关于算法语言 ALGOL 60 的修正报告

关于算法语言 ALGOL 60 的修正报告*)

Peter Naur 等

算法语言 ALGOL 60 是国际上通用的一种语言。由于它在理论上以及形式表达上的严谨明晰，它无疑是最重要的算法语言之一。

不论是研究、学习 ALGOL 60 本身，还是用算法语言编写源程序，以及为特定的计算机配 ALGOL 60 语言的编译程序，本修正报告都是必不可少的依据。

本修正报告系把原报告和 1962 年罗马会议发表的附件合并而成的一个完整的文件。

提　　要

这个报告是国际算法语言 ALGOL 60 的一个完整的定义性的描述，这个语言适宜于把一大类数值过程表示为一种充分简明的形式以便直接地自动翻译为程序自动计算机的语言。

序言部分对导致制定本语言的正式会议的预备工作做了说明。此外还解释了参考语言、出版语言和硬件表示的概念。

在第一章综述了本语言的基本成分和性质，和解释了用来定义，本语言语法结构的形式记法。

第二章列出了全部基本符号并定义了通称为标识符、数和行等语法单位、进而定义了诸如量和值等重要概念。

第三章解释形成表达式的规则和这些表达式的意义，存在三种不同类型的表达式：算术表达式，布尔（逻辑）表达式和命名表达式。

第四章描述本语言的运算单位，它们通称为语句，基本语句为：赋值语句（求一个式子的值），转向语句（明显地中断语句的执行顺序），空语句和过程语句（调用由过程说明定义的一个封闭过程并执行它）。解释了具有语句特征的更复杂结构的构造，它们包括：条件语句，循环语句，复合语句和分程序。

第五章所定义的单位通称为说明，它用以定义在本语言所描述的过程中出现的单位的不变的性质。

报告的结尾部分是本语言应用的两个详细例子和一个定义的字母索引。

序　　论

背　　景

在 1958 年苏黎世会议上准备的关于算法语言 ALGOL 的初步报告发表以后，引起了对 ALGOL 语言的很大兴趣。

由于 1958 年 11 月在美因兹举行的一次非正式会议的结果，从几个欧洲国家来的约四十位有关人员于 1959 年 2 月在哥本哈根举行了一次实现 ALGOL 的会议。成立了一个“硬件小组”，以便直到在纸带译码方面共同工作，此次会议还决定在哥本哈根计算中心出版由 P. 瑙尔主编的 ALGOL 公报，以作为进一步探讨的论坛，在 1959 年 6 月巴黎 ICIP 会议期间，举行了几次正式的和非正式的会议，这些会议表明，对原来负责拟定语言的小组的意图存在着不同的看法；但同时也表明，对已做的努力有着广泛的支持，作为这些讨论的结果，决定在1960年1月召开一次国际会议，以便改进 ALGOL 语言并准备一份正式报告。1959年 11 月在巴黎约有五十人参加的欧洲 ALGOL 会议上，选出了参加 1960 年 1 月会议的七名欧洲代表，他们

*) 中国科学院数学所计算站陆汝铃、周龙骧重校。最初由耿立大译出，后经许孔时重译收在《算法语言文集》中，本译文又对许孔时的译文重校而成（原文 R. Bauman, M. Feliciano, F. L. Bauer, K. Samelson, Introduction to ALGOL, 97—142）

代表下列组织：法国计算协会，英国计算机协会，应用数学与力学协会以及荷兰计算机协会，这七名代表于1959年12月在美因兹举行了最后的预备会议。

同时，在美国要求任何想建议改进和修正ALGOL的人把他们的意见寄给ACM通讯，并在那里发表。然后这些意见就成为考虑改进ALGOL语言的基础。SHARE和USE这两个组织都建立了ALGOL工作小组，并且这两个组织都在ACM程序设计语言委员会中有代表。ACM委员会于1959年11月在华盛顿开会，考虑了已寄给ACM通讯的关于ALGOL的全部意见，也选出七名代表去参加1960年1月的国际会议，这七名代表于1959年12月在波士顿举行了最后的预备会议。

1960年1月会议

来自丹麦、英国、法国、德国、荷兰、瑞士和美国的十三名代表自1960年1月11日至16日在巴黎举行了会议。

在这次会议之前，彼得、瑙尔根据初步报告和预备会议上的建议提出了一份全新的报告草案，会议接受了这个新形式作为会议报告的基础。然后会议开始工作，以便对报告的每一项取得一致意见。现在的这份报告代表着委员会的共同观点及其讨论后一致同意的意见。

1962年4月会议（M. WOODGER执笔）

由于国际计算中心提供便利和殷勤招待，ALGOL 60的一些作者于1962年4月2日和3日在意大利的罗马举行了一次会议。出席会议的有下列人士：

作　者

包厄尔（F. L. Bauer）
格林（J. Green）
卡茨（C. Katz）
科根（R. Kogon）——代表巴科斯（J. W. Backus）
瑙尔（P. Naur）
萨梅耳桑（K. Samelson）
韦格施太因（J. H. Wegstein）
万维恩格登（A. V. Wijngaarden）
伍哲（M. Woodger）

顾　问

保罗（M. Paul）
弗兰塞沃提（R. Franciotti）
英格曼（P. Z. Ingerman）
塞格谬勒（G. Seegmüller）
尤特曼（R. E. Utman）
兰丁（P. Landin）

观 察 员

博伊耳（W. L. Van. der Poel）——国际信息加工联合会技术委员会2.1. ALGOL工作组的主席。

会议的目的在于改正ALGOL 60报告中已知的错误，并企图消除报告中明显的有二义性之处（或不确切之处），以及在其他方面使ALGOL 60报告更清楚。在这次会议上没有考虑ALGOL 60的各种扩充问题。有关团体在答复ALGOL公报第14期上的意见征询表时所提出的各种修改和澄清的建议都是我们的依据。

本文构成ALGOL 60报告的附件*），它将解决原报告中的若干疑难之处；但并不能解决原报告中产生的全部问题。委员会决定，与其在若干微妙之处冒轻率下结论的危险（这些结论可能产生新的二义性或不确切之处），不如只把他们一致感到能清晰而确切地陈述的那些地方报告出来。

与下述范围有关的问题留待国际信息加工联合会2.1，工作组作进一步的考虑，希望目前关于高级程序设计语言的工作将导致这些问题得到更好的解决：

1. 函数的副作用。
2. 换名的概念。
3. 静态的OWN或动态的OWN.
4. 静态的循环语句或动态的循环语句。
5. 区分和说明之间的矛盾。

在知道国际信息加工联合会成立了ALGOL工作组之后，出席罗马会议的ALGOL 60的作者们认为，关于发展、解说和改善ALGOL语言他们可能担负的任何集体责任今后都将移交给该工作组。

本报告在1962年8月已由国际信息加工联合会关于程序设计语言的第2技术委员会审阅并已得到国际信息加工联合会会议的认可。

和ALGOL的初步报告一样，这里承认三级不同的语言，即参考语言、出版语言和若干种硬件表示。

参 考 语 言

1. 它是委员会的工作语言。
2. 它是起定义作用的语言。

---

*）在本修正报告中把这个附件同ALGOL 60报告合并成一个整体了。

3．字符决定于容易互相了解，并不决定于计算机的限制、译码者的表示法或纯粹的数学表示法。

4．它是编译程序人员的基本参考材料和指南。

5．它是所有硬件表示的指南。

6．它是从出版语言译成任何局部适用的硬件表示的指南。

7．ALGOL 语言本身的主要出版物将使用参考语言。

出版语言

1．出版语言允许按照印刷和书写的习惯而改变参考语言（例如下标、空白、指数，希腊字母）。

2．用它叙述和传递过程。

3．使用的字符在不同的国家中可以是不同的，但应保证和参考语言有唯一的对应。

硬件表示

1．每种硬件表示都是参考语言的一种缩减，因为在标准输入设备上字符的数目是有限的。

2．每种硬件表示使用特定的计算机的一组字符，并且是该计算机的翻译程序所接受的语言。

3．每种硬件表示应附有特殊的一组规则，以便从出版语言或参考语言进行翻译。

为了在参考语言和适合于出版的语言之间进行翻译，特别要提出下列规则作为建议：

参考语言	出版语言
下标括号[ ]	把括号中的行降下来并去掉括号、
乘方 ↑	将指数升上去。
圆括号 ( )	任何形状的圆括号、方括号、花括号
基底 + $_{10}$	把+和它后面的指数升起来，并加上所需的乘号。

