你的程序中磁盘BASIC的性质，我们写了这个简要介绍，它包括了最通常使用的语句。它将帮助你理解什么时候要用磁盘BASIC以及用它能干什么。如果你的计算机BASIC与我们介绍的不同，你可以自己改写程序以适用于你的计算机。

通常在我们所用的大程序中往往要用到磁盘驱动器，这是因为对磁盘的存取要比对磁带的读写快得多和比较方便。虽然在磁盘上能完成的几乎任何事情都能在磁带上做到，但后者却慢得多。假如你企图将一个为磁盘写的程序转换到适用于只有磁带存贮器

的计算机，它要求你做的决不仅仅是用新语句代替原有语句。你要充分注视程序流程使慢速的磁带造成的影响减少到最低限度，而且常常需要人们的干予。

在某些计算机的磁盘系统中，将BASIC调入内存是自动的。而其它一些计算机则要求打入象RUN BASIC或只打入BASIC这样的命令使系统从DOS状态进入BASIC解释 程序/编译程序工作状态。

一旦进入BASIC，要返回DOS状态要求打入一个命令（不同的计算机规定的命令是不同的）。典型的如：BYE, DOS, EXEC, MONITOR, SYSTEM或CMD“S”。有 时我们只需按“reset”按钮就能返回DOS。

打开和关闭文件

磁盘文件包含若干“记录”，如果是数据文件，记录中包含一些数字或字符，它是 程序运行时所需要用到的数据。“文件”是“记录”的集合，我们给文件定一个名字。文件是一个实体，例如一个BASIC程序存放在磁盘上，它就是一个文件。通常我们一次LOAD（装入）或SAVE（存贮）一个完整的程序文件。而数据文件则可以一次 READ（读）或修改一个单独的记录。

要从文件读写（称存取）信息，必须首先打开此文件。磁盘文件可以顺序存取，也就是说，如同在磁带上那样按次序逐个存取记录。也可以随机存取，如同留声机那样，可把唱针放在任何地方，而不需先经过前一个螺旋槽。

在程序中必须用OPEN语句指定存取方式、被打开的文件名字、以及与文件相关 联的引用号（即文件号），（引用号在程序的READ和PRINT语句中要用到）。

各种版本的磁盘BASIC不是完全一致的，但下面几个例子可以表示出一种 典型的OPEN语句：

```
10  OPEN “I”, 3, “MYDATA”
```

打开一个名为“MYDATA”的文件，用顺序输入（INPUT，缩写“I”）。由READ#3 语句可以读此文件。它只能用于“读入”，不能向它写。

```
20  OPEN NEW “SCORES” AS 1
```

打开一个新文件，名为SCORES，顺序输出（OUTPUT）。用PRINT#1 可以将信息 从计算机送到文件去。

```
30  OPEN “DATAFIL” FOR INPUT AS FILE 2
```

打开一个顺序输入的文件DATA. FIL，它作为文件 2。在程序中可以用READ#2 语句 读它。

后两个OPEN语句之一也可以用来打开一个随机文件。上面10语句如果改为随机 存取应将“I”改为“R”。在某些计算机中用FILE代替OPEN。

当为了使用文件而打开时，由于程序错误、电源切断、或其它意外的事故，会使文件中的数据出现意外的改变，文件受到“伤害”。因此，当一个文件在程序中不再需要使用时，应当用CLOSE 1或类似的语句加以关闭。在一个程序执行过程中，可以OPEN和CLOSE文件任意多次。

顺序文件和随机文件

READ#，PRINT#和INPUT#语句常用于顺序文件，但必须指出与此文件相关联的设备号。在某些磁盘BASIC中WRITE#和PRINT#都可以用。有几种计算机用WRITE#来将带引号的字符串数据存入文件，而其它一些计算机则用WRITE# 在每一个DATA行之前插入行号。

随机文件的记录必须事先规定其格式。FIELD语句定义出你所希望的记录格式。可以是如下形式：

10 FIELD# 1，20 AS N$，22 AS A$，15 AS C$，2 AS S$，5 AS Z$ FIELD语句指定了文件#1 的每一个记录包含了上面说明的五个项（我们在下面将文件内容印出来，在它的上一行印出顺序号，以便使大家形象化看到每一项的长度）。

```
123456789012345678901234567890123456789012
JOHN A. EDMUND        3206 BEAL ROAD

                  34567890123456789012 34
                  HARRISON         CA93888
```

