科技参考资料

算法语言
ALGOL60
介绍

西安交通大学　　1974·12·

前　言

随着我国在社会主义革命和建设的发展，作为现代的先进计算工具电子数字计算机的应用，正越来越被广大工农兵和技术人员所重视。特别在伟大的无产阶级文化大革命后，无论在计算机的研制方面，或是推广应用方面，均出现了崭新的局面，达到了新的水平。事实证明只要坚持毛主席的无产阶级革命路线，计算机科学的发展定将加速我国三个现代化的进程。

为了满足我校广大工农兵学员和教员对编制程序，使用计算机方面的需要，我们编了这本册子。ALGOL是英文'算法语言'(algorithmic language)的缩写。ALGOL和FORTRAN(英文'公式翻译' formula translation的缩写)，都是为了程序设计自动化，使人们免于繁重的手编程序工作和便于资料通讯而设计的一种面向科学计算问题的程序设计语言。它们产生于五十年代末期，虽然十多年来程序设计语言在深度和广度方面有了很多的发展，但是这二种语言的一些主要思想仍然保留着，也就是说，许多科学计算的语言是在它们的基础上发展起来的；另一方面，就ALGOL，FORTRAN本身而言，它们仍在目前世界上最通用的语言之列。在我国，文化大革命前已开始注意ALGOL的编译工作，现在FORTRAN,ALGOL等程序系统的研制更得到了各方面的重视。普及程序设计语言的应用已成为明显的趋势。以上这些就是我们把本书介绍给读者的动机。

产生ALGOL的一个重要原因是便于通讯。简单的意义就是在某计算机上编制的程序，在某种意义上对另一工厂生产的或别种型号的计算机仍是有益的。因此这种语言避开了计算机之间的差异，也就是不考虑机器的各种物理特征（如内存容量，资源的种类、数量，指令形式、种类，数的形式等等），这些特征只是对ALGOL的编译系统（编译系统是机器指令的一个程序，它专门把语言写的程序——叫做源程序，翻译成机器指令的程序——称为目标程序）才是决定性的。所以本书在叙述中亦不考虑机器的具体物理特征。

本书在写法上不是从ALGOL的概念和严格定义出发，而试图通过一系列的例子和相应的叙述逐步地介绍出ALGOL报告中最主要、最常用的概念，这些概念的深度、广度以国内常见的ALGOL编译系统为依据。因而当读者掌握了这些一般ALGOL的主要原理之后，一方面可以独立地去彻底弄清ALGOL报告中的所有概念，或理解有关资料上用ALGOL所书写的程序的含义；另一方面，更现实的是只要读者借助具体机器的使用手册说明，就不难在该机上编制具体的ALGOL程序。在内容安排上，六章是基本内容，每章凡涉及到的重要基本概念均有标题醒目；每章后面附有练习。这些练习的参考答案在附录1中给出。附录2翻印是由陆汝钤、周龙骧同志重校的ALGOL60的修正报告原文。

本书对读者在计算机知识方面没有什么特别的要求。在数学方法方面我们亦尽量举些浅易的例子。目的是让读者了解语言的本身，和掌握如何正确的用它来书写程序。

在编写中我们主要参考了许孔时编译的《算法语言文集》及R. BAUMANN等人著的《ALGOL引论》。限于水平，文中难免有错误和不当之处，恳盼读者指点。

计数教研室

77/08

目　　录

第一章　基本概念

算法语言，顾名思义就是用来描述算法的一种语言。所以对一位使用者而言，为了正确地用它来书写程序，须要掌握两方面的内容：该语言中能“直接”涉及的量的种类；语言所能“直接”表达的运算能力和逻辑能力。*ALGOL* 是属于高级的程序设计语言，将会看到它所涉及的范围和表达的能力就面向科学计算而言是比较广，比较强有力的，但绝不能说已经是非常完美了。事实上事物总是在不断发展的，决不停留在一个水平上，特别对我国程序工作者而言，创造出适用我国实际情况的高质量的程序设计语言是一个光荣的任务。

我们从例子开始逐步地的引出基本概念。

[例 1]　我们来讨论二次方程式

$$ax^2+bx+c=0$$

的求解。对给定的实数 a，b，c(设 $a\neq 0$)，程序要算出满足这方程的根，设二个根为 root1，root2，则有计算式为：

$$root1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$root2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