参考语言的描述

1．语言的结构

如序论中所说，算法语言有三种不同的表示——参考的，硬件的，和出版的——，以后的描述用参考语言、这就是说，在这个语言内定义的所有对象都用给定的一组符号来表示，只是在符号的选择上和其他两种表示可以不同，对所有三种表示而言结构和内容应当是一样。

算法语言的目的是描述计算过程，描述计算规则时用到的基本概念是熟知的算术表达式（它包含数、变量和函数为其组成部分）。使用算术的构成规则，从那些表达式就构成这个语言中自封的单位（明显的公式），叫作赋值语句。

为表明计算过程的进行情况，要加上某些非算术语句和语句子句，它们可以描述计算语句的选择或迭代重复等。因为对这些语句的作用而言一个语句涉及其他语句是必要的。所以语句可带有标号，一列语句可被括在语句括号 **begin** 和 **end** 之间从而构成一个复合语句。

语句伴有说明，说明本身不是计算指令，但它告诉翻译程序，语句中出现哪些对象以及对象的某些性质，例如作为变量值的数属于那一类，数组的维数，或者甚至是定义一个函数的一组规则。把一列说明及其后的一列语句都括在 **begin** 和 **end** 之间就构成一个分程序，分程序中的每个说明均以此种方式出现并且只对该分程序有效。

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。

下面给出参考语言的语法和语义[1]。

1.1．语法描述的形式

语法将借助于元语言公式[2]来描述，对它的解释最好用一个例子来说明：

〈ab〉::=(|[|〈ab〉(|〈ab〉〈d〉,

包在尖括号〈 〉内的字符序列代表元语言变量，它的值是符号的序列，记号::=和|（后者的意义是“或”）是元语言连接词。公式中的任何记号不是变量或连接词时就表示它自己（或一组与它相类似的记号），公式中记号和变量的并列、记号的并列或变量的并列都意味着它们所表示的序列的并列，因此，上述公式给出形成变量〈ab〉的值的一个递归规则，它指出，〈ab〉可以有值“(”或“[”；或者给定〈ab〉的某个合法值后，〈ab〉另外的值可由在它后面跟上符号“(”或跟上变量〈d〉的某个值而做出，假如〈d〉的值是十进制数字，则〈ab〉的某些值是：

---

1) 每当提到计算的精确度，一般说来尚未详细确定或某个过程的结果有待定义或没有定义时，要按下述意义加以解释：如果附加信息详细说明了所要的精确度和所作的计算的种类，并且对在完成计算时所有可能发生的情况都详细说明了要采取的操作顺序，只在此时才说程序完全确定了一个计算过程。

2) 参看 J. W. Backus, The syntax and semantics of the Proposed international algebraic languagl of the Zurich ACM-GAMM conference. ICIP Paris, June 1959.

[((((37(
(1 2 3 4 5(
(((
[8 6

为了便于研究，选定一些近似地描述了相应变量的性质的字作为区别元语言变量（即出现在尖括号〈 〉中的字符序列，如上例中的ab）的符号，在文中其他地方用到以这种方式出现的字时都要参照对应的语法定义。此外，有的公式不只在一处给出。

定义：

〈空〉::=

（即没有符号的空行）。

2. 基本符号、标识符、数和行、基本概念

参考语言是用下列基本符号构造起来的：〈基本符号〉::=〈字母〉|〈数字〉|〈逻辑值〉|〈定义符〉

2.1 字　　母

〈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

这个字母表可以任意地缩减，或者随意添加任何其他不同的字符（即不与任何数字、逻辑值或定义符相同的字符）而加以扩充。

字母没有单独的意义。它们是用来组成标识符和行[1]的（参看2.4.标识符，2.6.行）

2.2.1. 数　　字

〈数字〉::=0|1|2|3|4|5|6|7|8|9

数字是用以组成数、标识符和行的。

2.2.2. 逻辑值

〈逻辑值〉::=true|false

逻辑值有确定而明显的意义。

2.3. 定义符

〈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〉|〈关系运算符〉|〈逻辑运算符〉|〈顺序运算符〉

〈算术运算符〉::=+|−|×|/|÷|↑

〈关系运算符〉::=<|≤|=|≥|>|≠

〈逻辑运算符〉::=≡|⊃|∨|∧|¬

〈顺序运算符〉::=go to|if|then|else|for|do

〈分隔符〉::=,|.|$_{10}$|:|;|:=|#|step|until|while|Comment

〈括号〉::=(|)|[|]|'|'|begin|end

〈说明符〉::=Own|Boolean|integer|real|array|switch|Procedure

〈分类符〉::=string|label|value

定义符有确定的意义，大部分定义符的意义是明显的；否则，其意义将在后面适当的地方给出。

在参考语言中，象空白或另起一行等印刷上的特点是没有什么含义的。但是，为了容易阅读起见，可以随意地使用它们。

为了在程序的符号间加进一段话，下面关于“注解”的约定成立：

基本符号的序列	等价于
;Comment〈任何序列，但不包括“;”〉;	;
begin Comment〈任何序列，但不包括“;”〉;	begin
end〈任何序列，但不包括end或else或“;”〉;	end

这里等价的意义是：左列所示三种结构中的任何一种在行外出现时，换成右列同一排所示的符号后，对程序的执行没有任何影响，更进一步理解：从左往右读在文中首先遇到的注解结构比该序列中包含的以后的注解结构要优先替换。

2.4. 标　识　符

2.4.1. 语　　法

〈标识符〉::=〈字母〉|〈标识符〉〈字母〉|〈标识符〉〈数字〉

2.4.2. 例

q

soup

v17a

a34KTMNS

MARILYN

2.4.3. 语　　义

标识符没有固有的意义，只用于标识简单变量、数组、标号、开关和过程，可以随意地选用它们（但是请参看3.2.4标准函数）。

不能用同一标识符表示两个不同的量，除非程序的“说明”确定了这两个量有不相交的作用域（参看2.7.量、种类和作用域，5.说明）。

2.5. 数

2.5.1. 语　　法

〈无符号整数〉::=〈数字〉|〈无符号整数〉〈数字〉

---

[1] 应当特别注意，在整个参考语言中粗黑体字用于定义独立的基本符号（参看2.2.2和2.3）。这些粗黑体字被认为与组成它们的单独字母没有关系。在本报告中粗黑体字将不作别用，参考语言中用符号␣表示空白，由于印刷上的原因，这里用符号#代替。

〈整数〉::=〈无符号整数〉|＋〈无符号整数〉|－〈无符号整数〉

〈十进制小数〉::=·〈无符号整数〉

〈指数部分〉::=$_{10}$〈整数〉

〈十进制数〉::=〈无符号整数〉|〈十进制小数〉|〈无符号整数〉〈十进制小数〉

〈无符号数〉::=〈十进制数〉|〈指数部分〉|〈十进制数〉〈指数部分〉

〈数〉::=〈无符号数〉|＋〈无符号数〉|－〈无符号数〉

2.5.2. 例

0	－200.084	$-083_{10}-02$
177	＋07.43$_{10}$8	$-_{10}7$
.5384	9.34$_{10}$＋10	$_{10}-4$
＋0.7300	$2_{10}-4$	$+_{10}+5$

2.5.3. 语 义

十进制数有其通常的意义，指数部分是一个标度因子，表示为10的整数次方。

2.5.4. 类 型

整数是 integer 型，所有其他的数都是 real 型（参看 5.1. 类型说明）。

2.6. 行

2.6.1. 语 法

〈正则行〉::=〈基本符号的任一序列，但不包括'或'〉|〈空〉

〈开行〉::=〈正则行〉|'〈开行〉'|〈开行〉〈开行〉

〈行〉::='〈开行〉'

