REGNECENTRALEN
Gl. Carlsbergvej 2,
Copenha gen, Valby.

ALGOL-Bulletin no. 7.
3. November 1959.

7.1. EXPONENTIATION PARENTHESES.

Discussion. Ref. AB 2.2.1, 2.1.2.1, 3.9, 5.2.1.

Vote from Matematikmaskinnämndens Arbetsgrupp, Stockholm. 26.10.1959
"We agree with the proposal AB 5.2.1."

7.2. THE OPERATOR =.

Discussion. Ref. AB 3.1.1, 4.2.1, 5.2.2.

Vote from Matematikmaskinnämndens Arbetsgrupp, Stockholm: 26.10.1959
"We agree with the proposal AB 5.2.2."

Vote from Facit, Stockholm: 28.10.1959
"We agree."

7.3. THE EFFECT, FORM, AND SYMBOL FOR AUXILIARY INFORMATION.

Discussion. Ref. AB 5.8.

7.3.1. Vote from Matematikmaskinnämndens Arbetsgrupp, Stockholm: 26.10.1959
"We agree accept the proposals AB 5.8.1, 5.8.2, and 5.8.3."

7.3.2. Vote from Facit, Stockholm: 28.10.1959
"We agree."

7.3.3. Proposal from K. Samelson, Mainz:
"The following standards should be set up
a) It should be unmistakably characterized and readable.
b) It should be kept separate from ALGOL programs as far as possible.
c) At least amongst hardware-groups, it should be standardized as far as possible. Underlying machine characteristics usually are not unique."

See also the remarks by the Paris sub-committee, AB 7.28.4.
7.3.4. Vote from the ALGOL-group at Svenska Aeroplan AB:
"We agree".

7.4. ARRAY DECLARATIONS IN PROCEDURE HEADINGS.

Discussion. Ref. AB 5.12.

7.4.1. Comment from H. Rutishauser: 21.10.1959.
"In an array-declaration components of an array certainly may occur: ARRAY (x[1,1:n,y[s]]), but not arrays as a whole. This excludes a combination as given in 5.12, because at the moment such a declaration occurs, y[s,3] has no meaning."

7.4.2. Vote from Matematikmaskinnämndens Arbetsgrupp, Stockholm: 26.10.1959
"We accept the proposal AB 5.12".

7.4.3. Vote from Facit, Stockholm: 28.10.1959
"We agree."

7.4.4. Vote from the ALGOL-group at Svenska Aeroplan AB: 2.11.59.
"We agree."

7.5. <u>SYMBOLISM FOR LABELS AND SWITCHES AS OUTPUT PARAMETERS FOR PROCEDURES.</u>
Discussion. Ref. AB 5.16.

7.5.1. Vote from Matematikmaskinnämndens Arbetsgrupp, Stockholm: 26.10.1959
"We accept with pleasure the proposal AB 5.16.2."

7.5.2. Vote from Facit, Stockholm: 28.10.1959
"We agree."

7.5.3. Comment from Siemens: 27.10.1959.
"In AB 5.16 it is asserted that the proposed change would greatly simplify the work of the <u>translator</u>. We want to state that this is not in general true, but depends on the structure of the translator.

We want to support, however, the idea of characterizing tye labels among the procedure parameters, because of the <u>readability</u>, which is increased when one can immediately see from the heading, if the procedure has other exits as the normal one. Yet we do not find the proposed notation nice, but would instead recommend the suggestion of Backus (Backus' Paris report), according to which the possible labels are added in an extra parenthesis."

7.5.4. Vote from the ALGOL-group at Svenska Aeroplan AB:
"We do not agree." Cfr. AB 7.47.

7.6. Comment from H. Rutishauser concerning 21.10.1959
<u>THE EQUIVALENCE DECLARATION</u>(ed)Ref. AB 4.10.

"The equivalence decla ration seems a bit difficult to judge without knowing the intentions of the proposer. It seems to me however:

If the equivalence-declaration serves to exchange a subroutine at a certain place in a <u>written</u> ALGOL-program, then we have already a device for this:

Let exch(x) =: (y) be a procedure to be used in another procedure xxx, such tha t exch is only a dummy name for the procedure actually to be used. Then we can write:

```
PROCEDURE xxx(......, exch ( ) =: ( ), ...) =: (.......);
 .....
BEGIN xxxx
 ...
exch(u) =: (v);
 ...
END xxx;
```

Here the main program using xxx has to define what exch actually is to be, e.g.:

```
 ...
 ...
xxx(....., mocca( ) =: ( ), ...) =: (.....) ;
 ...
STOP ;
PROCEDURE mocca (p) =: (q) ;
 ...
BEGIN mocca:
 ...
END mocca ;
```

If however only in the <u>translated</u> program a subroutine which is used there is to be replaced by another subroutine which also is already in machine notation, then the description of the exchange is obviously not a matter of ALGOL, but rather some kind of "auxiliary information". Therefore I believe that the equivalence declaration is not needed."

7.7. Comment from H. Rutishauser concerning 21.10.1959.

ASSIGNMENT DECLARATION: CONSTANT (ed.) Ref. AB 4.7, 5.3.2.

"Agreement, but also subscripted variables should be possible inside the constant declaration. General form:

CONSTANT BEGIN V := N; V := N; .. .. V := N END

where V may be a simple or subscripted variable. Cf course BEGIN and END may be omitted of only one constant is delcared so. It is my feeling, that the constant declaration should serve at the same time to provide storage allocation, if the variables are subscripted. As to the question 5.3.2, I see no syntactical contradiction in assigning new values to the constants, but this would render the translated program worthless."

7.8. Comment from Siemens concerning 27.10.1959

ASSIGNMENT DECLARATIONS. Ref. AB 5.3.2.

"We do not quite understand the questions. A declaration is per definition independent of the running history of the program (or procedure) in which it occurs. "Constant" in sense of the assignment declaration therefore means "constant with respect to the running history of the program". The cases alluded to in the questions should be programmed by application of the assignment statement."

7.9. Comment from H. Rutishauser concerning 21.10.1959.

FUNCTION DECLARATION (ed.). Ref. AB 4.8.

"Agreement".

7.10. Comment from H. Rutishauser concerning 21.10.1959

FUNCTIONS IN PROCEDURE HEADINGS (ed.). Ref. AB 4.9.

