MATHEMATICAL CENTRE TRACTS 48

F.E.J. KRUSEMAN ARETZ
P.J.W. TEN HAGEN
H.L. OUDSHOORN

AN ALGOL 60 COMPILER IN ALGOL 60

TEXT OF THE MC-COMPILER FOR THE EL-X8

MATHEMATISCH CENTRUM AMSTERDAM 1973

ACM - Computing Reviews-category : 4.12

CONTENTS

PREFACE

The ALGOL 60 program, presented in this report, was written essentially in the years 1964-1965 at the Mathematical Centre in Amsterdam by F.E.J. Kruseman Aretz (now at Philips Research Laboratories, Eindhoven). It is able to translate programs, written in the source language ALGOL 60, into a target language, being bit-patterns in the memory of an EL-X8 computer. This ALGOL version of the ALGOL compiler has been used as a blueprint for a version in ELAN, the symbolic assembly language for the EL-X8 and has been in operation from november 1965.

This report is rather late, due to many other obligations on the part of one of the authors. However, although the EL-X8 will be obsolete soon, publication of this report still seems of some value, because most of it is rather machine-independent, and as such it may serve as an illustration of compiler techniques.

For the preparation of this report, P.J.W. ten Hagen updated the ALGOL 60 version of the compiler and brought it in close correspondence to the ELAN version. Later on some further improvements were made, especially in the algorithm generating the line number bookkeeping instructions. For the purpose of demonstration, a special output procedure has been added.

FKA

0. INTRODUCTION

0.0 HISTORICAL NOTES

The ALGOL 60 program presented in this report, constitutes a complete ALGOL 60 translator. It is able to read in a text, written in ALGOL 60 and punched in the so-called MC flexowriter code, and to produce from it a machine program for an EL-X8 computer.

The MC ALGOL 60 system for the EL-X8 has been in development for a couple of years, and it had to fulfil a number of (almost contradictory) requirements:

1) it should be adapted to the smallest X8 configuration possible (i.e. with a core memory of 16K words of 27 bits, no backing store),
2) it should have high efficiency at compile-time and produce reasonably efficient object programs,
3) it should have a modular structure, so that it could be easily adapted to future requirements,
4) it should check against program errors wherever possible,
5) the compiler should be of educational value, so that it could be used as an introduction to compiler-writing,

The development of the system has been the work of a team, in which, among others, cooperated: F.J.M. Barning (standard functions, EL-X8 simulator), J.A.Th.M. van Berckel (I/O routines, operating system "PICO"), F.E.J. Kruseman Aretz (run-time routines, compiler) and J.J.B.M. Nederkoorn (run-time routines, documentation). The key work, namely the design of the compiler output and the development of the run-time routines, was done by Nederkoorn and Kruseman Aretz in the years 1963-1964. It was partly described by Nederkoorn in a preliminary report [1], of which only a few copies were made and distributed. The compiler was written in ALGOL 60 first. So it could be tested out, in parts, using the ALGOL 60 implementations (I and II) for the EL-X1 computer at the MC. In this version full attention was paid to clarity. Afterwards the compiler was hand-coded into ELAN, the symbolic assembly language for the EL-X8, and debugged using the X8 simulator on the EL-X1 as developed by Barning.
Using the ALGOL text as a blueprint, the hand-coding took a few months

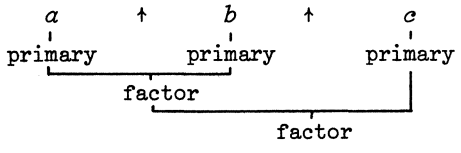
only, and relatively few coding errors occurred. The result was documented by Kruseman Aretz and Mailloux [2], together with the run-time routines and a simple operating system.

In the years from 1966 onwards two other documents were produced, one [3] defining the compiler output in great detail, the other [4] being written as lecture notes to a course given at the University of Amsterdam during the academic year 1969-1970. Both documents are in Dutch. Moreover, a very special aspect was described in [5].

It is hoped that the ALGOL text of the compiler is largely self-explanatory, even without any knowledge of either the structure of the EL-X8 instruction set or of the object program produced by the compiler. The following sections will therefore only sketch very briefly some general characteristics of the compiler and present some advice on how to study the program.

0.1 PARSING AND MACRO-GENERATION

The ALGOL 60-compiler given in this tract applies a top-to-bottom analysis which is closely related to the syntax of the language as presented in [6]. It is divided into a large number of often rather small procedures, each with a specific task. By way of example we discuss in the following a procedure that analyses and compiles a factor. According to [6] a factor either is a primary, or is constructed from (1) a factor, (2) the operator "↑" and (3) a primary. Each factor preceding a "↑" can be composite itself. Thus $a{\uparrow}b{\uparrow}c$ is a factor with the following structure:



The object code for a dyadic arithmetic operation will, apart from optimization, contain the following components:

1) code for the evaluation of the first operand, delivering the result in a register dedicated to arithmetic results,

2) code that puts the result on the top of the stack,
3) code for the evaluation of the second operand, delivering the result in the register mentioned above,
4) code for the operation, to be carried out on the number on the top of the stack and on the result in the register.

The result of the operation is delivered in the same register.

We can indicate the code for step 2 by a symbolic instruction STACK and the code for step 4, in the case of exponentiation, by TTP (short for "to the power"). Probably, such a symbolic instruction has to be replaced by a number of machine instructions in the actual object program. But the introduction of symbolic instructions (in the next we will often refer to them as *macros*) is extremely useful for the separation between the machine-independent part of the compiler (the syntax analyser and the macro generator) and its machine-dependent part (the macro processor). They can be seen as instructions for some hypothetical machine that can be implemented on various equipment.

If in the expression $a\uparrow b\uparrow c$ the identifiers a, b and c correspond to simple variables of type real and we introduce a macro TRV (short for "take real variable") for the action of loading the value of such a variable into the arithmetic register, the translation of the given expression in terms of macros reads:

TRV(a)	load the value of a into the register
STACK	save the register in the stack
TRV(b)	load the value of b into the register
TTP	compute (top of stack)$\uparrow$register, i.e. $a\uparrow b$
STACK	save the register (i.e. $a\uparrow b$) in the stack
TRV(c)	load the value of c into the register
TTP	compute (top of stack)$\uparrow$register, i.e. $(a\uparrow b)\uparrow c$

The identifiers a, b and c occur here as macro parameters, i.e. as parameters to the macro TRV. In the final machine instructions, to be produced out of this macro, they will be replaced by addresses, given to them by the compiler.

After these introductory remarks we present a procedure *Factor* as an example of a compiler procedure. This procedure has to analyse and compile a factor from an arithmetic expression. It assumes that the first basic symbol of that factor is available in the compiler variable *last symbol* (coded in some internal representation), that subsequent basic symbols can be read into that variable by means of calls of a procedure *next symbol*, and that it leaves in that variable the first basic symbol not belonging to the factor. For the analysis and compilation of primaries a procedure *Primary* is available with analogous specification. The text of *Factor* reads:

```
procedure Factor;
begin integer ls;
      Primary;
      for ls:= last symbol while ls = ttp do
      begin Macro(STACK);
            next symbol; Primary;
            Macro(TTP)
      end
end Factor;
```

In this procedure the identifier *ttp* denotes a variable initialised with the internal representation of the basic symbol "↑". In the translation for *a↑b↑c* given above the macro TRV is produced by (three successive calls of) *Primary*, the macros STACK and TTP by *Factor* itself.

We see that the procedure *Factor* serves two purposes: source text reading (and analysis) and macro generation. It uses a procedure *Primary* to deal with its substructures. In its turn, however, a factor is a building block for a term. This is reflected in the compiler procedure *Term* that calls *Factor* for the treatment of factors. Terms are components of simple arithmetic expressions and those in turn components of arithmetic expressions or of relations, and again these structures can be found back in the corresponding compiler procedures.

The procedure *Factor* given on page 46 differs from the one presented above in two respects:

1) it is divided into two separate procedures called *Factor* and *Next primary*,
2) iteration has been replaced by recursion.

The existence of a separate procedure *Next primary* is useful in those situations in which the first primary in an arithmetic expression has been read before it is decided to compile an arithmetic expression, either because this could not be decided in advance or for reasons of optimization. We come back to this point in the next section. The replacement of iteration by recursion has been carried out at many places throughout the compiler for theoretical reasons mainly.

0.2 LEFT-TO-RIGHT SCANS

In the previous section we stated that the ALGOL 60 compiler given in this tract consists of a large number of procedures. To most syntactic variables (like <statement>, <expression> or <primary>) correspond procedures, which, in general, read, analyse and compile a consecutive piece of source text, often calling other procedures for dealing with substructures. In this way, the complete source text of a program is read, analysed and compiled by one call of the procedure *Program* (see page 77). In this process, the source text is scanned from left to right by successive calls of *next symbol*, delivering the next basic symbol in the variable *last symbol*. Each basic symbol can, in this way, be inspected several times and from different procedures, until the next basic symbol is read. Then the previous one is lost forever, and all decisions to be based on it must have been taken; the compiler nowhere uses back-tracking and the syntactic analysis is completely a direct process (we will not go into what properties of the ALGOL 60 syntax allow for such a compiler).

In general, the process of analysis is one chain of decisions of how to continue and what to do next; those decisions are based on the basic symbol most recently read and on information about the identifier most recently read. Information about identifiers can be derived from their declarations or, in the case of formal parameters, from specifications and from the ways in which they are used elsewhere in the procedure they belong to.

As an example we look to the procedure *Stat* (see page 68). This procedure for the analysis of a statement reads essentially:

```
procedure Stat;
begin integer n;
      if letter last symbol
      then begin n:= Identifier;
                 if Designational(n)
                 then begin Label declaration(n); Stat end
                 else begin if Subscrvar(n) v last symbol = colonequal
                            then Assignstat(n)
                            else Procstat(n)
                      end
           end
      else ...
end Stat;
```

Decisions are taken here both on basic symbols (*letter last symbol* or *last symbol = colonequal)* or on properties of an identifier *(Designational(n)*, being true for label identifiers and for switch identifiers, or *Subscrvar(n)*, being true for array identifiers and for switch identifiers).

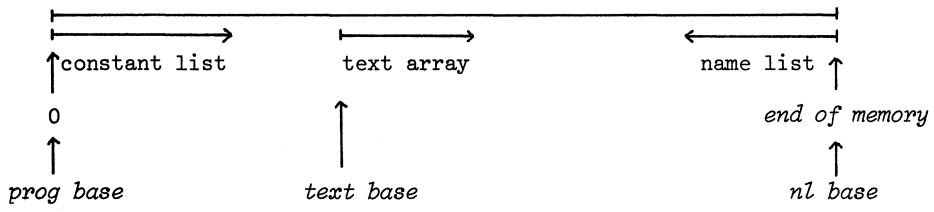
The information about identifiers, constituting a non-context-free element in the ALGOL 60 grammar, is collected in two separate scans. The compiler, therefore, is a three-pass compiler. In all three passes the same source text is scanned from left to right, and no "intermediate language" is used. In the first scan the source text is read from paper tape; in the next two scans the basic symbols are taken from a text array, in which they are stored during the first scan. The tasks of the three scans are:

prescan0 (pages 17/24):

1) construction of a name list (or symbol table), containing all identifiers declared in the program, all formal identifiers and all labels, with all relevant data. The block structure of the program is reflected in this list;

2) construction of a constant list, containing all unsigned numbers occurring in the program except small unsigned integers;
3) construction of a text array, containing all basic symbols of the source text except those occurring in comments (in this text array are also stored all "new-line-carriage-return" symbols for the purpose of keeping track of line numbers during the next two scans).

All three lists are stored in one and the same array called *space* in the following way:



prescan1 (pages 25/43):

1) collection of information about the use of formal parameters within their procedure body, e.g., occurrence as left part in an assignment statement, about whether they are used as array identifier or as procedure identifier, and if so the number of subscripts or parameters, possibly type information. This information is used in the third scan for checking consistent use of the identifier throughout the procedure body and for checking the correspondence between actual and formal parameters;
2) collection of information about the use of labels (for label identifiers that are not used otherwise than immediately following a *goto* in the same block as the one in which they occur as label, no "pseudo label variables" are introduced at run time);
3) address assignment to all variables and pseudo variables;
4) space reservation for all variables and pseudo variables that are global to all procedures in the program, and for all own variables and own arrays. This space is reserved in the array *space* immediately following the constant list; moreover, pseudo label variables placed in this "static space" are given an initial value;
5) addition to the name list of all identifiers of the program for which no declaration (or occurrence as formal parameter) is found.

translate (pages 45/90):

1) syntax checking;
2) address assignment to procedure-, switch- and label identifiers (with their "program addresses");
3) generation of the object program.

The object program is placed in the array *space* immediately following the static space. Thus, the object program is generated in core, ready for execution: the compiler is a "load-and-go" compiler.

Historically, the third scan was developed first; afterwards the two prescans were written in such a way that they collect the information required by the third scan. This can be seen on many places; some advantages and drawbacks will be discussed later on.

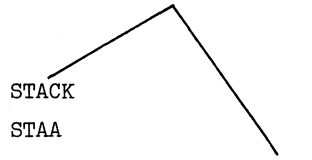
With the existence of a name list in mind the meaning of the statement *n:= Identifier* in the body of the procedure *Stat* given above will be clear: an identifier is read; then it is looked up in the name list; as function value of *Identifier* a pointer in this list is delivered. This pointer, then, can be used by Boolean procedures like *Designational* or *Subscrvar*, or can be handed over to other procedures (like *Assignstat*).

We conclude this section, dealing with scans, by some remarks about a number of constructions in which the first few symbols are not necessarily sufficient to recognize the syntactic category. As an example let us consider:

a:= b[i, j+1] ...

The basic symbol following "]" determines the meaning of the subscripted variable: a second left part or (part of) the right hand side expression. In such cases the object codes for both possibilities are chosen to have a common start and to differ only from the point where the discrimination is made. In the example given above the object codes read in terms of macros (with certain assumptions on the nature of the identifiers *b*, *i* and *j*):

macro	generated by	on page
TIV(i)	Arithname	47
STACK	Subscript list	48
TIV(j)	Arithname	47
STACK	Next term	46
TSIC(1)	Arithconstant	89
ADD	Next term	46
TAK(b)	Address description	48
STACK	Subscripted variable	47
STAA	Subscripted variable	47
TSR	Evaluation of	48



(the macro names TIV, TSIC, TAK, STAA and TSR are abbreviations for "take integer variable", "take small integer constant", "take array key", "stack A" and "take subscripted variable", respectively). These macros are generated, directly or indirectly, by a call of *Subscripted variable* occurring in the procedure *Realassign* on page 58 (or possibly in *Intassign*, if *a* is of type integer). If the symbol after "]" is not a colonequal, the subscripted variable is a first primary of an arithmetic expression. In that case, the first factor is completed by a call of *Next primary* (page 46), the first term by a call of *Next factor* (ibidem) and the arithmetic expression by a call of *Next term* (ibidem). These three procedures are called through a call of *Rest of arithexp* (page 52) from *Realassign*.

Other cases in which the syntactic category cannot be recognized immediately are:

for i:= <*arithmetic expression*> ...

in which the nature of the for list element is only known after the complete analysis and compilation of the arithmetic expression, or:

if (...) ...

where the expression between parentheses is either a Boolean expression

or an arithmetic expression. In some exceptional cases the object codes for two possible interpretations are generated both, e.g., in the case of an unsigned integer as actual parameter, if, at the given position, an interpretation as label is not excluded.

0.3 MODULES

An important factor that contributes to the readableness of especially the third scan is its modularity. By this we mean that for a study of the procedures that carry out the syntactic analysis and that generate the macros, no knowledge at all is necessary about, among others, the structure of the name list or the codes that are used in them for the description of kind and type of an identifier. All questions about properties of identifiers are put by means of procedure calls. Consequently, it is possible to choose any system whatsoever for the structure of the name list and for the codes used without any change in the procedures for text analysis and macro generation (the bulk of the third scan). In practice this fact was used during the development of the third scan for testing purposes at a time that no name list was available: all name list procedures then produced standard answers [7].
(The structure of the name list is relevant only for an analysis of the name-list procedures of the second or third scan themselves or for a study of the first scan, in which the construction of the name list - one of its major tasks - is integrated with syntax analysis to some extent. The information required for such a study is briefly presented in chapter 5).

In the sequel we describe the logical modules of the compiler. Some of them can be found in the code of more than one scan. Sometimes some of the procedures belonging to a logical module have been made global to all three scans by putting them in front of the procedures *prescan0*, *prescan1* and *translate*.

1) syntax analyser + macro generator
   present in all three scans; to it belong: in *prescan0* the procedures *Program* (page 17) upto and including *Begin statement* (page 23), in *prescan1* the procedures *Arithexp* (page 25) upto an including *Label declaration* (page 35) and *Scan code* (page 42), in *translate* the procedures

Arithexp (page 45) upto and including *Label declaration* (page 77), *Skip parameter list* (page 88), *Translate code* (page 88) and *Arithconstant* (page 89) upto and including *Relatmacro* (page 89). Also the global procedures *skip type declaration* (page 13) upto and including *skip rest of statement* (page 13) belong to this module.

2) macro processor

to be found in the third scan, running from the procedure *Substitute* (page 77) upto and including *B reaction* (page 82). Its main task is to generate one or more machine instructions to each macro and to put these in the array *space*. The macro processor is explained more fully in the next section.

3) name-list procedures

present in all three scans; to it belong the global procedures *read identifier* (page 11) upto and including *skip identifier* (page 13), in *prescan0* the procedures *Process identifier* and *Identifier* (page 23) and large portions of the procedures *Block* (pages 17, 18 and 19), *Declaration list* (pages 20 and 21), *Label declaration* (page 22) and *Int lab declaration* (page 23), in *prescan1* the procedures *Add type* (page 39) upto and including *Identifier* (page 42), and in *translate* the procedures *Code bits* (page 82) upto and including *Identifier* (page 87).

4) next symbol procedures

this module consists of the procedures *next symbol* (page 4) upto and including *operator last symbol* (page 9), all global. The isolation of the basic symbol from a sequence of flexowriter punchings is done hierarchically: *next tape symbol* converts to internal representation taking into account the most recent shift, *insymbol* isolates basic symbols, often by combining several symbols into one (e.g., ":" and "=" into ":="), *next basic symbol* deals with "new-line-carriage-return" symbols, and *next symbol* mainly skips comments.

5) unsigned-number procedures

unsigned number (page 10) reads and converts unsigned numbers; it is not realistic but intended only to illustrate some of the problems of number analysis and conversion. *unsigned number* delivers a mantissa,

being integer, in *value of constant*, a decimal exponent in *decimal exponent* and two Boolean values in *real number* and *small*. Besides *unsigned number* belong to this module: the global procedure *unsigned integer* (page 11), *Store numerical constant* (page 23) in *prescan0* and *Unsigned number* (page 89) in *translate*.

Note: procedure identifiers of global procedures are written with small letters only (except the procedure *ERRORMESSAGE*); the first letter of a procedure local to one of the three scans is always capital.

0.4 THE MACRO PROCESSOR

As has been stated before, the main task of the macro processor is to generate one or more machine instructions to each macro and to put these in the array *space*. This is actually done by the procedure *Produce* (page 79).

In addition to this task, the macro processor performs some local optimizations, carried out by the procedure *Macro2* (page 78). By these optimizations the code for expressions like $i + 2$, $x > y$ or $-k$ is simplified considerably. Instead of:

```
TIV(i)
STACK
TSIC(2)
ADD
```

for $i + 2$ (assuming i to be a simple variable of type integer),

```
TIV(i)
ADDSIC(2)
```

is obtained.

This optimization is performed on the following rules:

1) each macro triple consisting of the macro STACK, a *Simple arithmetic take macro* (one of the macros TRV, TIV, TRC, TIC or TSIC) and an *Optimizable operator* (one of the macros ADD, SUB, MUL, DIV, EQU, UQU,

LES, MST, MOR or LST) is replaced by one macro according to table 4.1;
2) each macro pair consisting of a Simple arithmetic take macro and the macro NEG is replaced by one macro according to the same table.
(The properties of each individual macro can be deduced from the value of its macro identifier; these values are given in section 2.1 and decoded in table 4.2).

For application of these rules the procedure *Macro2* can be in one out of four states, recorded in the global variable *state*:

state = 0 indicates the normal state,
state = 1 indicates that the macro STACK is in stock,
state = 2 indicates that the macro STACK and a Simple arithmetic take macro are in stock (the last being specified in the global variables *stack0* and *stack1*),
state = 3 indicates that a Simple arithmetic take macro is in stock (specified as above).

The fact that the procedure *Macro2* can have one or two macros in stock has some consequences in the case that the compiler needs the address at which the code for the next macro will be placed in the array *space*. This occurs e.g. frequently during the syntax analysis + macro generation for for-list elements. In these cases the macro processor has to be brought in its normal state first. This is done through the generation of the macro EMPTY (in the procedure *Ordercounter*, page 78) to which no instructions correspond at all.

The simple optimization rules given above are rather effective: on an average they eliminate 53 per cent. of the macro STACK and 62 per cent. of the macro NEG.

Another task of the macro processor is to keep track of the changes of the stack pointer that will occur at run time, for the purpose of minimizing the number of checks on stack-overflow at run time (the EL-X8 originally did not have memory protection; therefore certain macros had to check for overflow). An explanation of this aspect of the macro processor is too lengthy to be given here; it is concentrated in the procedure

Process stack pointer (page 80) and the resulting number is added to the object program as *sum of maxima* (see page 90).

To the macro processor belong also the procedures *Substitute* and *Subst2* (page 77). These have to do with forward references. As an example we discuss the expression *if b then 2 else 3*, for which the following macros are generated:

```
                TBV(b)              (take Boolean value)
                COJU(future1)       (conditional jump)
                TSIC(2)             (take small integer constant)
                JU(future2)         (jump)
future1:        TSIC(3)
future2:
```

The addresses future1 and future2 are unknown to the compiler at the moment that the macros COJU and JU, respectively, are generated. Therefore, they will be set zero initially, and in the local variables *future1* and *future2* of the procedure *Arithexp* (page 45) the locations in the array *space* will be recorded at which the jump instructions are placed (the assignment to these variables occurs through the last statement of the procedure *Macro2*). As soon as the true destinations for the jumps are known they can be substituted at those locations.

If several macros refer forward to the same, yet unknown location, they are linked together in a chain in the array *space*, ending at the macro that made the first reference to that location and therefore has a preliminary address part equal to zero. The same mechanism is used for references to procedures, switches and labels that are declared later. In that case the key address of the chain is recorded in the name list (cf. the procedure *Mark position in name list*, page 86).

0.5 ERROR HANDLING

An important task of any compiler is to check against syntactic and semantic errors in the source text. In general, it is relatively easy to find out whether a text is, according to certain rules, a correct program or not. It is far more difficult to interpret an incorrect text and to try to

indicate as many errors as possible. In fact, strictly spoken this is an ill-defined task, and to any strategy counter-examples with a very bad error-message performance are easily found.

The strategy adopted by the compiler in this tract is a rather simple one. It consists of two rules mainly:

1) if, in a given compiler procedure, the occurrence of a certain basic symbol is expected but inspection of *last symbol* shows that it is not there, an error message is produced and in general the task of the given compiler procedure will be considered to be completed. Perhaps the trouble is passed over then to the calling procedure that, in turn, may produce an error message and pass over the problem to its calling procedure;
2) there is only one level in the compiler that is allowed to skip a piece of source text until (a) certain basic symbol(s) is (are) met: the procedure *Compound tail*, present both in *prescan1* (page 31) and *translate* (page 69). It requires that after the translation of the first statement of (the remainder of) a compound tail a semicolon or an *end* is found. Otherwise it produces (in *translate*) an error message and skips to the next semicolon or *end* (during this skip, the syntax analysis is resumed temporarily at the occurrence of a compound statement or block for the purpose of matching corresponding *begin* and *end* symbols; for analogous reasons a string (like ';') is skipped as one integral unit).

As was indicated above, the performance of such a strategy is not always good. For the source text:

```
begin integer i;
      for i:= 1 step 1 until 10 do
      print(if 2 ≤ i < 4 then 0 else i)
end
```

the following error messages are produced, all with the same basic symbol ("<") in *last symbol*:

error number	meaning	produced by	page
331	"then" missing in expression of yet unknown type	Exp	55
334	expression starts with unadmissable symbol	Simplexp	56
332	"else" missing	Exp	55
361	")" missing	Parameter list	63
367	";" or "end" missing	Compound tail	69

These error messages do not seem very applicable! (note that the production of the messages 334 and 332 is caused by the fact that the procedure *Exp* does not strictly apply the first rule given above). However, in daily practice most error messages are to the point and in exceptional cases only it is hard to trace the true error(s) in the source text. Most important, moreover, is the fact that the compiler keeps on the rails, even in case of strong "misunderstanding" of a text.

A number of errors is detected in all three scans of the compiler. In general, these errors are reported in the third scan only (cf. the difference between the procedures *Compound tail* in *prescan1* and *translate*). This is done for two reasons: the third scan has been developed first, and it has to perform a more detailed analysis of the source text than the preceding scans. The fact that error messages are postponed to the last scan whenever possible has one important disadvantage: some texts never reach this scan (e.g. through an "end of file" in the first scan). A better approach would have been to report all errors related to delimiters in the first scan and all errors related to types in the third scan.
The check against errors is rather extended. It includes an effective check on the correspondence of actual and formal parameters, except for function designators and procedure statements with formal procedure identifier. A complete check on this correspondence can be carried out at run time only, but for the sake of run-time efficiency this has been omitted (this has caused troubles in, on an average, one out of 30 000 programs).

0.6 EVALUATION

The compiler presented in this tract has been in operation (in its machine-code version) for about 8 years now and it has had considerable influence on the use of ALGOL 60 in the Netherlands. Its success is due to several external factors, like the presence of an EL-X8 at several important institutions and the absence of an equivalent FORTRAN compiler for that machine. But certainly its success also resulted from a number of properties of the ALGOL system itself, like:

1) it is simple to use, even by the inexpert programmer;
2) it accepts rather complete ALGOL 60; the exceptions are well-defined;
3) it has good and rather complete diagnostics;
4) it is almost free of errors; exotic constructions and complex combinations do work properly;
5) the compiler is fast (although its name-list technique is rather primitive): on an average some 12 per cent. of the processing time of a program is compile time;
6) the compiler is rather short: it does not exceed 5000 instructions;
7) the object program is, given the absence of any advanced optimizing, rather efficient: on an average some 25 per cent. of the processing time of a program is used for arithmetic operations and for the evaluation of standard functions;
8) it has been easy to adapt the system to new situations, like extension of core memory, new operating systems or addition of a library of precompiled ALGOL procedures.

Most of these properties originate from the transparant structure of the compiler. It proved to be possible to run hundreds of ALGOL 60 programs a day, often produced by inexpert programmers, with only one or two people for user assistance.

There are also some weak points:

1) exceptionally, lacking correspondence of actual and formal parameters leads to troubles at run time;
2) exceptionally again, incorrect programs derail the compiler, causing addressing errors. This point is caused by the fact that, for some incorrect texts, the three scans of the compiler make a different interpretation of the block structure of that text. Partly this originates

from the structure of the first scan, which does not analyse statements except for the occurrence of labels, for clauses, compound statements or blocks (*prescan0* has been called sometimes "the art of skipping"). This fact could be improved by writing a more extended first scan. But that would not remedy the problem completely. For, another reason for incongruency of block-structure interpretation is the insystematic way in which, in the scans, decisions are taken: now on the contents of *last symbol*, then again on properties of some identifier. However, the properties of identifiers are not equally available in all three scans, and therefore they are not suited to base a parsing algorithm on;

3) some 23 per cent. of the processing time of a program is related to the access to array elements. The performance of the indexing system could have been improved by a number of measures like special index routines for one- and two-dimensional arrays or, in the case of an assignment to a subscripted variable, indexing before the right hand side expression is evaluated;

4) ambitious plans to implement own dynamic arrays and advanced string operations (hence the extension of the expression types with a type "string") never were completed, although the compiler is prepared for it completely.

It has been shown afterwards that the first three points can be fully circumvented with no essential change in the methods used for the ALGOL 60 system described here.

We conclude this section by a few remarks on the question how machine-(in) dependent this compiler is. The most important property of the EL-X8 in this respect is its arithmetic. The EL-X8 uses the Grau-representation [8] for floating-point numbers. Consequently, integers form a true subset of the reals and mixed-mode arithmetic expressions are no problem at all: no types are distinguished for arithmetic expressions. Secondly, all arithmetic operations are performed in the floating-point register and not in the top of a stack. In the other case no explicit stack operations would have been necessary. Another property of the EL-X8 is its addressing system. This, however, can hardly be found in the syntax analyzer; it has, of course, influenced the macro processor.

0.7 HINTS TO THE READER

For a study of the ALGOL text of the compiler the following order is proposed:

1) the compilation of arithmetic expressions (*Arithexp* upto and including *Subscript list*);
2) the macro processor (forget about *Process stack pointer* and *Process parameter*);
3) the compilation of Boolean expressions (*Boolexp* upto and including *Arboolrest*);
4) the compilation of string expressions (*Stringexp* sqq.), designational expressions (*Desigexp* sqq.) and expressions of unknown type (*Exp* sqq.);
5) the compilation of statements (*Statement* sqq.);
6) the compilation of blocks (*Block*);
7) the compilation of programs (*Program*);
8) the syntax-analysis part of *prescan1*;
9) the name-list procedures of *translate* and of *prescan1*;
10) the compilation of actual parameters (*Parameter list*);
11) *prescan0*.

Try always to formulate the task of yet unanalysed procedures!
The page on which each compiler procedure can be found is easily found in the cross-reference table (page 157).

0.8 REFERENCES

[1] NEDERKOORN, J.J.B.M., Prolegomena to X8 ALGOL,
Amsterdam, Mathematical Centre, report R 989 (1964).

[2] KRUSEMAN ARETZ, F.E.J. & B.J. MAILLOUX, The ELAN source text of the MC ALGOL 60 system for the EL X8,
Amsterdam, Mathematical Centre, report MR 84 (1966).

[3] KRUSEMAN ARETZ, F.E.J., Het objectprogramma, gegenereerd door de X8 - ALGOL 60 - vertaler van het MC,
Amsterdam, Mathematical Centre, report MR 121 (1971).