2.6.2. 例

'5K, , －'[[['∧=/：' Tt"

'.. This ♯is ♯a ♯ 'string"

2.6.3. 语 义

为使这个语言能处理基本符号的任一序列，引入行引号'和'。符号♯表示空白，在行外它没有意义。

行用作过程的实在参数（参看3.2. 函数命名符和 4.7. 过程语句）。

2.7. 量、种类和作用域

下面诸量的种类是有区别的：简单变量、数组、标号、开关和过程。

量的作用域是一组语句和表达式，在这组语句和表达式内该量的标识符的说明是有效的。关于标号的作用域参看第 4.1.3 节。

2.8. 值和类型

值是一些数的有序集合（特殊情况：单独一个数），是一些逻辑值的有序集合（特殊情况：单独一个逻辑值）或者是一个标号。

我们说，某些语法单位有值。一般说来，在执行程序时这些值是变动的。表达式及其组成部分的值在第 3 节中定义。数组标识符的值是相应下标变量数组的值的有序集合（参看 3.1.4.1）。

各种"类型"（integer, real, Boolean）基本上表示了值的性质。语法单位的类型，就是它们的值的类型。

3. 表 达 式

在这个语言中，描写算法过程的程序，其原始组成部分是算术表达式、布尔表达式和命名表达式。这些表达式的组成部分，除了某些定义符之外，是逻辑值、数、变量、函数命名符以及初等的算术运算符、关系运算符、逻辑运算符和顺序运算符。因为在变量和函数命名符二者的语法定义中都含有表达式，所以表达式及其组成部分的定义必须都是递归的。

〈表达式〉::=〈算术表达式〉|〈布尔表达式〉|〈命名表达式〉

3.1. 变 量

3.1.1. 语 法

〈变量标识符〉::=〈标识符〉

〈简单变量〉::=〈变量标识符〉

〈下标表达式〉::=〈算术表达式〉

〈下标表〉::=〈下标表达式〉|〈下标表〉,〈下标表达式〉

〈数组标识符〉::=〈标识符〉

〈下标变量〉::=〈数组标识符〉[〈下标表〉]

〈变量〉::=〈简单变量〉|〈下标变量〉

3.1.2 例

epsilon

det A

a 17

Q [7,2]

x[sin Cn×pi/2), Q[3, h, 4]]

3.1.3. 语 义

变量是给一个值起的名字。可用这个值在表达式中形成其它的值。可用赋值语句（参看 4.2）任意改变这个值。一个特定变量的值的类型由变量本身的说明（参看 5.1. 类型说明）确定，或由相应数组标识符的说明（参看 5.2. 数组说明）确定。

3.1.4. 下 标

3.1.4.1 下标变量是给多维数组（参看 5.2. 数组说明）分量的值起的名字。下标表中的每个算术表达式占有下标变量的一个下标位置，叫作一个下标，整个下标表包在下标括号[ ]中，下标变量所对应的数组分量由其下标的实际数值确定（参看 3.3. 算术表达式）。

3.1.4.2. 每个下标位置的作用类似于一个 integer 型的变量，术下标的值可以理解为等价于给这个虚构变量赋值（参看4.2.4）。下标变量的值只当下标表达式的值是在数组的下标界（参看5.2.数组说明）之内时才有定义。

3.2. 函数命名符

3.2.1. 语　法

〈过程标识符〉::=〈标识符〉

〈实在参数〉::=〈行〉|〈表达式〉|〈数组标识符〉|〈开关标识符〉|〈过程标识符〉

〈字母行〉::=〈字母〉|〈字母行〉〈字母〉

〈参数定义符〉::=, 1) 〈字母行〉:(

〈实在参数表〉::=〈实在参数〉|〈实在参数表〉〈参数定义符〉〈实在参数〉

〈实在参数部分〉::=〈空〉|(〈实在参数表〉)

〈函数命名符〉::=〈过程标识符〉〈实在参数部分〉

3.2.2. 例

```
sin(a—b)
J (v+s, n)
R
S(s—5)Temperature:(T) Pressure:(P)
Compile( ':=' )Stack:(Q)
```

3.2.3. 语　义

函数命名符定出一个数值或一个逻辑值，此值是对固定的实在参数组应用给定的一些规则而得的结果，这些规则由过程说明确定。（参看5.4.过程说明）。详细说明实在参数的规则参看 4.7. 过程语句。并非每个过程说明都定出函数命名符的值。

3.2.4. 标准函数

某些标识符要保留给数学分析中的标准函数使用，这些函数将表示为过程。建议保留的标识符表中应包括：

abs(E)	表达式 E 之值的模（绝对值），
sign(E)	E之值的符号（当E>0时是+1，当E=0时是0，当E<0时是—1）。
sqrt(E)	E之值的平方根。
sin(E)	E之值的正弦。
cos(E)	E之值的余弦。
arctan(E)	E之值的反正切的主值。
ln(E)	E之值的自然对数。
exp(E)	E之值的指数函数 (e^E)。

不言而喻，这些函数对 real 型变元和 integer 型变元二者进行运算时是没有区别的；它们得到的值是 real 型，只有 sign(E)除外，它的值是 integer 型。在本算法语言中，不必明确说明（参看 5. 说明）即可使用这些函数。

3.2.5. 转换函数

不言而喻，可以在任意一对量和任意一对表达式之间定义转换函数。建议在标准函数中有这样一个函数，即

Entier(E),

它将实数型的表达式“转换”为整数型的表达式，并且它的值就是不大于E之值的最大整数。

3.3. 算术表达式

3.3.1. 语　法

〈加法运算符〉::=+|−

〈乘法运算符〉::=×|/|÷

〈初等量〉::=〈无符号数〉|〈变量〉|〈函数命名符〉|(〈算术表达式〉)

〈因式〉::=〈初等量〉|〈因式〉↑〈初等量〉

〈项〉::=〈因式〉|〈项〉〈乘法运算符〉〈因式〉

〈简单算术表达式〉::=〈项〉|〈加法运算符〉〈项〉|〈简单算术表达式〉〈加法运算符〉〈项〉

〈如果子句〉::=if〈布尔表达式〉then

〈算术表达式〉::=〈简单算术表达式〉|〈如果子句〉〈简单算术表达式〉else〈算术表达式〉

3.3.2. 例

初等量：

```
7.394₁₀−8.
sum
w[i+2, 8]
cos(y+z×3)
(a−3/y+vu↑8)
```

因式：

```
omega
sum↑cos(y+z×3)
7.394₁₀−8↑W[i+2, 8]↑(a−3/y+vu
    ↑8)
```

项：

```
U
omega×sum↑cos(y+z×3)/7.394₁₀−8↑
    W[i+2, 8]↑(a−3/y+vu↑8)
```

简单算术表达式：

```
U−Yu+ omega×sum↑cos (y+z×3)/
    7.394₁₀−8↑W[i+2,8]↑(a−3/y+
    vu↑8)
```

算术表达式：

```
W×u−Q(S+Cu)↑2
if q>0 then S+3×Q/A else 2×S+
    3×q
if a<0 then U+V else if a×b>17
    then U/V else if k≠y then
```

V/U else 0

a×sin(omega×t)

$0.57_{10}12$×a[N×(N−1)/2, 0]

(A×arctan(y)+Z)↑(7+Q)

if q then n-1 else n

if a<0 then A/B else if b=0 then B/A else Z

3.3.3. 语　义

算术表达式是计算数值的规则。若是简单算术表达式，则对表达式中各初等量的实际数值执行指定的算术运算就可得到这个值，这在下面3.3.4中有详细的解释。若初等量是数，其实际数值是显然的。对变量而言其实际数值是当前值（动态意义下最近赋值的），对函数命名符而言其实际数值是对表达式中过程参数的当前值应用定义此过程的计算规则（参看第5.4.4节函数命名符的值）而得到的值。最后，对包在圆括号内的算术表达式而言，其值应当经过递归分析用其它三类初等量的值表示出来。