"I Propose:

REQUIRE-declaration as follows:

Functions and procedures which are used inside the procedure but not defined there (by a corresponding function- or procedure-declaration) should be listed in the heading of the procedure by a special declaration:

Form: REQUIRE (I, I, I, ...,I)

The word-delimiter REQUIRE, followed by the list of the names (identifiers I) of all these functions and procedures.

Purpose: The user of the program can immediately see, which functions and procedures are needed inside the procedure, and the translator may provide alarm if one of these is not fed into the machine together with the procedure requiring them.

It is not needed to list a procedure or function declared inside the procedure, or functions like sin, cos, abs,.. which are supposed to be permanently available. Nor must a function or procedure be listed which appears as a parameter of the procedure, because such a parameter must be delivered by the next higher procedure in the hierarchy (and must be delcared or required there)."

7.11. Question from the ALGOL group at Regnecentralen concerning

PROCEDURES AS INPUT PARAMETERS FOR PROCEDURES.

"The Zürich report is not explicit on the point of exactly how a procedure, which is to be entered as an input parameter to another procedure, should be specified. Should the complete formal structure, including the =:, be quoted, thus for example:

A(a, B(, ,) =: (,), c) =: (d, e);

where the heading of the declaration for A starts:

<u>PROCEDURE</u> A(x, y(, ,) =: (,), z) =: (u,v); - ?"

7.12. Suggestion from the ALGOL group at Regnecentralen concerning

EQUIVALENCE OF FUNCTIONS.

"We agree with the MAILUFTERL group that the word 'equivalence' is unfortunate. Further alternative:

substitution."

7.13. Question from the ALGOL group at Regnecentralen concerning

THE INTEGERS OCCURRING IN ARRAY DECLARATIONS.

"In an array declaration the subscript ranges are defined by 'lists of integers separated by commas' (Zürich report, page 14, line 6 from above). It is not clear whether these integers are to be understood as integers G, i.e. positive integers, or whether negative integers are permitted as well."

7.14. Remark from Jan V. Garwick concerning

THE OPERATION OF THE FOR-STATEMENT.

"A for-statement, e.g. of the form

<u>for</u> i := a(1)b - 1; Σ

will according to the Zürich report always be executed at least once. If a and b are computed numbers and a = b, this clearly means that Σ shall not be executed at all. Would it not be better if this was taken into consideration so one would not have to precede the for-statement by a test to see if a $\geq$ b and in that case skip the for-statement?"

7.15. Comment from H. Rutishauser concerning 21.10.1959

THE OPERATION OF THE FOR-STATEMENT. (ed.)

"If the signs of c-a and b are different (in <u>FOR</u> V:=a(b)c (ed.)), then the list a(b)c is empty. (With an empty list 1, <u>FOR</u> V := 1; has the same operational effect as <u>IF</u> false). "

7.16. Memorandum from H. Rutishauser concerning

ARRAY-DECLARATIONS IN PROCEDURES.

"It must be clear that the rules for Array-declarations in procedures*) as given in the ALGOL-report, are too restrictive and therefore would allow only the most trivial applications. In order to remove this difficulty, the ALGOL-groups at Zürich and Mainz have spent considerable time in discussing possible solutions. It was found that

a) although it would remove the difficulties, it would be highly illogical to place all parameter-dependent array-declarations (i.e. also those for variables occurring only "inside" the procedure) into the the heading; therefore this solution must be strongly rejected.

b) If we allow all array-declarations in a procedure to depend on the input-parameters, this would remove the restriction also. So far, however, this is not allowed, but all the same, the present writer adopted this "soft rule" to describe some examples.

In addition to those more or less syntactical rules, the present writer encountered several times serious trouble when he tried to describe certain computations with ALGOL under the present rules. That this trouble is by no means confined to "ambitious" examples may be shown by the following ALGOL-program, which describes the generation and storing of the reciprocals of all integers from 1 to n (where n is given from the keyboard into the machine). This rather modest problem, even after adoption of the "soft rule" mentioned above, could not be described by less than a main program, calling in an artificial procedure VOID, which itself calls in an equally artificial procedure REZ:

```
keyboard (1) =: (n) ;                **)
VOID (n) =: (x);                     ***)    ****)      } main routine
STOP ;

PROCEDURE VOID (n) =: (x) ;
     BEGIN ARRAY (r[1:n]) ;
     VOID: REZ (n) =: (r[ ]) ;                          } subroutine VOID
     x := 0 ;  RETURN
END VOID ;

PROCEDURE REZ (n) =: (r[ ]) ;
     ARRAY (r[1:n]) ;
     BEGIN
     REZ : FOR k := 1(1)n ; r[k]  := 1/k ;              } subroutine REZ
     RETURN
END REZ ;
```

*) that only the array-declarations concerning parameters may be dynamic, i.e. may contain expressions.

**) The subroutine "keyboard (p) =: (n)" brings a new number from the keyboard into the machine; it must be written already in machine notation.

***) Procedure VOID serves only to shield the array r[ ] against the outside world in which no vector r[ ] could exist (because there is no corresponding array-declaration.

****) keyboard has no input parameters, VOID has no output parameters; for this reason the positions for the corresponding parameters have been filled by dummy variables.

In view of all these difficulties we finally came to the conclusion that in order to remove the true source of all the trouble, we should no longer heitate to adopt the only contradictionfree solution, namely the introduction of fully dynamic array-declarations according to the following definition:

An array-declaration stands at the beginning (i.e. just after BEGIN) of the compound statement Σ_c for which it is valid. It may contain expressions which depend on variables to which no new value is assigned within Σ_c, nor in the FOR-statement (if any) immediately preceding Σ_c.

With this new definition, the example cited above may be described as follows:

```
keyboard (1) =: (n) ;
REZ: BEGIN  ARRAY (r[1:n]) ;
FOR  k := 1(1)n ; r[k]  := 1/k ;
STOP
END REZ ;
```

I can understand that this proposal may be a shock to the reader who supposed ALGOL to be a well established language and just got used to it. Moreover I am fully aware that dynamic array-declarations are not easy to handle because they not only allow to extend, but also to cancel storage reservations during computation. Therefore only by careful planning one can prevent such mishaps as too early cancellation of storage reservations or unnecessary repetition of storage reservations.

