算法言語 ALGOL 60 に関する報告（3）*

Peter Naur 編

J.W. Backus, C. Katz, H. Rutishauser, J.H. Wegstein, F.L. Bauer, J. McCarthy, K. Samelson, A. van Wijngaarden, J. Green, A. J. Perlis, B.Vauquois, M. Woodger

淵 一 博**訳

5. Declaration（宣言）

declaration は，プログラムの identifier のある性質を定義する働きをする．identifier に対する declaration は一つのブロックに対して有効である．このブロックの外で，特殊な identifier を他の目的に用いることもあろう（4・1・3 節参照）．

動作でいえば，これは次のことを意味する：ブロックに入る時（これは **begin** による，というのは，内部のラベルは局所的であり，したがって，外部からそれを呼び出すことはできないからである）このブロックに対して宣言されるすべての identifier は，与えられた declaration の性質が意味する意義を持つものと仮定する．もしこれらの identifier が外部で他の declaration によりすでに定義されているならば，これらはこの時あらためて新しい意味を与えられるのである．他方，そのブロックに対して宣言されなかった identifier はその古い意味を保持する．

ブロックから脱出する時（**end** によるか，go to statement によるかで），そのブロックに対して宣言されているすべての identifier は，再びその意味を失う．

declaration は付加的な declarator **own** で特徴づけられることがある．これには次のような効果がある：そのブロックに再び入る時，**own** の量の値は，それが前の脱出の時持っていた値を変らずに持っているものとする．一方，宣言されてはいるが，**own** として特徴づけられていない変数については定義をしない．ラベルや procedure declaration の formal parameter を別にし，また標準関数（3・2・4 および 3・2・5 節参照）についてのもののような例外もあり得るが，プログラムのすべての identifier は宣言されなければならない．

シンタックス

<declaration>::=<type declaration>|<array declaration>|<switch declaration>|<procedure declaration>

5・1 Type Declaration（型の宣言）

5・1・1 シンタックス

<type list>::=<simple variable>|<simple variable>, <type list>

<type>::=**real**|**integer**|**Boolean**

<local or own type>::=<type>|**own**<type>

<type declaration>::=<local or own type> <type list>

5・1・2 例

integer p, q, s,

own Boolean Acryl, n

5・1・3 セマンティックス

type declaration は，ある identifier が，与えられた型の単純変数を表わすのだということを宣言する働きをする．**real** であると宣言された変数については，0 も含み正負の値を仮定するだけである，**integer** であると宣言された変数は，0 も含み正負の整数を仮定するだけである．**Boolean** であると宣言された変数は **true** と **false** の値だけを仮定する．

算術式において，**real** であると宣言された変数の占め得る場所はまた，**integer** であると宣言された変数が占めることもできる．

own のセマンティックスについては，上記 5 章の第 4 番目のパラグラフを見よ．

5・2 Array Declaration（配列の宣言）

5・2・1 シンタックス

<lower bound>::=<arithmetic expression>

<upper bound>::=<arithmetic expression>

---

* Report on the Algorithmic Language ALGOL 60（Com. ACM 3 No. 5, 1960）の訳の第 3 部である．

** 電気試験所電子部

<bound pair>::=<lower bound>:<upper bound>

<bound pair list>::=<bound pair>|<bound pair list>, <bound pair>

<array segment>::=<array identifier>[<bound pair list>]|<array identifier>, <array segment>

<array list>::=<array segment>|<array list>, <array segment>

<array declaration>::=**array**<array list>|<local or own type>**array**<array list>

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 セマンティックス

array declaration は，一つあるいはいくつかの identifier について，それが添字つき変数の多次元の配列を表わすことを宣言し，配列の次元，添字の限界および変数の型を与えるものである．

5・2・3・1 添字の限界　配列に対する添字の限界は，この配列の identifier に続く最初の添字括弧の中で bound pair list の形で与えられる．このリストの各項は，限定詞によって分けられた二つの算術式の形で添字の上限と下限を与える．bound pair list は左から右の順に考えてすべての添字の限界を与える．

5・2・3・2 次元　次元は，bound pair list の中の内容の数で与えられる．

5・2・3・3 型　一つの declaration で宣言されたすべての配列は，同じ型である．型の宣言詞（type declarator）がない時の型は，**real** であると考える．