在更一般的含有如果子句的算术表达式中，根据布尔表达式的实际值（参看3.4.布尔表达式）从若干简单算术表达式中选出一个来。此选择进行如下：一个一个地依次由左向右计算如果子句中的布尔表达式直至找到一个值是 true 的布尔表达式为止。于是算术表达式的值就是此布尔表达式后面第一个算术表达式（这里指的是在这个位置上找到的最大算术表达式）的值。结构：

else 〈简单算术表达式〉

等价于结构：

else if true then 〈简单算术表达式〉

3.3.4. 运算符和类型

除如果子句中的布尔表达式外，简单算术表达式的组成部分应当是 real 型或 integer 型(参看5.1.类型说明)。基本运算符的意义和由它们导出的表达式的类型由下述规则给出：

3.3.4.1. 运算符＋、－和×有通常的意义（加、减和乘）。如果两个运算对象都是 integer 型，则表达式的类型也是 integer 型；否则是 real 型。

3.3.4.2. 运算〈项〉/〈因式〉和〈项〉÷〈因式〉都表示除法，可理解为项乘上因式的倒数，这里应注意到优先规则（参看3.3.5）。例如

a/b×7/(p−q)×V/S

表示

$((((a\times(b^{-1}))\times 7)\times((p-q)^{-1}))\times v)\times(S^{-1})$。

对 real 型和 iteger 型的所有四种组合而言，运算符 / 都有定义，并且在任何情况下得到的结果都是 real 型。运算符÷只当两个运算对象都是 integer 型时才有定义，并且所得的结果是 integer 型，其数学定义如下：

a÷b=sign(a/b)×entier(abs(a/b))

（参看3.2.4和3.2.5节）。

3.3.4.3. 运算〈因式〉↑〈初等量〉表示乘方，这里因式是底，而初等量是指数。例如

2↑n↑k表示　$(2^n)^k$

而

2↑(n↑m)　表示　$2^{(n^m)}$

用 i 表示 integer 型的数，用 r 表示real 型的数，用 a 表示 integer 型或 real 型的数，其结果由下列规则给出：

a↑i，若 i>0，它是 a×a×…×a（i 次），类型与 a 相同。

若 i=0 且 a≠0，它是 1，类型与 a 相同。

若 i=0 且a=0，它没有定义。

若 i<0 且 a≠0，它是1/(a×a×…×a)（分母有 −i 个因数），类型是 real。

若 i<0 且 a=0，它没有定义。

a↑r，若 a>0，它是 exp (r×ln(a))，类型是 real。

若 a=0 且 r>0，它是 0.0，类型是 real。

若 a=0 且 r≦0，它没有定义。

若 a<0，对任何 r 都没有定义。

3.3.5. 运算符的优先顺序

在一个表达式中运算的顺序一般是由左向右，并有下列附加规则：

3.3.5.1. 按照3.3.1中给出的语法，下述优先规则成立：

第一：　↑

第二：　×　/　÷

第三：　＋　－

3.3.5.2. 在左圆括号和对应的右圆括号之间的表达式按它本身计算，它的值在以后的计算中要用到。因此，在表达式中总能用适当地加圆括号的办法列出所要的进行运算的顺序。

3.3.6. real 量的计算

应在数值分析的意义下来解释 real 型的数和变量，即把它们定义为本来只有有限准确度的对象。类似地，在任何算术表达式中，与数学上定义的结果相比产生有限误差的可能性是显然可见的。将不详细说明什么叫精确的计算，但实际上我们了解，不同的硬件表示可以对算术表达式进行不同的计算。对这些差别的可能后果的控制，应当用数值分析的方法来实现。这种控制应该作为要描述的过程的一部分，并因此要用这个语言本身的术语来表示。

3.4. 布尔表达式

3.4.1. 语　法

〈关系运算符〉::=<|≦|=|≧|>|≠

〈关系式〉::=〈简单算术表达式〉〈关系运算符〉〈简单算术表达式〉

〈布尔初等量〉::=〈逻辑值〉|〈变量〉|〈函数命名符〉|〈关系式〉|(〈布尔表达式〉)

〈布尔二次量〉::=〈布尔初等量〉|¬〈布尔初等量〉

〈布尔因式〉::=〈布尔二次量〉|〈布尔因式〉∧〈布尔二次量〉

〈布尔项〉::=〈布尔因式〉|〈布尔项〉∨〈布尔因式〉

〈蕴涵式〉::=〈布尔项〉|〈蕴涵式〉⊃〈布尔项〉

〈简单布尔量〉::=〈蕴涵式〉|〈简单布尔量〉≡〈蕴涵式〉

〈布尔表达式〉::=〈简单布尔量〉|〈如果子句〉〈简单布尔量〉else〈布尔表达式〉

3.4.2. 例

x=−2

Y>V∨z<q

a+b>−5∧z−　>1↑2

p∧q∨x≠y

g≡¬a∧b∧¬c∨d∨e⊃¬f

if k<1 then S>W else h≦c

if if if a then b else c then d else f then g else h<k

3.4.3. 语　义

布尔表达式是计算逻辑值的规则。计算原理和3.3.3中给出的算术表达式的计算原理完全类似。

3.4.4. 类　型

变量和函数命名符用作布尔初等量时，应当说明是 Boolean 型（参看 5.1. 类型说明和 5.4.4. 函数命名符的值）。

3.4.5. 运 算 符

当所包含的表达式满足对应的关系时，关系式的值是 true，否则是 false。

逻辑运算符 ¬（非）、∧（和）、∨（或）、⊃（蕴涵）和三（等价）的意义由下表给出。

b1	false	false	true	true
b2	false	true	faise	true
.....				
¬b1	true	true	false	false
b1∧b2	false	false	false	true
b1∨b2	false	true	true	true
b1⊃b2	true	true	false	true
b1≡b2	true	false	false	true

3.4.6. 运算符的优先顺序

在一个表达式内，运算的顺序一般由左向右，并有下列附加规则：

3.4.6.1. 按照 3.4.1 中给出的语法，下列优先规则成立：

第一：按照 3.3.5 计算算术表达式

第二：<≦=≧>≠

第三：¬

第四：∧

第五：∨

第六：⊃

第七：≡

3.4.6.2. 对使用圆括号的解释与 3.3.5.2 的意思相同。

3.5. 命名表达式

3.5.1. 语　法

〈标号〉::=〈标识符〉|〈无符号整数〉

〈开关标识符〉::=〈标识符〉

〈开关命名符〉::=〈开关标识符〉[〈下标表达式〉]

〈简单命名表达式〉::=〈标号〉|〈开关命名符〉|(〈命名表达式〉).

〈命名表达式〉::=〈简单命名表达式〉|〈如果子句〉〈简单命名表达式〉else〈命名表达式〉

3.5.2. 例

17

p9

Choose[n−1]

Town[if y<0 then N else N+1]

if Ab<d then 17 else q [if W≦0 then 2 else n]

3.5.3. 语　义

命名表达式是求语句标号的规则（参看 4. 语句）。计算原理仍然与算术表达式的计算原理（参看 3.3.3）完全类似。在一般情况下，如果子句中的布尔表达式将选出一个简单命名表达式。如果这是一个标号，则所要的结果就已经得到了。开关命名符要参照对应的开关说明（参看 5.3. 开关说明），并且用开关命名符中下标表达式的实际数值在开关说明所列举的命名表达式中从左向右数，选出一个命名表达式。因为这样选出来的命名表达式可以又是一个开关命名符，所以这个计算显然是一个递归过程。

3.5.4. 下标表达式

下标表达式的计算类似于下标变量的计算（参看 3.1.4.2），开关命名符的值只当下标表达式取 1，2，3，…，n 中之一为自己的值时才有定义，这里 n 是开关表中入口个数。

3.5.5. 作标号用的无符号整数

无符号整数用作标号时有这样的性质，即开头的

零不影响它的意义，例如00217和217表示同一个标号。

4. 语　句

这个语言中的运算单位叫作语句，它们通常按所写的次序连续执行，但此运算序列可被转向语句（它明显地定出自己的后继）所中断或可被条件语句（它可使某些语句被跳过）所缩短。

