STAATSBEDRIJF DER POSTERIJEN, TELEGRAFIE EN TELEFONIE

REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART TWO:

THE INTERPRETER

DR. NEHER LABORATORIUM

REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART TWO:

THE INTERPRETER

BY: G. VAN DER MEY

WITH THE CO-OPERATION OF: W.L. VAN DER POEL
P.A. WITMANS
G.G.M. MULDERS

JULY 1962

comr

begin	The Interpre For special ALGOL features confer the heading of the tran tor. For object programmes to be interpreted and for their variables and arrays, the space PO ... QO is available. The object programme of the interpreter is suppose to be on address PO - 1 and to be realized accor to the translation process, given above;
own **integer** **array** st[PO, QO];	As this **own** array of the interpreter is declar first, it indeed occupies the absolute addres PO ... QO and is joined, within the store, b **own** variables, declared next. On address PO the interpreter contains a pass instruction, leadi to the object programme of compound statement ENTRY;
own **integer** A, B, chain, chain2, e, e2, I, J, mant, exp, N, E, p, p1, p2, P, Q, Q2, s1, y;	Confer table 4B;
procedure **wrong**;	This procedure, mentioned in the translator heading, reveals the mistakes, listed in table 5A. S1OL 24, 25, 27, 123;
procedure real(F, G);	Machine code, The procedure transfers integer F to the **real** representation, assigning the mantissa, and 1 + 2 × the exponent, of the result to F and G
	S1OL 27, 62, 65, 91, 122, 128;
procedure integer(F, G);	Machine code. f be the value of the number with the mantissa F and the exponent (G - 1)÷2. Then G : = 0 and F : = entier (f + 0.5). Switch for the non-extractive instructions. Table 1D shows the reversed order. S1OL15;

comment

<u>switch</u> sw1 : =
S1OL135, S1OL142, S1OL140, S1OL130,
S1OL128, S1OL120, S1OL96, S1OL91,
S1OL70, S1OL121, S1OL144, S1OL101,
S1OL145, S1OL58, S1OL55, S1OL106,
S1OL108, S1OL65, S1OL62, S1OL100;

Switch for the extractive instructions (table 1C).
S1OL19;

<u>switch</u> sw2 : =
S1OL88, S1OL87, S1OL81, S1OL144,
S1OL126;

As the arithmetic part of the interpreter is not described below, the labels in the next 2 switch lists are imaginary.
The operator ↑ is mentioned on S1OL22.
Switch for applying the operators (table 1A) to integral values mant and N.
The first division operator delivers a real result mant, exp. Otherwise an integral result is assigned to mant, for example:
<u>true</u> = 0 or <u>false</u> = -1 .
S1OL23;

<u>switch</u> sw3 : =
multiplication, division, integer
division, plus, minus, equal, not
equal, greater, greater or equal,
and, or, implies, equivalent;

Switch for applying the operators to real val[illegible]
mant, exp. and N. E.
Unless a type mistake occurs (cf. table 5A), a [illegible]
result is obtained, whose mantissa is assigned to
mant, while 1 + 2 × the exponent is assigned to exp.
The relational operators, combined, replace
extraction instruction[illegible] by an appropriate jump
instruction = [illegible] jump to the minus section,
return separa[illegible] [illegible] J, perform their te[illegible]
and, combined[illegible]
S1OL26;

con

switch sw4 : =	MULTIPLICATION, DI 10L147, PLUS, MINUS, EQUAL, NOT EQUAL, GREATER, GREATER OR EQUAL, S10L147, S10L147, S10L147, S10L147; S10L2, S10L3;
procedure JUMP (F);	Machine code. The procedure only executes the machine code jump instruction F, which is a positive w in ZEBRA code in contrary to the extraction ins tion of the interpreter, which is negative;

		comment	
	begin	Compound statement ENTRY;	
ENTRY:	A : = read;	which = $2^i - 1$, when exponents are wanted to occupy i digits (the right-most i digits of a location);	
	B : = p : = e2 : = 0;	B = st[Q0 + 2] may also be regarded as a permanent constant 0 of the interpreter which need not be cleared here;	
	Q : = Q0;	cf. table 1E;	+
	P : = st[P0];	which is the address P1 of table 4B;	
	e : = P0 + E0;		+ ×
	go to S10L1		
	end ENTRY;		

	comment
begin	Compound statement S10. S10L3, the instruction partres: S10L 41, 71;
S10L0: Q : = y : = Q − 2;	Addresses Q + 1 and Q + 2 are reserved for storing a partial result;
if Q < P then wrong	cf. table 5A. S10L124;
S10L0a: st[y + 1] : = mant; st[y + 2] : = exp;	Thus mantissa and exponent of a partial result stored separately. S10L 22, 63, 81, 89, 90, 99, 100, 104, 111, 1 130, 131, ENTRY;
S10L1: e : = e + 1;	Extraction instruction is increased with 1. S10L 44, 59, 95, 109; S11aL9;
S10L2: if e ≧ 0 then JUMP(e);	Then e is a machine code instruction and is normally executed. Otherwise 2 consecutive words are extracted from the object programme:;
I : = st[e − E0];	For E0 cf. table 1E;
N : = st[e + 1 − E0];	+ ×
S10L3: if I ≧ 0 then JUMP (I);	Then instruction I is normally executed. For example, instruction I = partres jumps to S10L0. A negative instruction I is interpreted as follows: S10L 73, 113, 139;
S10L4: y : = 8191 or I;	which is the address part of instruction I;
if y ≠ 0 then go to S10L9;	Otherwise address y is 0:;
S10L5: J : = I ÷ 2↑26;	which is analogous to S10L14; ×
S10L6: if (I or 2↑24) = 0 then go to S10L8;	Then N is a programme constant to be extracted by the instruction I preceding it in the object × programme.

