IA

stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 7/73 JULY

IA

L. AMMERAAL
AN INTERPRETER FOR SIMPLE ALGOL 68 PROGRAMS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

ACM - Computing Reviews - category: 4.13

68 A 15

Abstract

An interpretative implementation for a sublanguage of ALGOL 68 is informally described in this report. Since the implementation has been written in ALGOL 60 and there is no translation from ALGOL 68 source text to an object program, a high degree of machine independence has been achieved. The report includes both the text of the interpreter and some typical ALGOL 68 programs, along with the output produced by the interpreter.

Contents

1. Introduction

An implementer of a high-level programming language will have to pay attention to a number of general implementation aspects which may include:

- efficiency with respect to speed and to use of (core) memory,
- completeness,
- error diagnostics or even error correction,
- machine independence,
- flexibility with respect to future language revisions,
- readability.

As many implementers are concerned about the first three of these points in particular, and because of the ambitious nature of the language, the implementatiton of ALGOL 68 [1] is generally considered to be a very big software project. Although there is no doubt that these three points are very important, it may also be worthwile to gain some experience with a simpler type of ALGOL 68 implementation, with less emphasis on machine efficiency, no error correction facilities and perhaps some language restrictions, and to see whether other aspects, such as machine independence, flexibility and readability, may be improved as a consequence of these concessions.

A tolerant attitude towards the requirements of machine efficiency leads the implementer to the idea of using a high-level language like ALGOL 60 which has, of course, well-known advantages over the traditional use of assembler languages. Choosing ALGOL 60 for this purpose, at least two kinds of highly machine independent implementations can be thought of.

The first is an ALGOL 68 compiler, written in ALGOL 60, and producing an object program in ALGOL 60.

The second is to use ALGOL 60 for the implementation, but to produce no object program at all, i.e. to let the implementation consist of an ALGOL 68 interpreter, written in ALGOL 60. Since the latter approach involves no object program, it seems to be conceptually simpler than the former. As to machine efficiency, it is clear that we must not expect too much of such an interpreter, at least not if we are interested in the execution speed of

production jobs.
From a more theoretical point of view, this interpretative type of implementation might have some value in the following sense. Semantic actions are described in the Report [1] in a very concise and almost algorithmic way, but not in a form that can be presented to a machine in order to perform these actions. In an interpreter, however, the semantic actions are described in such a form. This description is more direct in an interpreter than in a compiler where the actions are only prepared.
In view of these considerations an interpreter was written in ALGOL 60. It will be discussed in the next chapters of this report.

2. Elaboration as a side-effect of syntactic analysis

The top-down method for syntactic analysis has well-known attractive properties. The method is presented in a tutorial manner by Knuth [4]. It has often been illustrated by parsing simple pieces of ALGOL 60 programs. Top-down parsing is less often described using ALGOL 60 as a tool, however. Yet this can be done very well by means of a set of Boolean procedures. A method for generating such a set automatically, was suggested by Koster [2]. Although this idea is very attractive, the Boolean procedures mentioned in the sequel of this report were written by hand. They are based on the assumption that the program to be analysed is stored in an integer array X, and that at any moment a global integer p, the so-called program pointer, has such a value that $X[p]$ is the point from which the analysis has to proceed at that moment. Initially, p=1. Now consider the production rule for some notion N of a given context-free grammar. Then there is a Boolean procedure N with the task to determine whether or not the array X contains a terminal production of N starting at $X[p]$ for the current value of p. If there is not such a terminal production the value *false* is delivered. If, however, there is, the value *true* is delivered, and the procedure has to perform at least one additional task, namely to set the program pointer p to the new value p' where $X[p']$ is the point in the program immediately after the found terminal production.
The two tasks mentioned, of which the second is a typical side effect, suffice for the syntactic analysis; the following hypothetic example may show how this can be done:

Example 1

```
A : B; p,q; C,D .
B : r,A; u .
C : t,u .
D : v; w,D,z .
```

In contrast to the ALGOL 68 report, here capital and small letters mean notions and (terminal) symbols, respectively.
To analyse in ALGOL 60 whether a given integer array X contains a terminal production of A, and assuming that p has a correct initial value, say 1,

we need just call the Boolean procedure *A*, given below, which will yield the desired answer *true* or *false*:

```
Boolean procedure INP (s); integer s;
if X[p] =s then begin INP:= true; p:=p+1 end else INP:=false;

Boolean procedure AND (U,V); Boolean U,V;
begin integer pold; pold:=p;
      if if U then V else false then AND:=true else
      begin AND:=false;
            p:=pold
      end
end;

Boolean procedure A;
A:= if B then true else
    if AND (INP(p), INP(q)) then true else
    AND(C,D);

Boolean procedure B;
B:= if AND(INP(r),A) then true else INP(u);

Boolean procedure C;
C:= AND(INP(t), INP(u));

Boolean procedure D;
D:= if INP(v) then true else
    AND(INP(w), AND(D,INP(z)));
```

It may be a useful exercise to verify by means of these procedures that

rrtuwwwvzzz

is a terminal production of the notion A.

Of course, the Boolean expressions

if E then F else false

and

if G then true else H,

must not be replaced by

$E \wedge F$

and

$G \vee H$,

respectively, since in the former expressions, it depends on E and G whether F and H, respectively, will be evaluated, but in the latter expressions both operands will be evaluated in any case.
Although this undesirable behaviour of the $\wedge$ - and $\vee$ - operators has often been mentioned by others, it may be observed that top-down parsing is another example where it would be particularly convenient if $\wedge$ and $\vee$ had the property that the second operand is only evaluated if the evaluation of the first operand does not suffice to determine the result. If they had this property, both expressions

if if A then B else false then C else false

and

if if A then (if B then C else false) else false,

e.g., could be replaced by

$$A \wedge B \wedge C \ .$$

The transformation of grammar rules into Boolean procedures, as sketched above, is not allowed for each context-free grammar. If it is for some CF-grammar, care has to be taken in choosing the order of the alternatives of a given notion.
For these reasons the following considerations may be useful.

Suppose that α is a string of n terminal symbols, β is the string composed of the first m symbols of α ($0 \leq m < n$) in the same order, and both α and β are produced by a notion N. Now if a terminal production of N has to be recognized, starting at $X[p]$,
and it happens that

$$\alpha = X[p] \ldots . X[p+n-1] \quad \text{and}$$

$$\beta = X[p] \ldots X[p+m-1] \ ,$$

we want to make a definitive choice.
An example of such a situation is the choice between the real denotation *12.3* and the integral denotation *12*, if a denotation has to be recognized and the four symbols *12.3* are available in *X[p]* ... *X[p+3]*. In this example the second alternative must not be taken, because otherwise the two symbols *.3* in *X[p+1]* *and* *X[p+2]* cannot be parsed afterwards. This knowledge cannot be derived from the rule for the notion "denotation", but other rules have to be considered as well.

Similarly, in the general case above we will reject the choice of the shorter string β if the longer string α is also available as a candidate. By this strategy, a grammar with the rules

N : P,Q .
P : u,v; u .
Q : v,w ,

cannot be used in this form to recognize the string uvw as a terminal production of N, so not all context free grammars are suited for our method. For the grammar rules

N : P,Q .
P : u,v ; u .
Q : w .

uvw can be recognized, however. The procedure body for P is

P := if AND (INP(u),INP(v)) then true else INP(u)

For reasons of efficiency, however, the rule for P is preferably replaced by the rules

P : u,R .
R : v ; .

Now the same considerations as above apply to R, with α = v, n = 1, β = ε (the null string), m = 0. The procedure body for R might be

R := if INP(v) then true else true .

It is well-known that for top-down parsing, left-recursive rules have to be excluded. This phenomenon and many other interesting topics concerning syntactic analysis are extensively discussed by Knuth [4].

Syntactic analysis is only part of an implementation. If the implementation had the task to produce an object program, this, or some preparation for it, would have been accomplished by another side-effect of the Boolean procedures. In an interpreter, however, the semantic actions themselves may be executed; in other words, the elaboration of a notion may take place as a side-effect of the Boolean procedure corresponding to that notion. If the notion was e.g. formula, and the found terminal production was *3+4* it may be our intention that the sum 7 is immediately yielded. On the other hand, it has sometimes to be decided whether there can be found a terminal production of a given notion, starting at *X*[*p*], and if so, *p* should be advanced, but the elaboration side-effect must not take place. Still another possibility exists, namely the requirement that only defining occurrences of either labels or modes, if present, are dealt with. This will be discussed for labels in chapter 8.
So there are three cases with respect to the desired elaboration side-effect. Information about this is supplied to the procedure by means of a formal parameter *q* of which the corresponding actual parameter may have one out of three values. These three actual parameter values are written here as the letters *f*, *g* and *h*, which might be associated with the speed of the scanning process:

q = *f* (fly)	: no elaboration required
q = *h* (hasten)	: only defining occurrences of labels and modes are to be dealt with
q = *g* (go)	: all other elaboration actions required.

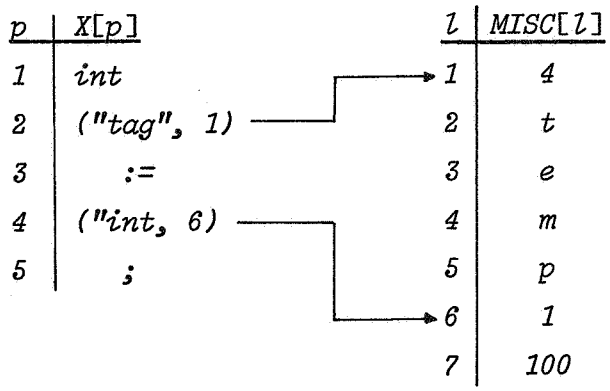
3. The internal representation of the ALGOL 68 text

The input for the interpreter is an ALGOL 68 program whose smallest compounding parts, at this level, are characters.
Thus the piece of ALGOL 68 text

'INT' TEMP:=100;

is a sequence of 16 characters.

The first step after the program has been read character by character, is to form symbols from small sequences of characters, e.g. the five character sequence *'INT'* is replaced by the integral symbol *int* and the two character sequence *:=* is replaced by the becomes symbol :=. Blank spaces that are not inside string denotations are omitted, of course. During this lexical scan the integer array *X*, mentioned before, is filled. For a tag, such as *TEMP*, only one element of *X* is used, containing both a reference to a point in an auxiliary integer array *MISC*, where the letters of the tag are stored if they were not already, and the mark *"tag"*, for recognition purposes later on. Likewise an integral denotation like *100*, is stored into the auxiliary array *MISC* and a reference to this is put into array *X*, together with the mark *"int"*.
Now the example above may be represented internally as follows



The array *MISC* is used for miscellaneous purposes. It contains variable length items, which are all mutually different. Immediately preceding each

item, its length is written. Now if e.g. the tag *TEMP* is encountered later on, perhaps with a completely different meaning like a label or a selector, a search for *TEMP* is performed in *MISC* and when it is found at the point l', this value l' is written into array *X*, together with the mark *"tag"*. This mark is simply a unique small integer, also called a mnemonic constant. A number pair like *("tag",l)* is stored as the integer *"tag"* $*$ $C + l$. Here C is some appropriate large integer, e.g. 100 000. Besides the mentioned marks *"tag"* and *"int"*, there are also the marks *"real"*, *"ind"*, *"char"* and *"string"* for real denotations, indications, character denotations and string denotations, respectively. They are all coupled with a reference l which, with the exception of *"real"*, points to the integer array *MISC*, where the characters composing the given denotation or indication are stored. To store real denotations a real array *RNR*, instead of the integer array *MISC*, is used. Here each item consists of only one element, so the item need not be preceded by the item length. A pleasant property of the sketched method of internal representation is the lack of limitations with respect to the lengths of tags and indications other than those which are implied by the finite size of array *MISC*.

4. The internal representation of values

In ALGOL 68 the elaboration of a notion like a serial clause, a unit, a tertiary, a secondary or a primary, delivers a value. Such a value may disappear shortly after it is created, like the value of *(a+b)* in the elaboration of the formula

*(a+b) * c* .

On the other hand, there may be a mode identifier (or an operator) that possesses the given value, which relationship should hold for some time. Consider, as an example, the identity declaration

int i = 5

As we are now using the output produced by the lexical scan, we have the number pairs *("tag", l^i)* for *i*, and *("int", l^5)* for *5* in array *X*, with

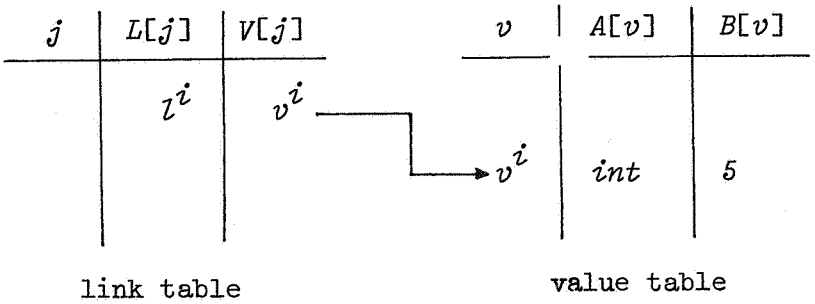
MISC [l^i] = *1*
MISC [l^i+*1*] = the integral representation for the character *i*
MISC [l^5] = *1*
MISC [l^5+*1*] = *5* .

Now as a result of the elaboration of the given identity declaration, l^i is written in the first free element *L[j]* of the integer array *L*. This array *L* will be regarded as the first column of a two column table, called link table, the second column being integer array *V*. Since the identifier *i* is to possess the value 5, it looks attractive to set *V[j]* to 5 by which the relationship to "possess" between an external and an internal object would simply be visualised by the contents of a row of our link table. Unfortunately this cannot be done this way in general because not every value in ALGOL 68 will fit in a single integer array element *V[j]*. Moreover, not only the value itself, but also information about the mode of this value has to be stored. A third reason for introducing something else than the link table to store values is the superfluity of column *L* for

"anonymous" values like *a+b* in the formula *(a+b)*c*. Here we meet something characteristic for ALGOL 68. In most other languages it is not necessary to bother about the way in which such an intermediate result *a+b* is stored since this value appears only very temporarily in a single machine word or even in a register, whereas in ALGOL 68 this value *a+b* may have quite a complicated structure. It would probably be efficient with respect to storage space, to store values of some simple mode in a table, different from a table used for values of a more complicated mode, or to use some method for mapping a given value onto a one-dimensional array. For reasons of simplicity, however, but at the cost of a less efficient use of storage, a very easy and general method to represent values is chosen in our interpreter. Values, simple or complicated, anonymous or possessed by either an identifier or an operator, are all stored in a so called value table. This consists of the five columns *A*, *B*, *C*, *D*, *E*.
Each of the columns *A*, *B*, *C* and *D* is an integer array. *E* is a real array.
Now as a result of the elaboration of

int i = 5

we obtain the following picture



In column *V* of the link table, a reference to the v^i-th row of the value table is stored, where v^i is obtained using a free-list technique. In the value table, columns *A* and *B* contain the mark *"int"* and the number 5, respectively. Columns *C* and *E* have no function for this simple case; column *D* has, as will appear in the sequel.

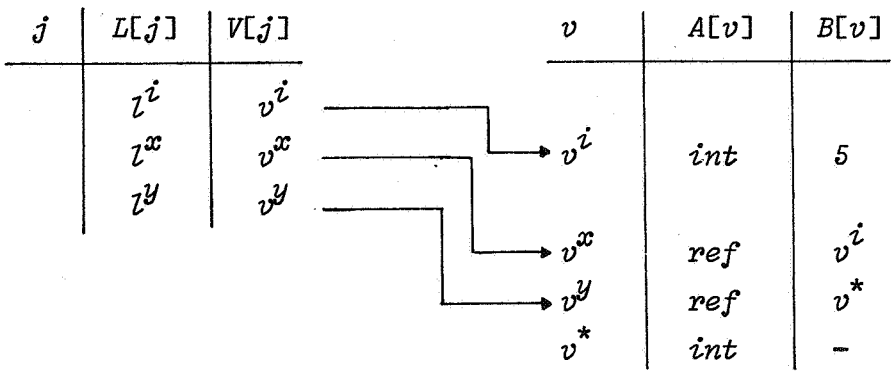
For a boolean value the mark "bool" will appear in column *A* and the integer 1 or 0 in column *B* for *true* or *false*, respectively.
Likewise for a character, the mark *"char"* is put into column *A*, and an integer representation of a character into column *B*. A real number will be put into column *E*, in which case column *B* is not used and the mark "real" is in column *A*.
Now suppose that after the identity declaration

int i = 5

with the resulting picture above, the variable declaration

int x:=i , int y

follows.
Then the given picture will be supplemented as follows.



A line in the link table appears for x and for y. The references v^x and v^y are put into column *V* of these lines, pointing to new lines in the value table with the mark *"ref"* in column *A*, since x and y have modes beginning with "reference to". Now x has to refer to the integral value i, so v^i is put into $B[v^x]$. Likewise, y has the mode "reference to integral" but no integral object for y to refer to is given by the declaration. Yet the given information about the mode of y has to be stored somehow. This is done here by creating a new integral value on a line v^* and by assigning v^* to $B[v^y]$. Note that $v^* \neq v^y+1$ may hold. It will be clear that a mode beginning

with more than one "reference to" will easily be represented in the value table by additional lines with *"ref"* in column *A* and values v in column *B*.

The representation of a multiple value consists of three parts in the value table. The first part is the so-called "initial line" with the mark *"row"*, a reference v^d to the descriptor , and the number (say n) of subscripts in columns *A*, *B* and *C*, respectively. The second part is the descriptor part, starting with line v^d, with
$A[v^d]$ = *1* or *0*, for the state (strict or flex),
$B[v^d] = v^{fe}$, a reference to the first element,
$C[v^d] = c$, the offset, being 1 in most cases.
This line is immediately followed by n lines, one for each subscript. For subscript i $(i=1,\ldots,n)$ we have
$A[v^{d+i}] = l_i$, the lower bound of the i-th subscript,
$B[v^{d+i}] = u_i$, the upper bound of the i-th subscript,
$C[v^{d+i}] = d_i$, the stride of the i-th subscript.

The third part is filled as follows. If $u_i \geq l_i$ for all i $(1 \leq i \leq n)$, the number of elements is $m = \prod_{i=1}^{n} (u_i - l_i + 1)$, and the elements, or at least their initial lines, are stored at the lines $v^{fe},\ldots,v^{fe+m-1}$, in the usual order; otherwise one fictitious element is stored in the same way, using line v^{fe}.

Thus the example

[5 : 6 , 21 : 23] int a ;

will result in the following scheme:

j	$L[j]$	$V[j]$
	:	
	l^a	v^a
	:	:

v	$A[v]$	$B[v]$	$C[v]$
v^a	*ref*	v'	
v'	*row*	v^d	*2*
v^d	*1*	v^{fe}	*1*
v^d+1	*5*	*6*	*3*
v^d+2	*21*	*23*	*1*
v^{fe}	*int*	-	
$v^{fe}+1$	*int*	-	
$v^{fe}+2$	*int*	-	
$v^{fe}+3$	*int*	-	
$v^{fe}+4$	*int*	-	
$v^{fe}+5$	*int*	-	

In the chosen method of representation, it is essential that, if an initial line of some value is complemented by other lines, these other lines need not immediately follow the initial line, but may appear at any point in the value table.

Another application of this principle is the representation of a structured value. Like a multiple value, a structured value is represented in the value table by several parts, the first of which is its initial line.
In the first three columns, this line contains the mark *"struct"*, a reference v^d to a "descriptor", and the number *(n)* of fields.
The descriptor, being the second part, consists of the lines
v^d, $v^d+1,\ldots,v^d+n-1$, with
$B[v^d + i - 1] = v^i$, a reference to (the initial line of) the i-th field
$C[v^d + i - 1] = l^i$, the unique representation, see chapter 3, of the tag used for the i-th selector
$(i=1,\ldots,n)$.
Finally, each field of the structured value is represented in the general way in which values are stored, i.e. the i-th field or its initial line is v^i. In contrast to a multiple value, the initial lines v^i of the elements of a structured value need not be consecutive.

The next example will show that the value table is not only used to store values. It will not only illustrate the representation of a structured value, but show the treatment of a mode declaration and a collateral clause as well.
Consider the declarations

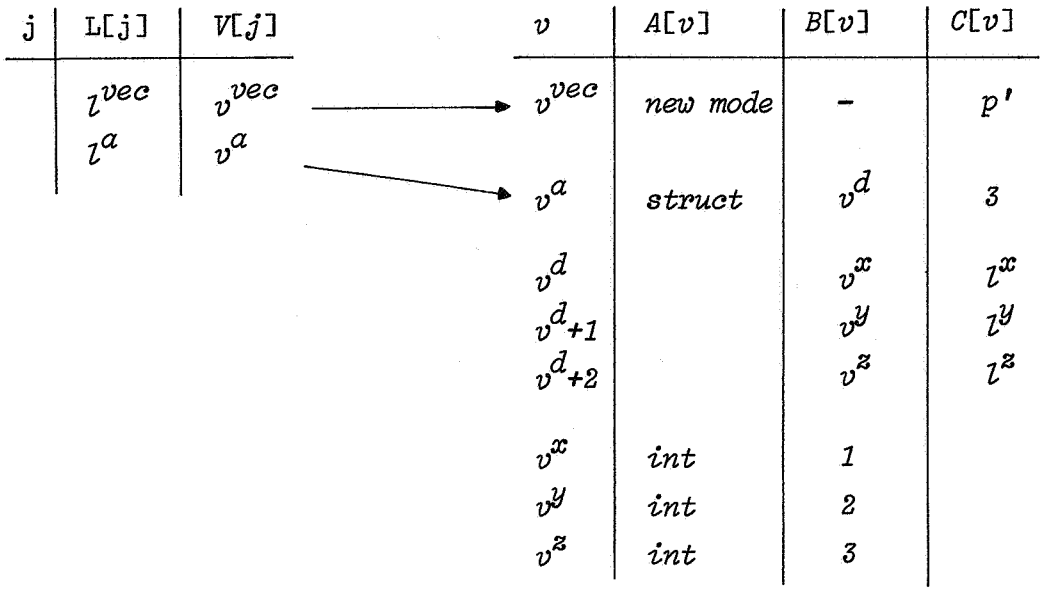
mode vec = struct (int x,y,z);
vec a = (1,2,3)

For the present version of the interpreter, these two declarations must not be interchanged.
The ALGOL 68 report states that the elaboration of a mode declaration involves no action. The interpreter, indeed, performs no substantial action in elaborating the mode declaration; it only makes a note where the mode declaration occurs in the program, to be able to use it afterwards. For this note, both the link table and the value table are used. In chapter 3, it was mentioned that an indication is represented as a number pair *("ind",l)* in a single element of array *X*. So both occurrences of the mode indication *vec* appear as the pair $("ind", l^{vec})$. Now, regardless of the text following the equals symbol, the effect of

mode vec = ...

will be both the addition of a line j, with $L[j] = l^{vec}$, $V[j] = v^{vec}$, to the link table and the addition of a line v^{vec} to the value table, where $A[v^{vec}]$ is set to the mark *"newmode"*, and $C[v^{vec}]$ is set to the value p', pointing to the beginning of the actual mode declarer in array *X*.
On encountering the second occurrence of *vec*, v^{vec} is looked up in the link table with the given l^{vec}, in a way to be described in chapter 6. Using v^{vec}, the value of $p' = C[v^{vec}]$ is found and now the current value p of the program pointer is temporarily replaced by p', in order to use the normal syntactic analysis and elaboration procedures, resulting in a new, fictitious value of the mode *struct (int x,y,z)*. This value is represented in the value table as described above. The next elementary step is to

elaborate the given collateral clause. Since a collateral clause may also be used in connection with a multiple value, the interpreter creates first an intermediate value, using in its initial line the mark *"collat"*, a reference v' and the number (say n) of elements; v' points to the consecutive lines v', $v'+1$,..., $v'+n-1$ where the elements or at least their initial lines are stored. Afterwards, this intermediate value is transformed into either a structured value or a multiple value and its space in the value table is released. The ultimate picture resulting from the two given declarations is

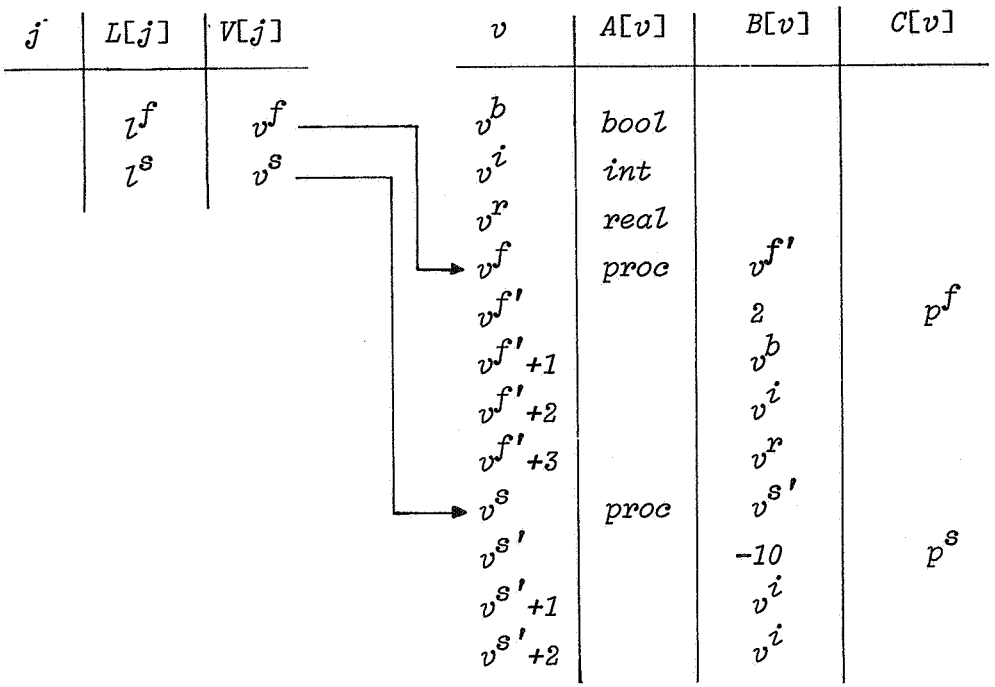


Since a routine is a value in ALGOL 68 which may be possessed by an identifier or an operator, it belongs in the value table like the values met before. It is more appropriate, however, to store only part of it in the value table, together with a reference p, pointing to the beginning p of the routine text $X[p]$, $X[p+1]$,... .
Consider the example

proc f = (*int* i, *real* r) *bool* : $5 * i \uparrow 2 > r$;

op s = (*int* i) *int* : $i * i + i$;

The resulting picture will be given first and then explained.