为了能定义特定的动态顺序，语句可附以标号。

因为语句的序列可以组成复合语句和分程序，所以语句的定义必然是递归的，因为第5节中所描述的说明也是语法结构的基本部分，所以语句的语法定义必须假定说明已经定义好了。

4.1. 复合语句和分程序

4.1.1. 语　法

〈无标号基本语句〉::=〈赋值语句〉|〈转向语句〉|〈空语句〉|〈过程语句〉

〈基本语句〉::=〈无标号基本语句〉|〈标号〉:〈基本语句〉

〈无条件语句〉::=〈基本语句〉|〈复合语句〉|〈分程序〉

〈语句〉::=〈无条件语句〉|〈条件语句〉|〈循环语句〉

〈复合尾部〉::=〈语句〉end|〈语句〉;〈复合尾部〉

〈分程序首部〉::=begin〈说明〉|〈分程序首部〉;〈说明〉

〈无标号复合语句〉::=begin〈复合尾部〉

〈无标号分程序〉::=〈分程序首部〉;〈复合尾部〉

〈复合语句〉::=〈无标号复合语句〉|〈标号〉:〈复合语句〉

〈分程序〉::=〈无标号分程序〉|〈标号〉:〈分程序〉

〈程序〉::=〈分程序〉|〈复合语句〉

此语法可解说如下：用字母S、D和L分别表示任意的语句、说明和标号，基本语法单位的形式为：

复合语句：

L: L:…begin S; S; …S; S end

分程序：

L: L:…begin D; D; …D; S; S; … S; S end

应当记住，每个语句S又可以是一个完整的复合语句或分程序。

4.1.2. 例

基本语句：

```
a:=p+q
go to Naples
START: CONTINUE: W:=7.993
```

复合语句：

```
begin x:=0;
    for y:=1 step 1 until n do x:
        =x+A[y];
    if x>q then go to STOP else
        if x>W-2 then go to S;
    AW: St: W:=x+b of end
```

分程序：

```
Q: begin integer i, k; real W;
    for i:=1 step 1 until m do
    for k:=i+1 step 1 until m do
    begin W:=A[i, k];
        A[i, k]:=A[k, i];
        A[k,i]:=W end for i and k
    end block Q
```

4.1.3. 语义

每个分程序自动地引入新的一层用语，其实现如下：可用适当的说明（参看5.说明）确定分程序内的任何标识符对该分程序而言是局部的，这就是说：(a) 分程序内此标识符代表的对象在分程序外不存在；(b) 不可能在分程序内使用分程序外此标识符代表的任何对象。

分程序内的标识符（除去那些代表标号的标识符）对此分程序未作说明者，对此分程序而言是非局部的，即它在分程序内和该分程序的直接外层中代表同一个对象，冒号隔开标号和语句，即标号标出该语句，标号的作用好似在“最小的包含分程序”的首部中有一个说明，所谓“最小的包含分程序”就是其括号begin和end包含着该语句的那个最小的分程序。此处应当认为过程体是好象被包在begin和end之间，并且要把过程体作为一个分程序来处理。

因为分程序中的一个语句自身可能又是一个分程序，所以应把对分程序而言局部和非局部的概念理解为递归的。因此当分程序A是分程序B的一个语句时，对A而言是非局部的标识符对B而言可能是非局部的也可能不是非局部的。

4.2. 赋值语句

4.2.1. 语法

〈左部〉::=〈变量〉:=|〈过程标识符〉:=

〈左部表〉::=〈左部〉|〈左部表〉〈左部〉

〈赋值语句〉::=〈左部表〉〈算术表达式〉|〈左部表〉〈布尔表达式〉

4.2.2. 例

```
s:=p[o]:=n:=n+1+s
n:=n+1
A:=B/C-v-g×s
s[v,k+2]:=3-arctan(s×zeta)
V:=Q>Y∧Z
```

4.2.3. 语义

赋值语句用于把一个表达式的值赋给一个或多个变量或过程标识符。只是在定义函数命名符之值的过程体内（参看第5.4.4节），才可能发生对过程标识符的赋值。在一般情况下，此过程被理解为采取下述三个步骤：

4.2.3.1. 由左向右依次计算左部变量中出现的任何下标表达式。

4.2.3.2. 计算语句中表达式的值。

4.2.3.3. 把表达式的值赋给所有的左部变量，左部变量中的任何下标表达式应取4.2.3.1那一步已计算出来的值。

4.2.4. 类型

左部表中所有变量和过程标识符的类型应当是相同的。如果这个类型是 Boolean，则表达式应当同样地是 Boolean。如果类型是 real 或 integer，则表达式应当是算术表达式。如果算术表达式的类型与变量和过程标识符的类型不同，则不言而喻地自动引入适当的转换函数。为了从 real 型转换到 integer 型，转换函数应产生一个等价于

entier (E+0.5)

的结果，这里 E 是表达式的值。

过程标识符的类型由说明符给出，此说明符作为相应的过程说明的第一个符号出现（参看第5.4.4节）。

4.3. 转向语句

4.3.1. 语法

〈转向语句〉::=go to 〈命名表达式〉

4.3.2. 例

```
go to 8
go to exit[n+1]
go to Town[if y<0 then N else N+1]
go to if Ab<C then 17 else q[if W<0 then 2 else n]
```

4.3.3. 语义

转向语句中断已写好的语句所确定的正常运算顺序，它用命名表达式的值明显地定出它的后继。因此，下一个要执行的语句是以此值为标号的语句。

4.3.4. 限制

因为标号固定地是局部的，所以没有转向语句能从外面进入分程序，但转向语句可以从外面进入一个复合语句。

4.3.5. 转向未定义的开关命名符

如果命名表达式是值没有定义的开关命名符，则转向语句等价于空语句。

4.4. 空语句

4.4.1. 语法

〈空语句〉::=〈空〉

4.4.2. 例

```
L:
begin…; John:end
```

4.4.3. 语义

空语句不进行运算。它可用于放置一个标号。

4.5. 条件语句

4.5.1. 语法

〈如果子句〉::=if 〈布尔表达式〉 then

〈无条件语句〉::=〈基本语句〉|〈复合语句〉|〈分程序〉

〈如果语句〉::=〈如果子句〉〈无条件语句〉

〈条件语句〉::=〈如果语句〉|〈如果语句〉 else 〈语句〉|〈如果子句〉〈循环语句〉|〈标号〉:〈条件语句〉

4.5.2. 例

```
if x>0 then n:=n+1
if v>u then V:q:=n+m else go to R
if s<0∨p≦Q then AA:begin if q<v then a:=v/s else y:=2×a end else if v>s then a:=v-q else if v>s-1 then go to S
```

4.5.3. 语义

条件语句根据特定的布尔表达式的当前值去执行某些语句或跳过某些语句。

4.5.3.1. 如果语句。若如果子句中布尔表达式为真，则执行如果语句中的无条件语句；否则就跳过它而继续执行下一个语句。

4.5.3.2. 条件语句。按照语法，条件语句有两种可能的不同的形式，今解说如下：

```
if B1 then S1 else if B2 then S2 else S3; S4
```

和

```
if B1 then S1 else if B2 then S2 else if B3 then S3; S4
```

这里 B1 到 B3 是布尔表达式，而 S1 到 S3 是无条件语句。S4 是整个条件语句后面的语句。

条件语句的执行情况可描述如下：逐个地依次从左向右计算如果子句中的布尔表达式，直至找到一个值是 true 的布尔表达式为止，然后执行这个布尔表达式后面的无条件语句，除非这个语句明显地定出了自己的后继，下一个要执行的语句将是S4，即整个条件语句后面的这个语句，因此，可以说定义符 else