On the other hand it must be clear that the new device is a quite natural but very powerful extension of the previous rule which includes it as a special case. Therefore the changeover might be made at any time without rendering old programs obsolete, and those who would like could still make programs according to the old rule. In fact the changeover might be adopted in steps, for this purpose the present writer offers some reduced versions of the proposal:

a) The new rule applies only to the following cases:
Σ_c is either a full program
or "the program part" of a procedure, which follows after the heading.
This is exactly the "soft rule" mentioned above; we have practically already agreed upon this.

b) The new rule applies only to compound statements which range from a given point to the very end of the program.
This is a true extension of the present rules and allows already to write the example given above in the condensed form. On the other hand, the rule b) would be still very easy to handle since it allows to extend storage reservations dynamically but not to cancel any more.

Some remarks must be made about the translators. Of course the new rule would lead to more complicated ALGOL-translators, but the additional difficulties would be rather modest. In a certain way, the placing of the array-declaration in front of the compound statement would even simplify some parts of the translators.

It is an open question, if also the other declarations (i.e. type declarations, function declarations, procedure declarations) should be placed at the beginning of the program."

7.17. Comment from H. Rutishauser concerning 21.10.1959

ARRAY-DECLARATIONS IN THE HEADING OF A PROCEDURE. Ref. AB 5.18.

"It is true, that they are superflous in most cases and may be omitted by the translator. However they serve, together with the require-declaration proposed above (AB 7.10), a very important purpose: The heading immediately shows what is actually going on in the procedure, otherwise the user of such a program has to search for the information through the whole description. In this connection it should not be forgotten that ALGOL is not only a means for automatic programming, but also for exchanging knowledge in the for of ALGOL-written numerical processes, therefore readability should not be neglected.

It has been mentioned that the same purpose can be reached by saying these facts in a comment-declaration. This is true, however I think very low of any rule which leaves it up to the producer of the program whether he likes to obey it or not. Therefore it is my feeling that the only way to enforce this information to be written into the heading is a strict syntactical rule regardless whether the translator uses it or not."

7.18. Comment from H. Rutishauser concerning 21.10.1959

THE LISTS OCCURRING IN FOR-STATEMENTS (ed.). Ref. AB 4.11.

"Agreement. This would lead to the following new definition:

FOR V := l , where l is a list, which is defined recursively as follows:

$$\left.\begin{array}{lll} l & \sim & a(b)c \\ & \sim & a \\ & \sim & l\ ,\ l \end{array}\right\}$$

where a, b, c are expressions not containing functions

(= concatenation of two lists)."

7.19. Comment from Siemens concerning 27.10.1959

BOOLEAN EXPRESSIONS. Ref. AB 5.17.

"It should be pointed out that the readability of Boolean expressions with round brackets or with square brackets to a great extent depends on the example you choose. Consider for example the following expression:

$$(a[m,n] \times b[p] \geq A[i]) \vee (Z[m,n] < B[i])$$

compared with

$$[a[m,n] \times b[p] \geq A[i]] \vee [Z[m,n] < B[i]]"$$

7.20. Suggestion from Facit, Stockholm, concerning 28.10.1959

THE OPERATOR ×.

"To avoid the chances to punch the letter X instead of the symbol × and the trouble with such an error punch, we suggest to change the symbol of multiplication form to an asterisk *."

7.21. THE ALPHABET OF ASSIGNMENT STATEMENTS (ed.).

Proposal from K. Samelson.

"The alphabet from which assignment statements are built up contains the class of labels and the class of quantities which consists of the different classes of symbols for numbers, variables with 0, 1, 2, ... subscripts, and functions with 1, 2, ... arguments. Members of different classes are distinguishable by class characteristics alone, and any reference to a quantity or label must contain its class characteristics. (Specifically, a single identifier always designates a variable with no subscripts.)

7.22. Proposal from K. Samelson concerning THE STATUS OF DECLARATIONS. (ed.)

"A declaration is a prefix to a statement (and not an independent entity). It is valid for, and part of, the statement following it: if Δ is a declaration, and Σ a statement, Δ,Σ is a statement and Δ is valid through Σ and Σ alone. Conflicting declarations on different levels of a statement are errors. (A labeled statement, when called by means of the label, begins with the label; declarations immediately preceding the label are not part of the statement called). For library procedures, modifications of the above definition may be desirable."

7.23. Proposal from K. Samelson concerning SUBSCRIPT BOUNDS IN ARRAY DECLARATIONS (ed.).

"In all array declarations, subscript bounds may be arbitrary integer valued expressions. An array is considered empty whenever any of its dimensions (difference between upper and lower bound of a subscript) is negative. "

7.24. Proposal from K. Samelson concerning THE STATUS OF PROGRAMS (ed.).

"Supplementary: a program is a statement which has neither predecessor nor successor."

7.25. Proposal from K. Samelson concerning IF- AND ALTERNATIVE STATEMENTS (ed.).

"Redefinition: (In the following, B are Boolean expressions, Σ statements.)

alternative statement   <u>if</u> B: Σ , <u>else</u> $\overline{\overline{\Sigma}}$

with the following supplementary rules:

7.25.1. If $\overline{\overline{\Sigma}}$ is empty, <u>else</u> may be omitted. This gives the conditional statement in the "single statement" form.

7.25.2. Concatenation is permissible, that is $\overline{\overline{\Sigma}}$ may again be an alternative statement. <u>if</u> B_1: Σ_1, <u>else</u> <u>if</u> B_2: Σ_2, <u>else</u> <u>if</u> ... , <u>else</u>

with the meaning of the alternative statement in its present form.

7.25.3. Substitution of an alternative statement for one of the Σ following a condition is permissible only by enclosing this statement in (statement) parentheses: if B_1; begin if B_{11}: Σ_{11}, else ... , else Σ_1 end else if B_2: Σ_2, ... else Σ .

This would replace both the present conditional and alternative statement. The possibility of entering, in the alternative, expressions E in place of the statements Σ, should be discussed. This is J.McCarthy's "conditional expression" and a possible alternative to logical multipliers which we shall have to deal with anyway. The other alternative is explicit introduction of the characteristic functions of predicates."

