```
begin array z, y1, y2, y3 [1 : n] ; real x1, x2, x3, H ; Boolean out ;
    integer k, j ; own real s, Hs ;
    procedure RK1ST (x, y, h, xe, ye) ; real x, h, xe ; array y, ye ;
        comment RK1ST integrates one single Runge-Kutta step 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 RK1ST as non local entities ;

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 j
end RK 1ST ;

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 : RK1ST (x, y, H, x2, y2) ; RK1ST (x2, y2, H, x3, y3) ;
        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 designator, the value of which is the absolute value
        of the difference of the mantissae of a and b, after the exponents of these quantities have
        been made equal to the largest 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
```

ALPHABETIC INDEX OF DEFINITIONS OF CONCEPTS AND SYNTACTIC UNITS

All references are given through section or clause numbers. The references are given in three groups:

- def — Following the abbrevation " def ", reference to the syntactic definition (if any) is given.
- synt — Following the abbreviation " synt ", references to the occurrences in metalinguistic formulae are given. References already quoted in the def-group are not repeated.
- text — Following the word " text ", the references to definitions given in the text are given.

The basic symbols represented by signs other than bold-faced words have been collected at the beginning. The examples have been ignored in compiling the index.

PART I B

REPORT ON LEVEL 1 AND 2 SUBSETS (ECMA SUBSET)

INTRODUCTION

The following report gives the exact working of the restrictions and changes necessary to change the report on the full language into a report on level 1 and level 2 subsets. Attention must be drawn to some slight deviations between this report and the worked in version in the boxes in Part I A. This is mainly caused by typographical difficulties. For example, where level 1 and 2 subsets require at least 26 letters in the alphabet, level 3 subset permits at most 26 letters.

The change listed under item 2 in the definitions does not apply to level 1 subset. This is the only difference between the subsets of levels 1 and 2.

DEFINITIONS AND REMARKS

Remarks to compiler designer	Remarks to programmer	Definition of Ecma subset in terms of ALGOL 60 Report (The numbers refer to sections and clauses of Part I A)
1. The declarator **own** need not be implemented.	The use of **own** variables should be avoided.	**5.** Delete first two sentences of fourth paragraph. **5.1.1** Delete " | **own** ⟨type⟩ ". **5.1.3** Delete last sentence.
2. The recursive use of procedures and recursively defined procedures need not be allowed. No recursive use of procedures means that no call of the procedure itself may occur during the execution of the statements of the body of any procedure, and during the evaluation of those of its actual parameters, the corresponding formal parameters of which are called by name, and during the evaluation of expressions occurring in declarations inside the procedure.	Do not write recursive procedures. Do not use procedures recursively. THIS CHANGE DOES NOT APPLY TO LEVEL 1 SUBSET	**5.4.4** Delete last sentence. Add to 4.7.5: " No call of the procedure itself may occur during the execution of the statements of the body of any procedure, and during the evaluation of those of its actual parameters, the corresponding formal parameters of which are called by name, and during the evaluation of expressions occurring in declarations inside the procedure."
3. Integer labels need not be provided for.	Do not use integer labels.	**3.5.1** Delete " | ⟨unsigned integer⟩ ". **3.5.5** Delete.

Remarks to compiler designer	Remarks to programmer	Definition of Ecma subset in terms of ALGOL 60 Report (The numbers refer to sections and clauses of Part I A)
4. Up to full specification for all parameters may be required. Up to equivalence of the type of an actual parameter and the specified type of the corresponding formal parameter, if called by name, may be required.	Provide each procedure declaration with a complete specification part and maintain equivalence of type between actual and formal parameter called by name.	**5.4.5** Third sentence – replaced by " Specifications of all formal parameters if any must be supplied." **4.7.5.5** Replaced by: " Kind and type of actual parameters must be the same as those of the corresponding formal parameters, if called by name."
5. The alphabet may not be restricted to less than one case of 26 letters.	Use only one case of letters. (Either small or capital—the typing for a particular implementation will transfer if necessary to the right case).	**2.1** Delete " | *A* ... *Z* " and " restricted, or ".
6. A limit may be put on the length of identifiers, but this must not be less than 6 significant basic symbols.	Do not rely on the differentiation between two identifiers which have the first six basic symbols in common.	**2.4.3** Replace " They may be chosen freely " by " Identifiers may be chosen freely; but the effects due to the occurrence of two different identifiers the first six basic symbols of which are common are undefined ".
7. If the type of an arithmetic expression depends upon the evaluation of an expression or upon the type or value of an actual parameter then it may be taken to be **real**.	Be careful when making essential use of the type of an arithmetic expression.	**3.3.4** Replace the words " the following rules " of the last sentence by " a set of rules. However if the type of an arithmetic expression according to the rules cannot be determined without evaluating an expression or ascertaining the type or value of an actual parameter, it is **real**. These rules are ".

Remarks to compiler designer	Remarks to programmer	Definition of Ecma subset in terms of ALGOL 60 Report (The numbers refer to sections and clauses of Part I A)
8. The requirement of 4.3.5 need not be implemented.	A go to statement involving an undefined switch designator need not have the effect of a dummy statement. You are well advised to program a check that the value of the subscript expression is within the bounds defined by the switch declaration.	**4.3.5** Replace " equivalent to a dummy statement " by " undefined ".

PART I C

REPORT ON LEVEL 3 SUBSET (IFIP SUBSET)

INTRODUCTION

The definition of level 3 subset appears in the form of informal explanations and changes to be made to the report on the full language ALGOL 60 to convert this into the level 3 subset (as has been done in the boxes in Part I A).

DEFINITIONS AND EXPLANATIONS

Section or clause of Part I A	Subset definitions	Explanation
2.3	Delete from definition of declarator " **own** | ".	The **own** concept is not included in the subset.
5	Delete first two sentences of fourth paragraph.	
5.1.1	Replace the last two metalinguistic formulae by: "⟨type declaration⟩ : := ⟨type⟩ ⟨type list⟩ ".	
5.1.3	Delete last sentence.	
5.2.1	Replace the last formula by: " ⟨array declaration⟩ : := **array** ⟨array list⟩ | ⟨type⟩ **array** ⟨array list⟩ ".	
5.2.5	Delete " even if an array is declared **own** ".	
4.7.5	Add clause 4.7.5.6: " No call of the procedure itself may occur during the execution of the statements of the body of any procedure, nor during the evaluation of those of its actual parameters, the corresponding formal parameters of which are called by name, nor during the evaluation of expressions occurring in declarations inside the procedure ".	Recursive procedures and recursive use of procedures are not included.
5.4.4	Delete last sentence.	
3.5.1	Delete " | ⟨unsigned integer⟩ ".	Integer labels are not provided for.
3.5.5	Delete.	
5.4.5	Replace third sentence by: " Specifications of all formal parameters if any must be supplied".	Complete specification parts are required.
4.7.5.5	Replace by: " Kind and type of actual parameters must be the same as those of the corresponding formal parameters, if called by name ".	

Section or clause of Part I A	Subset definitions	Explanation
2.1	Delete: " | *A* | *B* | ... | *Y* | *Z* ". Delete: " , or extended ... delimiter) ". Add: " If a particular implementation requires capitals rather than small letters, one must regard them as a hardware representation for the small letters".	Only one case of letters is provided for.
2.4.3	Replace: " They may be chosen freely " by: " Identifiers may be chosen freely; but the effects due to the occurrence of two different identifiers the first six basic symbols of which are common are undefined ".	In the subset identifiers are differentiated only up to six leading basic symbols.
3.3.4	Replace the words: " the following rules " of the last sentence by: " a set of rules. However, if the type of an arithmetic expression according to the rules cannot be determined without evaluating an expression or ascertaining the type or value of an actual parameter, it is **real**. These rules are ".	In the subset the type of an arithmetic expression will be in certain cases **real** where it will be **integer** in ALGOL 60. Thus arithmetic will be less precise in some cases.
4.3.5	Delete.	The effect of a **go to** statement involving an undefined switch designator is undefined in the subset.
5.4.4	Add to text: " A function designator must be such that all its possible uses in the form of a procedure statement are equivalent to dummy statements".	
3.3.4.3	Insert between " ... rules " and " : " : " with the exception that, if both the basis *a* and the exponent *i* are of **integer** type, then the exponent has to be an unsigned integer, otherwise the result is undefined ".	Exponentiation with integer basis and exponent is restricted in the subset.
2.3	Delete: " | ÷ ".	The so-called integer division is not included in the subset.
3.3.1	Delete: " | ÷ ".	
3.3.5.1	Delete: " ÷ ".	