	comment
	Instruction I must extract partial result from addresses Q + 1 and Q + 2 (cf. S10L0):;
S10L7: Q := Q + 2; N := st[Q - 1];	Addresses Q - 1 and Q are no longer occupied.
E := st[Q];	I can only be a calculative instruction, thus:;
go to S10L20;	
	S10L6;
S10L8: e := e + 1;	for programme constant N is not the instruction to be extracted next. I can only be an extractive or calculative instruction, thus:;
go to S10L17;	S10L4;
S10L9: J := I or 2↑23 - 2↑13;	which is 2^{13} × rank r0 contained in instruction I; ×
if J ≠ 0 then go to S10L10;	Then r0 must be looked up in the rank chain. r0 = 0 is the last element in the rank chain. The pre-value 0 corresponds to it, thus;
p1 := 0;	
go to S10L14;	S10L9;
S10L10: p1 := p; s1 := chain;	which are the resident values of p and c' = s + 2^{13}×r, in which s is a key addres
	S10L11;
S10L11: if s1 < J + 8192 then go to S10L12;	Then rank r looked up = r0. Otherwise r is still > r0 and next chain e is extracted:;
s1 := st[p1 + Q0 + 1];	s1 again having the form s1 = s + 2^{13} × r;
p1 := st[p1 + Q0 + 2];	×
go to S10L11;	S10L11;
S10L12: y := y + p1;	which addition is useful only when y i relative address of a variable thus no a ute address of a word of the object programme;
S10L13: if (I or 2↑23) = 0 then go to S10L30;	Then instruction I refers to a formal parameter. S10L 9, 35, 80, 77;

	comment
S10L14: J : = I ÷ 2↑26;	Then J = - 128 + number q, mentioned in table 1A, 1C or 1D for instruction I, the regr ive version included. The term - 128 is intr by the multiplication, because I is negativ,
S10L15: if J + 21 > 0 then go to sw1[J + 21];	Then instruction I is non-extractive. In ta the numbers 104 to 107 are not yet used. Otherwise an extraction is performed: S10L68;
S10L16: N : = st[y];	As N and st[y] are integral variables c interpreter, this is an assignment without of representation. Of course, st[y] may b variable of the object programme whose in tion is running. Yet the value of st[y] is lit ally copied. S10L8, S11L43;

S10L17: if (I or 2↑25) = 0
then E : = 0
else
begin
	For A cf. ENTRY on page 172;

y : = A; E : = y or N; N : = N-E; E : = 1 + 2 × (E -(2×E or y+1)) end	Bit I_7 = 0 indicates that N is of integer type. Then E : = 0. Otherwise N : = mantissa of extracted value, and E : = 1 + 2 × exponent with the correct sign digit E_0;
S10L18: if (I or 2↑24) ≠ 0 then go to S10L27;	Then the representation of the just extracted value of a formal parameter must be transferred. S10L 28, 51;
S10L19: if J + 33 > 0 then go to sw2[J + 33];	Then instruction I is extractive and its number q in table 1C ≥ 96;

	comment
	S10L7;
S10L20: if (J or 1) = 0 then go to S10L21;	Then the operation to be carried out is either commutative, or it must be applied to accu and the extracted value taken in the order as given here. Otherwise it must be applied to the values taken in the opposite order;
I : = N; N : = mant; mant : = I;	
I : = E; E : = exp; exp : = I;	S10L20;
S10L21: J : = (J + 64) ÷ 2;	Thus the gression bit is shifted off and J is no longer negative;
S10L22: if J = 0 then go to power;	exponentiation is performed and a return to S10L1. Otherwise:;
S10L23: if E ≠ 0 then go to S10L25;	Then E = 1 + 2 × exponent, and N = mantissa of a real value;
if exp = 0 then go to sw3[J];	Then operation is applied to integers mant and N;
S10L24: real (N, E);	As accu is real, integer N must also be given t e real representation;
go to S10L26;	S10L23;
S10L25: if exp = 0 then real (ma exp);	for the other value has also the real repre n, N and E.
	S10L24;
S10L26: go to sw4[J];	Thus operation is applied to real values.
	S10L 18, 51;
S10L27: if E = 0 then real (N, E) else integer (N, E);	
S10L28: go to S10L19;	S10L13;
S10L30: E : = st[y]; J : = st[y + 1];	Thus by-word and main word (table 3) of a formal parameter are extracted. S11L36;

	comment
S10L31: I : = I + (J or 2↑24 × 3);	Thus instruction I adopts the eventual type indication of key main word J;
S10L32: p1 : = I or 2↑26 × 127;	which is the operation part of instructio S10L 84, 86;
S10L33: y : = J or 8191;	which is the address part of key main word J;
S10L34: if J < 0 then go to S10L42;	Then parameter represents an expression;
S10L35: if 2 × J > 0 then go to S10L36;	Otherwise parameter represents simple variable
if p1 ≠ 2↑26 [illegible] S10L14;	
wrong;	cf. table 5A and jump in table 1D.
	S10L35;
S10L36: if 4 × J < 0 then go to S10L60;	Then parameter represents an array;
S10L37: if 8 × J < 0 then go to S10L82;	Then the parameter represents either a switch, or a label, or a constant actual parameter which constant also occurs in the text as a label;
S10L38: if 16 × J > 0 then wrong;	for a string has no type (cf. table 5A).
	Formal parameter f represents a procedure:;
S10L39: if p1 ≠ 2↑26 × 120 then go to S10L41;	Then f is either a designator having no actual parameters, or a left part element of an assignment statement.
	f(...). The prostat instruction (cf. table 1D) I which refers to parameter f and the key address N are the first 2 words of the object programme of designator f(...). The procedure, represented by parameter f, has, of course, formal parameters:;
S10L40: Q : = Q − 9;	
if Q < P then wrong;	cf. table 5A;