7.26. Proposal from K. Samelson concerning

THE CHARACTER OF 'FOR'.(ed.).

"Remove finally the statement character of the for which like the if is a subordinate clause."

7.27. Proposal from K. Samelson concerning

THE IDENTITY OF 'STOP' AND 'RETURN' (ed.).

"Stop and return have basically the same function. The difference is clear from context. Therefore a single symbol should be used (which should better not be stop since this leads to erroneous interpretations)."

7.28. Comment from the European members of the Paris sub-committee (E.W. Dijkstra, W.Heise, K. Samelson) concerning the sub-committee report. 27.10.195

"At the introduction of our report in Paris it was stated that the report was prepared in a hurry, and hence in no way complete and free of errors. After all, we still believe that this list of items to be remembered was better than nothing. - We add these remarks, particularly to clarify our opinion on some of the comments in AB5."

7.28.1. DUMMY STATEMENT. Ref. AB 5.1.

"The extra semicolon in our report is a pure error. We simply wanted to recommend Rutishausers proposal AB 3.4.2."

7.28.2. FUNCTIONS IN PROCEDURES. Ref. AB 4.9 and 4.10.

"These two items form an entity as in the original report p. 2 (There is of course no need for an equivalence declaration, if the function is an input parameter). What is said on functions in this item should apply to procedures as well."

7.28.3. DOMAIN OF VALIDITY FOR EQUIVALENCE DECLARATIONS. Ref. AB 5.6.2.

"Our opinion is that the equivalence declaration should be valid for all the procedures defined in the procedure declaration referred to by the equivalence declaration."

7.28.4. AUXILIARY INFORMATION. Ref. AB 5.8.

"In our report was stated: "This information is in no way connected with the reference language." This dim remark should be interpreted in the following way. a) The discussion of auxiliary information is beyond the scope of the American and European ALGOL committees. The question of this information should be settled independently by the different hardware groups. b) It should not be allowed to mix the auxiliary information and the ALGOL program itself. The auxiliary information should be given to the translator as a connected whole, for instance before the translation of the ALGOL program."

7.29. Suggestion from Siemens concerning 27.10.1959

THE NOTATION OF THE "FOR" STATEMENT.

"Our suggestion only affects the hardware group using 5-channel punched tape in accordance with AB 2.2.1. We suggest the following notation of the "for" statement determining an arithmetic progression:

'for' V := E'E'E

i.e. to replace the parentheses of the reference language by apostrophes. In this way one avoids the well-known difficulty in cases where the expressions contain functions."

7.30. Suggestion from Siemens concerning 27.10.1959

PROCEDURES IN GENERAL.

"Procedures in the form of the Zürich report are closed units, whose only connexion with the external program is via the input and output parameters. This structure is well suited for procedures with library character. It often occurs, however, that one wants to divide a program in blocks for different reasons, which have nothing to do with the use of or production of library procedures. These reasons might pay regard to the actual machine (for instance, adaptation to different levels of memory; core memory, drum memory etc.), but they might also be machine-independent (for instance, considering the clearness of the program, the ease of testing etc.).

In such cases the repeated transferring of information by each procedure call is rather cumbersome.

We therefore suggest that a new type of procedure be introduced, which differs from the original type in that it is not necessary by each call to transfer information about the variables, functions etc. to be used in and produced by the procedures.

A step in this direction is already taken in the suggestion of the Paris sub-committee (cfr. AB 4.9 and 4.10).

We do not want to make a detailed proposal at this moment, but we should like to draw the attention to this point, and to call upon the participants in the coming ALGOL conference to consider it thoroughly.

Questions in this connexion are for instance:

Should this proposed new type of procedures be close to the original type with respect to formal structure, or not? Should all variables, functions etc. be external, or should it be possible to distinguish in some way between external and internal ones?

Finally we want to express as our opinion that this point is an important one. If the problem is not solved, the consequence will be that each will make his own type of blocks by inserting "auxiliary information" between the statements and declarations of the ALGOL program itself (cfr. AB 5.8)."

7.31. Suggestion from A. van Wijngaarden and E.W. Dijkstra concerning

RANGE OF A DECLARATION.

"Replace the begin of chapter 5, Declarations, by something equivalent to

Declarations serve to state certain facts about entities referred to within the program. They have no operational meaning. They pertain to that part of the text which follows the declaration and which may be ended by a contradictory declaration. Their effect is not alterable by the running history of the program. Compatible declarations about the same entities can be given by writing ahead of one such declaration the declarators of the other declarations.

The meaning af "compatible declarations" should specified, e.g.

<u>array</u> <u>integer</u> (x[1:3])

tells that there is an array of integers x[1], x[2], x[3].

<u>complex</u> <u>integer</u> (z)

tells that there is a complex number z, real and imaginary part of which are integers (cfr. AB 7.34 (or rather AB 7.35 - ? Editor's note))."

7.32. Comment from A. van Wijngaarden and E.W. Dijkstra concerning

<u>THE USE OF NAMES</u> (ed.).

"Names (identifiers, numbers) should not be used in the same level (cfr AB 7.33) for different purposes, e.g. for a variable and a label. It has been shown (cfr. e.g. Bratman, CACM, <u>2</u>, 8(1959), p.4) that unexpected ambiguities may arise under special circumstances and there does not seem to be any serious need for multiple use of the same name. In particular integers should not be used as labels. Other difficulties are for instance <u>for</u> i = a(b)c, and <u>for</u> i = a(b)(d)c."

7.33. Suggestion from A. Van Wijngaarden and E.W. Dijkstra concerning

<u>LEVEL DECLARATIONS OLD, NEW</u>.

"In the begin of a procedure - the heading - automatically a new level of nomenclature is introduced that is left for good at the end in virtue of the sentence: "All identifiers and all labels contained in the procedure have identity only within the procedure, and have no relationship to identical identifiers or labels outside the procedure, etc.". This is useful and a nuisance at the same time (cfr. also H. Bratman, CACM, <u>2</u>, 8(1959), p.4). Apart from their suggestion (1) about the procedure statement, with which we agree, we suggest that the level declaration

<u>new</u> (I,I, ...)

has the effect that, the named entities have no relationship to identically named entities before in the following text, until the level declaration

<u>old</u> (I,I, ...)

which attributes to the entities named herein the meaning that they had before. These level declarations may be nested and form the only way to introduce a new meaning to a name. In particular in a procedure to be compiled along with the main program all variables that should have no relationship .. etc. should be declared <u>new</u> before they have appeared and declared <u>old</u> before the end.