FIELD语句为“名字”（JOHN A. EDMUND）保留20个字符（包括尾随空格），后面22个字符是“街道地址”，15个字符是“城市”，2个字符是“州”，5个字符是ZIP码。现在BASIC程序就知道怎样“理解”这个随机文件中的每一记录了。

存取的数据在缓冲区中

在向磁盘存取时，信息是放在计算机中一个称为缓冲区的临时存贮区中的。许多磁盘BASIC在送信息到磁盘之前先用LSET和RSET将数据放在缓冲区中。LSET是使数据在字符串变量中“左对齐”，而RSET使数据“右对齐”。例如：

10 LSET N$= “JOHN A. EDMUND”

将名字放到记录中，按上页所印出的位置。

10 RSET N$= “JOHN A. EDMUND”

使名字按以下位置出现在记录中。

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
          J O H N   A .   E D M U N D
```

随机字符串

一些磁盘BASIC要求所有随机文件的数据都必须是字符串形式（不能是数值）。为了将数值转换为字符串，以及将字符串转换为数值，这些BASIC中包含转换函数。

函数MKI $，MKS$，和MKD$ 将整数、单精度数和双精度数转换为字符串。CVI，CVS和CVD将字符串转换回数值。其它一些磁盘BASIC用CVT%$和CVTF$将整数和实数转换成字符串，用CVT$%和CVT$F把它们转换回去。它们是容易使用的，但是也是很容易用错的。

读和写

典型的随机输入和输出的语句是GET#1 （对输入）和PUT#1 （对输出）。有几种计算机用INPUT：1或READ：1代替GET#1 以及用 PRINT：1 或 WRITE：1代替PUT#1 。

先找出文件

在某些计算机上，在向一个顺序输出文件“写”之前必须用“SCRATCH”语句，如：

```
10  SCRATCH #4
```

恢复

某些计算机允许在读顺序文件后用RESTORE语句使之回到原来位置以便下次再从头读这一文件。如果RESTORE不起作用，把它关闭（CLOSE），然后再打开（OPEN）此文件，这时指针会自动置于数据文件的开头处，也就是实现了RESTORE功能。

EOF

当从磁盘或其它外部大容量存贮设备输入数据时，应当探测什么时候已将数据全部输入完毕并且遇到“文件结束记录”。在IF—THEN语句中可以用EOF函数（end-of-file）来完成这一工作。例如：

```
100 IF  EOF #1  THEN 520
```

可读作：“如果数据已经读完，转520行”。

文件长度

有些磁盘BASIC提供了LOF函数以决定文件的长度（Length-Of-File），以及LOC函数以指出当前文件指针的位置（即指向正在存取的记录）。用SET语句设定“指针”（如同置放留声机的唱针）到你想要打印或读的那个记录的位置。

下一个记录的类型

用TYPE（或TYP）来探测一个顺序文件中的下一个记录中的数据的种类。例如，在North star BASIC中，如果TYP的值为0，表示遇到文件结束记录。如果是1，下一个记录包含字符串数据，2表示包含数值数据。其它计算机用TYPE时，它的值所代表的含义有所不同。

例：