Section or clause of Part I A	Subset definitions	Explanation
3.3.4.2	Replace: " The operations ... both denote " by: " The operation ⟨term⟩/⟨factor⟩ denotes ". Delete last sentence.	
4.6.1	Replace: "⟨for clause⟩ ... **do** " by: "⟨for clause⟩ : := **for** ⟨variable identifier⟩ := ⟨for list⟩ **do** ".	The controlled variable in a for clause is restricted in the subset to be a variable identifier.
5.3.1	Replace: " ⟨switch list⟩ : := ⟨designational expression⟩ | ⟨switch list⟩, ⟨designational expression⟩ " by: "⟨switch list⟩ : := ⟨label⟩ | ⟨switch list⟩, ⟨label⟩ ".	In the subset the designational expressions in a switch list are restricted to be labels only.
3.5.1	Replace the last two formulae by: "⟨designational expression⟩ : := ⟨label⟩ | ⟨switch designator⟩ ".	In the subset only unconditional and unparenthesized designational expressions are provided for. See 5.3.1.
3.5.3	Delete: " In the general case ... is already found. " Replace " selects one of the designational expressions ... a recursive process " by: " selects one of the labels contained in the switch list of the switch declaration. The selection is obtained by counting these labels from left to right ".	
5.3.3	Replace: " These values ... its associated integer " by: " These values are given as labels entered in the switch list. With each of these labels is associated a positive integer *1*, *2*, ... obtained by counting the items in the list from left to right. The value of the switch designator corresponding to a given value of the subscript expression (see clause 3.5, " Designational expressions ") is the label in the switch list having this given value as its associated integer ".	
5.3.4	Delete.	

Section or clause of Part I A	Subset definitions	Explanation
5.3.5	Replace by: " Influence of scopes. If a switch designator occurs outside the scope of a label in the switch list, and an evaluation of this switch designator selects this label, then a possible conflict between the identifier used to denote this label and an identifier whose declaration is valid at the place of the switch designator will be avoided by a suitable change of this latter identifier ".	
4.7.3.2	Replace: " after enclosing this ... syntactically possible " by: " this actual parameter being an identifier, or string, otherwise the name replacement is undefined ".	In name replacement (call by name) the actual parameter can only be an identifier or a string.
4.7.5	Insert after: " ... ALGOL statement " and before " . " : " in the sense of this subset ".	
5	Insert after " ... any one block head " and before " Syntax ": " The identifier associated with a quantity declared in a declaration may not occur denoting that quantity more than once between the **begin** of the block in whose head that declaration occurs and the semicolon which ends that declaration, excepting the case where this occurrence is the occurrence of a procedure identifier in the left part list of an assignment statement in the sense of clause 5.4.4 ".	
5.4.3	Add: " No identifier may occur more than once in a formal parameter list ".	

PART II

SPECIFICATIONS OF INPUT-OUTPUT PROCEDURES

PART II A

PRIMITIVE INPUT-OUTPUT PROCEDURES FOR ALGOL 60

INTRODUCTION

It is recognized that some procedures to be used in connexion with input and output are considered as being primitives, which cannot be expressed otherwise than by means of a code body. Among these are the following ones:

insymbol
outsymbol
length
inreal
outreal
inarray
outarray

(1)

Apart from these primitives one needs in practice a fuller set of input-output procedures. However, the language ALGOL 60 is so flexible that different schemes of input-output procedures can be defined in it largely by means of the primitives mentioned above. A few examples of this will be given in section 3 of this Part.

1. DEFINITIONS

It is recommended that, if not otherwise declared, the identifiers (1) will be associated with procedures which transfer values between the variables of the program and values carried in any kind of foreign media not otherwise accessible from the program.

The corresponding procedure declarations are:

procedure *insymbol (channel, string, destination)*; **value** *channel*; **integer** *channel, destination*; **string** *string*; ⟨procedure body⟩

procedure *outsymbol (channel, string, source)*; **value** *channel, source*; **integer** *channel, source*; **string** *string*; ⟨procedure body⟩

integer procedure *length (string)*; **string** *string*; ⟨procedure body⟩

procedure *inreal (channel, destination)*; **value** *channel*; **integer** *channel*; **real** *destination*; ⟨procedure body⟩

procedure *outreal (channel, source)*; **value** *channel, source*; **integer** *channel*; **real** *source*; ⟨procedure body⟩

procedure *inarray (channel, destination)*; **value** *channel*; **integer** *channel*; **array** *destination*; ⟨procedure body⟩

procedure *outarray (channel, source)*; **value** *channel*; **integer** *channel*; **array** *source*; ⟨procedure body⟩

The procedure statements and the function designator calling these procedures must have the following forms:

insymbol (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨string⟩ ⟨parameter delimiter⟩ ⟨variable⟩)

outsymbol (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨string⟩ ⟨parameter delimiter⟩ ⟨arithmetic expression⟩)

length (⟨string⟩)
inreal (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨variable⟩)
outreal (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨arithmetic expression⟩)
inarray (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨array identifier⟩)
outarray (⟨arithmetic expression⟩ ⟨parameter delimiter⟩ ⟨array identifier⟩)

In all these cases, except for the call of length, the value of the first actual parameter must be a positive integer identifying an input or output channel available to the program.

2. ACTIONS OF THE PROCEDURE BODIES

The pair of procedures *insymbol* and *outsymbol* provides the means of communicating between foreign media and the variables of the program in terms of single basic symbols or any additional symbols. In either procedure the correspondence between the basic symbols and the values of variables in the program is established by mapping the sequence of the basic symbols given in the string supplied as the second parameter, taken in the order from left to right, onto the positive integers *1*, *2*, *3*, ... Using this correspondence the procedure *insymbol* will assign to the **integer** type variable given as the third parameter the value corresponding to the next basic symbol appearing on the foreign medium. If this next basic symbol does not appear in the string given as the second parameter, the value *0* will be assigned. If the next symbol appearing in the input is not a basic symbol of ALGOL 60 a negative integer, corresponding to the symbol, will be assigned.

Similarly the procedure *outsymbol* will transfer the basic symbol corresponding to the value of the third parameter to the foreign medium. If the value of the third parameter is negative a symbol corresponding to this value will be transferred. It is understood that where the foreign medium may be used both for *insymbol* and *outsymbol*, the negative integer values associated with each additional symbol will be the same for the two procedures. More generally, if additional symbols are used the corresponding values must be given as accompanying information with the program (see the footnote to section 1 of Part I A).

The type procedure *length* is introduced to enable the calculation of the length of a given (actual or formal) string to be made (see example *outstring*). The value of *length (s)* is equal to the number of basic symbols of the open string enclosed between the outermost string quotes.

The two procedures *inreal* and *outreal* form a pair. The procedure *inreal* will assign the next value appearing on the foreign medium to the **real** type variable given as the second parameter. Similarly, the procedure *outreal* will transfer the value of the second actual parameter to the foreign medium.

The representation of values on the foreign media will not be further described, except that it is understood that in so far as a medium can be used for both input and output a value which has been transferred to a given medium with the aid of a call of *outreal* will be represented in such a way that the same value, in the sense of numerical analysis (see clause 3.3.6), may be transferred back to a variable by means of procedure *inreal*, provided that an appropriate manipulation of the foreign medium has also been performed.