The first three lines of the value table are always filled as shown in the picture, so this is not a result of the given declarations.
A routine starts in the value table with an initial line v containing the mark *"proc"* and a reference v' to a second part of the table. If the routine is not possessed by an operator, $B[v']$ is the number (n) of formal parameters and $C[v']$ is a reference pointing to the beginning of the routine text in array X (if this routine text is contained in the particular program under consideration). The next n+1 lines contain the mode that the routine is to deliver and the modes of the formal parameters. Here a mode is given by a reference to a value having this mode. If the routine is possessed by an operator its internal representation is similar, except for $B[v']$. Here the priority number of the operator is stored negatively,

where -10 means that the operator is monadic. For routines whose routine texts are not contained in the particular program, but are available by virtue of the standard prelude, the value $C[v']$ is not a (positive) program pointer value, but a negative number corresponding uniquely to such a routine.

An identifier with some mode <u>united</u> from other modes is stored in the link table as a pair (l,v) like other identifiers. In the value table, the mark *"union"* is put into $A[v]$. If $v',v'',\ldots,v^{(n)}$ are the initial lines of values whose modes are exactly the modes from which the given mode is united, and if v^o is the value possessed by the identifier at a given moment, then this is represented internally by

$$B[v] = v^* \quad , \quad C[v] = v^o \ ,$$
$$B[v^*] = v' \quad , \quad C[v^*] = n \ ,$$
$$B[v^*+1] = v''$$
$$\vdots$$
$$B[v^*+n-1] = v^{(n)} \ .$$

5. The elaboration of a formula

It seems attractive to treat the operators of the standard prelude, called "elementary operators" here, and the operators declared in the program in the same way as much as possible. Therefore, data concerning these elementary operators are stored in the first part of the value table beforehand. This is done by putting a unique negative number for each elementary operator in column *C*, as mentioned in the previous chapter. Likewise, part of the link table is filled beforehand, with a line *(l,v)* for each elementary operator.

For the elaboration of a formula a modified version of a well-known method is used, namely the method of transforming the formula into Reverse Polish. Instead of this form, the value of the formula is the result, however. This result is delivered in the value table, and a global variable *z* is made to point to its initial line. A piece of ALGOL 60 text will show this principle and some general constructions as well. In this text, the neater form

while ... *do* ...

is used instead of

for dummy:=dummy while ... *do* ...

Some general self-explanatory procedures are:

```
Boolean procedure LOOK (A); Boolean A;
begin integer pold;
      pold:=p; LOOK:=A; p:=pold
end;

Boolean procedure LOOK2 (A,B); Boolean A,B;
begin integer pold; pold:=p;
      LOOK2:= if A then B else false;
      p:=pold
end;
```

Some procedures local to the procedure "formula" are only informally described here. The verb "to read" is used in this description in the sense

of the scanning process, with its possible side-effects, described in chapter 2. These procedures are:

Boolean procedure OPERATOR (lwbprio,upbprio,optr,prio);
value lwbprio,upbprio;integer lwbprio,upbprio,optr,prio;
[This procedure reads an operator, if possible, with a priority not lower than *lwbprio* and not higher than *upbprio*; if this attempt succeeds, this priority is assigned to *prio* and the found value table index of the operator is assigned to *optr*.]

procedure OPERAND;
[This procedure reads an "operand". Here "operand" means either a monadic formula or a secondary. The resulting value is delivered in the value table and pointed to by the global variable *z*, if parameter *q* of *FORMULA* equals *g*].

procedure DYADIC OPERATION(v,w); integer v,w;
[This procedure performs a dyadic operation. The value table index of the operator w is found at the top *OPST*[*w*] of the operator stack. The value table indices of both operands are at the top of a value stack *VLST*, such that the effect of this procedure can be given by

VLST[*v-1*] := *z*:= *VLST*[*v-1*] w *VLST*[*v*] ;
v :=*v-1* ;
w :=*w-1*]

If *OPST*[*w*] points to operator w, then *PRIOST*[*w*] is the priority of w; *PRIOST* can be regarded as a second column of the operator stack.
Now the main part of the procedure *FORMULA* is given by

```
Boolean procedure FORMULA (q); integer q;
begin [declarations omitted]

    if if LOOK (OPERATOR(10,10,dummy,dummy)) then true else
    LOOK2 (SECONDARY(f), OPERATOR(0,9,dummy,dummy)) then
    begin FORMULA:= true;
          OPERAND;
          if q=g then
          begin VLST [1] := z ; v:=1 ;
                PRIOST[0]:=0 ; w:=0 ;
                while OPERATOR (0,9,optr,prio) do
                begin while prio ≤ PRIOST [w] do
                      DYADIC OPERATION (v,w);
                      w:=w+1 ;
                      OPST [w]:=optr ; PRIOST [w]:=prio ;
                      OPERAND ;
                      v:=v+1 ; VLST[v]:= z
                end;
                while v > 1 do DYADIC OPERATION (v,w)
            end else
            while OPERATOR (0,9,dummy,dummy) do OPERAND
    end     else FORMULA:= false
end
```

Note that after a successful call of *FORMULA* with $q=g$, the global variable z has been made to point to the resulting value. This was done either by *DYADIC OPERATION* if the formula was dyadic, or by *OPERAND* if it was monadic. As to the task of the procedure *DYADIC OPERATION*, there are two possibilities. If the operator involved is elementary, the operation itself takes place within this procedure; if it is not, the operation is performed by using the operation declaration, contained in the ALGOL 68 program. This latter case has a strong analogy with the elaboration of a call, described in chapter 7.

6. The identification of identifiers and indications

Before the treatment of the elaboration of a call, something should be said about the identification method used. As mentioned in chapter 4, by the declaration

int i

the integer l^i, which corresponds uniquely to the tag *i*, is placed into the link table, together with a value table index v^i. This link table is filled consecutively starting at the top. At any moment it is known how far the link table has been filled, say up to line *j*, so for a new declaration, first *j* will be increased by one and then *L[j]* and *V[j]* will be used. Now if the identifier *i* has to be looked up, say for the elaboration of

i := 1 ,

a search for l^i is made in the link table, in the direction opposite to the way the link table was filled, i.e. starting at line *j*, then *j-1*, etc, until l^i is found. This search direction guarantees that the most recently elaborated declaration of the tag *i* will be taken, so if *i* is declared at the beginning of both a closed clause A and a closed clause B which is contained in A, an applied occurrence of i in the outermost reach of B will result in finding the declaration at the beginning of B.
Now on leaving B we have to get rid of this second declaration of *i*. This is accomplished by storing the current value of the link table pointer *j'* into a variable *jold* local to the Boolean procedure CLOSED CLAUSE on the entrance of this procedure, and by resetting *j* to the value *jold* on its exit. Moreover, all values in the value table referred to by the link table lines that are discarded at this step, are released as far as this is possible. More specifically, such a value is relaxed, which means that its reference counter, being the value in column *D*, is decreased by one. If, as a result of such a relaxation, the reference counter of a value obtains the value zero, then the value involved is actually released, which means that its initial line is added to a free list and that its auxiliary lines are also added to the free list if these lines are not at the same time initial lines of values; if they are, however, they are relaxed, possibly also resulting in releasing, depending on the value of their own reference counter. The elements of a multiple value are an

example of the latter kind of lines.
It is recognized that a straightforward use of this reference counter method had the disadvantage that values of a circular structure will not be released. This disadvantage is accepted in the present version, however.
To the above description of the use of the link table, an extension is necessary for the correct identification of an identifier, with a given tag, whose applied occurrence appears within a routinetext, since the dynamically last encountered declaration of an identifier with this tag need not be the defining occurrence we have to look for. This problem is solved by making part of the linktable "invisible", in a way similar to a method used for "protection" as will be described in the next chapter.
In this chapter the identification of identifiers was sketched; nearly the same method, including the use of the link table and the value table, applies to the identification of indications. A minor complication occurs in the search for an operator indication, because here the mode of the operands must also be taken into account during the identification.

7. The call

Consider the example

```
proc plus = (int a,b) int: a + b ;
plus (1,2) .
```

In the ALGOL 68 Report, the elaboration of a call is described by means of replacing the skip symbols of a routine by the corresponding actual parameters, thus constructing an identity declaration. The interpreter actually constructs such an identity declaration on the elaboration of a call, for which purpose a free part of the array *X*, immediately beyond the final program symbol *X[n]*, is used.
So for the above example, the identity declaration

```
int a=1 , b=2
```

will be constructed.
Then this identity declaration is elaborated like one that is contained in the program text. To do so, the program pointer *p* makes an "excursion", i.e. its value is saved, *p* is set to *n+1*, and afterwards *p* will be reset to the saved value. Next, the remaining part, i.e.

```
int:a+b ,
```

which is considered to be a cast, is elaborated by means of a similar excursion of *p*.
Of course, this description is too simplistic, as is shown by the following example:

```
int a:=1, b:=2;
proc plus = (int a,b) int : a+b;
plus (b,a) ,
```

where the identity declaration

int a=b , b=a

would arise, which is obviously wrong. Therefore the Report describes a "protection" process.
In the interpreter the protection problem is solved by making the formal parameters "invisible" when they enter the link table, which means that instead of the small integral values l^a and l^b in our example, the values $l^a+10000$ and $l^b+10000$, respectively, are taken. The term "invisible" is chosen because the incremented value of l^a is not "seen" on the search in the link table activated by the second part

b=a

of the identity declaration above, thus enabling the searching process to find the unaffected value l^a. After the completion of the identity declaration the formal parameters are made visible, by decreasing their l-values by *10000*. So the formal parameters need not be changed in the remaining part of the routine.

8. The jump

It has often been stressed that frequent use of explicitly written jumps does generally not improve the quality of programs. Therefore there is a tendency to reduce the use of goto statements in ALGOL-like languages. Typically, the interpreter itself contains only a single goto statement, used as an exit in the case of an abnormal termination of the program. Although it was proved by Van Wijngaarden [5] that even in ALGOL 60, a jump can always be avoided, it is an ALGOL 68 concept and it has therefore been implemented in the interpreter. Basically, a jump results in giving the program pointer p a new value. We have to distinguish, however, between a local jump of the form

..... ; goto l1 ; ... ; l1 : ...

and a non-local jump like

...; (...; goto l2;...);...);...;l2:...

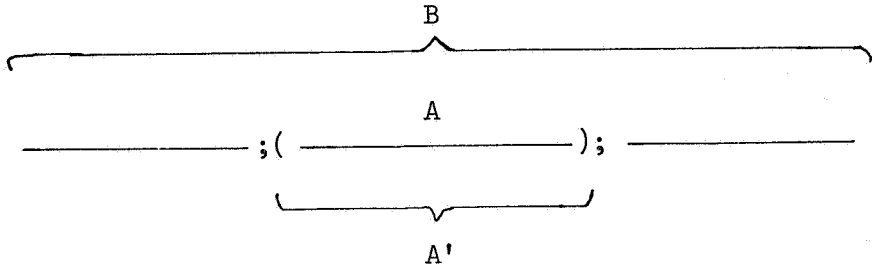
Especially for these forward jumps, it is appropriate to deal with defining occurrences of labels in the scan with $q=h$, i.e. before the more active scan with $q=g$.

If such an occurrence is found, the value l representing the label internally is put into the link table, together with the current value of the program pointer p. The latter value is supplied with a minus sign before putting it into column V, to be able to recognize it as a label. Moreover, in this label-searching scan the program pointer value p_2 pointing to the end of the serial clause is found, which we will need later on. During this scan, a new range, e.g. a closed clause, is skipped, since labels within it are not interesting at this stage. If now in the next scan (with $q=g$) of the serial clause a jump is encountered, the label involved is looked up in the link table, yielding the stored value of p. Now the jump is local if

$$p_1 < p < p_2 \qquad (1)$$

where p_1 and p_2 are the starting and ending point of the serial clause, respectively. In this case, the found value of p replaces the current value of the program pointer and the scan is continued.

If (1) does not hold, however, the jump is non-local. Simply changing the program pointer is not allowed now, since this would have the following effect. The program text after the defining occurrence of the label involved would be treated as if it was contained in the smallest serial clause containing the jump. Therefore something more complicated has to be done now. To sketch the treatment of both the local and the non-local jump in more detail, the reader has to remember that the interpreter contains a set of Boolean procedures calling each other in a hierarchical way. The appropriate place for making the test (1) and changing the program pointer in the case of a local jump, is the Boolean procedure *SERIAL CLAUSE*, because p_1 and p_2 are local variables of it. The occurrence of a jump, however, is detected in the Boolean procedure *JUMP* and the normal situation is that the elaboration of a notion takes place in the Boolean procedure corresponding to that notion. The former of these procedures does not directly call the latter, like it is not customary in an army that a general gives a direct command to a soldier. Similarly, information from a soldier to a general usually has to pass a large number of intermediate levels. In our case the parameter q is used as a channel for this latter type of information transport, i.e. from *JUMP* to *SERIAL CLAUSE*. It passes in this upward direction through the procedures *HIP*, *PRIMARY*, *SECONDARY*, *TERTIARY*, and *UNIT*. Consequently the parameter q is a call-by-name parameter in ALGOL 60 terms.
Now suppose that, in the Boolean procedure *SERIAL CLAUSE* the occurrence of a jump is perceived after a call of UNIT by comparing the values of q before and after this call, and that the found value of p does not satisfy (1), i.e. the jump is non-local. Furthermore, suppose that the serial clause under consideration, which we will call A, is preceded by an open symbol and followed by a close symbol, thus forming a closed clause A', which is in turn contained in a serial clause B.
So the situation is:



Now by virtue of the occurrence of a non-local jump in A, the remaning part of A is skipped, but the parameter *q* is left in its modified state. This means that a former but still existing activation of the Boolean procedures *SERIAL CLAUSE*, on analysing B, perceives the occurrence of a jump. Since *p1* and *p2* are local variables of *SERIAL CLAUSE*, test (1) will now be made for the wider range B etc.

9. Results and conclusions

Both ALGOL 68 and the interpreter discussed in the preceding chapters, are still subject to modifications. Therefore the present version of this interpreter, listed in appendix A, will not be the final one, but may be regarded as a first approximation. It consists of two separate programs. The first performs the lexical scan and produces output for the second. The interpreter has run successfully on a number of small but non-trivial ALGOL 68 programs, a few of which are listed in appendix B. The total time used by the Electrologica X8 computer for the listed programs was about 12 minutes.

As to the aspects mentioned in the introduction, the results are not discouraging if one does not assign too much weight to the requirements of efficiency, error handling and completeness. The most serious drawback for large-scale applications is the extensive use of core memory. The error messages are rather cryptical and situations may arise where the interpreter will not report an error but will deliver unpredictable results instead, when it is dealing with an incorrect program.

The language ALGOL 68 was not implemented completely. The implementation was not only based on the original Report [1], but also on a number of proposed modifications which will probably be accepted, such as the abandonment of the proceduring coercion.

Apart from these language revisions, a number of restrictions were introduced, including the following:

- no *bits* or *bytes* structures are available;
- no synchronization operations are available;
- no distinction is made between local and global generators;
- applied occurrences of mode indications must not dynamically precede their defining occurrences;
- as transput procedures, only print, newpage, newline and space are available in the present version; the layout of a printed number may differ from the description in the Report [1].

A more exact and definitive list of restrictions will be made when both a revised ALGOL 68 report has been published and some shortcomings of the interpreter have been eliminated.

The choice of ALGOL 60 and the absence of a machine dependent object code make the interpreter accessible, in principle, for both human beings and machines that can read ALGOL 60. As a final remark, it should be mentioned that some concepts of this language, such as blocks, recursion and conditional expressions have proved to be extremely facilitating tools in writing this interpreter.

References

1. A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck and C.H.A. Koster, Report on the Algorithmic Language ALGOL 68, Numerische Mathematiek, 14(1969).

2. C.H.A. Koster, A Compiler Compiler, Report MR 127/71, Mathematisch Centrum, Amsterdam.

3. T.B. Steel, Jr. Ed., Formal language description languages for computer programming, North-Holland Publishing Company (1966).