程序就依据上述有关的量 a，b，c 和算法来求出 root1 和 root2 。如果五个变量就是如此命名，用 *ALGOL* 写出的程序有：

```
begin real a, b, c, root1, root2;
    read(a, b, c);
    root1:=(-b+sqrt(b↑2-4×a×c))/(2×a);
    root2:=(-b-sqrt(b↑2-4×a×c))/(2×a);
    print(root1, root2)
end
```

这就是 *ALGOL* 的一个最简单的**分程序** (*BLOCK*)，只要是分程序就有资格作为一个**程序**(*PROGRAM*)。

begin 和 **end** 是起括号作用的符号，在这里表示一个分程序的起始和结束。由于起的是括号作用，所以它们在一个完整的程序或分程序中必须**成对**地出现。

```
real a, b, c, root1, root2;
```

这是程序的一个**说明**(declaration)部分，这里说明五个变量在这分程序中均取实数。

```
read (a, b, c)
```

这是我们在这里假设的一种**输入过程语句**。在 *ALGOL* 中对输入/出没有作为一个部分来明显的给出规定，但是对具体机器的具体编译系统而言，可以确定若干种输入/出语句而纳入整个算法语言系统。正是由于输入/出涉及到计算机的外围设备的种类和数量，以及有关的输入/出的格式，因此在 *ALGOL60* 报告的文本中没有明显的涉及它。我们为了今后举例的

方便，就作了这样的假定：编译系统中已有一段手编的输入子程序，它对过程语句

read（变量，变量，……，变量）

就能实现从某输入设备送进与括号内变量个数相应的数值，而且按序把值赋给这些变量。如 read(a, b, c)；就是从某输入设备送进三个实数值，而且按序赋给 a，b，c。这样，程序内的这些变量与外部的数据在这时刻联系了起来。

$root1:=(-b+sqrt(b\uparrow 2-4\times a\times c))/(2\times a)$

这是个**赋值语句**（assignment statement），它把记号“:=”右部计算出来的一个根的值赋给变量 $root1$，$sqrt$ 代表求平方根，是一个标准函数。

$root2:=(-b-sqrt(b\uparrow 2-4\times a\times c))/(2\times a)$

这个赋值语句是求第二个根且把值赋给 $root2$。

print($root1$, $root2$)

这也是我们假设的一个输出过程语句，我们规定它把 print 后面圆括号中的变量的值，按书写的次序在某输出设备上传输出去。

通过这个解说，对这小程序有一个感性的，粗浅的认识。总起来说，一个分程序是由一对括号 **begin** 和 **end** 括起来的，对它自己专门要用到的变量在说明部分加以说明，此后就是一些要执行的语句。要注意，各类说明的结束要用分号“;”与后面的部分分开。同样，语句间也要用分号隔开，但 **end** 前一个语句结束可不用分号，因为括号 **end** 亦兼起了分隔的作用。

在这个例子中涉及了算法语言中许多语法单位，关于这些基本概念，在本章中将先介绍一部分，在以后的章节中再逐步完整。

算法语言是由基本符号构成的，它们是字母，数字，逻辑值和定义符（逻辑值留在以后介绍）。

字母

字母通常就是大，小写的英文字母：

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*
n	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
N	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*

数字

0 1 2 3 4 5 6 7 8 9

定义符 (*delimiter*)

在 *ALGOL*60 中规定几十种定义符，有些是英文单字如例 1 中的 **begin** **end** 和 **real**；还有一些是些符号，如我们已见到的+、−、×、/、↑、,、;、(、)等。定义符是本语言规定的一种专门符号，它们的意义通常与人们的常识相接近，它们所固有的明确意义是不可能被使用者所改变的。在 *ALGOL* 中凡是英文单字的定义符一律用粗体的黑字印刷以与别的区别。例如定义符：

array **begin** **comment** **do** **else** **end** **for** **go to** **if**
integer **real** **step** **then** **until** **while**

有了上述这些基本符号就可引出其它的语法单位了。

标识符(*identifiers*)

标识符在*ALGOL*中用于命名某些量或对象的。在例1中我们用来命名**简单变量**(simple variable)如a，b，c，root1，root2等；它也可以用来命名函数如sqrt；以后可以看到标识符还可以命名别的语法单位。