Procedures *inarray* and *outarray* also form a pair; they transfer the ordered set of numbers forming the value of the array given as the second parameter, the array bounds being defined by the corresponding array declaration rather than by additional parameters (the mechanism for doing that is already available in ALGOL 60 for the value call of arrays). The order in which the

elements of the array are transferred corresponds to the lexicographic order of the values of the subscripts, i.e.:

$$\left.\begin{array}{l} a\,[k_1, k_2, \ldots, k_m] \text{ precedes } a\,[j_1, j_2, \ldots, j_m] \\ \text{provided } k_i = j_i \quad (i = 1, 2, \ldots, p-1) \\ \qquad k_p < j_p \end{array}\right\} \quad (1 \leqslant p \leqslant m) \qquad (2)$$

It should be recognized that the possibly multidimensional structure of the array is not reflected in the corresponding numbers on the foreign medium, where they appear only as a linear sequence as defined by (2).

The representation of the numbers on the foreign medium conforms to the same rules as given for *inreal* and *outreal*; in fact it is possible for example to input numbers by *inreal* which before have been output by *outarray*.

3. EXAMPLES

```
procedure outboolean (channel, boolean); value boolean; integer channel; Boolean boolean; comment
        this procedure outputs a Boolean value as a basic symbol true or false;
        if boolean then outsymbol (channel, ' true ', 1)
                   else outsymbol (channel, ' false ', 1)

procedure outstring (channel, string); value channel; integer channel; string string; comment
        outputs the string string to the foreign medium;
        begin integer i;
            for i := 1 step 1 until length (string) do outsymbol (channel, string, i)
        end

procedure ininteger (channel, integer); value channel; integer channel, integer; comment inputs
        an integer which on the foreign medium appears as a sequence of digits, possibly preceded
        by a sign, and followed by a comma. Any other symbol in front of the sign is discarded ;

        begin integer n, k; Boolean b;
            integer := 0; b := true;
            for k := 1, k + 1 while n = 0 do insymbol
                (channel, ' 0123456789 - + ', n);
            if n = 11 then b := false; if n > 10 then n := 1;
            for k := 1, k + 1 while n ≠ 13 do
            begin integer := 10 × integer + n - 1;
                insymbol (channel, ' 012345678 - +, ', n)
            end 1;
            if ¬ b then integer := - integer
        end
begin
    begin array a [1 : 10];
        ⟨statements⟩;
        outarray (15, a)
    end;
    begin array b [0 : 1, 1 : 5];
        inarray (15, b);
        ⟨statements⟩
    end
end
```

The following example exhibits the use of inarray and outarray for inversion of a matrix including transfer of the matrix elements from and to the foreign medium. It requires that an appropriate declaration for a matrix inversion procedure as well as the declaration of outstring as given above are inserted at appropriate places in the program.

```
begin integer n;
      inreal (5, n); comment the matrix elements must be preceded by the order;
      begin array a [1 : n, 1 : n];
            inarray (5, a);
            matrix inversion (n, a, singular);
            outarray (15, a);
            go to ex
      end;
singular : outstring (15, ' singular ');
ex : end
```

4. CONCLUDING REMARKS

No further means for input-output operations are proposed in this Part but attention is drawn to reference [7], and to the extensive list of references at the end of that report.

PART II B

GENERAL INPUT-OUTPUT PROCEDURES FOR ALGOL 60

1. FORMATS

In this section a certain type of string, which specifies the format of quantities to be input or output, is defined, and its meaning is explained.

1.1 Number formats (see part I A, clause 2.5)

1.1.1 *Syntax*

BASIC COMPONENTS:

⟨replicator⟩ : := ⟨unsigned integer⟩ | *X*
⟨insertion⟩ : := *B* | ⟨replicator⟩ *B* | ⟨string⟩ | ⟨replicator⟩ ⟨string⟩
⟨insertion sequence⟩ : := ⟨empty⟩ | ⟨insertion sequence⟩ ⟨insertion⟩
⟨Z⟩ : := *Z* | ⟨replicator⟩ *Z*
⟨Z part⟩ : := ⟨Z⟩ | ⟨Z part⟩ ⟨Z⟩ | ⟨Z part⟩ ⟨insertion⟩
⟨D⟩ : := *D* | ⟨replicator⟩ *D*
⟨D part⟩ : := ⟨D⟩ | ⟨D part⟩ ⟨D⟩ | ⟨D part⟩ ⟨insertion⟩
⟨T part⟩ : := ⟨empty⟩ | *T* ⟨insertion sequence⟩
⟨sign part⟩ : := ⟨empty⟩ | ⟨insertion sequence⟩ + | ⟨insertion sequence⟩ −
⟨integer part⟩ : := ⟨Z part⟩ | ⟨D part⟩ | ⟨Z part⟩ ⟨D part⟩

FORMAT STRUCTURES:

⟨unsigned integer format⟩ : := ⟨insertion sequence⟩ ⟨integer part⟩
⟨decimal fraction format⟩ : := . ⟨insertion sequence⟩ ⟨D part⟩ ⟨T part⟩ |
V ⟨insertion sequence⟩ ⟨D part⟩ ⟨T part⟩
⟨exponent part format⟩ : := $_{10}$ ⟨sign part⟩ ⟨unsigned integer format⟩
⟨decimal number format⟩ : := ⟨unsigned integer format⟩ ⟨T part⟩ | ⟨insertion sequence⟩ ⟨decimal fraction format⟩ | ⟨unsigned integer format⟩ ⟨decimal fraction format⟩
⟨number format⟩ : := ⟨sign part⟩ ⟨decimal number format⟩ | ⟨decimal number format⟩ + ⟨insertion sequence⟩ | ⟨decimal number format⟩ − ⟨insertion sequence⟩ | ⟨sign part⟩ ⟨decimal number format⟩ ⟨exponent part format⟩

1.1.2 *Examples.* Examples of number formats appear in the table below.

Number format	Result from −13.296	Result from 1007.999
+ *ZZZDDD.DD*	− *013.30*	+ *1008.00*
+ *3Z3D.2D*	− *013.30*	+ *1008.00*
− *3D2B3D.2DT*	− *000 013.29*	*001 007.99*
5Z.5D −	*13.29600* −	*1007.99900*
'*integer*␣*part*␣' − *4 ZV*',	*integer part* − *13, fraction*	*integer part 1007, fraction*
␣*fraction*' *B3D*	*296*	*999*
− . $5D_{10}$ + *2D* '...'	− $.13296_{10}$ + *02* ...	$.10080_{10}$ + *04* ...
+ $ZD_{10}2Z$	− *13*	+ 10_{10} *2*
+ $D.DDBDDBDDB_{10}$ + *DD*	− $1.32\ 96\ 00_{10}$ + *01*	+ $1.00\ 79\ 99_{10}$ + *03*
$XB.XD_{10}$ − *DDD*	(undefined)	(depends on call)

1.1.3 *Semantics.* The above syntax defines those strings which can comprise a " number format ". First the interpretation to be taken during output is described:

1.1.3.1 REPLICATORS. An unsigned integer *n* used as a replicator means that the quantity following it is repeated *n* times; thus *3B* is equivalent to *BBB*. The character *X* as a replicator means a number of replications which will be specified when the format is called (see clause 2.3.1).

1.1.3.2 INSERTIONS. Strings, delimited by string quotes, may be inserted anywhere within a number format. The proper string (without the outermost string quotes) will appear inserted in the same place with respect to the rest of the number. Similarly, the letter *B* may be inserted anywhere within a number format, and it stands for a blank space.

1.1.3.3 SIGN AND ZERO SUPPRESSION. The portion of a number format to the left of the decimal point consists of an optional sign, then either or both of a sequence of *Z*'s and a sequence of *D*'s, with possible insertion characters.