4. D.E. Knuth, Top-Down Syntax Analysis. Acta Informatica 1, pp. 79-110 (1971)

5. A. van Wijngaarden, Recursive definition of syntax and semantics. [3], pp. 13-24.

```
BEGIN COMMENT ALGOL 68 INTERPRETER, PART 1, L. AMMERAAL, MACHINE DEPENDENT PROCEDURES; FIXT(M,N,X) -- PRINTS X WITH M
      DIGITS BEFORE AND N DIGITS AFTER THE POINT, ABSFIXT(M,N,X) -- LIKE FIXT BUT WITHOUT SIGN, NLCR -- NEW LINE
      CARRIAGE RETURN, NEWPAGE -- GIVES A NEW PAGE, PRINT(X) -- PRINTS X IN A STANDARD FORM, PRINTTEXT($S$) --
      PRINTS THE STRING S, EXIT -- TERMINATES THE PROGRAM EXECUTION, RESYM -- READS A CHARACTER AND DELIVERS ITS
      INTEGRAL REPRESENTATION, (RESYM IS AN INTEGER PROCEDURE WITHOUT PARAMETERS) PRSYM(N) -- PRINTS THE CHARACTER
      WITH INTEGRAL REPRESENTATION N, CSYM(N) -- THE PUNCHING COUNTERPART OF PRSYM, CPUNCH(X) -- THE PUNCHING
      COUNTERPART OF PRINT, FIXC(M,N,X) -- THE PUNCHING COUNTERPART OF FIXT;
      INTEGER I;
      FOR I:= 1 WHILE TRUE DO
      BEGIN
            BEGIN COMMENT PAPERTAPE TO MAGN.TAPE,L.AMMERAAL;
                  BOOLEAN SEEN1, SUC, NORMAL;
                  INTEGER NRSGN;
                  INTEGER ABSV, AGN, AND, AP1, ARG, POLD1, AT, BEC, BOOL, BUS, BY, BYTES, CARRET, CL, CL1, CHAR, COL,
                  COL1, COM, COM1, COMPL, CONJ, DIV, DO, DUM, EIT, ELEM, ENT, EQ, FALSE, FLEX, FOR, FROM, GE, GOON,
                  GOON1, GOTO, GT, H, HEAP, I, IM, IND, INT, INTNR, INTERROGATION, ISS, ISNT, J, L, LE, LESS, LMIS,
                  LOC, LRNR, LWB, MOD, MODE, MIN, N, NA, NE, NIL, NLINE, NMIS, NOT, NQ, NRNR, NX, ODD, OF, OP, OPEN,
                  OPEN1, OR, OVER, P, PLIN, PLIT, PLUS, PNT, PNT1, PRIO, PROC, PX, QUOTE, QUOTE1, RE, REAL, REF,
                  REPR, RND, S, SGN, SKP, STR, STRING, SPACE, SUB, TAB, TAG, TEN, TIMS, TO, TPNR, TRUE, UN, UP, UPB,
                  VERT, VERT1, VOID, WHL, TMAB, PSAB, PLTO, MAB, DVAB, MDAB, OVAB, PLUS1, EQ1, MIN1, TIMS1, DIV1,
                  PCT, NOT1, LESS1, GT1, OR1, AND1, ZA, ZB, ZC, ZD, ZE, ZF, ZG, ZH, ZI, ZJ, ZK, ZL, ZM, ZN, ZO, ZP,
                  ZQ, ZR, ZS, ZT, ZU, ZV, ZW, ZX, ZY, ZZ;
                  REAL REALNR;

                  PROCEDURE PROUT(A); INTEGER ARRAY A;
                  BEGIN NLCR; NLCR; CSYM(119); CSYM(119);
                        FOR I:= 0 STEP 1 UNTIL A[0] DO
                        BEGIN IF I - I : 10 * 10 = 1 THEN
                              BEGIN NLCR; CSYM(119) END;
                              PRINT(8, 0, A[I]); FIXC(8, 0, A[I])
                        END
                  END;

                  ISS:= 87; ISNT:= 88; BEC:= 84; AGN:= 132; OPEN1:= 118; CL:= 119; PLIT:= 63; COL:= 128; GOON:= 163;
                  COM:= 122; QUOTE:= 174; VERT:= 131; PNT:= 36; FOR:= 168; FROM:= 169; BY:= 170; TO:= 171; WHL:= 172;
                  DO:= 173; INT:= 101; REAL:= 102; BOOL:= 103; CHAR:= 104; VOID:= 92; STR:= 95; COMPL:= 106; TRUE:= 38;
                  FALSE:= 39; REF:= 96; FLEX:= 97; EIT:= 98; PROC:= 99; UN:= 100; STRING:= 109; MODE:= 112; PRIO:= 113;
                  LOC:= 114; HEAP:= 115; OP:= 116; AT:= 129; OF:= 161; GOTO:= 165; SKP:= 166; NIL:= 167; NRSGN:= 125;
                  ENT:= 400031; ABSV:= 400027; ARG:= 400035; CONJ:= 400034; IM:= 400033; LWB:= 400024; ODD:= 400028;
                  RE:= 400032; REPR:= 400036; RND:= 400030; SGN:= 400029; UPB:= 400025; UP:= 400023; TMAB:= 400004;
                  TIMS:= 400018; PSAB:= 400002; PLTO:= 400001; PLUS:= 400016; MAB:= 400003; MIN:= 400017;
                  DVAB:= 400007; DIV:= 400019; MDAB:= 400006; MOD:= 400021; OVAB:= 400005; OVER:= 400020; NE:= 400011;
                  NOT:= 400026; LE:= 400014; LESS:= 400012; GE:= 400015; GT:= 400013; OR:= 400008; AND:= 400009;
                  EQ:= 400010; PLIT:= 400022; ZA:= 10; ZB:= 11; ZC:= 12; ZD:= 13; ZE:= 14; ZF:= 15; ZG:= 16; ZH:= 17;
                  ZI:= 18; ZJ:= 19; ZK:= 20; ZL:= 21; ZM:= 22; ZN:= 23; ZO:= 24; ZP:= 25; ZQ:= 26; ZR:= 27; ZS:= 28;
                  ZT:= 29; ZU:= 30; ZV:= 31; ZW:= 32; ZX:= 33; ZY:= 34; ZZ:= 35; TAG:= 3; IND:= 4; AP1:= 120;
                  PLUS1:= 64; MIN1:= 65; LESS1:= 72; COM1:= 87; PNT1:= 88; TEN:= 89; QUOTE1:= 121; NOT1:= 76; GT1:= 74;
                  INTERROGATION:= 122; COL1:= 90; VERT1:= 127; SPACE:= 93; TAB:= 118; PCT:= 132; OR1:= 79; AND1:= 80;
                  CARRET:= 119; NX:= 1000; NMIS:= 1000; NRNR:= 500; NA:= 5000; NQ:= 200; OPEN1:= 98; CL1:= 99;
                  GOON1:= 91; NLINE:= 500; EQ1:= 70; TIMS1:= 66; DIV1:= 67; SUB:= 100; BUS:= 101;
                  BEGIN INTEGER ARRAY X[0:NX], MISC[0:NMIS], A[1:NA], Q[1:NQ];
                        REAL ARRAY RNR[0:NRNR];

                        PROCEDURE OUT(I); VALUE I; INTEGER I;
                        BEGIN PX:= PX + 1; IF PX > NX THEN ER($ARRAY X EXC$); X[PX]:= I END;


                        PROCEDURE ER(S); STRING S;
                        BEGIN NLCR; NLCR;
```

```
      FOR I:= 1 STEP 1 UNTIL PX DO PRINT(X[I]); NLCR; NLCR; NLCR;
      FOR I:= 1 STEP 1 UNTIL LMIS DO PRINT(MISC[I]); NLCR; NLCR; NLCR; PRINTTEXT($ERROR !$);
      PRINTTEXT(S); EXIT
END;


BOOLEAN PROCEDURE LETGITS; IF BETWEEN(10, S, 35) THEN
BEGIN INTEGER H;
      LETGITS:= TRUE; N:= 1; Q[1]:= S;
      FOR I:= 1 WHILE BETWEEN(0, S, 35) DO
      BEGIN N:= N + 1; Q[N]:= S END;
      IF NEWITEM(H) THEN
      BEGIN LMIS:= LMIS + 1; IF LMIS + N > NMIS THEN ER($MISC,ARRAY EXC$); MISC[LMIS]:= N;
            FOR J:= 1 STEP 1 UNTIL N DO MISC[LMIS + J]:= Q[J]; OUT(100000 * TAG + LMIS);
            LMIS:= LMIS + N
      END
      ELSE OUT(100000 * TAG + H)
END
ELSE LETGITS:= FALSE;

PROCEDURE INTOUT(I); VALUE I; INTEGER I;
BEGIN INTEGER H;
      N:= 1; Q[1]:= I; IF NEWITEM(H) THEN
      BEGIN LMIS:= LMIS + 2; IF LMIS > NMIS THEN ER($MISC ARRAY EXC$); MISC[LMIS - 1]:= 1;
            MISC[LMIS]:= I
      END;
      OUT(100000 * INT + H)
END;


BOOLEAN PROCEDURE NEWITEM(H); INTEGER H;
BEGIN COMMENT IF ARRAY Q CONTAINS AN OLD ITEM THEN H WILL BE SET SUCH THAT IT POINTS TO THAT
      OLD ITEM IN MISC, OTHERWISE H WILL POINT TO THE FIRST FREE CELL IN MISC;

      BOOLEAN PROCEDURE DIFFERENT(N); VALUE N; INTEGER N;
      BEGIN INTEGER J;
            J:= 1;
            FOR I:= 1 WHILE (IF J > N THEN FALSE ELSE MISC[H + J] = Q[J]) DO J:= J + 1;
            DIFFERENT:= J ≤ N
      END;

      H:= 1;
      FOR I:= 1 WHILE IF MISC[H] = - 1 THEN FALSE ELSE IF MISC[H] ≠ N THEN TRUE ELSE
      DIFFERENT(N) DO H:= H + MISC[H] + 1; NEWITEM:= MISC[H] = - 1
END;


PROCEDURE REALOUT(R); VALUE R; REAL R;
BEGIN INTEGER H;
      H:= 1;
      FOR I:= 1 WHILE IF H > LRNR THEN FALSE ELSE RNR[H] ≠ R DO H:= H + 1; IF H > LRNR THEN
      BEGIN LRNR:= H; RNR[H]:= R END;
      PX:= PX + 1; X[PX]:= 100000 * REAL + H
END;


PROCEDURE OM(M); VALUE M; INTEGER M;
BEGIN INTEGER H, J;
      N:= M;
      FOR J:= 1 STEP 1 UNTIL M DO Q[J]:= A[P - M + J - 1]; IF NEWITEM(H) THEN
```

```
      BEGIN LMIS:= LMIS + 1; IF LMIS + N > NMIS THEN ER(4MISC, EXC.4); MISC[LMIS]:= M;
            FOR J:= 1 STEP 1 UNTIL M DO MISC[LMIS + J]:= Q[J]; OUT(100000 * IND + LMIS);
            LMIS:= LMIS + M
      END
      ELSE OUT(100000 * IND + M)
END;


BOOLEAN PROCEDURE REAL DENOTATION(R); REAL R;
BEGIN INTEGER POLD;
      REAL RR;
      POLD:= P; IF FLOATING(RR) THEN
      BEGIN REAL DENOTATION:= TRUE; R:= RR END
      ELSE IF VARPNT(RR) THEN
      BEGIN REAL DENOTATION:= TRUE; R:= RR END
      ELSE
      BEGIN REAL DENOTATION:= FALSE; P:= POLD END
END;


BOOLEAN PROCEDURE VARPNT(RR); REAL RR;
BEGIN INTEGER NDUM, INUM, N, POLD;
      POLD:= P; IF INT DENOTATION(INUM, NDUM) THEN RR:= INUM ELSE RR:= 0;
      IF (IF IS(PNT1) THEN INT DENOTATION(INUM, N) ELSE FALSE) THEN
      BEGIN RR:= RR + INUM / 10 ↑ N; VARPNT:= TRUE END
      ELSE
      BEGIN VARPNT:= FALSE; P:= POLD END
END;


BOOLEAN PROCEDURE EXPPART(IEXP); INTEGER IEXP;
BEGIN INTEGER POLD, ISIGN, INUM, NDUM;
      BOOLEAN SUC;
      POLD:= P; SUC:= FALSE; IF IS(TEN) THEN
      BEGIN ISIGN:= IF IS(PLUS) THEN 1 ELSE IF IS(MIN) THEN - 1 ELSE 1;
            SUC:= INTDENOTATION(INUM, NDUM)
      END;
      IF SUC THEN IEXP:= ISIGN * INUM ELSE P:= POLD; EXPPART:= SUC
END;


BOOLEAN PROCEDURE FLOATING(RR); INTEGER RR;
BEGIN BOOLEAN SUC;
      INTEGER POLD, INUM, NDUM, IEXP;
      REAL RRR;
      SUC:= FALSE; POLD:= P; IF VARPNT(RRR) THEN
      BEGIN IF EXPPART(IEXP) THEN
            BEGIN RR:= RRR; SUC:= TRUE END
      END
      ELSE IF INTDENOTATION(INUM, NDUM) THEN
      BEGIN IF EXPPART(IEXP) THEN
            BEGIN RR:= INUM; SUC:= TRUE END
      END;
      IF SUC THEN RR:= RR * 10 ↑ IEXP ELSE P:= POLD; FLOATING:= SUC
END;


BOOLEAN PROCEDURE INTDENOTATION(I, N); INTEGER I, N;
BEGIN INTEGER S, POLD;
      POLD:= P; IF BETWEEN(0, S, 9) THEN
```

```
      BEGIN I:= S; N:= 1;
            FOR DUM:= DUM WHILE BETWEEN(0, S, 9) DO
            BEGIN I:= 10 * I + S; N:= N + 1 END;
            INTDENOTATION:= TRUE
      END
      ELSE
      BEGIN INTDENOTATION:= FALSE; P:= POLD END
END;


BOOLEAN PROCEDURE BETWEEN(MIN, S, MAX); VALUE MIN, MAX; INTEGER MIN, MAX, S;
BEGIN FOR DUM:= DUM WHILE A[P] = CARRET ∨ A[P] = SPACE DO P:= P + 1;
      IF A[P] ≥ MIN ∧ A[P] ≤ MAX THEN
      BEGIN S:= A[P]; P:= P + 1; BETWEEN:= TRUE END
      ELSE BETWEEN:= FALSE
END;


BOOLEAN PROCEDURE IS(C); VALUE C; INTEGER C; IF A[P] = C THEN
BEGIN P:= P + 1; IS:= TRUE END
ELSE IS:= FALSE;

BOOLEAN PROCEDURE IS2(C1, C2); VALUE C1, C2; INTEGER C1, C2;
BEGIN INTEGER POLD;
      BOOLEAN SUC;
      POLD:= P; SUC:= IF IS(C1) THEN IS(C2) ELSE FALSE; IF ¬SUC THEN P:= POLD; IS2:= SUC
END;


BOOLEAN PROCEDURE IS3(C1, C2, C3); VALUE C1, C2, C3; INTEGER C1, C2, C3;
BEGIN INTEGER POLD;
      BOOLEAN SUC;
      POLD:= P; SUC:= IF IS(C1) THEN IS2(C2, C3) ELSE FALSE; IF ¬SUC THEN P:= POLD; IS3:= SUC
END;


BOOLEAN PROCEDURE IS4(C1, C2, C3, C4); VALUE C1, C2, C3, C4; INTEGER C1, C2, C3, C4;
BEGIN INTEGER POLD;
      BOOLEAN SUC;
      POLD:= P; SUC:= IF IS(C1) THEN IS3(C2, C3, C4) ELSE FALSE; IF ¬SUC THEN P:= POLD;
      IS4:= SUC
END;


BOOLEAN PROCEDURE IS8(C1, C2, C3, C4, C5, C6, C7, C8); VALUE C1, C2, C3, C4, C5, C6, C7, C8;
INTEGER C1, C2, C3, C4, C5, C6, C7, C8;
BEGIN INTEGER N, POLD;
      BOOLEAN SUC;
      POLD:= P; N:= IF C6 = 0 THEN 5 ELSE IF C7 = 0 THEN 6 ELSE IF C8 = 0 THEN 7 ELSE 8;
      SUC:= IF IS4(C1, C2, C3, C4) THEN (IF N = 5 THEN IS(C5) ELSE IF N = 6 THEN IS2(C5, C6)
      ELSE IF N = 7 THEN IS3(C5, C6, C7) ELSE IS4(C5, C6, C7, C8)) ELSE FALSE;
      IF ¬SUC THEN P:= POLD; IS8:= SUC
END;


BOOLEAN PROCEDURE IS12(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12);
VALUE C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12;
INTEGER C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12;
BEGIN INTEGER N, POLD;
      BOOLEAN SUC;
```

```
      POLD:= P; N:= IF C10 = 0 THEN 9 ELSE IF C11 = 0 THEN 10 ELSE IF C12 = 0 THEN 11 ELSE 12;
      SUC:= IF IS8(C1, C2, C3, C4, C5, C6, C7, C8) THEN (IF N = 9 THEN IS(C9) ELSE IF N = 10
      THEN IS2(C9, C10) ELSE IF N = 11 THEN IS3(C9, C10, C11) ELSE IS4(C9, C10, C11, C12))
      ELSE FALSE; IF ¬SUC THEN P:= POLD; IS 12:= SUC
END;

BOOLEAN PROCEDURE ISU(C); VALUE C; INTEGER C; ISU:= IS2(C, AP1);

BOOLEAN PROCEDURE ISU2(C1, C2); VALUE C1, C2; INTEGER C1, C2; ISU2:= IS3(C1, C2, AP1);

BOOLEAN PROCEDURE ISU3(C1, C2, C3); VALUE C1, C2, C3; INTEGER C1, C2, C3;
ISU3:= IS4(C1, C2, C3, AP1);

BOOLEAN PROCEDURE ISU4(C1, C2, C3, C4); VALUE C1, C2, C3, C4; INTEGER C1, C2, C3, C4;
ISU4:= IS8(C1, C2, C3, C4, AP1, 0, 0, 0);

BOOLEAN PROCEDURE ISU5(C1, C2, C3, C4, C5); VALUE C1, C2, C3, C4, C5; INTEGER C1, C2, C3, C4, C5;
ISU5:= IS8(C1, C2, C3, C4, C5, AP1, 0, 0);

BOOLEAN PROCEDURE ISU6(C1, C2, C3, C4, C5, C6); VALUE C1, C2, C3, C4, C5, C6;
INTEGER C1, C2, C3, C4, C5, C6; ISU6:= IS8(C1, C2, C3, C4, C5, C6, AP1, 0);

BOOLEAN PROCEDURE ISU7(C1, C2, C3, C4, C5, C6, C7); VALUE C1, C2, C3, C4, C5, C6, C7;
INTEGER C1, C2, C3, C4, C5, C6, C7; ISU7:= IS8(C1, C2, C3, C4, C5, C6, C7, AP1);

BOOLEAN PROCEDURE ISU8(C1, C2, C3, C4, C5, C6, C7, C8); VALUE C1, C2, C3, C4, C5, C6, C7, C8;
INTEGER C1, C2, C3, C4, C5, C6, C7, C8;
ISU8:= IS12(C1, C2, C3, C4, C5, C6, C7, C8, AP1, 0, 0, 0);

P:= 0; NEWPAGE; NORMAL:= IRUE;
FOR I:= 1, 2, 3, 4 DO CSYM(119); CSYM(1); S:= RESYM;
FOR I:= 1 WHILE S ≠ INTERROGATION DO
BEGIN PRSYM(S); CSYM(S); IF S = CARRET ∨ NORMAL ∧ S = SPACE THEN  ELSE
      BEGIN IF S = AP1 THEN NORMAL:= ¬NORMAL; P:= P + 1; IF P > NA THEN ER(
            ‡SOURCE PROGRAM TOO LONG‡); A[P]:= S
      END;
      S:= RESYM
END;

A[P + 1]:= INTERROGATION; NA:= P; PX:= LRNR:= 0; PLIN:= 2; P:= 1;
FOR H:= 1 STEP 1 UNTIL NMIS DO MISC[H]:= - 1; MISC[1]:= 35; LMIS:= 36;
FOR J:= 2, ZP, ZI, 4, ZS, ZQ, ZR, ZT, 3, ZE, ZX, ZP, 2, ZL, ZN, 3, ZC, ZO, ZB, 6, ZA, ZR, ZC,
ZC, ZO, ZS, 3, ZS, ZI, ZN, 6, ZA, ZR, ZC, ZS, ZI, ZN, 3, ZT, ZA, ZN, 6, ZA, ZR, ZC, ZT, ZA,
ZN, 5, ZP, ZR, ZI, ZN, ZT, 2, ZR, ZE, 2, ZI, ZM, 7, ZN, ZE, ZW, ZP, ZA, ZG, ZE, 7, ZN, ZE,
ZW, ZL, ZI, ZN, ZE, 5, ZS, ZP, ZA, ZC, ZE DO
BEGIN LMIS:= LMIS + 1; MISC[LMIS]:= J END;

FOR I:= 1 WHILE P ≤ NA DO
BEGIN SUC:= SEEN1:= IRUE; IF IS(SPACE) THEN ; POLD1:= P;
      IF LETGITS THEN  ELSE IF REALDENOTATION(REALNR) THEN REALOUT(REALNR) ELSE IF
      INTDENOTATION(INTNR, DUM) THEN INTOUT(INTNR) ELSE IF IS(AP1) THEN
      BEGIN IF IS(ZA) THEN
```

```
BEGIN IF !SU2(ZB, ZS) THEN OUT(ABSV) ELSE IF !SU2(ZR, ZG) THEN OUT(ARG) ELSE IF
   !SU(ZT) THEN OUT(AT) ELSE SUC:= FALSE
END
ELSE IF !S(ZB) THEN
BEGIN IF !SU4(ZE, ZG, ZI, ZN) THEN OUT(OPEN) ELSE IF !SU3(ZO, ZO, ZL) THEN
   OUT(BOOL) ELSE IF !SU(ZY) THEN OUT(BY) ELSE SUC:= FALSE
END
ELSE IF !S(ZC) THEN
BEGIN IF !SU3(ZA, ZS, ZE) THEN OUT(OPEN) ELSE IF !SU3(ZM, ZA, ZR) THEN OUT(CHAR)
   ELSE IF IF !SU(ZO) THEN TRUE ELSE !SU6(ZO, ZM, ZM, ZE, ZN, ZT) THEN
   BEGIN FOR I:= 1 WHILE ¬(IF !SU8(AP1, ZC, ZO, ZM, ZM, ZE, ZN, ZT) THEN TRUE ELSE
      !SU3(AP1, ZC, ZO)) DO
      BEGIN P:= P + 1; IF P > NA THEN ER(‡COMMENT‡) END
   END
   ELSE IF !SU4(ZO, ZM, ZP, ZL) THEN OUT(COMPL) ELSE IF !SU3(ZO, ZN, ZJ) THEN
   OUT(CONJ) ELSE SUC:= FALSE
END
ELSE IF !SU2(ZD, ZO) THEN OUT(DO) ELSE IF !S(ZE) THEN
BEGIN IF !SU3(ZL, ZS, ZE) THEN OUT(VERT) ELSE IF !SU3(ZL, ZS, ZF) THEN OUT(AGN)
   ELSE IF !SU2(ZN, ZD) THEN OUT(CL) ELSE IF !SU5(ZI, ZT, ZM, ZE, ZR) THEN
   OUT(EIT) ELSE IF !SU5(ZN, ZT, ZI, ZE, ZR) THEN OUT(ENT) ELSE IF !SU3(ZS, ZA,
   ZC) THEN OUT(CL) ELSE IF !SU3(ZX, ZI, ZT) THEN OUT(PNT) ELSE SUC:= FALSE
END
ELSE IF !S(ZF) THEN
BEGIN IF !SU4(ZA, ZL, ZS, ZE) THEN OUT(FALSE) ELSE IF !SU(ZI) THEN OUT(CL) ELSE IF
   !SU3(ZL, ZE, ZX) THEN OUT(FLEX) ELSE IF !SU2(ZO, ZR) THEN OUT(FOR) ELSE IF
   !SU3(ZR, ZO, ZM) THEN OUT(FROM) ELSE SUC:= FALSE
END
ELSE IF !SU4(ZG, ZO, ZT, ZO) THEN OUT(GOTO) ELSE IF !SU4(ZH, ZE, ZA, ZP) THEN
OUT(HEAP) ELSE IF !S(ZI) THEN
BEGIN IF !SU3(ZS, ZN, ZT) THEN OUT(ISNT) ELSE IF !SU2(ZN, ZT) THEN OUT(INT) ELSE
   IF !SU(ZN) THEN OUT(VERT) ELSE IF !SU(ZM) THEN OUT(IM) ELSE IF !SU(ZF) THEN
   OUT(OPEN) ELSE IF !S(AP1) THEN OUT(PLIT) ELSE SUC:= FALSE
END
ELSE IF !S(ZL) THEN
BEGIN IF !SU2(ZO, ZC) THEN OUT(LOC) ELSE IF !SU2(ZW, ZB) THEN OUT(LWB) ELSE SUC:=
   FALSE
END
ELSE IF !SU4(ZM, ZO, ZD, ZE) THEN OUT(MODE) ELSE IF !SU3(ZN, ZI, ZL) THEN OUT(NIL)
ELSE IF !S(ZO) THEN
BEGIN IF !SU2(ZO, ZD) THEN OUT(ODD) ELSE IF !SU(ZF) THEN OUT(OF) ELSE IF !SU(ZP)
   THEN OUT(OP) ELSE IF !SU3(ZU, ZS, ZE) THEN OUT(AGN) ELSE IF !SU2(ZU, ZT)
   THEN OUT(VERT) ELSE SUC:= FALSE
END
ELSE IF !S(ZP) THEN
BEGIN IF !SU7(ZR, ZI, ZO, ZR, ZI, ZT, ZY) THEN OUT(PRIO) ELSE IF !SU3(ZR, ZO, ZC)
   THEN OUT(PROC) ELSE SUC:= FALSE
END
ELSE IF !S(ZR) THEN
BEGIN IF !SU3(ZE, ZA, ZL) THEN OUT(REAL) ELSE IF !SU2(ZE, ZF) THEN OUT(REF) ELSE
   IF !SU(ZE) THEN OUT(RE) ELSE IF !SU3(ZE, ZP, ZR) THEN OUT(REPR) ELSE IF
   !SU4(ZO, ZU, ZN, ZD) THEN OUT(RND) ELSE SUC:= FALSE
END
ELSE IF !S(ZS) THEN
BEGIN IF !SU3(ZI, ZG, ZN) THEN OUT(SGN) ELSE IF !SU3(ZK, ZI, ZP) THEN OUT(SKP)
   ELSE IF !SU5(ZT, ZR, ZI, ZN, ZG) THEN OUT(STRING) ELSE IF !SU5(ZT, ZR, ZU,
   ZC, ZT) THEN OUT(STR) ELSE SUC:= FALSE
END
ELSE IF !S(ZT) THEN
BEGIN IF !SU3(ZH, ZE, ZN) THEN OUT(VERT) ELSE IF !SU(ZO) THEN OUT(TO) ELSE IF
```

```
            ISU3(ZR, ZU, ZE) THEN OUT(TRUE) ELSE SUC:= FALSE
        END
        ELSE IF IS(ZU) THEN
        BEGIN IF ISU4(ZN, ZI, ZO, ZN) THEN OUT(UN) ELSE IF ISU2(ZP, ZB) THEN OUT(UPB) ELSE
           SUC:= FALSE
        END
        ELSE IF ISU4(ZV, ZO, ZI, ZD) THEN OUT(VOID) ELSE IF ISU5(ZW, ZH, ZI, ZL, ZE) THEN
        OUT(WHL) ELSE SUC:= FALSE; IF ¬SUC THEN P:= POLD1
    END
    ELSE IF IF IS(OPEN1) THEN TRUE ELSE IS(SUB) THEN OUT(OPEN) ELSE IF IF IS(CL1) THEN TRUE
    ELSE IS(BUS) THEN OUT(CL) ELSE IF IS(GOON1) THEN OUT(GOON) ELSE IF IS(COM1) THEN
    OUT(COM) ELSE IF IS(COL1) THEN OUT(IF IS2(EQ1, COL1) THEN ISS ELSE IF IS(EQ1) THEN BEC
    ELSE COL) ELSE IF IS(VERT1) THEN OUT(IF IS(COL1) THEN AGN ELSE VERT) ELSE IF
    IS(TIMS1) THEN OUT(IF IS(TIMS1) THEN UP ELSE IF IS2(COL1, EQ1) THEN TMAB ELSE TIMS)
    ELSE IF IS(PLUS1) THEN OUT(IF IS2(COL1, EQ1) THEN PSAB ELSE IF IS2(EQ1, COL1) THEN
    PLTO ELSE PLUS) ELSE IF IS(MIN1) THEN OUT(IF IS2(COL1, EQ1) THEN MAB ELSE MIN) ELSE IF
    IS(DIV1) THEN OUT(IF IS2(COL1, EQ1) THEN OVAB ELSE DIV) ELSE IF IS(PCT) THEN OUT(IF
    IS(TIMS1) THEN (IF IS2(COL1, EQ1) THEN MDAB ELSE MOD) ELSE IF IS2(COL1, EQ1) THEN OVAB
    ELSE OVER) ELSE IF IS(NOT1) THEN OUT(IF IS(EQ1) THEN NE ELSE NOT) ELSE IF IS(LESS1)
    THEN OUT(IF IS(EQ1) THEN LE ELSE LESS) ELSE IF IS(GT1) THEN OUT(IF IS(EQ1) THEN GE
    ELSE GT) ELSE IF IS(EQ1) THEN OUT(EQ) ELSE IF IS(OR1) THEN OUT(OR) ELSE IF IS(AND1)
    THEN OUT(AND) ELSE IF IS(PNT1) THEN OUT(PNT) ELSE IF IS(NRSGN) THEN
    BEGIN FOR I:= 1 WHILE ¬IS(NRSGN) DO
        BEGIN P:= P + 1; IF P > NA THEN ER(‡COMMENT‡) END
    END
    ELSE IF IS(QUOTE1) THEN
    BEGIN IF A[P + 1] = QUOTE1 THEN
        BEGIN INTEGER H;
           N:= 1; Q[1]:= A[P]; P:= P + 2; IF NEWITEM(H) THEN
           BEGIN LMIS:= LMIS + 2; IF LMIS > NMIS THEN ER(‡MISC ARRAY EXC.‡);
              MISC[LMIS - 1]:= 1; MISC[LMIS]:= Q[1]
           END;
           OUT(100000 * CHAR + H)
        END
        ELSE
        BEGIN INTEGER H;
           N:= 0;
           FOR I:= 1 WHILE ¬IS(QUOTE1) DO
           BEGIN N:= N + 1; IF N > NQ THEN ER(‡Q ARRAY EXC‡); Q[N]:= A[P]; P:= P + 1 END;
           IF NEWITEM(H) THEN
           BEGIN IF LMIS + N + 1 > NMIS THEN ER(‡MISC ARRAY EXC.‡); LMIS:= LMIS + 1;