comment

```
        st[Q + 5]: = chain;
        st[Q + 6]: = p;
        st[Q + 7]: = P;
        st[Q + 8]: = N;
        I : = E;
        E : = st[y];
        J : = st[y + 1];
        go to S11L1;          J is the value of chain, belonging to the action of
        .                     the block to which the procedure to be called is local.
                              That value must be copied. In S11L4, the key address 0
                              is introduced in chain, which makes, when returning
                              from the procedure, the test on S11aL4 fail.
                              S10L39;
S10L41: if p1 ≠ 2↑26 × 116 then go to    Then a procedure having no formal parameters is   ×
        S10L42;               called.
                              Assignment to function name: (cf. store addre-
                              able 1D). A store procedure instruction is
                              ere instead of the store accu instruction of

        mant : = J + 2↑2                                                                  ×
        go to S10L0           S10L 34, 41;
S10L42: Q : = Q - 6;
        if Q < P then wrong;  cf. table 5A;
        st[Q + 1]: = mant;
        st[Q + 2]: = exp;
        st[Q + 3]: = chain;
        st[Q + 4]: = p;
        st[Q + 5]: = e;
        st[Q + 6]: = I;       The information, stored on addresses Q + 1 to Q + 6,
                              is used for the return on S10L47;
```

10L43: chain : = st[y]; p : = st[y + 1]; y	is the address Z mentioned for procedures and expressions in table 3;
S10L44: e : = E; if e < 0 then go to S10L2;	Then the parameter represents a (non- expression U. The object programme U' o now interpreted. When U is a subscripted variable, then, at of U', an ar2 instruction with subsequent proceeds to S10L74 for the return from U' U is another arithmetic or logical express instruction return at the end of U' jum L46 for the return from U'. U is a procedure:;
S10L45: I : = 1; go to S12L1;	The key address 1 is introduced in variable chain, and U is invoked. Instruction Y at the end of object programme U' jumps to S11aL1. The key address 1 makes S11aL3 proceed to S10L46 for return from U' S10L3, the instruction returns: S10L44, S11aL3;
S10L46: N : = mant; E : = exp;	Thus the value, obtained by evaluation of the expression or function, now occupies N and E, as does the value of an extracted variable on S10L17;
S10L47: mant : = st[Q + 1]; exp : = st[Q + 2]; chain: = st[Q + 3]; p : = st[Q + 4]; e : = st[Q + 5]; I : = st[Q + 6]; Q : = Q + 6;	Thus the values, retained by S10L42, are restored;
S10L48: J : = I ÷ 2↑26;	which is analogous to S10L14;
S10L49: if J + 24 < 0 then go to S10L51;	Then instruction I is calculative or extractive (cf. tables 1A and 1C);

	comment
S10L50: wrong;	cf. table 5A;
	S10L49;
S10L51: if (I or 2↑24) = 0 then go to	Then, in the text, no type is specified for parameter f; ×
S10L19;	
if (I + 2↑25×E or 2↑25) = 0	
then go to S10L19;	Then the representation of the value obtained is in accordance with the type specified for parameter f. Transfer is required:; ×
go to S10L27;	S10L82;
S10L52: if p1 ≠ 2↑26 × 122 then wrong;	cf. table 5A. ×
	E is the address Z mentioned in table 3 for labels and switches;
S10L53: s1 : = st[E]; p1 : = st[E+1];	which are the values to be restored before the jump;
S10L54: y : = y + p1;	only for joining the course coming from S10L12.
	S10L15, the jump instruction:;
S10L55: if e2 ≠ 0 then go to S10L57;	Otherwise the following data are stored, needed only wh[illegible] the jump instruction leads to a switch [illegible]truction (cf. S10L128) and, in addition, th[illegible] [illegible]u of the subscript concerned is "out of capa[illegible]
S10L56: chain2 : = cha[illegible]	
e2 : = e; p2 : = p[illegible]	[illegible]hen the jump instruction does not lead to a swi[illegible] instruction, it leads to a restore instruction (cf. S10L101).
	Then e2 is again cleared.
	S10L55;
S10L57: chain : = s1; p : = p1;	Thus, before the jump is performed, the appr[illegible] values of p and chain are restored.
	S10L130, S10L15, the pass instruction:;
S10L58: e : = y − p1 + E0;	Thus effect of S10L12 is compensated for;

	comment
S10L59: go to S10L2;	S10L36;
S10L60: if p1 ≠ 2↑26 × 126 then go to S10L64;	 Then I is no ar1 instruction (cf. table 1]
S10L61: y : = E − N;	st[E] is the first subscript factor, st[y] is the required subscript factor;
e : = e + 1;	which is analogous to S10L8; S10L15, the ar1 instruction:;
S10L62: if exp ≠ 0 then exp);	
S10L63: mant : = mant × st[illegible]; go to S10L1;	hich is a product of 2 integers; OL60;
S10L64: if p1 = 2↑26 × 125 then go to S10L65; if e > 0 then go to S11L40; wrong;	 for an ar2 instruction; for transport of a value array; cf. table 5A. S10L64, S10L15, the ar2 instruction:;
S10L65: if exp ≠ 0 then integer (mant, exp);	
S10L66: y : = mant + st[y];	The required subscripted variable is located on address y;
S10L67: if N = 0 then go to S10L74;	Then the ar2 instruction is the last instruction in the object programme of an actual parameter which is a subscripted variable. Return from that object programme is arranged;
if N = − 1 then go to S10L69;	Then assignment to a subscripted variable is prepared. Otherwise a subscripted variable must be extracted:;
S10L68: J : = − 30;	which is 2^{-26} × (operation part of extract normally in table 1C);