的作用是，它定出它前面的语句的后继就是整个条件语句后面的这个语句。

结构

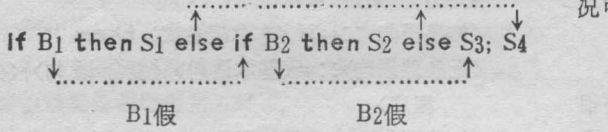
else 〈无条件语句〉

等价于

else if true then 〈无条件语句〉

若如果子句中的布尔表达式没有一个是真，则整个条件语句的作用等价于空语句。

为了做进一步的解说，下面的图可能是有用的：



4.5.4. 转向进入条件语句

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。

4.6. 循环语句

4.6.1. 语法

〈循环表元素〉::=〈算术表达式〉|〈算术表达式〉 **step** 〈算术表达式〉 **until** 〈算术表达式〉|〈算术表达式〉 **while** 〈布尔表达式〉

〈循环表〉::=〈循环表元素〉|〈循环表〉，〈循环表元素〉

〈循环子句〉::=**for** 〈变量〉:=〈循环表〉 **do**

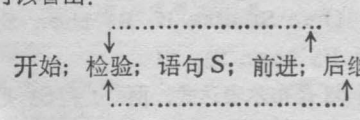
〈循环语句〉::=〈循环子句〉〈语句〉|〈标号〉:〈循环语句〉

4.6.2. 例

```
for q:=1 step S until n do A[q]:
    =B[q]
for k:=1, V1×2 while V1<N do
for j:=I+G, L, 1 step 1 until N,
    C+D do A[k,j]:=B[k,j]
```

4.6.3. 语义

循环子句使它后面的语句S不执行或执行多次。此外，它对自己的控制变量完成一系列赋值。这过程从下图可以看出：



在图中“开始”意味着完成循环子句的第一次赋值；“前进”意味着完成循环子句的下一个赋值；“检验”判断最后一个赋值是否已经做了，如果已经做了，就继续去执行循环语句的后继，如果还没有做，则执行循环子句后面的语句。

4.6.4. 循环表元素

循环表给出求值规则，这些值相继赋给控制变量，从循环表元素中按所写的顺序依次取这些元素就得到这个值的序列。由三种循环表元素中的每一种产生值的序列以及相应地执行语句S，其规则如下：

4.6.4.1 算术表达式。这种元素给出一个值，它就是相应地执行语句S之前刚刚算出来的已知算术表达式的值。

4.6.4.2. 步长型元素。形如A **step** B **until** C的元素，这里A，B和C是算术表达式。其执行情况可用附加的ALGOL语句极简明地描述如下：

```
V:=A;
L1: if (V−C)×sign (B)>0 then go to
        element exhausted;
    statement S;
    V:=V+B;
    go to L1;
```

这里V是循环子句的控制变量，元素用完（element exhausted）指出应按循环表中下一元素进行计算，或者如果此步长型元素是循环表的最后元素，就去执行程序中的下一个语句。

4.6.4.3. 当型元素。它的执行受循环表中形如E **while** F的元素的控制，这里E是算术表达式而F是布尔表达式，其执行情况可用附加的ALGOL语句极简明地描述如下：

```
L3:V:=E;
    if¬F then go to element exhausted;
    statement S;
    go to L3;
```

这里的记号和上面4.6.4.2的相同。

4.6.5. 在出口处控制变量的值

使用转向语句从语句S（假设S是复合语句）中出去时，在出口处控制变量的值和刚刚执行转向语句前控制变量的值相同。

另一方面，如果是因循环表已用完而转出的话，转出后控制变量的值没有定义。

4.6.6. 转向进入循环语句

在循环语句之外，用转向语句转向循环语句内的标号时，其作用没有定义。

4.7. 过程语句

4.7.1. 语法

〈实在参数〉::=〈行〉|〈表达式〉|〈数组标识符〉|〈开关标识符〉|〈过程标识符〉

〈字母行〉::=〈字母〉|〈字母行〉〈字母〉

〈参数定义符〉::=，|）〈字母行〉：（

〈实在参数表〉::=〈实在参数〉|〈实在参数表〉〈参数定义符〉〈实在参数〉

〈实在参数部分〉::=〈空〉|（〈实在参数表〉）

〈过程语句〉::=〈过程标识符〉〈实在参数部分〉

4.7.2. 例

Spur (A) Order : (7) Result to:(V)

Transpose (W,v+1)

Ahsmax (A,N,M,Yy,I,K)

Innerproduct(A[t,P,u], B[P],10, P, Y)

这些例子对应于5.4.2中给出的例子。

4.7.3. 语义

过程语句用于要求执行过程体（参看5.4.过程说明）。过程体是用ALGOL写出的语句，其执行的结果等价于在程序中完成下列运算的结果（在执行过程语句时）：

4.7.3.1. 赋值（代入值）

过程说明导引中的形式参数，凡在值部分中标出者，都应赋以相应的实在参数的值（参看2.8值和类型），这些赋值被认为显然是在进入过程体之前完成的，其作用是：好象造出一个包含此过程体的附加分程序，在此附加分程序中对所有局部变量（所谓局部系对此虚构的分程序而言）进行赋值，这些变量的类型在相应的区分部分中给出（参看第5.4.5节）。因此，赋值的变量被认为对过程体而言是非局部的，但对此虚构的分程序而言是局部的(参看第5.4.3.节)。

4.7.3.2. 换名（代入名字）

未列在值表中的任何形式参数在整个过程体中要换成对应的实在参数（无论何处当语法上可能时要把后者括在括号内以后再换）。在此过程中加进来的标识符与过程体中原有的其它标识符之间可能有冲突，但在对所包含的形式或局部标识符作适当的一系列改变后可以避免。

4.7.3.3. 过程体的代换与执行

最后将上面修改过的过程体嵌在过程语句的位置上并执行它。如果此过程在过程体的任何非局部量的作用域外的某处被调用，则代换过程体时加进来的那些标识符和其说明在过程语句或函数命名符处有效的那些标识符之间的矛盾，经过对后者作适当的一系列改变后可以避免。

4.7.4. 实在与形式的对应

过程语句的实在参数与过程导引的形式参数之间的对应关系建立如下：过程语句的实在参数表和过程说明导引的形式参数表应有相同数目的项，按同一次序取这两个表中的项就得到这个对应关系。

4.7.5. 一些限制

为了定义一个过程语句，显然必须的是，4.7.3.1和4.7.3.2中定义的施于过程体的运算导出一个正确的ALGOL语句。

这就在任何过程语句上加了限制，即每个实在参数的种类和类型与对应的形式参数的种类和类型要一致。这个一般规则的某些重要的特殊情况如下：

4.7.5.1. 如果一个行在过程语句或函数命名符中用作实在参数，并且定义此过程语句或函数命名符的过程体是一个ALGOL 60语句（与非ALGOL的代码相反，参看第4.7.8节），则这个行只能在此过程体内作为更进一层调用别的过程的实在参数使用。归根到底，行只能在非ALGOL的代码所表示的过程体中使用。

4.7.5.2. 过程体内赋值语句中作为左部变量出现的不属于值部分的形式参数，只能对应于是变量的实在参数（表达式的特殊情况）。

4.7.5.3. 过程体内用作数组标识符的形式参数只能对应于是同维数组的数组标识符的实在参数。此外如果该形式参数是要赋值的，则经此赋值得出的局部数组与实在数组要有相同的下标界。

4.7.5.4. 要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，因为后三者不具有值（形式参数部分为空（参看5.4.1）且定出函数命名符之值（参看5.4.4）的过程说明的过程标识符是例外。这个过程标识符本质上是一个完整的表达式）。

4.7.5.5. 任何形式参数对与它对应的实在参数在类型上可以有所限制（这些限制可以通过过程导引中的区分部分给出，或者可以不给出）。在过程语句中这些限制显然应被遵守。

4.7.7. 参数定义符[*]

所有的参数定义符被认为都是等价的。在过程语句中用到的参数定义符和过程导引中用到的那些参数定义符之间，除了它们的数目要相同之外不要求有对应关系。因此使用精心构造的参数定义符所传达的信息是完全多余的。