These declarations do not only solve the problem of having "old" and "new" variables alongside in a procedure, but are also extremely useful in an ordinary program. It should be noted that after new(x) the new x is fully independent of the old x and, therefore, type declarations, if necessary, have to be given anew. On the other hand after old(x) the type declarations of the old x are still valid."

7.34. Suggestion from A. van Wijngaarden and E.W. Dijkstra concerning

TYPE DECLARATION DUMMY AND THE FUNCTION DECLARATION.

7.34.1. "According to the Zürich report (cfr. Bock and Bratman, CACM 2, 8(1959) p. 3-4) it is impossible to apply type declarations to the input variables of a function, which are interpreted by the function declaration automatically as formal variables (dummies). We suggest to drop this interpretation as formal variables and to introduce the type declaration dummy. This permits among other things to descern between different dummies and apply other declarations to them.
Example:

```
dummy integer (e,d) ...
AT[e,d] = A[d,e] ...
```

7.34.2. defines the transpose of a matrix. In here, and this is the next suggestion, the misleading symbol := in the function declaration is replaced by the non-operational symbol =. This permits moreover to declare a function which does not depend on an input parameter, without making it necessary to follow the function identifier by empty parentheses, in other words the introduction of abbreviations is automatically included in the function declaration.
Example:

```
r = sqrt (x×x + y×y);
pi = 3.14
```

If one replaces in these declarations the symbol = (read "stands for") by := (read "is replaced by"), it turns into an assignment statement with a completely different meaning.

These suggestions are compatible with the suggestion made elsewhere to introduce the declaration function which seems to be in agreement with the tendency in Algol to start by saying something about the overall character of what follows (cfr. for, if, etc.) although logically these remarks can be dispensed with at the cost of more difficult interpretation."

7.35. Suggestion from A. van Wijngaarden and E.W. Dijkstra concerning

DECLARATIONS COMPLEX, VECTOR, MATRIX, LIST, ETC.

"It should be possible to declare entities to be other things than real variables, e.g. complex numbers, vectors, matrices, lists (sets) of quantities. A quantity defined by such a declaration may enjoy well defined properties which make it possible to apply operators like +,-,x, etc. "in the conventional meaning", i.e. in the meaning that is conventional for such types of quantities.

E.g. if a,b,c are declared to be vectors by a declaration like

vector (a,b,c,[1,n])

where the corresponding identifiers occur in an array declaration as

array (a,b,c,[1,10])

then the unambiguous assignment statement

c := a+b

should be permitted. Specifications about what is the "conventional meaning" (see Zürich Report, 3,v) of the operations on such quantities must be defined in detail.

So in the case of a list, which we added to the list of operational declarations on purpose, the conceivable operations are not so obviously connected to symbols as +,-,x, etc. However, addition of an element to a list, of a list to a list, removal of an element from a list are quite useful operations which may be represented by + and -.

The identifier of a list could occur e.g. in the definition of the range of a for statement, e.g.

for i := L

where L has been declared to be a list, the length and the elements of which may be changed by the program.

Even without giving a fully worked out list of definitions we want to stress the importance of being able to use well defined commonly used mathematical concepts. A vector is essentially more than an enumeration of its elements, e.g. its length need not be identical to that suggested by the array declaration. Of course, we are aware that some of these operations hinted at can be written down in Algol as it stands by using a proper set of procedures, but there seems no reason to use a clumsy notation where a perfect one is in daily use."

7.36. Proposal for discussion from H. Bottenbruch concerning 26.10.195

PARAMETERS IN PROCEDURES. Ref. Zürich rep. pag. 56, line 2 from below

7.36.1. "Names of those procedures and functions defined outside the procedure need not be given as input parameters.

Entries in the parameter list should be restricted to variables, whereas these otherwise defined procedures and functions have fixed meaning.

7.36.2. If a procedure P is defined within another procedure or program P 1, identifiers of P may be identical with identifiers of P 1. By this convention we would have the same possibilities for procedures as we have for functions, namely the "hidden parameters". The practical advantages are obvious.

The following difficulties arise:

a) How can we distinguish "auxiliary variables" (these of course bear no relationship with all variables even of the same name outside P) from "hidden parameters". (The problem does not arise with hidden parameters in functions).

b) Among the auxiliary variables we have two classes:
b1) Those which "loose identity" after leaving the procedure
b2) Those which retain their meaning after leaving the procedure (not for use outside the procedure, but for reuse after reentering the procedure).

How can we distinguish between these two classes?

It is of course not absolutely necessary to distinguish between the two types of auxiliary variables; but then we either must treat all auxiliary variables as to be of the second kind, or we must forbid those of the second kind.

Difficulties a and b could be resolved by declarations "auxiliary 1" and "auxiliary 2". These declarations would also help the translator."

7.37. Proposal for discussion from H. Bottenbruch concerning 26.10.1959

<u>DECLARATIONS AND DIFFERENCE OF STATIC AND DYNAMIC STATEMENTS.</u>
(also Comment to Proposal AB 7.22 . of K. Samelson).

7.37.1. "It may be dangerous to make declarations a prefix to a statement. The basic facilities which are provided by the proposed modifications are, however, also resolved by the following proposal: <u>Give the declarations a dynamic meaning</u>. That is e.g. a declaration <u>array</u> (a[1:10]) is valid until another declaration (say) <u>array</u> (a[1:12]) is encountered. This would be particularly advantageous with declarations <u>single precision</u> or <u>double precision</u>. (The lack of dynamic declarations of this kind has been stressed by some people on the Paris Conference). It might be difficult for the translator to provide for appropriate storage space for: "dynamic arrays" without statements of the kind <u>empty</u> (a [...]) (meaning that array a [...] is no longer used).

7.37.2. We should, however, provide a possibility to declare a statement to be valid throughout a complete program (this only to enable the translator to construct more efficient programs). These statements may be the old declarations, or they may be statements like "π := 3.141". This could be done by a prefix <u>Constant</u>."