The convention on signs, whether preceding or following the number, is as follows:

(*a*) if no sign appears in the format, the number is assumed to be positive, and the treatment of negative numbers is undefined;

(*b*) if a plus sign appears in the format, the sign of the number to be output will appear as + or − on the external medium;

(*c*) if a minus sign appears in the format, the sign will appear if the value of that number is negative, and will be suppressed, i.e. replaced by a blank space, if the value of the number is positive.

The letter *Z* stands for zero suppression, and the letter *D* stands for digit printing without zero suppression. Each *Z* and *D* stands for a single digit position; a zero digit specified by *Z* will be suppressed when all digits to its left are zero. A digit specified by *D* will always be printed. Note that the number zero printed with all *Z*'s in the format will give rise to all blank spaces, so at least one *D* should usually be given somewhere in the format.

Whenever zero suppression takes place, the sign (if any) is printed in place of the rightmost character suppressed.

1.1.3.4 DECIMAL POINTS. The position of the decimal point is indicated either by the character " . " or by the letter " *V* ". In the former case, the decimal point appears on the external medium; in the latter case, the decimal point is " implied ", i.e. it takes up no space on the external medium. Only *D*'s (no *Z*'s) may appear to the right of the decimal point except in an exponent part.

1.1.3.5 TRUNCATION. On output, non-integral numbers are usually rounded to fit the format specified. If the letter *T* is used, however, truncation takes place instead. Rounding and truncation of a number *X* to *d* decimal places are defined as follows:

Rounding: $10 \uparrow (-d) \times entier\ (10 \uparrow d \times X + .5)$
Truncation: $10 \uparrow (-d) \times sign\ (X) \times entier\ (10 \uparrow d \times abs\ (X))$

1.1.3.6 EXPONENT PART. The number following a " $_{10}$ " is treated exactly the same as the portion of a number to the left of a decimal point (clause 1.1.3.3), except that if the " *D* part " of the exponent is empty, i.e. no *D*'s appear, and if the exponent is zero, then the " $_{10}$ " and the sign are suppressed.

1.1.3.7 TWO TYPES OF NUMERIC FORMAT. Number formats are of two principal kinds:

(*a*) Decimal number with no exponent. In this case, the number is aligned according to the decimal point with the picture in the format, and it is then truncated or rounded to the appropriate number of decimal places. The sign may precede or follow the number.

(*b*) Decimal number with exponent. In this case, the number is transformed into the format of the decimal number with its most significant digit non-zero; the exponent is adjusted accordingly. If the number is zero, both the decimal part and the exponent part are output as zero.

If in case (a) the number is too large to be output in the specified form, or if in case (b) the exponent is too large, an overflow error occurs. The action which takes place on overflow is undefined.*

1.1.3.8 INPUT. A number input with a particular format specification should in general be the same as the number which would be output with the same format, except that less error checking occurs. The rules are, more precisely:

(*a*) Leading zeros may appear even though *Z*'s are used in the format. Leading spaces may appear even if *D*'s are used. In other words, no distinction between *Z* and *D* is made on input.

(*b*) Insertions take the same amount of space in the same positions, but the characters appearing there are ignored on input. In other words, an insertion specifies only the number of characters to ignore, when it appears in an input format.

(*c*) If the format specifies a sign at the left, the sign may appear in any *Z* or *D* position as long as it is to the left of the number. A sign specified at the right must appear in place.

(*d*) The following are checked: the positions of decimal points, "$_{10}$", and the presence of digits in place of *D* or *Z* after the first significant digit. If an error is detected in the data, the result is undefined.**

1.2 Other formats

1.2.1 *Syntax*

⟨S⟩ : := *S* | ⟨replicator⟩ *S*
⟨string format⟩ : := ⟨insertion sequence⟩ ⟨S⟩ | ⟨string format⟩ ⟨S⟩ | ⟨string format⟩ ⟨insertion⟩
⟨alpha format⟩ : := *A*
⟨standard format⟩ : := *N*
⟨nonformat⟩ : := *I* | *R* | *L*
⟨Boolean part⟩ : := *P* | *F*
⟨Boolean format⟩ : := ⟨insertion sequence⟩ ⟨Boolean part⟩ ⟨insertion sequence⟩
⟨title format⟩ : := ⟨insertion⟩ | ⟨title format⟩ ⟨insertion⟩
⟨alignment mark⟩ : := / | ↓ | *J* | ⟨replicator⟩ / | ⟨replicator⟩ ↓ | ⟨replicator⟩ *J*

* It is recommended that the number of characters used in the output be the same as if no overflow had occurred, and that as much significant information as possible be output (e.g. exponent increased by one digit and attempt to output the number again).

** It is recommended that the input procedure attempt to reread the data as if it were in standard format (clause 1.2.3.7) and also to give some error indication compatible with the system being used. Such an error indication might be suppressed at the programmer's option if the data became meaningful when it was reread in standard format.

⟨format item 1⟩ : := ⟨number format⟩ | ⟨string format⟩ | ⟨alpha format⟩ | ⟨nonformat⟩ | ⟨Boolean format⟩ | ⟨title format⟩ | ⟨standard format⟩ | ⟨alignment mark⟩ ⟨format item 1⟩

⟨format item⟩ : := ⟨format item 1⟩ | ⟨alignment mark⟩ | ⟨format item⟩ ⟨alignment mark⟩

1.2.2 *Examples*

```
↟ 5Z . 5D/ / /
3S ' = ' 6S4B
/A J J
↟ R
P
/ ' Execution. ' ↟
```

1.2.3 *Semantics*

1.2.3.1 STRING FORMAT. A string format is used for output of string quantities. Each of the S-positions in the format corresponds to a single basic symbol in the string which is output. If the string is longer than the number of *S*'s, the leftmost symbols are transferred; if the string is shorter, " ␣ " symbols are effectively added at the right of the string.

1.2.3.2 ALPHA FORMAT. The letter *A* means that one symbol is to be transmitted; this is the same as S-format, except that the ALGOL equivalent of the basic symbol is of integer type rather than a string. The translation between the external and internal codes will vary from one machine to another, hence the results of arithmetic operations and relations other than " = " and " ≠ " will be machine-dependent.

A programmer may work with these alphabetic quantities in a machine-independent manner by using the transfer function *equiv (S)* where *S* is a string consisting of one basic symbol; the value of *equiv (S)* is of integer type, and it is defined to have exactly the same value as if the string *S* had been input using alpha format. For example, one may write

if *X* = *equiv* (' *A* ') **then go to** *PROCESS ALPHA* ;

where the value of *X* has been input using the format " *A* ".

1.2.3.3 NONFORMAT. An *I*, *R* or *L* is used to indicate that the value of a single variable of integer, real, or Boolean type, respectively, is to be input or output from or to an external medium, using the internal machine representation. If a value of integer type is output with R-format or if a value of real type is input with I-format, the appropriate transfer function is invoked, i.e. the *I* or *R* specifies the format as it appears on the external medium. The precise behaviour of this format, and particularly its interaction with other formats, is undefined in general.

1.2.3.4 BOOLEAN FORMAT. When Boolean quantities are input or output, the format *P* or *F* must be used. The correspondence is defined as follows:

Internal to ALGOL	*P*	*F*
true	*1*	**true**
false	*0*	**false**

On input, anything failing to be in the proper form is undefined.

1.2.3.5 TITLE FORMAT. All formats discussed so far have given a correspondence between a single ALGOL real, integer, Boolean, or string quantity and a number of characters in the input or output. A format item containing a title format consists entirely of insertions and optional alignment marks, and so it does not require a corresponding ALGOL quantity. On input, it causes skipping of the characters, and on output it causes the emission of the insertion symbols it contains. (If titles are to be input, alpha format should be used; see clause 1.2.3.2.)