标识符是由一个字母带头的一串字母数字的字符列。因此它的长度的下限是一个字母，而上限在*ALGOL*报告中没有规定（但在具体的编译系统中一般要确定上限）。

因此下列均是标识符：

$A12$　　$M1A7$　　root1　　x

下面的则不是标识符：

$4F$　　（以数字带头）

$M1-H$　　（字母数字串中夹杂了定义符－）

标识符不能以数字开头，以免数字与标识符相混淆。标识符中不允许出现定义符是为了避免语法中可能出现的二义性。如果把$M1-H$也作为标识符，那么它又代表变量$M1$与H之差，这样就混淆了。

由于标识符亦用来命名标准函数，因此下面这些标识符要留给命名标准函数用：

abs　arctan　cos　entier　exp　ln　sin　sign　sqrt

数(number)

在*ALGOL*程序中要用到常数，这无论在计算式中或是说明部分均可能发生。如例1中就出现常数2和4等。因此我们要看一下常数有那些表示法。**常数**都是十进制的数，可以写成一个整数的形状，且可带一个正负号或不带正负号，如

$+1$　　0　　-3000　　123456789

常数也可带有一个十进制小数点，正负号情况同上，如

1.99　　+3.14159　　.5384　　−007.23　　732

第三种写常数的方式是用来对应叫作科学记法的，它是在上述写法的后面带上一个10的某次幂，以便利认别该常数的数量级。例如：

300000000 可写成 $+3.0_{10}8$ 或 $3.0_{10}+8$ 等

-0.00000003 可写成 $-3.0_{10}-8$ 或 $-.30_{10}-07$等

要注意的是在定义符$_{10}$的前面是一个整数或是一个带小数点的数，在其后面跟上一个幂次，这幂次一定是一个整数，意思是前面的常数乘'10的若干次方'。下列诸常数写法均是正确的且具有相同的值：

$3.14159_{10}0$　　$31.4159_{10}-1$　　$.000314159_{10}4$

算术表达式(arithmetic expression)

表达式在语言中是一能力很强的语法成份，这里先介绍一种**简单算术表达式**。在例1中方程两个根的求得就是通过赋值语句中的表达式而得到的。简单算术表达式与通常代数中的写法稍有不同，其原因多半是由于计算机的读写设备，以及为编译程序的识别提供便利。

简单算术表达式是由一些运算符，运算分量和括号所组成的。表达式

$A+2-B$ 和 $(A+B)/2$

中运算符是+，－和/，运算分量是简单变量A，B和常数2。

在*ALGOL*中运算符的意义可列表如下：

运算符	含义	例
+	加法	$A+B$
−	减法	$3-x$
×	乘法	$r1\times T$　$4\times A\times C$
/	除法	$M1/M2$　$(A+B)/(C+D)$
↑	指数运算	$C\uparrow 2$　$10\uparrow P$

二个运算分量之间的运算符一定要写出来，例如乘法符号“×”要特别注意，不能把 $M1\times M2$ 写成 $M1M2$。因为编译程序必须能对标识符 $M1M2$ 与表达式 $M1$ 的 $M2$ 倍加以区别，如果这样写了按规定只能代表一个标识符$M1M2$。同样也不能写出$(A+B)C$，（这时将引起语法出错）而应写成$(A+B)\times C$。

注：在 *ALGOL* 中还有运算符÷，以后解释。

在一个表达式中，运算的次序有下述的优先级关系：

最优先的是指数运算↑，次之是乘除法×、/、÷，最次的是加减法(或取负)+、−。

在相同的优先级中运算的次序是先左后右。与通常数学中的意义一样，为了改变运算次序，可以加圆括号。另外为了机器识别的需要，表达式的分母一般均要加圆括号。

对于指数运算也有由左向右的次序。因此$2\uparrow 3\uparrow 4$表示$(2^3)^4$，而$2\uparrow(3\uparrow 4)$表示$2^{(3^4)}$。

运算分量可以由变量，数，和标准函数等所组成。

标准函数(*Stantard function*)

算法语言所提供的标准函数名在标识符一节已经提及，标准函数是在表达式中作为运算分量而直接引用，它的一般用法是：

函数名　(E)

其中E是任意的表达式。在例1中的 sqrt$(b\uparrow 2-4\times a\times c)$，$E$就是$b\uparrow 2-4\times a\times c$。这里的$E$在引用时是起着实在参数（它的精确含义在以后给出）的作用。

下列的标准函数名有通常的意义：

sqrt(E)	E的平方根($E<0$ 时没有定义)
sin(E)	E的正弦（E为弧度）
cos(E)	E的余弦（E为弧度）
arctan(E)	E的反正切（值在区间$-\frac{\pi}{2}$，$+\frac{\pi}{2}$之内）
ln(E)	E的自然对数($E>0$)
exp(E)	E的指数函数
abs(E)	E的绝对值