7.38. Suggestion from the ALGOL group at Regnecentralen concerning

<u>PROGRAM HEADINGS.</u> 15.10.1959.

"It is suggested that complete ALGOL programs should be provided with a heading, somewhat similar to a procedure heading. Among the uses of such a heading the following might be mentioned:

7.38.1. If conventions concerning "auxiliary information" similar to those suggested in AB 5.8 are adopted, the heading might provide an explicit reference to the particular system of such information actually employed in the program. Thus for instance "x DASK 2" might indicate that the auxiliary information refers to a particular translator, DASK 2.

7.38.2. Built-in functions (sin, cos, abs, etc.) employed in the program might be entered (cfr. AB 7.39).

7.38.3. A program designation, referring to an entrance label into the program might be specified.

Example:

```
PROGRAM 143 Part B; x DASK 2; FUNCTION abs( ), sign( ), entier( ), sqrt( );
BEGIN ....
143 Part B: ...
..... END 143 Part B;"
```

Addendum: Cfr. AB 7.42.

7.39. Comments from the ALGOL group at Regnecentralen concerning

<u>THE PLACE OF LIBRARY FUNCTIONS WITHIN THE HIERARCHY OF PROCEDURES.</u>

"One of the points raised by Ehrling (AB 5.13.3) may be reformulated thus: What is the place of the built-in library functions (abs( ), sin ( ), etc.) within a program containing several subordinate procedures? To this question there are essentially two possible answers: (1) The library functions are completely equivalent to functions defined in the program, except for the feature that their declaration is implied. (2) Library functions are quite exceptional. Let us consider these two possibilities separately.

7.39.1. Library functions similar to other functions:
<u>Implied declaration occurring in one place only.</u>

In this case the only question is where the implied declaration for the library functions should be understood to be placed. There are two reasonable answers:

7.39.1.1. In the main program.

7.39.1.2. Outside the main program, the main program being treated as a unique kind of procedure.

The difference between these two possibilities is the following: In case 7.39.1.1 it is impossible to redefine a library function in the main program, since this would be equivalent to trying to introduce two different declarations for the same function identifier, which clearly is a grammatical error. In case 7.39.1.2 such a redefinition should be made possible, either by the convention that a declaration of one of the library functions automatically should delete the implied declaration, or through a specific PROGRAM HEADING, cfr. AB 7.38.

Where the treatment of library functions inside procedures is concerned, both of these interpretations lead to the same conclusion:

a) The library functions must be entered through the procedure heading, if they are needed within a procedure (either as an input parameter or through a declaration (REQUIRE, cfr. AB 7.10).

b) If a library function is not entered from the main program, the corresponding identifier may be used for some other function within the procedure.

7.39.2. Library functions as exceptions:
<u>Implied declaration occurring in several places.</u>

In this case there is considerable freedom for choice. Consider the following schemes:

7.39.2.1. Library functions as universal, unredefinables. This would correspond to the declarations for the library functions being repeated inside all procedures, and then considering them as ordinary functions.

7.39.2.2. Library functions as universal, but freely redefinable entities. The declarations for the library functions would be understood to be repeated inside each procedure, unless the functions in question were defined differently, in which case the new function declaration would be understood to have the double effect of deleting the implied declaration and introducing the new function. This would hold only at the same level, and not inside procedures defined in this level.

We ourselves tend to prefer the answer 7.39.1.2, i.e. the interpretation that library functions are similar to ordinary functions defined at a higher level than the main program, and that all library functions used inside a procedure must be declared in the heading. As far as we can see this agrees with Ehrling's proposal

AB 5.13.3 except for the point that Ehrling wishes to leave out the declaration in the heading. We recognize, in principle, the need for Ehrling's library declaration, but we feel that in practise the very complicated procedure hierarchies will not often occur, and would prefer avoiding to complicate the language by leaving it out.

7.39.3. Recommendations.

7.39.3.1. Built-in library functions should be treated as like other functions as possible.

7.39.3.2. They should be quoted in the heading of the procedures in which they are used.

7.39.3.3. They should be entered into the main program through a specific heading to the whole program (cfr. AB 7.38)."

7.40. Comment from the ALGOL group at Regnecentralen concerning

FULLY DYNAMIC ARRAY-DECLARATIONS. Ref. AB 7.16. 2.11.1959.

"We are not convinced by the arguments used by H. Rutishauser. First of all the program of the first example given does not seem to do its work properly, since the array r[ ] will not be available in the main program after the procedure statement VOID (n)=:(x). Again, in the second example there must clearly exist an upper bound to the permissible values of n. This is not evident anywhere in the program. However, this upper bound to the size of any array appears to us to be a very important part of the problem.

If we understand Rutishauser correctly the intention is that different arrays should be allowed to share the same storage locations, both in procedures and in the main program (this in spite of the title of Rutishausers memorandum, which might suggest that only procedures are involved). Where this problem is concerned however, it is our feeling that no matter how the facilities of the algorithmic language are arranged, the utilization of the available storage space will always finally rest with the programmer. In this view we do not seem to disagree with Rutishauser, who specifically states "that dynamic array declarations are not easy to handle ..... only by careful planning ...". On the other hand, with the facilities already incorporated in the ALGOL of the Zürich report this administration is not a difficult matter. The following example shows how it may be done in a simple case, similar to that considered by Rutishauser. A single array of fixed size is used to accomodate two different entities, which are known beforehand not to require more than a fixed number of components in total, while the distribution of the storage space among the two entities may be changed dynamically. The problem is this: Two numbers n and m are given manually. It is desired to store vectors $1/1, 1/2, \ldots 1/n$ and $1/1^2, 1/2^2, \ldots 1/m^2$. Altogether 100 locations are available for the purpose. Program:

```
ARRAY (a[1:100]);
keyboard (1) =: (n); keyboard (1) =: (m);
if (n + m > 100); BEGIN print ('fool'); STOP END ;
FOR i := 1(1)n; a[i]  := 1/i;
FOR i := 1(1)m; a[i+n]  :=  1/i↑2↓ ; STOP ;
```

From the practical point of view very little seems to be gained by special conventions, which allow one array to grow at the expense of another. In fact, problems in which this would be useful are rare. Ordinarily all arrays in a problem tend to grow or shrink in equal proportions.

Thus, as far as we can see the fully dynamic array declarations will be difficult to use, difficult to translate, and will not solve the real problem at stake. We therefore recommend: stick to the Zürich ALGOL at this point."

Memorandum from F.L. Bauer concerning

SOME PROPOSALS DISCUSSED IN THE USA.

7.41. USE OF DELIMITERS IN FOR- AND IF- STATEMENTS.

"It is proposed to use more distinguished delimiters in the reference language (this does not exclude, that for some of the delimiters the same hardware symbol is used).

7.41.1. Now $\underline{\text{if}}$ B $\underline{\text{then}}$ Σ instead of $\underline{\text{if}}$ B ; Σ

7.41.2. Now $\underline{\text{for}}$ V := E_1 $\underline{\text{step}}$ E_2 $\underline{\text{until}}$ E_3 $\underline{\text{exec}}$ Σ
instead of $\underline{\text{for}}$ V := $E_1(E_2)E_3$; Σ

7.41.3. Concerning alternative statement, see SAMELSON (AB 7.25).

(Note, that the word delimiters are used in order to indicate the structure of the proposal, their choice is secondary).

A corresponding change in the syntactical structure (now 'if-prefix', 'for-prefix'), see SAMELSON (AB 7.26), removes also some inconsistencies."

7.42. THE SO-CALLED 'END' OF A SO-CALLED 'PROGRAM'

"Program is a (compound) statement (cfr. also Samelson, AB 7.24. Editors note): it starts with $\underline{\text{begin}}$ and ends with $\underline{\text{end}}$. A sequence of compound statements, without being a compound statement itself, is to be considered as a set of independent programs.

This gives sufficient information for the translator in order to translate and to start computation.

A library program is a procedure declaration. This gives sufficient information for the translator in order to translate and to 'print out' the library in translated code. "

7.43. ADDENDUM TO PROCEDURE DECLARATION.

"If an input variable in a procedure declaration does not appear also at the output side, its (numerical) values are unchanged after leaving the procedure (it is 'saved'). Implementations for the translator are obvious. "
(Editor's note: cfr. AB 3.7).

7.44. INTERMEDIATE EXIT IN PROCEDURES.

"Intermediate exit of a procedure, that means re-entering a procedure as it was left, can always be done be appropriate (sub-)procedure parameters."

7.45. NON-SYNONYMOUS EXPRESSIONS IN (NUMERICAL) ARITHMETIC.

"In those types of arithmetic, where the numbers possibly are subject to rounding, using the associative or distributive law does not give synonyma. However, the strict commutative law for a pair of operands gives synonymous expressions.
Example: Synonymous are (a+B)+c, c+(a+b), (b+a)+c, c+(b+a).
Not synonymous are (a+b)+c, a+(b+c), (a+c)+b.

7.46. EXTENSIONS TO STRING-HANDLING OPERATIONS.

"Some people feel, that ALGOL, supplemented by some string-handling operations, is well adapted to so-called data processing (including the description of a translator). Several proposals have been made in the USA."

7.47. Comment from the ALGOL-group at Svenska Aeroplan AB concerning
SYMBOLISM FOR LABELS AND SWITCHES AS OUTPUT PARAMETERS FOR PROCEDURES 30.10.59.
Ref. AB 5.9.4, 5.15, 5.16, and 7.5.

"We are not in favor of introducing a new symbolism for labels and switches in procedure statements and declarations. To us it seems better to sestrict the use of identifiers in the procedure heading when denoting variables on the one hand and designational expressions on the other. This means that among the input- and outputparameters in the procedure heading a simple variable may not have the same identifier as a label, and an array with one subscript may not have the same identifier as a switch variable. On the other hand this could be permitted for procedure statements where the position of the parameter in the list of input- and outputparameters defines its function."

7.48. Comment from the ALGOL-group at Svenska Aeroplan AB concerning
ASSIGNMENT DECLARATION: CONSTANT. Ref. AB 5.3.2 and 7.7, 7.8. 2.11.59

"It should not be permitted to change the value of a constant declared variable. If the value of the constant is changed dynamically and it then has to be reset, the effect of the constant assignment would be the same as the ordinary assignment statement V := N (N number) which then could be used."

7.49. CHANGE OF REPRESENTATION:
Siemens and Halske AG, München, Germany, will from now on be represented by W. Heise, formerly of Regnecentralen.

7.50. Note from the editor:
Participants at the Paris Conference (cfr. AB 6) are requested to bring their own copies of the ALGOL BULLETIN, since only a few spare copies are available.
Further notes on new members will be brought in the next issue.

CLASSIFIED INDEX OF SUGGESTIONS AND DISCUSSION CONCERNING THE ALGOL LANGUAGE APPEARING WITHIN AB 1 - 7.

The main classification corresponds closely to the Zürich report. Hardware questions are omitted. Some items appear in several places.

2. BASIC SYMBOLS.

Use * for multiplication (Facit) 7.20.
New delimiters in for- and if- (Bauer) 7.41.

Numerical constants.

Permanent identifiers for π and e (SAAB, Heise) 2.6, Discussion 3.3, 4.4.
Constant coefficients for power series (SAAB) 2.7

Identifiers for variables, arrays and functions.

Does bracket structure identify? (Naur) 4.14, Yes! (Rutishauser) 5.9.3.
(Garwick) 5.9.1.
Suggestion for grouping of entities (Regnecentralen) 5.9.4.
Number of subscripts (arrays) and variables (functions) identifies (Samelson) 7.21.
Identifier alone identifies (van Wijngaarden, Dijkstra) 7.32.

3. iv) FUNCTIONS.

Enter library functions through headings! (Regnecentralen) 7.39.

3. v) ARITHMETIC EXPRESSIONS.

Non-synonymous expressions (Bauer) 7.45.

3. vi) BOOLEAN EXPRESSIONS.

Rule for precedence of log. operators (Paris sub.com.) 4.12. No, wait and see!
(Mailufterl) 5.7.
Generalization of log. operations (Garwick) 4.16. Criticism (Rutishauser) 5.11.
Use [ ] around arithmetic relations (Regnecentralen) 5.17. Criticism (Siemens) 7.19.

4. STATEMENTS.

Dummy statement.

Need for (Bring, Ehrling) 2.4.
Rutishausers suggestion: 3.4.2.
Paris sub.com. suggestion 4.5.2.
Agreement 5.1.
Paris sub-com. recommends Rutishausers suggestion 7.28.1.

4. i) ASSIGNMENT STATEMENTS.

What happens to a:= b/c; integer (a); - ? (Naur) 4.15.
Answer: round-off (Rutishauser) 5.10.
The alphabet of assignment statements (Samelson) 7.21.

4. iii) IF- STATEMENT.

Redefinition (Samelson) 7.25.

4. iv) FOR-STATEMENT.

Allow E(E)E,E,E,E(E)E ... (Paris sub-com) 4.11. Support (Rutishauser) 7.18.
Let a(1)b, b<a cause a skip (Garwick) 7.14, (Rutishauser) 7.15.
Remove statement character (Samelson) 7.26.
New delimiters (Bauer) 7.41.

4. v) ALTERNATIVE STATEMENTS.

Redefinition (Samelson) 7.25.

4. vii) STOP STATEMENTS.

Allow dynamic succession (Bring, Ehrling, Heise) 2.3.
Special label for stop (Rutishauser) 3.2.1. Answer (Heise) 3.2.1.1.
Not in favour of dynamic succ. (Bauer, Samelson) 3.2.2.
Agreement in favour of dynamic succession 4.3.
Replace by "RETURN" (Samelson) 7.27.

4. viii) RETURN STATEMENTS.

Similarity with "STOP" (Samelson) 7.27.

4. ix) PROCEDURE STATEMENTS.

Only output parameters may change (Naur) 3.7, (Bauer) 7.43.
May input and output parameters be identical?(Regnecentralen) 3.7.
Why are switches as output parameters not permitted? (Regnecentralen) 5.15.
Use special symbolism for labels and switches as output parameters (Regnecentralen) 5.16.2
Discussion (Siemens) 7.5, (SAAB) 7.47.
What formalism should be used for procedures as input parameters (Regnecentralen) 7.11.

5. DECLARATIONS.

Status of declarations.

Allow dynamic array-declarations (Rutishauser) 7.16.
Declarations as prefixes to statements (Samelson) 7.22.
Declarations operative for following program (van Wijngaarden, Dijkstra) 7.31, (Bottenbruch) 7.37.
Compatible declarations (van Wijngaarden, Dijkstra) 7.31.
Stick to Zürich ALGOL! (Regnecentralen) 7.40.

Constant-declaration.

Proposal (Paris sub-com.) 4.7.
Why a 'constant' change? (Regnecentralen) 5.3. Discussion 7.7, 7.8, 7,48.
Precise definition (Rutishauser) 7.7.

Length of procedure declaration (Ehrling) 3.6.

Equivalence declaration.

Proposal (Paris sub-com.) 4.10
Other designations: Coordination, transformation, identification, substitution (Hailufterl) 5.6.1 (Regnecentralen) 7.12.
Operational meaning in multi-identified procedure? (Regnecentralen) 5.6.2.
Operates for all proc. within same declaration (Paris sub-com.) 7.28.3.
Is not needed (Rutishauser) 7.6.

Old, new, declaration (van Wijngaarden, Dijkstra) 7.33.
Empty declaration (Bottenbruch) 7.37.1.
Library declaration (Ehrling) 5.13.3.
Auxiliary declarations (Bottenbruch) 7.36.2.

5. i) TYPE DECLARATIONS.

Dummy (van Wijngaarden, Dijkstra) 7.34.1.

5. ii) ARRAY DECLARATIONS.

Variable size arrays (Bring, Ehrling) 2.5.1
Subscripts are constants (Naur) 2.5.3.
Negative subscripts permitted? (Regnecentralen) 7.13.
Allow expressions in subscript bounds (Samelson) 7.23.
Introduce fully dynamic array declarations (Rutishauser) 7.16.

5.iv) FUNCTION DECLARATIONS.

Write: 'FUNCTION' (Paris sub-com.) 4.8. Approval (Hailufterl) 5.4, (Rutishauser) 7.9.
Use =, not := (van Wijngaarden, Dijkstra) 7.34.2.

5. vi) PROCEDURE DECLARATIONS.

Fundamental properties.
Only variables, no functions, through heading (Bottenbruch) 7.36.1.
Permit hidden parameters (Bottenbruch) 7.36.2
Old, new, declarations (van Wijngaarden, Dijkstra) 7.33.
Intermediate exits (Bauer) 7.44.

Array declarations.
Why are they there? (Bring, Ehrling) 2.5.2, 3.5, (Regnecentralen) 5.18.
To help reading (Rutishauser) 7.17.
Restrict permissible parameters in bounds! (Regnecentralen) 5.12. Discussion 7.4.

Functions and procedures entered through heading.
Proposal (Paris sub-com.) 4.9. Approval (Hailufterl) 5.5.
Use word "REQUIRE" (Rutishauser) 7.10.
Should hold for procedures also (Paris sub-com.) 7.28.2.

Built-in library functions.
Library declaration (Ehrling) 5.13.3.
Enter all functions through heading (Regnecentralen) 7.39.

Formalism for procedures as input parameters? (Regnecentralen) 7.11.

INPUT-OUTPUT.

INPUT, OUTPUT, FORMAT (Paris sub-com.) 4.6.

AUXILIARY INFORMATION.

Idea (Paris sub-com.) 4.13. Further explanation (Paris subcom.) 7.28.4.
Suggestion on effect, form and symbol (Regnecentralen) 5.8. Approval (Matematik-
Philosophy (Samelson) 7.3.3. maskinn.), (Facit), (SAAB) 7.3.

PROGRAMS.

The uses of a program heading (Regnecentralen) 7.38.
Programs as statements (Samelson) 7.24 (Bauer) 7.42,

EXTENSIONS OF FUNDAMENTAL OPERATIONS.

Generalize logical operations (Garwick) 4.16. Criticism (Rutishauser) 5.11.
Include vector- matrix- etc. arithmetics (van Wijngaarden, Dijkstra) 7.35.
Extension to string-handling (Bauer) 7.46.

THE NEED FOR A NEW TYPE OF PROGRAM SECTION (Siemens) 7.30.