1.2.3.6 ALIGNMENT MARKS. The characters " / ", " ↟ ", and " *J* " in a format item indicate line, page and character control actions. The precise definition of these actions will be given later (see clause 2.5.4.1); they have the following intuitive interpretation:

(*a*) " / " means: go to next line, in a manner similar to the carriage return operation on a typewriter;

(*b*) " ↟ " means: do a " / " operation and then skip to the top of the next page;

(*c*) " *J* " means: skip the character pointer to the next " tabulation " position, similar to the " tab " operation on a typewriter.

Two or more alignment marks indicate the number of times the operations are to be performed; for example, " // " on output means that the current line is completed and the next line is effectively set to all blanks. Alignment marks at the left of a format item cause actions to take place before the regular format operation, and alignment marks at the right of a format item cause actions to take place after the regular format operations.

1.2.3.7 " STANDARD " FORMAT. There is a format available without specification (see clause 2.5.4) which has the following characteristics:

(*a*) On input, any number written according to the ALGOL syntax for ⟨number⟩ is accepted with the conventional meaning. These are of arbitrary length, and they are delimited at the right by the following conventions:

(I) A letter or character other than a decimal point, sign, digit, or " $_{10}$ " occurring to the right of a decimal point, sign, digit, or " $_{10}$ " is a delimiter.

(II) A sequence of k blank spaces serves as a delimiter as in (I); a sequence of less than k blank spaces is ignored. This number $k \geqslant 1$ has an implementation defined initial value and may be interrogated or modified by a suitable call on the procedure *sysparam* (see clause 2.5.6).

(III) If the number contains a decimal point, sign, digit, or " $_{10}$ " on the line where the number begins, the right-hand margin of that line serves as a delimiter of the number. However, the right-hand margin does not serve as a delimiter in the case where the only characters remaining on the line are spaces or characters which do not enter into the number. In this case the only delimiters for this number are those specified in (I) or (II) above. (See clause 2.5.4.2 for further discussion of standard input format.)

(*b*) On output a number of **real** type is given in the form of a decimal number with an exponent. A number of **integer** type is given in the form of an integer.

These numbers must be suitable for input under standard format (i.e. followed by not less than *k* blanks).

Standard format may be invoked by the format item *N*, through the exhaustion of the format string, or by specifying an empty format string.

1.3 Format strings

The format items mentioned above are combined into format strings according to the rules in this clause.

1.3.1 *Syntax*

⟨format primary⟩ : := ⟨format item⟩ | ⟨replicator⟩ (⟨format secondary⟩) | (⟨format secondary⟩)
⟨format secondary⟩ : := ⟨format primary⟩ | ⟨format secondary⟩, ⟨format primary⟩
⟨format string⟩ : := ' ⟨format secondary⟩ ' | ' '

1.3.2 *Examples*

' 4 (15ZD), / / '
' ↟ '
' .5D_{10} + D, X (2(20B.8D_{10} + D), 10S) '
" ... This ␣ is ␣ a ␣ peculiar ␣ ' format string ' "

1.3.3 *Semantics.* A format string is a list of format items, which are to be interpreted from left to right. The construction: " ⟨replicator⟩ (⟨format secondary⟩) " denotes " replicator " repetitions of the parenthesized quantity (see clause 1.1.3.1). The construction: " (⟨format secondary⟩) " denotes an infinite repetition of the parenthesized quantity.

Spaces within a format string, except those which are part of insertion substrings, are irrelevant.

1.4 Summary of format codes

A - basic symbols represented as integers
B - blank space
D - digit.
F - representation of Boolean value in the form **true** or **false**
I - integer untranslated
J - tab function
L - Boolean untranslated
N - standard format
P - representation of Boolean value in the form *1* or *0*
R - real untranslated
S - string character
T - truncation
V - implied decimal point
X - variable replicator
Z - zero suppression
\+ - unconditional sign indicator
− - positive sign suppression indicator
10 - exponent part indicator
( ) - delimiters of replicated format secondaries
, - format primary separator
/ - line alignment
↟ - page alignment
' ' - string delimiters
. - decimal point

2. STANDARD PROCEDURES FOR INPUT AND OUTPUT

Certain identifiers should be reserved for the standard functions of input and output. In a particular representation these procedures may be available without explicit declaration. In either case if the identifier is declared in a block head as something different (for example, an array), the function it represents will be unavailable throughout that block.

2.1 General characteristics

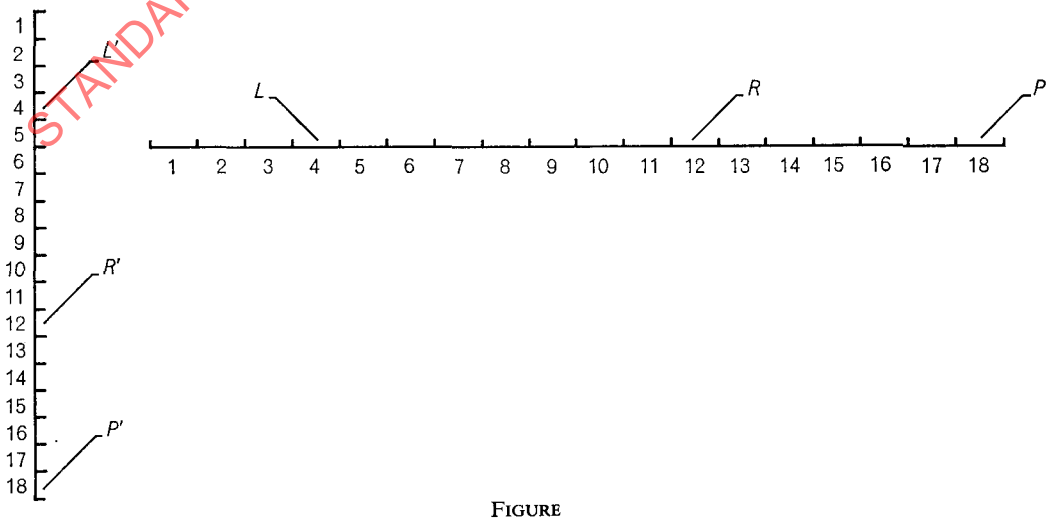
The term *CHANNEL* is utilized throughout to represent one or more foreign media, or a portion of such a medium. It is assumed that the characters on such a channel form a linearly ordered set, and that after suitable repositioning, information can be read from the medium in the same order it was written. It is further assumed that there need not be a one-to-one correspondence between basic symbols which exist in the program, and characters which represent them on the medium. For many basic symbols it is expected that a multiplicity of characters will be required on particular media. These one-to-many and many-to-one transformations are assumed to go on " behind the scenes ", and these transformations may be different from implementation to implementation, and medium to medium. Indeed, for some media this transformation may not be defined for some basic symbols, in which case the result of attempting to input or output such a symbol is undefined. The logical behaviour of these procedures, however, should be machine independent.

2.2 Horizontal and vertical control

This clause deals with the way in which the sequence of basic symbols described by the rules of formats in section 1 is mapped onto input and output devices. This is done in a manner which is essentially independent of the device being used, in the sense that with these specifications the programmer can anticipate how the input or output data will appear on virtually any device. Some of the features of this description will, of course, be more appropriately used on certain devices than on others.

The discussion assumes that data is to be output to a printer, then shows the manner in which other devices fit into the same general framework.

The page format is controlled by specifying the horizontal and the vertical lay-out. Horizontal lay-out is controlled in essentially the same manner as vertical lay-out, and this symmetry between the horizontal and vertical dimensions should be kept in mind for easier understanding of the concepts in this clause.