	comment
go to S1OL16;	S1OL67;
S1OL69: e : = e + 1;	as happens also on S1OL8.
	S1OL15, the store address instruction:;
S1OL70: mant : = y + (I or 2↑25)	cf. store accu in table 1D. This store accu
+ 2↑23 + 2↑26×117;	instruction is preliminarily stored as a "partial
	result". Its type indication has been copied from
	the ar2- or store address instruction I;
S1OL71: go to S1OL0;	The store accu instruction is extracted later on
	S1OL72.
	S1OL3, the instruction extract address : ;
S1OL72: Q : = Q + 2;	
I : = st[Q - 1];	
S1OL73: go to S1OL4;	The store accu instruction from S1OL70 or the
	store procedure instruction from S1OL41 is interpreted.
	S1OL67;
S1OL74: mant : = st[Q + 1];	
exp : = st[Q + 2];	
chain : = st[Q + 3];	
p : = st[Q + 4	
e : = st[Q +	
J : = st[Q + [illegible]	
Q : = Q + 6;	to be compared with S1OL47;
S1OL75: if(J or 2↑24) ≠ 0 then go to	Then, for the formal parameter which represents
S1OL78;	subscripted variable, a type has been specifie
	indicated by the bit J_7 (cf. S1OL31 and S1
	No type has been specified:;
S1OL76: I : = J + (I or 2↑24 × 3);	Thus, the type of the subscripted varia
	contained in the ar2 instruction, is [illegible]ced by the
	formal parameter;

	comment
S10L77: go to S10L14;	S10L75;
S10L78: I : = (I or 2↑25)-(J or 2↑25);	which is the difference of the type bit
S10L79: if I = 0 then I : = J - 2↑24	Thus either the type bit or the next bit
else I : = J + I;	instruction is inverted;
S10L80: go to S10L14;	S10L 87, 88,
	S10L19, the extract normally instruction:;
S10L81: mant : = N; exp : =	
go to S10L1	S10L37;
S10L82: if 16 × J <	Then formal parameter represents a label or a
	Parameter represents constant st[y],
	which constant occurs also as a label:;
S10L83: J : = st[y + 1];	which is the key main word of the constant
	parameter (cf. table 3);
S10L84: if (J or 2↑28) ≠ 0 then go to	Then the actual parameter key has already been
S10L33;	adjusted earlier on S10L85;
S10L85: st[y + 1] : = J : = if	
p1 = 2↑26 × 122	
then J + 2↑23 × 38	
else y + 2↑23 × 505;	Thus, depending on I being a jump instruction or
	not, the actual parameter must be a label or an
	arithmetic constant, and the key is adjusted
	accordingly. In the case of a label, E is already
	the correct key by-word of the formal parameter;
S10L86: go to S10L33;	S10L19, the extract inversion instruction:;
S10L87: N : = - N - 1;	which is the inversion of the previous value;
go to S10L81;	S10L19, the extract complement instruction:;
S10L88: N : = - N;	
go to S10L81;	S10L3, the instruction take inversion:;