这些函数的值均属于实数类型，sqrt 和 ln 的自变量的数值一定要适合函数定义域的要求，否则给出出错信息。标准函数 sign 和 entier 的值属于整数类型，其定义为：

sign(E)　E的符号函数

$$\text{sign}(E)=\begin{cases}+1 & E>0\\ 0 & E=0\\ -1 & E<0\end{cases}$$

entier(E)　其值不超过E的最大整数

例如

entier(3.14)=3

entier(−7.5)=−8

entier(2)=2

entier(−5)=−5

有了这些标准函数，现在再进一步来叙述指数运算是如何求值的：

当指数是整数时，用连乘法。如$y \uparrow 3$，它等价于$y \times y \times y$；

当指数是实数时，就取对数和指数运算，因此如$y \uparrow 2.1$，它将是$\exp(2.1 \times \ln(y))$。

简单变量(simple variables)

变量分简单变量和下标变量（后者在以后介绍）。在这里每个简单变量代表一个数值，它或是整型的，或是实型的。在具体的计算机上，一般实型值对应于一个浮点数值，整型值则对应一个定点数值。因此实型之间运算的结果通常会受到舍入的影响，而整型之间的运算（如果不参加除法）结果，则保持严格精确。

满足了下列两个条件，在表达式中才可用简单变量。

1. 在程序首部的说明部分，有对这简单变量的说明，以区别它是实型(real)或整型(integer)。

2. 该变量在进入表达式之前已被赋值。

类型说明(type declaration)

类型说明位于分程序的首部，在该分程序中自己专门要用到的所有变量都必须加以说明。说明是由定义符 real 或 integer 和后面的变量标识符的一个表所组成。

例如：

```
real    a, b, c, root1, root2;
integer n, m, i, grad;
```

这里，表的长度没有限制，同类变量标识符之间用逗号“，”来分开。分号是用来区分下一个说明或结束整个说明。

赋值语句

在例1中可看出对变量的赋值有两种途径，其一是依靠输入过程的引用，二是通过计算表达式的结果，由赋值语句把结果值与变量联系起来，赋值语句的一般形式：

$$V:=E$$

其中V代表任意一个变量，E是表达式。

语义是执行该语句后，‘:=’左边的变量V取得表达式E的结果值。符号‘:=’是一个带方向性的动作，表示把它右边的值赋给左边。注意它并不是通常数学中的等号概念。因此，在表达式E中也可以出现左边的变量V，这时，赋值语句的效果是用V的“旧”值参加表达式的计算，尔后把其结果给V，以作V“新”的值。

计算和

$$s=\sum_{i=1} a_i$$

可用如下的赋值语句串来得到：

```
s:=0;
```

$$s:=s+a1;$$
$$s:=s+a2;$$
$$s:=s+a3;$$

最后一个语句执行之后就得到和 s。

我们可以自己估算下面两个语句执行之后变量 VAR 将取什么值？

$$VAR:=4;$$
$$VAR:=VAR\times(VAR-1)+VAR/(VAR-2);$$

（答案是14）。

多重赋值

把同一个值赋给一个以上的变量往往可以精简语句，*ALGOL* 中有多重赋值语句。它的一般形式是

$$V:=V:=V:=\cdots\cdots V:=E$$

其中 $V:=V:=V:=\cdots\cdots V:=$ 称为赋值语句的左部表，左部表中的所有变量应已说明是同一类型的。左部表的长度没有限制。例如 X、Y 已说明是 Integer 型的，语句

$$X:=Y:=0$$

置变量 X，Y 均为零。

我们要记住赋值语句的左部只能是变量而不能是别的什么。

现对表达式再说几句，简单算术表达式也分 real 型和 integer 型：如果运算对象都是 integer 型，那经过 +，−，×的运算，表达式也是 integer 型的。但运算对象中只要有一个是 real 型，表达式就要成为 real 型。而对于除法/不论什么情况，结果均是 real 型。运算符整除'÷'只有当二个运算对象均是整型时才有定义，所得结果也是整型，$a\div b$ 等价于

$$\mathrm{sign}(a/b)\times\mathrm{entier}(\mathrm{abs}(a/b))$$

这里，a、b 均是整型的，且 $b\neq 0$。

因而对赋值语句也存在类型的关系问题，如果左部变量是 real 型，不管右边的表达式的类型它就取相应等价的值。如果左部变量是 integer 型，而表达式的结果是 real 型，那么认为自动地调用适当的转换函数 entier，变量赋的值等于