FIGURE

Referring to the Figure, the horizontal format is described in terms of three parameters (L, R, P), and the vertical format has corresponding parameters (L', R', P'). The parameters L, L' and R, R' indicate left, top, right and bottom margins, respectively. The Figure shows a case where $L = L' = 4$ and $R = R' = 12$. Only positions L through R of a horizontal line are used, and only lines L' through R' of the page are used; it is required that $1 \leqslant L \leqslant R$ and $1 \leqslant L' \leqslant R'$. The parameter P is the number of characters per line, and P' is the number of lines per page. Although L, R, L' and R' are chosen by the programmer, the values of P and P' are characteristics of the device and they are usually out of the programmer's control. For those devices on which P and P' can vary (for example, some printers have two settings, one on which there are 66 lines per page, and another on which there are 88), the values are specified to the system by a suitable call on the procedure sysparam (see clause 2.5.6). For certain devices, values of P or P' might be essentially infinite.

Although the Figure shows a case where $P \geqslant R$ and $P' \geqslant R'$, it is of course quite possible that $P < R$ or $P' < R'$ (or both) might occur, since P and P' are in general unknown to the programmer. In such cases, the algorithm described in clause 2.5.4 is used to break up logical lines which are too wide to fit on a physical line, and to break up logical pages which are too large to fit a physical page. On the other hand, the conditions $L \leqslant P$ and $L' \leqslant P'$ are insured by setting L or L' equal to 1 automatically if they happen to be greater than P or P', respectively.

Characters determined by the output values are put onto a horizontal line; there are three conditions which cause a transfer to the next line:

(a) normal line alignment, specified by a " / " in the format;

(b) R-overflow, which occurs when a group of characters is to be transmitted, or a tabulation is specified, which would pass position R, and

(c) P-overflow, which occurs when a group of characters is to be transmitted, or a tabulation is specified, which would not cause R-overflow but would pass position P.

When any of these things occurs, control is transferred to a procedure specified by the programmer in case special action is desired (for example, a change of margins in case of overflow; see clause 2.3.3).

Similarly, there are three conditions which cause a transfer to the next page:

(a') normal page alignment, specified by a " ↓ " in the format;

(b') R'-overflow, which occurs when a group of characters is to be transmitted which would appear on line $R' + 1$; and

(c') P'-overflow, which occurs when a group of characters is to be transmitted which would appear on line $P' + 1 < R' + 1$. The programmer may indicate special procedures to be executed at this time if he wishes, for example, to insert a page heading, etc.

Tabulation is controlled by a " J " in the format. This causes the character pointer to be advanced to the next " TAB " position, with intermediate positions being filled with blanks. The tabulation spacing for the medium may be specified external to the ALGOL system, or through a suitable call on tabulation (see clause 2.3.4).

If the tabulation spacing is N, then the first character positions of tabulation fields would be:

$$L,\ L + N,\ L + 2N,\ \ldots,\ L + KN \qquad \text{where} \qquad L + KN \leqslant min\ (P,\ R)$$

Further details concerning pages and lines will be given later. Now consideration will be given to how devices other than printers can be thought of in terms of the ideas above.

A typewriter is very much like a printer, and it requires no further comment.

Punched cards with, for example, 80 columns have $P = 80$ and $P' = \infty$. Vertical control would appear to have little meaning for punched cards, although an implementation might choose to interpret " ↓ " to mean the insertion of a coded or blank card.

With paper tape again, vertical control has little or no meaning; in this case, P could be the number of characters read or written at a time.

On magnetic tape capable of writing arbitrarily long blocks, $P = P' = \infty$. Each page might be thought of as being a " record ", i.e., an amount of contiguous information on the tape which is read or written at once. The lines are subdivisions of a record, and R' lines form a record; R characters are in each line. In this way, so-called " blocking of records " can be specified. Other interpretations might be more appropriate for magnetic tapes at certain installations, for example, a format which would correspond exactly to printer format for future off-line listing, etc.

These examples are given to indicate how the concepts described above for printers can be applied to other devices. Each implementation will decide on appropriate methods for particular devices, and if there are choices to be made they can be given by the programmer by some means. The manner in which this is done is of no concern to this Recommendation.

2.3 Lay-out procedures

Whenever input or output is done, certain " standard " operations are assumed to take place, unless otherwise specified by the programmer. Therefore one of the parameters of the input or output procedure is a " lay-out " procedure, which specifies all of the non standard operations desired. This is achieved by using any or all of the seven " descriptive procedures " *format*, *h end*, *v end*, *h lim*, *v lim*, *tabulation*, or *no data* described in this clause.

The precise action of these procedures can be described in terms of the fictitious concept of seven " hidden variables ", H1, H2, H3, H4, H5, H6, and H7. The effect of each descriptive procedure is to set one of these variables to a certain value; and that may be regarded as the total effect of a descriptive procedure. The programmer normally has no access to these hidden variables other than particular calls on *sysparam*. The hidden variables have a scope which is local to *inlist* and to *outlist*.

2.3.1 *Format procedures.* The descriptive procedure call:

format (STRING)

has the effect of setting the hidden variable H1 to indicate the string parameter. This parameter may be either a string explicitly written or a formal parameter; but in any event, the string it refers to must be a format string which satisfies the syntax of clause 1.3, and it must have no " *X* " replicators.

The procedure *format* is just one of a class of ten procedures which have the names *format n* ($n = 0, 1, \ldots, 9$). The name *format* is equivalent to *format 0*. In general, the procedure *format n* is used with format strings which have exactly n *X*-replicators. The call is

format n ($STRING, X_1, X_2, \ldots, X_n$)

where each X_i is an integer parameter called by value. The effect is to replace each *X* of the format string by one of the X_i, with the correspondence defined from left to right. Each X_i must be non-negative.

For example,

format 2 ('XB.XD$_{10}$ + DD', 5, 10)

is equivalent to

format ('5B.10D$_{10}$ + DD')

2.3.2 *Limits.* The descriptive procedure call

h lim (L, R)

has the effect of setting the hidden variable H2 to indicate the two parameters L and R. Similarly,

v lim (L', R')

sets H3 to indicate L' and R'. These parameters have the significance described in section 2.2. If *h lim* and *v lim* are not used, $L = L' = 1$ and $R = R' = \infty$. If $L > P$, the value *1* is substituted for L; similarly, if $L' > P'$, the value *1* is substituted for L'.

2.3.3 *End control.* The descriptive procedure calls:

h end (P_N, P_R, P_P);
v end ($P_{N'}$, $P_{R'}$, $P_{P'}$);

have the effect of setting the hidden variables H4 and H5, respectively, to indicate their parameters. The parameters P_N, P_R, P_P, $P_{N'}$, $P_{R'}$, $P_{P'}$, are names of procedures (ordinarily dummy procedures if *h end* and *v end* are not specified) which are activated in case of normal line alignment, R-overflow, P-overflow, normal page alignment, R'-overflow, and P'-overflow, respectively.

2.3.4 *Tabulation.* The descriptive procedure call:

tabulation (N)

has the effect of setting the hidden variable H6 to indicate the parameter N. Here N is the width of the tabulation field, measured in number of characters of the foreign medium. (See clause 2.5.4.1, process C.) If tabulation is not called then the tabulation field width is *1*.

2.3.5 *End of data.* The descriptive procedure call:

no data (L)

has the effect of setting the hidden variable H7 to indicate the parameter L. Here L is a label. End of data as defined here has meaning only on input, and it does not refer to any specific hardware features; it occurs when data is requested for input but no more data remains on the corresponding input medium. At this point, a transfer to the statement labelled L will occur. If the procedure *no data* is not used, transfer will occur to a " label " which has effectively been inserted just before the final **end** in the ALGOL program, thus terminating the program*.

2.3.6 *Examples*

A lay-out procedure might look as follows:

```
procedure LAYOUT;
begin format ('/');
    if B then
        begin format 1 ('XB', Y + 10); no data (L32) end;
    h lim (if B then 1 else 10, 30)
end
```

Note that lay-out procedures never have parameters; this procedure, for example, refers to three global quantities, B, Y and *L32*. Suppose Y has the value *3*, then this lay-out accomplishes the following:

* In this case, an implementation might provide an appropriate error comment.