	comment
S10L89: mant : = - mant - 1;	
go to S10L1;	S10L3, the instruction take complement :;
S10L90: mant : = - mant;	
go to S10L1;	S10L15, the store factor instruction (cf. S6aL14 and
	the explanation):;
S10L91: if exp ≠ 0 then integer	mant is the difference of the upper and lower bounds
(mant, exp);	of the bound pair.
	k be the number of the bound pairs;
S10L92: if mant < 0 then wrong;	cf. table 5A;
mant : = mant + 1;	
S10L93: st[y] : = mant;	
if N = 0 then go to S10L95;	Then y is the address reserved for H_k = the number
	of the array elements. The variable u and auxiliary
	variable v are located on addresses y + 1 and y + 2.
	If k > 1, the addresses y - 1 to y - k + 1 are
	reserved for the subscript factors h_{k-1} to h_1 to [illegible]
	[illegible]rmed by k further store factor instructi[illegible]
S10L94: y : = y + N;	[illegible]en y is the address of variable v;
st[y - 2] : = [illegible] 2] ×	[illegible]hich is the next partial value of H_k;
st[y - 1]: = [illegible]	[illegible]hich is the next partial value of u. st[y] i[illegible]
	lower bound, calculated last.
	S10L 93, 98;
S10L95: e : = e + 2;	
go to S10L2;	S10L15, the store pre-value instruction;
	(cf. S6aL24):;
S10L96: N : = N + y;	which is the address reserved for variable
	Addresses N + 1 to y are reserved for
	pre-values;
I : = st[N - 1];	which is the number of the array elements;

```
                                              comment
         J : = st[N];                         which is  u.
                                              S10L97;
S10L97:  st[y] : = P - J;
         P : = P + I;  y : = y - 1;
         if y ≠ N then go to S10L97;          Thus the pre-values are formed and stored.
                                              All pre-values have been stored:;
S10L98:  if P < Q then go
         wrong;                               cf. table 5A.
                                              S10L3,  the instruction retain:
                                              S10L 107, 110, S11L5;
S10L99:  st[Q + 1] : = P;                     Thus, the value of  p  is stored on address
         go to S10L1;                         S10L15,  the adjust instruction:;
S10L100: Q : = y;
         if Q < P then wrong;                 cf. table 5A;
         go to S10L1;                         S10L15,  the restore instruction:
                                              S10L129;
S10L101: e2 : = 0;                            Confer comment on S10L56.
                                              S10L128;
S10L102: Q : = y;                             Thus restore a previous value of Q;
S10L103: P : = st[y + 1];                     Restore also value of P;
S10L104: go to S10L1;                         S10L3,  the instruction for2:;
S10L105: mant : = 0;                          may also be another positive value;
         go to S10L107;                       S10L15, the for1 instruction:;
S10L106: I : = y + 2↑23 + 2↑26 × 72           which plus instruction (cf. table 1A) contains the
               + (I or 2↑23 × 6);             type indication of the for1 instruction.          - ×
                                              S10L105;
S10L107: P : = P + 4;                         Thus the addresses P - 4 to P - 1 are reserved
                                              for the for statement to be carried out;
         if Q < P then wrong;                 cf. table 5A;
```

```
                                        comment
        st[P - 4] : = e;
        st[P - 3] : = st[P - 2] : = I;      I = instruction for2 is positieve, the plus
                                            instruction is negative;
        go to S10L99;                       S10L15, the for instruction:;
S10L108: if st[P - 2] < 0 then mant : =
        - mant;
        if mant < 0 then go to S10L110;     Then cycle is no more executed;
S10L109: e : y + EC;                        cf. table 1E;
        go to S10L2;                        Thus execution of cycle begins.
                                            S10L108;
S10L110: P : = P - 4;                       Compare S10L107;
        go to S10L99;                       S10L3, the instruction for3:;
S10L111: st[P - 2] : := mant;
        st[P - 1] : = exp; go to S10L1;     Thus the step is stored.
                                            S10L3, the instruction for0:;

S10L112: e : = st[P - 4]: I : = st[P - 3];
        mant : = st[P - 2]: exp: = st[P   ]; cf. S10L107 and S10L111;
S10L113: if I < 0 then go to S10L4;         Then the step is added to the controlle     ;
S10L114: mant : = - 1;                      which may also be another negative value
        go to S10L1

                                            S10L15, the store procedure instruction:;
S10L120: N : = 0;                           S10L15, the store accu instruction:;
S10L121: if (I or 2↑25) ≠ 0 then go to S10L123;Then accu must be, or become, real;
S10L122: if exp ≠ 0 then integer (mant,exp); Thus accu is an integer;
        go to S10L124;                      S10L121;
S10L123: if exp = 0 then real (mant, exp);  S10L122;
S10L124: if N = 0 then go to S10L0a;        for assignment to function name;
                                            S11L38;
S10L125: N : = mant; E : = exp;             S10L19, cf. S11L42;
```

```
                                              comment
S10L126:  if E = 0 then go to S10L127;       Then integer N must be stored;
                                              N = mantissa, and E = 1 + 2 x exponent c
                                              be stored:;
          I : = A;                            For A cf. ENTRY on page 172;
          if E > 0 and E > I then
S10L126a: wrong;                              Then exponent is too large positive;
          E : = E - 1:                        which is 2 x the exponent;
          if E + I <
S10L126b: wrong;                              Then exponent is too large negative.
                                              E + I = 0 does not occur;
          N : = N + (N or I);                 Thus mantissa is rounded;
          N : = N - (N or I);                 Thus digits, reserved for exponent, are clea
          if N = 2↑32 then N : = N - I - 1;   Then there is no rounding;
          N : = N + (E÷2 or I);               S10L126;
S10L127:  st[y] : = N;
          go to S10L1;                        S10L15, the switch instruction:;
S10L128:  if exp ≠ 0 then integer(mant,exp);
          if mant ≦ 0 or mant > N then
          go to S10L129;                      Then subscript is "out of capacity";
          e : = e - mant - 1;
          go to S10L102;                      S10L128;
S10L129:  chain : = chain 2;
          e : = e2; p : = p2; y : = Q2;
          go to S10L101;                      S10L15, the test instruction:;
S10L130:  if mant = 0 then go to S10L1;
          go to S10L58;                       Thus, go to 1 for true = 0, go to 58 for false =
                                              - 1.
                                              S10L3, the extract procedure instruction:;
S10L131:  mant : = st [ p + Q0 + 4 ];
```

x +

		comment	
	exp : = st [p + Q0 + 5];		x +
	go to S10L1;	Thus value, assigned last to function name, is assigned to accu.	
		S10L15, the VERIFY instruction:;	
S10L135:	J : = I - 2↑23 - 2↑26 x 108;	cf. VERIFY in table 1D. Then J contains address- and rank part of VERIFY instruction;	x
	y : = 2↑26 x 125;	cf. ar2 in table 1D.	x
		S10L 141, 143;	
S10L136:	I : = st[Q + 6] or 2↑26 x 127;	which is the operation part of instruction st[Q + 6]. That instruction has been stored earlier on address Q + 6 on S10L42, where it was the instr. I, and it refers to a formal parameter, which represents the expression whose object programme contains the verification instruction which is going to be interpreted now;	x
S10L137:	I : = if I = 2↑26 x 122 then J + I else J + y;	Thus, if st[Q + 6] is a jump instruction, the verification instruction is replaced by th instruction with the same reference as con in the verification instruction. Otherwise verification instruction is replaced by the ar2- or extract normally instruction;	x
S10L138:	st[e - E0] : = I;	which word is definitive;	
S10L139:	go to S10L4;	Instruction I is interpreted. N has still the value, assigned on S10L3.	
		S10L15, the verify instruction:;	
S10L140:	J : = I - 2↑23 - 2↑26 x 110;	To be compared with S10L135;	
	y : = 2↑26 x 98;	cf. extract normally in table 1C;	x
S10L141:	go to S10L136;	S10L15, the Verify instruction:;	
S10L142:	J : = I - 2↑26 x 109;	To be compared with S10L135;	x

```
                                            comment
         y : = - J + 2↑23 + 2↑26 × 98;     When test on S10L137 fails, I beco..              x
                                            addressless extract normally instruct.

S10L143:  go to S10L136;
S10L144:  wrong;                            For dummy elements in switch lists.
S10L145:  wrong;                            A prostat instruction did not occur in conne
                                            with a formal parameter, cf. table 1D;
S10L146;  wrong;                            Integer operations may not be used with rea
                                            numbers;;

          end S10;
```