5・2・4 上下限の式

5・2・4・1 この式は，添字式（3・1・4・2 節参照）と同じ方法で値を求めるものとする．

5・2・4・2 この式は，array declaration があてはまるそのブロックに対して，非局所的な変数および procedure に依存することができるだけである．したがってプログラムの一番外側のブロックでは，定数の限界を持つ **array** declaration だけしか宣言されないことになる．

5・2・4・3 配列は，すべての添字上限の値が，対応する下限の値より小さくないときに限り定義される．

5・2・4・4 式の値はブロックに入るごとに一度ずつ求めるものとする．

5・2・5 添字つき変数の同一性

添字つき変数の同一性は，**array** declaration で与えられる添字限界に関係するものではない*．しかし，配列が **own** であると宣言されても，対応する添字つき変数の値は，どういう時でも，最も新しく計算され添字限界内にある添字をもつこれらの変数の値に対してだけ定義されるものとする．

5・3 Switch Declaration（スイッチの宣言）

5・3・1 シンタックス

<switch list>::=<designational expression>|<switch list>, <designational expression>

<switch declaration>::=**switch**<switch identifier>:=<switch list>

5・3・2 例

switch S:=S1, S2, Q[m], **if** v>−5 **then** *S*3 **else** *S*4

switch Q:=p1, w

5・3・3 セマンティックス

switch declaration は，switch identifier に対応する値を定義する．これらの値は，switch list に入っている指定式の値として一つずつ与えられる．これらの指定式のおのおのについて，リストの中で項を左から右に数えて得られる正の整数が付随している．添字式のある与えられた値に対応するスイッチ指定詞（switch designator）の値は（3・5 節 Designational Expression 参照），switch list の中でこの与えられた値を附随する整数として持つ指定式の値である．

5・3・4 switch list の中の式の評価

switch list の中の式の値は，その式の出てくるリストの項を参照するごとに，含まれているすべての変数の，そのときどきの値を用いて求めるものとする．

5・3・5 範囲の影響

指定式に入っている任意の量の範囲外から，この特別な値に対し，スイッチ指定詞の値を参照しようとしても，これは定義されない．

5・4 Procedure Declaration

5・4・1 シンタックス

<formal parameter>::=<identifier>

<formal parameter list>::=<formal

---

* たとえば **array** Q[0,n]，と宣言された変数 Q について，そのブロックに入るごとに n の値は変っているかもしれない．しかしたとえば n=5 でも n=10 でも，Q[3] は同じ変数を表わしているものと考える．n=5 の場合，たとえば Q[7] の値は定義されないというのが次の説明である〔訳注〕．

```
  parameter>|<formal parameter list>
  <parameter delimiter><formal parameter>
<formal parameter part>::=<empty>|
  (<formal parameter list>)
<identifier list>::=<identifier>|<identifier
  list>, <identifier>
<value part>::value<identifier list>;|<empty>
<specifier>::=string|<type>|array|<type>
  array|label|switch|procedure|<type>
  procedure
<specification part>::=<empty>|<specifier>
  <identifier list>;|<specification part>
  <specifier><identifier list>;
<procedure heading>::=<procedure
  identifier><formal parameter part>;
  <value part><specification part>
<procedure body>::=<statement>|<code>
<procedure declaration>::=procedure
  <procedure heading><procedure body>|
  <type> procedure <procedure heading>
  <procedure body>
```

5・4・2 例*（この報告の最後の例も見よ）

```
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∧≦u1 then 1 else 0

procedure Absmax(a)size: (n, m) Result:(y)
Subscripts: (i, k);
comment The absolute greatest element of
  the matrix a, of size n by m is transfered
  to y, and the subscripts of this element
  to i and 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, b, a;
begin real s;
s:=0;
for p:=1 step 1 until k do s:=s+a×b;
y:=s
end Innerproduct
```