```
190  IF  TYP (1) = 2  THEN 220
200  READ #1, AS
210  GOTO 230
220  READ#1, N
```

文件名

最后，关于文件名说几句。对文件名的命名方法是由计算机的DOS决定的而不是由磁盘BASIC决定的。由于OPEN语句和其它语句需要指定文件名，因此我们在这里也作些说明。

一般地，文件名由以下四项组成：

1．文件存贮所在的设备的代号。

2．文件的名字

3．扩展名

4．口令（password）（如果有的话）

例如：

LOAD 1：COINS/DAT，PASS

磁盘驱动器#1装入一个文件，它的名字为COINS，它有口令PASS。/DAT是扩展名（表示文件类别），指出它是数据文件（DATA file）。

“方言”的阻碍

由于BASIC的扩展功能没有统一的标准规定，因此各种磁盘BASIC之间有许多的不同。我们没有办法使磁盘语句有一致的格式，而进一步深入讨论必然会更多地涉及磁盘BASIC“方言”，每一种版本的“方言”间的差别是很大的，有的互相是难以理解的。因此只好到此为止。

如果想进一步了解有关这些的一般原理，可以看有关磁盘BASIC和磁盘操作系统的计算机软件的书刊。

索 引 和 记 录 卡 片

以下几页既是“BASIC 手册”的索引，又是你所用计算机的记录卡片。你可运行每一个“测试程序”，然后把结果记录在此。查这几页就可以使你很快地知道某一个特定的“字”在你的机器上能否被接受。

语句 / 字符	页码	测试结果 通过	测试结果 失败
A			
A	1, 4		
ABS	1		
AC.	2		
ACS	2		
ACSD	2		
ACSG	2		
AND	4		
APPEND	8		
ARCOS	2		
ARCSIN	11		
ARCTAN	15		
ASC	9		
ASCII	9		
ASN	11		
ASND	11		
ASNG	11		
AT	13		
ATAN	15		
ATN	15		
ATND	15		
ATNG	15		
AUDIO	318		
AUTO	17		
B			
BASE	19		
BGET			
BOLD			
BPUT			
BREAK	20		
BYE	21		
C			
C	41		
CALL	22		
CDBL	22		
CH	23		
CHAIN	24		
CHANGE	26		
CHAR	28		
CHAR$	28		
CHR乂	28		
CHR$	28		
CHR	28		
CINT	29		
CIRCLE	318		
CLEAR	30		
CLG	32		
CLK	33		
CLK$	33		
CLOAD	34		
CLOADM	318		
CLOG	32		
CLOSE	326		
CLR	30		
CLRDOT	35		
CLS	36		
CMD	326		

语句/字符	页码	测试结果 通过	失败
NEXT	165	____	____
NOFLOW	167	____	____
NORMAL	168	____	____
NOT	168	____	____
NOTE			
NOTRACE	170	____	____
NUM	170	____	____
NUM$	171	____	____

O

OLD	324	____	____
ON ERR GOTO	173	____	____
ON ERROR GOTO	173	____	____
ON- G.	175	____	____
ON- GOSUB	174	____	____
ON- GOS.	174	____	____
ON- GOT	175	____	____
ON- GOTO	175	____	____
OPEN	326	____	____
OPTION	178	____	____
OR	179	____	____
OUT	182	____	____

P

P.	193	____	____
P. A.	196	____	____
PAINT	320	____	____
PAUSE	183	____	____
PCLEAR	320	____	____
PCLS	320	____	____
PCOPY	320	____	____
PDL	184	____	____
PEEK	184	____	____
PI	185	____	____
PIN	186	____	____
PLAY	321	____	____
PLOT	187	____	____
PMODE	321	____	____
POINT	187	____	____
POKE	188	____	____
POP	189	____	____
POS	190	____	____
PPOINT	321	____	____
PRECISION	192	____	____
PRESET	321	____	____
PRI	193	____	____
PRINT	193	____	____
PRINT AT	196	____	____
PRINT USING	197	____	____
PRINT@	196	____	____
PSET	321	____	____
PTR	324	____	____
PUT #	327	____	____

R

R.	211	____	____
RAD	203	____	____
RADIAN	203	____	____
RAN	204	____	____
RANDOM	204	____	____
RANDOMIZE	204	____	____
REA	205	____	____
REA.	205	____	____
READ	205	____	____
RECALL	207	____	____
REM	208	____	____
REMARK	208	____	____
REN	208	____	____
RENAME	325	____	____
RENUM	208	____	____
RENUMBER	208	____	____
REPEAT$	211	____	____

语句/字符	页码	测试结果 通过	测试结果 失败
THE	247		
THEN	247		
TI	247		
TIM	247		
TIME	247		
TIMER	322		
TIME$	249		
TI$	249		
TLOAD	250		
TNH	244		
TOP	250		
TRACE	251		
TRACE OFF	252		
TRACE ON	253		
TROFF	253		
TRON	254		
TSAVE	255		
TYP	328		
TYPE	328		
U			
UNTIL	256		
USER	257		
USR	257		
V			
VAL	258		
VARPTR	259		
VIEWPORT			
VLIN- AT	260		
VTAB	261		
W			
WAIT	263		
WEAVE	8		
WHILE	265		
WRITE	327		
X			
XDRAW	267		
XOR	267		
XRA	267		
"	269		
,	271		
.	273		
;	274		
:	275		
( )	276		
[ ]	276		
@	277		
#	278		
$	281		
!	283		
%	285		
?	286		
\	287		
**	288		
+	289		
-	291		
/	291		
*	292		
=	293		
↑	295		
^	295		
<	297		
>	298		
<>	299		
<=	300		
>=	302		
'	303		
&	304		
¤	305		