4.7.8. 用代码表示的过程体

当过程语句要求的过程说明的过程体是用非ALGOL的代码表示时，则加在此过程语句上的限制显然只能从所用代码的特点和使用者的意图来推求，因此就超出了参考语言的范围。

5. 说　明

说明用来确定程序中用到的诸量的某些性质，并且把这些量与标识符连系起来。一个标识符的说明对一个分程序有效。在此分程序之外这个特定的标识符可用于其它目的（参看4.1.3）。

---

[*]译注　原修改报告的4.7.6.节在62年4月罗马会议的文件中删去

动态地说其含义如下：在进入分程序时（通过begin，因为里面的标号是局部的，所以从外面达不到），在此分程序中说明的所有标识符，其意义，就由相应说明的性质决定。如果这些标识符已被外面的其它说明所确定，它们就暂时得到新的意义。另一方面，对此分程序未作说明的标识符仍保持其原来的意义。

在离开分程序时（通过end，或者用一个转向语句），对此分程序已作说明的所有标识符失去其局部的意义。

说明可用附加的说明符 own 加以注记，被注记的量叫做固有量。这有下述作用：当重新进入分程序时，固有量的值和上次离开时它们的值相同，而被说明的变量中未注明是固有量的它们的值就没有定义。除了标号和过程说明的形式参数以及作标准函数用的那些可能的例外（参看3.2.4和3.2.5），程序的所有标识符应被说明。没有标识符在任何一个分程序首部中可被说明一次以上。

语法：

〈说明〉::=〈类型说明〉|〈数组说明〉|〈开关说明〉|〈过程说明〉

5.1. 类型说明

5.1.1. 语法

〈类型表〉::=〈简单变量〉|〈简单变量〉,〈类型表〉

〈类型〉::=real | integer | Boolean

〈局部或固有类型〉::=〈类型〉|own〈类型〉

〈类型说明〉::=〈局部或固有类型〉〈类型表〉

5.1.2. 例

```
integer  p, q, s
own Boolean  Acryl, n
```

5.1.3. 语义

类型说明用于说明某些标识符，以表示给定类型的简单变量。被说明为实（real）的变量只可取正值或负值（包括零）。被说明为整（integer）的变量只可取正整数值或负整数值（包括零）。被说明为布尔（Boolean）的变量只可取值 true 和 false。

在算术表达式中，任何能由被说明为实的变量所占有的位置，都可由被说明为整的变量所占用。

关于 own 的语义，请看上面第5节的第四段。

5.2. 数组说明

5.2.1. 语法

〈下界〉::=〈算术表达式〉

〈上界〉::=〈算术表达式〉

〈界偶〉::=〈下界〉:〈上界〉

〈界偶表〉::=〈界偶〉|〈界偶表〉,〈界偶〉

〈数组段〉::=〈数组标识符〉[〈界偶表〉]|〈数组标识符〉,〈数组段〉

〈数组表〉::=〈数组段〉|〈数组表〉,〈数组段〉

〈数组说明〉::=array〈数组表〉|〈局部或固有类型〉array〈数组表〉

5.2.2. 例

```
array  a, b, c [7:n, 2:m], s [-2:10]
own integer array A [if c<0 then 2 else 1:20]
real array q [-7:-1]
```

5.2.3. 语义

数组说明用于说明一个或几个标识符，以表示下标变量的多维数组，并给出数组的维数、下标的界和变量的类型。

5.2.3.1. 下标界。任何数组的下标界是在此数组的标识符后第一对下标方括号中以界偶表的形式给出的。表中每项给出一个下标的下界和上界，其形式为两个算术表达式中间用定义符“:”隔开，界偶表从左向右依次给出所有下标的界。

5.2.3.2. 维数。维数就是界偶表中项的数目。

5.2.3.3. 类型。在一个说明中说明的所有数组，都取该说明规定的同一类型，如果没有给出类型说明符，就认为类型是 real。

5.2.4. 下、上界表达式

5.2.4.1. 计算该表达式时，使用与计算下标表达式相同的方法（参看3.1.4.2）。

5.2.4.2. 表达式只能依赖于那些变量和过程，它们对数组说明有效的那个分程序而言是非局部的。因此，程序最外一层的分程序只可使用界是常数的数组说明。

5.2.4.3. 只当所有下标上界的值不小于对应的那些下标下界的值时，一个数组才有定义。

5.2.4.4. 每进入分程序一次，就要计算表达式一次。

5.2.5. 下标变量本身

下标变量本身与数组说明中给出的下标界无关，但是，即使一个数组已被说明为own，但对应下标变量的值在任何时候也只对这些变量中的那些下标在刚刚标出来的下标界之内的变量才有定义。

5.3. 开关说明

5.3.1. 语法

〈开关表〉::=〈命名表达式〉|〈开关表〉,〈命名表达式〉

〈开关说明〉::=switch〈开关标识符〉:=〈开关表〉

5.3.2. 例

```
switch S:=s1, s2, Q[m], if v>-5
```

```
then S3 else S4
switch Q:=p1, w
```

5.3.3. 语义

一个开关说明确定相应的开关命名符的一组值，这些值作为开关表中命名表达式的值一个一个地被定出。这些命名表达式中的每一个和在开关表中从左向右数它的项时得到的一个正整数1，2，…相关联。对应于下标表达式给出的一个值，开关命名符（参看3.5.命名表达式）的值是，开关表中以这个给出的值为其关联整数的命名表达式的值。

5.3.4. 开关表中表达式的计算

每逢开关表的项（表达式在其中出现）被引用时，就要去求开关表中该表达式的值，求值时用它包含的全体变量的当前值。

5.3.5. 作用域的影响

如果一个开关命名符在开关表的命名表达式中某量的作用域之外出现，并且开关命名符的值选到了这个命名表达式，则表达式中诸量的标识符和其说明在开关命名符处有效的那些标识符之间的冲突，经过对后者作适当的一系列改变后可以避免。

5.4. 过程说明

5.4.1. 语法

〈形式参数〉::=〈标识符〉

〈形式参数表〉::=〈形式参数〉|〈形式参数表〉〈参数定义符〉〈形式参数〉

〈形式参数部分〉::=〈空〉|(〈形式参数表〉)

〈标识符表〉::=〈标识符〉|〈标识符表〉,〈标识符〉

〈值部分〉::=value 〈标识符表〉;|〈空〉

〈区分符〉::=string|〈类型〉|array|〈类型〉array|label|switch|procedure|〈类型〉procedure

〈区分部分〉::=〈空〉|〈区分符〉〈标识符表〉;|〈区分部分〉〈区分符〉〈标识符表〉;

〈过程导引〉::=〈过程标识符〉〈形式参数部分〉;〈值部分〉〈区分部分〉

〈过程体〉::=〈语句〉|〈代码〉

〈过程说明〉::=procedure〈过程导引〉〈过程体〉|〈类型〉procedure〈过程导引〉〈过程体〉

5.4.2. 例（也请参看本报告末尾的例）

```
procedure Spur (a) Order: (n) Result:
    (s); value n; array a; integer
    n; real s;
begin integer k;
s:=0;
for k:=1 step 1 until n do s:=s+a
    [K,k]
end
procedure Transpose (a) Order: (n);
    value n; array a; integer n;
begin real w; integer i, k;
for i:=1 step 1 until n do
    for k:=1+i step 1 until n do
    begin w:=a[i, k]; a[i, k]:=a[k,
        i];a[k, i]:=w
    end
end Transpose
integer procedure Step (u); real u;
Step:=if 0≤u∧u≤1 then 1 else 0
procedure Absmax (a) size: (n, m)
    Result:(y) Subscripts: (i, k);
comment 把大小为n×m的矩阵a中绝对
    值最大的元素送到y，并把这个元素的
    下标送到i和k;
array a; integer n, m, i, k; real y;
begin integer p, q;
y:=0;
for p:=1 step 1 until n do for q:
    =1 step 1 until m do
if abs (a[p, q])>y then begin y:=
    abs (a[p, q]); i:=p; k:=q
end end Absmax
procedure Innerproduct (a, b) Order:
    (k, p) Result:(y); value k;
integer k, p; real y, a, b;
begin real s;
s:=0;
for p:=1 step 1 until k do
    s:=s+a×b
y:=s
end Innerproduct
```