[4] KRUSEMAN ARETZ, F.E.J., C.G. VAN DER LAAN & J.P. HOLLENBERG, Programmeren voor rekenautomaten; de MC ALGOL 60 vertaler voor de EL X8,
Amsterdam, Mathematical Centre, MC Syllabus 13 (1971).

[5] KRUSEMAN ARETZ, F.E.J., On the bookkeeping of source-text line numbers during the execution phase of ALGOL 60 programs, in MC - 25 Informatica Symposium,
Amsterdam, Mathematical Centre Tracts 37 (1971).

[6] NAUR, P. (ed.), Revised Report on the Algorithmic Language ALGOL 60,
Copenhagen, Regnecentralen (1962).

[7] KRUSEMAN ARETZ, F.E.J., ALGOL 60 Translation for Everybody,
Elektronische Datenverarbeitung 6 (1964), 233-244.

[8] GRAU, A.A., On a Floating-Point Number Representation For Use with Algorithmic Languages,
Comm. ACM 5 (1962), 160-161.

1. THE ALGOL 60 PROGRAM

```
begin
comment     ALGOL 60 - version of the ALGOL 60 - translator
            for the EL - X8, F.E.J. Kruseman Aretz;

comment     basic symbols;
integer plus, minus, mul, div, idi, ttp, equ, uqu, les, mst, mor, lst,
        non, qvl, imp, or, and, goto, for, step, until, while, do,
        comma, period, ten, colon, semicolon, colonequal, space sbl,
        if, then, else, comment, open, close, sub, bus, quote, unquote,
        begin, end, own, rea, integ, boole, stri, array, proced, switch,
        label, value, true, false, new line, underlining, bar;

comment     other global integers;
integer case, lower case, stock, stock1, last symbol, line counter,
        last identifier, last identifier1,
        quote counter, run number, shift,
        type, chara, character, value character, arr decla macro,
        value of constant, decimal exponent, decimal count,
        word count, nlp, last nlp, n, integer label,
        block cell pointer, next block cell pointer,
        dimension, forcount, instruct counter, dp0,
        function letter, function digit, c variant,
        nl base, prog base, text base, text pointer,
        end of text, end of memory, start, end of list,
        d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14,
        d15, d16, d17, d18, d19, d20, d21, d22, d23, d24, d25,
        re, in, bo, st, ar, nondes, des, un, arbo, intlab;

comment     macro identifiers;
integer STACK, NEG, ADD, SUB, MUL, DIV, IDI, TTP,
        EQU, UQU, LES, MST, MOR, LST, STAB, NON, QVL, IMP, OR, AND,
        STAA, TSR, TSI, TSB, TSST, TFSU, TSL, TFSL, TCST,
        STSR, STSI, SSTSI, STSB, STSST, STFSU,
        ENTRIS, TFD, SAS, DECS, FAD, TASR, TASI, TASB, TASST, TASU,
        EXITIS, FADCV, TRSCV, TISCV, TSCVU, EXIT, TEST1, TEST2,
        CRV, CIV, CBV, CSTV, CLV, CEN, CLPN, TAV, TIAV,
        RAD, IAD, BAD, STAD, ORAD, OIAD, OBAD, OSTAD,
        LOS, EXITP, EXITPC, REJST, JUA, EMPTY,
        ABS, SIGN, ENTIER, SQRT, EXP, LN, END;

comment     macro2 identifiers;
integer TRV, TIV, TRC, TIC, TSIC, TBV, TBC, TSTV, TLV, TAK, TSWE,
        STR, STI, SSTI, STB, STST, DOS, DOS2, DOS3,
        JU, JU1, LJU, LJU1, COJU, YCOJU, SUBJ, ISUBJ, DECB, DO,
        TBL, ENTRB, DPTR, INCRB, TDL, ENTRPB, NIL, LAST,
        LAD, TDA, TNA, TAA, SWP, EXITB, EXITC, EXITSV,
        CODE, SLNC, RLNC, LNC;

comment     global Booleans;
Boolean letter last symbol, digit last symbol, arr declarator last symbol,
        type declarator last symbol, in array declaration, in formal list,
        text in memory, own type, int labels, real number, small,
        erroneous, derroneous, wanted;
```

```
comment         global arrays;
integer array internal representation[0 : 127], word delimiter[0 : 23],
              macro list[0 : 511], tabel[5 : 59],
              instruct list[0 : 203], mask[0 : 9];

comment         start of initialization;
        plus:= read;    minus:= read;   mul:= read;     div:= read;
        idi:= read;     ttp:= read;     equ:= read;     uqu:= read;
        les:= read;     mst:= read;     mor:= read;     lst:= read;
        non:= read;     qvl:= read;     imp:= read;     or:= read;
        and:= read;     goto:= read;    for:= read;     step:= read;
        until:= read;   while:= read;   do:= read;      comma:= read;
        period:= read;  ten:= read;     colon:= read;   semicolon:= read;
        colonequal:= read; space sbl:= read; if:= read; then:= read;
        else:= read;    comment:= read; open:= read;    close:= read;
        sub:= read;     bus:= read;     quote:= read;   unquote:= read;
        begin:= read;   end:= read;     own:= read;     rea:= read;
        integ:= read;   boole:= read;   stri:= read;    array:= read;
        proced:= read;  switch:= read;  label:= read;   value:= read;
        true:= read;    false:= read;   new line:= read;
        underlining:= read; bar:= read; lower case:= read;

        STACK:= read;   NEG:= read;     ADD:= read;     SUB:= read;
        MUL:= read;     DIV:= read;     IDI:= read;     TTP:= read;
        EQU:= read;     UQU:= read;     LES:= read;     MST:= read;
        MOR:= read;     LST:= read;     STAB:= read;    NON:= read;
        QVL:= read;     IMP:= read;     OR:= read;      AND:= read;
        STAA:= read;    TSR:= read;     TSI:= read;     TSB:= read;
        TSST:= read;    TFSU:= read;    TSL:= read;     TFSL:= read;
        TCST:= read;    STSR:= read;    STSI:= read;    SSTSI:= read;
        STSB:= read;    STSST:= read;   STFSU:= read;   ENTRIS:= read;
        TFD:= read;     SAS:= read;     DECS:= read;    FAD:= read;
        TASR:= read;    TASI:= read;    TASB:= read;    TASST:= read;
        TASU:= read;    EXITIS:= read;  FADCV:= read;   TRSCV:= read;
        TISCV:= read;   TSCVU:= read;   EXIT:= read;    TEST1:= read;
        TEST2:= read;   CRV:= read;     CIV:= read;     CBV:= read;
        CSTV:= read;    CLV:= read;     CEN:= read;     CLPN:= read;
        TAV:= read;     TIAV:= read;    RAD:= read;     IAD:= read;
        BAD:= read;     STAD:= read;    ORAD:= read;    OIAD:= read;
        OBAD:= read;    OSTAD:= read;   LOS:= read;     EXITP:= read;
        EXITPC:= read;  REJST:= read;   JUA:= read;     EMPTY:= read;
        ABS:= read;     SIGN:= read;    ENTIER:= read;  SQRT:= read;
        EXP:= read;     LN:= read;      END:= read;
```

```
        TRV:= read;     TIV:= read;     TRC:= read;     TIC:= read;
        TSIC:= read;    TBV:= read;     TBC:= read;     TSTV:= read;
        TLV:= read;     TAK:= read;     TSWE:= read;    STR:= read;
        STI:= read;     SSTI:= read;    STB:= read;     STST:= read;
        DOS:= read;     DOS2:= read;    DOS3:= read;    JU:= read;
        JU1:= read;     LJU:= read;     LJU1:= read;    COJU:= read;
        YCOJU:= read;   SUBJ:= read;    ISUBJ:= read;   DECB:= read;
        DO:= read;      TBL:= read;     ENTRB:= read;   DPTR:= read;
        INCRB:= read;   TDL:= read;     ENTRPB:= read;  NIL:= read;
        LAST:= read;    LAD:= read;     TDA:= read;     TNA:= read;
        TAA:= read;     SWP:= read;     EXITB:= read;   EXITC:= read;
        EXITSV:= read;  CODE:= read;    SLNC:= read;    RLNC:= read;
        LNC:= read;

        d0 :=        1; d1 :=        2; d2 :=        4; d3 :=        8;
        d4 :=       16; d5 :=       32; d6 :=       64; d7 :=      128;
        d8 :=      256; d9 :=      512; d10:=     1024; d11:=     2048;
        d12:=     4096; d13:=     8192; d14:=    16384; d15:=    32768;
        d16:=    65536; d17:=   131072; d18:=   262144; d19:=   524288;
        d20:=  1048576; d21:=  2097152; d22:=  4194304; d23:=  8388608;
        d24:= 16777216; d25:= 33554432;

        re:= 0;         in:= 1;         bo:= 2;         st:= 3;
        ar:= 4;         nondes:= 5;     des:= 6;        un:= 7;
        arbo:= 8;       intlab:= 9;

        function letter:= read; function digit:= read; c variant:= read;

        for n:= 0 step 1 until 127 do internal representation[n]:= read;
        for n:= 0 step 1 until  23 do word delimiter[n]:= read;
        for n:= 0 step 1 until 511 do macro list[n]:= read;
        for n:= 5 step 1 until  59 do tabel[n]:= read;
        for n:= 0 step 1 until 203 do instruct list[n]:= read;
        for n:= 0 step 1 until   9 do mask[n]:= d20 × read;

        end of memory:= read;
        end of list:= instruct list[174];
        text in memory:= true; erroneous:= derroneous:= false;
        wanted:= read = 0;


begin   integer array space[0 : end of memory];

procedure  ERRORMESSAGE (n); integer  n;
begin   integer  i;
        erroneous:= true;
        if n = 122 ∨ n = 123 ∨ n = 126 ∨ n = 127 ∨ n = 129
        then derroneous:= true;
        if n > run number
        then begin NLCR; PRINTTEXT (‡er‡); print (n);
                   print (line counter); print (last symbol);
                   for i:= 0 step 1 until word count do
                   print (space[nl base − last nlp − i])
             end
end  ERRORMESSAGE;
```

```
integer procedure  next symbol;
begin   integer  symbol;
next0:  symbol:= if stock1 ≥ 0 then stock1 else next basic symbol;
        stock1:= - 1;
        if (last symbol = semicolon ∨ last symbol = begin) ∧
           symbol = comment
        then begin skip0: symbol:= next basic symbol;
                          if symbol ≠ semicolon then goto skip0;
                          goto next0
             end;
        if last symbol = end
        then begin
             skip1: if symbol ≠ end ∧ symbol ≠ semicolon ∧ symbol ≠ else
                    then begin symbol:= next basic symbol; goto skip1 end
             end
        else
        if symbol = 125
        then begin stock1:= next basic symbol;
                   if stock1 > 9 ∧ stock1 < 64
                   then begin skip2: stock1:= next basic symbol;
                                     if stock1 > 9 ∧ stock1 < 64
                                     then goto skip2;
                                     if stock1 = colon
                                     then stock1:= next basic symbol
                                     else ERRORMESSAGE (100);
                                     if stock1 = open  then stock1:= - stock1
                                                       else ERRORMESSAGE (101);
                                     symbol:= comma
                              end
                   else symbol:= close
             end;
        digit last symbol := symbol < 10 ∨ symbol = period ∨
                             symbol = ten;
        letter last symbol:= symbol < 64 ∧ ¬ digit last symbol;
        next symbol:= last symbol:= symbol;
        outsymbol (run number, symbol);
        test pointers
end  next symbol;


integer procedure  next basic symbol;
begin   integer  symbol;
next0:  insymbol (run number, symbol);
        if symbol = new line
        then begin line counter:= line counter + 1;
                   if quote counter = 0
                   then begin outsymbol (run number, symbol);
                              goto next0
                        end
             end;
        next basic symbol:= symbol
end  next basic symbol;
```

```
procedure  insymbol (source, destination); integer  source, destination;
begin   integer  symbol, i;
        if (source = 200 ∨ source = 300) ∧ text in memory
        then
        begin destination:= bit string(d8 × shift, shift,
                              space[text base + text pointer]);
              if shift < 257
              then shift:= d8 × shift
              else begin shift:= 1; text pointer:= text pointer + 1 end
        end
        else
        begin symbol:= if stock > 0 then stock else next tape symbol;
              stock:= – 1;
              if symbol > bus
              then
              begin if symbol = 123 then symbol:= space sbl;
                    if quote counter > 0
                    then
                    begin if symbol = bar
                          then
                          begin next0: stock:= next tape symbol;
                                       if stock = bar then goto next0;
                                if stock = les
                                then quote counter:= quote counter + 1
                                else
                                if stock = mor
                                then
                                begin if quote counter = 1
                                      then begin symbol:= unquote;
                                                 stock:= – symbol
                                           end
                                      else quote counter:=
                                           quote counter – 1
                                end
                          end
                          else if symbol = 124
                               then symbol:= colon
                               else if symbol = 125 then symbol:= close
                    end
                    else
                    if symbol ≥ new line
                    then
                    begin if symbol = bar
                          then
                          begin next1: symbol:= next tape symbol;
                                       if symbol = bar then goto next1;
                                symbol:= if symbol = and then ttp else
                                         if symbol = equ then uqu else
                                         if symbol = les then quote else
                                         if symbol = mor then unquote
                                                         else 160
                          end
                          else
```

```
                    if symbol = underlining
                    then
                    begin symbol:= the underlined symbol;
                          if symbol > 63
                          then symbol:=
                               if symbol = 124 then idi else
                               if symbol = les then mst else
                               if symbol = mor then lst else
                               if symbol = non then imp else
                               if symbol = equ then qvl
                                               else 161
                          else
                          begin stock:= next tape symbol;
                                if stock = underlining
                                then
                                begin
                                 symbol:= the underlined symbol +
                                          d7 × symbol;
                                 for i:= 0 step 1 until 23 do
                                 begin
                                 if word delimiter[i] : d7 = symbol
                                 then
                                  begin
                                  symbol:= word delimiter[i];
                                  symbol:= symbol -
                                           symbol : d7 × d7;
                                  goto next2
                                  end
                                 end;
                                 symbol:= 162;
                          next2: stock:= next tape symbol;
                                 if stock = underlining
                                 then
                                 begin the underlined symbol;
                                       goto next2
                                 end
                                end
                                else symbol:= 161
                          end
                    end
                    else
                    if symbol = 124
                    then begin stock:= next tape symbol;
                               if stock = equ
                               then begin symbol:= colonequal;
                                          stock:= - symbol
                                    end
                               else symbol:= colon
                          end
               end
               else insymbol (runnumber, symbol)
          end;
          destination:= symbol
     end
end  insymbol;
```

```
integer procedure  the underlined symbol;
begin   integer  symbol;
        symbol:= next tape symbol;
        the underlined symbol:= if symbol = underlining
                                then the underlined symbol
                                else symbol
end  the underlined symbol;


integer procedure  next tape symbol;
begin   integer  symbol, head;
        symbol:= internal representation[REHEP];
        if symbol > 0
        then begin head:= symbol : d8;
                   next tape symbol:= abs (if case = lower case
                                           then symbol - d8 × head
                                           else head)
             end
        else begin if symbol < - 2 then case:= - symbol else
                   if symbol =   0 then ERRORMESSAGE (102) else
                   if symbol = - 1 then ERRORMESSAGE (103);
                   next tape symbol:= next tape symbol
             end
end  next tape symbol;


procedure  outsymbol (destination, source); integer destination, source;
begin   if destination = 100 ∧ text in memory
        then begin space[text base + text pointer]:=
                   space[text base + text pointer] + shift × source;
                   if shift < 257
                   then shift:= d8 × shift
                   else begin shift:= 1; text pointer:= text pointer + 1;
                              space[text base + text pointer]:= 0
                        end
             end
end  outsymbol;


Boolean procedure  arithoperator last symbol;
begin   arithoperator last symbol:= last symbol = plus  ∨
                                    last symbol = minus ∨
                                    last symbol = mul   ∨
                                    last symbol = div   ∨
                                    last symbol = idi   ∨
                                    last symbol = ttp
end  arithoperator last symbol;
```

```
Boolean procedure  relatoperator last symbol;
begin   relatoperator last symbol:= last symbol = les ∨
                                    last symbol = mst ∨
                                    last symbol = equ ∨
                                    last symbol = lst ∨
                                    last symbol = mor ∨
                                    last symbol = uqu
end  relatoperator last symbol;


Boolean procedure  booloperator last symbol;
begin   booloperator last symbol:= last symbol = qvl ∨
                                   last symbol = imp ∨
                                   last symbol = or  ∨
                                   last symbol = and
end  booloperator last symbol;


Boolean procedure  declarator last symbol;
begin   own type:= last symbol = own; if  own type then  next symbol;
        type:= if last symbol = rea   then 0 else
               if last symbol = integ then 1 else
               if last symbol = boole then 2 else
               if last symbol = stri  then 3 else 1000;
        if type < 4 then next symbol
                    else begin if own type then  ERRORMESSAGE (104);
                               if last symbol = array then  type:= 0
                         end;
        arr declarator last symbol:= last symbol = array;
        if arr declarator last symbol ∧ run number = 300
        then arr decla macro:= if own type
                               then (if type = 0 then ORAD else
                                     if type = 1 then OIAD else
                                     if type = 2 then OBAD else OSTAD)
                               else (if type = 0 then RAD else
                                     if type = 1 then IAD else
                                     if type = 2 then BAD else STAD);
        chara:= if arr declarator last symbol
                then 8
                else if last symbol = switch
                     then 14
                     else if last symbol = proced
                          then (if type < 4 then 16 else 24)
                          else type;
        type declarator last symbol:= chara < 4;
        if own type ∧ chara > 8 then ERRORMESSAGE (105);
        if type < 4 ∧ last symbol = switch then ERRORMESSAGE (106);
        if chara < 25 ∧ run number = 100
        then character:= ((if type declarator last symbol
                           then type
                           else if type < 4
                                then type + chara
                                else chara) +
                          (if own type then 32 else 0)) × d19;
        declarator last symbol:= chara < 25
end  declarator last symbol;
```

```
Boolean procedure  specifier last symbol;
begin   type:= if last symbol = rea   then 0 else
               if last symbol = integ then 1 else
               if last symbol = boole then 2 else
               if last symbol = stri  then 3 else
               if last symbol = array then 5 else 1000;
        if type < 4 then next symbol;
        chara:= if last symbol = label  then 6 else
                if last symbol = switch then 14 else 1000;
        if type + chara < 1000 then ERRORMESSAGE (107);
        chara:= if last symbol = array  then 8 else
                if last symbol = proced then (if type < 4 then 16
                                                          else 24)
                                             else chara;
        if chara < 25 then next symbol;
        if chara + type < 2000 ∧ run number = 100
        then begin value character:= (if chara > 8 then type else
                                      if chara = 6 then    6 else
                                      if  type = 5 then    8
                                      else type + chara) + 64;
                    character:= ((if type > 5
                                  then chara
                                  else (if type > 1 then type else 4) +
                                       (if chara < 1000 then chara
                                                        else 0))
                                 + 96) × d19
             end;
        specifier last symbol:= chara + type < 2000
end  specifier last symbol;


Boolean procedure  operator last symbol;
begin   operator last symbol:= arithoperator last symbol ∨
                               relatoperator last symbol ∨
                               booloperator  last symbol
end  operator last symbol;
```

```
procedure  unsigned number;
begin   integer  sign of exponent;
        if last symbol < 10
        then begin value of constant:= unsigned integer (0);
                   real number:= digit last symbol
             end
        else begin value of constant:= if last symbol = ten then 1
                                                             else 0;
                   real number:= true
             end;
        decimal exponent:= 0;
        if real number
        then begin
             next0: if last symbol < 10
                    then begin decimal exponent:= decimal exponent + 1;
                               next symbol; goto next0
                         end;
                    if last symbol = period
                    then begin next symbol;
                               value of constant:=
                               unsigned integer (value of constant);
                               decimal exponent:= decimal exponent -
                                                  decimal count;
                        next1: if last symbol < 10
                               then begin next symbol; goto next1 end
                         end;
                    if last symbol = ten
                    then begin next symbol; sign of exponent:= 1;
                               if last symbol = plus
                               then next symbol
                               else if last symbol = minus
                                    then begin next symbol;
                                               sign of exponent:= - 1
                                         end;
                               decimal exponent:= decimal exponent +
                                                  sign of exponent ×
                                                  unsigned integer (0);
                               if last symbol < 10
                               then begin  ERRORMESSAGE (108);
                                    next2: if next symbol < 9
                                           then goto next2
                                    end
                         end
             end;
        small:= value of constant < d15 ∧ ¬ real number
end  unsigned number;
```

```
integer procedure  unsigned integer (start); integer  start;
begin   integer  word;
        word:= start; decimal count:= 0;
        if last symbol > 9 then ERRORMESSAGE (109);
next0:  if last symbol < 10
        then begin if word < 6710886 ∨ (word = 6710886 ∧ last symbol < 4)
                   then begin word:= 10 × word + last symbol;
                              decimal count:= decimal count + 1;
                              next symbol; goto next0
                        end
             end;
        unsigned integer:= word
end  unsigned integer;


procedure  read identifier;
begin   integer  word, count;
        word:= count:= word count:= 0;
        if letter last symbol
        then
        begin
        next0: if last symbol < 64
               then
               begin if count = 4
                     then begin word count:= word count + 1;
                                word:= count:= 0
                          end;
                     word:= space[nl base - nlp - word count]:=
                     d6 × word - last symbol - 1;
                     count:= count + 1; next symbol; goto next0
               end
               else
               begin last identifier:= space[nl base - nlp];
                     last identifier1:= if word count = 0
                                        then 0
                                        else space[nl base - nlp - 1]
               end
        end
        else begin ERRORMESSAGE (110); space[nl base - nlp]:= - 1 end;
        space[nl base - nlp - word count - 1]:= 127 × d19
end  read identifier;


integer procedure  next pointer (n); integer  n;
begin   integer  word, pointer;
        pointer:= n;
next0:  word:= - space[nl base - pointer];
        if word <   0 then begin pointer:= pointer + 1; goto next0 end;
        if word ≥ d25 then begin pointer:= word - word : d13 × d13;
                                 goto next0
                           end;
        next pointer:= pointer
end  next pointer;
```

```
integer procedure  look up;
begin   integer  count, pointer;
        pointer:= block cell pointer +
                  (if in formal list ∨ in array declaration
                   then 5 else 4);
next0:  pointer:= next pointer (pointer);
        for count:= 0 step 1 until word count do
        begin if space[nl base - pointer - count] ≠
                 space[nl base - last nlp - count]
              then goto next1
        end;
        pointer:= pointer + word count + 1;
        if space[nl base - pointer] < 0
        then begin next1: pointer:= pointer + 1;
                          goto if space[nl base - pointer] < 0 then next1
                                                               else next0
             end;
        look up:= pointer
end  look up;


Boolean procedure  in name list;
begin   integer  head;
        if real number ∨ ¬ int labels
        then in name list:= false
        else begin head:= value of constant : d18;
                   space[nl base - nlp]:= - d12 - head;
                   space[nl base - nlp - 1]:=
                   (head - 1) × d18 - value of constant;
                   word count:= 1;
                   space[nl base - nlp - 2]:= 6 × d19;
                   last nlp:= nlp; integer label:= look up;
                   in name list:= integer label < nlp
             end
end  in name list;


integer procedure  next identifier (n); integer  n;
begin   integer  pointer;
        pointer:= next pointer (n) + 1;
next0:  if  space[nl base - pointer] < 0
        then begin pointer:= pointer + 1; goto next0 end;
        next identifier:= pointer
end  next identifier;
```

```
procedure  skip identifier;
begin   if last symbol < 64 then begin next symbol; skip identifier end
end  skip identifier;


procedure  skip type declaration;
begin   if letter last symbol then skip identifier;
        if last symbol = comma
        then begin next symbol; skip type declaration end
end  skip type declaration;


procedure  skip value list;
begin   if last symbol = value
        then begin next symbol; skip type declaration;
                   if last symbol = semicolon then next symbol
             end
end  skip value list;


procedure  skip specification list;
begin   if specifier last symbol
        then begin skip type declaration;
                   if last symbol = semicolon then next symbol;
                   skip specification list
             end
end  skip specification list;


procedure skip string;
begin   quote counter:= 1;
next0:  if next symbol ≠ unquote then goto next0;
        quote counter:= 0
end  skip string;


procedure skip rest of statement (pr); procedure pr;
begin   if last symbol = do
        then begin next symbol; pr end
        else
        if last symbol = goto ∨ last symbol = for ∨
           last symbol = begin
        then pr;
        if last symbol = quote then skip string;
        if last symbol ≠ semicolon ∧ last symbol ≠ end
        then begin next symbol;
                   skip rest of statement (pr)
             end
end  skip rest of statement;
```

```
integer procedure bit string (kn, n, code word); integer kn,n,code word;
begin   integer  k;
        k:= code word : kn; bit string:= (code word - k × kn) : n
end  bit string;


integer procedure  display level;
begin   display level:=
        bit string (d6, d0, space[nl base - block cell pointer - 1])
end  display level;


integer procedure  top of display;
begin   top of display:=
        bit string (d13, d6, space[nl base - block cell pointer - 1])
end  top of display;


integer procedure  local space;
begin   local space:= space[nl base - block cell pointer - 1] : d13
end  local space;


integer procedure  proc level;
begin   proc level:=
        bit string (d6, d0, space[nl base - block cell pointer - 2])
end  proc level;


Boolean procedure  use of counter stack;
begin   use of counter stack:=
        bit string (d7, d6, space[nl base - block cell pointer - 2]) = 1
end  use of counter stack;

integer procedure  status;
begin   status:= space[nl base - block cell pointer - 2] : d13
end status;


Boolean procedure  in code (n); integer  n;
begin   in code:= bit string (d25, d24, space[nl base - n - 1]) = 1
end  in code;


integer procedure  type bits (n); integer  n;
begin   type bits:= bit string (d22, d19, space[nl base - n])
end type bits;
```

```
Boolean procedure  local label (n); integer  n;
begin   local label:=
        nonformal label (n) ∧
        bit string(d6, d0,
        space[nl base - corresponding block cell pointer (n) - 1]) =
        display level
end  local label;


Boolean procedure  nonformal label (n); integer  n;
begin   nonformal label:= space[nl base - n] : d19 = 6
end nonformal label;


integer procedure  corresponding block cell pointer (n); integer  n;
begin   integer  p;
        p:= block cell pointer;
next0:  if n < p ∨ (n > space[nl base - p - 2] : d13 ∧ p > 0)
        then begin p:= space[nl base - p] : d13; goto next0 end;
        corresponding block cell pointer:= p
end  corresponding block cell pointer;


procedure  entrance block;
begin   block cell pointer:= next block cell pointer;
        next block cell pointer:=
        bit string (d13, d0, space[nl base - block cell pointer])
end  entrance block;


procedure  exit block;
begin   block cell pointer:= space[nl base - block cell pointer] : d13
end  exit block;


procedure  init;
begin   stock:= stock1:= last symbol:= word count:= - 1;
        shift:= 1;
        line counter:= quote counter:= forcount:= 0;
        in formal list:= in array declaration:= false;
        case:= lower case; text pointer:= 0
end init;
```

```
procedure test pointers;
begin   integer fprog, fnl, i, shift;
        if text in memory
        then
        begin fprog:= text base +
                      (if runnumber = 300 then text pointer else 0) -
                      instruct counter;
              fnl:= nl base - nlp -
                    (text base +
                     (if runnumber = 100 then text pointer
                                         else end of text));
              if fprog + fnl < 40
              then begin text in memory:= false; test pointers end
              else if fprog < 20
                   then begin shift:= (fnl - fprog) : 2;
                              for i:= text base + text pointer
                              step - 1 until text base do
                              space[i + shift]:= space[i];
                              text base:= text base + shift
                        end
                   else if fnl < 20
                        then
                        begin shift:= (fprog - fnl) : 2;
                              for i:= text base step 1
                              until text base + text pointer do
                              space[i]:= space[i + shift];
                              text base:= text base - shift
                        end
        end
        else if nl base - nlp - instruct counter < 20
             then begin ERRORMESSAGE (492); goto endrun end
end  test pointers;
```

```
procedure prescan0;
begin   integer old block cell pointer, displ level, prc level,
                global count, local count, label count, local for count,
                max for count, internal block depth, string occurrence,
                subcount, array pointer;


procedure  Program;
begin   integer  n;
        character:= 6 × d19;
        if letter last symbol
        then begin read identifier;
                   if last symbol = colon
                   then begin n:= Process identifier;
                              Label declaration (n)
                        end
                   else ERRORMESSAGE (111);
                   Program
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if last symbol = colon then Int lab declaration
                                          else ERRORMESSAGE (112);
                   Program
             end
        else
        if last symbol = begin
        then Begin statement
        else begin ERRORMESSAGE (113); next symbol; Program end
end  Program;


integer procedure  Block (proc identifier); integer  proc identifier;
begin   integer  dump1, dump2, dump3, dump4, dump5, dump6, dump7, dump8,
                 n, formal count;
        dump1:= block cell pointer; dump2:= local for count;
        dump3:= max for count;      dump4:= local count;
        dump5:= label count;        dump6:= internal block depth;
        dump7:= string occurrence;  dump8:= prc level;
        local for count:= max for count:= local count:= label count:=
        internal block depth:= string occurrence:= 0;
        block cell pointer:= nlp + 1;
        space[nl base - old block cell pointer]:=
        space[nl base - old block cell pointer] + block cell pointer;
        old block cell pointer:= block cell pointer;
        space[nl base - block cell pointer]:= dump1 × d13;
        space[nl base - block cell pointer - 1]:= displ level:=
                                                  displ level + 1;
        space[nl base - block cell pointer - 3]:= 0;
        nlp:= nlp + 6;
```

```
if proc identifier > 0
then
begin prc level:= displ level; formal count:= 0;
      space[nl base - block cell pointer - 4]:= - d25 - nlp;
      if last symbol = open
      then begin  character:= 127 × d19;
           next0: next symbol; Identifier;
                  space[nl base - nlp]:= 0; nlp:= nlp + 1;
                  formal count:= formal count + 1;
                  if last symbol = comma then goto next0;
                  if last symbol = close then next symbol
                                         else ERRORMESSAGE (114)
           end;
      if last symbol = semicolon then next symbol
                                 else ERRORMESSAGE (115);
      space[nl base - proc identifier - 1]:=
                                 d22 + formal count + 1;
      if last symbol = value
      then
      begin
      next1: next symbol; n:= Identifier;
             if n > last nlp  then ERRORMESSAGE (116)
                              else space[nl base - n]:= 95 × d19;
             nlp:= last nlp;
             if last symbol = comma then goto next1;
             if last symbol = semicolon then next symbol
                                        else ERRORMESSAGE (117)
      end;
next2: if specifier last symbol
      then
      begin
      next3: n:= Identifier;
             if n > last nlp
             then ERRORMESSAGE (118)
             else if space[nl base - n] = 127 × d19
                  then space[nl base - n]:= character
                  else if space[nl base - n] ≠  95 × d19
                       then ERRORMESSAGE (119)
                       else if value character > 75
                            then ERRORMESSAGE (120)
                            else
                            begin space[nl base - n]:=
                                  value character × d19;
                                  if type = 3
                                  then string occurrence:= d6
                            end;
             nlp:= last nlp;
             if last symbol = comma
             then begin next symbol; goto next3 end;
             if last symbol = semicolon then next symbol
                                        else ERRORMESSAGE (121);
             goto next2
      end;
```

```
                space[nl base - nlp]:= - d25 - 4 - dump1; nlp:= nlp + 1;
                space[nl base - block cell pointer - 4]:= - d25 - nlp;
                if last symbol = quote
                then begin  space[nl base - proc identifier - 1]:=
                            space[nl base - proc identifier - 1] + d24;
                     next4: next symbol;
                            if last symbol ≠ unquote then goto next4;
                            next symbol
                     end
                else
                if last symbol = begin
                then begin next symbol;
                           if declarator last symbol then Declaration list;
                           Compound tail; next symbol
                     end
                else Statement
           end
           else
           begin space[nl base - nlp]:= - d25 - 4 - dump1; nlp:= nlp + 1;
                 space[nl base - block cell pointer - 4]:= - d25 - nlp;
                 Declaration list; Compound tail
           end;

           space[nl base - block cell pointer - 2]:=
           d13 × nlp + string occurrence + prc level;
           for n:= 0 step 1 until max for count - 1 do
           space[nl base - nlp - n]:= d19;
           space[nl base - block cell pointer - 1]:=
           space[nl base - block cell pointer - 1] +
           d6 × (internal block depth + 1);
           if prc level > 1
           then space[nl base - block cell pointer - 1]:=
                space[nl base - block cell pointer - 1] +
                d13 × (max for count + local count)
           else global count:= global count + max for count +
                               local count + label count;
           nlp:= nlp + max for count;
           space[nl base - nlp]:= - d25 - 5 - block cell pointer;
           nlp:= nlp + 1;
           space[nl base - block cell pointer + 1]:= - d25 - nlp;
           displ level:= space[nl base - dump1 - 1];
           Block:= internal block depth + 1;
           block cell pointer:= dump1; local for count:= dump2;
           max for count:= dump3;      local count:= dump4;
           label count:= dump5;        internal block depth:= dump6;
           string occurrence:= dump7;  prc level:= dump8
end  Block;


procedure  Compound tail;
begin   Statement; if  last symbol = semicolon
                   then begin next symbol; Compound tail end
end  Compound tail;
```

```
procedure  Declaration list;
begin   integer  n, count;
next0:  if type declarator last symbol
        then begin  count:= 0;
             next1: count:= count + 1;
                    n:= Identifier;
                    if n < last nlp then ERRORMESSAGE (122);