$$\mathrm{entier}(E+0.5)$$

这相当于对实数值进行四舍五入。

现在我们可以抽象出分程序的一般结构：一个分程序是由一些说明和语句串组成，而且是由 **begin** 与 **end** 括起来，所以它的一形式是：

$$\textbf{begin}D;\ D;\ \cdots\cdots D;\ S;\ S;\ \cdots\cdots S\ \textbf{end}$$

这里 D 代表任一说明（类型说明是其中一种），S 代表任一语句。说明只是告诉编译系统使用者定义了一些什么东西，所以它起定性的作用，而语句才指示动作，进行运算。例 1 中我们已见到了赋值语句和输入/出过程语句，（语句还不止这些种类）。有时一串语句描述了机器动作的一个确定过程，因此，我们可以设想把这样一串语句作为一个语句单位。为了实现这个事实，可以应用语句括号 **begin** 与 **end**，这样组合起来的语句称为**复合语句** (Compound statement)，它的一般形式是：

$$\textbf{begin}\ S;\ \cdots\cdots S;\ S\ \textbf{end}$$

这里 S 代表任一语句。

复合语句与分程序的唯一差别在于分程序的首部至少有一个说明，而复合语句不具有说明部分。它们所包括的语句是很一般的语句，即可以是简单的语句，也可以又是复合语句或分程序。

因此，在结构上，分程序和复合语句之间的关系是递归的，但根据它们的定义，对由一对 **begin** 与 **end** 所括起来的内容要确定它的语法成分还是明确的。

什么叫做 **ALGOL** 的程序？报告中说：程序是一个分程序或一个复合语句，该分程序或复合语句不包含在其它语句之中并且不使用它所不包含的其它语句。

对于例 1 我们还可以应用复合语句来写出程序：

```
begin real A,B,C,root1, root2;
    begin read(A, B, C);
        root1:=(-B+SQRT(B↑2-4×A×C))/(2×A);
        root2:=(-B-SQRT(B↑2-4×A×C))/(2×A);
        print(A, B, C, root1, root2)
    end
end
```

第一章 练 习

1. 下面那些可用来作标识符？

   (*a*) end cos arcsin

   (*b*) 2*A* *X2* *TRA*

   (*c*) *ALGOL60* *PL/1* *X2.5*

2. 把下列表达式写成 *ALGOL* 形式。

   (*a*) $\frac{n(n-1)}{2}$ $\frac{a}{b-c}$ $\frac{x-y}{x+\frac{x-y}{z}}$

   (*b*) $3x+2y^2$ $x+\frac{1+x}{x^2}$ $y+z\frac{1+x^2}{x}$

   (*c*) x^{a+b} x^{a+b^c} $(x^y)^z$

   (*d*) $\frac{1}{\frac{1}{a}+\frac{1}{b}}$ $\frac{\frac{u}{x}}{\frac{v}{y}}$ $\frac{x}{y}\left(1+\frac{a+b}{v}\right)\frac{\frac{1}{a}+\frac{1}{b}}{c+d}$

   (*e*) $\sqrt{a^2+b^2}$ $|x+y|$ $\tan a$

3. 说出下列程序输出的诸值。

   (*a*)
   ```
   begin integer i, j; real x, y, z;
       i:=5; j:=-2;
       x:=3.14; y:=i×x/j;
       z:=x+y×i; x:=y+z;
       print(i, j, x, y, z)
   end
   ```

(*b*) **begin integer** a, b, c; **real** u, v, ;
$a:=b:=2$; $u:=v:=1.2$;
$w:=a+b+u$; $c:=w+v+a$;
print(a, b, c, u, v, w)
end

4. 试编出 *ALGOL* 的程序。

(*a*) 读进两个代表直角三角形的两直角边的长度，然后用勾股弦定理计算斜边。（斜边$=\sqrt{a^2+b^2}$）

(*b*) 求圆柱形的表面积，设半径为 5.6，高度为 27。

第二章　流程的控制

在上章的例子中，用语言所写的程序执行的次序是按照语句的先后顺序。但这是一种特殊情况，对于较复杂一些的计算常常需要程序执行到某一语句之后即按一定的需要，改变原来的执行次序，而转向执行别的一个语句所开始的一串语句。这种转向有时是无条件的，有时是由某种条件来规定的。现在我们分别介绍这些有关概念。

