PTT

STAATSBEDRIJF DER POSTERIJEN, TELEGRAFIE EN TELEFONIE

REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART ONE:

THE TRANSLATOR

DR. NEHER LABORATORIUM

REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART ONE:

THE TRANSLATOR

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

JULY 1962

1.0 General comment
1.1 Procedure S0 : stores word in object program on address P.
1.2 Procedure S0a: extracts from list L and conjugates.
1.3 Procedure S0c: stores word in list L on address S.
1.4 Procedure S0d: equips object programme with storing and extracting a partial result.
1.5 Procedure S0e: stores declaration list I.
1.6 Procedure S0f: looks up identifier f in list I.
1.7 Procedure S0g: shifts list L to the mid of the range P...T.
1.8 Procedure S0h: completes dynamic introduction of a block.
1.9 Procedure S0i: reserves address P of the object programme for a pass instruction.
1.10 Procedure S0k: places extract normally instruction.
1.11 Procedure S0l: contra declares a label.
1.12 Procedure S0m: stores word in list L on address So-1.
1.13 Procedure S0n: stores contra declaration and internal equivalent of an identifier.
1.14 Procedure S0p: contra declares name of function having no parameters.
1.15 Procedure S0r: inverts gression bit of instruction d.
1.16 Procedure S0s: makes contra declaration of subscripted actual parameter.
1.17 Progressive and regressive after-actions of the opening symbols.
1.18 Compound statement "Entry".
1.19 Compound statement S1: Entry for delimiters being either an operator, or a separation or closing symbol.
1.20 Compound statement S2: Pre-action of the opening symbol if.
1.21 Compound statement S2a: After-actions of the opening symbol if.
1.22 Compound statement S3 : Pre-action of the opening symbol begin.
1.23 Compound statement S3a: Entry for declarators and specificators.
1.24 Compound statement S3b: After-action of the opening symbol begin.
1.25 Compound statement S3c: Entry for colon.
1.26 Compound statement S4 : Entry for identifiers.
1.27 Compound statement S4a: Entry for constants.
1.28 Compound statement S5 : Pre-action of the opening symbol (.
1.29 Compound statement S5a: After-actions of the opening symbol (.

1.30 Compound statement S5b: Action of the delimiter lsq.

1.31 Compound statement S6 : Pre-action of the opening symbol [.

1.32 Compound statement S6a: After-actions of the opening symbol [.

1.33 Compound statement S7 : Pre-action of the opening symbol for.

1.34 Compound statement S7a: After-actions of the opening symbol for.

1.35 Compound statement S8 : Pre-action of the opening symbol go to.

1.36 Compound statement S8a: After-actions of the opening symbol go to.

1.37 Compound statement S8b: After-actions of the opening symbol switch.

1.38 Compound statement S8d: After-actions of the opening symbol procedure.

1.39 Compound statement S9 : Pre-action of the opening symbol:=.

1.40 Compound statement S9a: After-actions of the opening symbol:=.

1.0 General comment

ALGOL Translator

begin

comment

Restrictions in action:

1 In any block head within a text to be translated, the declarations of simple varables and arrays are supposed to precede those of procedures and switches.

2 The bounds of own arrays are supposed to be integral constants.

3 The controlled variable of a for statement may not be a subscripted variable.

Notations:

1 Above each labeled line, all references to the label are gathered, represented by "approximate" labels.

2 Within the programme, many constants are written in a semi-binary notation, in which powers of 2 occur as factors and terms.
Crosses x on the right-hand side of the pages mark the lines, in which such non-variable expressions appear which, of course, should better be replaced in the text by their values.

3 Other constants, listed in table 1E, are conveniently referred to by names which, in praxis, are to be replaced in the text by the appropriate values. Crosses + on the right-hand side of the pages mark the lines concerred.

Composition:

The translator, a block, contains procedure wrong, the procedures of the SO-series,
the switches sw and SW, and the compounds S1, S2, S2a, S3, S3a, S3b, S3c,S4,S4a,S5,S5a,S5b,S6,S6a,S7,S7a,S8,S8a,S8b,S8d,S9,S9a, input and input1.

input, depending on the kind of string taken from the tape, goes to either S4L1 (for an identifier), or to S4aL1 (for an unsigned number), or to one of the labels mentioned in tables 1A and 1B. It assigns the value of the string to variable f of table 4A, which is, however, not necessary when the string is a delimiter
: : = ([<u>begin</u> <u>for</u> <u>go</u> <u>to</u> <u>if</u> <u>lsq</u> .
When f is a number, input assigns the value
<u>0 000000t01</u> 0...0 to variable g. The bit t indicates the representation of the number f.

input1 (cf. S5b).

Non-ALGOL features:

1 The translator considers boolean type to be the same as integer type. Thus variables declared boolean may assume integral values. Internally, <u>true</u> and <u>false</u> are represented by 0 and -1 respectively. Then the significance of operator <u>or</u> can be extended as follows:

2 When applied to integers p and q, p <u>or</u> q denotes the logical product of p and q which, of course, might also be obtained by a procedure logical product (par1, par2).

3 The ZEBRA, for which this translator has been developed, is a binary machine with 8192 locations in its store, each of which containing 33 bits b_0, b_1 ... b_{32}.
b_0 is the sign digit: $b_0=0 \rightarrow$ pos., $1 \rightarrow$ neg.
Then $x < 0$ is a short notation for logical product $(x, 2^{32}) \neq 0$;

	comment
own **integer** **array** st [P0: Q0] ;	In this space, object programmes are built up, and it also contains the lists I and L needed in translation time (cf. variables P, T, S of table 4A);
integer a, b, c, d, e, f, g, D, mark, P, q, R, S, S accent, S0, T, T1, T2, u, v;	cf. table 4A;
procedure wrong;	This hardware programme stops and prints address, from which it has been invoked. That address corresponds to the occurred kind of misery listed in table 5 as a function of a variable assuming labels as values;

+

```
                                  comment
procedure SO(F);                  stores word in object programme on address P;
begin
st[P] : = F; P : = P + 1;
if P>SO-2     then SOg;           for shifting list L away from object programme
end SO;
```

```
                                      comment
procedure S0a;                  extracts from list L and conjugates;
begin
c : = st[S - 3];
d : = st[S - 2];
e : = st[S - 1];
a : = 480 or d; b : = 480 or f; 480 is = 511 - 31, thus b is = 32 × rank of
                                delimiter f (cf. tables 1 A and B)
end S0a;
```

```
                                   comment
procedure SOc(F);                  stores word in list L on address S;
begin
st[S] : = F; S : = S + 2 ;
if S + 1 > T then SOg;             for shifting list L away from I
end SOc;
```

comment

```
procedure SOd;                 equips object programme with storing and extracting a
                               partial result;

begin
st[Saccent - 1] : =
2↑23 × 3 + 2↑26 × 63;                                                          ×
SO(partres)                                                                    +
end SOd;
```

```
                                    comment
procedure SOe(F)                    stores declaration in list I;
begin
st[T - 1]: = e;                     is internal equivalent of identifier F;
st[T]: = F; T : = T - 2;
if S + 1 > T then SOg;              for shifting list L away from I (cf.SOc)
end SOe;
```

	comment
procedure SOf;	looks up identifier f in list I on addresses T1 + 2, T1 + 4, ... T2. If f is found, then c : = lowest address where f occurs, and d : = corresponding internal equivalent, which is positive. Otherwise d : = -1. If f is found to be a simple variable or formal parameter, then e : = d. Otherwise e : = 2^{32} + f, which is the contra-identifier of f, being negative. If f is found to be an array identifier, then g : = the ar1 instruction which is the internal equivalent of the factor identifier, being neg.(cf.S6L8) and u : = d, which is used on S6L 17 Otherwise g : = 0;
begin	
g : = 0; c : = T1;	
SOfL1: **if** T2 = c **then** **go** **to** SOfL3;	Then identifier has not been found;
c : = c + 2;	
if st[c] ≠ f **then** **go** **to** SOfL1;	for extracting the next identifier;
d : = e : = st[c - 1];	which is the internal equivalent of identifier f;
if 2 × d > 0 **then**	
begin	
if 4 × d > 0 **then** **go** **to** SOfL4;	for label, switch- and procedure identifier;
e : = c - 1;	f is array identifier;

```
                                            comment

SOfL2 : e : = e - 2; g : = st[e];
        if g > 0 then go to SOfL2;  Then ar1 instruction is not yet found;
        u : = d; go to SOfL4;
SOfL3 : d : = -1;
SOfL4 : e : = f + 2↑32                                                     x
        end                         This compound statement is omitted in the case of a
                                    formal parameter or simple variable

        end SOf;
```

```
                                         comment
        procedure SOg;                   shifts list L to the mid of the range P ... T;
        begin
        a : = (P + T - S - SO) ÷ 2;      The distance from the mid of SO...S to that of
                                         P...T is = a + l with either l = 0, or ½, or -½;
        Saccent : = Saccent + a;
        S : = S + a;
        SO : = SO + a;
        if P + 2 > SO then
SOgL1:  wrong;                           Then translator is short of working space.
                                         Otherwise P + 2 is ≦ SO
                                         and, because of abs(l) ≦ ½, also S + 1 ≦ T
                                         as required in procedures SO, SOc, SOe, and SOm;
        if a > 0 then
        for b : = S step -1 until SO
        do st[b]: = st[b - a]
        else
        for b : = SO step 1 until S
        do st[b] : = st[b - a]
        end SOg;
```

```
                                            comment
procedure SOh;                              is invoked after reading the first array identifier
                                            of a list being not own (cf.S4L17), and also, when
                                            translation of a statement is beginning (cf. S2, S3,
                                            S5, S6, S7, S8, S9, S3cL2 and S3bL1). Bridging pass
                                            instructions are inserted in the dynamic introduction
                                            of a block on addresses previously reserved by
                                            procedure SOi, and the introduction is also completed
                                            by putting the instruction retain to its end.
                                            The procedure assigns the same values to variables c,
                                            d and e as does procedure SOa;
begin
a : = st[S - 7];
c : = st[S - 3];  d : = st[S - 2];
e : = st[S - 1];
if d = 322 and c ≠ 0 and a ≥ 0
then
begin                                       This only happens, when translation of a block head is
                                            still running;

if a > 0 then st[a] : =
P + 2↑23 + 2↑26 × 121;                      Then pass instruction (cf. table 1D) is inserted on    ×
                                            address a;

if D < 0 then a : = D else a:=0;
st[S - 7] : = a;
if a < 0 then
begin                                       Dynamic introduction of block is completed.
                                            q is smallest address reserved for simple variable
                                            which is local to block;

q : = q - 1 ;
```

comment

```
SO(retain);
end
end SOh
```

cf. table 1 E

+

Thus, when st[S - 2]= 322 which is the value of **begin** and st[S - 3]≠ 0, (cf. S3 and S3a), then st [S - 7] can be the address where to insert a pass instr. in dynamic introduction of block at next call of procedure SOh (cf. SOi), or 0 when no pass instr. is to be inserted, or negative, when introduction is ready;

comment

```
procedure SOi;                  is invoked on S4L10, S4L14, S4L24, S4L37.
                                Unless there has already been reserved an address, it
                                reserves address P of the object programme for a pass
                                instruction, which is to be inserted later by
                                procedure SOh;
if st[S - 7] = 0 then
begin
st[S - 7] : = P;
SO(0) ;                         A preliminary zero is inserted;
end
end SOi;
```

	comment
<u>procedure</u> SOk;	equips object programme with filling the accumulator, which is required whenever a computation is beginning in the object programme. For const. 98 cf. table 1C;
<u>if</u> g = 0 <u>then</u> <u>begin</u>	e, if > 0, is the internal equivalent of either a simple variable or formal parameter. If < 0, e is the contra-identifier of a function name. e = 0 occurs only at a call from S1L10. Then nothing is added to the object programme;
<u>if</u> e ≠ 0 <u>then</u> <u>begin</u> <u>if</u> e > 0 <u>then</u> SO(e + 2↑26 × 98) <u>else</u> SOp <u>end</u> if e > 0; <u>end</u> if g = 0 <u>else</u> <u>begin</u>	×
	Representation of constant e is indicated by bit t of g = <u>0 000000t01 0...0</u>; ×
SO (g + 2↑26 × 98); SO(e);	Thus, in the object programme, constant e is subsequent to the extract normally instruction concerned;
<u>end</u> else <u>end</u> SOk;	

```
                                         comment
     procedure SOl;                      contra-declares a label, which is either identifier
                                         st[S] or, when g ≠ 0, the constant e.
                                         For 127 = 0 001111111  cf. table 2A;
     if g = 0 then SOn(2↑23 × 127)                                                          ×
     else
     if e < 0 or  2↑24 ≤ e then                                                             ×
SOlL1: wrong;                            cf. table 5
     else
     begin
     a : = e + 2↑24 × 63 + 2↑32;         e + 2↑24 × 63 is the identifier form of a          ×
                                         constant label e;
     e : = P + 2↑23 × 127;
     SOe(a);
     SO(0)
     end SOl;
```

```
                                        comment
procedure SOm(F);                       stores word in list L on address SO - 1;
begin
SO : = SO - 1;  st[SO] : = F;
if  P + 2 > SO then SOg;                compare procedure SO
end SOm;
```

	comment
<u>procedure</u> SOn(F);	stores contra-declaration of identifier st [S] (cf. S4L23) in list I, with P + F as internal equivalent;
<u>begin</u>	
e : = P + F;	
SOe (st[S] + 2↑32);	×
SO(O);	Thus a preliminary zero is stored in the place of an instruction which is inserted later
<u>end</u> SOn;	

	comment
<u>procedure</u> SOp;	contra-declares name of function having no parameters;
<u>begin</u>	
a : = e;	which is the contra-identifier of the function name;
e : = P + 2↑23 × 31;	×
SOe(a);	
SO(0)	
<u>end</u> SOp;	

comment

```
procedure SOr;                              inverts gression bit of instruction d;
begin
g : = d or 2↑26;                                                                        ×
if g = 0 then
begin
a : = d or 2↑26 × 126 ;                                                                 ×
if a = 2↑26 × 64 or a = 2↑26 × 68                                                      × ×
or a = 2↑26 × 70 or a = 2↑26 × 74
or a = 2↑26 × 80 or a = 2↑26 × 82
or a = 2↑26 × 88
then g : = - 2↑26                                                                       ×
end                                         Then operator (cf. table 1A) is not commutative.
                                            When the operator is commutative, the bit d6 = 0
                                            is not replaced by 1;

d : = d - g
end SOr;
```

	comment	
<u>procedure</u> SOs;	This procedure makes a contra-declaration of identifier st[S] according to the value p = $\overline{0\ 011111111}$ listed in table 2A together with an explanation. The procedure is invoked only on S2aL19 and S5aL27 and S2aL19;	
<u>begin</u>		
a : = st[P];		
e : = 2↑23;		×
<u>if</u> a ≠ 0 <u>then</u> e : = e <u>or</u> a;	Then e = 0 indicates that a is an ar2 instruction (cf. S6aL61) referring to a formal parameter;	
<u>if</u> e = 0 <u>then</u> a : = a + 2↑23 + 2↑26 × (108-125);	Cf. VERIFY and ar2 in table 1D;	×
SO(a);	Thus either a VERIFY instruction is stored, or an ar2 instruction or 0 is restored;	
SOn(2↑23 × 255 - 1);	Then P is again increased, and a zero is stored on address P - 1, while the contra-declaration made refers to the address P - 2	×
<u>end</u> SOs		

comment

Progressive after-actions of opening symbols (cf. table 1 B).

S1L6;

<u>switch</u> sw : =
S5aL1, S6aL1, S3bL1, S9aL1, S7aL1,
S8aL2, S2aL1, S8dL0, S8bL2;

Regressive after-actions of opening symbols.

S1L21;

<u>switch</u> SW : =
S5aL3, S6aL5, S3bL2, S9aL2, S7aL2,
S8aL1, S2aL13, S8dL1, S8bL1;

The entries correspond to the after-actions of
([<u>begin</u> : = <u>for</u> <u>go</u> <u>to</u> <u>if</u> <u>procedure</u> <u>switch</u>;

	comment
begin	Compound statement entry.
	The constants
	LO, PO, QO, and QOO of tabel 1E
	are the initial values of the address variables.
entry: P : = PO + 1;	When, on S3bL8, the translation finishes, the + +
	first word of the object programme is inserted on
	address PO;
q : = QO; T : = T1: = QOO–2xh;	
T2 : = QOO;	
SO : = LO;	
S : = 1 + SO;	
st[SO – 1] : = R : = 0;	
go to S6aL11;	Then Saccent : = 8191,
	st[SO]: = g : = mark : = 0.
	As the opening symbol begin occurs in front
	of any text, input goes to compound S3.
	Procedure SOh cannot do any harm, for
	st[S – 2] = 0 differs from the internal value 322
	of begin.
end entry;	

	comment
begin	Compound statement S1. After reading a delimiter f which is either an operator (table 1A) or a separation or closing symbol (table 1B), procedure input goes to here;
S1L1: SOa;	Then c = st[S - 3], d = st[S - 2], e = st[S - 1], a = 32 × rank r_d of delimiter d, b = 32 × rank r_f of delimiter f;
S1L2: **if** Saccent = S **then** **go** **to** S1L4;	cf. S3aL4 and S5aL5. Then the identifier or constant preceding delimiter f in the text need not be examined;
if g < 0 **then** g : = 0;	Then identifier st[S] = e - 2^{32} represents an array (cf. S4L22). As this significance of the identifier is nonsens here, it must be non-local, and another local declaration of the identifier may be expected later. Only exception: identifier is actual parameter, representing array. Then g : = 0 does no harm;
if e = 0 **and** g = 0 **then** **go** **to** S1L3;	Then delimiter f is preceded in the text by the 0-identifier (there is no identifier and no constant);
if f ≠ 32 × 5 + 2↑26 × 97 **then** **go** **to** S1L4;	×
S1L2a: wrong;	Operator **not** is preceded by an identifier or constant. S1L2;
S1L3 : **if** a < 128 **then**	
S1L3a: wrong;	Then an identifier or constant has been omitted behind the arithmetic operator d;
if f = 32 × 3 + 2↑26 × 72 **then** **go** **to** input;	for skipping + in a = + b etc.; ×

	comment
if d = 322 then go to S1L21;	for a dummy statement. Then the after-action of d with f = semi-colon or end is considered to be regressive;
if f = 32 × 5 + 2↑26 × 97	for operator not;
then go to S1L4;	×
if f ≠ 32 × 3 + 2↑26 × 74 then	
S1L3: wrong;	Then an identifier or constant has been omitted; ×
f : = 32 × 3 + 2↑26 × 96;	In a = − b etc. − is treated this way.
	S1L2;
S1L4: if b < a then go to S1L26;	Then reading continues and d is a regressive operator or opening symbol. ×
	Otherwise reading is interrupted: ;
S1L5: if Saccent = S then go to S1L21;	Then d is either a regressive operator, or an opening symbol which is regressive with respect to the after-action with f (cf. S5aL5 and S3aL4).
	In general, however, an operator or opening symbol d which is translated first after interrupting the reading, is progressive: ;
S1L6: if a = 320 then go to sw [d−319];	for progressive after-actions of opening symbols;
S1L7: if Saccent + 2 < S then SOd;	Then object programme must be provided with storing and extracting a partial result;
S1L8: a : = d or 2↑23 × 5;	Then a is either = 0 or ×
	a = $\overline{0\ 000000t01\ 0...0}$ and c a programme const. whose representation is indicated by the bit t. For g and e it is analogous(cf. S1L26 and S1L29);
S1L9: d : = d − a + g;	which is thus associated to e instead of c;
S1L10: if Saccent + 2 = S then go to S1L11;	Otherwise object programme must be equipped with filling the accumulator:;
g : = a;	
a : = e; e : = c; c : = a;	c and e interchange their values;

	comment	
SOk; e : = c;	Thus previous value of e is restored.	
	S1L10, S1L25;	
S1L11: **if** d **or** 2↑23 ≠ 0 **then** **go** **to**	Then instruction d is ready and e is a programme	
S1L15;	constant to be used by instr. d;	×
S1L12: **if** e > 0 **then** **go** **to** S1L14;	Then e is the internal equivalent of either a simple variable or formal parameter.	
	Otherwise e − 2^{32} is an identifier (cf. S4L22) which, in this position, can only be the name of a function having no parameters: ;	
if d ≠ 32 × 3 + 2↑26 × 96 **and**	Otherwise d is an operator **not** or − such as	×
d ≠ 32 × 5 + 2↑26 × 97	does not, in operation time, require to store	
then **go** **to** S1L13;	a previous partial result: ;	×
SOp;	Compare S1L13;	
go **to** S1L22;	S1L12;	
S1L13: SO (partres);	for storing previous partial result;	
SOp;	Thus a contra-declaration of identifier c − 2^{32} is made according to p = 0 000011111 which constant is listed in table 2A together with an explanation;	
e : = 2↑23 × 507;	For, in operation time, instruction d must	×
	extract a partial result;	
SOr;	Thus the gression bit d_6 is inverted. When, for example, function − p × q + ... is translated, then operator − is regressive according to the general rule. Thus p × q is translated first. Before, in operation time, function may be invoked, the object programme must at first store the partial result accu which is p × q. Therefore the minus instruction is to be translated in the form accu : = accu − (partial result) which is progressive instead of regressive.	

	comment	
	S1L12;	
S1L14: e := e - 2↑26 × 63 + (d or 2↑26 × 127);		×
go to S1L16;	S1L11;	
S1L15: SO(d);	S1L14, S1L22, S1L23;	
S1L16: SO(e);	Thus instruction or programme constant is stored in the object programme. S3bL14, S7aL18, S8aL3, S9aL2;	
S1L17: Saccent := S := S - 2;	Operator st[S] has been translated and will be overwritten now;	
S1L18: SOa;	cf. S1L1;	
S1L19: if b < a then go to S3L5;	Then translator again proceeds to reading. Otherwise translation continues (cf.S1L4). S1L5, S1L3;	
S1L21; if a = 320 then go to SW[d - 319];	for regressive after-actions of opening symbols (cf. S1L6). S1L12;	
S1L22: if d ≠ 32 × 5 + 2↑26 × 97 then go to S1L23;		×
e := take inversion;	That code instruction of table 1E is the regressive version, and extract inversion of table 1C is the progressive version of operator not;	
go to S1L16;	S1L22;	
S1L23: if d ≠ 32 × 3 + 2↑26 × 96 then go to S1L24; e := take complement;	That code instruction of table 1E is the regressive version, and extract complement of table 1C which has been introduced at the end of S1L3 is the progressive version of operator -, when - is preceded by the 0-identifier;	×
go to S1L16;	S1L23;	

	comment
S1L24: SOr;	That procedure sets the gression bit d_6 which is still = 0, to 1, for operator d is regressive;
e : = c;	If that is a programme constant, the indication of its representation is already recorded in d;
S1L25: **go** **to** S1L11;	S1L4;
S1L26: SOc (f + g);	with either g = 0, or g = $\overline{0\ 000000t01\ 0...0}$, the bit t indicating the representation of e which is a programme constant (cf. S1L29). if, at the next action of this compound S1, operator st [S - 2] just listed in L is found to be progressive, the g-part is removed from it and replaced by the next value of g, on S1L9. S3L5, S5aL6, S6aL11;
S1L27: mark : = 0;	cf. table 4C.
S1L28: st [S - 1] : = 0;	S3aL4, S3cL6, S5L9, S6aL31, S6aL61, S8L2, S8dL8, S9L4; If the translator proceeds to S4L23 or S4aL2, this 0 is replaced by something else.
S1L29: g : = 0; **go** **to** input;	S4L21; After reading a constant, procedure input goes to compound S4a with g = $\overline{0\ 000000t01\ 0...0}$ in which bit t indicates the representation of the constant f read. On S4L22, occasionally an ar1 instruction is assigned to variable g as value;
end S1;	

```
                                    comment
    begin                           Compound statement S2.
                                    Pre-action of the opening symbol  if;
S2L1: f : = 326                     Cf.  if in table 1B;
      SOh;                          Cf.S3L2;
S2L2: SOc(mark);
      go to S3L3;                   Then, for translating an if-expression, there is
                                    listed in L:
                                    st[S - 4] = mark  (cf. table 4C),
                                    st[S - 3] = 0,
                                      from 1st after-action: minus address P'
                                      where to store a test instruction,
                                      from 2nd after-action: plus address P"
                                      where to store a pass instruction.
                                    st [S - 2] = value 326 of if,
                                    st [S - 1] = 0.
                                    Signal mark is set to 0. The listed value,
                                    st [S - 4], is re-introduced on the 1st and 2nd
                                    after-actions of  if;

    end S2;
```

	comment
begin	Compound statement S2a.
	After-actions of the opening symbol if.
	Delimiter f is either then or else or the closing symbol of a conditional expression or statement E. On S1L1, the value 0 or -P' or +P", mentioned in compound S2, has already been assigned to variable c.
	Progressive:
	The expression, preceding delimiter f, is a single identifier or constant i
	S1L6;
S2aL1: d : = st [S - 4];	being the value of mark retained by compound statement S2;
S2aL2: if c × d ≠ 0 then go to S2aL4;	Then the 2nd or 3rd after-action of if is beginning, f being else or the closing symbol, and, in addition, d is = 1 or 2.
	Then i may be a label.
	Otherwise i is no label: ;
S2aL3: SOk;	Thus object programme is equipped with filling the phantom accumulator;
go to S2aL20;	S2aL2;
S2aL4: if d = 2 then go to S2aL17;	Then i is a label (or a formal parameter that may only represent a label), and E is a designational expression.
	Otherwise expression E is an actual parameter;
S2aL5: if g ≠ 0 then go to S2aL11;	Then i is the constant e.
	Otherwise i is an identifier : ;
S2aL6: if e < 0 then go to S2aL9;	Then i = e - 2^{32} is found to be no simple variable and no formal parameter (cf.S4L22).

	comment	
	Identifier i is found to be a simple variable or formal parameter with the internal equivalent e:;	
S2aL7: **if** e - 2↑23 × 505 > 0 **then** **go** **to** S2aL18;	Then i is a simple variable, and an extract-normally instruction is added to the object programme. Otherwise i is a formal parameter:;	
S2aL8: g : = e + 2↑23 + 2↑26×(110-63);	which is a verify instruction (cf. table 1D).	×
	In operation time, the bit $g_9 = 1$ of instruction g makes the test on S10L13 fail, and the interpreter proceeds to S10L140. There the instruction X whose formal parameter represents the actual parameter in which expression E is contained, is examined: If X is no jump instruction, then the formal parameter i to which the verify instruction refers, cannot represent a label so that the verify instruction is to be interpreted as the extract normally instruction referring to i.	
	S2aL6;	
S2aL9: SO(g);	S2aL18;	
S2aL10:g : = st [S] + 2↑32;	being the contra-identifier of i (cf. S4L23);	×
e : = P + 2↑23 × 511 - 1;		×
go **to** S2aL12;	The contra-declaration g, e to be stored now is made according to p = $\overline{0\ 111111111}$ which constant is listed in table 2A together with an explanation. It refers to the address x = P - 1, where the object programme contains either the mentioned verify or extract normally instruction or 0.	
	S2aL5;	

	comment	
S2aL11: S0k;	Then st[P - 2] is the extract normally instr. which, in operation time, extracts the programme constant st [P - 1] = i;	
if e < 0 **or** 2↑24 ≦ e	Then constant i is not suitable for being a label.	
then **go** **to** S2aL20;	Constant i may occur as a label:;	×
g : = e + 2↑24×63 + 2↑32;	which is the contra-identifier of a constant label i;	
e : = P + 2↑31 × 3 - 2;	The contra-declaration g, e to be stored now is made according to p = $\overline{1\ 100000000}$ which constant is listed in table 2A together with an explanation. It refers to the address x = P - 2, where the object programme contains the extract normally instruction. The internal equivalent e is negative.	×
	S2aL10;	
S2aL12: S0e(g);	thus the contra-declaration is stored now;	
go **to** S2aL20;	Regressive:	
	The expression I preceding delimiter f is non-trivial (no identifier and no constant).	
	S1L21;	
S2aL13: d : = st[S - 4];	cf. S2aL1;	
S2aL14: **if** c × d = 0 **then** **go** **to** S2aL20;	Then expression I cannot be a switch designator (cf.S2aL2) and has already been translated.	
	Otherwise I = i [E] is either a subcripted variable or a switch designator. The subcript E has been translated in compound S6a;	
S2aL15: P : = P - 1;	Then st[P] is either the ar2 instruction referring to i, in which case i has been found to be either an array identifier or a formal parameter, or 0 (cf. S6aL61);	

	comment
S2aL16: <u>if</u> d = 1 <u>then</u> <u>go</u> <u>to</u> S2aL19;	Then i [E] is contained in an actual parameter. i [E] is a switch designator thus i a switch identifier:
	S2aL4;
S2aL17: SOl;	Label or switch identifier is contra-declared;
<u>go</u> <u>to</u> S2aL20;	S2aL7;
S2aL18: SOk;	
<u>go</u> <u>to</u> S2aL10;	S2aL16;
S2aL19: SOs;	S2aL3, S2aL11, S2aL12, S2aL14, S2aL17;
S2aL20: mark : = d;	At the after-actions with f = <u>then</u> or <u>else</u> signal mark is restored to the value retained by compound S2;
S2aL21: e : = P;	
S2aL22: <u>if</u> c < 0 <u>then</u> <u>go</u> <u>to</u> S2aL28;	for the 2nd after-action;
S2aL23: if c ≠ 0 <u>then</u> <u>go</u> <u>to</u> S2aL31;	for the 3rd after-action.
	1st after-action:;
S2aL24: e : = - e;	S2aL30;
S2aL25: st [S - 3] : = e;	
S2aL26: SO(0);	At the next after-action, this 0 is replaced by a test or pass instruction;
S2aL27: <u>go</u> <u>to</u> S1L28;	S2aL22;
S2aL28: <u>if</u> f ≠ 326 <u>then</u> e := e - 1;	Cf. <u>else</u> in table 1B;
S2aL29: st[-c]: =e+1+2↑23+2↑26×111;	Cf.test in table 1D;
S2aL30: <u>if</u> f ≠ 326 <u>then</u> <u>go</u> <u>to</u> S2aL32;	Then the separation symbol <u>else</u> is not present;
<u>go</u> <u>to</u> S2aL25;	<u>else</u> is present.
	S2aL23;
S2aL31: st[c]:= e+2↑23+2↑26×121;	Cf.pass in table 1D.
	S2aL30;
S2aL32: S : = S - 2;	Thus, at the last after-action of <u>if</u>, mark is set to 0
<u>go</u> <u>to</u> S8aL3	
<u>end</u> S2a;	

	comment
	Compound statement s3.
	Pre-action of the opening symbol begin;
S3L1: f := 322;	Cf. begin in table 1B;
D := 2↑23 × 1023	Cf. D in table 2;
S3L2: S0h;	Perhaps opening symbol begin immediately follows a block head. Then procedure S0h completes the dynamic introduction of the block.
	S2L2, S6L9;
S3L3: st[S - 1] := 0;	S7L2, S9L10;
S3L4: Saccent := 8191;	S1L19, S5L4, S6L19;
S3L5: S0c(f);	After pre-action of opening symbol, reading
go to S1L27;	continues.
	For translating a compound statement there is listed in L:
	st[S - 3] = 0,
	st[S - 2] = value 322 of begin,
	st[S - 1] = 0
end S3;	

```
        begin                                    comment
                                                 Compound statement S3a.
                                                 After reading a declarator or specificator (table 2),
                                                 procedure input goes to here;
 S3aL1:D : = D or f;                             Thus declaration is replaced by its logical product
                                                 with f;
        if (D or 2↑28) = 0 then go to            Then  f  is a specificator or specifying declarator;
        S3aL4;
S3aL2:  S0a;                                     for extracting opening symbol from list L;
        if d ≠ 322 then
S3aL2a: wrong;                                   Then declarator occurs in the wrong place;
S3aL3:  if c ≠ 0 then go to S3aL4                Then block has already been prepared in L by a
                                                 previous declarator.
                                                 First declarator of a block:;

        st[S - 1] : = q;
        S0c(P);
        S0(0);
        st[S - 1] : = T;
        S0c(322);                                S3aL1, S3aL3, S3bL2, S4L30, S5L2;
S3aL4:  Saccent : = 8;                           For translating a block there has been listed in L:
        go to S1L28;                             st[S - 7] = 0, which 0 may be replaced through
                                                 procedure S0i by an address to inform procedure S0h,
                                                 st[S - 5] = q', being the highest address occupied by
                                                 a local variable,
                                                 st[S - 4] = address P' where the block has its object
                                                 programme beginning,
                                                 st[S - 3] = T', being the highest address in the list I,
                                                 occupied by any identifier declared in the block,
                                                 st[S - 2] = value 322 of opening symbol begin,
                                                 st[S - 1] = 0

        end S3a;
```

begin	comment
	Compound statement S3b
	After-actions of opening symbol begin
	Delimiter f last read is either comma or semi-colon or end.
	Progressive:
	S1L6;
S3bL1: SOh;	A block head may be followed by a procedure statement consisting of a single procedure identifier (cf.S3L2);
SOn(2↑23 × 31);	A contra-declaration of the procedure identifier is made according to p = $\overline{0\ 000011111}$ which constant is listed in table 2a together with an explanation. ×
	Regressive:
	S1L21;
S3bL2: if f ≠ 330 then go to S3bL3;	Then f is no comma (cf. table 1B);
If D > 0 then go to S3aL4;	Then comma occurs in list of identifiers to be declared;
S3bL2a: wrong;	Comma occurs in a block as separator of statements.
	S3bL2;
S3bL3: T1: = T;	This bound is observed by procedure SOf;
S3bL4: D : = 2↑23 × 1023;	as happens also on S3L1; ×
S3bL5: if f = 332 then go to S6aL11;	for reading next declaration or statement of compound statement or block, which is subsequent to semi-colon f;
if f ≠ 322 then S3bL5a: wrong;	Then a declaration or statement of a compound statement or block is followed by a delimiter, differing from semi-colon and end.
	After-action of begin with end:
	On S1L1, st[S - 3], as left by either compound S3 or S3a, has been assigned to variable c;
S3bL6: if c ≠ 0 then go to S3bL9;	Then a block is closing.
	A compound statement is closing;

	comment
e : = Q00-2xh;	which is the highest value T can have; +
if S = S0 + 3 then go to S3bL13;	Then the text is a compound statement instead of a block. Labels may occur in it.
	S3$_b$L14;
S3bL7: if S > S0 + 3 then go to input;	input, in this comment situation, looks for the next delimiter f which is equal to either semi-colon or end or else, then going with f to S1L17. Then the delimiter preceding begin in the list L, is going to after-act with f.
	The whole text has been read:;
S3bL8: S0(0);	Thus last word of object programme is 0, but may be a jump to any stop of the machine;
st[P0] : = P;	Thus 1st word of object programme is address where working space begins; +
S3bL8a:wrong;	Object programme is ready!
	S3bL6;
S3bL9: S : = S - 2; S0a; S : = S - 2;	Now st[S - 4] is the delimiter preceding begin in the list L. For c = q' etc. cf. compound S3a;
S3bL10: st[d] : = q+R+2↑23+2↑26×127;	Thus, in object programme of block, an adjust instruction is the first word; ×
S3bL11: S0(c+R+2↑23+2↑26×119);	Thus in object programme of block, a restore instruction (cf.table 1D)is the last word. ×
	S8dL7;
S3bL12: q : = c;	Thus value of q, which was present when first declarator of block was read, is restored.
	S3bL6;
S3bL13: a : = T;	On S3bL3, the value of T was also assigned to T1.
	S3bL15, S3bL18, S3bL20, S3bL34;
S3$_b$L14: if a ≠ e then go to S3bL15;	Then the contra-identifier occurring next on the addresses a+2, a+4, ...e is looked up.

	comment
	All contra-identifiers have been considered:;
T1 : = T : = e;	Then in the list I t is the highest address, occupied by an identifier which is local to the translated block;
T2 : = Q00;	Cf. table 1E;
if st [S - 3]<0 **then** **go** **to** S3bL7;	Then it is a block that closes, st[S - 3] having been made negative by procedure S0h after reading the block head.
	A procedure is closing:;
f : = 332;	Thus semi-colon f of compound S8d is restored;
go **to** S1L17;	as ultimately happens too with a block (cf. S3bL7).
	S3bL14;
S3bL15: a : = a + 2;	Then st [a] is a normal (positive) identifier.
f : = st [a] + 2↑32;	st [a] is a contra-identifier:;
if f < 0 **then** **go** **to** S3bL14;	
S3bL16: T2 : = e; S0f;	Then identifier f corresponding to contra-identifier st [a] is looked for on the addresses T+2, T+4, ...T2;
b : = st[a - 1];	being the internal equivalent of the contra-identifier;
S3bL17: **if** d > 0 **then** **go** **to** S3bL21;	Then procedure S0f has found identifier f on address c, d = st[c - 1] being the internal equivalent (cf. table 2) of f. Then contra-identifier st [a] can be satisfied.
	Identifier f has not been found so that the contra-identifier remains unsatisfied:;
S3bL18: e : = T2;	Thus the value of e as it was on S3bL16, is restored.

	comment
if S > S0 + 3 then go to S3bL19;	Then the translator does not yet see the end of the text, thus may regard the unsatisfied contra-identifier as to be non-local to the closing block or procedure.
	Otherwise the text, either a block or a compound statement, is closing:;
if b < 0 then go to S3bL14;	Then f $- 2^{24} \times 63$ is a programme constant, that, because of its position within an actual parameter, was expected to be perhaps a reference to a label, but is found now to be not;
S3bL18a: wrong;	Any identifier of the text has not been defined. S3bL18;
S3bL19: st[a - 1] : = st [e - 1];	
st[a] : = st [e];	
st[e - 1] : = b;	
st[e] : = f + 2↑32;	Thus 2 declarations are interchanged; ×
a : = a - 2;	The non-satisfied contra-identifier is considered to
e : = e - 2;	be no longer local. Thus e is again the highest address where a local identifier is listed in I;
S3bL20: go to S3bL14;	S3bL17;
S3bL21: T1 : = c;	which is used on S3bL33;
S3bL22: e : = d or (2↑23 - 1);	being address + $2^{13} \times$ rank; ×
c : = b or 8191;	being the address where either an instruction or the main word of a parameter key is to be inserted. For internal equivalent b cf. table 2A;
b : = b - c;	Then b = 0 indicates, that the contra-declaration refers to the key of an actual parameter that is a single identifier.
	There must be formed either d = main word, and

```
                                          comment
                                          e = by-word of the key (cf. table 3) of an actual
                                          parameter that is an identifier or a constant,
                                          or d = instruction;
S3bL23: if b ≠ 0 then go to S3bL35;       for a constant parameter or an instruction.
                                          Parameter is an identifier:;
S3bL24: if 2 × d < 0 then go to S3bL29;   Then parameter represents either a formal parameter
                                          or a simple variable (cf. table 2);
S3bL25: if 4 × d < 0 then go to S3bL28;   Then parameter represents an array;
S3bL26: if 8 × d < 0 then go to S3bL30;   Then parameter represents a label or switch.
                                          Parameter represents a procedure:;
S3bL27: d : = e or (2↑23 - 2↑13);         being 2^13 x rank of procedure identifier;     ×
        e : = e + 1 - d;                  being 1 + address where procedure has its object
                                          programme beginning;
        d : = d + 2↑23 × 57;
        go to S3bL31;                     S3bL25;
S3bL28: e : = g - e - 2↑23 - 2↑26×126;    Cf. ar1 in table 1D;
        go to S3bL31;                     As procedure SOf assigned an ar1 instruction
                                          (cf. S6L8) to variable  g  as value, e  is a
                                          difference of addresses now.
                                          S3bL24;
S3bL29: if (d or 2↑23) = 0 then           Then  actual parameter represents a formal     ×
        d : = e + 2↑26 × 3;               parameter
                                          S3bL26;
S3bL30: e : = 0;                          Thus, for a parameter representing a simple variable
                                          or formal parameter or label or switch, the by-word
                                          is cleared, which is facultative.
                                          S3bL27, S3bL28;
S3bL31: st[c - 1]: = e;                   Thus by-word is stored.
                                          The key main word of a parameter, representing a simple
```

	comment
	variable or array or label or switch i, is equal to the internal equivalent of i.
	S3bL47;
S3bL32: st[c] : = d;	Thus key main word or instruction is stored.
	S3bL42, S3bL43;
S3bL33: SOf;	Cf. S3bL21;
if d > 0 **then**	
S3bL33a:wrong;	Any identifier has been defined twice in the same block or , or in the same procedure outside the blocks contained in it;
S3bL34: T1 : = T;	Thus the value of T1 as it was on S3bL14, is restored;
e : = T2;	Ditto;
go to S3bL14;	Satisfaction of contra-identifier st[a] is ready.
	S3bL23;
S3bL35: d : = d - e;	
g : = e **or** 8191;	Then g is the address taken from the declaration. Now the kind of the object referred to by the contra-declaration (cf. table 2A) is examined. It must be in accordance with that of the object referred to by the corresponding declaration (cf. table 2);
S3bL36: **if** b < 0 **then** **go** **to** S3bL48;	Then a constant occurring in an actual parameter (or being an actual parameter) is found to be used as a label too - the declaration of which having been found;
S3bL37: **if** 2 × b < 0 **then** **go** **to** S3bL43;	Then st[c] is a verify or extract normally instruction or 0, occurring in the object programme of an actual parameter (cf. S2aL6);
S3bL38: **if** 4 × b < 0 **then** **go** **to** S3bL42;	Then st[c] is a VERIFY or ar2 instruction or 0, occurring in the object programme of an actual parameter (cf. S2aL15);

```
                                          comment
S3bL39: if 8 × b < 0 then go to S3bL45;Then a jump instruction is to be inserted in the
                                        object programme of a switch or go to statement,
                                        identifier  f  having been contra-declared on S8aL2
                                        for example.
                                        In the object programme of a function designator or
                                        procedure statement, the 1st word must be adjusted,
                                        identifier  f  having been contra-declared on
                                        S5aL6 or S1L13 or S3bL1 for example;
S3bL40: if d ≠ 2↑26 × 63 then go to     Then identifier  f  may only have the significance
        S3bL41;                         of a procedure identifier.                          ×
                                        Identifier  f  has been found in the significance of
                                        a formal parameter:;
        g := if 16 × b < 0 then         Cf. extract normally and prostat in tables 1 C    - ×
        2↑26 × 120 else 2↑26 × 98;      and 1 D;
        go to S3bL47;                   S3bL40;
S3bL41: if d > 2↑29 then                                                                    ×
S3bL41a:wrong;                          A procedure statement or function designator
                                        invokes nonsense;

        if 16 × b > 0 then go to
        S3bL44;
        if st[g + 1] > 0 then go to
        S3bL44b;
S3bL41b:wrong;                          A function designator or procedure statement having
                                        actual parameters, invokes a function or procedure
                                        having no formal parameters.
                                        S3bL38;
S3bL42: if d = 2↑26 × 63 then go to     Then identifier  f  is a formal parameter;          ×
        S3bL33;
        if 2 × d < 0 then
```

	comment
S3bL42a:wrong;	A subcript has been attached to a simple variable;
if 4 × d < 0 then go to S3bL33;	Then f is an array identifier;
if 8 × d < 0 then go to S3bL46;	Then st[c] is replaced by a jump instruction;
S3bL42b:wrong;	A subscript has been attached to a procedure identifier. S3bL37;
S3bL43: if 2 × d < 0 then go to S3bL33;	Then identifier f is either a simple variable or a formal parameter and st[c] is not changed;
if 4 × d < 0 then	
S3bL43a:wrong;	An array identifier occurring within an expression is not followed by a subscript;
if 8 × d < 0 then go to	Then f is a switch identifier or a label;
S3bL46;	Identifier f is a function or procedure name. The function designator or procedure statement has no actual parameters: S3bL41;
S3bL44: if st[g + 1] > 0 then	
S3bL44a:wrong;	A function designator or procedure statement having no actual parameters, invokes a function or procedure having formal parameters. S3bL41;
S3bL44b:e : = JO;	Cf. table 1E;
go to S3bL47;	On address c , a code instruction is stored which, in operation time, jumps to address g where the object programme of the function or procedure begins. S3bL39;

+

```
                                              comment
S3bL45: if b = d then go to S3bL46;          Then f is a label or switch identifier.
                                             Identifier f is a formal parameter:;
        g : = 2↑26 × 122;                    Cf. jump in table 1D;                          ×
        if d = 2↑26 × 63 then go to
        S3bL47;                              Then f is indeed a formal parameter;           ×
S3bL45a:wrong;                               nonsense, referred to in a designational
                                             expression.
                                             S3bL42, S3bL43;
S3bL46: g : = 2↑23 + 2↑26 × 122;             Cf.jump in table 1D.                           ×
                                             S3bL40, S3bL44, S3bL45, S3bL48;
S3bL47: d : = e + g;
        go to S3bL32;                        for storing instruction or key main word.
                                             S3bL36;
S3bL48: g : = if 2 × b < 0
        then 2↑23 + 2↑26 × 109               Cf.Verify in table 1D and key main word of
        else 2↑23 × 89;                      constant parameter in table 3;                 ×
        go to S3bL47
        end S3b;
```

```
                                           comment
     begin                                 Compound statement S3c.
                                           Colon goes to here from input;
S3cL1: if(D or 2↑24) ≠ 0 then go to
       S3cL2;                              Then colon declares the label preceding it      ×
                                           in the text.
                                           Otherwise pattern  D  contains the type
                                           indication of an array declaration, colon being the
                                           separation symbol between a lower and upper bound:;
       f : = 331;                          Cf. colon in table 1B;
       go to S1L1;                         S3cL1;
S3cL2: SOh;                                Cf.S3L2;
S3cL3: if g > 0 then go to S3cL7;          Then label is an integer
                                           (compare procedure SOl);

       if e = 0 then
S3cL3a:wrong;                              Label has been omitted;
S3cL4: f : = st[S];                        Cf. S4L23.
                                           S3cL8;
S3cL5: e : = P + R + 2↑23 × 127;           Cf. label in table 2;                           ×
       SOe(f);                             Thus declaration of label is listed;
S3cL6: SO (q + R + 2↑23 + 2↑26×119);       Cf. restore in table 1D;                        ×
       go to S1L28;                        S3cL3;
S3cL7: if e < 0 or 2↑24 < = e then                                                         ×
S3cL7a:wrong;                              Cf. table 5;
S3cL8: f : = e + 2↑24 × 63;                being the internal value of constant label  f;  ×
       go to S3cL5
       end S3c;
```

```
                                             comment
    begin                                    Compound statement S4.
                                             After reading an identifier  f, procedure input
                                             goes to here.
S4L1: if D < 0 then go to S4L22;             Then identifier  f  must be looked for in the
                                             list I.
                                             Identifier  f  must be declared or specified
                                             (cf. table 2):;
S4L2: if 2 × D < 0 then go to S4L9;          Then identifier  f  is declared a simple variable;
S4L3: if 4 × D < 0 then go to S4L11;         array identifier;
S4L4: if 8 × D < 0 then go to S4L37;         switch identifier;
S4L5: if 16 × D < 0 then go to S4L24;        procedure identifier;                          ×
S4L6: if (D or 2↑23) ≠ 0                     Then f is either a value parameter, or is to be specified.
                    then go to S4L32;        Identifier  f  is a formal parameter to be listed in I:;
S4L7: P := P - 1;
      e := st[P];
      SO(D); SO(e);                          Then st[P - 2] = D, and st[P - 1] = -2^13 × rank of
                                             procedure (cf. S4L28);
S4L8: e := q + 2↑26 × 63;                                                                   ×
      q := q - 2;
      go to S4L19;                           Relative addresses q + 3 and q + 2 are reserved for
                                             the main - and by-word of the parameter key.
                                             S4L2;
S4L9: if 8 × D < 0 then go to S4L18;         Then variable is not own.
                                             Variable is own:;
S4L10: SOi;                                  Cf. S4L24;
      go to S4L15;                           S4L3;
S4L11: c := st[S - 1];
S4L12: if 8 × D < 0 then go to S4L16;        Then array is not own.
                                             Array is own:;
```

	comment
S4L13: <u>if</u> c ≠ 0 <u>then</u> <u>go</u> <u>to</u> S4L15;	Then f is not the 1st identifier of the list.
	1st identifier of the list:;
S4L14: SOi;	Cf. S4L24;
st[S − 1] : = P;	Then st [S − 1] is no longer = 0.
	Compound S6a will insert pre-value [a] of a = 1st array of declaration (cf. explanation) on address P.
	S4L10, S4L13;
S4L15: e : = P + D + 2↑29;	Then the bit e_3 of the internal equivalent = 1, × as it is in the case of a variable or array being not <u>own</u>;
SO(0);	A place for the simple variable or the pre-value of the array is reserved in the object programme;
<u>go</u> <u>to</u> S4L20;	S4L12;
S4L16: <u>if</u> c ≠ 0 <u>then</u> <u>go</u> <u>to</u> S4L18;	Then f is not the first identifier of the list.
	1st identifier of the list:;
S4L17: SOh;	Thus the space of <u>own</u> variables and - arrays and of the object programmes of procedures and switches, which eventually precede the non-<u>own</u> array declaration, is bridged by inserting a pass instruction in the dynamic introduction of the block;
st[S − 1]: = q;	The relative address q is reserved for the pre-value [a] (cf. S4L14), that is an internal (non-declared) variable of the object programme.
	S4L9, S4L16;
S4L18: e : = q + D; q : = q − 1;	Thus internal equivalent e of identifier f contains the address reserved.
	S4L8;
S4L19: e : = e + R;	cf. S4L26.
	S4L15;

	comment
S4L20: T1 : = T;	
T2 : = st[S - 3];	That is the address T' mentioned in compound S3a;
S0e(f);	Thus declaration is listed in I;
S0f;	
if d > 0 **then** **go** **to** S3bL33a;	Then identifier f is defined twice in the same block or procedure heading.
	S4L36;
S4L21: T2 : = Q00;	Cf. table 1E;
go **to** S1L29;	S4L1;
S4L22: S0f;	Identifier f is looked for on addresses T1 + 2, T1 + 4, ... T2, as explained in S0f;
S4L23: st[S - 1] : = e; st[S]: = f;	
go **to** input;	Because of the possible necessity to make a contra-declaration, identifier f remains available on address S, until the next delimiter is stored there in the list L.
	S4L5;
S4L24: S0i;	As the object programme of the procedure which is going to be translated now, is to be passed by the dynamic introduction of the block, a location is reserved for the pass instruction needed, unless this reservation has already been made earlier on S4L10, S4L14, S4L24 or S4L37;
S4L25: e : = P + R + D;	Thus the address P where the procedure has its object programme beginning, is contained in the internal equivalent e of procedure identifier f;
S4L26: R : = R + 2↑13;	R is cleared in compound entry, and decreased on S8dL3. $r = 2^{-13} \times R$ is the rank of the procedure, r - 1 is the rank of the procedure identifier;

	comment
S4L27: SO(X);	Cf. table 1E; +
S4L28: SO(-R);	The patterns of eventual formal parameters are inserted between instruction X and constant -R so that the latter must be shifted then (cf.S4L7);
S4L29: st[S - 1]: = q;	Cf. procedure in table 1B.
q : = Q0 - 1 ;	S4L39; + ×
d : = 327;	
S4L30: SOc(P);	
st[S - 1]: = T;	
SOc(d);	
SOe(f);	Cf. S4L20;
S4L31: go to S3aL4;	For translating a procedure, there has been listed in L: st[S - 5] = value q' of q which was present when declarator procedure was read, st[S - 4] = 2 + address P' where instruction X has been stored, st[S - 3] = address T' where procedure identifier f has been listed in I, st[S - 2] = value 327 of opening symbol procedure, st[S - 1] = 0. to be compared with the information listed by compound S3a. S4L6;
S4L32: e : = P - st[S - 4];	being the number of the formal parameters (cf.P' in S4L31);
T2 : = R + e + e;	On that address the 1st parameter identifier is listed in I;
SOf;	Thus identifier f is looked up in parameter list;