              MISC[LMIS]:= N;
              FOR I:= 1 STEP 1 UNTIL N DO MISC[LMIS + I]:= Q[I]; LMIS:= LMIS + N
           END;
           OUT(100000 * STRING + H)
        END
    END
    ELSE SEEN1:= FALSE; IF ¬SUC THEN
    BEGIN J:= 0; P:= P + 1;
        FOR I:= 1 WHILE A[P] ≠ AP1 DO
        BEGIN J:= J + 1; P:= P + 1 END;
        OM(J); P:= P + 1
    END;
    IF P ≤ NA ∧ ¬SEEN1 ∧ SUC THEN
    BEGIN NLCR; PRINTTEXT(‡WRONG!‡); ABSFIXT(4, 0, A[P]); ABSFIXT(4, 0, P); P:= P + 1 END;
 END;
CSYM(119); CSYM(INTERROGATION); CSYM(119); X[0]:= PX; MISC[0]:= LMIS; RNR[0]:= LRNR; PROUT(X);
PROUT(MISC); CSYM(119); NLCR; NLCR;
FOR L:= 0 STEP 1 UNTIL LRNR DO
BEGIN PRINT(RNR[L]); CPUNCH(RNR[L]) END
```

```
END
END
END
END
```

```
BEGIN COMMENT ALGOL 68 INTERPRETER, PART 2, L. AMMERAAL, MACHINE DEPENDENT PROCEDURES: FIXT(M,N,X) -- PRINTS X WITH M
      DIGITS BEFORE AND N DIGITS AFTER THE POINT, ABSFIXT(M,N,X) -- LIKE FIXT BUT WITHOUT SIGN, NLCR -- NEW LINE
      CARRIAGE RETURN, NEWPAGE -- GIVES A NEW PAGE, SPACE(N) -- GIVES N BLANK SPACES ON THE CURRENT LINE, PRINT(X) --
      PRINTS X IN A STANDARD FORM, PRINTTEXT(¢S¢) -- PRINTS THE STRING S, EXIT -- TERMINATES THE PROGRAM EXECUTION,
      READ -- READS A NUMBER. NUMBERS MAY BE SEPARATED BY TWO OR MORE BLANKS (READ IS A REAL PROCEDURE WITHOUT
      PARAMETERS) RESYM -- READS A CHARACTER AND DELIVERS ITS INTEGRAL REPRESENTATION, (RESYM IS AN INTEGER PROCEDURE
      WITHOUT PARAMETERS) PRSYM(N) -- PRINTS THE CHARACTER WITH INTEGRAL REPRESENTATION N;
      INTEGER PLTO, PSAB, MAB, TMAB, DVAB, OVAB, MDAB, OR, AND, EQ, NE, LESS, GT, LE, GE, PLUS, MIN, TIMS, DIV, OVER,
      MOD, UP, PLIT, LWB, UPB, NOT, ABSV, ODD, SGN, RND, ENT, RE, IM, CONJ, REPR, ARG, NSQR, NEXP, NLN, NCOS, NACS,
      NSIN, NASN, NTAN, LRE, LIM, STBL, STIN, STRL, STCP, STCR, STNG, RINT, RRL, RSTG, NATN, STRW, STVD, JGLB, PART,
      FE1, FE2, LEN1, LEN2, MASP, NASP, II, PP, MM, V1, V2, MINL, M1, M2, BV1, BV2;
      BOOLEAN UNEQ, MORF;
      REAL PI, EV1, EV2, RE1, RE2, IM1, IM2;
      INTEGER ARRAY ARR[1:50];
      INTEGER AGN, AT, ATL, BEC, BITS, BOOL, BY, CHAR, CL, COL, COLL, COM, DO, DUM, EIT, CMPL, F, FALSE, FIRM, FLEX,
      FML, FOR, FROM, FS, G, GOON, GOTO, H, HEAP, I, J, IND, INT, IS, ISNT, L, LOC, MEEK, MODE, NEW, NFST, NIL, OF,
      OP, OPEN, OPROUT, P, PLAB, PNT, PRIO, PRNT, PROC, PX, OOLD, Q, REAL, REF, ROW, SKP, STR, STRING, STRONG, TAG,
      TO, TRUE, UN, V, VOID, VERT, VRL, WHL, Z, Z1, NA, NX, NX1, NL, NMIS, NM1, NRNR, NR1;
      EQR I:= 1 WHILE IRUE DO
      BEGIN
      START: AGN:= 132; AT:= 129; ATL:= - 1; BEC:= 84; BITS:= 107; BOOL:= 103; BY:= 170; CHAR:= 104; CL:= 119;
            COL:= 128; COLL:= - 2; COM:= 122; DIV:= 56; DO:= 173; DUM:= 1; EIT:= 98; ENT:= 62; EQ:= 82; F:= - 3;
            FALSE:= 39; FIRM:= - 4; FLEX:= 97; FML:= - 5; FOR:= 168; FROM:= 169; G:= - 6; GOON:= 163; GOTO:= 165;
            H:= - 7; HEAP:= 115; J:= 0; IND:= 4; INT:= 101; IS:= 87; ISNT:= 88; LESS:= 52; MIN:= 81; LOC:= 114;
            LWB:= 60; UPB:= 61; MEEK:= - 8; MODE:= 112; NEW:= - 9; NIL:= 167; OF:= 161; OP:= 116; OPEN:= 118;
            OPROUT:= - 14; OVER:= 57; P:= 1; PLUS:= 78; PNT:= 36; PRIO:= 113; PROC:= 99; REAL:= 102; REF:= 96;
            RND:= 74; ROW:= - 11; SKP:= 166; STR:= 95; STRING:= 109; STRONG:= - 12; TAG:= 3; TIMS:= 83; TO:= 171;
            TRUE:= 38; UN:= 100; VOID:= 92; VERT:= 131; VRL:= - 13; WHL:= 172; MASP:= 20; NA:= 2000; NX:= 1000;
            NL:= 500; NMIS:= 1000; NRNR:= 500; NFST:= 10; PLTO:= 42; PSAB:= 43; MAB:= 44; TMAB:= 45; DVAB:= 46;
            OVAB:= 47; MDAB:= 48; OR:= 49; AND:= 50; EQ:= 51; NE:= 52; LESS:= 53; GT:= 54; LE:= 55; GE:= 56; PLUS:= 57;
            MIN:= 58; TIMS:= 59; DIV:= 60; OVER:= 61; MOD:= 62; UP:= 63; PLIT:= 64; LWB:= 65; UPB:= 66; NOT:= 67;
            ABSV:= 68; ODD:= 69; SGN:= 70; RND:= 71; ENT:= 72; RE:= 73; IM:= 74; CONJ:= 75; ARG:= 76; REPR:= 77;
            PI:= 4 * ARCTAN(1); NSQR:= - 909; NEXP:= - 908; NLN:= - 907; NCOS:= - 906; NACS:= - 905; NSIN:= - 904;
            NASN:= - 903; NTAN:= - 902; NATN:= - 901; CMPL:= 106;
            BEGIN INTEGER ARRAY X[1:NX], MISC[1:NMIS], A, B, C, D[1:NA], LIND, VIND[1:NL];
                  REAL ARRAY E[1:NA], RNR[1:NRNR];
                  INTEGER ARRAY VASP, WASP[1:MASP];

                  PROCEDURE ADMC(JOLD); VALUE JOLD; INTEGER JOLD; COMMENT ADMINISTRATION TO CLOSE A RANGE;
                  BEGIN FOR J:= J STEP - 1 UNTIL JOLD + 2 DO RX(VIND[J]); J:= JOLD END;


                  PROCEDURE ADMO(JOLD); INTEGER JOLD; COMMENT ADMINISTRATION TO OPEN A RANGE;
                  BEGIN JOLD:= J; J:= J + 1; LIND[J]:= PART; VIND[J]:= - 1 END;


                  BOOLEAN PROCEDURE ANDP(A, B); BOOLEAN A, B; COMMENT ADVANCE P ONLY IF A AND B;
                  BEGIN INTEGER POLD;
                        POLD:= P; IF IF A THEN B ELSE FALSE THEN ANDP:= IRUE ELSE
                        BEGIN ANDP:= FALSE; P:= POLD END
                  END;


                  REAL PROCEDURE ARGC(R, I); VALUE R, I; REAL R, I; COMMENT ARGUMENT OF THE COMPLEX NUMBER (R,I);
                  ARGC:= IF ABS(R) > ABS(I) THEN ARCTAN(I / R) + PI / 2 * (IF I < 0 THEN SIGN(R) - 1 ELSE 1 - SIGN(R))
                  ELSE - ARCTAN(R / I) + PI / 2 * SIGN(I);


                  INTEGER PROCEDURE ASK;
```

```
BEGIN INTEGER V;
      IF FS = 0 THEN ER(¢NO FREE SPACE¢)); ASK:= V:= FS; FS:= D[V]; D[V]:= 1
END;


INTEGER PROCEDURE ASKN(N); VALUE N; INTEGER N;
BEGIN INTEGER V, V0, V1, VN, I;
      BOOLEAN EXH;
      V:= V1:= FS; V0:= - 1; EXH:= V = 0; I:= 1;
      FOR I:= 1 WHILE ¬EXH ∧ I < N DO
      BEGIN VN:= D[V]; IF VN = 0 THEN EXH:= TRUE ELSE
            BEGIN IF VN = V + 1 THEN I:= I + 1 ELSE
                  BEGIN I:= 1; V1:= VN; V0:= V END;
                  V:= VN
            END
      END;
      IF EXH THEN ER(¢NO FREE SPACE¢) ELSE
      BEGIN ASKN:= V1; IF V0 = - 1 THEN FS:= D[V] ELSE D[V0]:= D[V] END;
      FOR V:= V1 + N - 1 STEP - 1 UNTIL V1 DO D[V]:= 1
END;


PROCEDURE ASTL(Q); INTEGER Q; COMMENT ASSIGNATION TAIL;
BEGIN INTEGER VD, VC, VGT, VCT, I, VG, VM, VMT, FEC, N, NELT, FEG;
      IF SOFTC(Q) THEN
      BEGIN M(A[Z] = REF, ¢DEST NO NAME¢); VD:= Z; VC:= B[VD]; M(UNIT(Q), ¢NO SOURCE¢);
            M(COER(STRONG, VC, Q), ¢ASS MODE¢); IF Q = G THEN
            BEGIN VG:= Z; MORF:= FALSE; IF A[VC] = ROW THEN
                  BEGIN VGT:= B[VG]; VCT:= B[VC]; N:= C[VC]; IF A[VCT] = 1 THEN
                     BEGIN FOR I:= 1 STEP 1 UNTIL N DO M(A[VCT + I] = A[VGT + I] ∧ B[VCT + I] = B[VGT
                        + I], ¢MULT ASS BOUND¢)
                     END
                     ELSE
                     BEGIN VM:= NINST(VG); VMT:= B[VM]; A[VMT]:= A[VCT]; B[VD]:= VM; RX(VC); VC:= VM;
                        VCT:= B[VC]
                     END;
                     FEC:= B[VCT] + C[VCT] - 1; FEG:= B[VGT] + C[VGT] - 1; NELT:= E[VGT];
                     FOR I:= 1 STEP 1 UNTIL NELT DO SPS(FEG - I + I, FEC - I + I)
                  END
                  ELSE SPS(VG, VC); RX(VG); Z:= VD
            END
      END
      ELSE M(UNIT(F), ¢SOURCE¢)
END;


BOOLEAN PROCEDURE BOP(S, Y); VALUE S; INTEGER S; BOOLEAN Y; IF PP = S THEN
BEGIN BOP:= TRUE; PUTBOOL(Y) END
ELSE BOP:= FALSE;

PROCEDURE BOP2(S1, B1, S2, B2); INTEGER S1, S2; BOOLEAN B1, B2; IF PP = S1 THEN PUTBOOL(B1) ELSE
BEGIN M(PP = S2, ¢BOOL OP,¢); PUTBOOL(B2) END;


BOOLEAN PROCEDURE BOP4(S1, B1, S2, B2, S3, B3, S4, B4); INTEGER S1, S2, S3, S4; BOOLEAN B1, B2, B3, B4;
BEGIN BOP4:= TRUE;
      IF PP = S1 THEN PUTBOOL(B1) ELSE IF PP = S2 THEN PUTBOOL(B2) ELSE IF PP = S3 THEN PUTBOOL(B3)
      ELSE IF PP = S4 THEN PUTBOOL(B4) ELSE BOP4:= FALSE
END;
```

```
BOOLEAN PROCEDURE CAST(Q); INTEGER Q;
BEGIN INTEGER T;
      IF LK2(DCL(VRL, DUM, F), IF INP(OPEN) THEN TRUE ELSE INP(COL)) THEN
      BEGIN CAST:= TRUE; M(DCL(VRL, T, Q), ¢ CAST ¢); IF INP(COL) THEN M(UNIT(Q),
            ¢ NO UNIT IN CAST ¢) ELSE M(ENCL(Q), ¢ NO ENCLOSED CL. IN CAST ¢);
            IF Q = G THEN MORF:= FALSE; M(COER(STRONG, T, Q), ¢ CAST COERCION ¢)
      END
      ELSE CAST:= FALSE
END;


PROCEDURE CDIN(Q, QOUT); INTEGER Q, QOUT; COMMENT CONDITION IN CLAUSE;

BEGIN BOOLEAN BL;
      INTEGER Q0, Q1;
      BL:= B[Z] = 1; EMP; Q0:= Q1:= IF BL THEN G ELSE F; M(SCL(Q1), ¢COND.IN¢); IF Q1 ≠ Q0 THEN Q:= F;
      QOUT:= IF Q0 = G THEN F ELSE G
END;

PROCEDURE CFIN(Q, QOUT); INTEGER Q, QOUT; COMMENT CONFORMITY IN CLAUSE;
BEGIN INTEGER U, V, W, Q0, Q1, Q2, JOLD, L;
      BOOLEAN JMP, SUC, DONE;
      U:= Z; V:= C[U]; JMP:= DONE:= FALSE;
      FOR I:= 1, I WHILE INP(COM) DO
      BEGIN Q0:= Q1:= IF JMP ∨ DONE THEN F ELSE G; M(INP(OPEN), ¢OPEN IN CONF.IN CL.¢); ADMO(JOLD);
            M(DCL(FML, W, Q1), ¢DCL IN CONF. IN CL.¢); IF IN(TAG, L) THEN POS(L, W, Q1);
            SUC:= IF Q1 = G THEN MEQ(V, W) ELSE FALSE; Q2:= IF SUC THEN G ELSE F; M(INP(COL),
            ¢: IN CONF. IN CLAUSE MISSING¢); M(UNIT(Q2), ¢UNIT IN CONF. IN CLAUSE¢);
            IF SUC THEN DONE:= TRUE; IF Q0 = G ∧ Q2 = F THEN JMP:= TRUE; ADMC(JOLD)
      END;
      IF JMP THEN Q1:= F; RX(U); QOUT:= IF JMP ∨ DONE THEN F ELSE G
END;


PROCEDURE CHCL(Q); INTEGER Q; COMMENT CHOOSER CLAUSE;
BEGIN INTEGER AV, QOUT;
      IF Q = G THEN
      BEGIN FOR I:= 1 WHILE DREF ∨ DPRO(Q) DO ; AV:= A[Z];
            IF AV = BOOL THEN CDIN(Q, QOUT) ELSE IF AV = INT THEN CSIN(Q, QOUT) ELSE
            BEGIN M(AV = UN, ¢CHOISE MODE¢); CFIN(Q, QOUT); END;
            IF INP(VERT) THEN M(SCL(QOUT),
            ¢OUT CLAUSE¢) ELSE IF INP(AGN) THEN CHCL(QOUT) ELSE IF QOUT = G THEN
            BEGIN Z:= ASK; A[Z]:= VOID END
      END
      ELSE P:= MATCH - 1
END;


PROCEDURE CLLC(Q); INTEGER Q; COMMENT COLLATERAL CLAUSE ;
BEGIN INTEGER N, V, W, I;
      M(INP(COM), ¢AFTER SER.CLAUSE¢); V:= ASK; A[V]:= COLL; C[V]:= N:= LSTC(UNIT(F)) + 1;
      B[V]:= W:= ASKN(N); MOVE(Z, W); EMP;
      FOR I:= 2 STEP 1 UNTIL N DO
      BEGIN IF I > 2 THEN M(INP(COM), ¢COLL.CL.COMMA¢); M(UNIT(Q), ¢COLL.CLAUSE¢); IF Q = G THEN
            BEGIN MOVE(Z, W + I - 1); EMP END
      END;
      Z:= V; M(INP(CL), ¢ ) MISSING IN COLL. CL.¢)
END;
```

```
PROCEDURE CLTL(Q); INTEGER Q; COMMENT CALL TAIL;
BEGIN INTEGER V, W, N, PR, JOLD, PX0, I, POLD, J1, J2;
      REAL Y;
      MORF:= TRUE; V:= Z; W:= B[V]; N:= B[W]; PP:= PR:= C[W]; IF PP = -1000 THEN
      BEGIN M(UNIT(Q), $PRINT$); GOUT(Z); Z:= V; M(INP(CL), $PRINT CL$) END
      ELSE IF PP < -996 THEN
      BEGIN P:= MATCH; IF PP = -999 THEN NEWPAGE ELSE IF PP = -998 THEN NLCR ELSE SPACE(1) END
      ELSE IF PP ≤ NATN THEN
      BEGIN M(UNIT(Q), $ST,FUN$); M(INP(CL), $ST,F,CL,$); M(COER(STRONG, STRL, Q), $ARG,ST,FUN$);
            IF Q = G THEN
            BEGIN PP:= PR; Y:= E[Z];
                  IF ROP4(NSQR, SQRT(Y), NEXP, EXP(Y), NLN, LN(Y), NCOS, COS(Y)) THEN ELSE IF
                  ROP4(NACS, ARGC(Y, SQRT(1 - Y * Y)), NSIN, SIN(Y), NASN, ARGC(SQRT(1 - Y * Y),
                  Y), NTAN, SIN(Y) / COS(Y)) THEN ELSE M(ROP(NATN, ARCTAN(Y)), $ILL,ST,FUN$)
            END
      END
      ELSE
      BEGIN ADMO(JOLD); J1:= JGLB - 1; J2:= J;
            FOR J1:= J1 + 1 WHILE LIND[J1] ≠ PART DO ; INVS(J1, J2); PX0:= PX; P:= P - 1;
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN CTX(PR); WTX(EQ); CTX(P); WTX(IF I = N THEN GOON ELSE COM); END;
            POLD:= P; P:= PX0 + 1; M(IYDN(Q), $PARM IN CALL$); P:= POLD; M(INP(CL),
            $) MISSING IN CALL$); POLD:= P; P:= PR + 1; M(CAST(Q), $CALL$); P:= POLD; PX:= PX0;
            VISB(J1, J2); ADMC(JOLD)
      END
END;


BOOLEAN PROCEDURE COER(SORT, APST, Q); VALUE SORT, APST, Q; INTEGER SORT, APST, Q; COMMENT COERCION;
BEGIN COER:= TRUE;
      IF Q ≠ G THEN ELSE IF A[APST] = VOID ∧ SORT = STRONG THEN ELSE IF (A[Z] = VOID ∨ A[Z] = REF
      ∧ B[Z] = 0) ∧ SORT = STRONG THEN
      BEGIN EMP; Z:= NINST(APST) END
      ELSE IF A[Z] = COLL THEN DCOL(APST) ELSE
      BEGIN INTEGER APRI, I, N, V, V0, V1;
            BOOLEAN TRY, DONE;
            REAL R;
            APRI:= Z; TRY:= TRUE;
            FOR I:= 1 WHILE IF TRY THEN ¬MEQ(APRI, APST) ELSE FALSE DO
            BEGIN IF SORT = MEEK THEN
                  BEGIN TRY:= IF DREF THEN TRUE ELSE DPRO(Q); APRI:= Z END
                  ELSE
                  BEGIN COMMENT FIRM OR STRONG;
                     IF A[APST] = UN THEN
                     BEGIN V:= B[APST]; I:= 0; N:= C[V];
                        FOR I:= I WHILE IF I < N THEN ¬MEQ(B[V + I], APRI) ELSE FALSE DO I:= I + 1;
                        IF I < N THEN
                        BEGIN Z:= NINST(APST); C[Z]:= NINST(APRI); RX(APRI); APRI:= Z END
                        ELSE TRY:= FALSE
                     END
                     ELSE IF SORT = FIRM THEN
                     BEGIN IF A[APRI] = REF THEN
                        BEGIN IF MEQ(B[APRI], APST) THEN
                           BEGIN M(DREF, $COERC,DEREF,$); APRI:= Z END
                           ELSE TRY:= FALSE
                        END
                        ELSE IF A[APRI] = PROC THEN
                        BEGIN IF MEQ(B[B[APRI] + 1], APST) THEN M(DPRO(Q),
                           $COERC, DEPROC$) ELSE TRY:= FALSE
```

```
                        END
                        ELSE TRY:= FALSE
                    END
                    ELSE IF SORT = STRONG THEN
                    BEGIN IF DREF THEN APRI:= Z ELSE IF DPRO(Q) THEN APRI:= Z ELSE
                        BEGIN DONE:= FALSE; IF CPL1(APST, RE1, IM1) THEN
                            BEGIN IF A[APRI] = INT THEN
                                BEGIN R:= B[APRI]; PUTREAL(R); RX(APRI); APRI:= Z END;
                                IF A[APRI] = REAL THEN
                                BEGIN PUTCPL(E[APRI], .0); RX(APRI); DONE:= TRUE; APRI:= Z END
                            END;
                            IF DONE THEN  ELSE
                            BEGIN IF A[APST] = REAL ∧ A[APRI] = INT THEN
                                BEGIN R:= B[APRI]; PUTREAL(R); RX(APRI); APRI:= Z; DONE:= TRUE END
                                ELSE IF A[APST] = ROW ∧ A[APRI] ≠ ROW THEN
                                BEGIN V1:= NINST(APRI); N:= C[APST]; V0:= ASKN(N + 2); A[V0]:= ROW;
                                    B[V0]:= V1:= V0 + 1; C[V0]:= N; A[V1]:= 1; B[V1]:= V; C[V1]:= 1;
                                    FOR I:= 1 STEP 1 UNTIL N DO A[V1 + I]:= B[V1 + I]:= C[V1 + I]:= 1;
                                    E[V1]:= 1; RX(APRI); APRI:= Z:= V0; DONE:= TRUE
                                END
                                ELSE TRY:= FALSE
                            END;
                            IF ¬DONE THEN TRY:= FALSE
                        END
                    END
                END
            END;
            COER:= TRY
        END
END;


PROCEDURE COPGEN(I, A, B, X, Y); VALUE A, B; INTEGER I, A, B, X, Y;
FOR I:= A STEP 1 UNTIL B DO Y:= X;


PROCEDURE COPY(V, W); VALUE V, W; INTEGER V, W;
BEGIN A[W]:= A[V]; B[W]:= B[V]; C[W]:= C[V]; E[W]:= E[V] END;


PROCEDURE COPYN(V, W, N); VALUE V, W, N; INTEGER V, W, N;
BEGIN INTEGER I, M;
      M:= N - 1;
      FOR I:= 0 STEP 1 UNTIL M DO COPY(V + I, W + I)
END;


BOOLEAN PROCEDURE CPL1(V, RE, IM); VALUE V; INTEGER V; REAL RE, IM;
BEGIN INTEGER BV, V1, V2;
      CPL1:= FALSE; IF A[V] = STR THEN
      BEGIN IF C[V] = 2 THEN
            BEGIN BV:= B[V]; IF C[BV] = LRE ∧ C[BV + 1] = LIM THEN
                  BEGIN V1:= B[BV]; V2:= B[BV + 1]; IF A[V1] = REAL ∧ A[V2] = REAL THEN
                        BEGIN CPL1:= TRUE; RE:= E[V1]; IM:= E[V2] END
                  END
            END
      END
END;
```

```
BOOLEAN PROCEDURE CPL2; CPL2:= IF CPL1(V1, RE1, IM1) THEN CPL1(V2, RE2, IM2) ELSE FALSE;


BOOLEAN PROCEDURE CPOP(S, RE, IM); INTEGER S; REAL RE, IM; IF PP = S THEN
BEGIN CPOP:= TRUE; PUTCPL(RE, IM) END
ELSE CPOP:= FALSE;

PROCEDURE CSIN(Q, QOUT); INTEGER Q, QOUT; COMMENT CASE IN CLAUSE;

BEGIN INTEGER N, I, Q0, Q1;
      BOOLEAN JMP;
      N:= B[Z]; EMP; I:= 0; JMP:= FALSE;
      FOR I:= 1, I WHILE INP(COM) DO
      BEGIN I:= I + 1; Q0:= Q1:= IF I = N THEN G ELSE F; M(UNIT(Q1), $UNIT IN CASE CLAUSE$);
            IF Q0 = G ∧ Q1 = F THEN JMP:= TRUE
      END;
      IF JMP THEN Q:= F; QOUT:= IF JMP ∨ N ≥ 1 ∧ N ≤ I THEN F ELSE G
END;

PROCEDURE CTX(P1); INTEGER P1; COMMENT COPY TO X;
BEGIN INTEGER K, X1;
      K:= 0; P1:= P1 + 1; X1:= X[P1];
      FOR I:= 1 WHILE K > 0 ∨ (X1 ≠ COM ∧ X1 ≠ GOON ∧ X1 ≠ CL) DO
      BEGIN IF X1 = OPEN THEN K:= K + 1 ELSE IF X1 = CL THEN K:= K - 1; PX:= PX + 1; X[PX]:= X1;
            P1:= P1 + 1; X1:= X[P1]
      END
END;


BOOLEAN PROCEDURE DCL(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q; COMMENT DECLARER;
BEGIN INTEGER L, J, V, POLD, VSH, V1, T1, NEXT;
      BOOLEAN POS;

      BOOLEAN PROCEDURE O(S); VALUE S; INTEGER S; IF INP(S) THEN
      BEGIN O:= TRUE; IF Q = G THEN
            BEGIN T:= ASK; A[T]:= S; END
      END
      ELSE O:= FALSE; POLD:= P; IF INP(OPEN) THEN P:= MATCH; NEXT:= X[P];
      POS:= IF NEXT > 91 ∧ NEXT < 119 THEN TRUE ELSE IN(IND, L); P:= POLD; IF POS THEN
      BEGIN DCL:= TRUE;
            IF O(INT) THEN ELSE IF O(REAL) THEN ELSE IF O(BOOL) THEN ELSE IF O(CHAR) THEN ELSE
            IF O(VOID) THEN ELSE IF O(STRING) THEN
            BEGIN IF Q = G THEN
                  BEGIN A[T]:= ROW; B[T]:= V:= ASKN(3); C[T]:= 1; A[V]:= 0; B[V]:= V + 2; C[V]:= 1;
                        E[V]:= 1; V1:= V + 1; A[V1]:= 1; B[V1]:= 0; C[V1]:= 1; A[V + 2]:= CHAR
                  END
            END
            ELSE IF INP(CMPL) THEN
            BEGIN IF Q = G THEN
                  BEGIN PUTCPL(.0, .0); T:= Z END
            END
            ELSE IF STDR(VIC, T, Q) THEN ELSE IF RFDR(VIC, T, Q) THEN ELSE IF RWDR(VIC, T, Q) THEN
            ELSE IF PRDR(VIC, T, Q) THEN ELSE IF UNDR(VIC, T, Q) THEN ELSE IF IF IN(IND, L)
            THEN (IF SRCH(L, J) THEN A[VIND[J]] = NEW ELSE FALSE) ELSE FALSE THEN
            BEGIN IF Q = G THEN
                  BEGIN V:= VIND[J]; VSH:= B[V]; IF VSH < 0 THEN
                     BEGIN POLD:= P; P:= C[V]; B[V]:= V1:= ASK; M(DCL(ATL, T1, Q), $NEWMODE DCL$);
                        P:= POLD; COPY(T1, V1); FREE(T1); T:= V1; B[V]:= - 1
                     END
                     ELSE T:= VSH
```

```
                END
            END
            ELSE DCL:= FALSE
      END
      ELSE DCL:= FALSE
END;