例1所写出的程序是用来计算任一个一元二次方程的解，如果我们来求一串一元二次方程的解，那么程序可改写为：

```
begin real A, B, C, root1, root2;
start: read(A, B, C);
    root1:=(-B+SQRT(B↑2-4×A×C))/(2×A);
    root2:=(-B-SQRT(B↑2-4×A×C))/(2×A);
    print(A, B, C, root1, root2;);
    go to start
end
```

这程序当执行到 **go to start** 后就无条件地转向由 **start** 所标出的语句即read(A,B,C)。假若在初始数据中已准备了一连串的系数，就实现所提出的要求。这里有二个概念：一是**标号**(label)，一是转向语句（**go to** statement)。

标号

对任一语句可以用标号来标出，**标识符**可用来作标号，**无符号整数**也可用来作标号。冒号“：”作为标号间隔，一个标号语句具有形式

$$L：S$$

这里L表示标号，S表示语句（简单的或复合的）。在我们的例中，标号为 start，语句为 read(A, B, C)。无符号整数作为标号时其开头的零不起作用，因此 00123 与 123 一样对待。有标号的语句被看作具有通常意义的语句，它们可任意地被用来作为复合语句的一个组成部分。下面这些都是有标号语句的例：

$$135：A:=(R+R)\times 3.14$$

$$B3：x:=y-a\times a$$

$$\text{determinant}：D:=a1\times b2-b1\times a2$$

在程序中运用标号的目的是为了将来能转向到它所标出的那个语句，因此不同的语句不允许具有同样的标号，以免混淆。标号还能用来加强一个程序清晰的程度，它们可用来作为副标题或语句的编号。

转向语句

定义符 **go to** 用来表示“转向”，简单的转向语句由这个符号并紧接着一个标号所组成，形式为：

go to L

程序执行它的效果是将流程无条件地转去执行 L 所标出的那个语句。

一个语句的执行，常常要受到某些条件的约束，例如，为了执行 root1:=(−B+SQRT(B↑2−4×A×C))/(2×A)，首先 $A\neq 0$，否则数学上就无意义，另外 A 的绝对值亦不能太小，太小的话有可能使运算溢出（超过了机器能表示的最大数的范围）。刚才写出求一串一元二次方程的介的程序，事实上亦要满足这些条件才能正常运算，而且上述程序也没有表明何时运行才结束。因此我们再把程序改为：

```
begin real A, B, C, root1, root2;
    start : read(A, B, C);
    if ABS(A)<10−19 then go to out;
    root1:=(−B+SQRT(B↑2−4×A×C))/(2×A);
    root2:=(−B−SQRT(B↑2−4×A×C))/(2×A);
    print(A, B, C, root1, root2);
     go to  start;
out: end
```

这程序执行时，当读进三个实数后，就分析一下 A 的绝对值是否小于 10^{-19}，如是则转向结束，否则照样进行下去，这样我们在初始数据中若特意安排最后一组系数的 A 的绝对值小于 10^{-19}，来结束运行。这里涉及下述二个概念：

空语句(dummy statement)

空语句就是什么也没有，本例就是空语句的一个典型用法，因为按题意满足 $ABS(A)<10^{-19}$ 时要结束运行，因此在 **go to** start 语句与 **end** 之间要安排一个空语句，而它具有的标号是 out。

条件语句(Conditional statement)

为了使条件形式化，我们利用二个算术表达式的数值之间的比较，下列是关系运算符：

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

它们具有通常数学中的意义。条件简单地表为

$$E_1\ \rho\ E_2$$

这里 ρ 表示上述六个关系运算符之一，E_1 和 E_2 是任意二个算术表达式。条件语句是由如果子句(if clause)所组成，它的简单形式为：

if $E_1\ \rho\ E_2$ **then**

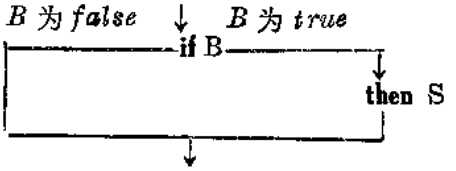
在本例中为 **if** abs(A)<10−19 **then**。

条件语句有二种形式：

不对称形式

if $E_1\ \rho\ E_2$ **then** S

二个表达式 E_1，E_2 经过关系运算符 ρ 的比较，有一个确切的答案，是“真”(true)还是“假”(false)。在这里若真，则执行 **then** 后面的语句 S，否则（即是“假”）就跳过 **then** 后面的 S 而继续执行下去。把条件写为 B 来表示，那么流程为：

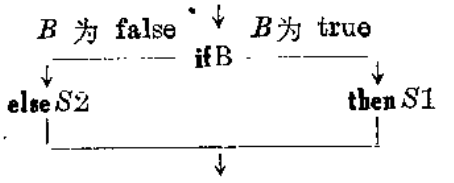


刚写出的程序中就是用到了不对称形式的条件语句，**then** 后面的语句是 **go to** out，因此当条件为“真”时就转向 out 所标出的语句。

对称形式：

if B **then** S_1 **else** S_2

它的意思是：若B为“真”则执行S_1，并跳过S_2，反之，若B为“假”则跳过S_1执行S_2。如果S_1，S_2均不是转向语句，那么，“真”或“假”的任一情况下，计算将继续执行条件语句后面的语句，因此它具有如下的对称形式：

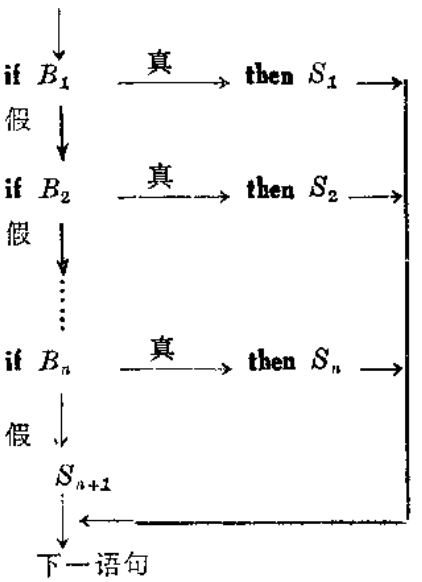


这里S_1，S_2可以是任何的简单或复合语句，但是条件语句中 **then** 后面的语句S_1不允许又是条件语句。

由于 **else** 后面可以再跟条件语句，这样条件语句可以控制任意多个而不只是两个不相容的语句。我们观察下列语句

if B_1 **then** S_1
else if B_2 **then** S_2
⋮
else if B_n **then** S_n
else S_{n+1};
下一语句；

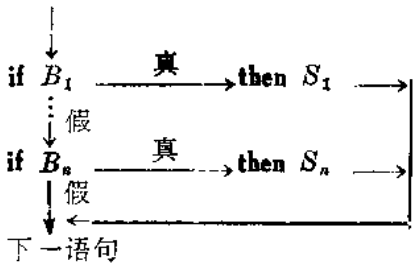
这里它的流程框图有：



还有一种复杂的条件语句是以不对称形式结束的：

if B_1 **then** S_1
⋮
else if B_n **then** S_n;
下一语句；

它的流程为：



我们列出这二类条件语句是有些差别的，前者在条件语句中列出的 S_1，…… S_{n+1} 中总会有一个被执行的。后者列出的 S_1，……S_n 中有可能没有一个被执行而均被跳过。

［例2］ 还是考虑介一元二次方程

$$ax^2+bx+c=0$$

其中 a，b，c 为任意实数。现利用条件语句考虑系数的各种情况得出相应结果。

首先当 $a\neq 0$ 时，方程具有二个实根或一对共轭复根；当 $a=0$，$b\neq 0$ 时具有一个实根；$a=0$，$b=0$ 时方程无意义。程序可写成：

```
begin real a, b, c, x1, x2, real, imagi, contiadiction, discr, r;
      read(a, b, c);
      x1:=x2:=real:=imagi:=contradiction:=0; discr:=b×b-4×a×c;
      if a≠0 then
   1 : begin
      if discr≥0 then
      11 : begin
           x1 : =(-b+sqrt(discr))/(2×a);
           x2 : =(-b-sqrt(discr))/(2×a)
           end
                else
      12 : begin
                r : =sqrt(abs (discr) );
                real:=-b/(2×a);
                imagi:=r/(2×a)
           end
      end
      else if b≠0 then
      2 : begin
                     x1 : =-c/b
                end
           else
      3 : begin
           contradiction:=c
           end;
```

```
        print(a, b, c, x1, x2, real, imagi, contradiction)
    end
```