5・4・3 セマンティックス

procedure declaration は，procedure identifier に随伴する procedure（手順）を定義する働きをする．**procedure** declaration の主な構成要素は，一つの statement または一つのコード**，すなわち procedure body であるが，これは頭に **procedure** の現われるブロックの他の部分で procedure statement および（または）関数指定詞を使うことにより働きはじめる***．body と共に heading があるが，これは formal parameter を表現するものとして body の中に出てくる，ある identifier を指定する．procedure body の中の formal parameter は，この procedure が働き出す時（3・2 節 Function Designator および 4・7 節 Procedure Statement 参照）はいつでも，actual parameter の値を付与されるか，あるいは actual parameter で置きかえられるかするものとする．procedure body の中にある formal でな

---

* 第4例の **comment** の内容は次のとおりである．
大きさ n×m の行列 a の絶対値最大の要素を y に移し，またこの要素の添字を i と k に移す〔訳注〕．

** ここでコード（code）と呼ぶのは機械語，あるいはそれに近い Algol でない言語で書かれたプログラムのことである〔訳注〕．

*** すなわち，いわゆるサブルーチンのようなものである〔訳注〕．

い identifier は，それがこの body の中で宣言されているか否かによって，この body に対し局所的あるいは非局所的であるとする．そのうちのこの body に対し非局所的なものも，頭に **procedure declaration** の現われるブロック内では局所的であっても構わないのである．

5・4・4 関数指定詞の値

procedure declaration が関数指定詞の値を定義するために，その procedure body の内部で，procedure identifier にある値を付与するということをしなければならない．そしてその上，この値の型は，procedure declaration のいの一番の記号として型の宣言詞（type declarator）を出すことにより宣言しておかなければならない．

procedure body の procedure identifier が，その他いつどこで出てきても，それはこの procedure を働かせるという意味である．

5・4・5 明細

formal parameter の種類と型を明瞭な記法で与えるものである指定部（specification part）を heading の中に含めてもよい．この部分で1回以上出てくる formal parameter はないようにしてもよいし，名前で呼び出される formal parameter（4・7・3・2 節参照）は全部省略してもよい．

5・4・6 procedure body としてのコード

procedure body は ALGOL でない言語で表現されていてもよいと考える．この特徴を利用することは全く hardware representation の問題であるという心づもりなのであるから，このコードの言語に関してこれ以上の規則を reference langage の中で与えるということはできない．

Procedure Declaration の例

例　1*

```
procedure euler (fct, sum, eps, tim); value
eps, tim; integer tim; real procedure fct; real
sum, eps;
comment euler compute the sum of fct (i)
for i from zero up to infinity by means of
a suitably refined euler transformation. The
summation is stopped as soon as tim times in
succesession the absolute value of the terms
of the tranformed series are found to be less
than eps. Hence, one should provide a function
fct with one integer argument, an upper bound
eps, and an interger tim. The output is the
sum sum. euler is particularly efficient in the
case of a slowly convergent or divergent alter-
nating series;
begin integer i, k, n, t; array m[0 : 15]; real
mn, mp, ds;
i:=n:=i:=0; m[0]:=fct(0); sum:=m[0]/2;
nextterm: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;
```

* Comment の内容は次のとおりである．
euler は0から無限大までの i に対し fct(i) の和を適当に修正したオイラー変換によって計算するものである．変換された級数の項の絶対値が tim 回引きつづき eps より小なることがわかったら直ちに求和を止める．ここで，一つの整数変数をもつ関数 fct，上限 eps および整数 tim はあらかじめ用意されているものとする．出力は和 sum である．euler は，収束の遅いあるいは発散する交番級数の場合特に有効である．

* 各 comment の内容は次のとおりである．
第1の comment:
RK は，微分方程式の系 $yk'=f_k(x, y_1, y_2, \cdots\cdots, y_n)$ を Runge-Kutta 法で，積分間隔の適当な長さを自動的に探しながら積分するものである．パラメータは次のとおり．系の階数 n．積分される系すなわち関数 f_k の集合を表わす FKT (x, y, n, z)．数値積分の精度を制御する許容値 eps および eta．積分間隔の終り xE．x=xE における解を表わす出力パラメータ yE．
論理変数 fi．これには RK に別々にあるいは最初に入る時，値 **true** をつねに与えておかなければならない．関数 y をいくつかの格子点 $x_0, x_1, \cdots\cdots, x_n$ について求めなければならないときは，RK を繰返し（$x=x_k$, $xE=x_{e+1}$，ここで $k=0, 1, \cdots\cdots, n-1$ について）呼び出さなければならない．この時後の方の呼び出しでは fi=**false** が起ることがあるが，これは計算時間を節約するものである．FKT の入力パラメータは x, y, n でなければならず出力パラメータ z は，x および実際の y に対する微係数z [k]=f_k(x, y[1], y[2]……y[n])] の集合を表わす．comp という procedure は非局所 identifier として入っている．
第2の comment:
RK1ST は，出力パラメータ xe=x+h および ye[k](後者は xe における解）を作り出す．初期値 x, y, [k] をもつ単一の RUNGE-KUTTA を積分するものである．
重要事項：パラメータ n, FKT, z は非局所的実体として RK1ST に入る．
第3の comment:
comp (a, b, c) は関数指定詞であり，その値は，はじめに与えられたパラメータ a, b, c の指数部のうち最大のものに a と b の指数部をそろえ，そうした後での a と b の仮数部の差の絶対値である．

```
    if t<tim then go to nextterm
end euler
```