PROCEDURE DCOL(APST); VALUE APST; INTEGER APST; COMMENT DECOLLATERATION;
BEGIN INTEGER VCOL, N, I, FECL, Z1, V1, V2, BAPS, STRV, MSTR, MZ;
      VCOL:= Z; N:= C[VCOL]; FECL:= B[VCOL]; IF A[APST] = STR THEN
      BEGIN M(C[APST] = N, ‡COLL.CL, TO STRUCT‡); Z:= ASKN(N + 1); BAPS:= B[APST]; COPY(APST, Z);
            Z1:= Z + 1; COPYN(BAPS, Z1, N); B[Z]:= Z1; FREE(VCOL);
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN Z:= FECL + 1 + I; STRV:= B[BAPS + 1 + I]; MSTR:= A[STRV]; MZ:= A[Z];
                  IF MZ = INT ∧ MSTR = REAL ∨ MZ = CHAR ∧ MSTR = ROW THEN M(COER(STRONG, STRV, 0),
                  ‡WID,IN COLL‡) ELSE IF MZ = COLL THEN DCOL(STRV); B[Z1 + 1 + I]:= Z
            END;
            Z:= Z1 + 1
      END
      ELSE
      BEGIN M(A[APST] = ROW, ‡COLL.CL NO ROW OR STRUCT‡); Z:= ASKN(N + 3); A[Z]:= ROW;
            B[Z]:= V1:= Z + 1; C[Z]:= 1; M(IF A[B[APST]] = 0 THEN IBUE ELSE A[B[APST] + 1] = 1,
            ‡COLL,TO ROW,BOUND‡); A[V1]:= 1; B[V1]:= V1 + 2; C[V1]:= 1; E[V1]:= N; V2:= V1 + 1;
            A[V2]:= 1; B[V2]:= N; C[V2]:= 1; COPYN(FECL, V2 + 1, N); FREE(VCOL); FREEN(FECL, N)
      END;
      RMCL(APST)
END;


BOOLEAN PROCEDURE DEN(Q); VALUE Q; INTEGER Q; COMMENT DENOTATION;
BEGIN INTEGER L;
      BOOLEAN SUC;
      SUC:= IBUE; IF IN(INT, L) THEN
      BEGIN IF Q = G THEN PUTINT(MISC[L + 1]) END
      ELSE IF IN(REAL, L) THEN
      BEGIN IF Q = G THEN PUTREAL(RNR[L]) END
      ELSE IF INP(TRUE) THEN
      BEGIN IF Q = G THEN PUTBOOL(IBUE) END
      ELSE IF INP(FALSE) THEN
      BEGIN IF Q = G THEN PUTBOOL(FALSE) END
      ELSE IF IN(CHAR, L) THEN
      BEGIN IF Q = G THEN PUTCHAR(MISC[L + 1]) END
      ELSE IF IN(STRING, L) THEN
      BEGIN IF Q = G THEN PUTSTRING(L) END
      ELSE IF IN(BITS, L) THEN
      BEGIN IF Q = G THEN PUTBITS(MISC[L]) END
      ELSE SUC:= FALSE; DEN:= SUC; IF Q = G ∧ SUC THEN MORF:= FALSE
END;


INTEGER PROCEDURE DEV(V); VALUE V; INTEGER V; COMMENT DEVELOP;
BEGIN INTEGER N, I, VI, VD, N1, VN, NN, IN;
      BOOLEAN U;
      NN:= N:= C[V]; U:= FALSE;
      FOR I:= 1 STEP 1 UNTIL N DO
      BEGIN VI:= V + I - 1; IF A[VI] = UN THEN
            BEGIN VD:= B[VI]; U:= IBUE; N1:= C[VD]; NN:= NN + N1 - 1 END
      END;
      IF U THEN
      BEGIN DEV:= VN:= ASKN(NN); IN:= 0;
```

```
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN V1:= V + I - 1; IF A[V1] = UN THEN
                  BEGIN VD:= B[V1]; N1:= C[VD];
                     FOR J:= 1 STEP 1 UNTIL N1 DO
                     BEGIN IN:= IN + 1; B[VN + IN - 1]:= B[VD + J - 1] END
                  END
                  ELSE
                  BEGIN IN:= IN + 1; B[VN + IN - 1]:= B[V1] END
            END;
            FREEN(V, N)
      END
      ELSE DEV:= V
END;


PROCEDURE DM(V); VALUE V; INTEGER V; COMMENT DISMISS;
BEGIN INTEGER AV, BV, CV, N, I, NV, BBV, ABV;
      AV:= A[V]; BV:= B[V]; CV:= C[V]; IF AV = REF THEN RX(BV) ELSE IF AV = PROC THEN
      BEGIN ABV:= A[BV]; BBV:= B[BV]; NV:= IF BBV ≥ 0 THEN BBV ELSE IF BBV = -10 THEN 1 ELSE 2;
            FOR I:= 0 STEP 1 UNTIL NV DO RS(B[BV + 1 + I]); FREEN(BV, NV + 2)
      END
      ELSE IF AV = STR THEN
      BEGIN FOR I:= 1 STEP 1 UNTIL CV DO RX(B[BV + I - 1]); FREEN(BV, CV) END
      ELSE IF AV = ROW THEN
      BEGIN FREEN(BV, CV + 1); RXN(B[BV], E[BV]) END
      ELSE IF AV = UN THEN
      BEGIN IF CV > 0 THEN RX(CV); N:= C[BV];
            FOR I:= 1 STEP 1 UNTIL N DO RS(B[BV + I - 1]); FREEN(BV, N)
      END
END;


BOOLEAN PROCEDURE DPRO(Q); INTEGER Q; COMMENT DEPROCED, ;
IF IF A[Z] = PROC THEN B[B[Z]] = 0 ∧ C[B[Z]] > 0 ELSE FALSE THEN
BEGIN INTEGER POLD, V;
      POLD:= P; V:= Z; P:= C[B[Z]]; M(CAST(Q), ¢DEPROC¢); RX(V); P:= POLD; DPRO:= TRUE
END
ELSE DPRO:= FALSE;

BOOLEAN PROCEDURE DREF; COMMENT DEREFERENCING;

IF A[Z] = REF THEN
BEGIN INTEGER V;
      V:= Z; Z:= NINST(B[V]); RX(V); DREF:= TRUE
END
ELSE DREF:= FALSE;

INTEGER PROCEDURE DTC; COMMENT DECLARER, TAG COUNT;
BEGIN INTEGER N, POLD;
      POLD:= P; IF IF DCL(FML, DUM, F) THEN IN(TAG, DUM) ELSE FALSE THEN
      BEGIN N:= 1;
            FOR I:= 1 WHILE IF IF INP(COM) THEN DCL(FML, DUM, F) ∨ TRUE ELSE FALSE THEN IN(TAG, DUM)
            ELSE FALSE DO N:= N + 1
      END
      ELSE N:= 0; DTC:= N; P:= POLD
END;


PROCEDURE EMP; RX(Z);
```

```
BOOLEAN PROCEDURE ENCL(Q); INTEGER Q; COMMENT ENCLOSED CLAUSE;
BEGIN INTEGER JOLD, POLD;
      ENCL:= FALSE; IF Q ≠ G THEN
      BEGIN IF INP(OPEN) THEN
            BEGIN ENCL:= TRUE; P:= MATCH END
      END
      ELSE IF INP(OPEN) THEN
      BEGIN POLD:= P; ADMO(JOLD); IF SCL(Q) THEN
            BEGIN ENCL:= TRUE; IF INP(CL) THEN  ELSE IF INP(VERT) THEN
                  BEGIN IF Q = G THEN
                     BEGIN CHCL(Q); M(INP(CL), ¢CHOOSER CL¢) END
                     ELSE P:= MATCH
                  END
                  ELSE IF Q = G THEN CLLC(Q) ELSE P:= MATCH; ADMC(JOLD)
            END
            ELSE P:= POLD - 1
      END
END;


PROCEDURE ER(S); STRING S;
BEGIN NLCR; PRINTTEXT(¢ERROR ! ¢); PRINTTEXT(S); SPACE(3); PRINTTEXT(¢ERROR AT POSITION¢);
      ABSFIXT(6, 0, P); NLCR; GOTO START
END;


PROCEDURE FILL;
BEGIN II:= II + 1; ARR[II]:= MM END;


BOOLEAN PROCEDURE FORM(Q); INTEGER Q; COMMENT FORMULA;
BEGIN INTEGER W, VDEX, ODEX, PRIO;
      INTEGER ARRAY OPST, PRST, VLST[0:NFST];

      BOOLEAN PROCEDURE OPND; COMMENT OPERAND;
      BEGIN INTEGER OPMD, PMD, W, U, J;
            REAL R;
            IF OP(10, 10, J, DUM) THEN
            BEGIN M(OPND, ¢NO OPERAND AFTER MON, OPERATOR¢); OPND:= TRUE; IF Q = G THEN
                  BEGIN FOR II:= 1, I WHILE IF COER(FIRM, PMD, Q) THEN FALSE ELSE FURT(LIND[J], = 10,
                     J) DO
                     BEGIN W:= B[VIND[J]]; PMD:= B[W + 2] END;
                     M(J > 0, ¢ILL.MON.OPTOR¢); PP:= C[W]; U:= Z; OPMD:= A[U];
                     IF PP > 0 THEN OPRT(1, PP, U, DUM) ELSE
                     BEGIN PP:= - PP; IF OPMD = INT THEN
                        BEGIN I:= B[U];
                           IF IOP4(ABSV, ABS(I), MIN, - I, PLUS, I, SGN, SIGN(I)) THEN  ELSE IF
                           BOP(ODD, I ≠ I 1 2 * 2) THEN  ELSE
                           BEGIN M(PP = REPR, ¢ILL.MON.INT OP¢); PUTCHAR(I) END
                        END
                        ELSE IF OPMD = REAL THEN
                        BEGIN R:= E[U]; I:= R;
                           IF ROP4(ABSV, ABS(R), MIN, - R, PLUS, R, 0, 0) THEN  ELSE M(IOP4(SGN,
                           SIGN(R), ENT, ENTIER(R), RND, 1, 0, 0), ¢ILL.MON.REAL OP¢)
                        END
                        ELSE IF OPMD = ROW THEN
                        BEGIN I:= B[U] + 1; IF IOP(UPB, B[I]) THEN  ELSE M(IOP(LWB, A[I]),
                           ¢ILL.MON.ROW OP¢)
                        END
```

```
                ELSE IF OPMD = CHAR THEN M(IOP(ABSV, B[U]),
                $ILL.MON CHAROP$) ELSE IF OPMD = BOOL THEN M(IOP(ABSV, B[U]),
                $ILL.MON.BOOL OP$) ELSE
                BEGIN M(CPL1(U, RE1, IM1), $ILL.MON.OP$);
                   IF ROP4(RE, RE1, IM, IM1, ABSV, SQRT(RE1 * RE1 + IM1 * IM1), 0, 0) THEN
                   ELSE IF CPOP(PLUS, RE1, IM1) THEN  ELSE IF CPOP(MIN, - RE1, - IM1)
                   THEN  ELSE IF CPOP(CONJ, RE1, - IM1) THEN  ELSE
                   BEGIN M(PP = ARG ^ (RE1 # 0 v IM1 # 0), $ILLMON.CPL OP$);
                      PUTREAL(ARGC(RE1, IM1))
                   END
                END;
                RX(U)
             END
          END
    END
    ELSE OPND:= SEC(Q)
END OPND;


BOOLEAN PROCEDURE OP(PRL, PRH, OPTR, PRIO); VALUE PRL, PRH; INTEGER PRL, PRH, OPTR, PRIO;
COMMENT OPERATOR;
IF Q = G THEN
BEGIN INTEGER JJ, POLD, L;

      BOOLEAN PROCEDURE DIFP;
      BEGIN PRIO:= - B[B[VIND[JJ]]]; DIFP:= PRIO < PRL v PRIO > PRH END;

      JJ:= J; POLD:= P; OP:= FALSE; IF IN(IND, L) THEN
      BEGIN FOR I:= 1 WHILE IF JJ > 0 THEN (IF IF L = LIND[JJ] THEN A[VIND[JJ]] = PROC ELSE
            FALSE THEN DIFP ELSE TRUE) ELSE FALSE DO JJ:= JJ - 1; IF JJ > 0 THEN
            BEGIN OP:= TRUE; OPTR:= JJ END
            ELSE P:= POLD
      END
END
ELSE OP:= IN(IND, DUM);

PROCEDURE OPRT(N, PRTN, V, W); VALUE N, PRTN, V, W; INTEGER N, PRTN, V, W;
BEGIN COMMENT OPERATOR ROUTINE;
      INTEGER PX0, POLD, I, JOLD;
      POLD:= P; ADMO(JOLD); PX0:= PX;
      FOR I:= 1 STEP 1 UNTIL N DO
      BEGIN CTX(PRTN); WTX(FQ); WTX(OPROUT); WTX(IF I = 1 THEN V ELSE W);
            WTX(IF I = N THEN GOON ELSE COM)
      END;
      P:= PX0 + 1; PX:= PX0; M(IYDN(Q), $OPERAND$); P:= PRTN + 1; M(CAST(Q), $OP DECL$);
      P:= POLD; ADMC(JOLD)
END;


PROCEDURE DYOP(V, O); INTEGER V, O; COMMENT DYADIC OPERATION;

BEGIN INTEGER PRY, W, MP1, MP2, L, J, ABV1, RR, X, K;
      REAL Y, DD, REP, IMP, REP1;
      BOOLEAN TRY;
      V2:= VLST[V]; V:= V - 1; V1:= VLST[V]; J:= OPST[O]; O:= O - 1; TRY:= TRUE; L:= LIND[J];
      FOR I:= 1 WHILE TRY DO
      BEGIN W:= B[VIND[J]]; MP1:= B[W + 2]; MP2:= B[W + 3]; Z:= V1; PRY:= B[W];
            IF COER(FIRM, MP1, Q) THEN
            BEGIN V1:= Z; Z:= V2; IF COER(FIRM, MP2, Q) THEN
               BEGIN V2:= Z; TRY:= FALSE END
```

```
      ELSE TRY:= FURT(L, PRY, J)
   END
   ELSE TRY:= FURT(L, PRY, J)
END;
M(J > 0, $ILL.DYADIC OPERATOR$); PP:= C[W]; IF PP > 0 THEN OPRT(2, PP, V1, V2) ELSE
BEGIN PP:= - PP; M1:= A[V1]; M2:= A[V2]; BV1:= B[V1]; BV2:= B[V2]; EV1:= E[V1];
   EV2:= E[V2]; IF PP < PLUS THEN
   BEGIN IF PP < EQ THEN
      BEGIN IF PP < OR THEN
         BEGIN IF PP = PLTO THEN
            BEGIN M(M2 = REF, $+=:$); M(STCH(V1, BV2), $ILL.OP,$); SUMS; SPS(Z, BV2);
               EMP; Z:= NRT(BV2)
            END
            ELSE
            BEGIN M(M1 = REF, $LEFT OP, NO NAME$); ABV1:= A[BV1]; IF ABV1 = INT THEN
               BEGIN M(M2 = INT, $RIGHT OP, NO INT$); X:= B[BV1];
                  IF !OP4(PSAB, X + BV2, MAB, X - BV2, TMAB, X * BV2, DVAB, X / BV2)
                  THEN  ELSE
                  BEGIN RR:= X - X / BV2 * BV2;
                     M(!OP(MDAB, IF RR < 0 THEN RR + ABS(BV2) ELSE RR), $IL ABOP$)
                  END;
                  SPS(Z, BV1); EMP; Z:= NRT(BV1)
               END
               ELSE IF ABV1 = REAL THEN
               BEGIN Y:= E[BV1]; IF M2 = INT THEN EV2:= BV2;
                  M(ROP4(PSAB, Y + EV2, MAB, Y - EV2, TMAB, Y * EV2, DVAB, Y / EV2),
                  $ILL.AB OP$); SPS(Z, BV1); EMP; Z:= NRT(BV1)
               END
               ELSE
               BEGIN M(STCH(BV1, V2) ^ PP = PSAB, $ILL.+!=$); SUMS; SPS(Z, BV1); EMP;
                  Z:= NRT(BV1)
               END
            END
         END
         ELSE BOP2(OR, BV1 = 1 v BV2 = 1, AND, BV1 = 1 ^ BV2 = 1)
      END
      ELSE IF M1 = INT ^ M2 = INT THEN
      BEGIN IF BOP4(EQ, BV1 = BV2, NE, BV1 # BV2, LESS, BV1 < BV2, GT, BV1 > BV2)
         THEN  ELSE BOP2(LE, BV1 ≤ BV2, GE, BV1 ≥ BV2)
      END
      ELSE IF RL2 THEN
      BEGIN IF BOP4(EQ, EV1 = EV2, NE, EV1 # EV2, LESS, EV1 < EV2, GT, EV1 > EV2)
         THEN  ELSE BOP2(LE, EV1 ≤ EV2, GE, EV1 ≥ EV2)
      END
      ELSE IF SCRL THEN
      BEGIN IF BOP4(EQ, IF UNEQ THEN LEN1 + LEN2 = 0 ELSE LEN1 = LEN2, NE, IF UNEQ
         THEN LEN1 + LEN2 > 0 ELSE LEN1 # LEN2, LESS, LSTR(FE1, LEN1, FE2,
         LEN2), GT, LSTR(FE2, LEN2, FE1, LEN1)) THEN  ELSE BOP2(LE, ¬LSTR(FE2, LEN2,
         FE1, LEN1), GE, ¬LSTR(FE1, LEN1, FE2, LEN2))
      END
      ELSE IF M1 = BOOL ^ M2 = BOOL THEN BOP2(EQ, BV1 = BV2, NE, BV1 # BV2) ELSE IF
      CPL2 THEN BOP2(EQ, RE1 = RE2 ^ IM1 = IM2, NE, RE1 # RE2 v IM1 # IM2)
   END PP<PLUS
   ELSE IF PP < PLIT THEN
   BEGIN IF M1 = INT ^ M2 = INT THEN
      BEGIN IF !OP4(PLUS, BV1 + BV2, MIN, BV1 - BV2, TIMS, BV1 * BV2, OVER, BV1 / BV2)
         THEN  ELSE IF ROP(DIV, BV1 / BV2) THEN  ELSE IF !OP(UP, BV1 ↑ BV2) THEN
         ELSE
         BEGIN RR:= BV1 - BV1 / BV2 * BV2;
            PUTINT(IF RR < 0 THEN RR + ABS(BV2) ELSE RR)
```

```
                END
              END
              ELSE IF STCH(V1, V2) THEN
              BEGIN M(PP = PLUS, $ILL CHAR OR STRING OP$); SUMS END
              ELSE IF PP < UP THEN
              BEGIN IF RL2 THEN M(ROP4(PLUS, EV1 + EV2, MIN, EV1 - EV2, TIMS, EV1 * EV2, DIV,
                 EV1 / EV2), $REAL OVER OR MOD$) ELSE IF CPL2 THEN
                 BEGIN IF CPOP(PLUS, RE1 + RE2, IM1 + IM2) THEN  ELSE IF CPOP(MIN, RE1 - RE2,
                    IM1 - IM2) THEN  ELSE IF CPOP(TIMS, RE1 * RE2 - IM1 * IM2, RE1 * IM2 +
                    IM1 * RE2) THEN  ELSE
                    BEGIN DD:= RE2 * RE2 + IM2 * IM2;
                       M(CPOP(DIV, (RE1 * RE2 + IM1 * IM2) / DD, ( - RE1 * IM2 + IM1 * RE2)
                       / DD), $ILL.COMPLEX OP.$)
                    END
                 END
              END
              ELSE
              BEGIN M(M2 = INT, $EXP NO INT$); IF M1 = REAL THEN PUTREAL(EV1 ↑ BV2) ELSE
                 BEGIN M(CPL1(V1, RE1, IM1), $ILL.UP$); REP:= 1; IMP:= 0; K:= ABS(BV2);
                    FOR I:= 1 STEP 1 UNTIL K DO
                    BEGIN REP1:= REP * RE1 - IMP * IM1; IMP:= IMP * RE1 + REP * IM1;
                       REP:= REP1
                    END;
                    IF BV2 < 0 THEN
                    BEGIN DD:= REP * REP + IMP * IMP; PUTCPL(REP / DD, - IMP / DD) END
                    ELSE PUTCPL(REP, IMP)
                 END
              END UP
           END
           ELSE IF PP = PLIT THEN
           BEGIN IF M1 = INT THEN EV1:= BV1 ELSE M(M1 = REAL, $PLIT$);
              IF M2 = INT THEN EV2:= BV2 ELSE M(M2 = REAL, $PLIT$); PUTCPL(EV1, EV2)
           END PLIT
           ELSE
           BEGIN M((PP = LWB ∨ PP = UPB) ∧ BV1 ≥ 1 ∧ BV1 ≤ C[V2] ∧ M1 = INT ∧ M2 = ROW,
              $ILL.DYADIC OP.$); PUTINT(IF PP = LWB THEN A[BV2 + BV1] ELSE B[BV2 + BV1])
           END LWB OR UPB;
           RX(V1); RX(V2)
     END;
     VLST[V]:= Z
END DYOP;

PROCEDURE STVL;
BEGIN VDEX:= VDEX + 1; M(VDEX ≤ NFST, $VALUE STACK EXC.$); VLST[VDEX]:= Z END;


PROCEDURE STOR;
BEGIN ODEX:= ODEX + 1; M(ODEX ≤ NFST, $OP.STACK EXC.$); OPST[ODEX]:= W; PRST[ODEX]:= PRIO END;

COMMENT FORMULA STARTS HERE;
IF IF LOOK(OP(10, 10, DUM, DUM)) THEN TRUE ELSE LK2(SEC(F), OP(0, 9, DUM, DUM)) THEN
BEGIN FORM:= TRUE; M(OPND, $NO OPERAND$); IF Q = G THEN
     BEGIN VDEX:= ODEX:= 0; PRST[0]:= 0; STVL; MORF:= TRUE;
           FOR I:= 1 WHILE OP(0, 9, W, PRIO) DO
           BEGIN FOR I:= 1 WHILE PRIO ≤ PRST[ODEX] DO DYOP(VDEX, ODEX); STOR; M(OPND,
              $NO OPERAND AFTER OPERATOR$); STVL
           END;
           FOR I:= 1 WHILE VDEX > 1 DO DYOP(VDEX, ODEX)
     END
     ELSE
```

```
            FOR I:= 1 WHILE ANDP(OP(0, 9, DUM, DUM), OPND) DO;
      END
      ELSE FORM:= FALSE
END FORM;


PROCEDURE FREE(V); VALUE V; INTEGER V;
BEGIN D[V]:= FS; FS:= V END;


PROCEDURE FREEN(V, N); VALUE V, N; INTEGER V, N;
BEGIN INTEGER W;
      W:= V + N - 1;
      FOR V:= V STEP 1 UNTIL W DO FREE(V)
END;


BOOLEAN PROCEDURE FURT(L, PR, J); VALUE L, PR; INTEGER L, PR, J; COMMENT FURTHER SEARCH FOR OPERATOR;
BEGIN J:= J - 1;
      FOR I:= 1 WHILE IF J > 0 THEN (IF L = LIND[J] ^ A[VIND[J]] = PROC THEN B[B[VIND[J]]] ≠ PR ELSE
      TRUE) ELSE FALSE DO J:= J - 1; FURT:= J > 0
END;


BOOLEAN PROCEDURE GEN(Q); VALUE Q; INTEGER Q; COMMENT GENERATOR;
IF IF INP(LOC) THEN TRUE ELSE INP(HEAP) THEN
BEGIN INTEGER T;
      GEN:= TRUE; M(DCL(ATL, T, Q), $NO DCL IN GEN$); IF Q = G THEN
      BEGIN Z:= NRT(T); MORF:= FALSE END
END
ELSE GEN:= FALSE;

PROCEDURE GOUT(V); VALUE V; INTEGER V; COMMENT GENERAL OUTPUT;

BEGIN INTEGER AZ, BZ, CZ, Q, N, FE, I;
      Q:= G; Z:= V;
      FOR I:= 1 WHILE DREF ∨ DPRO(Q) DO ; M(Q = G, $IMPL,RESTR,OUTPUT JUMP$); AZ:= A[Z]; BZ:= B[Z];
      CZ:= C[Z]; IF AZ = STR THEN
      BEGIN FOR I:= 1 STEP 1 UNTIL CZ DO GOUT(B[BZ - 1 + I]) END
      ELSE IF AZ = ROW THEN
      BEGIN N:= E[BZ]; FE:= B[BZ];
            FOR I:= 1 STEP 1 UNTIL N DO GOUT(FE - 1 + I)
      END
      ELSE IF AZ = UN THEN GOUT(CZ) ELSE IF AZ = COLL THEN
      BEGIN FOR I:= 1 STEP 1 UNTIL CZ DO GOUT(BZ - 1 + I) END
      ELSE IF AZ = INT ∨ AZ = BOOL THEN PRINT(BZ) ELSE IF AZ = REAL THEN PRINT(E[Z]) ELSE IF AZ =
      CHAR THEN PRSYM(BZ) ELSE IF AZ = PROC THEN
      BEGIN PP:= C[BZ];
            IF PP = - 999 THEN NEWPAGE ELSE IF PP = - 998 THEN NLCR ELSE IF PP = - 997 THEN SPACE(1)
            ELSE ER($ILL.OUTP.$)
      END
      ELSE ER($ILL.OUTPUT$)
END;

BOOLEAN PROCEDURE HIP(Q); INTEGER Q;
HIP:= IF SKIP(Q) THEN TRUE ELSE IF JUMP(Q) THEN TRUE ELSE IF NIML(Q) THEN TRUE ELSE VAC(Q);


BOOLEAN PROCEDURE IDF(Q); VALUE Q; INTEGER Q; COMMENT MODE IDENTIFIER;
BEGIN INTEGER L, J, POLD;
```

```
      POLD:= P; IF IN(TAG, L) THEN
      BEGIN IDF:= TRUE; IF Q = G THEN
            BEGIN IF SRCH(L, J) THEN
                  BEGIN Z:= VIND[J]; IF Z < 0 THEN
                     BEGIN IDF:= FALSE; P:= POLD END
                     ELSE
                     BEGIN D[Z]:= D[Z] + 1; MORF:= TRUE END
                  END
                  ELSE
                  BEGIN NLCR; NLCR;
                     FOR J:= 1 STEP 1 UNTIL MISC[L] DO ABSFIXT(4, 0, MISC[L + J]); ER(
                     (UNKNOWN IDENTIFIER))
                  END
            END
      END
      ELSE IDF:= FALSE
END;

BOOLEAN PROCEDURE IN(K, L); VALUE K; INTEGER K, L; COMMENT READS NEXT ELEMENT OF X, IF THIS HAS MARK K ;
BEGIN INTEGER XX;
      XX:= X[P]; IF XX / 100000 = K THEN
      BEGIN IN:= TRUE; P:= P + 1; L:= XX - K * 100000 END
      ELSE IN:= FALSE
END;

BOOLEAN PROCEDURE INP(S); VALUE S; INTEGER S; COMMENT READS NEXT ELEMENT OF X, IF THIS IS S ;
BEGIN INTEGER L;
      IF X[P] = S THEN
      BEGIN INP:= TRUE; P:= P + 1 END
      ELSE INP:= IF S = EQ THEN ANDP(IN(IND, L), L = 10) ELSE FALSE
END;

PROCEDURE INVS(J1, J2); VALUE J1, J2; INTEGER J1, J2; COMMENT MAKES PART OF THE LINK TABLE INVISIBLE;
BEGIN INTEGER JJ;
      FOR JJ:= J1 + 1 STEP 1 UNTIL J2 DO IF LIND[JJ] < 10000 THEN LIND[JJ]:= LIND[JJ] + 10000
END;

BOOLEAN PROCEDURE IOP(S, Y); VALUE S; INTEGER S, Y; IF PP = S THEN
BEGIN IOP:= TRUE; PUTINT(Y) END
ELSE IOP:= FALSE;

BOOLEAN PROCEDURE IOP4(S1, I1, S2, I2, S3, I3, S4, I4); INTEGER S1, I1, S2, I2, S3, I3, S4, I4;
BEGIN IOP4:= TRUE;
      IF PP = S1 THEN PUTINT(I1) ELSE IF PP = S2 THEN PUTINT(I2) ELSE IF PP = S3 THEN PUTINT(I3)
      ELSE IF PP = S4 THEN PUTINT(I4) ELSE IOP4:= FALSE.
END;

BOOLEAN PROCEDURE IRDN(Q); VALUE Q; INTEGER Q; IRDN:= IF IVDN(Q) THEN TRUE ELSE VDN(Q);

COMMENT IDENTIFIER DECLARATION;