程序中的一些标号仅是为了醒目作用，标号 1 的复合语句表示 $a \neq 0$ 时方程是非退化的，标号 11 的复合语句用来求出二个实根，标号 12 的复合语句是求共轭复根的实部和虚部。标号 2 的复合语句是当 $a=0$ 但 $b \neq 0$ 时求得一个实根。标号 3 表示 $a=b=0$ 方程没有意义。以上这些情况从输出结果的分析可以明白看出。

有了条件语句，我们就可以做很多重要的计算，例如我们若要用迭代的方法求某一方程的近似解就可以实现。

[例 3] 求方程

$$0.9\sin x - x + 0.5 = 0$$

的一个近似根。

这个方程可以用简单迭代法求出近似根，为此先把它化为等价的方程：

$$x = 0.9\sin x + 0.5。$$

这样，迭代的公式将有

$$x_{n+1} = 0.9\sin x_n + 0.5 \quad n = 0, 1, 2, \cdots\cdots$$

可以证明，对任意给出一个初值 x_0，则迭代序列 $\{x_n\} \to x$，这里 x 是方程的一个真根。我们当然不能无穷次地迭代下去，只要满足某规定的精确度 ε 就可以了。因此当计算到 $|x_{n+1} - x_n| \leqslant \varepsilon$ 就可停止计算，并把 x_{n+1}(或 x_n)就当作方程某个真根的近似值。根据题意，希望最后输出的数据应该有这些内容：最后相邻二个迭代解，另外，再把初始值 x_0 亦打印出，这程序可写成：

```
begin real x0, E, x, y;
        read (x0, E);
        x:=x0;
    200: y:=0.9× sin (x)+0.5;
        if ABS(y-x)>E then
        begin x:=y;
            go to 200
        end
        else
    100: print(x, y, E, x0)
end
```

具有上面的这些知识，我们已经可以独立地编制一些程序来解决相当范围的计算。但我们要注意一些原则，即为了达到同样的计算目的，你一定要设法使程序尽量简练，就是说尽量少用一些语句或各种记号和符号。因为用多了，笼统地说可能要多花费编译及运算时间，更重要的是增加内存的占用。因此，编制较大型题目的程序时应细致地考虑结构的简练。

第二章　练　　习

1. 计算函数值

$$y=f(x) \qquad 0\leqslant x<6$$

其中

$$f(x)=\begin{cases} f_1(x) & 0\leqslant x<2 \\ f_2(x) & 2\leqslant x<4 \\ f_3(x) & 4\leqslant x<6 \end{cases}$$

这里

$$f_1(x)=-x+2.5$$
$$f_2(x)=2-1.5(x-3)^2$$
$$f_3(x)=\frac{x}{2}-1.5。$$

2. 用下列“牛顿迭代法”来求任一实数 a 的三次根

$$x_{n+1}=\frac{1}{3}\left(2x_n+\frac{a}{x_n^2}\right) \quad n=0,1,2,\cdots\cdots$$

对指定的初值 x_0，要求相邻近似值有 10 位有效数字相同。

（注：10 位有效数字相同的条件是 $|x_{n+1}-x_n|\leqslant 0.5\times 10^{-10}\times|x_n|$）

3. 试用梯形求积公式，编出下列定积分程序：

$$I=\int_0^1 \frac{\sin x}{x}$$

其中积分区间[0，1]分成三段：[0，$^1/_3$]，[$^1/_3$，$^2/_3$]，[$^2/_3$，1]，要求在每段上应用梯形求积公式。

注：梯形求积公式是：对积分

$$I=\int_b^a f(x)dx$$

有近似值

$$I=\frac{b-a}{2}(f(a)+f(b))$$

4. 用二分法求方程

$$f(x)=-x^3+x^2+1$$

在区间[1，2]中的一个近似根 $\bar{x}$，计算到 $|\bar{x}-x|<10^{-10}$，其中 x 是方程的一个真根。

注：对于任意满足条件 $f(x_1)\cdot f(x_2)<0$ 的二个初值 x_1，x_2，则下一近似 $x_3=\frac{x_1+x_2}{2}$，如果 $f(x_3)=0$ 则 x_3 就是方程介，否则，若 $f(x_i)\cdot f(x_3)<0$，则又可得一近似介 $x_4=\frac{x_i+x_3}{2}$（$i=1$，2），类似地计算下去，直到满足规定的要求。

5. 下面程序片段是否错误的？为什么？

```
real a, b, c, x, y, z;
if x<y then if x>z then a:=b else a:=c;
```