5.4.3. 语义

过程说明用于确定与过程标识符相关联的过程，过程说明的主要部分（即过程体）是一个语句或一组代码，使用过程语句和（或）函数命名符即可从分程序（此分程序首部有该过程说明）的其他部分调用过程体。过程体配有一个导引，它详细说明过程体内代表形式参数的某些标识符。每当过程（参看3.2函数命名符和4.7过程语句）被用到时，过程体中的形式参数要被赋以实在参数的值或被代以实在参数。过程体中不是形式参数的标识符依其在过程体内被说明与否，对过程体而言或是局部的或是非局部的。其中那些对过程体而言是非局部的，对分程序（此分程序首

部有该过程说明）而言完全可以是局部的。不论一个过程体具有或不具有分程序的形式，它的作用总是和分程序的作用一样。因此当任何标号标出过程体内的一个语句或标出过程体本身时，此标号的作用域永远不能达到过程体之外。此外，如果一个形式参数的标识符在过程体内重新被说明（包括它被用作一个标号的情况，象在第4.1.3节中那样），则它就有了局部的意义；并且这个局部量的整个作用域是对应于该形式参数的实在参数所不能达到的。

5.4.4. 函数命名符的值

对于定义函数命名符之值的过程说明而言，在过程体内应当有一个或多个明显的赋值语句，并且赋值语句的左部有过程标识符；这些赋值语句中至少有一个应当被执行，并且过程标识符的类型应被说明，其方法是以类型说明符作为过程说明的第一个符号。在这些赋值中最近一次被赋的值要用来继续求表达式（函数命名符在此表达式中出现）的值。在过程体内过程标识符的任何出现若不在一个赋值语句的左部内时，就表示引用这个过程。

5.4.5. 区分

导引中可含有一个区分部分，它用明显的记号给出关于形式参数的种类和类型的信息。在这部分内任何形式参数最多只能出现一次。应当有赋值的形式参数的区分部分（参看第4.7.3.1节），但换名的形式参数的区分部分可以略去（参看第4.7.3.2节）.

5.4.6. 代码作为过程体

过程体可用非ALGOL的语言表示。由于使用这种功能完全是硬件表示的问题，所以在参考语言内不能给出关于代码语言的更进一步的规则。

过程说明的例

例1.

```
procedure euler (fct,sum,eps, tim); value eps, tim;
    integer tim; real procedure fct; real sum, eps;
    comment euler过程用适当改进过的欧拉变换来计算当 i 从零到无穷时 fct(i) 的和数。一俟变换过的级数的项的绝对值连续 tim 次都小于 eps，就停止求和。因此，应当有一个整数变量的函数 fct，一个上界 eps，和一个整数 tim，输出是和数 sum，在收敛很慢的级数或发散交错级数的情况，euler过程特别有效;
begin integer i, k, n, t; array m [0:15];
    real mn, mp, ds;
i:=n:=t:=0;m[0]:=fct(0);sum:=m[0]/2;
next term:i:=i+1; mn:=fct (i);
    for k:=0 step 1 until n do
        begin mp:=(mn+m[k])/2;m[k]:=mn;
            mn:=mp end means;
    if(abs(mn)<abs(m[n]))∧(n<15) then
        begin ds:=mn/2; n:=n+1;
            m[n]:=mn end accept
    else ds:=mn;
    sum:=sum+ds;
    if abs (ds)<eps then t:=t+1 else t:=0;
    if t<tim then go to next term
end euler
```

例2.

```
procedure RK (x, y, n, FKT, eps, eta, xE,
    yE, fi); value x,y; integer n; Boolean
    fi; real x, eps, eta, xE; array y, yE;
    procedure FKT;
```

comment:RK过程用自动寻求合适的积分步长的龙格一库塔方法求微分方程组 $y'_k=f_k(x,y_1,y_2,\cdots,y_n)$ $(k=1,2,\cdots,n)$ 的积分，参数是：x的初值x，未知函数 $y_k(x)$ 的初值 $y[k]$，方程组的阶 n，表示被积方程组的过程FKT(x,y,n,z)（即函数 f_k 的集合），控制数值积分精确度的容许值 eps 和 eta，积分区间的终点xE，输出参数yE（它表示在$x=xE$处的解），布尔变量 fi，每当单独进入 RK 时或第一次进入RK时，fi 的值应永远是 true，但若在某些网点 x_0，$x_1,\cdots,x_n$ 上都要用到函数 y 的值时，就必须重复地使用过程RK（这里当 $k=0$，$1,\cdots,n-1$ 时，$x=x_k$，$xE=x_{k+1}$），此时除第一次使用外，后面每次使用时都是 $fi=$false，这样可以节省计算时间。FKT 的输入参数是 x，y，n，输出参数 z 表示一组微商$z[K]=f_k(x,y[1],y[2],\cdots,y[n])$，它们是 x 和那些实在参数 y 的函数。过程 comp 作为非局部的标识符出现;

```
begin
    array z, y1, y2, y3 [1:n]; real x1, x2.
        x3, H;
    Boolean out; integer k, j; own real s,
        Hs;
    procedure RKIST (x, y, h, xe, ye);real
        x, h, xe;
        array y, ye;
        comment: RKIST 过程用龙格一库塔方法单独积分一步，初值是x，y[k]，得到的输出参数是xe=x+h和ye[k]，这里ye[k]是在 xe 处的解。重要的是：参数 n，FKT，
```

```
        z在RKIST中作为非局部量;
        begin
          array w[1:n], a[1:5]; integer k,
          j; a[1]:=a[2]:=a[5]:=h/2; a[3]:
          =a[4]:=h; xe:=x;
          for k:=1 step 1 until n do ye[k]:
            =w[k]:=y[k];
          for j:=1 step 1 until 4 do
          begin
            FKT (xe, w, n, z);
            xe:=x+a[j];
            for k=1 step 1 until n do
            begin
              w[k]:=y[k]+a[j]×z[k];
              ye[k]:=ye[k]+a[j+1]×z[k]/3
            end k
          end
        end RKIST;
Begin of program:
    if fi then begin H:=xE-x; s:=0
      end else H:=Hs;
    out:=false;
AA: if (x+2.01×H-xE>0)≡(H>0) then
    begin Hs:=H; out:=true; H:=(xE-x)/2
    end if;
    RKIST (x, y, 2×H, x1, y1);
BB: RKIST (x, y, x2, y2); RKIST (x2, y2,
    H, x3, y3);
    for k:=1 step 1 nntil n do
      if comp (y1[k], y3[k], eta)>eps then
        go to cc;
    comment: comp (a, b, c) 是函数命名符,
    其值的求法如下:
        把原先给定的参数a, b, c按阶之最大者实
        行对阶, 当a和b的阶都等于最大的阶后,
        求a和b的小数部分之差的绝对值就是
        comp (a, b, c) 的值;
    x:=x3; if out then go to DD;
    for k:=1 step 1 until n do y[k]:=y3[k];
    if s=5 then begin s:=0; H:=2×H end
      if;
    s:=s+1; go to AA;
CC: H:=0.5×H; out:=false; x1:=x2;
    for k:=1 step 1 until n do y1[k]:
      =y2[k];
    go to BB;
DD: for k:=1 step 1 until n do yE[k]:
      =y3[k]
end RK
```

索　　引

为便于查阅，将本报告中所用到的逻辑值、定义符、全部元语言变量及其它有关的语法概念逐条列出，每条后面注明有关的章节号码，表示可在该处查到；号码前注有“定”字者表示在该节中有它的语法定义，注有“元”字者表示它在该节的某些元语言公式中出现（已注过“定”字者不再重复），注有“文”字者表示在该节的叙述中有关于它的解释。逐条列出时的次序是：首先列出定义符中的+、−、×，…等三十个符号；其次将逻辑值与其它定义符按英文字典中的顺序依次列出；再次按中文笔画依次列出全部元语言变量及其它有关的语法概念，并在每条后面附有英文以供参考。