PROCEDURE IRTL(Q, IT); VALUE IT; INTEGER Q, IT; COMMENT IDENTITY RELATION TAIL;
BEGIN BOOLEAN IS1, IDNT;
      INTEGER I, J, K, U1, U2;
```

```
      INTEGER ARRAY N, U, V[1:2];
      IF Q ≠ G THEN M(TERT(F), ¢IDENT.REL.¢) ELSE
      BEGIN MORF:= FALSE;
            FOR I:= 1 WHILE DPRO(Q) DO ; U1:= V[1]:= Z; IS1:= IT = IS; M(TERT(Q), ¢ID,IR,M,S,¢);
            FOR I:= 1 WHILE DPRO(Q) DO ; U2:= V[2]:= Z; M(A[U1] = REF ∧ A[Z] = REF,
            ¢NO NAMES IN ID, REL.¢); IF B[U1] = B[Z] THEN IDNT:= TRUE ELSE
            BEGIN FOR I:= 1, 2 DO
                  BEGIN V[I]:= U[I]; N[I]:= 1;
                     FOR I:= 1 WHILE A[V[I]] = REF DO
                     BEGIN N[I]:= N[I] + 1; V[I]:= B[V[I]] END
                  END;
                  J:= IF N[1] > N(2) THEN 1 ELSE 2; K:= 3 - J;
                  FOR I:= N[J] - N[K] STEP - 1 UNTIL 1 DO U[J]:= B[U[J]]; IDNT:= B[U[J]] = B[U[K]]
            END;
            RX(U1); RX(U2); Z:= ASK; A[Z]:= BOOL; B[Z]:= IF IDNT ≡ IS1 THEN 1 ELSE 0
      END
END;


BOOLEAN PROCEDURE IYDF(Q, V); VALUE Q, V; INTEGER Q, V; COMMENT IDENTITY DEFINITION;
BEGIN INTEGER L;
      IF ANDP(IN(TAG, L), INP(EQ)) THEN
      BEGIN IYDF:= TRUE; IF Q = G THEN
            BEGIN IF INP(OPROUT) THEN
                  BEGIN Z:= X[P]; P:= P + 1 END
                  ELSE M(UNIT(Q), ¢NO UNIT IN IDENTITY DEF¢); M(COER(STRONG, V, Q),
                  ¢MODE IN IDENTITY DEF¢); POS(L + 10000, Z, Q)
            END
            ELSE IF INP(OPROUT) THEN P:= P + 1 ELSE M(UNIT(F), ¢IDENTITY DEF¢)
      END
      ELSE IYDF:= FALSE
END;


BOOLEAN PROCEDURE IYDN(Q); VALUE Q; INTEGER Q; COMMENT IDENTITY DECLARATION;
BEGIN BOOLEAN SUC;
      INTEGER JOLD;
      JOLD:= J;
      IYDN:= SUC:= IF IF Q = F THEN TRUE ELSE LOOK(IYDN(F)) THEN (IF LIST(PIDN(Q)) THEN TRUE ELSE
      LIST(MIDN(Q))) ELSE FALSE; IF SUC THEN VISB(JOLD, J)
END;


BOOLEAN PROCEDURE JUMP(Q); INTEGER Q;
BEGIN INTEGER L, J;
      IF INP(GOTO) THEN ; IF IN(TAG, L) THEN
      BEGIN JUMP:= TRUE; IF Q = G THEN
            BEGIN Q:= F; M(SRCH(L, J), ¢LABEL NOT FOUND¢); PLAB:= - VIND[J]; M(PLAB > 0, ¢LABEL¢)
            END
      END
      ELSE JUMP:= FALSE
END;


BOOLEAN PROCEDURE LAB(Q); VALUE Q; INTEGER Q; COMMENT DEFINING OCC. OF LABELS;
BEGIN INTEGER L;
      IF ANDP(IN(TAG, L), INP(COL)) THEN
      BEGIN LAB:= TRUE; IF Q = H THEN
            BEGIN J:= J + 1; LIND[J]:= L; VIND[J]:= - P END
      END
```

```
      ELSE LAB:= FALSE
END;


BOOLEAN PROCEDURE LBUN(Q); INTEGER Q;
BEGIN IF SEQ(LAB(Q)) THEN ; LBUN:= UNIT(Q) END;

COMMENT LABELLED UNIT;

BOOLEAN PROCEDURE LIST(B); BOOLEAN B; IF B THEN
BEGIN FOR I:= 1 WHILE ANDP(INP(COM), B) DO ; LIST:= TRUE END
ELSE LIST:= FALSE;

BOOLEAN PROCEDURE LK2(A, B); BOOLEAN A, B;
BEGIN INTEGER POLD;
      POLD:= P; LK2:= IF A THEN B ELSE FALSE; P:= POLD
END;


BOOLEAN PROCEDURE LOOK(A); BOOLEAN A;
BEGIN INTEGER POLD;
      POLD:= P; LOOK:= A; P:= POLD
END;


BOOLEAN PROCEDURE LOOP(Q); INTEGER Q;
BEGIN INTEGER L, V, ST, ENDV, P1, Q1, Q0, JOLD, P2, CV, Z1;
      BOOLEAN EX1, EXH, HALT, BND, FRST, JMP;

      BOOLEAN PROCEDURE CONT;
      BEGIN IF Q1 = G THEN CV:= B[V]; Q0:= Q1; IF ¬FRST THEN P:= P1;
            EXH:= IF Q1 ≠ G THEN TRUE ELSE IF ¬BND ∨ ST = 0 THEN FALSE ELSE SIGN(ST) * (CV - ENDV)
            > 0; IF INP(WHL) THEN
            BEGIN M(SCL(Q1), ‡AFTER WHILE‡); JMP:= Q1 ≠ Q0; M(COER(MEEK, STBL, Q1),
                  ‡MODE AFTER WHILE‡); EXH:= EXH ∨ B[Z] = 0
            END;
            M(INP(DO), ‡DO MISSING‡); IF FRST THEN
            BEGIN CONT:= TRUE; IF Q1 = G ∧ EXH THEN Q1:= F; FRST:= FALSE END
            ELSE
            BEGIN HALT:= Q1 ≠ G ∨ EXH; IF HALT THEN P:= P2; CONT:= ¬HALT END
      END;

      IF (IF SPY(FOR) THEN TRUE ELSE IF SPY(FROM) THEN TRUE ELSE IF SPY(BY) THEN TRUE ELSE IF SPY(TO)
      THEN TRUE ELSE IF SPY(WHL) THEN TRUE ELSE SPY(DO)) THEN
      BEGIN LOOP:= TRUE; EX1:= Q = G; IF EX1 THEN
            BEGIN ADMO(JOLD); V:= ASK; A[V]:= INT END;
            IF INP(FOR) THEN
            BEGIN M(IN(TAG, L), ‡NO ID AFTER FOR‡); POS(L, V, 9) END;
            IF INP(FROM) THEN
            BEGIN M(UNIT(Q), ‡UNIT AFTER FROM‡); IF EX1 THEN B[V]:= MEEKINT(Q) END
            ELSE IF EX1 THEN B[V]:= 1; IF INP(BY) THEN
            BEGIN M(UNIT(Q), ‡UNIT AFTER BY‡); IF EX1 THEN ST:= MEEKINT(Q) END
            ELSE IF EX1 THEN ST:= 1; IF INP(TO) THEN
            BEGIN M(UNIT(Q), ‡UNIT AFTER TO‡); IF EX1 THEN
                  BEGIN ENDV:= MEEKINT(Q); BND:= TRUE END
            END
            ELSE BND:= FALSE; JMP:= FALSE; P1:= P; FRST:= TRUE; Q1:= Q;
            FOR I:= 1 WHILE CONT DO
            BEGIN Q0:= Q1; M(UNIT(Q1), ‡UNIT AFTER DO‡); P2:= P; Z1:= Z; IF Q1 = G THEN
                  BEGIN B[V]:= B[V] + ST END
```

```
            END;
            IF JMP ∨ Q1 ≠ Q0 THEN Q:= F; IF EX1 THEN ADMC(JOLD)
      END
      ELSE LOOP:= FALSE
END;


INTEGER PROCEDURE LSTC(B); BOOLEAN B; COMMENT COUNTS THE MEMBERS OF A LIST;

BEGIN INTEGER K, POLD;
      POLD:= P; IF B THEN
      BEGIN K:= 1;
            EQB ::= ! WHILE ANDP(INP(COM), B) DO K:= K + 1; P:= POLD; LSTC:= K
      END
      ELSE LSTC:= 0
END;

BOOLEAN PROCEDURE LSTR(F1, L1, F2, L2); VALUE F1, L1, F2, L2; INTEGER F1, L1, F2, L2;
COMMENT FOR COMPARISON OF STRINGS;
LSTR:= IF UNEQ THEN B[F1 - 1 + I] < B[F2 - 1 + I] ELSE L1 < L2;


PROCEDURE M(B, S); BOOLEAN B; STRING S; COMMENT MUST;
IF ¬B THEN ER(S);


BOOLEAN PROCEDURE MADE(V, W); VALUE V, W; INTEGER V, W; COMMENT LOOK IF ASSUMPTION OF MODE EQ, WAS MADE;
BEGIN INTEGER I;
      I:= 0;
      EQB I:= I + 1 WHILE IF I ≤ NASP THEN ¬(V = VASP[I] ∧ W = WASP[I] ∨ V = WASP[I] ∧ W = VASP[I])
      ELSE FALSE DO ; MADE:= I ≤ NASP
END;


PROCEDURE MAKE(V, W); VALUE V, W; INTEGER V, W; COMMENT MAKE ASSUMPTION: MODES EQ.;
BEGIN NASP:= NASP + 1; M(NASP ≤ MASP, ‡SPACE FOR MODE EQ, EXH.‡); VASP[NASP]:= V; WASP[NASP]:= W END;


INTEGER PROCEDURE MATCH; COMMENT SKIP TO MATCHING CLOSE SYMBOL;
BEGIN INTEGER I;
      I:= 1;
      EQB I:= I WHILE I > 0 DO IF INP(CL) THEN I:= I - 1 ELSE IF INP(OPEN) THEN I:= I + 1 ELSE P:= P
      + 1; MATCH:= P
END;


BOOLEAN PROCEDURE MDF(Q); VALUE Q; INTEGER Q; COMMENT MODE DEFINITION;
BEGIN INTEGER L, P1, V;
      IF IN(IND, L) THEN
      BEGIN MDF:= TRUE; M(INP(EQ), ‡ NO= IN MODE DEF.‡); P1:= P; M(DCL(ATL, DUM, F),
            ‡ NO ACT DCL IN MODEDEF ‡); IF Q = H THEN
            BEGIN V:= ASK; A[V]:= NEW; B[V]:= - 1; C[V]:= P1; POS(L, V, G) END
      END
      ELSE MDF:= FALSE
END;


BOOLEAN PROCEDURE MDN(Q); VALUE Q; INTEGER Q; COMMENT MODE DECLARATION;
IF INP(MODE) THEN
BEGIN MDN:= TRUE; M(LIST(MDF(Q)), ‡ NO MODE DEF ‡) END
```

```
ELSE MON:= FALSE;

INTEGER PROCEDURE MEEKINT(Q); INTEGER Q; COMMENT MEEK COERCION TO INTEGRAL;

BEGIN M(COER(MEEK, STIN, Q), $ NO MEEK INT UNIT WHERE REQUIRED $); MEEKINT:= B[Z]; EMP END;

BOOLEAN PROCEDURE MEQ(V, W); VALUE V, W; INTEGER V, W;
BEGIN COMMENT MODE EQUIVALENCE;
      NASP:= 0; MEQ:= MEQL(V, W)
END;


BOOLEAN PROCEDURE MEQL(V, W); VALUE V, W; INTEGER V, W; COMMENT MODE EQ, USED BY MEQ;
BEGIN INTEGER AV, AW, BV, BW, CV, CW, I, NV, NW;
      AV:= A[V]; AW:= A[W]; IF V = W ∨ AV = ROW ∧ W = STRW THEN MEQL:= TRUE ELSE IF AV = AW THEN
      BEGIN BV:= B[V]; BW:= B[W]; CV:= C[V]; CW:= C[W];
            IF AV = REF THEN MEQL:= MEQL(BV, BW) ELSE IF AV = STR THEN
            BEGIN IF MADE(V, W) THEN MEQL:= TRUE ELSE IF CV = CW THEN
                  BEGIN MAKE(V, W); I:= - 1;
                     FOR I:= I + 1 WHILE IF I = CV THEN FALSE ELSE IF C[BV + I] ≠ C[BW + I] THEN
                     FALSE ELSE MEQL(B[BV + I], B[BW + I]) DO ; MEQL:= I = CV
                  END
                  ELSE MEQL:= FALSE
            END
            ELSE IF AV = ROW THEN MEQL:= IF CV = CW THEN MEQL(B[BV], B[BW]) ELSE FALSE ELSE IF AV =
            UN THEN
            BEGIN NV:= C[BV]; NW:= C[BW]; I:= - 1;
                  FOR I:= I + 1 WHILE IF I = NV THEN FALSE ELSE UNMB(BV + I, BW) DO ; IF I = NV THEN
                  BEGIN I:= - 1;
                     FOR I:= I + 1 WHILE IF I = NW THEN FALSE ELSE UNMB(BW + I, BV) DO ;
                     MEQL:= I = NW
                  END
                  ELSE MEQL:= FALSE
            END
            ELSE IF AV = PROC THEN
            BEGIN NV:= B[BV]; IF NV = - 10 THEN NV:= 1 ELSE IF NV < 0 THEN NV:= 2; NW:= B[BW];
                  IF NW = - 10 THEN NW:= 1 ELSE IF NW < 0 THEN NW:= 2; IF NV = NW THEN
                  BEGIN I:= 0;
                     FOR I:= I + 1 WHILE IF I = NV + 2 THEN FALSE ELSE MEQL(B[BV + I], B[BW + I]) DO ;
                     MEQL:= I = NV + 2
                  END
                  ELSE MEQL:= FALSE
            END
            ELSE MEQL:= TRUE
      END
      ELSE MEQL:= FALSE
END;

COMMENT MODE IDENTITY DECLARATION;

BOOLEAN PROCEDURE MIDN(Q); VALUE Q; INTEGER Q;
BEGIN INTEGER V, POLD;
      BOOLEAN SUC;
      POLD:= P; SUC:= FALSE; IF DCL(FML, V, Q) THEN
      BEGIN SUC:= LIST(IYDF(Q, V)); IF Q = G THEN RS(V) END;
      MIDN:= SUC; IF ¬SUC THEN P:= POLD
END;


PROCEDURE MOVE(V, W); VALUE V, W; INTEGER V, W;
```

```
BEGIN INTEGER AV, BV, CV, U, W1, V1, N, FE2, I;
      AV:= A[V]; BV:= B[V]; CV:= C[V]; COPY(V, W); IF AV = REF THEN
      BEGIN U:= BV; D[U]:= D[U] + 1 END
      ELSE IF AV = STR THEN
      BEGIN W1:= B[W]:= ASKN(CV); COPYN(BV, W1, CV);
            FOR I:= 1 STEP 1 UNTIL CV DO B[W1 + I - 1]:= NINST(B[BV + I - 1])
      END
      ELSE IF AV = ROW THEN
      BEGIN B[W]:= W1:= ASKN(CV + 1); COPYN(BV, W1, CV + 1); N:= E[BV]; IF N = 0 THEN N:= 1;
            B[W1]:= FE2:= ASKN(N); MOVEN(B[BV], FE2, N)
      END
      ELSE IF AV = UN THEN
      BEGIN N:= C[BV]; B[W]:= U:= ASKN(N); C[U]:= N; IF CV > 0 THEN C[W]:= NINST(CV);
            FOR I:= 1 STEP 1 UNTIL N DO B[U - 1 + I]:= NINST(B[BV - 1 + I])
      END
      ELSE IF AV = PROC THEN
      BEGIN N:= B[BV]; B[W]:= W1:= ASKN(N + 2); COPY(BV, W1);
            FOR I:= 0 STEP 1 UNTIL N DO B[W1 + 1 + I]:= NINST(B[BV + 1 + I])
      END
END;


PROCEDURE MOVEN(V, W, N); VALUE V, W, N; INTEGER V, W, N;
BEGIN INTEGER I, M;
      M:= N - 1;
      FOR I:= 0 STEP 1 UNTIL M DO MOVE(V + I, W + I)
END;


BOOLEAN PROCEDURE MVDN(Q); VALUE Q; INTEGER Q; COMMENT MODE VARIABLE DECLARATION;
BEGIN INTEGER V, POLD;
      BOOLEAN SUC;
      POLD:= P; SUC:= FALSE; IF INP(HEAP) THEN ; IF DCL(ATL, V, Q) THEN
      BEGIN SUC:= LIST(VDF(Q, V)); IF Q = G THEN RS(V) END;
      MVDN:= SUC; IF ¬SUC THEN P:= POLD
END;


BOOLEAN PROCEDURE NIHL(Q); VALUE Q; INTEGER Q; IF INP(NIL) THEN
BEGIN NIHL:= TRUE; IF Q = G THEN
      BEGIN Z:= ASK; A[Z]:= REF; B[Z]:= 0 END
END
ELSE NIHL:= FALSE;

INTEGER PROCEDURE NINST(V); VALUE V; INTEGER V; COMMENT NEW INSTANCE OF;

BEGIN INTEGER W;
      NINST:= W:= ASK; MOVE(V, W)
END;

INTEGER PROCEDURE NRT(V); VALUE V; INTEGER V; COMMENT NEW REFERENCE TO;
BEGIN INTEGER W;
      NRT:= W:= ASK; A[W]:= REF; B[W]:= V; D[V]:= D[V] + 1
END;


PROCEDURE OD(OP, M1, M2, MD, PRIO); INTEGER OP, M1, M2, MD, PRIO; COMMENT DYADIC OPERATOR;
BEGIN Z:= ASKN(5); A[Z]:= PROC; B[Z]:= Z1:= Z + 1; A[Z1]:= 0; B[Z1]:= - PRIO; C[Z1]:= - OP;
      B[Z1 + 1]:= MD; B[Z1 + 2]:= M1; B[Z1 + 3]:= M2; IF OP ≠ OOLD THEN
      BEGIN L:= L + 1; OOLD:= OP END;
```

```
      POS(L, Z, G)'
END;


PROCEDURE OM(OP, M1, MD); INTEGER OP, M1, MD; COMMENT MONADIC OPERATOR;
BEGIN Z:= ASKN(4); A[Z]:= PROC; B[Z]:= Z1:= Z + 1; A[Z1]:= 0; B[Z1]:= - 10; C[Z1]:= - OP;
      B[Z1 + 1]:= MD; B[Z1 + 2]:= M1; IF OP ≠ OOLD THEN
      BEGIN L:= L + 1; OOLD:= OP END;
      POS(L, Z, G)
END;


BOOLEAN PROCEDURE OPDN(Q); VALUE Q; INTEGER Q; COMMENT OPERATION DECL. ;
IF INP(OP) THEN
BEGIN INTEGER L, U;
      OPDN:= TRUE; M(IF IN(IND, L) THEN INP(EQ) ELSE FALSE, ‡ AFTER OPSYMBOL ‡); M(RTXT(Q),
      ‡ ROUTINETEXT MISSING IN OP. DCL ‡); IF Q = G THEN
      BEGIN POS(L, Z, G); U:= B[Z]; B[U]:= IF B[U] = 1 THEN - 10 ELSE - 1 END;
 END
ELSE OPDN:= FALSE;

BOOLEAN PROCEDURE PACK(X); BOOLEAN X; PACK:= ANDP(INP(OPEN), ANDP(X, INP(CL)));


BOOLEAN PROCEDURE PIDN(Q); VALUE Q; INTEGER Q; COMMENT PROC IDENTITY DECL. ;

BEGIN INTEGER L;
      IF ANDP(INP(PROC), ANDP(IN(TAG, L), INP(EQ))) THEN
      BEGIN PIDN:= TRUE; M(RTXT(Q), ‡ NO ROUTINTEXT AFTER= ‡); POS(L + 10000, Z, Q) END
      ELSE PIDN:= FALSE
END;

BOOLEAN PROCEDURE PLAN(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q;
BEGIN INTEGER POLD, N, I, T1, L, W;
      POLD:= P; IF Q = G THEN
      BEGIN IF PLAN(VIC, T, F) THEN
            BEGIN PLAN:= TRUE; P:= POLD; T:= ASK;
                  N:= IF INP(OPEN) THEN (IF VIC = FML THEN DTC ELSE LSTC(DCL(VRL, T, F))) ELSE 0;
                  A[T]:= PROC; B[T]:= W:= ASKN(N + 2); A[W]:= 0; B[W]:= N; C[W]:= 0;
                  FOR I:= 1 STEP 1 UNTIL N DO
                  BEGIN M(DCL(VIC, T1, Q) ∨ VIC = FML ∧ I > 1, ‡DCL MISSING IN PLAN‡);
                     IF VIC = FML THEN M(IN(TAG, L), ‡ MISSING IDENTIFIER IN PLAN ‡);
                     B[W + I + 1]:= T1; IF I < N THEN M(INP(COM),
                     ‡ MISSING, IN PLAN ‡) ELSE M(INP(CL), ‡ MISSING) IN PLAN ‡)
                  END;
                  M(DCL(VRL, T1, Q), ‡ DCL MISSING AFTER PLAN ‡); B[W + 1]:= T1
            END
            ELSE PLAN:= FALSE
      END
      ELSE
      BEGIN IF INP(OPEN) THEN P:= MATCH; PLAN:= DCL(VRL, T, F) END
END;


PROCEDURE POS(L, V, Q); VALUE L, V, Q; INTEGER L, V, Q; COMMENT POSSESS;
IF Q = G THEN
BEGIN JGLB:= J:= J + 1; LIND[J]:= L; VIND[J]:= V END;


BOOLEAN PROCEDURE PRDR(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q; COMMENT PROCEDURE DECLARATOR;
```

```
PROR:= ANDP(INP(PROC), PLAN(VRL, T, Q));


BOOLEAN PROCEDURE PRIM(Q); INTEGER Q; COMMENT PRIMARY;
BEGIN BOOLEAN PR;
      INTEGER S;
      PRIM:= PR:= IF DEN(Q) THEN TRUE ELSE IF IDF(Q) THEN TRUE ELSE IF HIP(Q) THEN TRUE ELSE IF
      CAST(Q) THEN TRUE ELSE ENCL(Q);
      FOR I:= 1 WHILE IF PR THEN INP(OPEN) ELSE FALSE DO
      BEGIN IF Q = G THEN
            BEGIN FOR I:= 1 WHILE A[Z] = REF ∧ A[B[Z]] ≠ ROW DO DREF; S:= A[Z];
                  IF S = PROC THEN CLTL(Q) ELSE SLTL(Q)
            END
            ELSE P:= MATCH
      END;
END;


PROCEDURE PTOP(N, OP, PR); INTEGER N, OP, PR; COMMENT MOVES N OPERATORS FROM ARR TO VALUE TABLE;
BEGIN INTEGER JJ;
      FOR JJ:= 1 STEP 1 UNTIL N DO
      BEGIN I:= (JJ - 1) * 3; OD(OP, ARR[I + 1], ARR[I + 2], ARR[I + 3], PR) END;
      I:= 0
END;


PROCEDURE PUTBITS(X); VALUE X; INTEGER X;
BEGIN Z:= ASK; A[Z]:= BITS; B[Z]:= X END;


PROCEDURE PUTBOOL(X); VALUE X; BOOLEAN X;
BEGIN Z:= ASK; A[Z]:= BOOL; B[Z]:= IF X THEN 1 ELSE 0 END;


PROCEDURE PUTCHAR(X); VALUE X; INTEGER X;
BEGIN Z:= ASK; A[Z]:= CHAR; B[Z]:= X END;


PROCEDURE PUTCPL(RE, IM); VALUE RE, IM; REAL RE, IM;
BEGIN INTEGER V, VRE, VIM, V1;
      Z:= ASK; A[Z]:= STR; B[Z]:= V:= ASKN(2); C[Z]:= 2; V1:= V + 1; B[V]:= VRE:= ASK; C[V]:= LRE;
      B[V1]:= VIM:= ASK; C[V1]:= LIM; A[VRE]:= A[VIM]:= REAL; E[VRE]:= RE; E[VIM]:= IM
END;


PROCEDURE PUTINT(I); VALUE I; INTEGER I;
BEGIN Z:= ASK; A[Z]:= INT; B[Z]:= I END;


PROCEDURE PUTREAL(R); VALUE R; REAL R;
BEGIN Z:= ASK; A[Z]:= REAL; E[Z]:= R END;


PROCEDURE PUTSTRING(L); VALUE L; INTEGER L;
BEGIN INTEGER V, W, I, N, K;
      N:= MISC[L]; K:= IF N > 1 THEN N ELSE 1; Z:= ASKN(3 + K); V:= Z + 1; W:= Z + 2; A[Z]:= ROW;
      B[Z]:= V; C[Z]:= 1; A[V]:= 0; B[V]:= W + 1; C[V]:= 1; E[V]:= K; A[W]:= 1; B[W]:= N; C[W]:= 1;
      COPGEN(I, 1, K, CHAR, A[W + I]); COPGEN(I, 1, N, MISC[L + I], B[W + I])
END;
```

```
BOOLEAN PROCEDURE PVDN(Q); VALUE Q; INTEGER Q; COMMENT PROC VARIABLE DECL, ;
BEGIN INTEGER L, V;
      IF ANDP(IF INP(HEAP) = TRUE THEN INP(PROC) ELSE FALSE, ANDP(IN(TAG, L), INP(DEC))) THEN
      BEGIN PVDN:= TRUE; M(RTXT(Q), ¢ NO ROUTINETEXT ¢); IF Q = G THEN
            BEGIN V:= ASK; POS(L, V, G); A[V]:= REF; B[V]:= Z END
      END
      ELSE PVDN:= FALSE
END;


BOOLEAN PROCEDURE PYDF(Q); VALUE Q; INTEGER Q; COMMENT PRIORITY DEF, ;
BEGIN INTEGER L, L1, J, V;
      IF IN(IND, L) THEN
      BEGIN PYDF:= TRUE; M(INP(EQ), ¢= MISSING IN PRIODEF ¢); M(IN(INT, L1),
            ¢ PRIONUMBER MISSING ¢); IF Q = G THEN
            BEGIN M(SRCH(L, J), ¢ PRIODEF FOR UNKNOWN OP,¢);
                  EQR V:= B[VIND[J]], A[V] WHILE V > 0 DO IF B[V] ¢ = 10 THEN B[V]:= = MISC[L1 + 1]
            END
      END
      ELSE PYDF:= FALSE
END;


BOOLEAN PROCEDURE PYDN(Q); VALUE Q; INTEGER Q; COMMENT PRIORITY DECL, ;
IF INP(PRIO) THEN
BEGIN PYDN:= TRUE; M(LIST(PYDF(Q)), ¢ PRIORITY DEF MISSING ¢) END
ELSE PYDN:= FALSE;

BOOLEAN PROCEDURE RFDR(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q;
COMMENT REFERENCE TO MODE DECLARATOR;

IF INP(REF) THEN
BEGIN INTEGER V;
      RFDR:= TRUE; M(DCL(IF VIC = FML ^ X[P] ¢ REF THEN FML ELSE VRL, V, Q), ¢AFTER REF¢)
      IF Q = G THEN
      BEGIN T:= NRT(V); RX(V) END
END
ELSE RFDR:= FALSE;

BOOLEAN PROCEDURE RL2; COMMENT WIDENING TO REAL FOR INT/REAL OPERATIONS;
BEGIN BOOLEAN BRL2;
      BRL2:= TRUE; IF M1 = INT THEN EV1:= BV1 ELSE BRL2:= M1 = REAL;
      IF M2 = INT THEN EV2:= BV2 ELSE BRL2:= BRL2 ^ M2 = REAL; RL2:= BRL2
END;


PROCEDURE RMCL(APST); VALUE APST; INTEGER APST; COMMENT REMOVE COLLAT,;
IF ¬MEQ(Z, APST) THEN
BEGIN INTEGER NSUB, BZ, FEZ, NELZ, NCL, VNEW, VNEW1, NELN, FEN, VNN, VCLI, VII, V1, BPST, FEP, VI,
      VD, VD1, Z0, N, I, BFE1;
      M(A[Z] = A[APST], ¢RMCL¢); IF A[Z] = ROW THEN
      BEGIN NSUB:= C[Z]; BZ:= B[Z]; FEZ:= B[BZ]; NELZ:= E[BZ]; M(A[FEZ] = COLL, ¢NO COLL, IN ROW¢);
            NCL:= C[FEZ]; IF NSUB < C[APST] THEN
            BEGIN VNEW:= ASKN(NSUB + 3); A[VNEW]:= ROW; B[VNEW]:= VNEW1:= VNEW + 1;
                  C[VNEW]:= NSUB + 1; A[VNEW1]:= 1; NELN:= NELZ * NCL; E[VNEW1]:= NELN;
                  B[VNEW1]:= FEN:= ASKN(NELN); C[VNEW1]:= 1;
                  FOR I:= 1 STEP 1 UNTIL NSUB DO
                  BEGIN A[VNEW1 + I]:= A[BZ + I]; B[VNEW1 + I]:= B[BZ + I];
                        C[VNEW1 + I]:= C[BZ + I] * NCL
```

```
                ENDJ
                VNN:= VNEW1 + 1 + NSUB; A[VNN]:= 1; B[VNN]:= NCL; C[VNN]:= 1;
                FOR I:= 1 STEP 1 UNTIL NELZ DO
                BEGIN VCL:= FEZ - 1 + I; M(A[VCL] = COLL ∧ C[VCL] = NCL, ‡N ELTS IN RMCL‡);
                   V:= B[VCL]; COPYN(V, FEN + (I - 1) * NCL, NCL); FREEN(V, NCL)
                ENDJ
                Z:= VNEW; RMCL(APST)
          END
          ELSE
          BEGIN M(NSUB = C[APST], ‡RMCL NSUB‡); BPST:= B[APST]; FEP:= B[BPST]; M(A[FEP] = STR,
                ‡NO STRUCT IN RMCL‡); M(C[FEP] = NCL, ‡N STRUCT ELTS IN RMCL‡); BFE1:= B[FEP] - 1;