例 2*,8)

```
procedure RK (x, y, n, FKT, eps, eta, xE, yE,
fi); value x, y; integern ; Boolean fi ; real x,
eps, eta, xE; array y, yE; procedure FKT;
  comment: RK integrates the system yk'=fk
  (x, y1, y2,……,yn) (k=1, 2,……,n) of differen-
  tial equations with the method of Runge-Kutta
  with automatic search for appropriate length of
  integration step.
```

Parameters are: The initial values x and y[k] for x and the unknown functions $y_k(x)$. The order n of the system. The procedure FKT (x, y, n, z) which represents the system to be integrated, i. e. the set of functions f_k. The tolerance values eps and eta which govern the accuracy of the numerical integration. The end of integration interval xE. The output parameter zE which represents the solution at x=xE. The Boolean variable fi, which must always be given the value **true** for an isolated or first entry into RK. If however the functions y must be available at several mesh points $x_0, x_1, \cdots, x_n$, then the procedure must be called repeatedly (with $x=x_k$, $xE=x_{k+1}$, for $k=0, 1, \cdots\cdots, n-1$) and then the later calls may occur with fi=**false** which saves computing time. The input parameters of FKT must be x, y, n, the output parameter z represents the set of derivatives z[k]=fk (x, y[1], y[2],……,y[n]) for x and the actual y's. A procedure comp enters as a non-local identifier;

```
begin
    array z, y1, y2, y3 [1 : n]; real x1, x2,
    x3, H; Boolean out; integer k, j; own
    real s, Hs;
    procedure RK 1 ST (x, y, h, xe, ye); real
    x, h, xe; array y, ye;
      comment: RK1ST integrates one single
      RUNGE-KUTTA with initial values x, y[k]
      which yields the output parameters xe=x
      +h and ye[k], the latter being the solution
      at xe. Important: the parameters n, FKT,
      z enter RK 1 ST as nonlocal entities;
begin
  array w[1 : n], a[1 : 5]; integer k, j;
  a[1]:=a[2]:=a[5]:=k/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 (ne, 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 j
  end RK1ST
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;
  RK1ST (x, y, 2×H, x1, y1);
BB: RK 1 ST (x, y, H, x2, y2); RK 1 ST
    (x2, y2, H, x3, s3);
  for k:=1 step 1 until n do
    if comp (y1[k], y3[k], eta)>eps then
    go to CC;
  comment: comp(a, b, c) is a function design-
    ator, the value of which is the absolute
    value of the difference of the mantissa of
    a and b, after the exponents of these quan-
    tities have been made equal to the largest
```

8) この RK プログラムには, S. Gill, A process for the step-by-step integration of differential equations in an automatic computing machine *Proc. Camb. Phil. Soc.*, Vol 47 (1951) p. 96 および E. Fröberg, On the solution of ordinary differential equations with digital con puting machines, *Fysiograf. Sällsk. Förhd.*20 Nr. 11 (1956) p. 136-152 の着想に関連した新しい着想のいくつかが入っている. しかし, 計算時間および丸め誤差についてはこれは最適でないかもしれないし, またこれは実際に計算機にかけてテストしてあるものでもないことを明らかにしておかねばならない.

```
    of the exponents of the originally given
    parameters a, b, c;
  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
```

諸概念およびシンタックスでの諸単位の定義の索引

すべて節の番号によって参照する．参照は次の三つのグループに分れる．

def　略語 def に続くものは，シンタックスでの定義（もしあれば）を参照するものとする．

synt　略語 synt に続くものは，メタ言語の公式の中に出てくるものを参照するものとする．すでに def のグループで引用したものは参照を繰返さない．

text　text の語に続くものは，本文に出てくる定義を参照するものとする．

太字の語以外の，符号で表わされている基本記号は最初に集めてある．

索引を作るのに例は無視した．

+　plus を見よ
−　minus を見よ
×　multiply を見よ
／, ÷　divdie を見よ
<, ≦, =, ≧, >, ≠, <relational operator> を見よ
≡, ⊃, ∨, ∧, ¬, <logical operator> を見よ
,　comma を見よ
.　decimal point を見よ
10　ten を見よ
:　colon を見よ
;　semicolon を見よ
:=　colon equal を見よ
#　space を見よ
( )　parentheses を見よ
〔 〕　subscript bracket を見よ
' '　string quote を見よ
<actual parameter>, def 3.2.1, 4.7.1
<actual parameter list>, def 3.2.1, 4.7.1
<actual parameter part>, def 3.2.1, 4.7.1
<adding operator>, def 3.3.1
alphabet, text 2.1
arithmetic, text 3.3.6
<arithmetic expression>, def 3.3.1 synt 3, 3.1.1, 3.3.1, 3.4.1, 4.2.1, 4.6.1, 5.2.1 text 3.3.3
<arithmetic operator>, def 2.3 text 3.3.4
array, synt 2.3, 5.2.1, 5.4.1
array, text 3.1.4.1
<array declaration>, def 5.2.1 synt 5 text 5.2.3
<array identifier>, def 3.1.1 synt 3.2.1, 4.7.1, 5.2.1 text 2.8
<array list>, def 5.2.1
<array segment>, def 5.2.1
<assignment statement>, def 4.2.1 synt 4.1.1 text 1, 4.2.3
<basic statement>, def 4.1.1 synt 4.5.1
<basic symbol>, def 2
begin, synt 2.3, 4.1.1
<block>, def 4.1.1 synt 4.5.1 text 1, 4.1.3, 5
<block head>, def 4.1.1
Boolean, synt 2.3, 5.1.1 text 5.1.3
<Boolean expression>, def 3.4.1 synt 3, 3.3.1, 4.2.1, 4.5.1, 4.6.1 text 3.4.3
<Boolean factor>, def 3.4.1
<Boolean primary>, def 3.4.1
<Boolean secondary>, def 3.4.1
<Boolean term>, def 3.4.1
<bound pair>, def 5.2.1
<bound pair list>, def 5.2.1
<bracket>, def 2.3
<code>, synt 5.4.1 text 4.7.8, 5.4.6
colon :, synt 2.3, 3.2.1, 4.1.1, 4.5.1, 4.6.1, 4.7.1, 5.2.1
colon equal :=, synt 2.3, 4.2.1, 4.6.1, 5.3.1
comma ,, synt 2.3, 3.1.1, 3.2.1, 4.6.1, 4.7.1, 5.1.1, 5.2.1, 5.3.1, 5.4.1
comment, synt 2.3
comment convention, text 2.3
<compound statement>, def 4.1.1 synt 4.5.1 text 1

<compound tail>, def 4.1.1
<conditional statement>, def 4.5.1 synt 4.1.1 text 4.5.3
<decimal fraction>, def 2.5.1
<decimal number>, def 2.5.1 text 2.5.3
decimal point ., synt 2.3, 2.5.1
<declaration>, def 5 synt 4.1.1 text 1, 5 (complete section)
<declarator>, def 2.3
<delimiter>, def 2.3 synt 2
<designational expression>, def 3.5.1 synt 3, 4.3.1, 5.3.1 text 3.5.3
<digit>, def 2.2.1 synt 2, 2.4.1, 2.5.1
dimension, text 5.2.3.2
divide /÷, synt 2.3, 3.3.1 text 3.3.4.2
do, synt 2.3, 4.6.1
<dummy statement>, def 4.4.1 synt 4.1.1 text 4.4.3
else, synt 2.3, 3.3.1, 3.4.1, 3.5.1, 4.5.1 text 4.5.3.2
<empty>, def 1.1 synt 2.6.1, 3.2.1, 4.4.1, 4.7.1, 5.4.1
end, synt 2.3, 4.1.1
entier, text 3.2.5
exponentiation ↑, synt 2.3, 3.3.1 text 3.3.4.3
<exponent part>, def 2.5.1 text 2.5.3
<expression>, def 3 synt 3.2.1, 4.7.1 text 3 (complete section)
<factor>, def 3.3.1
false, synt 2.2.2
for, synt 2.3, 4.6.1
<for clause>, def 4.6.1 text 4.6.3
<for list>, def 4.6.1 text 4.6.4
<for list element>, def 4.6.1 text 4.6.4.1, 4.6.4.2, 4.6.4.3
<formal parameter>, def 5.4.1 text 5.4.3
<formal parameter list>, def 5.4.1
<formal parameter part>, def 5.4.1
<for statement>, def 4.6.1 synt 4.1.1, 4.5.1 text 4.6 (complete section)
<function designator>, def 3.2.1 synt 3.3.1, 3.4.1 text 3.2.3, 5.4.4
go to, synt 2.3, 4.3.1
<go to statement>, def 4.3.1 synt 4.1.1 text 4.3.3
<identifier>, def 2.4.1 synt 3.1.1, 3.2.1, 3.5.1, 5.4.1 text 2.4.3
<identifier list>, def 5.4.1
if, synt 2.3, 3.3.1, 4.5.1
<if clause>, def 3.3.1, 4.5.1 synt 3.4.1, 3.5.1 text 3.3.3, 4.5.3.2
<if statement>, def 4.5.1 text 4.5.3.1
<implication>, def 3.4.1
integer, synt 2.3, 5.1.1 text 5.1.3
<integer>, def 2.5.1 text 2.5.4
label, synt 2.3, 5.4.1
<label>, def 3.5.1 synt 4.1.1, 4.5.1, 4,6,1 text 1.4.1.3
<left part>, def 4.2.1
<left part list>, def 4.2.1
<letter>, def 2.1 synt 2, 2.4.1, 3.2.1, 4.7.1
<letter string>, def 3.2.1, 4.7.1
local, text 4.1.3
<local or own type>, def 5.1.1 synt 5.2.1
<logical operator>, def 2.3 synt 3.4.1 text 3.4.5
<logical value>, def 2.2.2 synt 2, 3.4.1
<lower bound>, def 5.2.1 text 5.2.4
minus —, synt 2.3, 2.5.1, 3.3.1 text 3.3.4.1
multiply ×, synt 2.3, 3.3.1 text 3.3.4.1
<multiplying operator>, def 3.3.1
nonlocal, text 4.1.3
<number>, def 2.5.1 text 2.5.3, 2.5.4
<open string>, def 2.6.1
<operator>, def 2.3
own, synt 2.3, 5.1.1 text 5, 5.2.5
<parameter delimiter>, def 3.2.1, 4.7.1 synt 5.4.1 text 4.7.7
parentheses ( ), synt 2.3, 3.2.1, 3.3.1, 3.4.1, 3,5.1, 4.7.1, 5.4.1 text 3.3.5.2
plus +, synt 2.3, 2.5.1, 3.3.1 text 3.3.4.1
<primary>, def 3.3.1
procedure, synt 2.3, 5.4.1
<procedure body>, def 5.4.1
<procedure declaration>, def 5.4.1 synt 5 text 5.4.3
<procedure heading>, def 5.4.1 text 5.4.3
<procedure identifier>, def 3.2.1, synt 3.2.1, 4.7.1, 5.4.1 text 4.7.5.4
<procedure statement>, def 4.7.1 synt 4.1.1 text

4.7.3
program, text 1
<proper string>, def 2.6.1
quantity, text 2.7
real, synt 2.3, 5.1.1 text 5.1.3
<relation>, def 3.4.1 text 3.4.5
<relational operator>, def 2.3, 3.4.1,
scope, text 2.7
semicolon ;, synt 2.3, 4.1.1 5.4.1
<separator>, def 2.3
<sequential operator>, def 2.3
<simple arithmetic expression>, def 3.3.1 text 3.3.3
<simple Boolean>, def 3.4.1
<simple designational expression>, def 3.5.1
<simple variable>, def 3.1.1 text 2.4.3
space #, synt 2.3 text 2.3, 2.6.3
<specification part>, def 5.4.1 text 5.4.5
<specificator>, def 2.3
<specifier>, def 5.4.1
standard function, text 3.2.4, 3.2.5
<statement>, def 4.1.1, synt 4.5.1, 4.6.1, 5.4.1 text 4 (complete section)
statement bracket, see: **begin end**
step, synt 2.3, 4.6.1 text 4.6.4.2
string, synt 2.3, 5.4.1
<string>, def 2.6.1 synt 3.2.1, 4.7.1 text 2.6.3
string quotes ‘ ’, synt 2.3, 2.6.1 text 2.6.3
subscript, text 3.1.4.1
subscript bound, text 5.2.3.1
subscript bracket [ ], synt 2.3, 3.1.1, 3.5.1, 5.2.1
<subscripted variable>, def 3.1.1 text 3.1.4.1
<subscript expression>, def 3.1.1 synt 3.5.1
<subscript list>, def 3.1.1
successor, text 4
switch, synt 2.3, 5.3.1, 5.4.1
<switch declaration>, def 5.3.1 synt 5 text 5.3.3
<switch designator>, def 3.5.1 text 3.5.3
<switch identifier>, def 3.5.1 synt 3.2.1, 4.7.1, 5.3.1
<switch list>, def 5.3.1
<term>, def 3.3.1
ten_{10}, synt 2.3, 2.5.1
then, synt 2.3, 3.3.1, 4.5.1
transfer function, text 3.2.5
true, synt 2.2.2
<type>, def 5.1.1 synt 5.4.1 text 2.8
<type declaration>, def 5.1.1 synt 5 text 5.1.3
<type list>, def 5.1.1
<unconditional statement>, def 4.1.1, 4.5.1
<unlabelled basic statement>, def 4.1.1
<unlabelled block>, def 4.1.1
<unlabelled compound>, def, 4.1.1
<unsigned integer>, def 2.5.1, 3.5.1
<unsigned number>, def 2.5.1 synt 3.3.1
until, synt 2.3, 4.6.1 text 4.6.4.2
<upper bound>, def 5.2.1 text 5.2.4
value, synt 2.3, 5.4.1
value, text 2.8, 3.3.3
<value part>, def 5.4.1 text 4.7.3.1
<variable>, def 3.1.1 synt 3.3.1, 3.4.1, 4.2.1, 4.6.1 text 3.1.3
<variable identifier>, def 3.1.1
while, synt 2.3, 4.6.1 text 4.6.4.3

訳語との対照表

<arithmetic expression>	算術式
<arithmetic operator>	算術作用素
<array declaration>	配列の宣言
<assignment statement>	附値 statement
<basic symbol>	基本記号
<block>	ブロック
<Boolean expression>	論理式
<code>	コード
<compound statement>	複合 statement
<conditional statement>	条件 statement
<decimal number>	10進数
<declaration>	宣言
<declarator>	宣言詞
<delimiter>	限定詞
<designational expression>	指定式
<digit>	数字
<empty>	空
<exponent part>	指数部
<function designator>	関数指定詞
<integer>	整数
<label>	ラベル

local	局所的
⟨logical operator⟩	論理作用素
⟨logical value⟩	論理値
⟨lower bound⟩	下限
nonlocal	非局所的
⟨number⟩	数
⟨operator⟩	作用素
procedure	手順
program	プログラム
⟨relation⟩	関係
⟨relational operator⟩	関係作用素
scope	範囲
⟨sequential operator⟩	順序作用素
⟨simple arithmetic expression⟩	単純算術式
⟨simple Booleam⟩	単純論理式
⟨simple designational expression⟩	単純指定式
⟨simple variable⟩	単純変数
standard function	標準関数
⟨string⟩	連鎖
subscript	添字
subscript bound	添字限界
subscript bracket	添字括弧
⟨subscripted variable⟩	添字つき変数
⟨subscript expression⟩	添字式
switch	スイッチ
⟨type⟩	型
⟨unconditional statement⟩	無条件 statement
⟨upper bound⟩	上限
value	値
⟨variable⟩	変数