Hidden variable	Procedure	If *B* = **true**	If *B* = **false**
H1	*format*	*' 13B '*	*'/'*
H2	*h lim*	*(1, 30)*	*(10, 30)*
H3	*v lim*	*(1, ∞)*	*(1, ∞)*
H4	*h end*	*(, ,)*	*(, ,)*
H5	*v end*	*(, ,)*	*(, ,)*
H6	*tabulation*	*1*	*1*
H7	*no data*	*L32*	end of program

As a more useful example, a procedure *LAYOUT* can be written so that the horizontal margins *(11, 110)* are used on the page, except that if *P*-overflow or *R*-overflow occurs the margins *(16, 105)* are to be used for overflow lines.

procedure *LAYOUT ;*
begin *format 1 (' 4, (X(BB-ZZZZ.DD),/ /)', N) ;*
h lim (11, 110) ; h end (K, L, L)
end *;*
procedure *K; h lim (11, 110);*
procedure *L; h lim (16, 105);*

This causes the limits *(16, 105)* to be set whenever overflow occurs, and the " / " in the format will reinstate the original margins when it causes procedure *K* to be called. (If the programmer wishes a more elaborate treatment of the overflow case, depending on the value of *P*, he may do this using the procedure of clause 2.5.6.)

2.4 List procedures

2.4.1 *General characteristics.* The purpose of a list procedure is to describe a sequence of items which is to be transmitted for input or output. A procedure is written in which the name of each item *V* is written as the argument of a procedure, say *ITEM*, thus: *ITEM (V)*. When the list procedure is called by an input-output system procedure, another procedure (such as the internal system procedure *outitem*) will be " substituted " for *ITEM*, *V* will be called by name, and the value of *V* will be transmitted for input or output. The standard sequencing of ALGOL statements in the body of the list procedure determines the sequence of items in the list.

A simple form of list procedure might be written as follows:

procedure *LIST (ITEM);*
begin *ITEM (A) ; ITEM (B) ; ITEM (C)*
end;

which says that the values of *A*, *B* and *C* are to be transmitted. A more typical list procedure might be:

procedure *PAIRS (ELT) ;*
for *I* := **step** *1* **until** *N* **do**
begin *ELT (A[I]) ; ELT (B[I])* **end**;

This procedure says that the values of the list of items *A*[*1*], *B*[*1*], *A*[*2*], *B*[*2*], ..., *A*[*N*], *B*[*N*] are to be transmitted, in that order. Note that if $N \leqslant 0$ no items are transmitted at all.

The parameter of the " item " procedure (i.e. the parameter of *ITEM* or *ELT* in the above examples) is called by name. It may be an arithmetic expression, a Boolean expression, or a string, in accordance with the format which will be associated with the item. Any of the ordinary features of ALGOL may be used in a list procedure, so there is great flexibility.

A list procedure is executed step by step with the input or output procedure, with control transferring back and forth. This is accomplished by special system procedures such as *initem* and *outitem* which are " interlaced " with the list procedure, as described in clause 2.5.4. The list procedure is called with *initem* (or *outitem*) as actual parameter, and whenever this procedure is called within the list procedure, the actual input or output is taking place. Through the interlacing, special format control, including the important device-independent overflow procedures, can take place during the transmission process. Note that a list procedure may change the hidden variables by calling a descriptive procedure; this can be a valuable characteristic, for example, when changing the format, based on the value of the first item which is input.

2.5 Input-output calls

Here procedures are described which cause the actual transmission of information between a foreign medium and the variables of the program.

2.5.1 *Symbol transmission.* The procedure calls:

insymbol (CHANNEL, STRING, DESTINATION)
outsymbol (CHANNEL, STRING, SOURCE)

where *CHANNEL* and *SOURCE* must be arithmetic expressions called by value, *STRING* is a string, and *DESTINATION* is an integer variable called by name, provide the means of communicating between foreign media and the variables of the program in terms of single basic symbols or any additional symbols. In either procedure the correspondence between the basic symbols and the values of variables in the program is established by mapping the sequence of the basic symbols given in the string supplied as the second parameter, taken in the order from left to right, into the positive integers *1*, *2*, *3*, ... Using this correspondence the procedure *insymbol* will assign to an **integer** type variable given as the third parameter the value corresponding to the next basic symbol appearing on the foreign medium. If this next basic symbol does not appear in the string given as the second parameter, the number *0* will be assigned. If the next symbol appearing in the input is not a basic symbol of ALGOL 60, a negative integer, corresponding to the symbol, will be assigned.

Similarly the procedure *outsymbol* will transfer the basic symbol corresponding to the value of the third parameter to the foreign medium. If the value of the third parameter is negative a symbol corresponding to this value will be transferred. It is understood that where the foreign medium may be used both for *insymbol* and *outsymbol*, the negative integer values associated with each additional symbol will be the same for the two procedures. More generally, if additional symbols are used the corresponding values must be given as accompanying information with the program.

2.5.2 *Transmission of* **real** *type.* Transmission of information of **real** type between variables of the program and a foreign medium may be accomplished by the procedure calls:

inreal (CHANNEL, DESTINATION)
outreal (CHANNEL, SOURCE)

where *CHANNEL* and *SOURCE* are arithmetic expressions and *DESTINATION* is a variable of **real** type.

The two procedures *inreal* and *outreal* form a pair. The procedure *inreal* will assign the next value appearing on the foreign medium to the **real** type variable given as the second parameter. Similarly, procedure *outreal* will transfer the value of the second actual parameter to the foreign medium.

The representation of values on the foreign media will not be further described, except that it is understood that in so far as a medium can be used for both input and output a value which has been transferred to a given medium with the aid of a call of *outreal* will be represented in such a way that the same value, in the sense of numerical analysis (see Part I A, clause 3.3.6), may be transferred back to a variable by means of procedure *inreal*, provided that an appropriate manipulation of the foreign medium has also been performed.

2.5.3 *Transmission of arrays.* Arrays may be transferred from and to a foreign medium by means of the procedure calls:

inarray (CHANNEL, DESTINATION)
outarray (CHANNEL, SOURCE)

where *CHANNEL* must be an arithmetic expression and *DESTINATION* and *SOURCE* are arrays of **real** type.

Procedures *inarray* and *outarray* also form a pair; they transfer the ordered set of numbers forming the value of the array given as the second parameter, the array bounds being defined by the corresponding array declaration rather than by additional parameters (the mechanism for doing that is already available in ALGOL 60 for the value call of arrays).

The order in which the elements of the array are transferred corresponds to the lexicographic order of the values of the subscripts, i.e.

$$\begin{aligned} &a\,[k_1, k_2, \ldots, k_m] \text{ precedes} \\ &a\,[j_1, j_2, \ldots, j_m] \text{ provided} \\ &k_i = j_i \qquad (i = 1, 2, \ldots, p-1) \\ &\text{and} \quad k_p < j_p \qquad (1 \leqslant p \leqslant m) \end{aligned} \tag{1}$$

It should be recognized that the possibly multidimensional structure of the array is not reflected in the corresponding numbers on the foreign medium where they appear only as a linear sequence as defined by (1).

The representation of the numbers on the foreign medium conforms to the same rules as given for *inreal* and *outreal*; in fact it is possible, for example, to input numbers by *inreal* which before have been output by *outarray*.

2.5.4 *Formatted input-output calls.* A set of procedures exists to accomplish formatted input or output as specified by a format string (see section 1). The detailed behaviour of these procedures is described below.

2.5.4.1 OUTPUT. An output process is initiated by the call:

outlist (CHANNEL, LAYOUT, LIST)

Here *CHANNEL* is an integer parameter called by value, which is the number associated with a foreign medium. The parameter *LAYOUT* is the name of a lay-out procedure (clause 2.3), and *LIST* is the name of a list procedure (clause 2.4).