                FOR I:= 1 STEP 1 UNTIL NELZ DO
                BEGIN V:= FEZ - 1 + I; M(A[V] = COLL, ‡ELT OF ROW NOT COLL‡); A[V]:= STR;
                   V1:= B[V] - 1; B[V]:= VD:= ASKN(NCL); VD1:= VD - 1; Z0:= Z;
                   FOR L:= 1 STEP 1 UNTIL NCL DO
                   BEGIN Z:= V1 + L; IF A[Z] = COLL THEN DCOL(B[BFE1 + L]); B[VD1 + L]:= Z;
                      C[VD1 + L]:= C[BFE1 + L]
                   ENDJ
                   Z:= Z0
                END
          END
     END
     ELSE
     BEGIN M(A[Z] = STR, ‡NO ROW OR STR IN RMCL‡); BZ:= B[Z]; BPST:= B[APST]; Z0:= Z; N:= C[Z];
          M(N = C[APST], ‡N ELTS OF STRUC IN RMCL‡);
          FOR I:= 1 STEP 1 UNTIL N DO
          BEGIN Z:= B[BZ - 1 + I]; DCOL(B[BPST - 1 + I]); B[BZ - 1 + I]:= Z ENDJ
          Z:= Z0
     END
ENDJ


PROCEDURE RNGE(I, S)J VALUE IJ INTEGER IJ STRING SJ
BEGIN NLCR; NLCR; ABSFIXT(6, 0, I); PRINTTEXT(‡=TH INDEXES TOO‡); PRINTTEXT(S); ER(‡ ‡) ENDJ


BOOLEAN PROCEDURE ROP(S, Y)J VALUE SJ INTEGER SJ REAL YJ IF PP = S THEN
BEGIN ROP:= TRUE; PUTREAL(Y) END
ELSE ROP:= FALSEJ

BOOLEAN PROCEDURE ROP4(S1, R1, S2, R2, S3, R3, S4, R4)J INTEGER S1, S2, S3, S4J REAL R1, R2, R3, R4J
BEGIN ROP4:= TRUEJ
     IF PP = S1 THEN PUTREAL(R1) ELSE IF PP = S2 THEN PUTREAL(R2) ELSE IF PP = S3 THEN PUTREAL(R3)
     ELSE IF PP = S4 THEN PUTREAL(R4) ELSE ROP4:= FALSE
ENDJ


PROCEDURE RS(V)J VALUE VJ INTEGER VJ IF V > 0 THEN
BEGIN DM(V)J FREE(V) ENDJ


BOOLEAN PROCEDURE RTXT(Q)J VALUE QJ INTEGER QJ COMMENT ROUTINE TEXTJ

IF LK2(PLAN(FML, DUM, F), INP(COL)) THEN
BEGIN INTEGER P1, VJ
     RTXT:= TRUEJ P1:= PJ M(IF IF PLAN(FML, V, Q) THEN INP(COL) ELSE FALSE THEN UNIT(F) ELSE FALSE,
     ‡ ROUTINETEXT ‡)J IF Q = G THEN
     BEGIN Z:= VJ C[B[V]]:= P1J MORF:= TRUE END
END
ELSE RTXT:= FALSEJ
```

```
BOOLEAN PROCEDURE RWDR(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q; COMMENT ROW DECLARATOR;
BEGIN INTEGER POLD, V, ST, N, VD, I, NELT, VM, V1, J;

      BOOLEAN PROCEDURE RWRW(VIC, Q); VALUE VIC, Q; INTEGER VIC, Q;
      BEGIN BOOLEAN WORK;
            INTEGER J, LB, UB;
            WORK:= Q = G; IF WORK THEN J:= VD + 1; RWRW:= FALSE; IF VIC = ATL THEN
            BEGIN IF UNIT(Q) THEN
                  BEGIN RWRW:= TRUE; M(COER(MEEK, STIN, Q), ¢ROWROWER MODE¢); IF INP(COL) THEN
                     BEGIN IF WORK THEN
                        BEGIN LB:= A[J]:= B[I]; EMP END;
                        M(UNIT(Q), ¢MISSING U,B,¢); M(COER(MEEK, STIN, Q), ¢U,B, MODE IN DCL¢);
                        IF WORK THEN
                        BEGIN UB:= B[J]:= B[I]; EMP END.
                     END
                     ELSE IF WORK THEN
                     BEGIN LB:= A[J]:= 1; UB:= B[J]:= B[I]; EMP END;
                     IF WORK THEN NELT:= NELT * (IF UB < LB THEN 0 ELSE UB - LB + 1)
                  END
            END
            ELSE
            BEGIN IF WORK THEN A[J]:= B[J]:= 1; RWRW:= TRUE; IF UNIT(Q) ^ WORK THEN A[J]:= MEEKINT(Q);
                  IF INP(COL) THEN ; IF UNIT(Q) ^ WORK THEN B[J]:= MEEKINT(Q);
                  IF WORK THEN NELT:= NELT * (IF B[J] < A[J] THEN 0 ELSE B[J] - A[J] + 1)
            END
      END;

      IF Q = G THEN
      BEGIN POLD:= P; IF RWDR(VIC, T, F) THEN
            BEGIN RWDR:= TRUE; P:= POLD; T:= V:= ASK; A[V]:= ROW; ST:= IF INP(FLEX) THEN 0 ELSE 1;
                  IF INP(EIT) THEN ; M(INP(OPEN), ¢MISSING[IN DCL¢); C[V]:= N:= LSTC(RWRW(VIC, F));
                  B[V]:= VD:= ASKN(N + 1); A[VD]:= ST; C[VD]:= 1; NELT:= 1;
                  FOR I:= 1 STEP 1 UNTIL N DO
                  BEGIN RWRW(VIC, G); IF I = N THEN M(INP(CL), ¢MISSING] IN DCL¢) ELSE M(INP(COM),
                     ¢MISSING, IN ROWDCL¢)
                  END;
                  M(DCL(IF X[P] = REF THEN VRL ELSE VIC, VM, Q), ¢ROW DCL¢);
                  IF VIC = VRL THEN NELT:= 1; E[VD]:= NELT; IF NELT ≤ 1 THEN B[VD]:= VM ELSE
                  BEGIN B[VD]:= V1:= ASKN(NELT);
                     FOR I:= 1 STEP 1 UNTIL NELT DO MOVE(VM, V1 + I - 1); RS(VM)
                  END;
                  IF N > 0 THEN
                  BEGIN C[VD + N]:= 1;
                     FOR I:= N STEP - 1 UNTIL 2 DO
                     BEGIN J:= VD + I; C[J - 1]:= (B[J] - A[J] + 1) * C[J] END
                  END.
            END
            ELSE RWDR:= FALSE
      END
      ELSE RWDR:= ANDP(INP(FLEX) ∨ TRUE, ANDP(PACK(LIST(RWRW(VIC, F)) ∨ TRUE), DCL(VIC, T, F)))
END;


PROCEDURE RX(V); VALUE V; INTEGER V; IF V > 0 THEN
BEGIN D[V]:= D[V] - 1; IF D[V] ≤ 0 THEN
      BEGIN DM(V); FREE(V) END
END;
```

```
PROCEDURE RXN(V, N); VALUE V, N; INTEGER V, N;
BEGIN INTEGER W;
      W:= V + N - 1;
      FOR V:= V STEP 1 UNTIL W DO RX(V)
END;


BOOLEAN PROCEDURE SCL(Q); INTEGER Q; COMMENT SERIAL CLAUSE;
BEGIN INTEGER P1, P2, Q1, I;
      BOOLEAN LAB1, QF;

      BOOLEAN PROCEDURE JMP; IF Q = G ∧ Q1 = F THEN
      BEGIN JMP:= TRUE; IF PLAB > P1 ∧ PLAB < P2 THEN
            BEGIN P1:= PLAB; Q1:= G END
            ELSE
            BEGIN Q:= F; P1:= P2; QF:= TRUE END
      END
      ELSE JMP:= FALSE;

      BOOLEAN PROCEDURE SEP; IF INP(GOON) THEN
      BEGIN SEP:= TRUE; IF JMP THEN  ELSE IF Q = G THEN
            BEGIN IF ¬MORF THEN  ELSE
                  FOR I:= 1 WHILE DPRO(Q) ∨ DREF DO ; EMP
            END
      END ;
      ELSE IF INP(PNT) THEN
      BEGIN SEP:= TRUE; IF JMP THEN  ELSE IF Q = G THEN P1:= P2 END
      ELSE SEP:= JMP ∧ FALSE; P1:= P; SCL:= FALSE; QF:= FALSE;
      IF (IF Q = G THEN SCL(M) ELSE TRUE) THEN
      BEGIN P2:= P; P:= P1; LAB1:= FALSE;
            FOR I:= 0, 1 WHILE IF SEP THEN (IF Q = G ∨ QF THEN P < P2 ELSE TRUE) ELSE FALSE DO
            BEGIN Q1:= Q; IF LAB(Q1) THEN LAB1:= TRUE;
                  IF (IF LAB1 THEN FALSE ELSE LIST(UNDN(Q1))) THEN  ELSE
                  BEGIN IF I = 0 THEN SCL:= LBUN(Q1) ELSE M(LBUN(Q1), ‡ AFTER ; OR ,‡); SCL:= TRUE
                  END
            END
      END
END;

BOOLEAN PROCEDURE SCRL; COMMENT STRING OR CHAR PREP, FOR LSTR,;
IF STCH(V1, V2) THEN
BEGIN SCRL:= TRUE; MINL:= IF LEN1 < LEN2 THEN LEN1 ELSE LEN2; I:= 0;
      FOR I:= I + 1 WHILE IF I ≤ MINL THEN B[FE1 + 1 + I] = B[FE2 + 1 + I] ELSE FALSE DO ;
      UNEQ:= I ≤ MINL
END
ELSE SCRL:= FALSE;

BOOLEAN PROCEDURE SEC(Q); INTEGER Q; SEC:= IF GEN(Q) THEN TRUE ELSE IF SEL(Q) THEN TRUE ELSE PRIM(Q);

COMMENT SECONDARY;


BOOLEAN PROCEDURE SEL(Q); INTEGER Q; COMMENT SELECTION;
BEGIN INTEGER L, I, W, N, U;
      BOOLEAN NAME;
      IF ANDP(IN(TAG, L), INP(OF)) THEN
      BEGIN SEL:= TRUE; M(SEC(Q), ‡AFTER OF‡); IF WEAKC(Q) THEN
            BEGIN NAME:= A[Z] = REF; IF NAME THEN Z:= B[Z]; MORF:= TRUE; M(A[Z] = STR, ‡SEL‡);
                  W:= B[Z]; N:= C[Z]; I:= 1;
                  FOR I:= I WHILE IF I > N THEN FALSE ELSE C[W + I - 1] ≠ L DO I:= I + 1; M(I ≤ N,
```

```
                $SELECTOR$)) U:= B[W + I - 1]; Z:= IF NAME THEN NRT(U) ELSE NINST(U)
            END
        END
        ELSE SEL:= FALSE
END;


BOOLEAN PROCEDURE SEQ(A); BOOLEAN A; COMMENT SEQUENCE;
IF A THEN
BEGIN SEQ:= TRUE;
      FOR I:= 1 WHILE A DO
END
ELSE SEQ:= FALSE;

BOOLEAN PROCEDURE SKIP(Q); INTEGER Q; IF INP(SKP) THEN
BEGIN SKIP:= TRUE; IF Q = G THEN
      BEGIN Z:= ASK; A[Z]:= VOID END
END
ELSE SKIP:= FALSE;

PROCEDURE SLTL(Q); INTEGER Q; COMMENT SLICE TAIL;

BEGIN INTEGER V0, VD, N, FE, K, OFST, I, N1, J, NELT, LTR, UTR, W, VNW, FENW, VREF, VD1, TS1, L1;
      BOOLEAN NAME, TS1R;
      MORF:= TRUE; NAME:= A[Z] = REF; IF NAME THEN VREF:= Z; V0:= IF NAME THEN B[VREF] ELSE Z;
      M(A[V0] = ROW, $ SLICE $); VD:= B[V0]; N1:= N:= C[V0]; FE:= B[VD]; OFST:= C[VD];
      VD1:= ASKN(N + 1); COPYN(VD, VD1, N + 1); A[VD1]:= 1; NELT:= E[VD];
      FOR I:= 1 STEP 1 UNTIL N DO
      BEGIN TS1R:= UNIT(Q); J:= VD1 + I; IF TS1R THEN TS1:= MEEKINT(Q);
            IF SPY(COL) ∨ SPY(AT) ∨ ¬TS1R THEN
            BEGIN COMMENT TRIMMER;
                  LTR:= IF TS1R THEN TS1 ELSE A[J]; UTR:= B[J]; IF INP(COL) THEN
                  BEGIN IF UNIT(Q) THEN UTR:= MEEKINT(Q) END;
                  IF INP(AT) THEN
                  BEGIN M(UNIT(Q), $ AFTER AT $); L1:= MEEKINT(Q) END
                  ELSE L1:= 1; IF LTR < A[J] THEN RNGE(I, $ LOW $); IF UTR > B[J] THEN RNGE(I,
                  $ HIGH $); OFST:= OFST + (LTR - A[J]) * C[J]; A[J]:= L1; B[J]:= L1 - LTR + UTR
            END
            ELSE
            BEGIN COMMENT SUBSCRIPT;
                  K:= TS1; OFST:= OFST + (K - A[J]) * C[J]; B[J]:= - 1; N1:= N1 - 1
            END;
            IF I < N THEN M(INP(COM), $ TOO FEW TRIMSCRIPT $) ELSE M(INP(CL),
            $ TOO MANY TRIMSCRIPTS $)
      END;
      C[VD1]:= OFST; IF N1 > 0 THEN
      BEGIN IF N1 < N THEN
            BEGIN W:= ASKN(N1 + 1);
                  FOR I:= 1 STEP 1 UNTIL N DO IF B[I] ≠ - 1 THEN
                  BEGIN COPY(VD1 + I, W + I); B[W]:= W + 1 END;
                  FREEN(VD1, N + 1)
            END
            ELSE W:= VD1; VNW:= ASK; A[VNW]:= ROW; B[VNW]:= W; C[VNW]:= N1
      END
      ELSE
      BEGIN VNW:= FE + OFST - 1; FREEN(VD1, N + 1) END;
      IF NAME THEN Z:= NRT(VNW) ELSE IF N1 > 0 THEN
      BEGIN B[W]:= FENW:= ASKN(NELT); MOVEN(FE, FENW, NELT); Z:= VNW END
      ELSE
      BEGIN Z:= ASK; MOVE(VNW, Z) END
```

```
END;


BOOLEAN PROCEDURE SOFTC(Q); INTEGER Q; COMMENT SOFT COERCION;
BEGIN FOR I:= 1 WHILE IF Q = G THEN DPRO(Q) ELSE FALSE DO ; SOFTC:= Q = G END;


PROCEDURE SPS(V, W); VALUE V, W; INTEGER V, W; COMMENT SUPERSEDE;
BEGIN DM(W); MOVE(V, W) END;


BOOLEAN PROCEDURE SPY(S); VALUE S; INTEGER S; IF INP(S) THEN
BEGIN SPY:= TRUE; P:= P - 1 END
ELSE SPY:= FALSE;

BOOLEAN PROCEDURE SRCH(L, K); VALUE L; INTEGER L, K; COMMENT SEARCH IN LINK TABLE;

BEGIN INTEGER JJ;
      JJ:= J;
      FOR I:= 1 WHILE IF JJ > 0 THEN L # LIND(JJ) ELSE FALSE DO JJ:= JJ - 1; K:= JJ; SRCH:= JJ > 0
END;

BOOLEAN PROCEDURE STCH(U, V); VALUE U, V; INTEGER U, V;
COMMENT OBTAINS FIRST ELEMENTS AND LENGTHS OF TWO STRINGS IF POSSIBLE;
STCH:= IF STRC(U, FE1, LEN1) THEN STRC(V, FE2, LEN2) ELSE FALSE;


BOOLEAN PROCEDURE STDR(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q; COMMENT STRUCT DECLARATOR;
IF INP(STR) THEN
BEGIN INTEGER V, VD, N, V1, I, VM, L;
      BOOLEAN F1;
      STDR:= TRUE; M(INP(OPEN), $NO(AFTER STRUCT$)); IF Q = G THEN
      BEGIN V:= ASK; A[V]:= STR; C[V]:= N:= DTC; B[V]:= VD:= ASKN(N); F1:= TRUE;
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN IF F1 THEN F1:= FALSE ELSE M(INP(COM), $MISSING, IN STRUCTDCL$);
                  M(DCL(IF X[P] = REF THEN VRL ELSE VIC, VM, Q), $MISSING DCL$); B[VD + I - 1]:= VM;
                  M(IN(TAG, L), $MISSING TAG$); C[VD + I - 1]:= L
            END;
            T:= V; M(INP(CL), $MISSING) IN STRUCTDCL$)
      END
      ELSE P:= MATCH
END
ELSE STDR:= FALSE;

PROCEDURE STFN(F, N); VALUE F, N; INTEGER F, N; COMMENT STANDARD FUNCTION;

BEGIN OM( - F, STRL, STRL); B[Z1]:= 1; L:= L + N END;

BOOLEAN PROCEDURE STRC(V, FE, LEN); VALUE V; INTEGER V, FE, LEN;
BEGIN INTEGER VD;
      COMMENT LOOK IF V IS STRING OR CHAR;
      STRC:= FALSE; IF A[V] = ROW THEN
      BEGIN IF C[V] = 1 THEN
            BEGIN VD:= B[V]; FE:= B[VD]; IF A[FE] = CHAR THEN
                  BEGIN STRC:= TRUE; LEN:= B[VD + 1] END
            END
      END
      ELSE IF A[V] = CHAR THEN
      BEGIN STRC:= TRUE; FE:= V; LEN:= 1 END
END;
```

```
PROCEDURE SUMS; COMMENT FORMS SUM OF TWO STRINGS;
BEGIN INTEGER N, Z1, Z2, Z3;
      N:= LEN1 + LEN2; Z:= ASKN(N + 3); A[Z]:= ROW; B[Z]:= Z1:= Z + 1; C[Z]:= 1; A[Z1]:= 0;
      B[Z1]:= Z3:= Z + 3; C[Z1]:= 1; E[Z1]:= N; Z2:= Z + 2; A[Z2]:= 1; B[Z2]:= N; C[Z2]:= 1;
      COPYN(FE1, Z3, LEN1); COPYN(FE2, Z3 + LEN1, LEN2)
END;


BOOLEAN PROCEDURE TERT(Q); INTEGER Q; TERT:= IF FORM(Q) THEN TRUE ELSE SEC(Q);

COMMENT TERTIARY;

BOOLEAN PROCEDURE UNDN(Q); VALUE Q; INTEGER Q; COMMENT UNITARY DECLARATION;
BEGIN BOOLEAN SUC;
      INTEGER P0;
      P0:= P;
      UNDN:= SUC:= IF MDN(Q) THEN TRUE ELSE IF PYDN(Q) THEN TRUE ELSE IF OPDN(Q) THEN TRUE ELSE
      IRDN(Q); IF SUC THEN
      BEGIN IF X[P] ≠ GOON ∧ X[P] ≠ COM THEN
            BEGIN UNDN:= SUC:= FALSE; P:= P0 END
      END;
      IF SUC ∧ Q = G THEN
      BEGIN D[1]:= D[1] + 1; Z:= 1 END
END;


BOOLEAN PROCEDURE UNDR(VIC, T, Q); VALUE VIC, Q; INTEGER VIC, T, Q; COMMENT UNION DECLARATOR;
IF INP(UN) THEN
BEGIN INTEGER N, W, U, I;
      UNDR:= TRUE; M(INP(OPEN), ‡MISSING IN UNIONDCL ‡); IF Q = G THEN
      BEGIN N:= LSTC(DCL(VRL, T, F)); T:= ASK; A[T]:= UN; W:= B[T]:= ASKN(N); C[T]:= 0; C[W]:= N;
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN M(DCL(VRL, U, Q), ‡DCL MISSING IN UNIONDCL ‡); B[W + I - 1]:= U;
                  IF I < N THEN M(INP(COM), ‡, MISSING IN UNIONDCL ‡) ELSE M(INP(CL),
                  ‡) MISSING IN UNIONDCL ‡)
            END;
            B[T]:= DEV(W)
      END
      ELSE P:= MATCH
END
ELSE UNDR:= FALSE;

BOOLEAN PROCEDURE UNIT(Q); INTEGER Q; IF LOOP(Q) THEN UNIT:= TRUE ELSE
BEGIN INTEGER POLD, L, NEXT;
      BOOLEAN POS;
      POLD:= P; IF INP(OPEN) THEN P:= MATCH; NEXT:= X[P];
      POS:= IF NEXT > 91 ∧ NEXT < 119 THEN TRUE ELSE IN(IND, L); P:= POLD;
      IF IF POS THEN RTXT(Q) ELSE FALSE THEN UNIT:= TRUE ELSE IF TERT(Q) THEN
      BEGIN UNIT:= TRUE;
            IF INP(SEC) THEN ASTL(Q) ELSE IF INP(IS) THEN IRTL(Q, IS) ELSE IF INP(ISNT) THEN
            IRTL(Q, ISNT)
      END
      ELSE UNIT:= FALSE
END;


BOOLEAN PROCEDURE UNMB(V, W); VALUE V, W; INTEGER V, W; COMMENT UNION MEMBER;
```

```
BEGIN INTEGER N, I;
      N:= C[W]; I:= - 1;
      FOR I:= I + 1 WHILE IF I = N THEN FALSE ELSE ¬MEQ(B[V], B[W + I]) DO ; UNMB:= I < N
END;

BOOLEAN PROCEDURE VAC(Q); VALUE Q; INTEGER Q; IF ANDP(INP(OPEN), INP(CL)) THEN
BEGIN VAC:= TRUE; IF Q = G THEN
      BEGIN Z:= ASK; A[Z]:= VOID END
END
ELSE VAC:= FALSE;

BOOLEAN PROCEDURE VDF(Q, U); VALUE Q, U; INTEGER Q, U; COMMENT VARIABLE DEFINITION;

BEGIN INTEGER L, V, W;
      IF IN(TAG, L) THEN
      BEGIN VDF:= TRUE; IF Q = G THEN
            BEGIN V:= NINST(U); W:= NRT(V); IF INP(BEC) THEN
                  BEGIN M(UNIT(Q), ‡AFTER:=‡); M(COER(STRONG, U, Q), ‡SOURCE MODE IN VAR DEF‡);
                     SPS(Z, V); EMP
                  END;
                  RX(V); POS(L, W, Q)
            END
            ELSE IF IF INP(BEC) THEN UNIT(F) ELSE FALSE THEN
      END
      ELSE VDF:= FALSE
END;

BOOLEAN PROCEDURE VDN(Q); VALUE Q; INTEGER Q; VDN:= IF PVDN(Q) THEN TRUE ELSE MVDN(Q);

COMMENT VARIABLE DECLARATION;

PROCEDURE VISB(J1, J2); VALUE J1, J2; INTEGER J1, J2; COMMENT MAKE VISIBLE;
BEGIN INTEGER JJ;
      FOR JJ:= J1 + 1 STEP 1 UNTIL J2 DO IF LIND[JJ] ≥ 10000 THEN LIND[JJ]:= LIND[JJ] - 10000
END;


BOOLEAN PROCEDURE WEAKC(Q); INTEGER Q; COMMENT WEAK COERCION;
BEGIN FOR I:= I WHILE IF Q = G THEN (IF IF IF A[Z] = REF THEN A[B[Z]] = REF ELSE FALSE THEN DREF
      ELSE FALSE THEN TRUE ELSE DPRO(Q)) ELSE FALSE DO ; WEAKC:= Q = G
END;


PROCEDURE WTX(S); VALUE S; INTEGER S;
BEGIN PX:= PX + 1; X[PX]:= S END;

COMMENT WRITE TO X;
PART:= 1000000;
FOR I:= 2 STEP 1 UNTIL NA DO D[I - 1]:= I; D[NA]:= 0; FB:= 1;
COMMENT NOW FIRST FOLLOWS STANDARD PRELUDE INFORMATION;
LRE:= 91; LIM:= 94; MISC[1]:= MISC[2]:= 1; PUTBOOL(TRUE); STBL:= Z; PUTINT(1); STIN:= Z;
PUTREAL(4 * ARCTAN(1)); STRL:= Z; PUTCPL(1.0, 1.0); STCP:= Z; PUTCHAR(1); STCR:= Z; PUTSTRING(1);
STNG:= Z; RINT:= NRT(STIN); RRL:= NRT(STRL); RSTG:= NRT(STNG); STVD:= ASK; A[STVD]:= VOID;
STRW:= ASK; A[STRW]:= ROW; LI:= I:= OOLD:= 0; OD(PLTO, STCR, RSTG, RSTG, 1);
OD(PLTO, STNG, RSTG, RSTG, 1);
FOR MM:= RINT, STIN, RINT, RRL, STIN, RRL, RRL, STRL, RRL, RSTG, STCR, RSTG, RSTG, STNG, RSTG DO FILL;
PTOP(5, PSAB, 1); PTOP(3, MAB, 1); PTOP(3, TMAB, 1); OD(OVAB, RINT, STIN, RINT, 1);
OD(MDAB, RINT, STIN, RINT, 1); OD(DVAB, RRL, STIN, RRL, 1); OD(DVAB, RRL, STRL, RRL, 1);
OD(OR, STBL, STBL, STBL, 2); OD(AND, STBL, STBL, STBL, 3);
FOR MM:= STIN, STIN, STBL, STIN, STRL, STBL, STRL, STIN, STBL, STRL, STRL, STBL, STCR, STCR, STBL,
```

```
            STCR, STNG, STBL, STNG, STCR, STBL, STNG, STNG, STBL, STBL, STBL, STBL, STIN, STCP, STBL, STRL,
            STCP, STBL, STCP, STIN, STBL, STCP, STRL, STBL, STCP, STCP, STBL DO FILL; PTOP(14, EQ, 4);
            PTOP(14, NE, 4); PTOP(8, LESS, 5); PTOP(8, GT, 5); PTOP(8, LE, 5); PTOP(8, GE, 5);
            EQB MM:= STIN, STIN, STIN, STIN, STRL, STRL, STRL, STIN, STRL, STRL, STRL, STRL, STIN, STCP, STCP,
            STRL, STCP, STCP, STCP, STIN, STCP, STCP, STRL, STCP, STCR, STCR, STNG, STCR,
            STNG, STNG, STNG, STCR, STNG, STNG, STNG, STNG DO FILL; PTOP(13, PLUS, 6); OM(PLUS, STIN, STIN);
            OM(PLUS, STRL, STRL); PTOP(9, MIN, 6); OM(MIN, STIN, STIN); OM(MIN, STRL, STRL); PTOP(9, TIMS, 7);
            PTOP(9, DIV, 7); OD(OVER, STIN, STIN, STIN, 7); OD(MOD, STIN, STIN, STIN, 7);
            OD(PLIT, STIN, STIN, STCP, 9); OD(PLIT, STIN, STRL, STCP, 9); OD(PLIT, STRL, STIN, STCP, 9);
            OD(PLIT, STRL, STRL, STCP, 9); OD(UP, STIN, STIN, STIN, 8); OD(UP, STRL, STIN, STRL, 8);
            OD(UP, STCP, STIN, STCP, 8); OD(LWB, STIN, STRW, STIN, 8); OM(LWB, STRW, STIN);
            OD(UPB, STIN, STRW, STIN, 8); OM(UPB, STRW, STIN); OM(NOT, STBL, STBL); OM(ABSV, STBL, STIN);
            OM(ABSV, STIN, STIN); OM(ABSV, STRL, STRL); OM(ABSV, STCP, STRL); OM(ABSV, STCR, STIN);
            OM(ODD, STIN, STBL); OM(SGN, STIN, STIN); OM(SGN, STRL, STIN); OM(RND, STRL, STIN);
            OM(ENT, STRL, STIN); OM(RE, STCP, STRL); OM(IM, STCP, STRL); OM(CONJ, STCP, STRL);
            OM(ARG, STCP, STRL); OM(REPR, STIN, STCR); L:= L + 1; POS(L, STRL, G); L:= L + 2; STFN(NSQR, 4);
            STFN(NEXP, 3); STFN(NLN, 2); STFN(NCOS, 3); STFN(NACS, 6); STFN(NSIN, 3); STFN(NASN, 6);
            STFN(NTAN, 3); STFN(NATN, 6); OM(1000, STRL, STVD); L:= L + 11; OM(999, STIN, STVD); L:= L + 7;
            OM(998, STIN, STVD); L:= L + 7; OM(997, STIN, STVD); L:= L + 5; I:= READ; IF I = -1 THEN EXIT;
            M(I = 1, ‡INCORRECT INPUT‡); NEWPAGE;
            EQB I:= RESYM WHILE I ≠ 122 DO PRSYM(I); NLCR; NLCR; COMMENT 122= INTERROGATION MARK = END OF PROGRAM;
            PX:= Nx1:= READ;
            EQB I:= 1 STEP 1 UNTIL Nx1 DO X[I]:= READ; NM1:= READ;
            EQB L:= 1 STEP 1 UNTIL NM1 DO MISC[L]:= READ; NR1:= READ;
            EQB L:= 1 STEP 1 UNTIL NR1 DO RNR[L]:= READ; NLCR; NLCR; NLCR; Q:= G; SEQ(LAB(H)); M(ENCL(Q),
            ‡PROGRAM‡);
        END
    END
END
```