begin

comment

Compound statement S11.

The big transporter.

The object programme of any procedure having formal parameters contains, as its first word, machine code instruction X which jumps to here.

There is presented:

I = the machine code instruction which just jumped to X and which is the first word in the object programme of a designator having actual parameters, and key address N = the second word in that object programme.

The following transport is performed:

resident value of

I	→	Q - 2
p	→	Q - 3
chain	→	Q - 4

key main- and by-word of

first param.	→	Q - 5, Q - 6
second param.	→	Q - 7, Q - 8
⋮		
last param.	→	Q' + 3, Q' + 2.

These keys, derived from the offered actual parameter keys, contain only absolute addr (cf. table 3).

When a value param. does not represen its value becomes the by-word of

When present, value arrays are st n the lower part of the working space P...Q of the

comment

procedure to be invoked, and the va[...]r
P is increased accordingly.
There happens also:
p : = Q - 5 - Q0,
chain : = N + 2^{13} x rank of procedure to be in[...]
Q : = Q',
st[Q + 1] : = new value of P, as happens also a[...]
end of the dynamic introduction of a block.
The previous value of P is still available on
address p + Q0 + 3.
assignment to name of type procedure means [...]
[...]u on addresses p + Q0 + 4 and p + Q0 + 5

S11L0:	Q : = Q - 4;	
	<u>if</u> Q < P <u>then</u> wrong;	cf. table 5A;
	E : = chain; J : = p;	In S11L2, the value of chain will be stored on address Q, as is already mentioned above;
	I : = I + 1 - J0;	cf. table 1E
	y : = - P;	S10L40;
S11L1:	st[Q + 2] : = - y;	Thus, when coming from S11L0, st[Q + 2] is positive = P. When coming from S10L40, it is negative;
	st[Q + 3] : = N - Q;	
	st[Q + 4] : = I;	
	st[P] : = Q;	Thus: st[st[P] + 4] = address I where the procedure's object programme contains the specification pattern of its first formal parameter, being the second word,

+ x

```
                                          comment
                                          st[ st[P] + 3 ] = N - Q,
                                          which is the difference of the addresses N-1 and
                                          Q-1, occupied by the key main words of the first
                                          actual respectively first formal parameter,
                                          st[ st[P] + 2 ] is either = + P  (when coming from
                                          S11L0), or negative (when coming from S10L40).
                                          S11L 35, 38, 39, 41, 45;

S11L2:   Q : = Q - 2;
         if Q < P then wrong;             cf. table 5A;
         st[Q + 2]  : = E;
         st[Q + 3]  : = J;                Thus, first time, chain and p are stored. Next times,
                                          the parameter keys are stored.
                                          Cycle for forming and storing parameter keys:;
S11L3:   mant : = N : = st[P];            Bein the retained value of Q;
         y : = st[N + 2];
         J : = st[N + 3];                 wh ch is the ment oned difference of addres
         exp : = Q + J;                    ctual parameter key is extracted:;
         E : = st[exp];
         I : = st[                        The latter being the main word;
S11L4:   if I ≠ 0                         Then actual parameter list is not yet exhaust
                                          Actual parameter list exhausted:
                                          Q has already the mentioned value Q';
         if y < 0 then I : = N + J;       Otherwise I remains = 0 because the programme
                                          come from S10L40 instead of S11L0.
                                          I is the next key address;
         E : = st[N + 4];                 On address E the procedure's object pro
                                          contains - 2^13 x (rank of procedur
                                          S12L2;
S11L5:   p : = N - Q0 - 1;                                                           +x
```

		comment
	chain : = I - st[E] ;	being key address I + 2^{13} x rank of ... invoked;
	if chain < 0 then wrong;	Then the formal parameter list contains mo... than does the actual parameter list (compare S1... cf. table 1E;
	e : = E + E0;	
	go to S10L99;	Thus object programme of procedure body is execu... ...1L4;
S11L6:	if y < 0 t...	...hus, when coming from S10L40, N is = man... + 5... ...n both cases, st[N] ...s the value of chain wh... ...ill resident an... which must also be res...or... ...urn from the procedure. When coming from S10L40, st[N - 5] is the value ... chain belonging to the action of the block, to which the procedure to be invoked is local;
S11L7:	if I < 0 then go to S11L19;	Then the actual parameter is an expression. The key main word J, ... a formal parameter which represents an expression, must refer to the address N of the location which contains the value of chain belonging to the action of the block or procedure body, in which the expression occurs;
S11L8:	y : = 8191 or I;	which is the address, contained in the main word of the actual parameter key;
	J : = I or 2↑23 - 2↑13;	which is the rank part; x
	I : = I - J - y;	which is characteristic for the kind of the key;
S11L9:	if I ≠ 0 then go to S11L10;	Actual parameter is a string:;
	J : = y;	The key is simply copied;
	go to S11L24;	S11L9;
S11L10:	if J ≠ 0 then go to S11L12;	Then address N is not necessarily changed;

comment

S11L11:	N : = Q0 + 1;		+x
	p1 : = 0;	These values belong to the rank 0. On address Q0 + 2, the interpreter contains the constant 0, being the pre-value corresponding to the rank 0;	
	<u>go</u> <u>to</u> S11L14;	S11L10;	
S11L12:	p1 : = p; s1 : = chain;	S11L13;	
S11L13:	<u>if</u> s1 < J + 8192 <u>then</u> <u>go</u> <u>to</u> S11L14;	Compare S10L11;	
	N : = p1 + Q0 + 1;		+x
	s1 : = st[N]; p1 : = st[N + 1];		
	<u>go</u> <u>to</u> S11L13;	Thus the value of chain has been extracted from address N.	
		S11L11, 13;	
S11L14:	J : = p1 + y;	Thus, to a relative address y contained in an actual parameter key, corresponds an <u>absolute</u> address J to be included in the formal par key;	
S11L15:	<u>if</u> 2 x I < 0 t <u>go</u> <u>to</u> S	Then actual parameter is a simple variable.	
S11L16:	<u>if</u> 4 x	Then actual parameter is an array;	
S11L17:	<u>if</u> 8 x	Then actual parameter is either a switch or label, or it is a constant which also occurs the text as a label;	
S11L18:	<u>if</u> 16 x I < 0 <u>then</u> <u>go</u> <u>to</u> S11L19;	Then actual parameter is a procedure.	
		Actual parameter refers to a formal param any procedure:;	
	E : = st [J]; J : = st [J + 1] ;	Thus the key of that formal param copied;	
	<u>go</u> <u>to</u> S11L24;	S11L 7, 18;	

comment

S11L19: J : = I + N;
go to S11L24;

Thus, when a formal parameter represe
procedure, its key main word J refers
location which contains the value of chain
belonging to the action of the block to whi
that procedure is local.
S11L17;

S11L20: J : =
else I +

S11L21: E : = N;
go to S11L24;

Thus, when a formal parameter represents
or a label or a constant which also occurs
label, its key by-word E is the address of the
location which contains the value of chain
belonging to the action of the block to which
the label or switch is local. The value of exp.
is the address where the object programme contains
the key by-word of the actual parameter.
Thus the key main word J of a formal parameter
which represents a constant, refers to that
constant, which property is used on S10L82.
S11L16;

S11L22: E : = E + J;

Thus, when a formal parameter represents an
array, its key by-word E is an address. On
addresses E + 2, E + 1 and E are available: the
difference (address of first array element minus
pre-value), the number of array elements, and,
if the array has more than 1 dimension, the first
subscript factor.

S11L23: J : = I + J;

S11L24: I : = st[mant + 4] ;
st[mant + 4] : = I + 1;
I : = st[I] ;

if I < 0 then wrong;

S11L25: if (I or 2↑24) ≠ 0 then go to
S11L35;

S11L26: if J
S11L27: if 4 x

S11L28: if 8 x J < 0 then go
S11L29: if 16 x J < 0 then go to S11L32;
S11L30: wrong;

S11L31: if 16 x J < 0 then wrong;

comment

S11L15;
In the case of an array, the key main word J refers to the pre-value = the absolute address of the array element with all subscripts 0 (that element need not be contained in the space reserved for the array).

S11L 9, 18, 19, 21;

That is the spe tion patte the formal parameter, havir orm
0 000 1tx0 0...0;
Then the formal parameter list contains less ele ents than does the actual parameter list (c . test in S1 L5);
hen no type ha been specified for the fo parameter be ng considered.
A type has b er specified:;
Then par met r repi sents an expression
Then par mete r pres ts a simple var able.

Then parameter represents a constant;
Then parameter represents a procedure;
A type has been specified for a formal par representing a string.
S11L28;
Then a type has been specified fo parameter representing a

comment

When a type has been specified fo[illegible] representing a constant, that consta[illegible] the significance of a label, though a[illegible] elsewhere may be equal to it:;

st[exp + 1] : = J : = exp + [illegible] — Thus the parameter represents now a "simple [illegible] with pr[illegible] given value", located on [illegible]s exp;

go to [illegible] — S11L 2[illegible] 29;

S11L32: J : = [illegible] or [illegible] (- 2↑26 + 2↑24 [illegible] or J), — Thus the bi[illegible] of the main word is se[illegible] to the type bi[illegible] of the specification p[illegible]tt[illegible] copied. The ke[illegible] main word is consulted f[illegible] on S10L51; x

go to S11L35; — S[illegible]L27;

S11L33: J : = J or - 2↑25 + 2↑24 - 1; — Th[illegible] the bit J_8 of the main word is set to 0. x

S11L[illegible];

S11L34: if (J - I or 2↑25) ≠ 0 then J : = J + 2↑24; — T[illegible] when type [illegible]corded in main word J, differs from type, rec[illegible] the bit J_8 is set to [illegible] interpreter [illegible] informed, on S10L[illegible], [illegible] transfer is required for the value of the variable just extracted.

S11L 25, 32;

S11L35: if (2↑27 or I) ≠ 0 then go to S11L2; — Then the parameter is not value, and the key is ready to be stored in the working space of the procedure.