There is also another class of procedures, named *output n*, for $n = 0, 1, 2, ..., 9$, which is used for output as follows:

output n (CHANNEL, FORMAT STRING, E_1, E_2, ..., E_n)

Each of these latter procedures can be defined in terms of *outlist* as follows:

procedure *output n (CHANNEL, FORMAT STRING, E_1, E_2, ..., E_n)*;
begin procedure *A*; *format (FORMAT STRING)*;
    procedure *B (P)*;
    begin *P(E_1)* ; *P(E_2)* ; ... ; *P(E_n)* **end**;
    outlist (CHANNEL, A, B)
end;

In the following rules it is therefore assumed that *outlist* has been called.

Let the variables ρ and ρ' indicate the current position in the output for the channel under consideration, i.e. lines *1*, *2*, ..., ρ' of the current page have been completed, as well as character positions *1*, *2*, ..., ρ of the current line (i.e. of line $\rho' + 1$). At the beginning of the program, $\rho = \rho' = 0$. The symbols *P* and *P'* denote the line size and page size (see clause 2.2). Output takes place according to the following algorithm:

Step 1 (Initialization)

The hidden variables are set to standard values:

H 1 is set to the " standard " format ' '.
H 2 is set so that $L = 1$, $R = \infty$.
H 3 is set so that $L' = 1$, $R' = \infty$.
H 4 is set so that P_N, P_R, P_P are all effectively equal to the *DUMMY* procedure defined as follows:
    " **procedure** *DUMMY*; ;".

H5 is set so that $P_{N'}$, $P_{R'}$, $P_{P'}$ are all effectively equal to *DUMMY*.
H 6 is set so that *TAB* = *1*.

Step 2 (Lay-out)

The lay-out procedure is called; this may change some of the variables H1, H2, H3, H4, H5, H6. Set *T* to **false**. (*T* is a Boolean variable used to control the sequencing of data with respect to title formats; when *T* is **true** a value which has been transmitted to the procedure has not yet been output.)

Step 3 (Communication with list procedure)

The next format item of the format string is examined. (After the format string is exhausted, " standard " format, clause 1.2.3.7, is used from then on until the end of the procedure.)

If the next format item is a title format, i.e. requires no data item, proceed directly to step 4. If *T* is **true** proceed to step 4. Otherwise, the list procedure is activated; this is done the first time by calling the list procedure, using as actual parameter a procedure named *outitem*; this is done on all subsequent times by returning from the procedure *outitem*, which will cause the list procedure to be continued from the latest *outitem* call. (The identifier *outitem* has scope local to *outlist*, so a programmer may not call this procedure directly.)

After the list procedure has been activated in this way, it will either terminate or call the procedure *outitem*. In the former case, the output process is completed;

in the latter case, T is set to **true** and any assignments to hidden variables that the list procedure may have invoked will cause adjustment to the variables H1, H2, H3, H4, H5, H6. Continue at step 4.

Step 4 (Alignment marks)

If the next format item includes alignment marks at its left, remove it from the format string and execute process A for each " / ", process B for each " ↟ ", and process C for each " J ". Note that overflow procedures may cause the format strings to be changed. In such cases, the new format string is not examined before step 6.

Step 5 (Get within margins)

Execute process G to ensure proper page and line alignment.

Step 6 (Formatting the output)

Take the next item from the format string. (In unusual cases, the list procedure or an overflow procedure may have called the descriptive procedure *format*, thereby changing the format string. In such cases, the new format string is examined from the beginning; it is conceivable that the format items examined in steps 3, 4 and 6 might all be different. At this point the current format item is effectively removed from the format string and copied elsewhere. The format string itself, possibly changed by further calls of *format*, will not be interrogated until the next occurrence of step 3.)

Alignment marks at the left of the format item are ignored. If the format item is not composed only of alignment marks and insertions, the value of T is examined.

If T is **false**, undefined action takes place. (A nontitle format was substituted for a title format in an overflow procedure, and this is not allowed.) Otherwise, the output item is evaluated and T is set to **false**. Now the rules of format are applied and the characters X_1, X_2 ... X_s which represent the formatted output on the external medium are determined. (Note that the number of characters, s, may depend on the value being output using " A " or " S " format, as well as on the output medium.)

Step 7 (Check for overflow)

If $\rho + s \leqslant R$ and $\rho + s \leqslant P$, where s is the size of the item as determined in step 6, the item will fit on this line, so continue at step 9. Otherwise, if the present item uses " A " format, output a special symbol, which is recognizably not a basic symbol; this is done to ensure that input will be inverse to output. Go to step 8.

Step 8 (Processing of overflow)

Perform process H ($\rho + s$). Then if $\rho + s \leqslant R$ and $\rho + s \leqslant P$, go to step 9; otherwise let $k = min\ (R, P) - \rho$. Output $X_1\ X_2$ ... X_k, set $\rho = min\ (R, P)$, and then let $X_1\ X_2$ ... $X_{s-k} = X_{k+1}\ X_{k+2}$ ... X_s. Decrease s by k and repeat step 8.

Step 9 (Finish the item)

Output $X_1\ X_2$ ... X_s, and increase ρ by s. Any alignment marks at the right of the format item now cause activation of process A for each " / ", process B for each " ↟ ", and process C for each " J ". Return to step 3.

Process A (" / " operation)

Check page alignment with process F, then execute process D and call procedure P_N.

Process B (" ↓ " operation)

If $\rho > 0$, execute process D, call procedure P_N. Then execute process E and call procedure $P_{N'}$.

Process C (" *J* " operation)

Check page and line alignment with process G. Then let $k = ((\rho - L + 1) \div TAB + 1) \times TAB + L - 1$ (the next " tab " setting for ρ), where *TAB* is the " tab " spacing for this channel. If $k \geqslant min\ (R, P)$, perform process H (k); otherwise effectively insert blanks until $\rho = k$.

Process D (New line)

Skip the output medium to the next " line ", set $\rho = 0$, and set $\rho' = \rho' + 1$.

Process E (New page)

Skip the output medium to the next " page ", and set $\rho' = 0$.

Process F (Page alignment)

If $\rho' + 1 < L'$ execute process D until $\rho' = L' - 1$. If $\rho' + 1 > R'$ execute process E, call $P_{R'}$, and repeat process F. If $\rho' + 1 > P'$ execute process E, call $P_{P'}$, and repeat process F. This process must terminate because $1 \leqslant L' \leqslant R'$ and $1 \leqslant L' \leqslant P'$.

Process G (Page and line alignment)

Execute process F. Then, if $\rho + 1 < L$, effectively output blank spaces until $\rho + 1 = L$. If $\rho + 1 > R$ or $\rho + 1 > P$, perform process H $(\rho + 1)$. This process must terminate because $1 \leqslant L \leqslant R$ and $1 \leqslant L \leqslant P$.

Process H (k) (Line overflow)

Perform process D. If $k > R$, call P_R; otherwise call P_P. Then perform process G to ensure page and line alignment.

NOTE.—Upon return from any of the overflow procedures, any assignments to hidden variables that have been made by calls on descriptive procedures will cause adjustment to the corresponding variables H1, H2, H3, H4, H5, H6.

2.5.4.2 INPUT. The input process is initiated by the call:

inlist (CHANNEL, LAYOUT, LIST)

The parameters have the same significance as they did in the case of output, except that *CHANNEL* is in this case the number of an input medium. There is a class of procedure *input n* which stand for a call with a particularly simple type of layout and list, just as discussed in clause 2.5.4.1 for the case of output. In the case of input, the parameters of the " item " procedure within the list must be variables.

The various steps which take place during the execution of *inlist* are very much the same as those in the case of *outlist*, with obvious changes. Instead of transferring characters of title format, the characters are ignored on input. If the data is improper, some standard error procedure is used (see clause 1.1.3.8).

The detailed algorithm for *inlist* is as follows: