算法语言 ALGOL 60 导引

周龙骧（科学院数学所计算站）

"ALGOL 60 导引" 的编写希望对学习 ALGOL 60 正式报告（即 P. Naur 执笔的 1960 年报告和 1962 年 M. Woodger 执笔的报告附件 [1]）的人能有所帮助。

本文编写的另一个目的是希望对国产DJS 21（即 121）计算机的自动化语言（由北京大学和南丰机械厂编制[2]）的使用提供一个简要的说明。

程序自动化

所谓程序自动化就是用计算机代替人来编程序。

由于计算机硬件系统的飞速发展，计算速度越来越快，存贮容量越来越大，处理信息的速度和能力有了极大的提高，相形之下通过人来编程序的工作是不适应了，即使是中速的计算机例如 121 机往往程序员劳动一周甚至一个月所编出的程序，上机几分钟就算完了，提高效率的矛盾愈来愈集中在编程序上面。因此提出了利用计算机来代替人编程序的办法。

我们知道每台计算机都有一套指令系统，由机器进行操作。121 机有 59 条指令，例如：

N→I D 就是把内存地址为 D的内容送到第 I 寄存器。

I→N D 就是把第 I 寄存器的内容送到内存地址为 D的单元中去。

\+ D 就是把内存地址为 D的内容同第 I 寄存器的内容相加，结果放在第 I 寄存器中。

× D 就是把内存地址为 D的内容同第 I 寄存器的内容相乘，结果放在第 I 寄存器中。

.....

如果我们要计算一个式子：

$$A \times B + C$$

T. O. Weighted pseudoinverses with singular weights, SIAM J. Appl. Math. 21, 480—2 (1971)

77. Tewarson, R. P. An iterative method for computing generalized inverses, International J. of computer Mathematics, 3, 65—74 (1971)

78. Tewarson, R. P. On two direct methods for computing generalized inverses, Computing 7, 236—9 (1971)

79. Wall, J. R. and Plemmons, R. J. Spectral inverses of stochastic matrices, SIAM J. Appl. Math. 22, 22—6 (1972)

80. Ward, J. F, Boullion, T. L. and Lewis, T. O. Weak spectral inverses, SIAM J. Appl. Math. 22, 514—8 (1972)

81. Berman, A. and Plemmons, R. J. Monotonicity and the generalized inverse. SIAM J. Appl. Math. 22, 155—161 (1972)

82. Plemmons, R. J. and Cline, R. E. The generalized inverse of a nonnegative matrix, Proc. AMS 31, 46—50 (1972)

83. Stallings. W. T. and Boullion, T. L. Computation of pseudoinverse matrices using residue arithmetic. SIAM Rev. 14, 152—163 (1972)

84. Voith, P. P., Vogt, W. G. and Mickle, M. H., On the computation of the generalized inverse by classical minimization. Computing 9, 175—187 (1972)

85. P. Å Wedin, Perturbation theory of pseudo-inverse. BIT. 13, 217—232 (1973)

86. Rao. C. R. and Mitra, S. K. Generalized inverse of matrices and its applications. 1971.

87. Boullion, T. and Odell, P., Generalized inverse matrices. 1971

算法语言ALGOL 60导引

周龙骧（科学院数学所计算站）

"ALGOL 60导引"的编写希望对学习ALGOL 60正式报告（即P. Naur执笔的1960年报告和1962年M. Woodger执笔的报告附件[1]）的人能有所帮助。

本文编写的另一个目的是希望对国产DJS 21（即121）计算机的自动化语言（由北京大学和南丰机械厂编制[2]）的使用提供一个简要的说明。

程序自动化

所谓程序自动化就是用计算机代替人来编程序。

由于计算机硬件系统的飞速发展，计算速度越来越快，存贮容量越来越大，处理信息的速度和能力有了极大的提高，相形之下通过人来编程序的工作是不适应了，即使是中速的计算机例如121机往往程序员劳动一周甚至一个月所编出的程序，上机几分钟就算完了，提高效率的矛盾愈来愈集中在编程序上面。因此提出了利用计算机来代替人编程序的办法。

我们知道每台计算机都有一套指令系统，由机器进行操作。121机有59条指令，例如：

N→I D 就是把内存地址为D的内容送到第I寄存器。

I→N D 就是把第I寄存器的内容送到内存地址为D的单元中去。

\+ D 就是把内存地址为D的内容同第I寄存器的内容相加，结果放在第I寄存器中。

× D 就是把内存地址为D的内容同第I寄存器的内容相乘，结果放在第I寄存器中。

……

如果我们要计算一个式子：

$$A\times B+C$$

T. O. Weighted pseudoinverses with singular weights, SIAM J. Appl. Math. 21, 480—2 (1971)

77. Tewarson, R. P. An iterative method for computing generalized inverses, International J. of computer Mathematics, 3, 65—74 (1971)

78. Tewarson, R. P. On two direct methods for computing generalized inverses, Computing 7, 236—9 (1971)

79. Wall, J. R. and Plemmons, R. J. Spectral inverses of stochastic matrices, SIAM J. Appl. Math. 22, 22—6 (1972)

80. Ward, J. F, Boullion, T. L. and Lewis, T. O. Weak spectral inverses, SIAM J. Appl. Math. 22, 514—8 (1972)

81. Berman, A. and Plemmons, R. J. Monotonicity and the generalized inverse. SIAM J. Appl. Math. 22, 155—161 (1972)

82. Plemmons, R. J. and Cline, R. E. The generalized inverse of a nonnegative matrix, Proc. AMS 31, 46—50 (1972)

83. Stallings, W. T. and Boullion, T. L. Computation of pseudoinverse matrices using residue arithmetic. SIAM Rev. 14, 152—163 (1972)

84. Voith, P. P., Vogt, W. G. and Mickle, M. H., On the computation of the generalized inverse by classical minimization. Computing 9, 175—187 (1972)

85. P. Å Wedin, Perturbation theory of pseudo-inverse. BIT. 13, 217—232 (1973)

86. Rao, C. R. and Mitra, S. K. Generalized inverse of matrices and its applications. 1971.

87. Boullion, T. and Odell, P., Generalized inverse matrices. 1971

我们可以这样编程序：

```
N→I  A
×    B
+    C
I→N  D
```

机器顺序执行这个程序即算出了上述表达式。

但是对于上述的指令符号机器并不认得，因而把每条指令编号，内存址址也进行编号：

```
002   0178
00$\bar{2}$   1354
008   1042
004   0203
```

其中 $\bar{2}$ 是12的简写将上述符号编的程序及地址都代成号码，叫做代真。再用穿孔机穿成纸带，用光电输入机输入机器进行计算。

我们可以发现这样编程序是很麻烦的、首先它缺乏通用性、在121机上算就得用121机的指令编程序，在109乙机上算就得用109乙机的指令编程序、而各种计算机的指令往往不同。第二是不直观、例如上面举的A×B+C是很直观的，但编成程序后就很不直观，特别是代真成代码后就更不直观了，一般人是很不习惯，不易接受的。第三是有错很难查，查出来要改也很麻烦。

根据这些原因我们即可看出为什么编程序是薄弱环节了，它使得计算机由于等待编程序而降低使用效率，如果要提高机器效率就需要一支庞大的编程序的队伍，这也是很不经济的。

为此，考虑如何利用计算机来编程序以提高效率。

计算机只接受机器语言即机器指令，通常的数学语言它是不“认得”的。我们来分析一下这两种语言的优缺点：

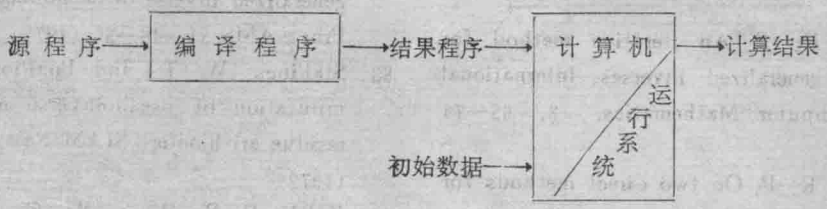
机器语言（机器指令）		数学语言
优点：1.机器能接受。	←	缺点：1.机器不能接受。
2.能描述数值计算的细节（一步一步的分解动作）。	←	2.不能描述数值计算的细节（例如 $\lim_{n\to\infty} 1/n^2$（求序列 $1/n^2$ 当 $n\to\infty$ 时的极限），x_0 是 $x^2-5x+6=0$ 的根）。
缺点：3.一般人不习惯，不易接受。	→	优点：3.一般人习惯，易接受。
4.难读。	→	4.易读。

算法语言

考虑创造一种兼有机器语言和数学语言的优点而无其缺点的语言。算法语言（ALGORITHMIC LANGUAGE）就是其中的一种，最通用的是ALGOL 60和FORTRAN。目前国内采用较多的是ALGOL 60。

用算法语言编的程序叫源程序(SOURCE PROGRAM)，机器还不能认识它，必须给机器配一套编译程序(COMPILER)。编译程序是用机器指令编制的，事先把编译程序输入到机器中。再输入源程序，机器就通过编译程序对源程序进行加工，编出结果程序(OBJECT PROGRAM)也叫目标程序或目的程序，结果程序是用机器语言编制的，然后机器按照结果程序进行操作，最后将计算结果输出，

这个过程的框图是：



由于这种“翻译”工作是由编译程序进行的，翻出来的目的程序和手编程序不可能完全一样，它自己还不能单独运行，需要一种辅助程序，叫“运行系统”的，来帮助它运行。

编译程序相对于计算机（硬件，Hard ware）来说就叫做软件（Soft ware）。当然编识程序仅仅是软件的一部分，其他如语言，操作系统等也都是软件。

使用了程序自动化系统之后使编程序的效率大为提高，某厂算一个凸轮，采用手编程序时需编一个月，但上机计算只需几分钟就算完了，后来采用算法语言编程序，只用了一天就编出源程序，上机计算连同编译结果程序在内也只用了几分钟时间就完成了，此外，一般算题同志只需看看说明书，即会编源程序，而不必通过专门的程序员编程序，这就大大便利了计算机的推广使用并提高了机器的效率。

由于使用自动化语言效率高，方便，故在计算机使用普遍的地方，很少使用手编程序。另外，算法语言是面向问题的，它不依赖于具体的计算机，例如用

ALGOL 60 语言编的源程序，对使用ALGOL 60 系统的机器都能用（各机器的语言系统只有很少的差别，只需略加修改即可），国外杂志上经常发表用ALGOL 语言写的各种问题的算法。

算法语言提出至今已近20年了，最初是 H. RUTISHAUSER 在1951年提出的，但因为不太合用未引起重视，也未在机器上实现过，以后美国IBM公司 1957 年发表了程序设计语言 FORTRAN (FORMULA TRANSLATION)，西欧在 1958 年提出算法语言ALGOL 58，美国和西欧共同在 1960 年提出了算法语言 ALGOL 60，目前使用比较广泛的数值计算的自动化语言是 FORTRAN 和 ALGOL 60，这两种语言都是科学计算的语言，它们使用比较少的数据而完成复杂的计算。此外比较广泛使用的语言还有COBOL(COMMON BUSINESS ORIENTED LANGUAGE)，它是1960年美国发表的，这种语言主要用于处理数据，它处理大量的数据而进行的计算比较简单，新发展的语言有 PL/1，它较近于 FORTRAN，兼蓄了已有的FORTRAN 和 COBOL 的科学计算和数据处理的功能。PL/1 是 1964年设计的，西欧发展了ALGOL 68，它较近于ALGOL 60，这是一种可扩充的语言，此外还有各种各样的面向过程的专用语言。

ALGOL 60

第 1 节　赋值语句(ASSIGNMENT STATEMENT)

假设我们要计算

$$A=B+C$$

在 ALGOL 60 中记为

$$A:=B+C$$

读为 A 赋值 B+C，其含意是：

1. 计算表达式B+C的值。

2. 把计算出来的值赋给变量A。

其中变量B和变量C的值应当是已经定义好的，这里“变量”这个词我们按通常的意义去理解。

应当注意赋值号：=是一个有方向性的等号，它的左右是不对称的，它是一种操作而不是一个简单的等号。例如：

$$E:=-E$$

的含意就是把E的值算出来（叫做E的当时值或者叫E的旧值），改变符号后得到一个新值再赋给变量E。

像开头的例子所述那样，变量A的值通过赋值语句定义好之后它就一直保有这个值，直到用一句新的赋值语句给它赋以新的值，这时变量A就失去旧的值而保有新的值。

把一个表达式E的值算出来赋给变量A这有着“缩记”的意义，当一个表达式很庞大复杂，而在程序中又要多次用到它时，我们可以通过赋值语句把它的值赋给变量A，从而在以后每次再用该表达式的值时就不必每次都进行计算而只需引用A就可以了，这种“缩记”的功能是赋值语句的本质功能之一。

赋值语句允许一个表达式的值同时赋给几个变量：

$$X:=Y:=Z:=E$$

在后面我们将要讲述变量的类型，在这种多重赋值的情形下，ALGOL 60 要求变量 X，Y，Z 具有相同的类型。

以上所述的赋值语句是仅仅对简单变量定义的，即赋值号的左P例如 A，X，Y，Z 等是简单变量，在 ALGOL 60 中赋值语句也可以对下标变量或函数名字进行定义。关于它们的赋值语句的规则在以后给出。

第 2 节　标识符和数(IDENTIFIERS AND NUMBERS)

ALGOL 60 是一种算法语言，它通过基本符号来构造语言中各种成分，这类似于在自然语言例如英语中通过英文字母来构造单词等等。

ALGOL 60的基本符号一共分四部分，即字母，数字，逻辑值和定义符。关于逻辑值和定义符在以后介绍，在这里我们首先介绍字母和数字。

ALGOL 60使用的字母是26个大写英文字母A，B，…，Z，26个小写英文字母a，b，…，z．这52个字母可以任意地缩减，即根据具体情况我们可以从其中任意去掉一部分字母。例如121机的自动化语言去掉了26个小写英文字母，原因是121机所使用的外部设备电传打字机的键盘上只有大写英文字母，52个字母还可以任意地添加别的字符而加以扩充，例如苏联的自动化语言就常常添加了俄文字母，一些国内的算法语言系统还添加了一些汉字字符，当然同时从52个字符中既去掉一些字符又添加一些字符也是允许的，一个具体的算法语言系统（在[1]中叫机器表示）中到底决定采用那些基本符号是根据需要和可能来定的，它在很大程度上取决于计算机的外部设备。

ALGOL 60 的字母没有单独的意义，它们是用来组成标识符和行的，行在以后再讨论。

ALGOL 60 基本符号的第二部分是数字，数字一共10个即0，1，…，9。它们是用来组成数，标识符和行的。

ALGOL 60 中的标识符也叫做名字，它是由任

意个字母（至少一个）和任意个数字组成的字母数字串，它的开头一个字符必须是字母。例如

Q
AB
C4
ALPHA

都是标识符，而

A—CD
5LM
EPS'T
GT.15

都不是标识符，它们分别由于开头的字符是数字或使用了非字母非数字的字符而破坏了标识符的形成规则。

标识符不允许数字开头是为了避免二义性。我们看

A:=B−3

如果允许标识符以数字开头我们就无法断定赋给A的值到底是B的值减去3还是B的值减去标识符"3"的值。

同样，在标识符中允许非字母非数字的其他字符也只会造成误会和混乱。

ALGOL 60规定在书写标识符的时候行外空格不提供任何信息，即

COSALPHA
COS ALPHA
C OS AL PHA

代表同一个标识符。

ALGOL 60中把标识符用来标识简单变量，数组，标号，开关，过程等。它究竟代表什么，在使用时由程序加以说明，关于所提到的数组等等概念以后将会逐步介绍，在ALGOL 60中有少数几个标识符例如SIN是保留作为标准函数用的，不能把它们派作别用，它们在121机自动化语言中不保留。

现在我们看一些ALGOL 60中数的例子。

0，+4，234，−571，.88312，−.03，
+.7，3.14159，+46.77，−10.003，
$+_{10}+6$，$-_{10}-8$，$_{10}8$，$-314159_{10}-5$，
$+467.7_{10}-1$，$.767_{10}+3$

上面例子中的数有整数（第一到第四个数），纯小数（第五到第七个数），既有整数部分又有小数部分的数（第八到第十个数），纯指数部分（第十一到第十三个数），以及既有数值部分又有指数部分的数（第十四到第十六个数），所有各种数之前都可以带正负号或不带正负号（叫无符号数）。从例子中我们可以看出无符号整数是由至少一个十进数字的十进数字串组成，无符号的纯小数（[1]中叫十进制小数）则由小数点后面跟上一个无符号整数组成，而小数点之前和之后都至少有一个十进数字或换一种说法整数之后跟一个十进制小数的则组成一般的有理数，上述三者就组成ALGOL 60中的十进制数，上例中的纯指数部分是由小10后面跟上整数组成，该整数可以是无符号整数，或带正负号的无符号整数，象第十四到十六个例子那样十进制数后面跟上指数部分也是ALGOL 60中的数，这些就构成了ALGOL 60中数的组成规则。

在印刷上有意义的空格，另起一行等在算法语言ALGOL 60中是没有意义的，但是为了阅读方便起见可以随意地使用它们。

第3节　表达式(EXPRESSION)

表达式是整个ALGOL的基础，作为科学计算的算法语言，这一部分是起着核心作用的，作为开始我们首先讨论简单算术表达式。

简单算术表达式是算术表达式的简单情形，它是由简单变量和数通过加，减，乘，除，乘幂（+，−，×，/，××。乘号×也可以记成*以避免同字母X混淆，乘幂××可以记成**或↑，在121机中采用×和××）组合而成。