Parameter is value:;

S11L36: e : = j37 - 1; — j37 being the code instruction for jumping to the point corresponding to label S11L37; +x

I : = 2↑26 x 98; — cf. extract normally in table 1C; x

```
                                     comment
P : = P + 1;                         Then the space P...Q is free,
                                     st[P-1] being the retained value of Q;
go to S10L31;                        Then the extract normally instruction is regarded
                                     as to refer to the formal parameter with the key
                                     J, E. When the parameter represents a constant,
                                     expression, procedure, or variable, accu is
                                     given the value obtained by either extraction or
                                     eva[illegible]ation and the interpreter returns through
                                     [illegible]tion [illegible]S10L2) to S11L37 below.
                                     [illegible]ameter repres[illegible]rray,
                                     [illegible]ns[illegible]
                                     [illegible]erpreter
                                     [illegible]                           to S11L40
                                     [illegible]of cours[illegible]hout [illegible]racting the value
                                     of [illegible]array.
                                     S10[illegible] (cf. S11L36);
S11L37:  P : = P - 1;                Th[illegible]s st[P] is th[illegible] retained value of Q;

J : = if exp = 0
then Q + [illegible]      [illegible]lse [illegible]         Now the by-w[illegible]r (t[illegible] be stored on addre[illegible] Q.
2↑23 x 5[illegible]                   Will [illegible] [illegible]cons[illegible]ed [illegible]"si[illegible]le variabl[illegible] wit[illegible]
                                     [illegible]n [illegible]a [illegible]ccu" [illegible]ts address Q i[illegible] rec[illegible]
                                     [illegible]ain word.
if exp = 0 then go to S11L39;
S11L38:  y : = a[illegible]              On that address variable E is located;
         [illegible] : = [illegible] - 1;           j2[illegible]s the code instruction jumping to [illegible]
         [illegible]o to S10L125;            The[illegible]e the real value accu is "packed"
                                     as[illegible]d to variable E.

S[illegible]      E : = mant;
```

```
                    S11L2;      comment
                    : y;        S  cf. S11L36);
                                The N is address part of main word
                                st N] is (cf. S11L23) the pre-value.
                                st N]+ st[E + 2] is (cf. S11L22) the addre
                                reserved for the first array element;

         y : = P;
         P : = y                now to the addresses , y
                                1 a  ll,  its new positio , be
                                called a "value array";
         if Q < P then wr       Then there is no space enough for storing
                                value array
         st[P] : = st[y - 1];   Thus the retained value of Q is removed.
                                The value array will be stored now
                                pre-value of the value array will be stored now
                                on address y - 1.;
         st[y - 1] : = I : = y - st[E + 2]; i.e. lowest address minus difference
                                address minus pre-value) = pre-value;
         s1 : = st[N] - I;      ence of
         I : = J - N + y - 1;   the              paramet
                                d of the
                                value array.
                                As the value array has its other data in common
                                with the mother array, the by-word E which
                                refers to them need not be changed;
         if (I or 2↑24) ≠ 0 then go to   for transport with transfer of type.
         S11L42;                Fast transport:
                                S11L41;
S11L41:  st[y] : = st[s1 + y]; y : = y + 1;
         if y ≠ P then go to S11L41;
         J : = I;
         go to S11L2;
```

comment

```
                                        S10L40;
S11L42:   mant : = E;                   Thus by-word is put to safety;
          J : = - 28;                   Then switch on S10L19 goes to S10L126
                                        (cf. table 1C).
                                        S11L44;
 43:      e : = j44 - 1;                Code instruction j44 jumps to S11L44 below;     +x
          N : = st[s1 + y];
          go to S10L17;                 S    2;
 1L44:    y : = y + 1;
          if y ≠ P then go to S11L43
 1L45:    E : = mant;                   L42;
          J : = if I + 2↑23 x 260 < 0
          then I + 2↑23 x 1018 else     Th    he bit J8 of the new main word is 0, while
          I + 2↑24;                     bi    J7 differs from I7;
          go to S11L2
          end S11;
```

:comment

d statement S11a.

The restorer:

The last word in the object programme procedure is the e instruction Y, which to ... ;

... ich was resident bef ...

... stor ...

wh ... s th ... y a ... ss ... return;

Then the p ... dur has been called th ... ug formal para ... er and has no formal pa ... ame (cf. S10L4 ...);

Then the procedure has been called direc... without use of a formal parameter).

Th... procedure has been called through a formal parameter and has formal parameters:;

... s the ... tored on S10L40.

... instruction for return;

1aL1: Q : =

11aL2: I : =

11aL3: if I ... S10

11aL4: if I ≠ 0 then go to S11aL6;

11aL5: Q : = Q + 5;
I : = st[Q - 1] ;

aL6: e : = I + EO;

L7: chain : = st[
p : = st[Q - 3] ;
I : = st[Q - 2] ;

I is either a previous value of pointer P or a previous extraction instruction;

S11aL8: i... then e : = I else P:=I; When the procedure has no formal parameters, I is the required extraction instruction (cf. compound statement S12):

S11aL9: go to S10L2
end S11a;

```
                                   comment
     begin                         Compound statementS12.
                                   The small transporter.
                                   The object programme of a procedure which has
                                   no formal parameters, contains, as its first
                                   word, code instruction X1 which jumps to here.
                                   For explanation cf. S11;
     E : = 1 : = I - JO + 1        In   object programme of the procedure to be   +x
                                         st[P    13  x rank is the second
                                                  > 1.

1:   N : = Q - 4;
     Q : = N - 2;
     if Q < P then wrong;          cf.  ble 5A;
     st[N]      : = chain;
     st[N + 1]  : = p;
     st[N + 2]  : = e + 1;         hen coming  ro  S12L0, e + 1 is the ex ra
                                   instruction  or return.
                                   When coming  r  n S10L45, e + 1 need not be
     go to S
     end S12
     end
```