                    if last symbol = comma
                    then begin next symbol; goto next1 end;
                    if type = 0 ∨ type = 3 then count:= 2 × count;
                    if own type then global count:= global count + count
                                else local count:= local count + count;
                    if type = 3 then string occurrence:= d6
             end
        else
        if arr declarator last symbol
        then begin  count:= array pointer:= 0;
             next2: count:= count + 1;
                    next symbol; n:= Identifier;
                    if n < last nlp then ERRORMESSAGE (123);
                    space[nl base - nlp]:= array pointer;
                    array pointer:= nlp; nlp:= nlp + 1;
                    if last symbol = comma then goto next2;
                    dimension:= 0;
                    if last symbol = sub
                    then
                    begin  subcount:= 1;
                    next3: next symbol;
                           if letter last symbol
                           then skip identifier
                           else if digit last symbol
                                then begin unsigned number;
                                           Store numerical constant
                                     end;
                           if last symbol = quote then skip string;
                           if last symbol = colon
                           then begin dimension:= dimension + 1;
                                      goto next3
                                end;
                           if last symbol = sub
                           then begin subcount:= subcount + 1;
                                      goto next3
                                end;
                           if last symbol ≠ bus then goto next3;
                           if subcount > 1
                           then begin subcount:= subcount - 1;
                                      goto next3
                                end;
                           next symbol;
                           if dimension = 0 then ERRORMESSAGE (124)
                           else dimension:= dimension + 1
                    end
                    else ERRORMESSAGE (125);
```

```
            next4: n:= space[nl base - array pointer];
                   space[nl base - array pointer]:= dimension;
                   array pointer:= n;
                   if n ≠ 0 then goto next4;
                   if own type
                   then global count:=
                        global count + (3 × dimension + 3) × count
                   else local count:= local count + count;
                   if last symbol = comma
                   then begin count:= 0; goto next2 end;
                   if type = 3 then string occurrence:= d6
            end
        else
        if last symbol = switch
        then begin  next symbol; n:= Identifier;
                   if n < last nlp then ERRORMESSAGE (126);
                   space[nl base - nlp]:= 0; nlp:= nlp + 1;
            next5: next symbol;
                   if letter last symbol
                   then skip identifier
                   else if digit last symbol
                        then begin unsigned number;
                                   Store numerical constant
                             end;
                   if last symbol = quote then skip string;
                   if last symbol ≠ semicolon then goto next5
            end
        else begin  next symbol; n:= Identifier;
                   if n < last nlp then ERRORMESSAGE (127);
                   nlp:= nlp + 1;
                   if type < 4
                   then begin space[nl base - nlp]:= type × d19;
                              nlp:= nlp + 1
                        end;
                   Block (n)
            end;
        if last symbol = semicolon then next symbol
                                   else ERRORMESSAGE (128);
        if declarator last symbol then goto next0
end  Declaration list;
```

```
procedure  Statement;
begin   integer  n, lfc;
        lfc:= local for count;
next0:  character:= 6 × d19;
next1:  if  letter last symbol
        then begin read identifier;
                   if last symbol = colon
                   then begin n:= Process identifier;
                              Label declaration (n);
                              goto next1
                        end
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if last symbol = colon
                   then begin Int lab declaration; goto next1 end
                   else Store numerical constant
             end
        else
        if last symbol = for
        then begin local for count:= local for count + 1;
                   if local for count > max for count
                   then max for count:= local for count
             end
        else
        if last symbol = begin
        then begin Begin statement; next symbol; goto next1 end
        else
        if last symbol = quote then skip string;
        if last symbol ≠ semicolon ∧ last symbol ≠ end
        then begin next symbol; goto next1 end;
        local for count:= lfc
end  Statement;


procedure  Label declaration (n); integer  n;
begin   if n < last nlp then ERRORMESSAGE (129);
        if label count = 0
        then space[nl base - block cell pointer - 3]:= d13 × (nlp - 1);
        label count:= label count + 2;
        space[nl base - nlp]:= d18; nlp:= nlp + 1;
        next symbol
end  Label declaration;


procedure  Int lab declaration;
begin   if real number
        then begin ERRORMESSAGE (130); next symbol end
        else begin int labels:= true;
                   in name list; nlp:= nlp + 3;
                   Label declaration (integer label)
             end
end  Int lab declaration;
```

```
procedure  Begin statement;
begin    integer  n;
         next symbol;
         if declarator last symbol
         then begin n:= Block (0);
                    if n > internal block depth
                    then internal block depth:= n
              end
         else Compound tail
end  Begin statement;


procedure  Store numerical constant;
begin    if ¬ small
         then begin space[prog base + instruct counter]:=
                    value of constant;
                    space[prog base + instruct counter + 1]:=
                    decimal exponent;
                    instruct counter:= instruct counter + 2
              end
end  Store numerical constant;


integer procedure  Process identifier;
begin    last nlp:= nlp; nlp:= nlp + word count + 2;
         space[nl base - nlp + 1]:= character;
         Process identifier:= look up
end  Process identifier;


integer procedure  Identifier;
begin    read identifier;
         Identifier:= Process identifier
end  Identifier;
```

```
main program of prescan0:
        runnumber:= 100; init;
        local for count:= max for count:= local count:= label count:=
        global count:= internal block depth:= string occurrence:=
        displ level:= prc level:= 0;
        old block cell pointer:= block cell pointer:= nlp;
        int labels:= false;
        space[text base]:=
        space[nl base - block cell pointer]:=
        space[nl base - block cell pointer - 1]:=
        space[nl base - block cell pointer - 3]:= 0;
        nlp:= block cell pointer + 6;
        space[nl base - block cell pointer - 4]:= - d25 - nlp;
        next symbol;
        Program;
        space[nl base - block cell pointer - 1]:=
                (global count + max for count + label count) × d13 +
                (internal block depth + 1) × (d13 + d6);
        space[nl base - block cell pointer - 2]:= nlp × d13;
        for n:= 0 step 1 until max for count - 1 do
        space[nl base - nlp - n]:= d19;
        nlp:= nlp + max for count;
        space[nl base - block cell pointer - 5]:= - d25 - nlp;
        end of text:= text pointer;
        output
end  prescan0;
```

```
procedure prescan1;
begin


procedure  Arithexp;
begin   if last symbol = if then Ifclause (Arithexp)
                          else Simple arithexp
end  Arithexp;


procedure  Simple arithexp;
begin   integer  n;
        if last symbol = plus ∨ last symbol = minus
        then
next0:  next symbol;
        if last symbol = open
        then begin next symbol; Arithexp;
                   if last symbol = close then next symbol
             end
        else
        if digit last symbol then unsigned number
        else
        if letter last symbol
        then begin n:= Identifier; Arithmetic (n);
                   Subscripted variable(n); Function designator(n)
             end
        else
        if last symbol = if then Arithexp;
        if arithoperator last symbol then goto next0
end  Simple arithexp;


procedure  Subscripted variable (n); integer  n;
begin   if last symbol = sub then begin Subscrvar (n);
                                        dimension:= Subscrlist;
                                        List length (n)
                                  end
end  Subscripted variable;


integer procedure  Subscrlist;
begin   next symbol; Arithexp;
        if last symbol = comma then Subscrlist:= Subscrlist + 1
                               else begin if last symbol = bus
                                          then next symbol;
                                          Subscrlist:= 1
                                    end
end  Subscrlist;


procedure  Boolexp;
begin   if last symbol = if then Ifclause (Boolexp)
                          else Simple boolean
end  Boolexp;
```

```
procedure  Simple boolean;
begin   integer  n, type;
        if last symbol = non  then next symbol;
        if last symbol = open then begin next symbol; Exp (type);
                                         if last symbol = close
                                         then next symbol
                                   end
        else
        if letter last symbol then begin n:= Identifier;
                                         Subscripted variable (n);
                                         Function designator (n);
                                         if arithoperator last symbol V
                                            relatoperator last symbol
                                         then Arithmetic (n)
                                         else Boolean (n)
                                   end
        else
        if digit last symbol V last symbol = plus V last symbol = minus
        then Simple arithexp
        else
        if last symbol = true V last symbol = false then next symbol;
        Rest of exp (type)
end  Simple boolean;


procedure  Stringexp;
begin   if last symbol = if then Ifclause (Stringexp)
                         else Simple stringexp
end  Stringexp;


procedure  Simple stringexp;
begin   integer  n;
        if last symbol = open
        then begin next symbol; Stringexp;
                   if last symbol = close then next symbol
             end
        else
        if letter last symbol
        then begin n:= Identifier; String (n);
                   Subscripted variable (n);
                   Function designator (n)
             end
        else
        if last symbol = quote
        then begin  quote counter:= 1;
             next0: next symbol;
                    if last symbol = unquote
                    then begin quote counter:= 0;
                               next symbol
                         end
                    else goto next0
             end
end  Simple stringexp;
```

```
procedure  Desigexp;
begin   if last symbol = if then Ifclause (Desigexp)
                          else Simple desigexp
end  Desigexp;


procedure  Simple desigexp;
begin   integer  n;
        if last symbol = open
        then begin next symbol; Desigexp;
                   if last symbol = close then next symbol
             end
        else
        if letter last symbol
        then begin n:= Identifier; Designational (n);
                   Subscripted variable (n)
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if in name list
                   then Designational (integer label)
             end
end  Simple desigexp;


procedure  Exp (type); integer  type;
begin   if last symbol = if
        then begin next symbol; Boolexp;
                   next symbol; Simplexp (type);
                   if last symbol = else
                   then begin next symbol; Type exp (type) end
             end
        else Simplexp (type)
end  Exp;


procedure  Type exp (type); integer  type;
begin   if type = ar ∨ type = re ∨ type = in
        then Arithexp
        else if type = bo
             then Boolexp
             else if type = st
                  then Stringexp
                  else if type = des
                       then Desigexp
                       else Exp (type)
end  Type exp;
```

```
procedure  Simplexp (type); integer  type;
begin   integer  n;
        type:= un;
        if last symbol = open
        then begin next symbol; Exp (type);
                   if last symbol = close then next symbol
             end
        else
        if letter last symbol
        then begin n:= Identifier; Subscripted variable (n);
                   Function designator (n);
                   if arithoperator last symbol ∨
                      relatoperator last symbol
                   then Arithmetic (n)
                   else if booloperator last symbol
                        then Boolean (n)
                        else begin if nonformal label (n)
                                   then Designational (n);
                                   type:= type bits (n)
                             end
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if in name list
                   then Designational (integer label)
                   else  type:= ar
             end
        else
        if last symbol = plus ∨ last symbol = minus
        then begin Simple arithexp; type:= ar end
        else
        if last symbol = non ∨ last symbol = true ∨ last symbol = false
        then begin Simple boolean; type:= bo end
        else
        if last symbol = quote
        then begin Simple stringexp; type:= st; goto end end;
        Rest of exp (type);
end:
end  Simplexp;


procedure  Rest of exp (type); integer  type;
begin   if arithoperator last symbol
        then begin next symbol; Simple arithexp;
                   type:= ar
             end;
        if relatoperator last symbol
        then begin next symbol; Simple arithexp;
                   type:= bo
             end;
        if booloperator last symbol
        then begin next symbol; Simple boolean;
                   type:= bo
             end
end  Rest of exp;
```

```
procedure  Assignstat (n); integer  n;
begin   Subscripted variable (n);
        if  last symbol = colonequal then  Right hand side (n)
end  Assignstat;


procedure  Right hand side (n); integer  n;
begin   integer  m, type, type n;
        Assigned to (n); type n:= type bits (n);
        next symbol;
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if  last symbol = colonequal
                   then
                   begin Insert (type n, m);
                         Right hand side (m); type:= type bits (m)
                   end
                   else
                   begin Function designator (m);
                         if arithoperator last symbol ∨
                            relatoperator last symbol
                         then Arithmetic (m)
                         else if booloperator last symbol
                              then Boolean (m)
                              else
                              begin Arbost (m);
                                    type:= if type n = re ∨ type n = in
                                           then ar
                                           else type n;
                                    Insert (type, m);
                                    type:= type bits (m);
                                    if type = re ∨ type = in
                                    then type:= ar
                              end;
                         Rest of exp (type)
                   end
             end
        else begin m:= type n; Type exp (type n);
                   if m ≠ nondes then type n:= m;
                   type:= if type n = re ∨ type n = in then ar
                                                      else type n
             end;
        Insert (type, n)
end  Right hand side;
```

```
procedure  Insert (type, n); integer  type, n;
begin   if type = re
        then Real (n)
        else if type = in
             then Integer (n)
             else if type = bo
                  then Boolean (n)
                  else if type = ar then Arithmetic (n)
end  Insert;


procedure  Function designator (n); integer  n;
begin   if last symbol = open then begin Function (n);
                                         dimension:= Parlist;
                                         List length (n)
                                   end
end  Function designator;


integer procedure  Parlist;
begin   next symbol; Actual parameter;
        if last symbol = comma
        then Parlist:= Parlist + 1
        else begin if last symbol = close then next symbol;
                   Parlist:= 1
             end
end  Parlist;


procedure  Actual parameter;
begin   integer  type;
        Exp (type)
end  Actual parameter;


procedure  Procstat (n); integer  n;
begin   Proc (n);
        dimension:= if last symbol = open then Parlist else 0;
        List length (n)
end  Procstat;
```

```
procedure  Statement;
begin   integer  n;
        if letter last symbol
        then begin n:= Identifier;
                   if last symbol = colon
                   then Labelled statement (n)
                   else begin if last symbol = sub ∨
                                 last symbol = colonequal
                              then Assignstat (n)
                              else Procstat (n)
                        end
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if last symbol = colon
                   then  Intlabelled statement
             end
        else
        if last symbol = goto then Gotostat
        else
        if last symbol = begin
        then begin next symbol;
                   if declarator last symbol then Block
                                             else Compound tail;
                   next symbol
             end
        else
        if last symbol = if then Ifclause (Statement)
        else
        if last symbol = for then Forstat
end  Statement;


procedure  Gotostat;
begin   integer  n;
        next symbol;
        if letter last symbol
        then begin n:= Identifier;
                   if ¬ local label (n)
                   then begin Designational (n);
                              Subscripted variable (n)
                        end
             end
        else Desigexp
end  Gotostat;


procedure  Compound tail;
begin   Statement;
        if last symbol ≠ semicolon ∧ last symbol ≠ end
        then skip rest of statement (Statement);
        if last symbol = semicolon
        then begin next symbol; Compound tail end
end  Compound tail;
```

```
procedure  Ifclause (pr); procedure  pr;
begin   next symbol; Boolexp;
        if last symbol = then then next symbol;
        pr;
        if last symbol = else then begin next symbol; pr end
end  Ifclause;


procedure  Forstat;
begin   integer  n;
        next symbol;
        if letter last symbol
        then begin  n:= Identifier; Arithmetic (n);
                    Subscripted variable (n);
                    if last symbol = colonequal
                    then
             next0: next symbol; Arithexp;
                    if last symbol = step
                    then begin next symbol; Arithexp;
                               if last symbol = until
                               then begin next symbol;
                                          Arithexp
                                    end
                         end
                    else
                    if last symbol = while
                    then begin next symbol; Boolexp end;
                    if last symbol = comma then goto next0;
                    if last symbol = do    then next symbol;
                    forcount:= forcount + 1;
                    Statement;
                    forcount:= forcount - 1
             end
end  Forstat;


procedure  Switch declaration;
begin   integer  n;
        next symbol;
        if letter last symbol
        then begin n:= Identifier;
                   if last symbol = colonequal
                   then begin dimension:= Switchlist;
                              Switch length (n)
                        end
             end
end  Switch declaration;


integer procedure  Switchlist;
begin   next symbol; Desigexp;
        if last symbol = comma then Switchlist:= Switchlist + 1
                               else Switchlist:= 1
end  Switchlist;
```

```
procedure  Array declaration;
begin   integer  i, n, count;
        next symbol; n:= Identifier; count:= 1;
next0:  if last symbol = comma then begin next symbol;
                                          if letter last symbol
                                          then skip identifier;
                                          count:= count + 1; goto next0
                                     end;
        if last symbol = sub   then begin in array declaration:= true;
                                          dimension:= Bound pair list;
                                          in array declaration:= false
                                     end
                               else dimension:= 0;
        Check dimension (n);
        if own type then for i:= 1 step 1 until count do
                         begin Address (n, instruct counter);
                               instruct counter:= instruct counter +
                                                  3 × dimension + 6;
                               n:= next identifier (n)
                         end;
        if last symbol = comma then Array declaration
end  Array declaration;


integer procedure  Bound pair list;
begin   next symbol; Arithexp;
        if last symbol = colon then begin next symbol; Arithexp end;
        if last symbol = comma
        then Bound pair list:= Bound pair list + 1
        else begin if last symbol = bus then next symbol;
                   Bound pair list:= 1
             end
end  Bound pair list;
```

```
procedure  Procedure declaration;
begin   integer  n, m;
        next symbol; n:= Identifier; entrance block;
        if last symbol = open
        then begin  in formal list:= true ;
             next0: next symbol; m:= Identifier;
                    if space[nl base - m] = 95 × d19
                    then begin ERRORMESSAGE (201);
                               space[nl base - m]:= 127 × d19
                         end;
                    if last symbol = comma then goto next0;
                    if last symbol = close then next symbol;
                    in formal list:= false
             end;
        if last symbol = semicolon then next symbol;
        skip value list; skip specification list;
        if in code (n)
        then Scan code (n)
        else begin if space[nl base - n] : d19 = 19 ∧
                   ¬ use of counter stack
                   then space[nl base - block cell pointer - 2]:=
                        space[nl base - block cell pointer - 2] + 64;
                   if last symbol = begin
                   then begin next symbol;
                              if declarator last symbol
                              then Declaration list;
                              Compound tail; next symbol
                         end
                   else Statement;
                   Addressing of block identifiers (n)
             end
end  Procedure declaration;


procedure  Block;
begin   entrance block; Declaration list; Compound tail;
        Addressing of block identifiers (0)
end  Block;


procedure  Declaration list;
begin   if typedeclarator last symbol then skip type declaration
        else
        if arr declarator last symbol then Array declaration
        else
        if last symbol = switch       then Switch declaration
                                      else Procedure declaration;
        if last symbol = semicolon    then next symbol;
        if declarator last symbol     then Declaration list
end  Declaration list;
```

```
procedure  Program;
begin   integer  n;
        if letter last symbol
        then begin n:= Identifier;
                   if last symbol = colon
                   then Label declaration (n);
                   Program
             end
        else
        if digit last symbol
        then begin unsigned number;
                   if in name list
                   then Label declaration (integer label);
                   Program
             end
        else
        if last symbol = begin
        then begin next symbol;
                   if declarator last symbol
                   then Block
                   else Compound tail
             end
        else begin next symbol; Program end
end  Program;


procedure  Labelled statement (n); integer  n;
begin   if nonformal label (n) then Label declaration (n);
        Statement
end  Labelled statement;


procedure  Intlabelled statement;
begin   if in name list then Label declaration (integer label);
        Statement
end  Intlabelled statement;


procedure  Label declaration (n); integer  n;
begin   if proc level = 0
        then begin Designational (n); Address (n, instruct counter);
                   space[nl base - n - 1]:=
                   space[nl base - n - 1] + instruct counter +
                   d20 × forcount;
                   space[prog base + instruct counter]:= 0;
                   space[prog base + instruct counter + 1]:=
                   d18 × display level + dp0;
                   instruct counter:= instruct counter + 2
             end
        else space[nl base - n - 1]:= space[nl base - n - 1] +
                                      d20 × forcount;
        next symbol
end  Label declaration;
```

```
procedure  Addressing of block identifiers (n); integer  n;
begin   integer  counter, f, code, code1;
        if n = 0 then space[nl base - block cell pointer - 1]:=
                      space[nl base - block cell pointer - 1] + d13;
        if proc level > 0
        then
        begin counter:= d9 × display level + d8;
              if n = 0
              then counter:= counter + 1 + d18
              else
              begin counter:= counter + display level + top of display;
                    f:= block cell pointer + 5;
              next0: f:= next identifier (f);
                    if f > block cell pointer
                    then
                    begin Address (f, counter);
                          code1:= space[nl base - f] : d18;
                          code:= code1 : 2;
                          counter:= counter +
                          (if code = 64 ∨ code = 67 ∨ code = 70
                           then 2
                           else if code < 96
                                then 1
                                else if code1 = 2 × code
                                     then 2 else 4);
                          goto next0
                    end;
                    counter:= counter + d18;
                    code:= space[nl base - n] : d19;
                    if code ≠ 24
                    then
                    begin f:= if wanted then 3 else
                              if code = 16 ∨ code = 19 then 2 else 1;
                          Address (n + 2, counter);
                          counter:= counter + f;
                          space[nl base - block cell pointer - 1]:=
                          space[nl base - block cell pointer - 1] +
                          d13 × f
                    end
              end;
```

```
            f:= status;
      next1: if space[nl base - f] > 0
            then begin Address (f, counter); counter:= counter + 1;
                       f:= f + 1;
                       goto next1
                 end;
            f:= block cell pointer + 4;
      next2: f:= next identifier (f); code:= space[nl base - f] : d19;
            if f > block cell pointer ∧ f < status ∧ code < 64
            then begin if code > 24
                       then
                       begin if  code < 36
                             then
                             begin Address (f, instruct counter);
                                   instruct counter:=
                                   instruct counter +
                                   (if code = 32 ∨ code = 35
                                    then 2 else 1)
                             end
                       end
                       else
                       if code < 14
                       then
                       begin if code ≠ 6 ∨
                                (code = 6 ∧
                                 bit string (d19, d18,
                                 space[nl base - f - 1]) = 0)
                             then
                             begin Address (f, counter);
                                   counter:=
                                   counter +
                                   (if code = 0 ∨ code = 3 ∨ code = 6
                                    then  2 else  1)
                             end
                       end;
                       goto next2
                 end;
            if counter > d18 + d9 × (display level + 1)
            then ERRORMESSAGE (202);
            exit block
        end
        else Static addressing
end  Addressing of block identifiers;
```

```
procedure  Static addressing;
begin   integer  f, code;
        f:= status;
next0:  if space[nl base - f] > 0
        then begin Address (f, instruct counter);
                   instruct counter:= instruct counter + 1; f:= f + 1;
                   goto next0
             end;
        f:= block cell pointer + 4;
next1:  f:= next identifier (f); code:= space[nl base - f] : d19;
        if f > block cell pointer ∧ f < status
        then begin if code > 24 ∧ code < 36 ∨ code < 14 ∧ code ≠ 6
                   then begin Address (f, instruct counter);
                              instruct counter:=
                              instruct counter +
                              (if code = 0 ∨ code = 3 ∨
                                  code = 32 ∨ code = 35 then 2
                                                        else 1)
                        end;
                   goto next1
             end;
        exit block
end Static addressing;
```

```
procedure  Add type (n, t); integer  n, t;
begin   integer  code, new code, type;
        new code:= code:= space[nl base - n] : d19;
        if code > 95
        then begin if code = 127
                   then new code:= 96 + t
                   else if code = 120 ∧ t < 6
                        then new code:= 112 + t
                        else
                        begin type:= code - code : 8 × 8;
                              if type = un ∨ (type = nondes ∧ t < 5) ∨
                                 (type = ar ∧ t < 2)
                              then new code:= code - type + t
                        end;
                   space[nl base - n]:=
                   space[nl base - n] - (code - new code) × d19
             end
end  Add type;

procedure  Real (n); integer  n;  begin Add type (n, re) end Real;

procedure  Integer (n); integer  n;  begin Add type (n, in) end Integer;

procedure  Boolean (n); integer  n;  begin Add type (n, bo) end Boolean;

procedure  String (n); integer  n;  begin Add type (n, st) end String;

procedure  Arithmetic (n); integer  n;
begin  Add type (n, ar) end  Arithmetic;

procedure  Arbost (n); integer  n;
begin   Add type (n, nondes) end  Arbost;
```

```
procedure  Designational (n); integer  n;
begin   integer  p;
        if nonformal label (n)
        then
        begin if bit string (d19, d18, space[nl base - n - 1]) = 1
              then
              begin space[nl base - n - 1]:=
                    abs (space[nl base - n - 1] - d18);
                    p:= corresponding block cell pointer (n);
                    if bit string (d6, d0, space[nl base - p - 2]) > 0
                    then begin space[nl base - p - 3]:=
                                space[nl base - p - 3] + 1;
                                space[nl base - p - 1]:=
                                space[nl base - p - 1] + d14
                         end
              end
        end
        else Add type (n, des)
end  Designational;


procedure  Assigned to (n); integer  n;
begin   integer  code;
        code:= space[nl base - n] : d19;
        if code > 95
        then
        begin if code = 127 then code:= 101;
              if code < 102 then space[nl base - n]:= code × d19 + d18
                            else Add type (n, nondes)
        end
end Assigned to;


procedure  Subscrvar (n); integer  n;
begin   integer  code, new code;
        code:= space[nl base - n] : d19;
        if code > 95
        then begin new code:= if code = 127
                              then 111
                              else if code < 104
                                   then code + 8
                                   else code;
                   space[nl base - n]:= space[nl base - n] +
                                        (new code - code) × d19
             end
end  Subscrvar;
```

```
procedure  Proc (n); integer  n;
begin   integer  code, new code;
        code:= space[nl base - n] : d19;
        if code > 95
        then begin new code:= if code = 127
                              then 120
                              else if code < 102
                                   then code + 16
                                   else code;
                   space[nl base - n]:= space[nl base - n] +
                                        (new code- code) × d19
             end
end  Proc;

procedure  Function (n); integer  n;
begin   Arbost (n); Proc (n) end  Function;

procedure  List length (n); integer  n;
begin   integer  word;
        if space[nl base - n] : d19 > 95
        then begin word:= space[nl base - n - 1];
                   if bit string (d18, d0, word) = 0
                   then space[nl base - n - 1]:= word + dimension + 1
             end
end  List length;

procedure  Switch length (n);  integer  n;
begin   space[nl base - n - 1]:= dimension + 1 end  Switch length;

procedure  Address (n, m); integer  n, m;
begin   integer  word;
        word:= space[nl base - n] : d18;
        space[nl base - n]:= word × d18 + m
end  Address;

procedure  Check dimension (n); integer  n;
begin   if space[nl base - n - 1] ≠ dimension + 1
        then begin ERRORMESSAGE (203);
                   space[nl base - n - 1]:= dimension + 1
             end
end  Check dimension;
```

```
integer procedure  Identifier;
begin   integer  n;
        last nlp:= nlp; read identifier; Identifier:= n:= look up;
        if n > nlp then Ask librarian;
        if n > nlp then begin ERRORMESSAGE (204);
                              nlp:= nlp + word count + 3;
                              space[nl base - nlp + 1]:= 0
                        end
end  Identifier;


procedure  Scan code (n); integer  n;
begin   block cell pointer:= space[nl base - block cell pointer] : d13;
next0:  next symbol; if  last symbol = minus then next symbol;
        if letter last symbol then Identifier else unsigned integer (0);
        if last symbol = comma then goto next0;
        if last symbol = unquote then next symbol
end  Scan code;


procedure  Ask librarian;
begin   comment if the current identifier occurs in the library
                then this procedure will add a new namecell to
                the name list and increase nlp;
end  Ask librarian;
```

```
main program of prescan1:
        if ¬ text in memory
        then begin NEWPAGE;
                   PRINTTEXT (*input tape for prescan1*)
             end;
        runnumber:= 200; init;
        block cell pointer:= next block cell pointer:= 0;
        dp0:= instruct counter;
        instruct counter:= instruct counter + top of display;
        space[nl base - nlp]:= - 1;
        next symbol; entrance block;
        Program; Static addressing;
        output
end  prescan1;
```

```
procedure translate;
begin

integer last lnc, lnc, last lncr, macro, parameter, state,
        stack0, stack1, b, ret level, max depth,
        ret max depth, max depth isr, max display length,
        max proc level, ecount, controlled variable, increment,
        10, 11, 12, 13, 14, 15, number of switch elements,
        switch identifier, switch list count, sword,
        address of constant, sum of maxima;

Boolean in switch declaration, in code body, if statement forbidden,
        complicated, complex step element;


procedure  Arithexp;
begin   integer  future1, future2;
        if last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (300)
                                       else next symbol;
                    Simple arithexp;
                    if last symbol = else
                    then begin  Macro2 (JU, future2);
                                Substitute (future1);
                                next symbol; Arithexp;
                                Substitute (future2)
                         end
                    else ERRORMESSAGE (301)
             end
        else Simple arithexp
end  Arithexp;


procedure  Simple arithexp;
begin   if last symbol = minus then begin  next symbol; Term;
                                            Macro (NEG)
                                    end
                               else begin  if last symbol = plus
                                           then next symbol;
                                           Term
                                    end;
        Next term
end  Simple arithexp;
```

```
procedure  Next term;
begin   if last symbol = plus  then begin  Macro (STACK);
                                            next symbol; Term;
                                            Macro (ADD); Next term
                                       end
        else
        if last symbol = minus then begin  Macro (STACK);
                                            next symbol; Term;
                                            Macro (SUB); Next term
                                       end
end  Next term;

procedure  Term;  begin  Factor; Next factor end  Term;

procedure  Next factor;
begin   if last symbol = mul then begin  Macro (STACK);
                                          next symbol; Factor;
                                          Macro (MUL); Next factor
                                     end
        else
        if last symbol = div then begin  Macro (STACK);
                                          next symbol; Factor;
                                          Macro (DIV); Next factor
                                     end
        else
        if last symbol = idi then begin  Macro (STACK);
                                          next symbol; Factor;
                                          Macro (IDI); Next factor
                                     end
end  Next factor;

procedure  Factor;  begin  Primary; Next primary end  Factor;

procedure  Next primary;
begin   if last symbol = ttp then begin  Macro (STACK);
                                          next symbol; Primary;
                                          Macro (TTP); Next primary
                                     end
end  Next primary;
```

```
procedure  Primary;
begin   integer  n;
        if last symbol = open then begin  next symbol; Arithexp;
                                          if last symbol = close
                                          then next symbol
                                          else ERRORMESSAGE (302)
                                  end
        else
        if digit last symbol  then begin  Unsigned number;
                                          Arithconstant
                                  end
        else
        if letter last symbol then begin  n:= Identifier;
                                          Subscripted variable (n);
                                          Function designator (n);
                                          Arithname (n)
                                  end
        else
        begin  ERRORMESSAGE (303);
               if last symbol = if ∨ last symbol = plus ∨
                  last symbol = minus
               then Arithexp
        end
end  Primary;


procedure  Arithname (n); integer n;
begin   if Nonarithmetic (n) then  ERRORMESSAGE (304);
        complicated:= Formal (n) ∨ Function (n);
        if Simple (n)
        then begin  if Formal (n)  then  Macro2 (DOS, n) else
                    if Integer (n) then  Macro2 (TIV, n)
                                   else  Macro2 (TRV, n)
             end
end  Arithname;


procedure  Subscripted variable (n); integer  n;
begin   if Subscrvar (n) then begin  Address description (n);
                                     if last symbol = colonequal
                                     then begin  Macro (STACK);
                                                 Macro (STAA)
                                          end
                                     else Evaluation of (n)
                              end
end  Subscripted variable;
```

```
procedure  Address description (n); integer  n;
begin   if last symbol = sub
        then begin  next symbol; dimension:= Subscript list;
                    Check dimension (n);
                    if Formal (n)        then  Macro2 (DOS, n) else
                    if Designational (n) then  Macro2 (TSWE, n)
                                         else  Macro2 (TAK, n)
             end
        else ERRORMESSAGE (305)
end  Address description;


procedure  Evaluation of (n); integer  n;
begin   if  Designational(n)
        then begin  if Formal (n) then  Macro (TFSL)
                                  else  Macro (TSL)
             end
        else
        if  Boolean (n) then Macro (TSB)  else
        if  String (n)  then Macro (TSST) else
        if  Formal (n)  then Macro (TFSU) else
        if  Integer (n) then Macro (TSI)  else Macro (TSR)
end  Evaluation of;


integer procedure  Subscript list;
begin   Arithexp;
        if last symbol = comma
        then begin  Macro (STACK); next symbol;
                    Subscript list:= Subscript list + 1
             end
        else begin  if last symbol = bus
                    then next symbol
                    else ERRORMESSAGE (306);
                    Subscript list:= 1
             end
end  Subscript list;
```

```
procedure  Boolexp;
begin   integer future1, future2;
        if  last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (307)
                                       else next symbol;
                    Simple boolean;
                    if last symbol = else
                    then begin Macro2 (JU, future2);
                               Substitute (future1);
                               next symbol; Boolexp;
                               Substitute (future2)
                         end
                    else ERRORMESSAGE (308)
             end
        else Simple boolean
end Boolexp;

procedure  Simple boolean;
begin   Implication; Next implication end  Simple boolean;

procedure  Next implication;
begin   if last symbol = qvl then begin  Macro (STAB);
                                          next symbol; Implication;
                                          Macro (QVL); Next implication
                                  end
end  Next implication;

procedure  Implication;  begin  Boolterm; Next boolterm end  Implication;

procedure  Next boolterm;
begin   if last symbol = imp then begin  Macro (STAB);
                                          next symbol; Boolterm;
                                          Macro (IMP); Next boolterm
                                  end
end  Next boolterm;

procedure  Boolterm;  begin  Boolfac; Next boolfac end  Boolterm;

procedure  Next boolfac;
begin   if last symbol = or  then begin  Macro (STAB);
                                          next symbol; Boolfac;
                                          Macro (OR); Next boolfac
                                  end
end  Next boolfac;

procedure  Boolfac;  begin  Boolsec; Next boolsec end  Boolfac;
```

```
procedure  Next boolsec;
begin   if last symbol = and then begin  Macro (STAB);
                                          next symbol; Boolsec;
                                          Macro (AND); Next boolsec
                                  end
end  Next boolsec;


procedure  Boolsec;
begin   if last symbol = non then begin  next symbol; Boolprim;
                                          Macro (NON)
                                  end
                           else Boolprim
end  Boolsec;


procedure  Boolprim;
begin   integer type, n;
        if last symbol = open
        then begin next symbol; Arboolexp (type);
                   if last symbol = close then next symbol
                                          else ERRORMESSAGE (309);
                   if type = ar   then Rest of relation else
                   if type = arbo then begin if arithoperator last symbol
                                             then Rest of relation
                                             else Relation
                                       end
             end
        else
        if letter last symbol then begin n:= Identifier;
                                         Subscripted variable (n);
                                         Boolprimrest (n)
                                   end
        else
        if digit last symbol ∨ last symbol = plus ∨ last symbol = minus
        then begin Simple arithexp; Rest of relation end
        else
        if last symbol = true ∨ last symbol = false
        then begin Macro2 (TBC, last symbol); next symbol end
        else ERRORMESSAGE (310)
end  Boolprim;
```

```
Boolean procedure  Relation;
begin   integer relmacro;
        if  relatoperator last symbol
        then begin  relmacro:= Relatmacro; Macro (STACK);
                    next symbol; Simple arithexp;
                    Macro (relmacro); Relation:= true
             end
        else Relation:= false
end  Relation;


procedure  Rest of relation;
begin   Rest of arithexp;
        if ¬ Relation then  ERRORMESSAGE (311)
end  Rest of relation;


procedure  Boolprimrest (n); integer  n;
begin   Function designator (n);
        if Arithmetic (n) ∨ arithoperator last symbol
           ∨ relatoperator last symbol
        then begin  Arithname (n); Rest of relation end
        else Boolname (n)
end  Boolprimrest;


procedure  Boolname (n); integer  n;
begin   if Nonboolean (n) then ERRORMESSAGE (312);
        if Simple (n)     then begin if Formal (n) then  Macro2 (DOS, n)
                                                   else  Macro2 (TBV, n)
                                   end
end  Boolname;


procedure  Arboolexp (type); integer  type;
begin   integer  future1, future2;
        if last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (313)
                                       else next symbol;
                    Simple arboolexp (type);
                    if last symbol = else
                    then
                    begin  Macro2 (JU, future2); Substitute (future1);
                           next symbol;
                           if type = bo then Boolexp else
                           if type = ar then Arithexp
                                        else Arboolexp (type);
                           Substitute (future2)
                    end
                    else ERRORMESSAGE (314)
             end
        else Simple arboolexp (type)
end  Arboolexp;
```

```
procedure  Simple arboolexp (type); integer  type;
begin   integer  n;
        if last symbol = open
        then begin  next symbol; Arboolexp (type);
                    if last symbol = close then next symbol
                                           else ERRORMESSAGE (315);
                    if type = bo ∨
                       type = arbo ∧ booloperator last symbol
                    then begin Rest of boolexp; type:= bo end
                    else if type = ar ∨
                               arithoperator last symbol ∨
                               relatoperator last symbol
                         then Rest of arboolexp (type)
             end
        else
        if letter last symbol
        then begin  n:= Identifier; Subscripted variable (n);
                    Arboolrest (type, n)
             end
        else
        if digit last symbol ∨ last symbol = plus ∨ last symbol = minus
        then begin Simple arithexp; Rest of arboolexp (type) end
        else
        if last symbol = non ∨ last symbol = true ∨ last symbol = false
        then begin Simple boolean; type:= bo end
        else
        begin  ERRORMESSAGE (316); type:= arbo end
end  Simple arboolexp;


procedure  Rest of arithexp;
begin   Next primary; Next factor; Next term end  Rest of arithexp;


procedure  Rest of boolexp;
begin   Next boolsec; Next boolfac; Next boolterm; Next implication
end Rest of boolexp;


procedure  Rest of arboolexp (type); integer  type;
begin   Rest of arithexp;
        if Relation
        then begin  Rest of boolexp; type:= bo end else type:= ar
end  Rest of arboolexp;
```

```
procedure  Arboolrest (type, n); integer  type, n;
begin   Function designator (n);
        if Boolean (n) ∨ booloperator last symbol
        then begin  Boolname (n); Rest of boolexp; type:= bo end
        else
        if  Arithmetic (n) ∨ arithoperator last symbol ∨
            relatoperator last symbol
        then begin  Arithname (n); Rest of arboolexp (type) end
        else begin  if String (n) ∨ Designational (n)
                    then ERRORMESSAGE (317);
                    Macro2 (DOS, n); type:= arbo
             end
end  Arboolrest;


procedure  Stringexp;
begin   integer  future1, future2;
        if last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (318)
                                       else next symbol;
                    Simple stringexp;
                    if last symbol = else
                    then begin  Macro2 (JU, future2);
                                Substitute (future1);
                                next symbol; Stringexp;
                                Substitute (future2)
                         end
                    else ERRORMESSAGE (319)
             end
        else Simple stringexp
end  Stringexp;


procedure  Simple stringexp;
begin   integer  future, n;
        if last symbol = open
        then begin  next symbol; Stringexp;
                    if last symbol = close then next symbol
                                       else ERRORMESSAGE (320)
             end
        else
        if  letter last symbol
        then begin  n:= Identifier; Subscripted variable (n);
                    Stringname (n)
             end
        else
        if last symbol = quote
        then begin  Macro (TCST); future:= 0; Macro2 (JU, future);
                    Constant string; Substitute (future)
             end
        else ERRORMESSAGE (321)
end  Simple stringexp;
```

```
procedure  Stringname (n); integer  n;
begin   if Nonstring (n) then  ERRORMESSAGE (322);
        Function designator (n);
        if Simple (n) then begin  if Formal (n) then  Macro2 (DOS, n)
                                                else  Macro2 (TSTV, n)
                               end
end  Stringname;


procedure  Desigexp;
begin   integer  future1, future2;
        if last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (323)
                                       else next symbol;
                    Simple desigexp;
                    if last symbol = else
                    then begin  Macro2 (JU, future2);
                                Substitute (future1);
                                next symbol; Desigexp;
                                Substitute (future2)
                         end
                    else ERRORMESSAGE (324)
             end
        else Simple desigexp
end  Desigexp;


procedure  Simple desigexp;
begin   integer  n;
        if  last symbol = open
        then begin  next symbol; Desigexp;
                    if last symbol = close then next symbol
                                           else ERRORMESSAGE (325)
             end
        else
        if letter last symbol then begin n:= Identifier;
                                         Subscripted variable (n);
                                         Designame (n)
                                   end
        else
        if  digit last symbol then begin Unsigned number;
                                         if in name list
                                         then Macro2 (TLV, integer label)
                                         else ERRORMESSAGE (326)
                                   end
        else ERRORMESSAGE (327)
end  Simple desigexp;
```

```
procedure  Designame (n); integer  n;
begin   if Nondesignational (n) then ERRORMESSAGE (328);
        if Simple (n)
        then begin  if Formal (n) then Macro2 (DOS, n)
                                  else Macro2 (TLV, n)
             end
end  Designame;


procedure  Ardesexp (type); integer  type;
begin   Exp (type);
        if type = bo ∨ type = st then ERRORMESSAGE (329);
        if type = un             then type:= intlab else
        if type = nondes         then type:= ar
end  Ardesexp;


procedure  Nondesexp (type); integer  type;
begin   Exp (type);
        if type = des    then ERRORMESSAGE (330);
        if type = un     then type:= nondes else
        if type = intlab then type:= ar
end  Nondesexp;


procedure  Exp (type); integer  type;
begin   integer  future1, future2;
        if last symbol = if
        then begin  future1:= future2:= 0;
                    next symbol; Boolexp; Macro2 (COJU, future1);
                    if last symbol ≠ then then ERRORMESSAGE (331)
                                       else next symbol;
                    Simplexp (type);
                    if last symbol = else
                    then
                    begin  Macro2 (JU, future2);
                           Substitute (future1); next symbol;
                           if type = ar     then Arithexp        else
                           if type = bo     then Boolexp         else
                           if type = st     then Stringexp       else
                           if type = des    then Desigexp        else
                           if type = intlab then Ardesexp (type) else
                           if type = nondes then Nondesexp (type)
                                            else Exp (type);
                           Substitute (future2)
                    end
                    else  ERRORMESSAGE (332)
             end
        else  Simplexp (type)
end  Exp;
```

```
procedure  Simplexp (type); integer  type;
begin   integer  n;
        if last symbol = open
        then begin  next symbol; Exp (type);
                    if last symbol = close then next symbol
                                           else ERRORMESSAGE (333);
                    if type = bo ∨ (type = nondes ∨ type = un) ∧
                       booloperator last symbol
                    then begin Rest of boolexp; type:= bo end
                    else
                    if type ≠ st ∧ type ≠ des ∧ operator last symbol
                    then  Rest of arboolexp (type)
             end
        else
        if letter last symbol
        then begin  n:= Identifier; Subscripted variable (n);
                    Exprest (type, n)
             end
        else
        if digit last symbol
        then begin  Unsigned number; Arithconstant;
                    if in name list ∧ ( ¬ operator last symbol)
                    then begin  Macro2 (TLV, integer label);
                                type:= intlab
                         end
                    else Rest of arboolexp (type)
             end
        else
        if last symbol = plus ∨ last symbol = minus
        then  Simple arboolexp (type)
        else
        if last symbol = non ∨ last symbol = true ∨ last symbol = false
        then begin  Simple boolean; type:= bo end
        else
        if last symbol = quote then begin Simple stringexp; type:= st end
        else
        begin  ERRORMESSAGE (334); type:= un end
end  Simplexp;
```

```
procedure  Exprest (type, n); integer  type, n;
begin   if Designational (n) then begin Designame (n); type:= des end
        else
        if String (n)        then begin Stringname (n); type:= st end
        else
        begin  Function designator (n);
               if Boolean (n) ∨ booloperator last symbol
               then begin Boolname (n); Rest of boolexp; type:= bo end
               else
               if Arithmetic (n) ∨ arithoperator last symbol ∨
                  relatoperator last symbol
               then begin Arithname (n); Rest of arboolexp (type) end
               else begin if Simple (n) then  Macro2 (DOS, n);
                          type:= if Unknown (n) then un else nondes
                    end
        end
end  Exprest;


procedure  Assignstat (n); integer  n;
begin   Subscripted variable (n);
        if last symbol = colonequal then Distribute on type (n)
                                    else ERRORMESSAGE (335)
end  Assignstat;


integer procedure  Distribute on type (n); integer  n;
begin   if Integer (n)
        then begin  Intassign (n); Distribute on type:= in end
        else
        if  Real (n)
        then begin  Realassign (n); Distribute on type:= re end
        else
        if Boolean (n)
        then begin  Boolassign (n); Distribute on type:= bo end
        else
        if String (n)
        then begin  Stringassign (n); Distribute on type:= st end
        else Distribute on type:= if Arithmetic (n) then Arassign (n)
                                                    else Unassign (n)
end  Distribute on type;


procedure  Prepare (n); integer  n;
begin   if  Function (n)
        then begin  if Formal (n) then ERRORMESSAGE (336)
                    else
                    if  Outside declaration (n) then ERRORMESSAGE (337)
                    else n:= Local position (n)
             end
        else if Simple (n) ∧ Formal (n) then Macro2 (DOS2, n);
        next symbol
end  Prepare;
```

```
Boolean procedure  Intassign (n); integer  n;
begin   integer m;  Boolean  rounded;
        if  Noninteger (n) then  ERRORMESSAGE (338);
        Prepare (n); rounded:= false;
        if letter last symbol
        then begin  m:= Identifier; Subscripted variable (m);
                    if last symbol = colonequal
                    then rounded:= Intassign (m)
                    else begin  Function designator (m);
                                Arithname (m); Rest of arithexp
                         end
             end
        else Arithexp;
        if Subscrvar (n)
        then begin  if Formal (n) then Macro (STFSU)
                    else
                    if rounded    then Macro (SSTSI)
                                  else Macro (STSI)
             end
        else if Formal (n) then Macro2 (DOS3, n)
        else if rounded    then Macro2 (SSTI, n)
                           else Macro2 (STI, n);
        Intassign:= Formal (n) ┐ rounded
end  Intassign;


procedure  Realassign (n); integer  n;
begin   integer  m;
        if Nonreal (n) then ERRORMESSAGE (339);
        Prepare (n);
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if last symbol = colonequal
                   then Realassign (m)
                   else begin Function designator (m);
                              Arithname (m); Rest of arithexp
                        end
             end
        else Arithexp;
        if  Subscrvar (n)
        then begin if Formal (n) then Macro (STFSU)
                                 else Macro (STSR)
             end
        else if Formal (n) then Macro2 (DOS3, n)
                           else Macro2 (STR, n)
end  Realassign;
```

```
procedure  Boolassign (n); integer  n;
begin   integer  m;
        if Nonboolean (n) then  ERRORMESSAGE (340);
        Prepare (n);
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if last symbol = colonequal
                   then Boolassign (m)
                   else begin Boolprimrest (m); Rest of boolexp end
             end
        else Boolexp;
        if  Subscrvar (n) then Macro (STSB)
                          else if Formal (n) then Macro2 (DOS3, n)
                                             else Macro2 (STB, n)
end  Boolassign;


procedure  Stringassign (n); integer  n;
begin   integer  m;
        if  Nonstring (n) then  ERRORMESSAGE (341);
        Prepare (n);
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if last symbol = colonequal
                   then Stringassign (m)
                   else Stringname (m)
             end
        else Stringexp;
        if Subscrvar (n) then Macro (STSST)
                         else if Formal (n) then Macro2 (DOS3, n)
                                            else Macro2 (STST, n)
end  Stringassign;


integer procedure  Arassign (n); integer  n;
begin   integer  type, m;
        if  Nonarithmetic (n) then  ERRORMESSAGE (342);
        Prepare (n); type:= ar;
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if last symbol = colonequal
                   then begin if Nonarithmetic (m)
                              then ERRORMESSAGE (343);
                              type:= Distribute on type (m)
                        end
                   else begin Function designator (m);
                              Arithname (m); Rest of arithexp
                        end
             end
        else Arithexp;
        if Subscrvar (n) then Macro (STFSU) else Macro2 (DOS3, n);
        Arassign:= type
end  Arassign;
```

```
integer procedure  Unassign (n); integer  n;
begin   integer  type, m;
        if  Nontype (n) then  ERRORMESSAGE (344);
        Prepare (n);
        if letter last symbol
        then begin m:= Identifier; Subscripted variable (m);
                   if Nontype (m) then ERRORMESSAGE (345);
                   if last symbol = colonequal
                   then type:= Distribute on type (m)
                   else Exprest (type, m)
             end
        else Nondesexp (type);
        if Subscrvar (n)
        then begin if type = bo then  Macro (STSB)
                   else
                   if type = st then  Macro (STSST)
                                else  Macro (STFSU)
             end
        else Macro2 (DOS3, n);
        Unassign:= type
end  Unassign;


procedure  Function designator (n); integer  n;
begin   if Proc (n)
        then begin if Nonfunction (n) then ERRORMESSAGE (346);
                   Procedure call (n)
             end
end  Function designator;


procedure  Procstat (n); integer  n;
begin   if  Proc (n)
        then begin Procedure call (n);
                   if  ¬ (In library (n) ∨  Function (n))
                   then last lnc:= - n;
                   if  Formal (n) ∨ (Function (n) ∧ String (n))
                   then  Macro (REJST)
             end
        else ERRORMESSAGE (347)
end  Procstat;
```

```
procedure  Procedure call (n); integer  n;
begin   integer  number of parameters;
        if Operator like (n)
        then Process operator (n)
        else begin number of parameters:= List length (n);
                   if number of parameters ≠ 0
                   then Parameter list (n, number of parameters)
                   else if Formal (n)
                        then Macro2 (DOS, n)
                        else if In library(n) then Macro2 (ISUBJ, n)
                                              else Macro2 (SUBJ, n)
             end
end  Procedure call;


integer procedure  Ordinal number (n); integer  n;
begin   Ordinal number:= if Formal (n) then 15
                         else
                         if Subscrvar (n)
                         then (if Arithmetic (n)
                               then (if Real (n) then 8 else 9)
                               else if Boolean (n)
                                    then 10 else 11)
                         else
                         if Function (n)
                         then (if  Arithmetic (n)
                               then (if  Real (n)  then 24 else 25)
                               else if Boolean (n) then 26 else 27)
                         else
                         if Proc (n) then 30
                         else
                         if Arithmetic(n)
                         then (if Real (n) then 0 else 1)
                         else if Boolean (n)
                              then 2
                              else if String (n) then 3 else 14
end  Ordinal number;
```

```
procedure  Parameter list (n, number of parameters);
integer  n, number of parameters;
begin integer  count, m, f, apd, type, future;
      Boolean  simple identifier;
      integer  array  descriptor list[1 : number of parameters];
      count:= future:= 0; f:= n;
      if last symbol = open
      then
      begin
      next: count:= count + 1; next symbol;
           Actual parameter (apd, simple identifier, type, future);
           if count ≤ number of parameters
           then
           begin descriptor list[count]:= apd;
                 if  ¬ Formal (n)
                 then
                 begin f:= Next formal identifier (f);
                       if simple identifier
                       then
                       begin if Subscrvar (f)
                             then
                             begin if Nonsubscrvar (type)
                                   then  ERRORMESSAGE (348);
                                   Check type (f, type);
                                   Check list length (f, type)
                             end
                             else
                             if Proc (f)
                             then
                             begin if Nonproc (type)
                                   then ERRORMESSAGE (349);
                                   Check list length (f, type);
                                   if Function (f)
                                   then begin if Nonfunction (type)
                                              then ERRORMESSAGE (350);
                                              Check type (f, type)
                                        end
                             end
                             else
                             if Simple (f)
                             then
                             begin if Nonsimple (type)
                                   then ERRORMESSAGE (351);
                                   Check type (f, type)
                             end
                       end
```

```
                    else
                    begin if Subscrvar.(f) ∨ Proc (f)
                          then ERRORMESSAGE (352);
                          if Assigned to (f) ∧ Nonassignable (apd)
                          then ERRORMESSAGE (353);
                          if Arithmetic(f) ∧
                             (type = bo ∨ type = st ∨ type = des)
                          then ERRORMESSAGE (354) else
                          if Boolean (f) ∧
                             type ≠ bo ∧ type ≠ nondes ∧ type ≠ un
                          then ERRORMESSAGE (355) else
                          if String (f) ∧
                             type ≠ st ∧ type ≠ nondes ∧ type ≠ un
                          then ERRORMESSAGE (356) else
                          if Designational (f) ∧
                             type ≠ des ∧ type ≠ un
                          then ERRORMESSAGE (357) else
                          if Arbost (f) ∧ type = des
                          then ERRORMESSAGE (358)
                    end
               end
          end
          else ERRORMESSAGE (359);
          if last symbol = comma then goto next;
          if last symbol = close
          then begin next symbol;
                     if count < number of parameters
                     then ERRORMESSAGE (360)
               end
          else ERRORMESSAGE (361)
     end
     else ERRORMESSAGE (362);
     if future ≠ 0 then  Substitute (future);
     if Formal (n) then  Macro2 (DOS, n) else  if In library (n)
                                               then Macro2 (ISUBJ, n)
                                               else Macro2 (SUBJ, n);
     m:= 0;
next apd: if m < count ∧ m < number of parameters
     then begin m:= m + 1; apd:= descriptor list[m];
                Macro2 (CODE, apd); goto next apd
          end
end  Parameter list;
```

```
procedure  Actual parameter (apd, simple identifier, type, future);
integer  apd, type, future; Boolean  simple identifier;
begin integer  n, begin address;
      begin address:= Order counter + (if future = 0 then 1 else 0);
      simple identifier:= false;
      if letter last symbol
      then
      begin n:= Identifier;
            if last symbol = comma ∨ last symbol = close
            then
            begin type:= n; simple identifier:= true;
                  if Proc (n) ∧ ¬ Formal (n)
                  then
                  begin if  future = 0 then Macro2 (JU, future);
                        Macro (TFD);
                        if In library (n) then Macro2 (IJU1, n)
                                          else Macro2 (JU1, n);
                        apd:= d20 × Ordinal number (n) + begin address
                  end
                  else if Subscrvar (n) ∧ Designational (n) ∧
                        ¬ Formal (n)
                       then begin if future = 0
                                  then Macro2 (JU, future);
                                  Macro2 (TSWE, n);
                                  apd:= 12 × d20 + begin address
                            end
                       else apd:= d20 × Ordinal number (n) +
                                  Address (n) +
                                  (if  Dynamic (n) then d18 else 0)
            end
            else
            begin Start implicit subroutine (future);
                  if Subscrvar (n) then Address description (n);
                  if (last symbol = comma ∨ last symbol = close) ∧
                     ( ¬ Designational (n))
                  then
                  begin if Unknown (n) then Macro (SAS);
                        Macro2 (EXITSV, - 2 × dimension);
                        apd:= d20 × (if Boolean (n) then 18 else
                                     if String (n)  then 19 else
                                     if Formal (n)  then 32 else
                                     if Real (n)    then 16 else 17)
                              + Order counter;
                        type:= if Arithmetic (n) then ar else
                               if Boolean (n)    then bo else
                               if String (n)     then st else
                               if Arbost (n)     then nondes else  un;
                        Macro2 (SUBJ, - begin address);
                        if Boolean (n) then  Macro (TASB)  else
                        if String (n)  then  Macro (TASST) else
                        if Formal (n)  then  Macro (TASU)  else
                        if Integer (n) then  Macro (TASI)
                                       else  Macro (TASR);
                        Macro (DECS); Macro2 (SUBJ, - begin address);
                        Macro (FAD)
                  end
```

```
                else
                begin if Subscrvar (n) then Evaluation of (n);
                      Exprest (type, n); Macro (EXITIS);
                      apd:= mask[type] + begin address
                end
        end
end
else
if digit last symbol
then begin Unsigned number;
           if (last symbol = comma ∨ last symbol = close) ∧
              ( ¬ in name list)
           then begin type:= ar; apd:= Number descriptor end
           else begin Start implicit subroutine (future);
                      Arithconstant;
                      if in name list ∧ ( ¬ operator last symbol)
                      then begin Macro2 (TLV, integer label);
                                 type:= intlab
                           end
                      else Rest of arboolexp (type);
                      Macro (EXITIS);
                      apd:= mask[type] + begin address
                end
     end
else
if last symbol = plus
then
begin next symbol;
      if digit last symbol
      then begin Unsigned number;
                 if last symbol = comma ∨ last symbol = close
                 then begin type:= ar; apd:= Number descriptor end
                 else begin Start implicit subroutine (future);
                            Arithconstant;
                            Rest of arboolexp (type);
                            Macro (EXITIS);
                            apd:= mask[type] + begin address
                      end
           end
      else begin Start implicit subroutine (future);
                 Arboolexp (type);
                 Macro (EXITIS); apd:= mask[type] + begin address
           end
end
```

```
      else
      if last symbol = minus
      then
      begin next symbol;
            if digit last symbol
            then begin Unsigned number;
                       if (last symbol = comma ∨ last symbol = close) ∧
                          small
                       then
                       begin type:= ar;
                             apd:= d20 × 13 + value of constant
                       end
                       else
                       begin Start implicit subroutine (future);
                             Arithconstant; Next primary; Next factor;
                             Macro (NEG); Rest of arboolexp (type);
                             Macro (EXITIS);
                             apd:= mask[type] + begin address
                       end
                 end
            else begin Start implicit subroutine (future);
                       Term; Macro (NEG);
                       Rest of arboolexp (type);
                       Macro (EXITIS); apd:= mask[type] + begin address
                 end
      end
      else
      if last symbol = true ∨ last symbol = false
      then
      begin type:= bo; n:= last symbol; next symbol;
            if last symbol = comma ∨ last symbol = close
            then apd:= d20 × 6 + (if n = true then 0 else 1)
            else begin Start implicit subroutine (future);
                       Macro2 (TBC, n);
                       Rest of boolexp;
                       Macro (EXITIS);
                       apd:= mask[type] + begin address
                 end
      end
      else begin Start implicit subroutine (future); Exp (type);
                 Macro (EXITIS); apd:= mask[type] + begin address
           end
end  Actual parameter;
```

```
procedure  Start implicit subroutine (future); integer  future;
begin   if future = 0 then  Macro2 (JU, future);
        Macro (ENTRIS)
end  Start implicit subroutine;


integer procedure  Number descriptor;
begin   Number descriptor:=
        if small then d20 × 7 + value of constant
                 else d20 × (if  real number then 4 else 5)
                      + address of constant
end  Number descriptor;


procedure  Process operator (n); integer  n;
begin   integer  count;
        count:= 0;
        if last symbol = open
        then begin
             next: next symbol; Arithexp; count:= count + 1;
                   if last symbol = comma
                   then begin Macro (STACK); goto next end;
                   if last symbol = close
                   then next symbol
                   else ERRORMESSAGE (361)
             end;
        if count ≠ List length (n) then ERRORMESSAGE (363);
        Macro (Operator macro (n))
end  Process operator;


Boolean procedure  Nonassignable (apd); integer  apd;
begin   integer  rank;
        rank:= apd : d20;
        Nonassignable:= (rank ≠ 15) ∧ (rank - rank : 16 × 16) > 3
end  Nonassignable;


procedure  Line;
begin   if lnc ≠ last lnc then Line1 end  Line;


procedure  Line1;
begin   if wanted then begin last lnc:= lnc; Macro2 (LNC, lnc) end
end  Line1;
```

```
procedure  Statement;
begin   ifstatement forbidden:= false; Stat end  Statement;


procedure  Unconditional statement;
begin   ifstatement forbidden:= true; Stat end  Unconditional statement;


procedure  Stat;
begin   integer  n, save lnc;
        if letter last symbol
        then begin save lnc:= line counter;
                   n:= Identifier;
                   if Designational (n)
                   then begin Label declaration (n); Stat end
                   else begin lnc:= save lnc; Line;
                              if Subscrvar (n) ∨ last symbol = colonequal
                              then Assignstat (n)
                              else Procstat (n)
                        end
             end
        else
        if digit last symbol
        then begin Unsigned number;
                   if in name list
                   then begin Label declaration (integer label); Stat end
                   else ERRORMESSAGE (364)
             end
        else begin if last symbol = goto
                   then begin lnc:= line counter; Line; Gotostat end
                   else
                   if last symbol = begin
                   then begin save lnc:= line counter; next symbol;
                              if declarator last symbol
                              then begin lnc:= save lnc; Line; Block end
                              else Compound tail;
                              next symbol
                        end
                   else
                   if last symbol = if
                   then begin if ifstatement forbidden
                              then ERRORMESSAGE (365);
                              lnc:= line counter; Line; Ifstat
                        end
                   else
                   if last symbol = for
                   then begin lnc:= line counter; Line; Forstat;
                              if last symbol = else
                              then ERRORMESSAGE (366)
                        end
             end
end  Stat;
```

```
procedure  Gotostat;
begin   integer  n;
        next symbol;
        if letter last symbol
        then begin n:= Identifier; Subscripted variable (n);
                   if local label (n)
                   then begin Test forcount (n); Macro2 (JU, n) end
                   else begin Designame (n); Macro (JUA) end
             end
        else begin Desigexp; Macro (JUA) end
end  Gotostat;


procedure  Compound tail;
begin   Statement;
        if last symbol ≠ semicolon ∧ last symbol ≠ end
        then begin ERRORMESSAGE (367);
                   skip rest of statement (Statement)
             end;
        if last symbol = semicolon
        then begin next symbol; Compound tail end
end  Compound tail;


procedure  Ifstat;
begin   integer  future1, future2, save lnc, last lnc1;
        future1:= future2:= 0; save lnc:= line counter;
        next symbol; Boolexp; Macro2 (COJU, future1);
        if last symbol = then then next symbol else ERRORMESSAGE (368);
        Unconditional statement;
        if last symbol = else
        then begin Macro2 (JU, future2); Substitute (future1);
                   last lnc1:= last lnc; last lnc:= save lnc;
                   next symbol; Statement; Substitute (future2);
                   if last lnc > last lnc1 then last lnc:= last lnc1
             end
        else begin Substitute (future1);
                   if last lnc > save lnc then last lnc:= save lnc
             end
end  Ifstat;


procedure  Forstat;
begin   integer  future, save lnc;
        save lnc:= line counter;
        10:= 0; next symbol; For list;
        future:= 0; Macro2 (JU, future); if 10 ≠ 0 then Substitute(10);
        if last symbol = do then next symbol else ERRORMESSAGE (369);
        Increase status (increment); forcount:= forcount + 1;
        Statement;
        Increase status (- increment); forcount:= forcount - 1;
        if last lnc < 0 ∨ lnc ≠ save lnc
        then begin lnc:= save lnc; Line1 end;
        Macro2 (IJU, status); Substitute (future)
end  Forstat;
```

```
procedure  Store preparation;
begin   if  Subscrvar (controlled variable) then Macro2 (SUBJ, - 12)
        else
        if  Formal (controlled variable)
        then Macro2 (DOS2, controlled variable)
end  Store preparation;


procedure  Store macro;
begin   if  Subscrvar (controlled variable)
        then begin if Formal (controlled variable)  then Macro (STFSU)
                   else
                   if Integer (controlled variable) then Macro (STSI)
                                                    else Macro (STSR);
                   Macro2 (DECB, 2)
             end
        else if Formal (controlled variable)
             then Macro2 (DOS3, controlled variable)
             else if Integer (controlled variable)
                  then Macro2 (STI, controlled variable)
                  else Macro2 (STR, controlled variable)
end  Store macro;


procedure  Take macro;
begin   if  Subscrvar (controlled variable)
        then Macro2 (SUBJ, - 11)
        else Arithname (controlled variable)
end  Take macro;


procedure  For list;
begin if letter last symbol
      then
      begin controlled variable:= Identifier;
            if Nonarithmetic (controlled variable)
            then ERRORMESSAGE (370);
            if Subscrvar (controlled variable)
            then
            begin 13:= 0; Macro2 (JU, 13);
                  14:= Order counter;
                  Address description (controlled variable);
                  Macro2 (EXITSV, 1 - 2 × dimension);
                  11:= Order counter;
                  Macro2 (SUBJ, - 14);
                  if Formal (controlled variable)  then Macro (TSCVU)
                  else
                  if Integer (controlled variable) then Macro (TISCV)
                                                   else Macro (TRSCV);
                  12:= Order counter;
                  Macro2 (SUBJ, - 14); Macro (FADCV);
                  Substitute (13)
            end
            else if Function (controlled variable)
                 then ERRORMESSAGE (371);
            if last symbol ≠ colonequal then ERRORMESSAGE (372);
```

```
list: 13:= Order counter;
      Macro2 (TSIC, 0); Macro2 (SSTI, status);
      14:= Order counter;
      Store preparation;
      next symbol; Arithexp;
      if last symbol = comma ∨ last symbol = do
      then begin Store macro; Macro2 (JU, 10);
                 Substitute (13)
           end
      else
      if last symbol = while
      then begin Store macro;
                 next symbol; Boolexp;
                 Macro2 (YCOJU, 10); Subst2 (14, 13)
           end
      else
      if last symbol = step
      then begin 15:= 0; Macro2 (JU, 15); 14:= Order counter;
                 next symbol; complicated:= false; Arithexp;
                 complex step element:=
                 complicated ∨ Order counter > 14 + 1;
                 if complex step element then  Macro (EXIT);
                 Substitute (13);
                 Store preparation; Take macro; Macro (STACK);
                 if complex step element then  Macro2 (SUBJ, - 14)
                                       else  Macro2 (DO, 14);
                 Macro (ADD);
                 Substitute (15);
                 Store macro;
                 if Subscrvar (controlled variable) ∨
                    Formal (controlled variable)
                 then Take macro;
                 Macro (STACK);
                 if last symbol = until
                 then begin next symbol; Arithexp end
                 else ERRORMESSAGE (373);
                 Macro (TEST1);
                 if complex step element then Macro2 (SUBJ, - 14)
                                       else Macro2 (DO, 14);
                 Macro (TEST2); Macro2 (YCOJU, 10)
           end
      else ERRORMESSAGE (374);
      if last symbol = comma then goto list
 end
 else ERRORMESSAGE (375)
end  For list;
```

```
procedure  Switch declaration;
begin integer  m;
      next symbol;
      if letter last symbol
      then
      begin switch identifier:= Identifier;
            number of switch elements:= List length (switch identifier);
            if last symbol = colonequal
            then
            begin integer array
                  sword list[1 : number of switch elements];
                  switch list count:= 0; in switch declaration:= true;
            next: switch list count:= switch list count + 1;
                  next symbol;
                  if letter last symbol
                  then
                  begin m:= Identifier;
                        if Nondesignational (m) then ERRORMESSAGE (376);
                        if Subscrvar (m)
                        then
                        begin sword:= - 45613055 + Order counter;
                              Subscripted variable (m); Macro (EXIT)
                        end
                        else
                        sword:= (if Formal (m)
                                 then -33685503
                                 else 4718592 + (if Dynamic (m)
                                                 then function digit
                                                 else 0)) +
                                    Address (m)
                  end
                  else
                  if digit last symbol
                  then
                  begin Unsigned number;
                        if  in name list
                        then  sword:= 4718592 +
                                      (if Dynamic (integer label)
                                       then function digit
                                       else 0) +
                                      Address (integer label)
                        else ERRORMESSAGE (377)
                  end
                  else
                  begin sword:= - 45613055 + Order counter;
                        Desigexp; Macro (EXIT)
                  end;
```

```
                if switch list count > number of switch elements
                then ERRORMESSAGE (378);
                sword list[switch list count]:= sword;
                if last symbol = comma then goto next;
                if switch list count < number of switch elements
                then ERRORMESSAGE (379);
                Mark position in name list (switch identifier);
                in switch declaration:= false;
                Macro2 (CODE, number of switch elements);
                m:= 0;
          next sword: if m < switch list count ∧
                         m < number of switch elements
                then begin m:= m + 1; sword:= sword list[m];
                           Macro2 (CODE, sword); goto next sword
                     end
          end
          else ERRORMESSAGE (380)
     end
     else ERRORMESSAGE (381)
end  Switch declaration;


procedure  Array declaration;
begin   integer  n, count;
        next symbol; lnc:= line counter; Line;
        n:= Identifier; dimension:= List length (n); count:= 1;
next:   if last symbol = comma then begin next symbol; Identifier;
                                           count:= count + 1; goto next
                                    end;
        if  last symbol = sub  then begin in array declaration:= true;
                                          Bound pair list;
                                          in array declaration:= false
                                    end
                               else ERRORMESSAGE (382);
        Macro2 (TNA, count); Macro2 (TDA, dimension);
        Macro2 (TAA, n); Macro (arr decla macro);
        if last symbol = comma then Array declaration
end  Array declaration;


procedure  Bound pair list;
begin   next symbol; Arithexp; Macro (STACK);
        if last symbol = colon then begin next symbol; Arithexp;
                                          Macro (STACK)
                                    end
                               else ERRORMESSAGE (383);
        if last symbol = comma then Bound pair list
                               else if last symbol = bus
                                    then next symbol
                                    else ERRORMESSAGE (384)
end  Bound pair list;
```

```
procedure  Procedure declaration;
begin integer  n, f, count, save lnc;
      next symbol; f:= n:= Identifier;
      Skip parameter list; skip value list; skip specification list;
      if ¬ In library (n) then Mark position in name list (n);
      if in code (n)
      then Translate code
      else begin if Function (n) then Set inside declaration (n, true);
                 entrance block;
                 Macro2 (DPTR, display level);
                 Macro2 (INCRB, top of display);
                 for  count:= List length (n) step  - 1 until  1 do
                 begin f:= Next formal identifier (f);
                       if In value list (f)
                       then
                       begin if Subscrvar (f)
                             then Macro (CEN)
                             else
                             begin if Arithmetic (f)
                                   then begin if Integer (f)
                                              then Macro (CIV)
                                              else Macro (CRV)
                                        end
                                   else if Boolean (f) then Macro (CBV)
                                   else if String (f)  then Macro (CSTV)
                                                       else Macro (CLV)
                             end
                       end
                       else if Assigned to (f) then Macro (CLPN)
                                               else Macro (CEN)
                 end;
                 Macro2 (TDL, display level);
                 Macro2 (ENTRPB, local space);
                 Label list; f:= n;
                 for  count:= List length (n) step  - 1 until  1 do
                 begin f:= Next formal identifier (f);
                       if In value list (f) ∧ Subscrvar (f)
                       then begin Macro2 (TAA, f);
                                  if Integer (f) then Macro (TIAV)
                                                 else Macro (TAV)
                            end
                 end;
```

```
                    save lnc:= last lnc; last lnc:= - line counter;
                    Save and restore lnc (SLNC, n);
                    if last symbol = begin
                    then begin next symbol; if declarator last symbol
                                                then Declaration list;
                               Compound tail; next symbol
                         end
                    else Statement;
                    lnc:= last lnc:= save lnc;
                    if Function (n)
                    then begin Set inside declaration (n, false);
                               f:= Local position (n);
                               if Arithmetic (f) then Arithname (f) else
                               if Boolean (f)    then Boolname (f)
                               else begin Stringname (f); Macro (LOS) end
                         end;
                    Save and restore lnc (RLNC, n);
                    if use of counter stack  then Macro (EXITPC)
                                             else Macro (EXITP);
                    exit block
               end
end  Procedure declaration;


procedure Save and restore lnc (macro, n); integer  macro, n;
begin  if wanted ∧ Function (n) then Macro2 (macro, Local position1 (n))
end  Save and restore lnc;


procedure  Block;
begin    entrance block;
         Macro2 (TBL, display level); Macro2 (ENTRB, local space);
         Label list; Declaration list; Compound tail;
         if use of counter stack then Macro2 (EXITC, display level)
                                 else Macro2 (EXITB, display level);
         exit block
end  Block;
```

```
procedure  Declaration list;
begin    integer  future, arr dec;
         future:= arr dec:= 0;
next:    if type declarator last symbol then skip type declaration
         else
         if arr declarator last symbol
         then begin  if future ≠ 0
                     then begin Substitute (future);
                                future:= 0
                          end;
                     arr dec:= 1; Array declaration
              end
         else
         begin if future = 0 then Macro2 (JU, future);
               if last symbol = switch then Switch declaration
                                       else Procedure declaration
         end;
         if last symbol = semicolon then next symbol
                                    else ERRORMESSAGE (385);
         if declarator last symbol  then goto next;
         if future ≠ 0  then Substitute (future);
         if arr dec ≠ 0 then Macro2 (SWP, display level)
end  Declaration list;


procedure  Label list;
begin    integer  n, count;
         count:= Number of local labels;
         if count > 0
         then begin Macro2 (DECB, 2 × count);
                    Macro2 (LAD, display level);
                    n:= 0; for count:= count step - 1 until 1 do
                    begin next: n:= Next local label (n);
                                if Super local (n) then goto next;
                                if count = 1 then Macro2 (LAST, n)
                                             else Macro2 (NIL, n)
                    end
              end
end  Label list;
```

```
procedure  Program;
begin   integer  n;
        if letter last symbol
        then begin n:= Identifier;
                   if last symbol = colon
                   then Label declaration (n);
                   Program
             end
        else
        if digit last symbol
        then begin Unsigned number;
                   if in name list ∧ last symbol = colon
                   then Label declaration (integer label);
                   Program
             end
        else
        if last symbol = begin
        then begin next symbol;
                   if declarator last symbol then Block
                                             else Compound tail;
                   Macro (END)
             end
        else begin next symbol; Program end
end  Program;


procedure  Label declaration (n); integer  n;
begin   last lnc:= - line counter;
        if Subscrvar (n)       then begin ERRORMESSAGE (388);
                                          Subscripted variable (n)
                                   end
                               else Mark position in name list (n);
        if last symbol = colon then next symbol else  ERRORMESSAGE (389)
end  Label declaration;


procedure  Substitute (address); integer  address;
begin   Subst2 (Order counter, address) end  Substitute;


procedure  Subst2 (address1, address2);
value  address1, address2; integer  address1, address2;
begin   integer  instruction, instruct part, address part;
        address2:= abs (address2);
        instruction:= space[prog base + address2];
        instruct part:= instruction : d15 × d15 -
                        (if  instruction < 0 then 32767 else 0);
        address part:= instruction - instruct part;
        space[prog base + address2]:= instruct part + address1;
        if address part = 0
        then begin if instruct part = end of list
                   then space[prog base + address2]:=
                        - space[prog base + address2]
             end
        else Subst2 (address1, address part)
end  Subst2;
```

```
integer procedure  Order counter;
begin   Macro (EMPTY); Order counter:= instruct counter
end  Order counter;


procedure  Macro (macro number); integer  macro number;
begin   Macro2 (macro number, parameter) end  Macro;


procedure  Macro2 (macro number, metaparameter);
integer  macro number, metaparameter;
begin   macro:= if macro number < 512 then macro list[macro number]
                                      else macro number;
        parameter:= metaparameter;
        if state = 0
        then begin if macro = STACK then state:= 1
                   else
                   if Simple arithmetic take macro then Load (3)
                   else
                   Produce (macro, parameter)
             end
        else
        if state = 1
        then begin Load (2);
                   if ¬ Simple arithmetic take macro
                   then begin Produce (STACK, parameter); Unload end
             end
        else
        if state = 2
        then begin if Optimizable operator then Optimize
                   else
                   begin Produce (STACK, parameter); state:= 3;
                         Macro2 (macro, parameter)
                   end
             end
        else
        if state = 3
        then begin if macro = NEG then Optimize
                   else
                   begin Unload; Macro2 (macro, parameter) end
             end;
        if Forward jumping macro ∧ metaparameter ≤ 0
        then Assign (metaparameter)
end  Macro2;
```

```
procedure  Load (state i); integer  state i;
begin   stack0:= macro; stack1:= parameter; state:= state i end  Load;


procedure  Unload;
begin   Produce (stack0, stack1); state:= 0 end  Unload;


procedure  Optimize;
begin   stack0:= tabel[5 × Opt number (macro) + Opt number (stack0)];
        Unload
end  Optimize;


procedure  Assign (metaparameter); integer  metaparameter;
begin   metaparameter:= - (instruct counter - 1) end  Assign;


procedure  Produce (macro, parameter); integer  macro, parameter;
begin   integer  number, par number, entry, count;
        if macro = EMPTY then
        else
        if macro = CODE
        then begin space[prog base + instruct counter]:= parameter;
                   instruct counter:= instruct counter + 1;
                   test pointers
             end
        else begin number:= Instruct number (macro);
                   par number:= Par part (macro);
                   entry:= Instruct part (macro) - 1;
                   if par number > 0
                   then Process parameter (macro, parameter);
                   Process stack pointer (macro);
                   for count:= 1 step 1 until number do
                   Produce (CODE, instruct list[entry + count] +
                                  (if  count = par number
                                   then parameter else 0))
             end
end  Produce;
```

```
procedure  Process stack pointer (macro); integer  macro;
begin   if ⅂ in code body
        then
        begin integer  reaction;
              reaction:= B reaction (macro);
              if reaction < 9
              then begin b:= b + reaction - 4;
                         if b > max depth then max depth:= b
                   end
              else
              if reaction = 10 then b:= 0
              else
              if reaction = 11 then b:= b - 2 × (dimension - 1)
              else
              if reaction = 12
              then begin if ecount = 0
                         then
                         begin ret level:= b;
                               ret max depth:= max depth;
                               b:= 0; max depth:= max depth isr
                         end;
                         ecount:= ecount + 1
                   end
              else
              if reaction = 13
              then begin if macro = EXITSV
                         then
                         begin if b > max depth isr
                               then max depth isr:= b;
                               b:= b - 2 × (dimension - 1)
                         end;
                         if ecount = 1
                         then
                         begin if max depth > max depth isr
                               then max depth isr:= max depth;
                               b:= ret level;
                               max depth:= ret max depth
                         end;
                         if ecount > 0 then ecount:= ecount - 1
                   end
              else
              if reaction = 14
              then begin b:= display level + top of display;
                         if b > max display length
                         then max display length:= b;
                         ret max depth:= max depth
                   end
              else
              if reaction = 15
              then begin if b > max proc level
                         then max proc level:= b;
                         b:= 0; max depth:= ret max depth
                   end
        end
end  Process stack pointer;
```

```
procedure  Process parameter (macro, parameter);
integer  macro, parameter;
begin   if Value like (macro)
        then
        begin if macro = TBC
              then parameter:= if parameter = true then 0 else 1
              else
              if macro = SWP then parameter:= d9 × parameter
              else
              if macro ≠ EXITSV then parameter:= abs (parameter)
        end
        else
        begin if macro = JU ∨ macro = SUBJ ∨ macro = NIL ∨ macro = LAST
              then begin if parameter ≤ 0
                         then parameter:= - parameter
                         else parameter:= Program address (parameter)
                   end
              else parameter:= Address (parameter) +
                               (if Dynamic (parameter)
                                then (if macro = TLV ∨ macro = TAA
                                      then function digit
                                      else if macro = STST
                                           then function letter
                                           else c variant)
                                else 0)
        end
end  Process parameter;


Boolean procedure  Simple arithmetic take macro;
begin   Simple arithmetic take macro:= bit string (d1, d0, macro) = 1
end  Simple arithmetic take macro;


Boolean procedure  Optimizable operator;
begin   Optimizable operator:= bit string (d2, d1, macro) = 1
end  Optimizable operator;

Boolean procedure  Forward jumping macro;
begin   Forward jumping macro:= bit string (d3, d2, macro) = 1
end  Forward jumping macro;


Boolean procedure  Value like (macro); integer  macro;
begin   Value like:= bit string (d4, d3, macro) = 1 end  Value like;


integer procedure  Opt number (macro); integer  macro;
begin   Opt number:= bit string (d8, d4, macro) end  Opt number;


integer procedure  Instruct number (macro); integer  macro;
begin   Instruct number:= bit string (d10, d8, macro)
end  Instruct number;
```

```
integer procedure  Par part (macro); integer  macro;
begin   Par part:= bit string (d12, d10, macro) end  Par part;

integer procedure  Instruct part (macro); integer  macro;
begin   Instruct part:= bit string (d21, d12, macro) end  Instruct part;

integer procedure  B reaction (macro); integer  macro;
begin   B reaction:= macro : d21 end  B reaction;

integer procedure  Code bits (n); integer  n;
begin   Code bits:= space[nl base - n] : d19 end  Code bits;

integer procedure  Character (n); integer  n;
begin   Character:= bit string (d24, d19, space[nl base - n])
end  Character;

Boolean procedure  Arithmetic (n); integer  n;
begin   integer  i;
        i:= type bits (n);
        Arithmetic:= Character (n) ≠ 24 ∧ (i < 2 ∨ i = 4)
end  Arithmetic;

Boolean procedure  Real (n); integer  n;
begin   Real:= Character (n) ≠ 24 ∧ type bits (n) = 0 end  Real;

Boolean procedure  Integer (n); integer  n;
begin   Integer:= type bits (n) = 1 end  Integer;

Boolean procedure  Boolean (n); integer  n;
begin   Boolean:= type bits (n) = 2 end  Boolean;

Boolean procedure  String (n); integer  n;
begin   String:= type bits (n) = 3 end  String;

Boolean procedure  Designational (n); integer  n;
begin   Designational:= type bits (n) = 6 end  Designational;

Boolean procedure  Arbost (n); integer  n;
begin   Arbost:= Character (n) ≠ 24 ∧ type bits (n) < 6 end  Arbost;

Boolean procedure  Unknown (n); integer  n;
begin   Unknown:= type bits (n) = 7 end  Unknown;
```

```
Boolean procedure  Nonarithmetic (n); integer  n;
begin   integer  i;
        i:= type bits (n);
        Nonarithmetic:= Character (n) = 24 ∨ i = 2 ∨ i = 3 ∨ i = 6
end  Nonarithmetic;


Boolean procedure  Nonreal (n); integer  n;
begin   Nonreal:= Nonarithmetic (n) ∨ type bits (n) = 1 end  Nonreal;


Boolean procedure  Noninteger (n); integer  n;
begin   Noninteger:= Nonarithmetic (n) ∨ type bits (n) = 0
end  Noninteger;


Boolean procedure  Nonboolean (n); integer  n;
begin   integer  i;
        i:= type bits (n); Nonboolean:= i ≠ 2 ∧ i ≠ 5 ∧ i ≠ 7
end  Nonboolean;


Boolean procedure  Nonstring (n); integer  n;
begin   integer  i;
        i:= type bits (n); Nonstring:= i ≠ 3 ∧ i ≠ 5 ∧ i ≠ 7
end  Nonstring;


Boolean procedure  Nondesignational (n); integer  n;
begin   Nondesignational:= type bits (n) < 6 end  Nondesignational;


Boolean procedure  Nontype (n); integer  n;
begin   Nontype:= type bits (n) = 6 ∨ (Proc (n) ∧ Nonfunction (n))
end  Nontype;


Boolean procedure  Simple (n); integer  n;
begin   Simple:= Code bits (n) = 127 ∨ Simple1 (n) end  Simple;


Boolean procedure  Simple1 (n); integer  n;
begin   Simple1:= Character (n) : d3 = 0 end  Simple1;


Boolean procedure  Subscrvar (n); integer  n;
begin   Subscrvar:= Character (n) : d3 = 1 end  Subscrvar;

Boolean procedure  Proc (n); integer  n;
begin   Proc:= Character (n) : d3 > 1 ∧ Code bits (n) ≠ 127 end  Proc;


Boolean procedure  Function (n); integer  n;
begin   Function:= Character (n) : d3 = 2 end  Function;
```

```
Boolean procedure  Nonsimple (n); integer  n;
begin  Nonsimple:= ¬ (Simple (n) ∨ (if Proc (n)
                                    then (Formal (n) ∨ Function (n)) ∧
                                          List length (n) < 1
                                    else false ))
end  Nonsimple;


Boolean procedure  Nonsubscrvar (n); integer  n;
begin  Nonsubscrvar:= Simple1 (n) ∨ Proc (n) end  Nonsubscrvar;


Boolean procedure  Nonproc (n); integer  n;
begin  Nonproc:= ¬ (Character (n) : d3 > 2 ∨
                    (Formal (n) ∧ Simple1 (n) ∧ ¬ Assigned to (n)))
end  Nonproc;


Boolean procedure  Nonfunction (n);  integer  n;
begin  Nonfunction:= ¬ (Function (n) ∨ Formal (n)) end  Nonfunction;


Boolean procedure  Formal (n); integer  n;
begin  Formal:= Code bits (n) > 95 end  Formal;


Boolean procedure  In value list (n); integer  n;
begin  In value list:= Code bits (n) > 63 ∧ ¬ Formal (n)
end  In value list;


Boolean procedure  Assigned to (n); integer  n;
begin  Assigned to:= bit string (d19, d18, space[nl base - n]) = 1
end  Assigned to;


Boolean procedure  Dynamic (n); integer  n;
begin  Dynamic:= Code bits (n) > 63 ∨ Assigned to (n) end  Dynamic;


Boolean procedure  In library (n); integer  n;
begin  In library:= space[nl base - n - 1] ≥ d25 end  In library;


Boolean procedure  Id1 (k, n); integer  k, n;
begin  Id1:= bit string (2 × k, k, space[nl base - n - 1]) = 1 end  Id1;


Boolean procedure  Operator like (n); integer  n;
begin  Operator like:= Id1 (d23, n) end  Operator like;


Boolean procedure  Outside declaration (n); integer  n;
begin  Outside declaration:= Id1 (d22, n) end  Outside declaration;
```

```
Boolean procedure  Ass to function designator (n); integer  n;
begin   Ass to function designator:= Id1 (d21, n)
end Ass to function designator;


Boolean procedure  Declared (n); integer  n;
begin   Declared:= Id1 (d19, n) end  Declared;


Boolean procedure  Super local (n); integer  n;
begin   Super local:= Id1 (d18, n) end  Super local;


procedure  Change (k, n); integer  k, n;
begin   integer  i, j;
        i:= space[nl base - n - 1]; j:= i - i : (2 × k) × (2 × k);
        space[nl base - n - 1]:= i + (if  j < k then  k else  - k)
end  Change;


integer procedure  Local position (n); integer  n;
begin   if  ¬ Ass to function designator (n) then  Change (d21, n);
        Local position:= Local position1 (n)
end  Local position;


integer procedure  Local position1 (n); integer  n;
begin   Local position1:= n + 2 end  Local position1;


procedure  Set inside declaration (n, bool); integer  n; Boolean  bool;
begin   Change (d22, n);
        if  ¬ (bool ∨ Ass to function designator (n))
        then  ERRORMESSAGE (390)
end  Set inside declaration;
```

```
procedure  Mark position in name list (n); integer  n;
begin   integer  address;
        if Declared (n)
        then ERRORMESSAGE (391)
        else begin address:= Program address (n);
                   if address ≠ 0 then Substitute (address);
                   Change (d19, n)
             end
end  Mark position in name list;


integer procedure  Program address (n); integer  n;
begin   integer  word, head, m;
        m:= if Code bits (n) = 6 then n + 1 else n;
        word:= space[nl base - m]; head:= word : d18 × d18;
        if ¬ Declared (n)
        then space[nl base - m]:= head + Order counter;
        Program address:= word - head
end  Program address;


integer procedure  Address (n); integer  n;
begin   integer  word, tail, level;
        word:= Code bits (n);
        if word > 13 ∧ word < 25
        then tail:= Program address (n)
        else begin word:= space[nl base - n];
                   tail:= word - word : d18 × d18;
                   if Dynamic (n)
                   then begin level:= tail : d9;
                              if level = proc level ∧
                                 ¬ in switch declaration
                              then tail:= tail + d9 × (63 - level)
                        end
             end;
        Address:= tail
end  Address;


integer procedure  List length (n); integer  n;
begin   List length:= bit string (d18, d0, space[nl base - n - 1]) - 1
end  List length;


procedure  Test forcount (n); integer  n;
begin   if space[nl base - n - 1] : d20 > for count
        then ERRORMESSAGE (392)
end  Test forcount;
```

```
procedure  Check dimension (n); integer  n;
begin   integer  i;
        i:= if Code bits (n) = 14 then 1 else List length (n);
        if i ≥ 0 ∧ i ≠ dimension then ERRORMESSAGE (393)
end  Check dimension;


procedure  Check list length (f, n); integer  f, n;
begin   integer  i, j;
        i:= List length (f);
        j:= if Code bits (n) = 14 then 1 else List length (n);
        if i ≥ 0 ∧ j > 0 ∧ i ≠ j then ERRORMESSAGE (394)
end  Check list length;


procedure  Check type (f, n); integer  f, n;
begin   if (Designational (f) ∧ Nondesignational (n)) ∨
           (Arbost (f)        ∧ Nontype (n))          ∨
           (Arithmetic (f)    ∧ Nonarithmetic (n))    ∨
           (Boolean (f)       ∧ Nonboolean (n))       ∨
           (String (f)        ∧ Nonstring (n))
        then ERRORMESSAGE (395)
end  Check type;


integer procedure  Number of local labels;
begin   Number of local labels:=
          bit string (d13, d0, space[nl base - block cell pointer - 3])
end  Number of local labels;


integer procedure  Next local label (n); integer  n;
begin   Next local label:=
          if n = 0 then space[nl base - block cell pointer - 3] : d13
                   else next identifier (n)
end  Next local label;


integer procedure  Next formal identifier (n); integer  n;
begin   Next formal identifier:=
          next identifier (n + (if Formal (n) ∨ In library (n) ∨
                                   In value list (n)
                                then 2
                                else if Function (n) then 9 else 8))
end  Next formal identifier;


procedure  Increase status (increment); integer  increment;
begin   space[nl base - block cell pointer - 2]:=
        space[nl base - block cell pointer - 2] + increment
end  Increase status;


integer procedure  Identifier;
begin   read identifier; Identifier:= look up end  Identifier;
```

```
procedure  Skip parameter list;
begin   if last symbol = open
        then begin next symbol; skip type declaration;
                   if last symbol = close then next symbol
             end;
        if last symbol = semicolon then next symbol
end  Skip parameter list;


procedure  Translate code;
begin   integer  macro, parameter;
        if last symbol = quote
        then begin in code body:= true;
             next: next symbol;
                   if digit last symbol
                   then
                   begin macro:= unsigned integer (0);
                         if macro < 512 then macro:= macro list[macro];
                         if Par part (macro) > 0
                         then
                         begin if last symbol = comma
                               then next symbol
                               else ERRORMESSAGE (396);
                               if letter last symbol
                               then parameter:= Identifier
                               else
                               if digit last symbol
                               then parameter:= unsigned integer (0)
                               else
                               if last symbol = minus
                               then
                               begin next symbol;
                                     if digit last symbol
                                     then parameter:=
                                          - unsigned integer (0)
                                     else ERRORMESSAGE (397)
                               end
                               else ERRORMESSAGE (398);
                               Macro2 (macro, parameter)
                         end
                         else Macro (macro)
                   end
                   else ERRORMESSAGE (399);
                   if last symbol = comma then goto next;
                   if last symbol = unquote then next symbol
                                            else ERRORMESSAGE (400);
                   in code body:= false
             end
        else ERRORMESSAGE (401);
        entrance block; exit block
end  Translate code;
```

```
procedure  Unsigned number;
begin   integer  p;
        unsigned number;
        if ⌝ small
        then begin p:= 0;
             next: if p = dp0 then goto found;
                   if space[prog base + p] ≠ value of constant ∨
                      space[prog base + p + 1] ≠ decimal exponent
                   then begin p:= p + 2; goto next end;
             found: address of constant:= p
             end
end  Unsigned number;


procedure  Arithconstant;
begin   if small then Macro2 (TSIC, value of constant)
        else
        if real number then Macro2 (TRC, address of constant)
                        else Macro2 (TIC, address of constant)
end  Arithconstant;


integer procedure  Operator macro (n); integer  n;
begin   Operator macro:= space[nl base − n − 2] end  Operator macro;


procedure  Constant string;
begin   integer  word, count;
        quote counter:= 1;
next0:  word:= count:= 0;
next1:  next symbol;
        if last symbol ≠ unquote
        then begin word:= d8 × word + last symbol;
                   count:= count + 1;
                   if count = 3
                   then begin Macro2 (CODE, word); goto next0 end;
                   goto next1
             end;
next2:  word:= d8 × word + 255; count:= count + 1;
        if count < 3 then goto next2;
        Macro2 (CODE, word); quote counter:= 0; next symbol
end  Constant string;


integer procedure  Relatmacro;
begin   Relatmacro:=  if  last symbol = les then  LES else
                      if  last symbol = mst then  MST else
                      if  last symbol = mor then  MOR else
                      if  last symbol = lst then  LST else
                      if  last symbol = equ then  EQU else  UQU
end  Relatmacro;
```

```
main program of translate scan:
        if ¬ text in memory
        then begin NEWPAGE;
                   PRINTTEXT (⟨input tape for translate scan⟩)
             end;
        start:= instruct counter; last nlp:= nlp;
        runnumber:= 300; init; increment:= d13;
        state:= b:= max depth:= max depth isr:=
        max display length:= max proc level:= ecount:= 0;
        in switch declaration:= in code body:= false;
        next block cell pointer:= 0;
        entrance block; next symbol;
        Program;
        sum of maxima:= max depth + max depth isr +
                        max display length + max proc level;
        Macro2 (CODE, sum of maxima);
        output
end  translate;
```

```
procedure  output;
begin   integer  i, k, apostrophe, instruct number, par, address;

        procedure  pucar (n); integer  n;
        begin  integer  i;
               for i:= 1 step 1 until n do PUNLCR
        end  pucar;

        procedure  tabspace (n); integer  n;
        begin  integer  i, k;
               k:= n : 8;
               for i:= 1 step 1 until k do PUSYM (118);
               PUSPACE (n - k × 8)
        end  tabspace;

        procedure  absfixp (k); integer  k;
        begin  ABSFIXP (4, 0, k); pucar (2) end  absfixp;

        procedure  punch (bool); Boolean  bool;
        begin  if bool then PUTEXT (‹ true›)
                       else PUTEXT (‹false›);
               pucar (2)
        end  punch;

        procedure  punch octal (n);  value  n;  integer  n;
        begin  integer  i, k;
               Boolean  minussign;
               minussign:= n < 0; n:= abs (n);
               PUSYM (if minussign then minus else plus);
               PUSYM (apostrophe);
               for i:= d24, d21, d18, d15, d12, d9, d6, d3, d0 do
               begin k:= n : i; n:= n - k × i; PUSYM (k) end;
               PUSYM (apostrophe)
        end  punch octal;

        apostrophe:= 120;
        PUNLCR;
        if runnumber = 100
        then
        begin tabspace (22); PUTEXT (‹prescan0›); pucar (2);
              PUTEXT (‹erroneous›); PUSPACE (14);
              punch (erroneous); PUTEXT (‹text length›);
              PUSPACE (12);
              absfixp (if text in memory then text pointer + 1 else 0);
              PUTEXT (‹namelist›); pucar (2);
              for i:= 0 step 1 until nlp - 1 do
              begin tabspace (7); ABSFIXP (4, 0, i); PUSPACE (5);
                    punch octal (space[nl base - i]); PUNLCR
              end;
              STOPCODE;
              PUNLCR; PUTEXT (‹dp0›); pucar (2);
              PUTEXT (‹start›); pucar (2);
              PUTEXT (‹program›); pucar (2);
```

```
            for i:= prog base step 1 until instruct counter - 1 do
            begin tabspace (7); ABSFIXP (4, 0, i);
                  FIXP (16, 0, space[i]); PUNLCR
            end;
            RUNOUT; STOPCODE
      end
      else if runnumber = 200
      then
      begin tabspace (38); PUTEXT (‡prescan1‡); pucar (2);
            tabspace (39); punch (erroneous); tabspace (39);
            absfixp (if text in memory then text pointer + 1 else 0);
            pucar (2);
            for i:= 0 step 1 until nlp - 1 do
            begin tabspace (34); punch octal (space[nl base - i]);
                  PUNLCR
            end;
            STOPCODE; pucar (7);
            for i:= prog base step 1 until instruct counter - 1 do
            begin tabspace (32); FIXP (13, 0, space[i]); PUNLCR end;
            RUNOUT; STOPCODE
      end
      else
      begin tabspace (54); PUTEXT (‡translate‡); pucar (2);
            tabspace (55); punch (erroneous); tabspace (55);
            absfixp (if text in memory then text pointer + 1 else 0);
            pucar (2);
            for i:= 0 step 1 until nlp - 1 do
            begin tabspace (50); punch octal (space[nl base - i]);
                  PUSPACE (2); ABSFIXP (4, 0, i); PUNLCR
            end;
            STOPCODE; PUNLCR;
            tabspace (55); absfixp (dp0);
            tabspace (55); absfixp (start); pucar (2);
            for i:= prog base step 1 until start - 1 do
            begin tabspace (48); FIXP (13, 0, space[i]);
                  PUSPACE (2); ABSFIXP (4, 0, i); PUNLCR
            end;
            PUNLCR;
            for i:= start step 1 until instruct counter - 1 do
            begin k:= space[i]; par:= k : 32768;
                  address:= k - par × 32768;
                  instruct number:= par : 10;
                  par:= par - instruct number × 10;
                  tabspace (48); ABSFIXP (3, 0, instruct number);
                  ABSFIXP (1, 0, par); ABSFIXP (5, 0, address);
                  PUSPACE (2); ABSFIXP (4, 0, i); PUNLCR
            end
      end
end output;
```

```
main program:
        for n:= 0 step 1 until end of memory do space[n]:= 0;
        instruct counter:= prog base:= nlp:= 0;
        text base:= end of memory : 3;
        nl base:= end of memory;


        prescan0;
        if ¬ derroneous
        then begin  prescan1;
                    translate
             end;

endrun:
end
end
```

2. FIXED DATA

2.0 DESCRIPTION

This section contains a summary description of the data (given in section 2.1) to be offered to the compiler in front of the ALGOL 60 text that has to be translated. The major part of pages 2 and 3 of the compiler text is dedicated to reading and storing these data. We will briefly comment on the function, and possibly the coding, of categories of data items:
The values of the *basic symbols* are the internal values of the <delimiter>s ([6], 2.3.), the <logical value>s ([6], 2.2.2.) and the symbols "new line", "underlining", "bar" and "lower case".
The information stored into the *macro identifiers* is explained in section 4.2, while the variables *function letter*, *function digit* and *c variant* play a role in the procedure *Process parameter* (page 81). It is the task of the latter three to change the addressing mode of instructions into dynamic.
The i-th entry in the table *internal representation* gives the internal value of the flexowriter symbol corresponding to its 7-holes MC-flexowriter code punching, the binary pattern of which equals the number i. The coding is as follows:

0 means parity error (MC-flexowriter code has odd parity),
-1 stands for a punching without any corresponding character,
-2 means the corresponding symbol has to be skipped (blank, stopcode, backspace, erase),
-122 corresponds to a lower case punching, and
-124 to an upper case punching.

For all other values n we have:

n = (internal value of the upper shift symbol) * d8 + internal value of the lower shift symbol,
e.g., 9482 = 37 * d8 + 10, where 37 is the internal value of "A" and 10 is the internal value of "a".

The table *word delimiter* gives the correspondence between the internal values of a word delimiter and its first two symbols as follows:

n = (internal value of the first letter of the word delimiter) * d14 +
(internal value of the second letter of the word delimiter) * d7 +
internal value of the word delimiter,
e.g., 232160 = 14 * d14 + 21 * d7 + 96, where 14 is the internal value of "e", 21 that of "l", and 96 that of the word delimiter *else*.

Hence no distinction is made between the basic symbols *step* and *string*.
The values in the table *macro list* only matter if code procedures are to be accepted in full extent. In that case the i-th entry in the table should contain the properties of the macro with macro number i (cf. table 4.2).
In *tabel* the macrowords of the optimized macros are given analogous to the list of macro identifiers. For their decoding see table 4.2.
The *instruct list* should contain the bit-patterns for the machine instructions which constitute the macros generated by the compiler (those instructions are given in table 4.3). For easy interpretation of the output here instruct list[i] = i * 10 * d15.
For each of the ten possible types of an expression the table *mask* gives a code to be used in an actual parameter descriptor.
Finally, the variable *end of memory* gives the upper bound of the array *space*, and *wanted* determines whether objectcode for line number administration has to be generated (wanted = 0) or not (wanted ≠ 0).

2.1 DATA

basic symbols:

64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	110	84	85	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111
112	113	114	115	116	117	119	126
127	122						

macro identifiers:

12583168	8401168	4210978	4207154	4215106	4195154
4219136	4223232	4227426	4227698	4235906	4244114
4244386	4256434	10555904	8425728	6369792	6377984
6386176	6394368	10596608	23183872	23192064	23200256
23208448	23216384	8540416	8544512	8548608	21135616
21139712	21143808	21147904	21152000	21156096	25354496
8581376	8585472	8589568	8593664	8597760	8601856
8605952	8610048	8614144	27492608	8622336	8626432
8630528	8634624	8638720	4227328	8643072	12845312
10752256	10756352	12857600	12861696	12865792	17064192
8679680	8683776	21270784	21274880	21278976	21283072
21287168	21291264	21295360	21299456	8720640	8724736
8728832	8732928	8737024	8388608	8741376	8749824
8761600	8765696	8769792	8773888	8777984	8783105
8787217	8783145	8787257	8791369	8963328	8967432
8971520	8975616	8971520	8975616	8983808	8990464
8996096	9001472	8791552	9008384	9012480	9016576
9020676	8975872	9024768	9028864	9032972	9037068
9041152	9045248	9049352	9053448	9058824	9065992
30046984	9065736	9086216	32159240	9098496	9102592
9106952	9106696	8791304	8975616	9114888	9118984
9119240	28001800	9098504	9202176	9209344	9217544

function letter: 32768

function digit: 65536

c variant: 98304

internal representation:

-2	20225	16898	0	17924	0	0	25863
25096	0	0	-2	0	-1	32638	0
31611	0	0	17155	0	23301	25606	0
0	32009	30583	0	-1	0	0	-1
20480	0	0	14365	0	14879	15136	0
0	15907	-2	0	-1	0	0	-1
0	19016	14108	0	14622	0	0	15393
15650	0	0	30809	0	-1	30326	0
19521	0	0	12309	0	12823	13080	0
0	13851	-1	0	-1	0	0	-1
0	11795	12052	0	12566	0	0	13337
13594	0	0	31319	0	-1	-1	0
0	9482	9739	0	10253	0	0	11024
11281	0	0	31832	0	-1	-1	0
31040	0	0	9996	0	10510	10767	0
0	11538	-122	0	-124	0	0	-2

word delimiter:

296926	477407	232160	182120	232425	265297	248914	462574
494548	526549	216150	199777	397418	444267	297964	625773
183405	167407	413168	462961	345458	509299	478708	247157

macro list:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

tabel:

8799488	8803584	8799496	8803592	8807688	8811776
8815872	8811784	8815880	8819976	8824064	8828160
8824072	8828168	8832264	8836352	8840448	8836360
8840456	8844552	8848640	8852736	8848648	8852744

8856840	8860928	8869120	8860936	8869128	8877320
8861184	8869376	8861192	8869384	8877576	8886016
8898304	8886024	8898312	8910600	8922624	8930816
8922632	8930824	8939016	8947200	8955392	8946952

8955144	8910088	8885760	8898048	8885768	8898056
8910344					

instruct list:

0	327680	655360	983040	1310720	1638400
1966080	2293760	2621440	2949120	3276800	3604480
3932160	4259840	4587520	4915200	5242880	5570560
5898240	6225920	6553600	6881280	7208960	7536640
7864320	8192000	8519680	8847360	9175040	9502720
9830400	10158080	10485760	10813440	11141120	11468800
11796480	12124160	12451840	12779520	13107200	13434880
13762560	14090240	14417920	14745600	15073280	15400960
15728640	16056320	16384000	16711680	17039360	17367040
17694720	18022400	18350080	18677760	19005440	19333120
19660800	19988480	20316160	20643840	20971520	21299200
21626880	21954560	22282240	22609920	22937600	23265280
23592960	23920640	24248320	24576000	24903680	25231360
25559040	25886720	26214400	26542080	26869760	27197440
27525120	27852800	28180480	28508160	28835840	29163520
29491200	29818880	30146560	30474240	30801920	31129600
31457280	31784960	32112640	32440320	32768000	33095680
33423360	33751040	34078720	34406400	34734080	35061760
35389440	35717120	36044800	36372480	36700160	37027840
37355520	37683200	38010880	38338560	38666240	38993920
39321600	39649280	39976960	40304640	40632320	40960000
41287680	41615360	41943040	42270720	42598400	42926080
43253760	43581440	43909120	44236800	44564480	44892160
45219840	45547520	45875200	46202880	46530560	46858240
47185920	47513600	47841280	48168960	48496640	48824320
49152000	49479680	49807360	50135040	50462720	50790400
51118080	51445760	51773440	52101120	52428800	52756480
53084160	53411840	53739520	54067200	54394880	54722560
55050240	55377920	55705600	56033280	56360960	56688640
57016320	57344000	57671680	57999360	58327040	58654720
58982400	59310080	59637760	59965440	60293120	60620800
60948480	61276160	61603840	61931520	62259200	62586880
62914560	63242240	63569920	63897600	64225280	64552960
64880640	65208320	65536000	65863680	66191360	66519040

mask:

20 21 22 23 20 31 28 31 31 29

end of memory: 4096

wanted: 0

3. EXAMPLES

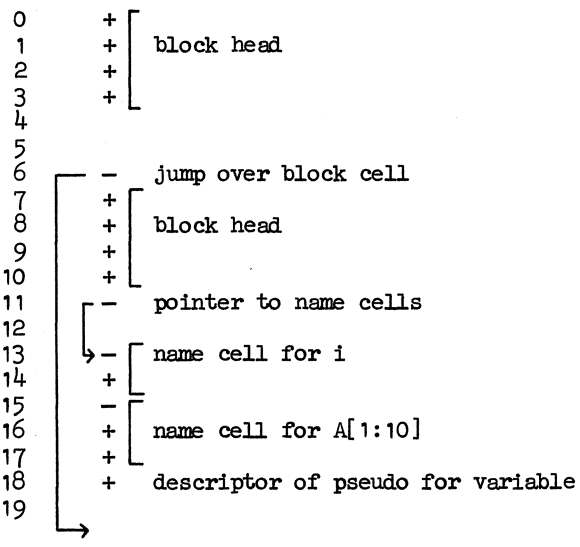
3.0 DESCRIPTION

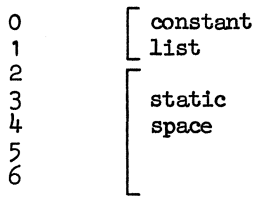
In the following two examples the translation of an array declaration, a for statement and a (recursive) procedure declaration is demonstrated. On each left page the state of affairs after each scan is given as provided by the procedure *output*. The contents of the name list are given in octal notation, the other numbers being decimal. On each right page, opposite to the name list some indication about its structure is given (see also chapter 5), and opposite to the object program the macros, generated by the compiler and its corresponding ELAN instructions are added, followed by some explanatory comments. For a more detailed explanation one is referred to [3], sections 4.8, 5.5 and 6.

3.1 EXAMPLE 1

```
begin integer i;
      real array A[1:10];
      for i := 1 step 1 until 10 do A[i]:= i*3.14
end
```

	prescan0	prescan1	translate	
erroneous	false	false	false	
text length	14	14	14	
namelist				
0	+'000000007'	+'000000007'	+'000000007'	0
1	+'000120200'	+'000120200'	+'000120200'	1
2	+'000500000'	+'000500000'	+'000500000'	2
3	+'000000000'	+'000000000'	+'000000000'	3
4	-'200000006'	-'200000006'	-'200000006'	4
5	-'200000024'	-'200000024'	-'200000024'	5
6	-'200000024'	-'200000024'	-'200000024'	6
7	+'000000000'	+'000000000'	+'000000000'	7
8	+'000000101'	+'000020101'	+'000020101'	8
9	+'000440000'	+'000440000'	+'000440000'	9
10	+'000000000'	+'000000000'	+'000000000'	10
11	-'200000015'	-'200000015'	-'200000015'	11
12	-'200000004'	-'200000004'	-'200000004'	12
13	-'000000023'	-'000000023'	-'000000023'	13
14	+'002000000'	+'002000005'	+'002000005'	14
15	-'000000046'	-'000000046'	-'000000046'	15
16	+'020000000'	+'020000006'	+'020000006'	16
17	+'000000002'	+'000000002'	+'000000002'	17
18	+'002000000'	+'002000004'	+'002000004'	18
19	-'200000014'	-'200000014'	-'200000014'	19
dp0		2	2	
start			7	
program				
0	+314	+314	+314	0
1	-2	-2	-2	1
		+0	+0	2
		+0	+0	3
		+0	+0	4
		+0	+0	5
		+0	+0	6


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
block head
jump over block cell
block head
pointer to name cells
name cell for i
name cell for A[1:10]
descriptor of pseudo for variable


0
1
2
3
4
5
6
constant
list
static
space

163	0	0	7
164	0	1	8
165	0	1	9
166	0	0	10
202	0	2	11
203	0	0	12
98	0	1	13
0	0	0	14
98	0	10	15
0	0	0	16
98	0	1	17
175	0	1	18
143	0	6	19
73	0	0	20
177	0	512	21
202	0	3	22
203	0	0	23
98	0	29	24
148	0	4	25
98	0	1	26
154	0	33	27
98	0	1	28
97	0	5	29
0	0	0	30
162	0	28	31
4	0	0	32
146	0	0	33
147	0	0	34
148	0	5	35
0	0	0	36
98	0	10	37
8	0	0	38
162	0	28	39
62	0	0	40
63	0	0	41
158	0	44	42
154	0	52	43
97	0	5	44
142	0	6	45
0	0	0	46
27	0	0	47
97	0	5	48
109	0	0	49
40	0	0	50
155	0	4	51
178	0	1	52
95	0	0	53
		+4	54

```
 7  TBL (1)
 8
 9  ENTRB (1)
10
11  LNC (2)
12
13  TSIC (1)
14  STACK
15  TSIC (10)
16  STACK
17  TSIC (1)
18  TDA (1)
19  TAA (6)
20  RAD
21  SWP (d9x1)
22  LNC (3)
23
24  TSIC (29)
25  SSTI (4)
26  TSIC (1)
27  JU (33)
28  TSIC (1)
29  TIV (5)
30  STACK
31  DO (28)
32  ADD
33  STI (5)
34
35
36  STACK
37  TSIC (10)
38  TEST1
39  DO (28)
40  TEST2
41
42  YCOJU (44)
43  JU (52)
44  TIV (5)
45  TAK (6)
46  STACK
47  STAA
48  TIV (5)
49  MULRC (0)
50  STSR
51  LJU (4)
52  EXITB (1)
53  END

54    4
```

```
            A = :MC
            F = :MD[1]
            B + 1
            SUB2 (:ENTRB)
            S = 2
            linecounter = S
            F = 1
            MC = F
            F = 10
            MC = F
            F = 1
            S = 1
            A = 6
            SUB3 (:RAD)
            M1 = B
            S = 3
            linecounter = S
            F = :M29
            M[4] = G
            F = 1
            GOTO (:M33)
M28:        F = 1
M29:        G = M[5]
            MC = F
            DO (M28)
            F + MC[-2]
M33:        S = F , Z
         N, SUBC (:RND)
            M[5] = G
            MC = F
            F = 10
            F - MC[-2] , Z
            DO (M28)
         N, F = F , E
         N, F = F , Z
         Y, GOTO (:M44)
            GOTO (:M52)
M44:        G = M[5]
            A = M[6]
            MC = F
            MC = A
            G = M[5]
            F x M[0]
            SUB2 (:STSR)
            GOTO (M[4])
M52:        B = MD[1]
            GOTO (:ENDRUN)

            4
```

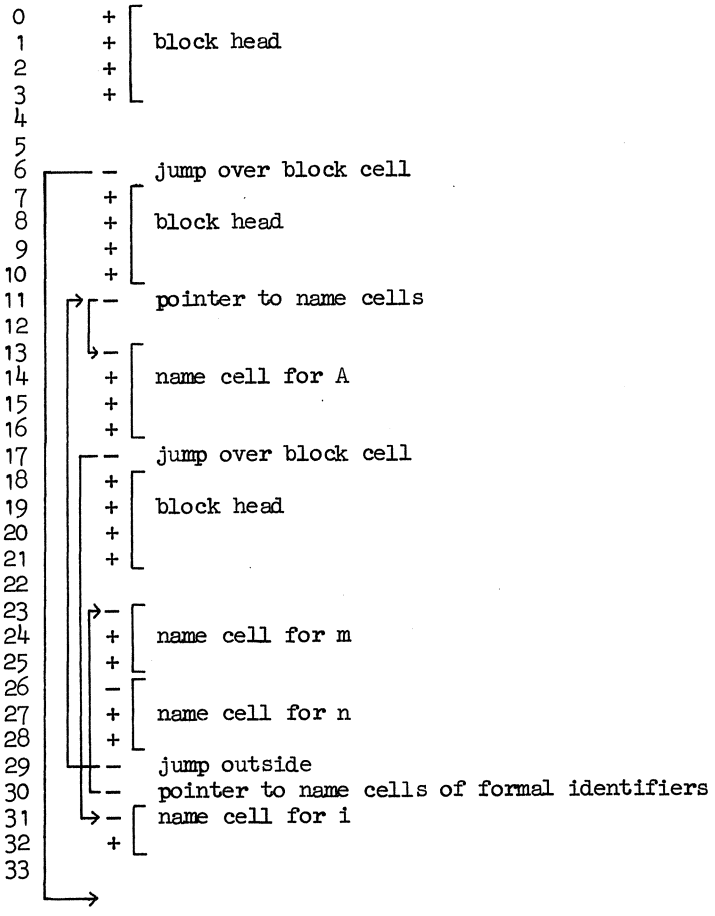
```
real array

A[1 : 10];


for i:=
1

step 1

until
10

do

A[i]:=

i
× 3.14
```

sum of maxima

3.2 EXAMPLE 2

```
begin comment Ackermann-function;
      integer procedure A(m,n); value m,n; integer m,n;
      A:= if m=0 then n+1 else if n=0 then A(m-1,1) else A(m-1,A(m,n-1));
      integer i;
      i:= A(4,0)
end
```

	prescan0	prescan1	translate	
erroneous	false	false	false	
text length	27	27	27	
namelist				
0	+'000000007'	+'000000007'	+'000000007'	0
1	+'000060200'	+'000060200'	+'000060200'	1
2	+'001040000'	+'001040000'	+'001040000'	2
3	+'000000000'	+'000000000'	+'000000000'	3
4	-'200000006'	-'200000006'	-'200000006'	4
5	-'200000042'	-'200000042'	-'200000042'	5
6	-'200000042'	-'200000042'	-'200000042'	6
7	+'000000022'	+'000000022'	+'000000022'	7
8	+'000000101'	+'000020101'	+'000020101'	8
9	+'001020000'	+'001020000'	+'001020000'	9
10	+'000000000'	+'000000000'	+'000000000'	10
11	-'200000015'	-'200000015'	-'200000015'	11
12	-'200000004'	-'200000004'	-'200000004'	12
13	-'000000046'	-'000000046'	-'000000046'	13
14	+'042000000'	+'042000000'	+'042000010'	14
15	+'020000003'	+'020000003'	+'032000003'	15
16	+'002000000'	+'003002405'	+'003002405'	16
17	-'200000037'	-'200000037'	-'200000037'	17
18	+'000160000'	+'000160000'	+'000160000'	18
19	+'000000102'	+'000060102'	+'000060102'	19
20	+'000740002'	+'000740002'	+'000740002'	20
21	+'000000000'	+'000000000'	+'000000000'	21
22	-'200000036'	-'200000036'	-'200000036'	22
23	-'000000027'	-'000000027'	-'000000027'	23
24	+'202000000'	+'202002403'	+'202002403'	24
25	+'000000000'	+'000000000'	+'000000000'	25
26	-'000000030'	-'000000030'	-'000000030'	26
27	+'202000000'	+'202002404'	+'202002404'	27
28	+'000000000'	+'000000000'	+'000000000'	28
29	-'200000013'	-'200000013'	-'200000013'	29
30	-'200000027'	-'200000027'	-'200000027'	30
31	-'000000023'	-'000000023'	-'000000023'	31
32	+'002000000'	+'002000002'	+'002000002'	32
33	-'200000014'	-'200000014'	-'200000014'	33


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
block head
jump over block cell
block head
pointer to name cells
name cell for A
jump over block cell
block head
name cell for m
name cell for n
jump outside
pointer to name cells of formal identifiers
name cell for i

dp0	0		0	
start			3	
program				
	+0		+0	0
	+0		+0	1
	+0		+0	2
	163	0	0	3
	164	0	1	4
	165	0	1	5
	166	0	0	6
	154	0	64	7
	167	0	0	8
	168	0	2	9
	169	0	0	10
	165	0	1	11
	65	0	0	12
	65	0	0	13
	170	0	2	14
	171	0	3	15
	172	0	0	16
	198	0	0	17
	199	3	32517	18
	202	0	3	19
	203	0	0	20
	97	3	32515	21
	119	0	0	22
	157	0	27	23
	97	3	32516	24
	105	0	1	25
	154	0	57	26
	97	3	32516	27
	119	0	0	28
	157	0	39	29
	154	0	35	30
	46	0	0	31
	97	3	32515	32
	108	0	1	33
	56	0	0	34
	159	0	8	35
	64	0	31	36
	22	4	1	37
	154	0	57	38

```
0
1          ⌈static space
2          ⌊
```

```
3  TBL (1)
4
5  ENTRB (1)
6
7  JU (64)
8  DPTR (2)
9
10
11 INCRB (1)
12 CIV
13 CIV
14 TDL (2)
15 ENTRPB (3)
16
17 SLNC ([D,5])
18
19 LNC (7)
20
21 TIV ([D,3])
22 EQUSIC (0)
23 COJU (27)
24 TIV ([D,4])
25 ADDSIC (1)
26 JU (57)
27 TIV ([D,4])
28 EQUSIC (0)
29 COJU (39)
30 JU (35)
31 ENTRIS
32 TIV ([D,3])
33 SUBSIC (1)
34 EXITIS
35 SUBJ (8)
36   apd
37   apd
38 JU (57)
```

```
            A = :MC
            F = :MD[1]
            B + 1
            SUB2 (:ENTRB)
            GOTO (:M64)
M8:         S = D
            A = -0 , Z
            SUB (:DPTR)
            B + 1
            SUBC (:CIV)
            SUBC (:CIV)
            F = :MA[2]
            B + 3
            SUB2 (:ENTRPB)
            S = linecounter
            MD[7] = S
            S = 7
            linecounter = S
            G = MD[3]
            F - 0 , Z
          N, GOTO (:M27)
            G = MD[4]
            F + 1
            GOTO (:M57)
M27:        G = MD[4]
            F - 0 , Z
          N, GOTO (:M39)
            GOTO (:M35)
ISR31:      SUB2 (:ENTRIS)
            G = MD[3]
            F - 1
            GOTO (:EXITIS)
M35:        SUBC (:M8)
            (20xd20+ISR31)
            ( 7xd20+ 1)
            GOTO (:M57)
```

```
if m
= 0

then n
+ 1

else if n
= 0

then
        ⌈ISR for
        m
        - 1
        ⌊
A
(m - 1,
 1)
```

154	0	54	39
46	0	0	40
97	3	32515	41
108	0	1	42
56	0	0	43
46	0	0	44
154	0	50	45
46	0	0	46
97	3	32516	47
108	0	1	48
56	0	0	49
159	0	8	50
4	0	32515	51
64	0	46	52
56	0	0	53
159	0	8	54
64	0	40	55
64	0	44	56
146	0	0	57
147	0	0	58
148	3	32517	59
97	3	32517	60
200	3	32517	61
201	0	0	62
82	0	0	63
202	0	5	64
203	0	0	65
159	0	8	66
22	4	4	67
22	4	0	68
146	0	0	69
147	0	0	70
148	0	2	71
178	0	1	72
95	0	0	73
		+8	74

```
39  JU (54)
40  ENTRIS
41  TIV ([D,3])
42  SUBSIC (1)
43  EXITIS
44  ENTRIS
45  JU (50)
46  ENTRIS
47  TIV ([D,4])
48  SUBSIC (1)
49  EXITIS
50  SUBJ (8)
51    apd
52    apd
53  EXITIS
54  SUBJ (8)
55    apd
56    apd
57  STI ([D,5])
58
59
60  TIV ([D,5])
61  RLNC ([D,5])
62
63  EXITP
64  LNC (9)
65
66  SUBJ (8)
67    apd
68    apd
69  STI (2)
70
71
72  EXITB
73  END

74    8
```

```
M39:      GOTO (:M54)
ISR40:    SUB2 (:ENTRIS)
          G = MD[3]
          F - 1
          GOTO (:EXITIS)
ISR44:    SUB2 (:ENTRIS)
          GOTO (:M50)
ISR46:    SUB2 (:ENTRIS)
          G = MD[4]
          F - 1
          GOTO (:EXITIS)
M50:      SUBC (:M8)
          ( 1×d20+d18+[D,3])
          (20×d20+ISR46)
          GOTO (:EXITIS)
M54:      SUBC (:M8)
          (20×d20+ISR40)
          (20×d20+ISR44)
M57:      S = F , Z
       N, SUBC (:RND)
          MD[5] = G
          G = MD[5]
          S = MD[7]
          linecounter = S
          GOTO (:EXITP)
M64:      S = 9
          linecounter = S
          SUBC (:M8)
          ( 7×d20+ 4)
          ( 7×d20+ 0)
          S = F , Z
       N, SUBC (:RND)
          M[2] = G
          B = MD[1]
          GOTO (:ENDRUN)

            8
```

```
else
          ⌈ISR for
          |m
          |- 1
          ⌊
          ⌈ISR for
          |
          |    ⌈ISR for
          |    |n
          |    ⌊- 1
          |
          |A
          |(m,
          | n - 1)
          ⌊
A
(m - 1,
 A (m, n - 1))
          ⌈assign
          |to
          ⌊A
A
(4,
 0)
          ⌈assign
          |to
          ⌊i

            sum of maxima
```

4. TABLES

4.1 OPTIMIZED MACROS

In the translation of arithmetical operations certain sequences of two or three macros are replaced by one macro, a so-called *optimized macro*, as described in section 0.3. In the table below the names of these optimized macros can be found. The numbers between parentheses are the opt numbers (see section 4.2). E.g., the sequence of macros TRC, NEG is replaced by the optimized macro TNRC and the sequence of macros STACK, TSIC, MUL by MULSIC.

	TRV (0)	TIV (1)	TRC (2)	TIC (3)	TSIC (4)
NEG (1)	TNRV	TNIV	TNRC	TNIC	TNSIC
ADD (2)	ADDRV	ADDIV	ADDRC	ADDIC	ADDSIC
SUB (3)	SUBRV	SUBIV	SUBRC	SUBIC	SUBSIC
MUL (4)	MULRV	MULIV	MULRC	MULIC	MULSIC
DIV (5)	DIVRV	DIVIV	DIVRC	DIVIC	DIVSIC
EQU (6)	EQURV	EQUIV	EQURC	EQUIC	EQUSIC
UQU (7)	UQURV	UQUIV	UQURC	UQUIC	UQUSIC
LES (8)	LESRV	LESIV	LESRC	LESIC	LESSIC
MST (9)	MSTRV	MSTIV	MSTRC	MSTIC	MSTSIC
MOR (10)	MORRV	MORIV	MORRC	MORIC	MORSIC
LST (11)	LSTRV	LSTIV	LSTRC	LSTIC	LSTSIC

4.2 MACRO DESCRIPTIONS

This table lists information on all macro identifiers and optimized macros used in the ALGOL 60 version of the compiler and on some additional macros used in the ELAN version for standard I/O routines.
On each left page is given
in column 1: the macro name,

2: the macro number (if any) used in the ELAN version. There a macro is not identified by its name, but by its number, which serves as an entry into the macro list,

3: the order number of the constituent instructions, followed, if relevant, by a code for the parameter type as follows:

code	parameter
c	pointer into the constant list
n	pointer into the name list, leading to an address in the object program
p	if value like then an address in the objectprogram else n
v	value (small integer)
np	if parameter > 0 then n else |parameter| is an address in the object program
=	instruction,

4: the corresponding ELAN instructions,

5: the meaning of the macro name.

On each right page the macro name is repeated, followed by the decoding of the value of the macro identifier or optimized macro. The coding is as follows:

```
value = B react * d21 + instr part * d12 + par part * d10 +
        instr nbr * d8 + opt nbr * d4 + value like * d3 +
        forw jump * d2 + opt op * d1 + sat macro.
```

B reaction is used in *Process stack pointer* (page 80) to estimate the amount by which the stack will grow during execution (see section 0.4). *Instruct part* gives the key to the first instruction of the macro in the

instruction table (4.3). *Par part* gives the order number of the instruction to which the macro parameter has to be added. *Instruct number* gives the number of consecutive instructions from the instruction table which constitute the macro. So for a macro the instructions in instruct list [instr part] upto and including that in instruct list [instr part + instr nbr - 1] are generated. *Opt number* gives for optimizable operators and simple arithmetic take macros (see section 0.4) the table entries into the table of optimized macros (4.1). *Value like* splits the parameters into two classes, which are treated separately in *Process parameter* (page 81). A value like parameter can never be a pointer into the name list. *Forward jumping* denotes whether *Macro2* (page 78) possibly has to assign the value of *instruct counter* to its *metaparameter* (cf. 0.4). Opt op and sat macro are the abbreviations of *Optimizable operator* and *Simple arithmetic take macro* respectively.

macro name	macro nr.	instr. nr.		elan instructions generated	meaning of macro name
ABS	76	1		F = F, P	absolute value
		2		N, F = -F	
ABSFIXP	134	1		SUBC (:ABSFIXP)	
ADD	2	1		F + MC[-2]	add
ADDIC	-	1	c	G + M[par]	add integer constant
ADDIV	-	1	n	G + Mp[q]	add integer variable
ADDRC	-	1	c	F + M[par]	add real constant
ADDRV	-	1	n	F + Mp[q]	add real variable
ADDSIC	-	1	v	F + :M[par]	add small integer constant
AND	19	1		N, B - 1	logical 'and'
		2		Y, S = MC[-1], P	
ARCTAN	131	1		SUBC (:ARCTAN)	arctangent
BAD	64	1		SUB3 (:BAD)	boolean array declaration
CBV	55	1		SUBC (:CBV)	call boolean by value
CEN	58	1		SUB (:CEN)	call expression by name
CIV	54	1		SUBC (:CIV)	call integer by value
CLPN	59	1		SUB1 (:CLPN)	call left part by name
CLV	57	1		SUBC (:CLV)	call label by value
CODE	128	1	=	par	store par in object code
COJU	106	1	p	N, GOTO (:M[par])	conditional jump
COS	130	1		SUBC (:COS)	cosine
CRV	53	1		SUBC (:CRV)	call real by value
CSTV	56	1		SUB1 (:CSTV)	call string by value
DECB	110	1	v	B - :M[par]	decrease B
DECS	38	1		S - 2	decrease S

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
ABS	4	86	0	2	0	0	0	0	0
ABSFIXP	0	187	0	1	0	0	0	0	0
ADD	2	4	0	1	2	0	0	1	0
ADDIC	4	104	1	1	0	1	0	0	0
ADDIV	4	104	1	1	0	0	0	0	0
ADDRC	4	103	1	1	0	1	0	0	0
ADDRV	4	103	1	1	0	0	0	0	0
ADDSIC	4	105	1	1	0	1	0	0	0
AND	3	25	0	2	0	0	0	0	0
ARCTAN	4	184	0	1	0	0	0	0	0
BAD	10	75	0	1	0	0	0	0	0
CBV	5	66	0	1	0	0	0	0	0
CEN	6	69	0	1	0	0	0	0	0
CIV	5	65	0	1	0	0	0	0	0
CLPN	8	70	0	1	0	0	0	0	0
CLV	6	68	0	1	0	0	0	0	0
CODE	4	173	1	1	0	1	0	0	0
COJU	4	157	1	1	0	1	1	0	0
COS	4	183	0	1	0	0	0	0	0
CRV	6	64	0	1	0	0	0	0	0
CSTV	6	67	0	1	0	0	0	0	0
DECB	4	161	1	1	0	1	0	0	0
DECS	4	49	0	1	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
DIV	5	1	MC = F	divide
		2	F = MC[-4]	
		3	F / MC	
DIVIC	-	1 c	G / M[par]	divide by integer constant
DIVIV	-	1 n	G / Mp[q]	divide by integer variable
DIVRC	-	1 c	F / M[par]	divide by real constant
DIVRV	-	1 n	F / Mp[q]	divide by real variable
DIVSIC	-	1 v	F / :M[par]	divide by small integer constant
DO	111	1 p	DO (M[par])	do an instruction
DOS	99	1 n	DOS (Mp[q])	dos apic 0
DOS2	100	1 n	DOS (Mp[q + 2])	dos apic 2
DOS3	101	1 n	DOS (Mp[q + 3])	dos apic 3
DPTR	114	1	S = D	display transport
		2 v	A = -:M[par - 2], Z	
		3	SUB (:DPTR)	
EMPTY	75			
END	82	1	GOTO (:END RUN)	end of program
ENTIER	78	1	SUBC (:ENTIER)	entier
ENTRB	113	1 v	B + :M[par]	enter block
		2	SUB2 (:ENTRB)	
ENTRIS	35	1	SUB2 (:ENTRIS)	enter implicit subroutine
ENTRPB	117	1 v	B + :M[par]	enter procedure body
		2	SUB2 (:ENTRPB)	
EQU	8	1	F - MC[-2], Z	equal
EQUIC	-	1 c	G - M[par], Z	equal to integer constant
EQUIV	-	1 n	G - Mp[q] , Z	equal to integer variable
EQURC	-	1 c	F - M[par], Z	equal to real constant
EQURV	-	1 n	F - Mp[q] , Z	equal to real variable
EQUSIC	-	1 v	F - :M[par], Z	equal to small integer constant

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
DIV	2	0	0	3	5	0	0	1	0
DIVIC	4	113	1	1	0	1	0	0	0
DIVIV	4	113	1	1	0	0	0	0	0
DIVRC	4	112	1	1	0	1	0	0	0
DIVRV	4	112	1	1	0	0	0	0	0
DIVSIC	4	114	1	1	0	1	0	0	0
DO	4	162	1	1	0	1	0	0	0
DOS	4	151	1	1	0	0	0	0	0
DOS2	4	152	1	1	0	0	0	0	0
DOS3	4	153	1	1	0	0	0	0	0
DPTR	14	167	2	3	0	1	0	0	0
EMPTY	4	0	0	0	0	0	0	0	0
END	4	95	0	1	0	0	0	0	0
ENTIER	4	91	0	1	0	0	0	0	0
ENTRB	4	165	1	2	0	1	0	0	0
ENTRIS	12	46	0	1	0	0	0	0	0
ENTRPB	15	171	1	2	0	1	0	0	0
EQU	2	8	0	1	6	0	0	1	0
EQUIC	4	117	1	1	0	1	0	0	0
EQUIV	4	117	1	1	0	0	0	0	0
EQURC	4	115	1	1	0	1	0	0	0
EQURV	4	115	1	1	0	0	0	0	0
EQUSIC	4	119	1	1	0	1	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
EXIT	50	1	GOTOR (MC[-1])	exit
EXITB	125	1 v	B = MD[par]	exit block
EXITC	126	1 v 2	B = MD[par] SUBC (:FREE CHAIN)	exit block using counter stack
EXITIS	45	1	GOTO (:EXITIS)	exit implicit subroutine
EXITP	71	1	GOTO (:EXITP)	exit procedure
EXITPC	72	1	GOTO (:EXITPC)	exit procedure using counter stack
EXITSV	127	1 v 2	S = :MC[par] GOTO (MS)	exit subscripted variable
EXP	80	1	SUBC (:EXP)	exponential function
FAD	39	1	GOTO (:FAD)	finish address description
FADCV	46	1	GOTO (:FADCV)	finish addr descr of controlled variable
FIXP	133	1	SUBC (:FIXP)	
FLOP	135	1	SUBC (:FLOP)	
HAND	142	1	SUBC (:HAND)	
IAD	63	1	SUB3 (:IAD)	integer array declaration
IDI	6	1	SUBC (:IDI)	integer division
IJU	104	1 n	GOTO (M[par])	indirect jump
IJU1	105	1 p	GOTO (M[par + 1])	indirect jump
IMP	17	1 2	Y, B - 1 N, S = -MC[-1], P	logical 'implies'
INCRB	115	1 v	B + :M[par]	increase B
ISUBJ	109	1 p	SUBC (M[par])	indirect subroutine jump
JU	102	1 np	GOTO (:M[par])	jump
JU1	103	1 p 2	A = :M[par] GOTO (:MA[1])	jump
JUA	74	1	GOTO (:JUA)	jump via A

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
EXIT	4	61	0	1	0	0	0	0	0
EXITB	4	178	1	1	0	1	0	0	0
EXITC	4	178	1	2	0	1	0	0	0
EXITIS	13	56	0	1	0	0	0	0	0
EXITP	4	82	0	1	0	0	0	0	0
EXITPC	4	83	0	1	0	0	0	0	0
EXITSV	13	180	1	2	0	1	0	0	0
EXP	4	93	0	1	0	0	0	0	0
FAD	4	50	0	1	0	0	0	0	0
FADCV	4	57	0	1	0	0	0	0	0
FIXP	0	186	0	1	0	0	0	0	0
FLOP	0	188	0	1	0	0	0	0	0
HAND	4	195	0	1	0	0	0	0	0
IAD	10	74	0	1	0	0	0	0	0
IDI	2	6	0	1	0	0	0	0	0
IJU	4	155	1	1	0	0	0	0	0
IJU1	4	156	1	1	0	0	0	0	0
IMP	3	21	0	2	0	0	0	0	0
INCRB	4	165	1	1	0	1	0	0	0
ISUBJ	4	160	1	1	0	0	0	0	0
JU	4	154	1	1	0	0	1	0	0
JU1	4	143	1	2	0	0	0	0	0
JUA	4	85	0	1	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
LAD	120	1 v	S = :M[par]	label declaration
		2	SUB (:LAD)	
LAST	119	1 p	–(A – M[par])	last element of label list
LES	10	1	F – MC[–2], P	less
		2	Y, F – 0 , P	
LESIC	–	1 c	G – M[par], P	less than integer constant
		2	N, F = F , Z	
		3	S = –T , P	
LESIV	–	1 n	G – Mp[q] , P	less than integer variable
		2	N, F = F , Z	
		3	S = –T , P	
LESRC	–	1 c	F – M[par], P	less than real constant
		2	N, F = F , Z	
		3	S = –T , P	
LESRV	–	1 n	F – Mp[q] , P	less than real variable
		2	N, F = F , Z	
		3	S = –T , P	
LESSIC	–	1 v	F – :M[par], P	less than small integer constant
		2	N, F = F , Z	
		3	S = –T , P	
LN	81	1	SUBC (:LN)	natural logarithm
LNC	147	1 v	S = :M[par]	set line counter
		2	line counter = S	
LOS	70	1	loose string = B	loose string
LST	13	1	F – MC[–2], Z	at least
		2	N, S = –1 , E	
LSTIC	–	1 c	G – M[par], P	at least integer constant
		2	N, F = F , Z	
LSTIV	–	1 n	G – Mp[q] , P	at least integer variable
		2	N, F = F , Z	
LSTRC	–	1 c	F – M[par], P	at least real constant
		2	N, F = F , Z	
LSTRV	–	1 n	F – Mp[q] , P	at least real variable
		2	N, F = F , Z	

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
LAD	4	175	1	2	0	1	0	0	0
LAST	4	174	1	1	0	0	0	0	0
LES	2	10	0	2	8	0	0	1	0
LESIC	4	124	1	3	0	1	0	0	0
LESIV	4	124	1	3	0	0	0	0	0
LESRC	4	121	1	3	0	1	0	0	0
LESRV	4	121	1	3	0	0	0	0	0
LESSIC	4	127	1	3	0	1	0	0	0
LN	4	94	0	1	0	0	0	0	0
LNC	4	202	1	2	0	1	0	0	0
LOS	4	81	0	1	0	0	0	0	0
LST	2	15	0	2	11	0	0	1	0
LSTIC	4	124	1	2	0	1	0	0	0
LSTIV	4	124	1	2	0	0	0	0	0
LSTRC	4	121	1	2	0	1	0	0	0
LSTRV	4	121	1	2	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
LSTSIC	–	1 v	F – :M[par], P	at least small integer
		2	N, F = F , Z	constant
MOR	12	1	F – MC[–2], P	more
		2	N, F = F , Z	
		3	S = –T , P	
MORIC	–	1 c	G – M[par], P	more than integer constant
MORIV	–	1 n	G – Mp[q] , P	more than integer
		2	Y, F – 0 , P	variable
MORRC	–	1 c	F – M[par], P	more than real constant
MORRV	–	1 n	F – Mp[q] , P	more than real variable
		2	Y, F – 0 , P	
MORSIC	–	1 v	F – :M[par], P	more than small integer constant
MST	11	1	F – MC[–2], P	at most
		2	N, F = F , Z	
MSTIC	–	1 c	G – M[par], Z	at most integer
		2	N, S = –1 , E	constant
MSTIV	–	1 n	G – Mp[q] , Z	at most integer
		2	N, S = –1 , E	variable
MSTRC	–	1 c	F – M[par], Z	at most real constant
		2	N, S = –1 , E	
MSTRV	–	1 n	F – Mp[q] , Z	at most real variable
		2	N, S = –1 , E	
MSTSIC	–	1 v	F – :M[par], Z	at most small integer
		2	N, S = –1 , E	constant
MUL	4	1	F × MC[–2]	multiply
MULIC	–	1 c	G × M[par]	multiply by integer constant
MULIV	–	1 n	G × Mp[q]	multiply by integer variable
MULRC	–	1 c	F × M[par]	multiply by real constant
MULRV	–	1 n	F × Mp[q]	multiply by real variable
MULSIC	–	1 v	F × :M[par]	multiply by small integer constant

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
LSTSIC	4	127	1	2	0	1	0	0	0
MOR	2	12	0	3	10	0	0	1	0
MORIC	4	138	1	1	0	1	0	0	0
MORIV	4	138	1	2	0	0	0	0	0
MORRC	4	136	1	1	0	1	0	0	0
MORRV	4	136	1	2	0	0	0	0	0
MORSIC	4	127	1	1	0	1	0	0	0
MST	2	12	0	2	9	0	0	1	0
MSTIC	4	132	1	2	0	1	0	0	0
MSTIV	4	132	1	2	0	0	0	0	0
MSTRC	4	130	1	2	0	1	0	0	0
MSTRV	4	130	1	2	0	0	0	0	0
MSTSIC	4	134	1	2	0	1	0	0	0
MUL	2	5	0	1	4	0	0	1	0
MULIC	4	110	1	1	0	1	0	0	0
MULIV	4	110	1	1	0	0	0	0	0
MULRC	4	109	1	1	0	1	0	0	0
MULRV	4	109	1	1	0	0	0	0	0
MULSIC	4	111	1	1	0	1	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
NEG	1	1	F = -F	invert sign
NIL	118	1 p	A + M[par]	not last element in label list
NON	15	1	S = -T, P	logical 'not'
OBAD	68	1	SUB3 (:OBAD)	own boolean array declaration
OIAD	67	1	SUB3 (:OIAD)	own integer array declaration
OR	18	1 2	Y, B - 1 N, S = MC[-1], P	logical 'or'
ORAD	66	1	SUB3 (:ORAD)	own real array declaration
OSTAD	69	1	SUB3 (:OSTAD)	own string array declaration
PUHEP	141	1	SUBC (:PUHEPG)	
PUNCH	136	1	SUBC (:PUNCH)	
PUNLCR	137	1	SUBC (:PUNLCR)	
PUSPACE	138	1	SUBC (:PUSPACE)	
QVL	16	1 2	S = T , P S = MC[-1], E	logical 'equivalent'
RAD	62	1	SUB3 (:RAD)	real array declaration
READ	132	1	SUBC (:READ)	
REHEP	140	1	SUBC (:REHEPG)	
REJST	73	1	SUBC (:REJST)	reject string
RLNC	146	1 n 2	S = Mp[q + 2] line counter = S	restore line counter
RUNOUT	139	1	SUBC (:RUNOUT)	
SAS	37	1	stock 3 = S	save S
SIN	129	1	SUBC (:SIN)	sine
SIGN	77	1 2 3	F = F, Z N, F = 1, E N, F = -1	sign

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
NEG	4	3	0	1	1	0	0	0	0
NIL	4	173	1	1	0	0	0	0	0
NON	4	9	0	1	0	0	0	0	0
OBAD	10	79	0	1	0	0	0	0	0
OIAD	10	78	0	1	0	0	0	0	0
OR	3	23	0	2	0	0	0	0	0
ORAD	10	77	0	1	0	0	0	0	0
OSTAD	10	80	0	1	0	0	0	0	0
PUHEP	4	194	0	1	0	0	0	0	0
PUNCH	4	189	0	1	0	0	0	0	0
PUNLCR	4	190	0	1	0	0	0	0	0
PUSPACE	4	191	0	1	0	0	0	0	0
QVL	3	19	0	2	0	0	0	0	0
RAD	10	73	0	1	0	0	0	0	0
READ	4	185	0	1	0	0	0	0	0
REHEP	4	193	0	1	0	0	0	0	0
REJST	4	84	0	1	0	0	0	0	0
RLNC	4	200	1	2	0	0	0	0	0
RUNOUT	4	192	0	1	0	0	0	0	0
SAS	4	48	0	1	0	0	0	0	0
SIN	4	182	0	1	0	0	0	0	0
SIGN	4	88	0	3	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
SLNC	145	1	S = line counter	save line counter
		2 n	Mp[q + 2] = S	
SQRT	79	1	SUBC (:SQRT)	square root
SSTI	96	1 n	Mp[q] = G	simple store integer
SSTSI	31	1	SUB2 (:SSTSI)	simple store subscripted integer
STAA	20	1	MC = A	stack A
STAB	14	1	S = T	stack boolean
		2	MC = S	
STAD	65	1	SUB3 (:STAD)	string array declaration
STACK	0	1	MC = F	stack F
STB	97	1	S = T	store boolean
		2 n	Mp[q]= S	
STFSU	34	1	SUB2 (:STFSU)	store formal subscripted of unknown type
STI	95	1	S = F, Z	store integer
		2	N, SUBC (:RND)	
		3 n	Mp[q]= G	
STOP	144	1	SUBC (:STOP)	
STR	94	1 n	Mp[q] = F	store real
STSB	32	1	SUB2 (:STSB)	store subscripted boolean
STSI	30	1	SUB2 (:STSI)	store subscripted integer
STSR	29	1	SUB2 (:STSR)	store subscripted real
STSST	33	1	SUB2 (:STSST)	store subscripted string
STST	98	1 n	F = :Mp[q]	store string
		2	SUB2 (:STST)	
SUB	3	1	F = -F	subtract
		2	F + MC[-2]	
SUBIC	-	1 c	G - M[par]	subtract integer constant
SUBIV	-	1 n	G - Mp[q]	subtract integer variable
SUBJ	108	1 np	SUBC (:M[par])	subroutine jump

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
SLNC	4	198	2	2	0	0	0	0	0
SQRT	4	92	0	1	0	0	0	0	0
SSTI	4	148	1	1	0	0	0	0	0
SSTSI	10	42	0	1	0	0	0	0	0
STAA	5	27	0	1	0	0	0	0	0
STAB	5	17	0	2	0	0	0	0	0
STAD	10	76	0	1	0	0	0	0	0
STACK	6	0	0	1	0	0	0	0	0
STB	4	149	2	2	0	0	0	0	0
STFSU	10	45	0	1	0	0	0	0	0
STI	4	146	3	3	0	0	0	0	0
STOP	4	197	0	1	0	0	0	0	0
STR	4	145	1	1	0	0	0	0	0
STSB	10	43	0	1	0	0	0	0	0
STSI	10	41	0	1	0	0	0	0	0
STSR	10	40	0	1	0	0	0	0	0
STSST	10	44	0	1	0	0	0	0	0
STST	4	98	1	2	0	0	0	0	0
SUB	2	3	0	2	3	0	0	1	0
SUBIC	4	107	1	1	0	1	0	0	0
SUBIV	4	107	1	1	0	0	0	0	0
SUBJ	4	159	1	1	0	0	0	0	0

macro name	macro nr.	instr. nr.		elan instructions generated	meaning of macro name
SUBRC	–	1	c	F – M[par]	subtract real constant
SUBRV	–	1	n	F – Mp[q]	subtract real variable
SUBSIC	–	1	v	F – :M[par]	subtract small integer constant
SWP	124	1	v	Mp[0] = B	set working space pointer
TAA	123	1	n	A = :Mp[q]	take address of array reference
TAK	92	1	n	A = Mp[q]	take array key
TASB	42	1		GOTO (:TASB)	take actual subscripted boolean
TASI	41	1		GOTO (:TASI)	take actual subscripted integer
TASR	40	1		GOTO (:TASR)	take actual subscripted real
TASST	43	1		GOTO (:TASST)	take actual subscripted string
TASU	44	1		GOTO (:TASU)	take actual subscripted of unknown type
TAV	60	1		SUB4 (:TAV)	transport array called by value
TBC	89	1	v	S = :M[par], Z	take boolean constant
TBL	112	1		A = :MC	take block level
		2	v	F = :MD[par]	
TBV	88	1	n	S = Mp[q] , P	take boolean variable
TCST	28	1		SUBC (:TCST)	take constant string
TDA	121	1	v	S = :M[par]	take dimension of array
TDL	116	1	v	F = :MA[par]	take display level
TEST1	51	1		F – MC[–2], Z	test for list element exhausted
TEST2	52	1		N, F = F, E	test for list element
		2		N, F = F, Z	exhausted
TFD	36	1		S = MS[1]	take formal display pointer
TFSL	27	1		SUBC (:TFSL)	take formal subscripted label
TFSU	25	1		SUB2 (:TFSU)	take formal subscripted of unknown type
TIAV	61	1		SUB4 (:TIAV)	transport integer array called by value

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
SUBRC	4	106	1	1	0	1	0	0	0
SUBRV	4	106	1	1	0	0	0	0	0
SUBSIC	4	108	1	1	0	1	0	0	0
SWP	4	177	1	1	0	1	0	0	0
TAA	4	143	1	1	0	0	0	0	0
TAK	4	142	1	1	0	0	0	0	0
TASB	4	53	0	1	0	0	0	0	0
TASI	4	52	0	1	0	0	0	0	0
TASR	4	51	0	1	0	0	0	0	0
TASST	4	54	0	1	0	0	0	0	0
TASU	4	55	0	1	0	0	0	0	0
TAV	4	71	0	1	0	0	0	0	0
TBC	4	141	1	1	0	1	0	0	0
TBL	4	163	2	2	0	1	0	0	0
TBV	4	140	1	1	0	0	0	0	0
TCST	4	39	0	1	0	0	0	0	0
TDA	4	175	1	1	0	1	0	0	0
TDL	4	170	1	1	0	1	0	0	0
TEST1	2	8	0	1	0	0	0	0	0
TEST2	4	62	0	2	0	0	0	0	0
TFD	4	47	0	1	0	0	0	0	0
TFSL	4	38	0	1	0	0	0	0	0
TFSU	11	36	0	1	0	0	0	0	0
TIAV	4	72	0	1	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
TIC	86	1 c	G = M[par]	take integer constant
TISCV	48	1	GOTO (:TISCV)	take integer subscripted controlled variable
TIV	84	1 n	G = Mp[q]	take integer variable
TLV	91	1 n	A = :Mp[q]	take label variable
TNA	122	1 v	F = :M[par]	take number of arrays
TNIC	–	1 c	G = –M[par]	take negative integer constant
TNIV	–	1 n	G = –Mp[q]	take negative integer variable
TNRC	–	1 c	F = –M[par]	take negative real constant
TNRV	–	1 n	F = –Mp[q]	take negative real variable
TNSIC	–	1 v	F = –:M[par]	take negative small integer constant
TRC	85	1 c	F = M[par]	take real constant
TRSCV	47	1	GOTO (:TRSCV)	take real subscripted controlled variable
TRV	83	1 n	F = Mp[q]	take real variable
TSB	23	1 2	SUB1 (:INDB) S '×' MA, Z	take subscripted boolean
TSCVU	49	1	GOTO (:TSCVU)	take subscr contr var of unknown type
TSI	22	1 2	SUB (:IND) G = MA	take subscripted integer
TSIC	87	1 v	F = :M[par]	take small integer constant
TSL	26	1	SUBC (:TSL)	take subscripted label
TSR	21	1 2	SUB (:IND) F = MA	take subscripted real
TSST	24	1 2	SUB (:IND) A = MA	take subscripted string
TSTV	90	1 n	A = Mp[q]	take string variable
TSWE	93	1 p	A = :M[par]	take switch entry
TTP	7	1	SUBC (:TTP)	to the power

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
TIC	4	97	1	1	3	1	0	0	1
TISCV	4	59	0	1	0	0	0	0	0
TIV	4	97	1	1	1	0	0	0	1
TLV	4	143	1	1	0	0	0	0	0
TNA	4	98	1	1	0	1	0	0	0
TNIC	4	101	1	1	0	1	0	0	0
TNIV	4	101	1	1	0	0	0	0	0
TNRC	4	100	1	1	0	1	0	0	0
TNRV	4	100	1	1	0	0	0	0	0
TNSIC	4	102	1	1	0	1	0	0	0
TRC	4	96	1	1	2	1	0	0	1
TRSCV	4	58	0	1	0	0	0	0	0
TRV	4	96	1	1	0	0	0	0	1
TSB	11	32	0	2	0	0	0	0	0
TSCVU	4	60	0	1	0	0	0	0	0
TSI	11	30	0	2	0	0	0	0	0
TSIC	4	98	1	1	4	1	0	0	1
TSL	4	37	0	1	0	0	0	0	0
TSR	11	28	0	2	0	0	0	0	0
TSST	11	34	0	2	0	0	0	0	0
TSTV	4	142	1	1	0	0	0	0	0
TSWE	4	143	1	1	0	0	0	0	0
TTP	2	7	0	1	0	0	0	0	0

macro name	macro nr.	instr. nr.	elan instructions generated	meaning of macro name
UQU	9	1	F – MC[–2], Z	unequal
		2	S = –T , P	
UQUIC	–	1 c	G – M[par], Z	unequal to integer
		2	S = –T , P	constant
UQUIV	–	1 n	G – Mp[q] , Z	unequal to integer
		2	S = –T , P	variable
UQURC	–	1 c	F – M[par], Z	unequal to real
		2	S = –T , P	constant
UQURV	–	1 n	F – Mp[q] , Z	unequal to real
		2	S = –T , P	variable
UQUSIC	–	1 v	F – :M[par], Z	unequal to small
		2	S = –T , P	integer constant
XEEN	143	1	SUBC (:XEEN)	
YCOJU	107	1 p	Y, GOTO (:M[par])	yes–conditional jump

macro name	B react.	instr. part	par. part	instr. nbr.	opt. nbr.	value like	forw. jump.	opt. op.	sat macro
UQU	2	8	0	2	7	0	0	1	0
UQUIC	4	117	1	2	0	1	0	0	0
UQUIV	4	117	1	2	0	0	0	0	0
UQURC	4	115	1	2	0	1	0	0	0
UQURV	4	115	1	2	0	0	0	0	0
UQUSIC	4	119	1	2	0	1	0	0	0
XEEN	4	196	0	1	0	0	0	0	0
YCOJU	4	158	1	1	0	1	1	0	0

4.3 INSTRUCTION TABLE

The instruction table given below is taken from the ELAN version of the compiler. It contains all machine instructions needed by the macro processor (in particular by *Produce* (page 79)) to build up the object program from the macros generated by the compiler.

```
INSTR LST[0]:     MC =  F                 " DIV, STACK
                  F = MC[-4]
                  F /  MC
                  F = - F                 " SUB, NEG
                  F + MC[-2]              " ADD
                  F × MC[-2]              " MUL
                  SUBC (:IDI)             " IDI
                  SUBC (:TTP)             " TTP
                  F - MC[-2], Z           " EQU, UQU, TEST1
                  S = - T, P              " NON

                  F - MC[-2], P           " LES
               Y, F -   0, P
                  F - MC[-2], P           " MST, MOR
               N, F =   F, Z
                  S = - T, P
                  F - MC[-2], Z           " LST
               N, S = - 1, E
                  S =   T                 " STAB
                  MC =  S
                  S =   T, P              " QVL

                  S = MC[-1], E
               Y, B -   1                 " IMP
               N, S = - MC[-1], P
               Y, B -   1                 " OR
               N, S = MC[-1], P
               N, B -   1                 " AND
               Y, S = MC[-1], P
                  MC =  A                 " STAA
                  SUB (:IND)              " TSR
                  F =  MA

                  SUB (:IND)              " TSI
                  G =  MA
                  SUB1 (:INDB)            " TSB
                  S '×' MA, Z
                  SUB (:IND)              " TSST
                  A =  MA
                  SUB2 (:TFSU)            " TFSU
                  SUBC (:TSL)             " TSL
                  SUBC (:TFSL)            " TFSL
                  SUBC (:TCST)            " TCST

                  SUB2 (:STSR)            " STSR
                  SUB2 (:STSI)            " STSI
                  SUB2 (:SSTSI)           " SSTSI
                  SUB2 (:STSB)            " STSB
                  SUB2 (:STSST)           " STSST
                  SUB2 (:STFSU)           " STFSU
                  SUB2 (:ENTRIS)          " ENTRIS
                  S = MS[1]               " TFD
                  stock3 = S              " SAS
                  S -  2                  " DECS
```

```
INSTR LST[50]:     GOTO (:FAD)          " FAD
                   GOTO (:TASR)         " TASR
                   GOTO (:TASI)         " TASI
                   GOTO (:TASB)         " TASB
                   GOTO (:TASST)        " TASST
                   GOTO (:TASU)         " TASU
                   GOTO (:EXITIS)       " EXITIS
                   GOTO (:FADCV)        " FADCV
                   GOTO (:TRSCV)        " TRSCV
                   GOTO (:TISCV)        " TISCV

                   GOTO (:TSCVU)        " TSCVU
                   GOTOR (MC[-1])       " EXIT
                N, F =   F, E           " TEST2
                N, F =   F, Z
                   SUBC (:CRV)          " CRV
                   SUBC (:CIV)          " CIV
                   SUBC (:CBV)          " CBV
                   SUB1 (:CSTV)         " CSTV
                   SUBC (:CLV)          " CLV
                   SUB (:CEN)           " CEN

                   SUB1 (:CLPN)         " CLPN
                   SUB4 (:TAV)          " TAV
                   SUB4 (:TIAV)         " TIAV
                   SUB3 (:RAD)          " RAD
                   SUB3 (:IAD)          " IAD
                   SUB3 (:BAD)          " BAD
                   SUB3 (:STAD)         " STAD
                   SUB3 (:ORAD)         " ORAD
                   SUB3 (:OIAD)         " OIAD
                   SUB3 (:OBAD)         " OBAD

                   SUB3 (:OSTAD)        " OSTAD
                   loose string = B     " LOS
                   GOTO (:EXITP)        " EXITP
                   GOTO (:EXITPC)       " EXITPC
                   SUBC (:REJST)        " REJST
                   GOTO (:JUA)          " JUA
                   F =   F, P           " ABS
                N, F = - F
                   F =   F, Z           " SIGN
                N, F =   1, E

                N, F = - 1
                   SUBC (:ENTIER)       " ENTIER
                   SUBC (:SQRT)         " SQRT
                   SUBC (:EXP)          " EXP
                   SUBC (:LN)           " LN
                   GOTO (:END RUN)      " END
                   F = M[0]             " TRV, TRC
                   G = M[0]             " TIV, TIC
                   F =   0              " TSIC, TNA, STST
                   SUB2 (:STST)
```

```
INSTR LST[100]:    F = - M[0]          " TNRV, TNRC
                   G = - M[0]          " TNIV, TNIC
                   F = - 0             " TNSIC
                   F + M[0]            " ADDRV, ADDRC
                   G + M[0]            " ADDIV, ADDIC
                   F +   0             " ADDSIC
                   F - M[0]            " SUBRV, SUBRC
                   G - M[0]            " SUBIV, SUBIC
                   F -   0             " SUBSIC
                   F × M[0]            " MULRV, MULRC

                   G × M[0]            " MULIV, MULIC
                   G ×   0             " MULSIC
                   F / M[0]            " DIVRV, DIVRC
                   G / M[0]            " DIVIV, DIVIC
                   F /   0             " DIVSIC
                   F - M[0], Z         " EQURV, EQURC, UQURV, UQURC
                   S = - T, P
                   G - M[0], Z         " EQUIV, EQUIC, UQUIV, UQUIC
                   S = - T, P
                   F -   0, Z          " EQUSIC, UQUSIC

                   S = - T, P
                   F - M[0], P         " LESRV, LESRC, LSTRV, LSTRC
                N, F =   F, Z
                   S = - T, P
                   G - M[0], P         " LESIV, LESIC, LSTIV, LSTIC
                N, F =   F, Z
                   S = - T, P
                   F -   0, P          " LESSIC, LSTSIC, MORSIC
                N, F =   F, Z
                   S = - T, P

                   F - M[0], Z         " MSTRV, MSTRC
                N, S = - 1, E
                   G - M[0], Z         " MSTIV, MSTIC
                N, S = - 1, E
                   F -   0, Z          " MSTSIC
                N, S = - 1, E
                   F - M[0], P         " MORRV, MORRC
                Y, F -   0, P
                   G - M[0], P         " MORIV, MORIC
                Y, F -   0, P

                   S = M[0], P         " TBV
                   S =   0, Z          " TBC
                   A = M[0]            " TSTV, TAK
                   A =   0             " TLV, TSWE, TAA, JU1
                   GOTO (:MA[1])
                   M[0] = F            " STR
                   S =   F, Z          " STI
                N, SUBC (:RND)
                   M[0] = G            " SSTI
                   S =   T             " STB
```

```
INSTR LST[150]:    M[0] = S
                   DOS (M[0])          " DOS
                   DOS (M[2])          " DOS2
                   DOS (M[3])          " DOS3
                   GOTO (:M[0])        " JU
                   GOTO (M[0])         " LJU
                   GOTO (M[1])         " LJU1
                N, GOTO (:M[0])        " COJU
                Y, GOTO (:M[0])        " YCOJU
                   SUBC (:M[0])        " SUBJ

                   SUBC (M[0])         " ISUBJ
                   B -   0             " DECB
                   DO (M[0])           " DO
                   A = :MC             " TBL
                   F = :MD[0]
                   B +   0             " INCRB, ENTRB
                   SUB2 (:ENTRB)
                   S =   D             " DPTR
                   '0327 77776'
                   SUB (:DPTR)

                   F = :MA[0]          " TDL
                   B +   0             " ENTRPB
                   SUB2 (:ENTRPB)
                   A + M[0]            " NIL
                   A - M[0]            " LAST
                   S =   0             " TDA, LAD
                   SUB (:LAD)
                   M0[0] = B           " SWP
                   B = MD[0]           " EXITB, EXITC
                   SUBC (:FREE CHAIN)

                   S = :MC[0]          " EXITSV
                   GOTO (MS)
                   SUBC (:SIN)         " SIN
                   SUBC (:COS)         " COS
                   SUBC (:ARCTAN)      " ARCTAN
                   SUBC (:READ)        " READ
                   SUBC (:FIXP)        " FIXP
                   SUBC (:ABSFIXP)     " ABSFIXP
                   SUBC (:FLOP)        " FLOP
                   SUBC (:PUNCH)       " PUNCH

                   SUBC (:PUNLCR)      " PUNLCR
                   SUBC (:PUSPACE)     " PUSPACE
                   SUBC (:RUNOUT)      " RUNOUT
                   SUBC (:REHEPG)      " REHEP
                   SUBC (:PUHEPG)      " PUHEP
                   SUBC (:HAND)        " HAND
                   SUBC (:XEEN)        " XEEN
                   SUBC (:STOP)        " STOP
                   S = line counter    " SLNC
                   M[2] = S
```

```
INSTR LST[200]:    S = M[2]              " RLNC
                   line counter = S
                   S =   0               " LNC
                   line counter = S
```

4.4 ACTUAL PARAMETER DESCRIPTORS

In the objectprogram every actual parameter is characterized by an *Actual Parameter Descriptor*. These APD's directly follow the procedure call, one for each parameter, in the order of the actual parameter list. Each APD uses one 27-bits word (d26, d25, ..., d0) as follows:

d0 - d14	contain an address or a value,
d15 - d17	0,
d18	is 1 when the address in d0 - d14 is a dynamic one, else 0,
d19	0,
d20 - d25	contain a number (≤32) indicating the type of the actual parameter,
d26	0.

Some types of actual parameters are too complicated to be described fully by an APD. For these parameters there also exists an *implicit subroutine* (ISR), that is a piece of object program, preceding the procedure call. The address in the APD then gives the (start) address of the ISR. Again the order of the ISR's is that of the actual parameter list. If there is at least one ISR, the first of them is preceded by a "jump over the implicit subroutines", that is a jump instruction leading directly to the procedure call.

In the following table the correspondence between the type of an actual parameter and its APD is given.

actual parameter	APD
real variable identifier (r)	(0 × d20 + address r)
integer variable identifier (i)	(1 × d20 + address i)
Boolean variable identifier (b)	(2 × d20 + address b)
string variable identifier (st)	(3 × d20 + address st)
floating point constant (fp)	(4 × d20 + address fp)
integer constant (n)	(5 × d20 + address n)
logical value (lv)	(6 × d20 + if lv then 0 else 1)
small positive integer (s)	(7 × d20 + s)
real array identifier (ar)	(8 × d20 + address ar)
integer array identifier (ai)	(9 × d20 + address ai)
Boolean array identifier (ab)	(10 × d20 + address ab)
string array identifier (ast)	(11 × d20 + address ast)
switch identifier (sw)	(12 × d20 + address sw)
small negative integer (-s)	(13 × d20 + s)
label identifier (lb)	(14 × d20 + address lb)
formal identifier (f)	(15 × d20 + address f)

actual parameter	APD
real array element	(16 × d20 + address ISR)
integer array element	(17 × d20 + address ISR)
Boolean array element	(18 × d20 + address ISR)
string array element	(19 × d20 + address ISR)
not assignable arithmetic expression	(20 × d20 + address ISR)
not assignable Boolean expression	(22 × d20 + address ISR)
not assignable string expression	(23 × d20 + address ISR)
real procedure identifier	(24 × d20 + address ISR)
integer procedure identifier	(25 × d20 + address ISR)
Boolean procedure identifier	(26 × d20 + address ISR)
string procedure identifier	(27 × d20 + address ISR)
designational expression	(28 × d20 + address ISR)
integer constant or designational expression	(29 × d20 + address ISR)
procedure identifier	(30 × d20 + address ISR)
not assignable expression of unknown type	(31 × d20 + address ISR)
possibly assignable expression of unknown type	(32 × d20 + address ISR)

5. STRUCTURE OF THE NAME LIST

In this chapter we briefly describe the structure of the name list in order to facilitate an analysis of the name-list procedures. No advanced techniques whatsoever are used: no hash tables, no binary search etc. The main design objectives have been compactness and simplicity, and never efficiency. In spite of this compile time has never been felt to be a bottle-neck to the system, except for ALGOL programs like the one presented in this tract. In the sequel we discuss the structure of a name cell first; then the overall structure of the name list and the look-up process for an identifier will be discussed.

5.1 NAME CELLS

Each name cell consists of:

1) a variable number of negative words, containing the component letters and digits of an identifier;

2) a descriptor of one, two or at most three positive words.

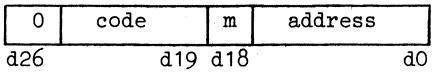
The word length of the EL-X8 is 27 bits; the digits 0:9 have internal representation 0:9, the letters a:z have 10:35, the letters A:Z have 37:62. In a name cell these values have been increased by 1. Each letter or digit is coded in 6 bits, and each word can contain 4 letters or digits. For the representation of an identifier of k letters and digits, (k+3) ÷ 4 (negative) words are needed. The identifier "a0123" is coded in octal:

space [*nl base - pointer*] : - '013010203'

space [*nl base - pointer - 1*]: - '000000004'

(cf. *read identifier*, page 11).
In this way identifiers of any length can be represented and discriminated.

The first word of any descriptor has the following lay-out:



in which:

code: a number specifying information on the nature of the object identified by the identifier;

m: for formal parameters called by name: whether the formal parameter occurs as left part somewhere in the procedure body or not,
for value formal parameters: always 0,
for non-formal identifiers: whether the address is a static address (in the array *space*) or a dynamic address (in the run-time stack); addresses of formal parameters always are dynamic (cf. *Assigned to* and *Dynamic* on page 84);

address: the address assigned to the identifier (in the case of a label identifier this address is the address of a pseudo label variable; however, not every label has such a variable assigned to it, cf. section 0.2).

The following codes can occur in a name cell:

1) for simple variables, labels, arrays, switches and procedures:

decimal	octal	
0	'000'	real variable identifier
1	'001'	integer variable identifier
2	'002'	Boolean variable identifier
3	'003'	string variable identifier
6	'006'	label identifier
8	'010'	real array identifier
9	'011'	integer array identifier
10	'012'	Boolean array identifier
11	'013'	string array identifier
14	'016'	switch identifier
16	'020'	real procedure identifier
17	'021'	integer procedure identifier
18	'022'	Boolean procedure identifier
19	'023'	string procedure identifier
24	'030'	procedure identifier
32	'040'	own real variable identifier
33	'041'	own integer variable identifier
34	'042'	own Boolean variable identifier
35	'043'	own string variable identifier

40	'050'	own real array identifier
41	'051'	own integer array identifier
42	'052'	own Boolean array identifier
43	'053'	own string array identifier

2) for value parameters:

decimal	octal	
64	'100'	specified *real* or *real procedure*
65	'101'	specified *integer* or *integer procedure*
66	'102'	specified *Boolean* or *Boolean procedure*
67	'103'	specified *string* or *string procedure*
70	'106'	specified *label*
72	'110'	specified *array* or *real array*
73	'111'	specified *integer array*
74	'112'	specified *Boolean array*
75	'113'	specified *string array*

3) for name parameters:

3.1) occurring in expressions or left parts as "primaries":

decimal	octal	
96	'140'	real
97	'141'	integer
98	'142'	Boolean
99	'143'	string
100	'144'	arithmetic (real or integer)
101	'145'	non-designational (real, integer, Boolean or string)
102	'146'	designational

3.2) occurring as array or switch identifier:

decimal	octal	
104	'150'	real array
105	'151'	integer array
106	'152'	Boolean array
107	'153'	string array
108	'154'	real or integer array
109	'155'	array of unknown type

110	'156'	switch
111	'157'	array or switch

3.3) occurring as procedure identifier:

decimal	octal	
112	'160'	real procedure
113	'161'	integer procedure
114	'162'	Boolean procedure
115	'163'	string procedure
116	'164'	real or integer procedure
117	'165'	function of unknown type
120	'170'	procedure

3.4) otherwise:

decimal	octal	
127	'177'	no information available

Specification of name parameters leads in *prescan0* to octal codes equal to '142' (*Boolean*), '143' (*string*), '144' (*real* or *integer*), '146' (*label*), '152' (*Boolean array*), '153' (*string array*), '154' (*real array* or *integer array*), '155' (*array*),'156' (*switch*), '162' (*Boolean procedure*), '163' (*string procedure*), '164' (*real procedure* or *integer procedure*) or '170' (*procedure*).

The other codes can result only from code transitions during *prescan1* by virtue of the contexts in which the parameter is used.

The following direct transitions can occur in this process:

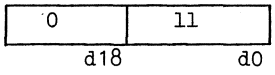
'140' to '150' or '160';
'141' to '151' or '161';
'142' to '152' or '162';
'143' to '153' or '163';
'144' to '140', '141', '154' or '164';
'145' to '140', '141', '142', '143', '144', '155' or '165';
'146' to '156';
'154' to '150' or '151';
'155' to '150', '151', '152', '153' or '154';
'157' to '150', '151', '152', '153', '154', '155' or '156';

'165' to '162', '163' or '164';
'170' to '162', '163','164' or '165';
'177' to '140', '141', '142', '143', '144', '145', '146', '157' or '170'.

Several of these transitions may occur one after another; one and the same context can induce two transitions in succession. In
procedure p(x,y); x:= y+2;...
the initial code '177' for x is transformed into '145' first by a call of *Assigned to* (page 40) and later on into '144' by a call of *Insert* (page 30) with type = 4; the initial code '177' for y is transformed into '144' by a call of *Arithmetic* (page 39).

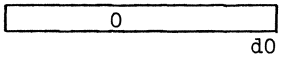
For simple <local or own type> variables the descriptor consists of one word only. All other name cells have a second descriptor word. The lay-out depends on the nature of the object identified by the identifier (as given by its code in the first descriptor word) in the following way:

1) for array identifiers and for formal parameters specified or used as array identifier, switch identifier of procedure identifier:

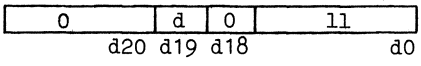


ll = 0 if the dimension or the number of parameters is unknown,
ll = 1 + (dimension or number of parameters), otherwise;

2) for all other formal parameters:

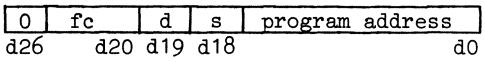


3) for switch identifiers:



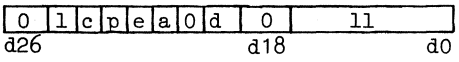
ll = 1 + number of entries in the switch list,
d = 1 after assignment of a "program address" to the switch identifier, i.e. after analysis of the switch list in the third scan;

4) for label identifiers:



fc = (for count =) the number of nested for statements the label "declaration" is enveloped in (this number is used for a very rough check against jumps from outside into a for statement),
d = 1 after assignment of a "program address" to the label, i.e. after its occurrence as label during the third scan,
s = 1 for labels to which no pseudo label variable is assigned;

5) for procedure identifiers:



ll = 1 + number of parameters,
l = 1 for library procedures (not declared in the program),
c = 1 for code procedures (declared in the program with code body),
p = 1 for some library procedures like *abs* that are compiled as monadic operator,
e = 1 except during the analysis of the procedure body in the third scan,
a = 1 after the occurrence of the assigment of a function value to the procedure identifier within the body of a type procedure during the third scan,
d = 1 after the assignment of a "program address" to the procedure in the third scan.

A third descriptor word exists for type procedure identifiers (codes '20', '21', '22' or '23') only. It contains the descriptor of a simple variable of corresponding type. This variable is assigned to the procedure to record the function value (cf. the compiler procedure *Local position*, page 85).

The declaration *array a [1:10]* in a program leads to the following name cell for *a*:

```
space  [nl base - pointer]       :  - '000000013'
space  [nl base - pointer - 1]   :    '021002407'  (code='10',m=1)
space  [nl base - pointer - 2]   :    '000000002'  (ll=2)
```

assuming a dynamic address '002407' for *a*.

For integer labels a special name cell is constructed with a normal descriptor preceded by two negative words containing together the integer value of the label (cf. *in name list*, page 12). No identifier can produce the same name cell as an integer label on account of a maximum value for integers of 2 ↑ 26 - 1.
The result of the procedure *Identifier*, both in *prescan1* and in *translate*, always points to the first descriptor word.

5.2 BLOCK CELLS

To each block in the program and to each procedure declaration corresponds a block cell in the name list. A block cell consists of the following sections:

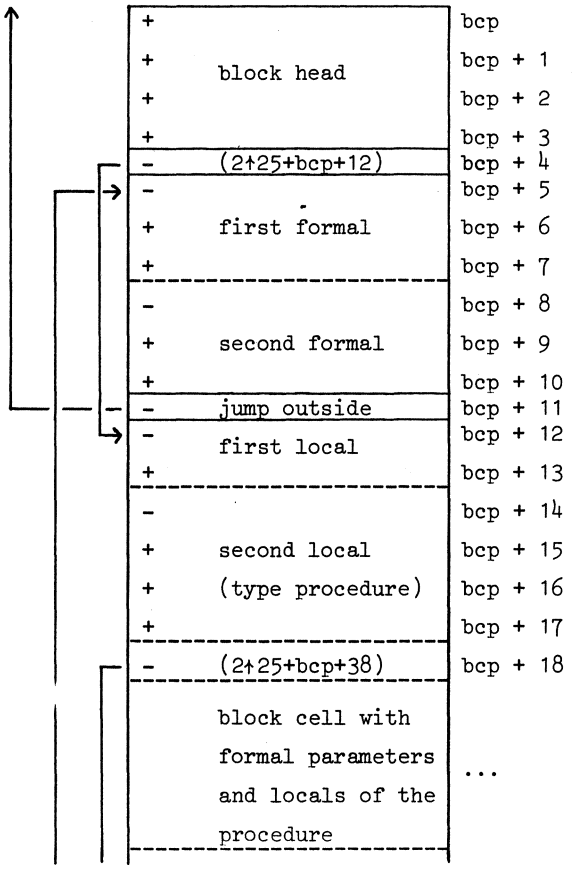
1) a block head of 4 positive words containing general information about the block or procedure body;
2) a negative word acting to the name-look-up procedure as a jump instruction and leading to the fifth section of the block cell (with the name cells of local identifiers);
3) the name cells of the formal parameters of the block cell, simply one after another without any separators (for blocks and for procedures without parameters this section is empty);
4) a negative word acting as a jump and leading to the fifth word of the block cell corresponding to the smallest embracing block (or procedure). This fifth word itself points to the locals of that block as described in section 2;
5) the name cells of the local identifiers of the block, simply one after another without any separators. In between these name cells, however, block cells may occur, either for procedures or for inner blocks. These block cells are each preceded by a jump over the block cell;
6) a number of consecutive one-word descriptors of pseudo for variables, coded as simple variables of type integer (i.e. code = '1');

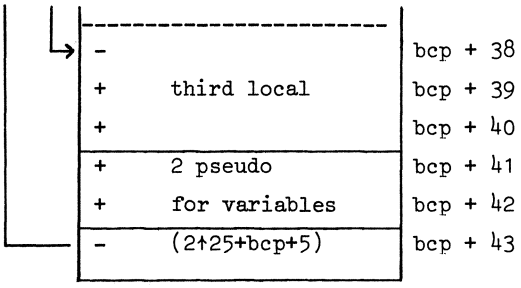
7) a negative word acting as a jump and leading to the third section of the block cell (with the name cells of formal identifiers).

Some of these sections can be empty, viz. section 3 (see above), section 5 (for procedures without locals) and section 6 (for blocks or procedures without for statements).

A jump pointing to a word in the array *space* with index *nl base - i* is coded as - (2↑25 + i); it can thus be distinguished from negative words of name cells, never that negative (cf. *next pointer*, page 11).

A typical block cell (for a procedure) is visualized in the following picture displaying the contents of *space* [*nl base - i*] for i = bcp (1) bcp + 43:





In this picture bcp is the "address" of the block cell.

The search process by which an identifier is looked up in the name list during *prescan1* or *translate* is rather simple and direct: the search is started at the fifth word of the appropriate block cell, leading to the first local identifier of the block cell. To skip an inapplicable name cell the momentary value of the pointer used in the search process is decreased by one until the first word of the next name cell is found (jumps encountered during this skip are honoured, of course). When the locals are exhausted the search is automatically continued with the formals of the same block and next with the locals of the smallest embracing block and so on until the identifier is found (due to a trick to be discussed below it will certainly be found). For identifiers occurring inside the bound pair list of an array declaration the search is started with the formal identifiers of the appropriate block cell rather than with the locals (cf. the procedure *look up*, page 12).

For the program as a whole, an outermost block cell is constructed starting at *space* [*nl base*] (its "address", therefore, equals 0), which is open-ended: section 7 is lacking. In this block cell labels global to all blocks of the program are inserted as well as, during *prescan1*, identifiers for which no declaration can be found. At the end of this block cell (found through the global compiler variable *nlp*), *read identifier* (page 11) assembles identifiers. If an identifier is looked up that has not been encountered before, the search process automatically ends at this temporary cell at the end of the name list.

During *prescan0*, the name list is constructed as an abstract of block structure and declarations. Identifiers to be added to the name list are

looked up (see *Process identifier*, page 23) for the purpose of detecting doubly declared identifiers. Identifiers occurring in the value part or in the specification part of a procedure declaration are looked up as well. In both cases the search is, of course, restricted to the current block cell which, then, is still open-ended.

The global compiler variable *block cell pointer* always contains the address in the name list of the current block cell. It has to be updated both at block entrance and at block exit. During *prescan1* and *translate* this is performed in the following way: at block entrance to *block cell pointer* is assigned the value of another global compiler variable *next block cell pointer*. This last one is updated with the contents of the least significant part of the first word of the new block cell (all block heads in the name list are linked together in a forward chain); at block exit *block cell pointer* is given the value of the most significant part of the first word of the (old) block cell (each block head in the name list points to the block head of the smallest embracing block, if any). These chains are constructed during *prescan0* which uses a different method for updating *block cell pointer*.

We do not specify the contents of the other three words of a block head, except for the most significant part of its third word (called *status*), which points to one of the descriptors of pseudo for variables (if any) in the block cell and which is decreased or increased by one before or after the analysis of the statement after *do* of a for statement during *translate*. In this way parallel for statements share the same pseudo for variable but nested for statements use different ones.

6. CROSS-REFERENCE TABLE

The cross-reference table is meant to be an aid to the study of the compiler. To all relevant procedure identifiers occuring in the ALGOL text an order number is added. The identifiers are listed alphabetically, each item preceded by its order number and the number of the page where its declaration occurs, and followed by a list or order numbers of the procedures from where it is directly called.