Appendix B

```
( 'COMMENT' RECURSIVE FACTORIAL OPERATOR 'COMMENT'
  'OP' 'FAC' = ('INT' N) 'INT' : ( N=0 | 1 | N* 'FAC'(N-1)))
  PRINT( 'FAC' 5 )
)

+120
```

```
'BEGIN' 'CO' COMPLEX SQUARE ROOT 'CO'
    'COMPL' Z;
    'PROC' COMPSQRT = ('COMPL' Z) 'COMPL';
    'BEGIN' 'REAL' X=:'RE' Z, Y=:'IM' Z;
        'REAL' RP=:SQRT(('ABS' X + SQRT(X**2+Y**2))/2);
        'REAL' IP = (RP=:0|0|Y/(2=:RP));
        (X>=:0|RP '|' IP |'ABS' IP '|' (Y>=:0|RP|-:RP))
    'END';
    Z:=:COMPSQRT(-1);
    PRINT(('RE' Z, "|" ,'IM' Z));
    Z:=:COMPSQRT(1 '|' -1);
    PRINT((NEWLINE, 'RE' Z, "|" , 'IM' Z))
'END'
```

```
          +0       |            +1
+.1098684113469=+  1 |=.4550898605617=-  0
```

```
'BEGIN' 'CO' TOWERS OF HANOI 'CO'
     'PROC' P = ('INT' A,B,K) 'VOID';
     'IF' K > 0 'THEN'
          P(A,6-A-B,K-1);
          PRINT((NEWLINE,"FROM",A,"TO",B,"PIECE",K));
          P(6-A-B,B,K-1)
     'FI';

     P(1,2,4)
'END'
```

```
FROM          +1      TO          +3      PIECE         +1
FROM          +1      TO          +2      PIECE         +2
FROM          +3      TO          +2      PIECE         +1
FROM          +1      TO          +3      PIECE         +3
FROM          +2      TO          +1      PIECE         +1
FROM          +2      TO          +3      PIECE         +2
FROM          +1      TO          +3      PIECE         +1
FROM          +1      TO          +2      PIECE         +4
FROM          +3      TO          +2      PIECE         +1
FROM          +3      TO          +1      PIECE         +2
FROM          +2      TO          +1      PIECE         +1
FROM          +3      TO          +2      PIECE         +3
FROM          +1      TO          +3      PIECE         +1
FROM          +1      TO          +2      PIECE         +2
FROM          +3      TO          +2      PIECE         +1
```

```
'BEGIN'  # TRANSLATION FROM DECIMAL TO ROMAN #
     'MODE' 'M'='STRUCT' ('INT' VALUE,'STRING' R);
     'PROC' ROMAN=('INT' NUMBER) 'STRING':
  *  'BEGIN' 'INT' N:=NUMBER,'STRING' RESULT,
     [] 'M' TABLE =
          ((1000,"M"),(900,"CM"),(500,"D"),(400,"CD"),
          (100,"C"),(90,"XC"),(50,"L"),(40,"XL"),
          (10,"X"),(9,"IX"),(5,"V"),(4,"IV"),(1,"I"));
          'FOR' I 'TO' 'UPB' TABLE 'DO'
          'BEGIN' 'INT' V = VALUE 'OF' TABLE[I],
               'STRING' R= R 'OF' TABLE[I];
               'WHILE' V <= N 'DO' (RESULT +:=R; N -:=V)
          'END';
          RESULT
     'END';
     PRINT(ROMAN(1968))
'END'
```

```
MCMLXVIII
```

```
'BEGIN' # TRANSLATION FROM ROMAN TO DECIMAL #
    'PROC' CHAR IN STRING = ('CHAR' C, 'REF' 'INT' I,'STRING' S) 'BOOL':
    ('FOR' K 'TO' 'UPB' S 'DO' (C=S[K]|:I:=K;L)) 'FALSE', L: 'TRUE'));
    'PROC' VALUE OF ROMAN = ('STRING' TEXT) 'INT':
    'IF' TEXT = "" 'THEN' 0 'ELSE'
        'OP' 'ABS' = ('CHAR' S) 'INT':
        'CASE' 'INT' P; CHAR IN STRING(S,P,"IVXLCDM"); P 'IN'
            1,5,10,50,100,500,1000
        'ESAC'};
        'INT' V, MAXV:=0, MAXP;
        'FOR' P 'TO' 'UPB' TEXT 'DO'
        'IF' (V:='ABS' TEXT[P]) > MAXV 'THEN' MAXP:=P; MAXV:=V 'FI'};
        MAXV + VALUE OF ROMAN (TEXT[: MAXP-1])
            + VALUE OF ROMAN (TEXT[MAXP+1 :])
    'FI';
    PRINT(VALUE OF ROMAN("MCMLXVIII"))
'END'
```

```
        +1968
```

```
'BEGIN' # IN SITU PERMUTATION #
    [1:5] 'REAL' X:=(3.14, 2.72, .577, -33.0, 1.41);
    'PROC' PERMVEC = ('REF' [1:] 'REAL' VEC, [1:] 'INT' P) 'VOID';
    'FOR' J 'TO' 'UPB' P 'DO'
    'BEGIN' 'INT' K:=P[J]; 'WHILE' K>J 'DO' K:=P[K];
        'IF' K=J 'THEN'
            'REAL' H=VEC[J], 'INT' L:=P[K];
            'WHILE' L /=J 'DO'
            'BEGIN' VEC[K]:=VEC[L]; K:=L; L:=P[K] 'END';
            VEC[K]:=H
        'FI'
    'END';
    PERMVEC(X, (3,5,4,1,2)); PRINT(X)
'END'
```

```
+.5770000000002e-  0 +.1410000000000e+  1          -33       +.3139999999999e+  1 +.2720000000001e+  1
```

```
'BEGIN' # NUMERICAL QUADRATURE #
    'PROC' INT= ('REAL' A,B,EPS,        'PROC' ('REAL') 'REAL' Y) 'REAL';
    'BEGIN' 'PROC' INT1= ('REAL' X0,X4,F0,F2,F4) 'REAL';
        'BEGIN' 'REAL' X2= (X0+X4)/2; 'REAL' F1=Y((X0+X2)/2),
            F3 = Y((X2+X4)/2);
            'REAL' T;
            'IF' T:= 6*F2-4*(F1+F3)+F0+F4; 'ABS' T < DELTAMAX 'THEN'
            (X4-X0)*(4*(F1+F3)+2*F2+F0+F4-T/15) 'ELSE'
            INT1(X0,X2,F0,F1,F2) + INT1(X2,X4,F2,F3,F4)
            'FI'
        'END' #INT1#,
        'REAL' DELTAMAX= 180 * EPS/'ABS' (B-A);
        INT1(A,B,Y(A),Y((A+B)/2),Y(B))/12
    'END';
    PRINT(INT(0,2,.01  ,('REAL'X) 'REAL':X**2-3*X)) #-3.3333...#
'END'
```

```
-.3333333333332=*  1
```