应当注意在写表达式时乘号必须明显地写出来，例如

A×B

不能写成

AB

因为乘法运算符×起着分隔表达式中两个变量A和B的作用，如果省去了它，编译程序就把AB看作是一个标识符了，

也不可以把"×"写成"·"，例如

13.5

就很难区分它到底是27/2还是13×5=65。

表达式的运算有一套优先规则，它们是从左往右算；先乘除后加减；乘幂比乘除优先，如果有括号（圆括号，不使用方括号和花括号）则括号中的先计算，脱括号的规则是从里往外一层层脱。

例：

A:=B+C−D×E/F××G

其计算次序是：

1. B+C　　体现由左往右
2. D×E　　体现×比−优先（及自左往右）
3. F××G　　体现××比/优先
4. D×E/F××G　　体现/比−优先
5. B+C−D×E/F××G

简单算术表达式是计算数值的规则，它的结果总是一个数，我们按普通的算术运算规则加上优先规则，对表达式中的数和变量的实际值进行计算，就可以得到作为结果的这个数，表达式中变量的实际值是指该变量的当前值，即在动态意义之下最近一次赋给的值。

第4节 语句序列，复合语句和分程序(SEQUENCE OF STATEMENT, COMPOUND STATEMENT AND BLOCK)

我们看一个例子，计算复数乘法X+Yi乘0.6+0.8i。

首先要引入辅助变量U

```
U:=0.6×X-0.8×Y;
Y:=0.6×Y+0.8×X;
X:=U
```

这里为了计算两个复数相乘使用了三个赋值语句，它们组成一个语句序列，并且引进了新的字符“;”用来分隔语句，它使得前头一个语句的末尾标识符何处结束，接上的一个语句的头一个标识符何处开始得以区分开来。

如果我们以S代表语句的话，则语句序列的一般形式为：S; S;…;S

我们在这里还要引进新的语句括号'BEGIN'和'END'。这种用引号“'”和“'”括起来的字母串叫字定义符(WORD DELIMITER)。它们在印刷时是印成粗黑体字，在手写时为了方便写成在字母串之下加一杠，即 **BEGIN**。这种字定义符整个合起来作为一个整体表示一个意思，以后还将陆续引进别的字定义符，字定义符是ALGOL 60基本符号的一部分。

标识符和字定义符是有本质区别的，标识符没有固定的意义而字定义符有固定的意义，此外它们的区别还在于对字定义符在印刷，书写，打印和穿孔上有上述规则的限制。

语句括号'BEGIN'和'END'总是成对出现的，一个语句序列用语句括号'BEGIN'和'END'括起来就构成一个复合语句，例如：

```
'BEGIN' X:=5/13; Y:=12/13;
        U:=0.6×X-0.8×Y;
        Y:=0.8×X+0.6×Y;
        X:=U
'END'
```

一个复合语句可以把它看作是一个单个的语句，它的一般形式是：

'BEGIN'S;S;…;S'END'

在上面的例子中变量X,Y,U没有说明它们是实数还是整数，在121机中实数对应于浮点运算，整数对应于定点运算，整数应该完全精确地表出，实数则只有有限的精度。显然，实数比整数所能表达的数的范围要大得多，但定点数的优点在于它能完全精确地表达。

我们把上面的例子加上类型说明，就成为：

```
'BEGIN' 'REAL'X,Y,U;
        X:=5/13; Y:=12/13;
        U:=0.6×X-0.8×Y;
        Y:=0.8×X+0.6×Y;
        X:=U
'END'
```

这种在开头加上了说明的复合语句叫做分程序，在这个分程序中X,Y,U被说明为实变量，字定义符'REAL'叫做说明符，

类型说明的一般形式是在类型说明符(例如'REAL'或'INTEG')之后跟上一个或几个标识符，如果是几个标识符就构成一个标识符表(LIST)，表中的标识符彼此用逗号“,”隔开，最后加上分号“;”以便同别的说明或其他语句隔开，例如：'REAL'X,Y; 和'REAL'X,Y; 'INTEG'I,J;

以D代表说明，则分程序的一般形式是：

'BEGIN'D D,…;D,S;S;…;S'END'

显然，分程序也是语句，当赋值语句把表达式的值同时赋给几个变量时，ALGOL 60要求赋值号左部的几个变量应具有相同的类型。

考虑一个赋值语句

X:=E

其中X是实型或整型的简单变量，E是算术表达式。

当X和E的类型相同时该赋值语句的意义是确定的。

为了讨论X和E的类型不同时的相互转换，我们需要引入转换函数。

ALGOL 60中定义了一个标准函数。

entier(E)

它将实型表达式转换为整型表达式，其值就是不大于E的值的最大整数。例如 entier(3.35)=3。entier(-7.4)=-8。

当X和E的类型不同时，ALGOL 60规定自动地引进适当的转换函数，如果X是整型而E是实型，则转换函数定义为

entier(E+0.5)

也就是四舍五入的意思。

例如赋值语句K:=N×(N+1)×(N+2)/6，若K已说明为整，当计算右端表达式的结果同整数相差一点时（例如由于四舍五入误差所产生的），则在任何情况下都将通过entier(E+0.5)取整后再赋给K。

这种类型转换在某种程度上是依赖于运用(Implementation)的，亦即依赖于具体的机器和具体的编译程序的。各种机器的各种编译程序对于这种类型转换的规定可以有不同。

当X是实型而E是整型时121机编译程序自动将E的值化为浮点值赋给X。

在这里我们附带介绍整除运算÷，ALGOL 60的整除运算定义为：

$$A \div B = \mathrm{sign}(A/B) \times \mathrm{entier}(\mathrm{abs}(A/B))$$

即 $$\mathrm{abs}(A/B) - 1 < \mathrm{abs}(A \div B) \leqslant \mathrm{abs}(A/B)$$

它只对两个运算对象A,B都是整型时才有定义，且结果是整型，sign(E)是ALGOL 60的标准函数，当E＞0时它等于1，当E＝0时它等于0，当E＜0时它等于－1，标准函数abs(E)表示E的绝对值。

121机编译程序所定义的整除运算有所不同，它定义A÷B为将A/B的值四舍五入取整，而且不论A,B的类型如何都有定义，其结果是整型，在电传打字机上将整除运算符÷代之以//。

A+B,A－B,A×B当A,B均整型时才是整型，否则总是实型。

A/B不论A,B如何类型则总是实型的。

第5节 转向语句(GO TO STATEMENT)

在上面举的复数乘法的例子中，如果我们要重复乘多次，可通过ALGOL 60中的转向语句来实现：

```
'BEGIN' 'REAL'X,Y,U;
        X:=5/13; Y:=12/13;
    L:U:=0.6×X－0.8×Y;
      Y:=0.8×X+0.6×Y;
      X:=U;
      'GO TO'L
'END'
```

转向语句的作用在于改变程序执行的顺序，它相当于机器指令中的无条件转移，转向语句的构成是顺序运算符'GO TO'（或'GO TO'）后面跟上一个标号，标号可以是任意的一个标识符，对标号不需加以说明，转向语句所要转去的地方是一个语句，它称为转向语句的后继，其结构为标号之后跟上一个冒号":"，再跟上所要转去的语句：

```
L:S
```

一经转到语句L:S之后，程序就从L:S开始执行，直到程序结束或者产生新的转向为止，这里不会发生自动转回原处的情况。

ALGOL 60允许无符号的整数作为标号，但在一般的编译程序中都仅仅允许标识符作为标号。

转向语句的一般形式是

'GO TO'D_L

其中D_L是命名表达式，关于命名表达式的介绍将在以后给出，命名表达式的最简单情况就是标号。

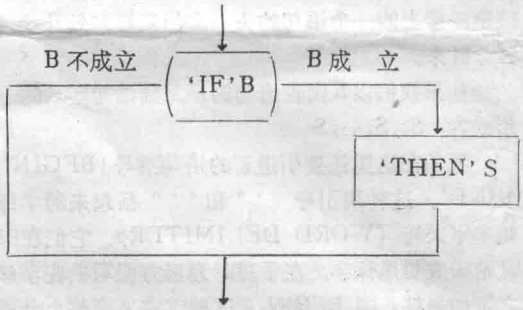
第6节 条件语句(CONDITIONAL STATEMENT)

条件语句相当于机器指令中根据一定条件实行条件转移，它有两种形式，一种是：

'IF'B'THEN'S

其中B是条件，S是无条件语句，如果B成立则执行'THEN'之后的语句S，若B不成立则不执行S，也就是改变程序的正常执行顺序而跳过了无条件语句S。'IF'B'THEN'叫做如果子句('IF'CLAUSE)。条件语句的第一种形式在ALGOL 60中叫如果语句。

把条件语句的上述形式画成框图就是：



在如果子句之后跟的语句S可以是转向语句，这两种语句的这种结合形式是非常有用处的，例如以前所举的复数乘法的例子，单有转向语句虽然能多次反复乘，但却停不下来，程序循环不已，但结合上条件语句之后就能够想乘多少次就乘多少次，譬如乘1000次，就可将程序写成：

```
'BEGIN' 'REAL'X,Y,U; 'INTEG'K;
        X:=5/13; Y:=12/13; K:=0;
    L:U:=0.6×X－0.8×Y;
      Y:=0.8×X+0.6×Y;
      X:=U;
      K:=K+1;
      'IF'K'LS'1000'THEN''GO TO'L
'END'
```

其中'LS'是关系运算符＜在电传打字机上的表示。

再举一个例子，用牛顿法求实数a的立方根，近

似公式是$X_{n+1}=1/3(2X_n+a/X_n^2)$。设所给的初值是$X_0$，当相接的两个近似根有9位有效数字重合时就停止。

```
'BEGIN' 'REAL'A,X0,X,Y;
        A:=N1; X0:=N2;
        X:=X0;
NEWTON:Y:=X;
        X:=(2×Y+A/(Y×Y))/3;
        'IF'ABS(Y-X)>.5₁₀-9×ABS(X)
            'THEN''GO TO' NEWTON
'END'
```

'IF'子句之后跟复合语句的例子是：

```
'IF'A'LS'O'THEN'
    'BEGIN''IF'B'LS'O'THEN'
        B:=-B
    'END'
```

意思是当A小于零时B就取自己的绝对值，

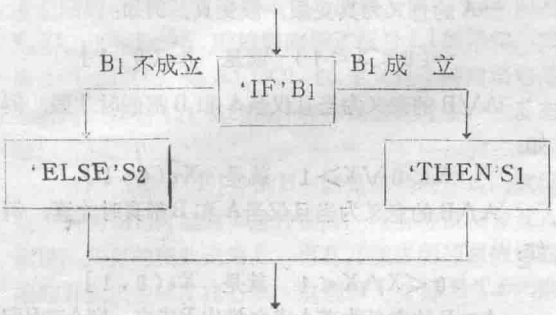
应注意，为了避免可能产生的二义性，规定凡是在'IF'子句之后即'THEN'之后跟有如果子句的话就必须用语句括号将它们括起来。

关于这一点我们看条件语句的第二种形式就更清楚了。

第二种形式定义为：

'IF'B_1'THEN'S_1'ELSE'S_2

画成框图形式就是



意为若B_1成立则执行S_1，若B_1不成立则执行S_2，它与第一种形式即如果语句不同，它不论B_1成立与否都要改变程序的正常执行顺序，或者跳过语句S_2（当B_1成立）或者跳过语句S_1（当B_1不成立），而如果语句只是在B不成立时才改变程序的正常执行顺序（跳过S）。

如果在'THEN'之后跟了如果子句，例如

```
'IF'B1'THEN''IF'B2'THEN'S1
    'ELSE'S2; S3
```

就会产生两种解释：

1. 'IF'B_1'THEN''BEGIN''IF'B_2
   'THEN'S_1'ELSE'S_2'END'; S_3
2. 'IF'B_1'THEN''BEGIN''IF'B_2
   'THEN'S_1'END''ELSE'S_2; S_3

在情形1之下，若B_1成立，B_2成立则执行S_1，S_3。
若B_1成立，B_2不成立则执行S_2，S_3。
若B_1不成立则执行S_3。

在情形2之下，若B_1成立，B_2成立则执行S_1，S_3。
若B_1成立，B_2不成立则执行S_3。
若B_1不成立则执行S_2，S_3。

条件语句的一般形式可以有两种样子：

1. 'IF'B_1'THEN'S_1'ELSE''IF'B_2
   'THEN'S_2'ELSE'S_3; S_4
2. 'IF'B_1'THEN'S_1'ELSE''IF'B_2
   'THEN'S_2'ELSE''IF'B_3'THEN'
   S_3; S_4

其执行情况是从左到右计算如果子句中的条件，直到找到一个成立的为止，接着就执行这个条件后面的无条件语句，如果这个无条件语句没有明显定出自己的后继，则接下去要执行的是语句S_4，从而定义符'ELSE'的作用是当它前面的条件成立时即它前面的无条件语句被执行时，它定出该无条件语句的后继是整个条件语句之后的语句，即S_4，如果'ELSE'之前的条件不成立时则它定出应执行的语句是它后面的语句。

如果在条件语句中的所有条件都统统不成立，则应执行的语句是所有不成立的条件之后的'ELSE'后面的语句，如果这个'ELSE'不存在，则整个条件语句等价于空语句（DO NOTHING AND GO ON）。对于上面列出的第1种样子当B_1，B_2都不成立时则执行S_3，然后执行整个条件语句之后的语句S_4。对于第2种样子，当B_1，B_2，B_3都不成立时则该条件语句等价于空语句，就紧接着执行整个条件语句之后的语句S_4。

这里附带说明一下空语句，空语句是语句的一种，它不进行运算，在写空语句时就是什么也不写，仅仅写下一个附加的分号“；”表示该处有一个语句但是是空语句，空语句类似于机器语言的空指令，空语句可以用来在某处放置标号，否则该处就不允许放置标号，例如：

```
'BEGIN' D;D;…;
        S;S;…S; 'IF'B'THEN''GO TO'
            L;
        S;
    L:
'END'
```

标号L标出'END'前面的空语句，它是用最后一个

语句S末端的附加引号“；”引入的，有了这个标号L，就能实现转到分程序的末尾，否则‘GO TO’‘END’是不允许的。

在P. NAUR关于ALGOL 60的正式报告[1]中对于条件语句的第1.5.4.款：转向进入条件语句，所说的“进入条件语句的转向语句其作用可直接从上面解说的关于‘ELSE’的作用得到。”其意思就是说：首先从条件语句的外部通过转向语句转向条件语句内部是允许的，其次转向条件语句内的一个无条件语句之后，就意味着执行该无条件语句，从而如果该无条件语句没有明显定出自己的后继的话，它的后继就是整个条件语句之后的语句。例如

```
        ⋮
'IF' B1 'THEN' L:A:=3 'ELSE' S2;
  S3;
        ⋮
'GO TO' L;
        ⋮
```

执行语句‘GO TO’L之后，就执行A:=3，再接下就执行整个条件语句的后继S3。

下面举些例子：

1.
```
'IF' K 'NQ' N 'THEN' 'BEGIN' Z:=
  Z+1; Y:=Y/Z 'END' 'ELSE'
  Y:=Y+X;
L:…
```

这程序也可改写为：

```
'IF' K 'NQ' N 'THEN' 'BEGIN' Z:=
    Z+1; Y:=Y/Z; 'GO TO' L
'END';
Y:=Y+X;
L:…
```

也就是说当‘THEN’之后的无条件语句是‘GO TO’语句时，‘ELSE’可以代之以分号“；”而不影响程序的内容。

2. 求三个变量A，B，C的最大者。

```
'IF' A 'LS' B 'THEN'
  'BEGIN' 'IF' B 'LS' C 'THEN' MAX:=
    C 'ELSE' MAX:=B
  'END'
'ELSE'
  'IF' A 'LS' C 'THEN' MAX:=C 'ELSE'
    MAX:=A
```

在如果子句中的条件B，最简单的一种是通过在两个量中间用一个关系运算符联起来而产生的。在ALGOL 60中有六个关系运算符：

$<$ $\leqslant$ $=$ $>$ $\geqslant$ $\neq$

它们在121机上是用

‘LS’‘LQ’‘EQ’‘GR’‘GQ’‘NQ’

来实现的，在这些关系运算符两端的量可以是数，简单变量或任意的算术表达式，这种由关系运算符加上它两端的量所构成的表达式叫布尔表达式（BOOLEAN EXPRESSION），它只可能取两个逻辑值之一，一个是‘TRUE’（真），一个是‘FALSE’（假）。如上所述真就表示条件成立，假就表示条件不成立，逻辑值‘TRUE’和‘FALSE’是ALGOL的基本符号的一部分。

在ALGOL 60中还有五种逻辑运算符，它们可以从一个或两个逻辑值产生一个新的逻辑值，它们是$\equiv$（等价），$\supset$（蕴涵），$\vee$（逻辑加），$\wedge$（逻辑乘），$\neg$（非）。在121机上它们分别是用‘EQV’，‘IMP’，‘OR’，‘AND’，‘NOT’来实现的。它们的运算规则可以列表于下：

A	‘TRUE’	‘FALSE’	‘TRUE’	‘FALSE’
B	‘TRUE’	‘TRUE’	‘FALSE’	‘FALSE’
$\neg$A	‘FALSE’	‘TRUE’	‘FALSE’	‘TRUE’
A$\vee$B	‘TRUE’	‘TRUE’	‘TRUE’	‘FALSE’
A$\wedge$B	‘TRUE’	‘FALSE’	‘FALSE’	‘FALSE’
A$\supset$B	‘TRUE’	‘TRUE’	‘FALSE’	‘TRUE’

$\neg$A的意义为真变假，假变真。例如：

$\neg(0<X\leqslant 1)$ 就是 $X\bar{\epsilon}(0,1]$

A$\vee$B的意义为当且仅当A和B都假时才假，例如：

$X\leqslant 0 \vee X>1$ 就是 $X\bar{\epsilon}(0,1]$

A$\wedge$B的意义为当且仅当A和B都真时才真，例如：

$0<X\wedge X\leqslant 1$ 就是 $X\epsilon(0,1]$

A$\supset$B的意义为若A成立推出B成立，则A$\supset$B取真值，否则取假值，A$\supset$B等价于$\neg$A$\vee$B。

ALGOL 60规定布尔表达式运算的优先规则是从左往右算，并且取以下的优先顺序：

最优先：计算算术表达式的规则

其次：关系运算符$<$ $\leqslant$ $=$ $>$ $\geqslant$ $\neq$

第三：$\neg$

第四：$\wedge$

第五：$\vee$

第六：$\supset$

第七：$\equiv$

圆括号中的表达式总是先算的。

第 7 节　条件表达式(CONDITIONAL EXPRESSION)

上一节所介绍的条件语句，它根据条件的是否满足而选择两个语句中的一个，现在介绍条件表达式，它根据条件的是否满足而选择两个表达式中的一个。

和条件语句的最大不同在于此处的 'ELSE' 不可省，其原因在于空表达式是没有意义的 (EMPTY EXPRESSION)，而空语句是有意义的 (DO NOTHING AND GO ON)。

对条件表达式有一个规则，即在 'THEN' 和 'ELSE' 之间如果是一个条件表达式则必须用括号括起来，例如上一节的例 2 利用条件表达式可简洁地写出：

```
MAX:='IF'A'LS'B'THEN'('IF'B'LS'C
    'THEN'C'ELSE B)'ELSE''IF'A'LS'
    C'THEN'C'ELSE'A
```

第 8 节　开关说明 (SWITCH DECLARATION)

在ALGOL 60 中通常使用的开关说明的形式是：

'SWITCH'T:=L_1,L_2,…,LN

其中 'SWITCH' 是说明符，T 是开关标识符，L_1，L_2,…,LN 是命名表达式，它最简单的情况是标号，这些赋值号右端的L_1,…,LN 叫开关表。

开关说明总是和转向语句'GO TO'T[E]联合起来使用的，它确定该转向语句的后继。例如'GO TO' T[I]，$1 \leqslant I \leqslant N$，则就转向标了标号 LI 的语句，如果 $I \in [1,N]$，则 ALGOL 60 定义这个转向语句是空语句，但是 121 机的编译程序规定这时要产生动态错。

'GO TO'T[E] 中的E一般是算术表达式，执行这个转向语句时就对 E进行计算，将所得值四舍五入取整，所得的整数记为 I，再在开关说明的赋值号右端的开关表中从左往右数，数到第 I 个标号 LI，则标了标号 LI 的语句就是要转去的语句。

在 ALGOL 60 中还允许 L_1,L_2,…,LN 中的某一个本身又是一个开关 S[E]，这样就产生了多重开关，不过在通常情况下是不会出现这种情况的。

例：

1.
```
'SWITCH'T:=L1,L2,'IF'I=K
    'THEN'L3'ELSE'L4,T[ABS(I-K)]
```

则对于转向语句

'GO TO' T[E]

我们有：

当 E=1 则 'GO TO'L_1。

当 E=2 则 'GO TO'L_2。

当 E=3 则 'GO TO' 'IF' I=K 'THEN' L3 'ELSE'L4。

(等价于 'IF' I=K 'THEN' 'GO TO'L3 'ELSE''GO TO'L4)。

当 E=4 则'GO TO'T[ABS(I−K)]。

在这个例子中有双重开关，此外其中的

'IF'I=K'THEN'L3'ELSE'L4

是一个表达式叫命名表达式，它还是特殊情况即是条件命名表达式。

命名表达式是求语句标号的规则，就象算术表达式是计算数值的规则，布尔表达式是计算逻辑值的规则一样，命名表达式可以是一个标号，则此时它的运算意义就是取这个标号，也可以是一个开关变量 T[E]，则此时它的运算意义应当由开关标识符 T 的开关说明来确定，即计算 E，将 E 四舍五入取整得整数 I，再从 T的开关说明的开关表中从左往右数，取开关表中的第 I 个元素。

2.
```
'BEGIN' 'INTEG'I,J;
        'SWITCH'A:=L1,L2,L3,L4;
        'SWITCH B:=M1,M2,A[I];
        D;…;D;
        S;…;S;
        'GO TO'B[J];
        S;…;S
'END'
```

转向语句 B[J]仅当J=1,2 或 J=3，I=1,2,3,4 时才有意义。

第 9 节　数组(ARRAY)

在ALGOL 60 中引入数组，使 ALGOL 语言使用起来很方便，便于处理各种数值计算的问题。

所谓数组它相当于通常数学中的多维向量，例如 'REAL''ARRAY'A[1:10]表示实型一维数组，其元素个数是10个。数组的元素叫下标变量，对实型来说它们都取实值，这里举出的实型数组 A的元素分别是A[1],A[2],…,A[10]。方括号中的整数是下标变量的下标。

又如 'INTEG' 'ARRAY' B [−4:5, 7:18, −9:−1] 表示三维整型数组，其元素都取整值。下标变量 B[−4, 7, −9]，B [−4, 8, −9]，…，B[5,18,−1]是它的元素，但是 B[−4,10,1]没有定义，因为它的第三个下标不在数组说明所指出的下标界偶−9:−1 的范围之内。

注意：下标界偶的下界值不应超过上界值。

同类型，同维数，同界的数组可以写在一起，

例如：

'BOOLE''ARRAY'C,D,E[4:7, −6:3]

表示二维的三个布尔型数组C, D, E，它们的元素都取布尔值，它们的下标变化范围都是从4到7和从−6到3。

当'ARRAY'之前的说明符'REAL','INTEG','BOOLE'未出现时，就认为它是'REAL'型的。

从本质上说下标都应当是整数，当下标是以算术表达式的形式出现时，就对该表达式进行计算，所得的值四舍五入取整就是相应的下标值。

现在我们引入了下标变量，以前介绍过简单变量，ALGOL 60中的变量就定义为且仅定义为这两种变量。ALGOL 60中算术表达式所包含的变量不仅可以是简单变量，也可以是下标变量，因此下标表达式也可以包含下标变量，从而象如下结构的下标变量是允许的：

H[U[K], V[K], W[U[I], V[I]]]

但通常是不会用到的。

应当注意数组说明中数组的各维的上界和下界通常是算术表达式，这些算术表达式中所含的变量应当是已经定义好的，否则编译程序将无法计算它的值到底是多少，从而无法确定该数组这一维的长度是多少，也就无法确定数组有多少个元素，因而无法分配其存贮，应当记住程序的最外层分程序其数组说明中各维的上、下界必须是常数，关于这些注记的更确切的讨论在以后介绍作用域以及局部量的概念时还要补充。

我们已经引进了下标变量，现在可以补充关于下标变量的赋值语句的规则了，例如：

A[E1, E2]:=F

则应计算左部下标变量的下标表达式E1, E2，将所得的值四舍五入取整该整数就是下标变量相应维的下标，同时计算右端表达式F，将表达式的值赋给上述的下标变量。

第10节 标准函数(SPECIAL FUNCTION)

ALGOL 60中建议了一些标准函数，前面我们已经用过一些了，例如ABS(E)，SIGN(E)，ENTIER(E)等。这些标准函数的名字ABS等是保留字，它们不能用作别的用处（但在121机中不保留）。在写源程序时可以不加说明地引用它们，它们可以象数一样出现在表达式中参加运算。

标准函数在很大程度上是依赖于运用的。

ALGOL 60建议标准函数应包括：

ABS(E) 求表达式E值的绝对值。

SIGN(E) 求表达式E值的符号。

SQRT(E) 求表达式E值的平方根（$\sqrt{E}$）。在121机算法语言中记为GN_2(E)。

SIN(E) 求E值的正弦。

COS(E) 求E值的余弦。

ARCTAN(E) 求表达式E值的反正切的主值。

LN(E) 求表达式E值的自然对数。

EXP(E) 求E值的指数函数。

ENTIER(E)求不大于E值的最大整数。

在121机算法语言中还增加了一些标准函数，它们是：

GN_3(E)求表达式E值的立方根（$\sqrt[3]{E}$）。

TAN(E)求E值的正切。

ARCSIN(E)求E值的反正弦主值。

TOINTG(E)对E值四舍五入取整。

TOREAL(E)把E值化为实数（浮点数）。

本节介绍的标准函数对'REAL'型和'INTEG'型变元都可进行运算，其中除SIGN(E)，ENTIER(E),TOINTG(E)的结果类型是'INTEG'型外，其余标准函数结果类型都是'REAL'型。

第11节 标准函数(SPECIAL PROCEDURE)

在ALGOL 60中未对输入/输出运算作出规定，但每个真正实现的ALGOL 60必然需要包含一些关于输入/输出运算的语句，以及一些显示，控制功能的语句。

这些语句并非ALGOL 60本身规定的，而是完全依赖于运用的，亦即依赖于具体的计算机，特别是计算机的外部设备，以及依赖于编译系统的，这些语句是过程语句（关于过程我们将在以后详细讨论），在写源程序时可以不加说明地直接引用它们，使用这些语句所调用的过程相当于手编程序中的子程序，自然这些过程都是用机器指定写成的。

在121机自动化语言中所提供的标准过程有以下几种：

READ R
READ I
READ B
READ (A)
PRINT (E)
APRINT (A)
TPRINT (B, E, "行内符号")
TEST (E)
PUSH (K, E)
DCXC (E1, E2, "行内符号", E3, E4)

```
GUDXIN | (A, B)
XINDGU | (A, B)
JUMP (K, L)
```

这些标准过程的解释见[2]。

第12节 循环语句 (FOR STATEMENT)

循环语句不是ALGOL 60的基本成分，凡是用循环语句能办到的事，用上面介绍过的语句也能办到，但是引进循环语句可以大大简化ALGOL程序，带来很大方便。

例如上面举过的复数乘法乘1000次的例子可以用循环语句来写：

```
'BEGIN' 'REAL'X,Y,U; 'INTEG'K;
    X:=5/13; Y:=12/13;
    'FOR'K:=1'STEP'1'UNTIL'1000'DO'
    'BEGIN'U:=0.6×X-0.8×Y;
           Y:=0.8×X+0.6×Y;
           X:=U
    'END'
'END'
```

这个例子中的'FOR'K:=1'STEP'1'UNTIL'1000'DO'叫做循环子句('FOR'CLAUSE)，其中K叫控制变量(也叫循环变量)，循环子句把一串值赋给控制变量，每赋一次就执行'DO'后的语句一次。

循环子句加上'DO'后的语句叫循环语句。

象如果子句一样，为了避免可能产生的二义性，在循环子句之后不要跟如果子句，如果跟的是如果子句的话则应当用语句括号'BEGIN'和'END'把它们括起来，这样做就不会产生二义性。例如：

```
'IF'B1'THEN''FOR'K:=1'STEP'
    1'UNTIL'100'DO''IF'B2'THEN'
    A:=SIN(E)'ELSE'ALPHA:=T
```

它可以有两种解释：

```
'IF'B1'THEN''FOR'K:=1'STEP' 1
    'UNTIL'100'DO''BEGIN''IF'B2
    'THEN'A:=SIN(E)'ELSE'
    ALPHA:=T'END'
```

或者：

```
'IF'B1'THEN''FOR'K:=1'STEP'
    1'UNTIL'100'DO''BEGIN''IF'
    B2'THEN'A:=SIN(E)'END'
'ELSE'ALPH A:=T
```

如果确实有把握不会产生二义性则'DO'之后如果子句的语句括号是可以不必加的。

以上引进的是循环语句的一种类型，叫步长型('STEP'—'UNTIL'型)，它的一般形式是：

```
'FOR'V:=A'STEP'B'UNTIL'C'DO'S
```

其中V是控制变量，ALGOL 60规定它可以是简单变量也可以是下标变量，但通常使用中一般只用V为简单变量，上式中的A是初值，B是步长也可以叫增量(或减量)，C是末限，我们不叫C为末值，因为不一定刚好取到它。例如：

```
'FOR'V:=0'STEP'3'UNTIL'24'DO'S
```

同

```
'FOR'V:=0'STEP'3'UNTIL'25.3'DO'S
```

是等价的，我们也不叫C为上限，因为B可能是负的，A，B，C都允许是算术表达式，且步长B和末限C都可以随控制变量V改变，亦即可以是V的函数。

这种步长型的循环语句其执行情况可准确而简明地用ALGOL语句描述如下：

```
    V:=A;
L:'IF' (V-C) ×SIGN (B) 'GR' O 'THEN'
    'GO TO'本循环之后;
    语句S;
    V:=V+B;
    'GO TO'L;
```

例：

1. $F(X,N)=X\cdot(X-1)\cdot(X-2)\cdots(X-N+1)$

```
'BEGIN''REAL'X,FACT;'INTEG'I;
    FACT:=1;
    'FOR'I:=1'STEP'1'UNTIL'N-1
        'DO'FACT:=FACT×(X-I)
'END'
```

2. 用HORNER法则计算多项式

$$F(X)=\sum_{I=0}^{N} A[I]\,X^{I}$$

及其导数

$$F_1(X)=\sum_{I=1}^{N} IA[I]\,X^{I-1}$$

在给定点的值。

```
'BEGIN''REAL'F, F1, X; 'INTEG'I;
    'ARRAY'A[1:100];
    X:=READR;
    F:=F1:=0;
    'FOR'I:=1'STEP'1'UNTIL'100 'DO'
        A[I]:=READR;
```

```
'FOR'I:=100'STEP'-1'UNTIL'0'DO'
    'BEGIN'F1:=F1×X+F;
           F:=F×X+A[I]
    'END';
PRINT(F);
PRINT(H)
'END'
```

循环语句的第二种类型叫离散型也叫单值元素表(SIGNAL VALUE ELEMENT LIST)。其一般形式为：

'FOR'V:=$E_1,E_2,\cdots,EN$'DO'S

$E_1,E_2,\cdots,EN$ 是算术表达式，当然也可以是算术表达式的最简单的情况即数，其执行步骤是算出 E_1 的值赋给控制变量V然后执行语句S一次，再算出 E_2 的值赋给控制变量V然后再执行语句S一次，…，最后算出表达式EN的值赋给控制变量V再执行语句S一次，完成循环。接着执行循环语句之后的语句。

循环语句的第三种类型是当型('WHILE'型)。例如：

```
V:=10;
'FOR'V:=V+5'WHILE'V<100'DO'
    P:=V115
```

当型的循环语句其一般形式是：

'FOR'V:=E'WHILE'B'DO'S

其中E是算术表达式，B是布尔表达式，其执行情况可以极其准确而简明地用ALGOL语句描述如下：

```
L:V:=E;
'IF'¬B'THEN''GO TO' 本循环之后;
语句S;
'GO TO'L;
```

例如第6节用牛顿法求实数的立方根的例子可以用当型的循环语句来写：

```
'BEGIN''REAL'A,X0,X,Y,D;
    A:=READR;
    X0:=READR;
    X:=X0;
    D:=ABS(X);
    'FOR'Y:=X'WHILE'D'GR'·5₁₀-9×
        ABS(X)'DO'
        'BEGIN'X:=(2×Y+A/(Y×Y))/3;
                D:=ABS(X-Y)
        'END';
    PRINT(X)
'END'
```

以上介绍的循环语句的三种类型可以混合使用以缩短源程序。

例：

```
1. 'FOR'V:=0'STEP'2'UNTIL'100'DO'
   S1; S2
```

它等价于：

```
'FOR'V:=0'STEP'2'UNTIL'50,52,54,56
    'STEP'2'UNTIL'100'DO'S1; S2
2. 'FOR'V:=0'STEP'.001'UNTIL'.05,
   .055,.065,.080,.09'STEP'.01'UNTIL'
   1'DO'S; S1
3. 'BEGIN''INTEG'K,B,W;
   'FOR'K:=1,2,3'STEP'2'UNTIL'9,
      5+B'WHILE'K'LS'100'DO'
      'BEGIN' B:=K;
              W:=B+K
      'END'
'END'
```

跟在'DO'之后的语句可以被标号，若有多个语句，它们必须用语句括号括起来。语句括号中的每个语句也可以被标号，'DO'之后的这些要执行的语句叫循环体。

使用转向语句从循环体中转出去时，在出口处控制变量的值有意义，它等于刚刚执行转向语句之前控制变量所具有的值，但是[1]规定在循环表元素用完而离开循环时，控制变量的值没有意义，但[4]规定此时有意义，它就是最后所得到的那个值。

[1]规定从循环语句之外使用转向语句转向进入循环体之中是禁止的。

第13节 注解(COMMENT)

在ALGOL语言中为了使编程序的人阅读源程序方便，常可在源程序中引进一些注解。这些注解仅仅是为了使人读起来方便，对编译程序来说它不起作用，就好象它不存在一样。

注解通常通过引入字定义符'COMMENT'来引进或者在闭语句括号'END'之后来引进，当编译程序碰到这种情况时就整段地跳过它们，我们有：

;'COMMENT'〈不包含分号";"的任何序列〉;

等价于；。即'COMMENT'和紧接的下一个分号之间的全部字符序列包括该'COMMENT'和紧接的下一个分号在内部全部被编译程序忽略。

'BEGIN''COMMENT〈不包括分号";"的任何序列〉;

等价于'BEGIN'。即'COMMENT'和紧接的下一个分号之间的全部字符序列包括该'COMMENT'和紧接的下一个分号在内都全部被编译程序忽略。

如果注解结构中的“任何序列”之中又包含了注解结构，则自左至右读过去时首先碰到的注解结构要优先按上述等价规则替换掉，否则会产生错误，例如：

;'COMMENT'…'BEGIN''COMMENT'…;

如果没有所述的优先性，则可以先替换后者，变成：

;'COMMENT'…'BEGIN'

这种形式就不是一个正确的注解规则了。

'END'〈不包含'END'和'ELSE'以及分号";"的任何序列〉

等价于'END'，即'END'同紧接着的下一个'END'或'ELSE'或分号";"之间的任何字符序列都将全部被编译程序忽略。

第14节　巴科斯(BACKUS)元语言公式

巴科斯元语言公式简记为BNF (BACKUS NORMAL FORM或BACKUS NAUR FORM)。它是一种上下文无关文法的一种记法，ALGOL 60的语法就是用BNF来刻划的，由于它简洁明了并且严谨完整，所以甚至有人认为只要对一个语言给了BNF，就等于刻划了该语言，虽然ALGOL 60的语法是由BNF给出的但ALGOL 60的语义并未形式化。

我们看一些例子：

〈基本符号〉::=〈字母〉|〈数字〉|〈逻辑值〉|〈定义符〉其中::=的意义是“定义为”，|表示“或”的意思，上式的意思就是基本符号定义为字母或者数字或者逻辑值或者定义符。

〈字母〉::=A|B|…|Y|Z

而字母又定义为A,B,…,Y,Z.

〈数字〉::=0|1|…|9

数字定义为0,1,…,9.

〈逻辑值〉::='TRUE'|'FALSE'

逻辑值定义为'TRUE'或'FALSE'.

〈定义符〉::=〈运算符〉|〈分隔符〉|〈括号〉|〈说明符〉|〈分类符〉。

〈运算符〉::=〈算术运算符〉|…

〈算术运算符〉::=+|-|×|/|//|××

……

这样利用BNF一层层定义是既严格又清楚的。

第15节　过程(PROCEDURE)

现在我们介绍被认为是ALGOL语言中比较困难但也是十分有用的概念——过程，它也叫做“单用过程”。

过程有点类似手编程序中的子程序，引入它使源程序的编写十分灵活，方便。

我们看一个例子：

A:=9×A×A-1;

它是对变量A进行修改，如果这种修改在程序中经常要使用，那么为了方便我们用一个标识符B表示上述运算，

'PROC'B; A:=9×A×A-1; (*)

在程序中我们写一个B就表示执行9×A×A-1送A的运算。

若A是已有明确定义的量，则它构成一个完整的语句，叫做过程语句，它与别的语句之间也象通常那样用分号“;”隔开。

象(*)那样的写法就叫做过程说明，B;后面的一段是过程的执行部分叫过程体。

我们看另一个例子，把变量A,B互换：

```
'PROC' CHANG; 'BEGIN' 'REAL'S;
        S:=A;
        A:=B;
        B:=S
    'END'
```

这也是一个过程说明，CHANG;之后用语句括号括住的部分是过程体，在程序中当我们要互换变量A,B时，我们只需写一个过程语句CHANG，那么就会完成上述的操作。

但在程序中我们需要互换的变量可能不止一对，因此引入形式参数的概念是方便的：

```
'BEGIN' 'REAL'A,B,C,D,E,…;
    ……
    'ARRAY'S[2:12],…;
    'PROC'W2(E,F); 'REAL'E,F;
        'BEGIN''REAL'S;
            S:=E;
            E:=F;
            F:=S
        'END'
    ……
    W2(A,B);
    ……
    W2(D,C);
    ……
    W2(S[3],A);
    ……
'END'
```

过程标识符W2有两个“形式参数”，在源程序中使用过程语句调用这个过程时，只需在形式参数的位置上代之以需要对换的变量即可。例如上面写出的W2

(A,B), W2(D,C), W2(S[3],A), 这里的A, B; D, C; S[3],A; 就叫做“实在参数”。应当注意实在参数都应当是已经说明了的。

形式参数只在过程体中有效，在过程体之外无效，亦即它是局限于过程体之中的。如果在过程体之外某个形式参数已经作过说明，例如上例中的E，它在过程体外已说明为实型变量，但在过程说明中又作为形式参数引入，则在过程体中E的实型变量的作用停止，待出了过程体之后E的实型变量的作用再恢复，在过程体之中E是作为形式参数来起作用的，形式参数在动态上的直观的意义相当于在过程体中空出位置，当通过过程语句调用该过程时，就把过程语句中的实在参数（它是程序中已说明过的，亦即有确定的定义的）填入该相应的位置，

在过程W2的过程体中通过说明‘REAL’S引入了一个暂时的辅助变量S，这是一个局部量，即它仅在过程体中有效，在过程体之外无意义，上面的例子中S曾被说明为数组标识符，到了过程体S又被说明为实型变量，这是两个具有不同意义的同名量，它们是在不同的分程序中分别说明的，这在ALGOL 60中是允许的。上面例子中的S它在进入过程体这个小分程序之前是数组，在进入过程体这个小分程序之后，S的数组的意义暂停，而作实型变量用，待出了过程体这个小分程序时，S的实型变量的意义失效，又恢复数组的意义。

这里说到的形式参数和局部量的概念虽然在语汇的意义上是相同的，即它们都局限于过程体这个小分程序中有效，出了过程体就无效，但它们在动态的意义上是本质不同的，局部量S在过程体这个小分程序中是通过说明（DECLARATION）来引入的，它的意义完全由小分程序中的这个说明确定，而形式参数E，F是通过过程说明中的区分部分（SPECIFICATION）指明它们是实型的，它们的含义仅仅在于E，F所对应的实在参数是实型的，而这些实在参数是已经说明了的，亦即是已经有了确切的定义的。

我们顺便介绍一下量的存在域和作用域的概念。

首先分程序是由加了说明的复合语句组成的，分程序可以彼此没有公共语句而互相独立，也可以一个套在另一个之中，但不允许两个分程序仅仅是一部分重叠，整个源程序也就是一个大的总分程序或复合语句，每一分程序中新出现的所有标识符，其中除了标号，过程说明中的形式参数，以及标准函数标准过程的名字之外都必须在分程序开始时加以说明，标识符是它被说明的那个分程序中的局部量，所谓局部的意思是当进入该分程序时，该分程序说明的标识符就具有了该说明所确定的意义，如果在进入分程序之前标识符原来已说明有意义，则进入分程序之后它的原有意义就暂停，而暂时具有该分程序对它说明的新的意义，当离开分程序时，这新具有的暂时的意义就失去，又重新恢复它进入分程序之前的原有的意义，如果标识符进入分程序，该分程序没有对它说明，则它就继续保有它原有的意义，并且对该分程序来说它是非局部量，

对分程序来说是局部的有：

1. 在分程序开头说明的所有标识符。

2. 出现在分程序中标分程序的组成语句的标号。

3. 当分程序是过程体时的所有形式参数，

所谓量的存在域就是对该量说明的那个分程序，亦即是包含该量说明的最小分程序，例如前面例子中数组S的存在域是整个大分程序，实型变量S的存在域是过程体，所谓作用域就是该量的说明有效的范围，即存在域中又包含了一个小分程序，对该量进行了同名量的再说明，那么从存在域减去该小分程序就是该量的作用域，例如上例中数组S的作用域是大分程序减去过程体这个小分程序，实型变量S的作用域和它的存在域相同。

分程序，循环体，过程体在习惯上叫三体，三体之外的转向语句ALGOL 60规定禁止转向进入三体之内，因为三体内的语句标号是局部于三体的，从三体之外不能得到它们的信息，因而也无法转向它们，

ALGOL 60中还允许形式参数的形式为‘PROC’A(I)RESULT:(R); …其中I,R为形式参数，I,R的分隔字符不是逗号而是)RESULT:(，它叫参数定义符，它与参数定义符逗号的意义等价，利用它可给出下一个形参的信息（例如用作结果），在运算上同逗号完全一样。

第16节 过程的值表(VALUE LIST)

至此所定义的过程还不太完全，不太实用，还需加进一些机构，我们看一个例子：

```
‘PROC’ MOD (P,Q,L,R); ‘INTEG’P,Q;
    ‘LABEL’L; ‘REAL’R;
    ‘BEGIN’‘INTEG’S;
        ‘IF’Q=O‘THEN’‘GO TO’L;
        S:=P−P//Q×Q;
        ‘IF’S‘LS’O‘THEN’R:=S+ABS(Q)
        ‘ELSE’ R:=S
    ‘END’
```

若有一个过程语句是：

MOD(N×(2×N+1), M×(M+7)−7, L, R)

其中形式参数P，Q对应的实在参数是算术表达式，

在过程体中P要用到两次，Q要用到3－4次，每次都得把表达式计算一下，这显然是很不方便的，为了克服这一缺点，我们引入两个局部变量P1，Q1；

```
'PROC' MOD (P,Q,L,R); 'INTEG'P,Q;
    'LABEL'L; 'REAL'R;
    'BEGIN''INTEG'S,P1,Q1;
        P1:=P; Q1:=Q;
        'IF'Q1=O'THEN''GO TO'L;
        S:=P1-P1//Q1×Q1;
        'IF'S'LS'O'THEN' R:=S+ABS(Q1)
        'ELSE' R:=S
    'END'
```

虽然这克服了上面例子中多次计算表达式的麻烦，但引入局部变量还是不方便的，我们在过程中引进一个机构叫做值表，即在过程的区分部分之前引入一个分类符'VALUE'，其后跟一个形式参数标识符表，它的意义是在调用该过程时，凡出现在值表中的形式参数，在进入过程体之前都换以对应实在参数的值，在进入过程体后它们就以这些值参加运算，用准确一点的语言说就是虚构一个包含该过程体的分程序，在这个虚构的分程序中对在值表中出现的局部变量（所谓局部是对此虚构的分程序而言）进行赋值，从而被赋值的变量（即在值表中出现的形式参数）被认为对过程体而言是非局部的，但对虚构的分程序而言是局部的。

引入了值表后上面的例子就可以改写成：

```
'PROC'MOD(P,Q,L,R); 'VALUE'P,Q;
    'INTEG'P,Q; 'LABEL'L; 'REAL'R;
    'BEGIN''INTEG'S;
        'IF'Q=0'THEN''GO TO'L;
        S:=P-P//Q×Q;
        'IF'S'LS'O'THEN' R:=S+ABS(Q)
        'ELSE' R:=S
    'END'
```

显然引入值表并未从本质上丰富ALGOL的表达功能，例如引入局部变量P1，Q1也可以，但值表的引入简化了ALGOL 60源程序的编写。

过程说明中出现的参数叫形式参数。凡是在值表中出现的形式参数叫赋值形式参数。凡是不在值表中出现的形式参数叫换名形式参数。赋值形式参数的区分部分一定要有，但换名形式参数的区分部分可以略去，121机的算法语言规定换名形参的区分部分也不允许略去。

过程说明的一般形式是：

〈过程说明〉::='PROC'〈过程导引〉〈过程体〉

〈过程体〉::=〈语句〉

〈过程导引〉::=〈过程标识符〉〈形式参数部分〉;〈值部分〉〈区分部分〉

……

过程语句的一般形式是：

〈过程语句〉::=〈过程标识符〉〈实在参数部分〉

在过程语句中的实在参数同过程导引中的形式参数在数目，次序以及对应参数的种类和类型上都要一致，它们的对照如下：

形式参数的种类和类型	实在参数的种类和类型
'STRING'	行
'REAL'	实型算术表达式
'INTEG'	整型算术表达式
'BOOLE'	布尔型算术表达式
'ARRAY'	实型数组标识符
'INTEG''ARRAY'	整型数组标识符
'BOOLE''ARRAY'	布尔型数组标识符
'LABEL'	命名表达式
'PROC'	过程标识符
'INTEG''PROC'	整型函数过程标识符
'REAL''PROC'	实型函数过程标识符
'BOOLE''PROC'	布尔型函数过程标识符
'SWITCH'	开关标识符

其中“行”是基本符号包括空格的任一序列，这序列中的行引号‘和’必须成对出现，而全部序列又括在行引号‘和’之中。函数过程则将在下一节介绍。

关于形式参数与实在参数对应的一些特殊情况在这里介绍一下。

在P. NAUR报告[1]的4，7，5，1中说：如果一个行在过程语句中用作实在参数，且定义该过程语句的过程体是用ALGOL 60语句写成的而不是用机器代码编成的，则这个行只能在该过程体内作为更进一层调用别的过程的实在参数使用。这个别的过程应该是用代码写成的过程。归根到底，行只能在非ALGOL的代码所表示的过程体中使用。

例如过程语句F(…,“行”,…)，F的过程体中有：

```
'BEGIN'…;
    TPRINT(B,E,“行”);
    ……
'END'
```

就属于这种情况。

另外出现在过程体中的赋值语句左部的换名形式参数它只能对应是变量的实在参数。如果它对应的实在参数是数或表达式的话，则它就必须是赋值形式参数。

最后，赋值的形式参数一般不能对应于开关标识符，过程标识符或行。（对函数过程例外。）

最后我们把过程的换名形参和赋值形参的区别再强调一下，赋值形参是当调用过程体时把实参的值在执行过程体的其他语句之前就赋给过程体的局部量，在执行过程体的其他语句时，是以该局部量的身分来参加（或者来出现）的，例如本节开初的例子中的P1，Q1．如果在执行过程体的其他语句时该局部量的值被改变，则由于它是局部量，这改变后的值不可能输出过程体，因此其值要作为结果来输出的过程的形式参数应该是换名的而不应该是赋值的，对于换名形参，当调用过程体时，是把相应的实参直接填入过程体中各对应的形参位置，这是与赋值形参本质不同的。

我们看一个例子就清楚了，设一个过程它有两个形式参数A，B．其过程体是：

```
'BEGIN'  A:=A+1;
         C:=SIN (A+B);
'END'
```

其中C是非局部量。

我们对两组实参来分别考察赋值和换名时的情形，设一组是(D,E)，其中D的值是2，C的值也是2，另一组是(D,D)，D的值是2。

对于实参(D,E)，当赋值时，过程体是：

```
D1:=D1+1;
C:=SIN(D1+E1);
```

亦即D1的值是3，C的值是SIN(3+2)=SIN(5)。

当换名时，过程体是：

```
D:=D+1;
C:=SIN(D+E);
```

其中D的值是3，C的值是SIN(5)。

对于实参(D,D)，当赋值时，过程体是：

```
D1:=D1+1;
C:=SIN(D1+E1);
```

其中D1的值是3，C的值是SIN(5)。

当换名时，过程体是：

```
D:=D+1;
C:=SIN(D+D);
```

D的值是3，C的值是SIN(6)。

第17节　函数过程(FUNCTION PROCEDURE)

以上定义的过程在调用时是通过过程语句来实现的，语句是算法语言的运算单位，它不能出现在表达式中，即不允许语句本身象数或变量那样参加运算。

现在我们定义一个新的过程叫函数过程，它可以象函数那样出现在表达式中参加运算，作为例子，前面定义的标准函数是无说明的函数过程。

函数过程的说明同过程的说明有两点不同：

1.函数过程的值的类型要明显写出，类型说明符'REAL'，'INTEG'等放在过程说明符'PROC'之前。例如：

```
'REAL''PROC'γ;γ:=GN2(X×X+Y×Y);
```

2.在函数过程的过程体中必须有一个赋值语句，对该函数过程标识符赋值。

例：计算5!+10!把结果放在X单元中。

```
'BEGIN''INTEG'X;
    'INTEG''PROC'F(N); 'VALUE'N;
        'INTEG'N;
        'BEGIN''INTEG'I, K;
            K:=1;
            'FOR'I:=2'STEP'1'UNTIL'
                N'DO'
            K:=K×I;
            F:=K
        'END'
    X:=F(5)+F(10)
'END'
```

在表达式中调用函数过程时，执行顺序是先执行函数过程，计算出此过程的值，然后以此值参加表达式的计算。

121机的算法语言规定在表达式中不允许出现有付作用的函数(SIDE EFFECT)。

什么是有付作用的函数？我们看一个P.NAUR举出的例子：

```
'REAL''PROC'SNEAKY(Z); 'REAL'Z;
    'BEGIN'SNEAKY:=Z+(Z-2)××2;
        W:=Z+1
    'END'
```

这个函数过程中有一个非局部量W，执行这个过程就会悄悄地改变W的值，这就是一个有付作用的函数过程，因此以下两个语句是不同的：

```
PIP:=SNEAKY(K)×W;
PIP:=W×SNEAKY(K);
```

我们看一个例子：

求 $\sum_{K=1}^{100}\frac{1}{K}$ 及 $\sum_{J=5}^{10}\mathrm{SIN}\,\frac{1}{J}$。

```
'BEGIN''INTEG'K, J;
    'PROC'F(I,N,M,P); 'VALUE'N, M;
        'INTEG'I,N,M; 'REAL'P;
        'BEGIN''REAL'Y;
            Y:=0;
```