	comment
if d < 0 then	
S4L33: wrong;	Then specified identifier does not occur in the parameter list of the procedure;
S4L34: e : = P - 1 + (T - c)÷2;	On that address the pattern of the parameter to be specified, is listed in the object programme;
S4L35: st[e] : = st[e] or D;	
S4L36: go to S4L21;	S4L4;
S4L37: S0i;	Cf. S4L24;
S4L38: D : = 2↑23 × 1023;	
T1 : = T;	Cf. S3bL 3 and 4. The expressions of the switch list must be translated now;
q : = q - 1;	This is compensated by S8bL9. As there is supposed that the declarations of all local variables and arrays are precedent to those of switches and procedures, q + 1 is the lowest address, occupied by a local variable (namely the internal variable which is used by the adjust and restore instructions) the switch list may contain deliberate statements (jumping or not) as its elements;
S0m(0)	Within the list L, this 0 will be preceded by the pass instructions leading to the object programmes of the switch list elements, until the list of these pass instructions can be stored in the object programme of the switch (cf. S8bL7);
S4L39: d : = 328;	
go to S4L30;	For translating a switch, there is to be listed now in L: st[S - 4] = address where the switch list element to be read now has its object programme beginning,

```
                                        comment
                                    st[S - 3] = address where switch identifier is
                                    listed in I, the internal equivalent being stored
                                    in I  on  S8bL10,
                                    st[S - 2] = value 328 of opening symbol switch,
                                    st[S - 1] = 0
          end S4;
```

```
begin

S4aL1: if D > 0 then
S4aL1a: wrong;

S4aL2: st[S - 1] : = f;
       go to input;

end S4a;
```

comment

Compound statement S4a.
After reading a constant f, input goes to here with f and $g = 2^{23} \times \overline{0\ 000000t01}$ in which the bit t indicates the representation of f: t = 1 → f is real, 0 → integral or boolean;

Then a constant occurs within a list of identifiers to be declared;

The delimiter which is subsequent to the constant is read

	comment
begin	Compound statement S5.
	Pre-action of the opening symbol (;
S5L1: SOh;	Cf. S3L2;
if D < 0 then go to S5L3;	Then either a function designator or a procedure statement or an expression enclosed within parentheses is going to be translated.
	A formal parameter list is beginning.
	Procedure SOh has assigned the value of opening symbol procedure (cf. table 1B) to variable d;
if d ≠ 327 then	
S5L1a:wrong;	Then (occurs in an identifier list of a block head;
	S8dL8;
S5L2: D : = 2↑23 × 30;	Cf. table 2; ×
go to S3aL4;	The parentheses of a formal parameter part are not listed in L.
	S5L1;
S5L3: if g > 0 then	
S5L3a:wrong;	Then (is preceded by a constant;
S5L4: if Saccent < S then SOd;	Compare this with S1L7, where the operator st[S - 2] which is on the point of being translated, has already been listed in L;
	as happens too on S6aL11;
S5L5: Saccent : = 8191;	
f : = 320;	Cf. (in table 1B;
if e = 0 then go to S3L5;	Then an expression enclosed within parentheses is beginning in the text.
	function designator or procedure statement:;
S5L6: SOn(2↑23 × 63);	Thus identifier f is contra-declared according to p = $\overline{0\ 000111111}$, which constant is listed in table 2A together with an explanation; ×

	comment
S5L7: SOm(0); SOm(0);	The keys of the actual parameters will be listed in L, until all actual parameters will have been translated (cf. S5aL17);
S5L8: SOc(P);	
SO(0);	
SOc(f);	S5aL15;
S5L9: st[S - 3] ; = P;	Cf. table 4C;
mark : = 1;	For translating a function designator or procedure statement, having actual parameters,
<u>go</u> <u>to</u> S1L28;	there has been listed in L:
	st[S - 4] = address where to store the key address of the function designator or procedure statement (cf. S5aL23).
	st[S - 3] = address where the object programme (it may be empty) of the next actual parameter begins,
	st[S - 2] = value 320 of (,
	st[S - 1] = 0
<u>end</u> S5;	

	comment
begin	Compound statement S5a.
	After-actions of the opening symbol (.
	Delimiter f read last is either a comma or).
	On S1L1 the value of st[S - 3], as left by compound S5, has already been assigned to variable c.
	Progressive:
	Delimiter f is preceded in the text by p which is either a string or a trivial expression (identifier or constant).
	S1L6;
S5aL1: if c ≠ 0 then go to S5aL7;	Then p is an actual parameter.
	p, though enclosed within parentheses, is no parameter;
S5aL2: SOk;	Thus the object programme is equipped with filling the accumulator.
	Regressive:
	Delimiter f is preceded in the text by a non-trivial expression p.
	S1L21;
S5aL3: if c ≠ 0 then go to S5aL24;	Then p is an actual parameter.
	p is no actual parameter:;
if f = 321 then go to S5aL5;	Then expression p is enclosed within brackets (cf. S6L12).
	S5aL23;
S5aL4: if f ≠ 320 then	
S5aL4a:wrong;	Actual parameter, or expression intended to be enclosed within parentheses (or brackets), is followed by a wrong delimiter.
	S5aL3, S6aL26, S6aL42;

comment

S5aL5: Saccent : = S : = S - 2;	
S5aL6: go to S1L27;	Delimiter st[S-2] is regressive, for it is precedent to parentheses or brackets in the text. If its rank should be not greater than that of the delimiter to be read next regressivity is detected on S1L5.
	S5aL1;
S5aL7: if g > 0 then go to S5aL10;	Then p is a constant, e, and g = $\bar{0}\ \overline{000000t01...0}$, bit t indicating the representation of e;
if c > 0 then go to S5aL8;	Then p is an identifier, st [S] (cf. S4L23), c being the address listed in L on S5L8.
	p is a string, c being the complement of the mentioned address (cf. compound S5b):;
d : = -c; e : = P ;	
go to S5aL14;	Thus the object programme of the string occupies the addresses d, d + 1, ... e - 1.
	d, e becomes the string key (cf. table 3).
	S5aL7;
S5aL8: c : = st [S] + 2↑32;	As the parameter may be a label or a procedure- or switch identifier declared later in the text, the parameter key may (and possibly can) not yet be formed. Thus a contra-declaration of identifier st [S] will be made according to p = $\overline{0\ 000000000}$ which constant is listed in table 2A together with a general explanation;
S5aL9: d : = T;	
go to S5aL13;	The contra-declaration is to point to the address x where the key main word will be inserted in the object programme. As x is still unknown in this stage of the translation, the address d where contra-identifier

x

	comment
	c will be listed in I, is stored in L as a - preliminary key main word to be used on S5aL20 for storing the internal equivalent of c.
	S5aL7;
S5aL10: d : = g + 2↑26 × 63;	which is similar to the internal equivalent of a simple variable (cf. table 2); ×
S5aL11: <u>if</u> e < 0 <u>or</u> 2↑24 ≤ e <u>then</u> <u>go</u> <u>to</u> S5aL14;	Then constant e is either no integer or anyhow unsuitable for being used as a label. Thus no contra-declaration is made. ×
	Otherwise the constant parameter e will be contra-declared according to p = $\overline{1\ 000000000}$ which constant is listed in table 2A together with an explanation:;
S5aL12: c : = e + 2↑24 × 63 + 2↑32;	which is a contra-identifier;
d : = d + T;	Again (cf. S5aL9) the preliminary key main word points to an address in I.
	S5aL9;
S5aL13: SOe(c);	Thus contra-identifier c is listed in I. On S5aL20 its internal equivalent is stored.
	S5aL7, S5aL11, S5aL26;
S5aL14: <u>if</u> f ≠ 330 <u>then</u> <u>go</u> <u>to</u> S5aL16;	Then delimiter f is no comma, thus the actual parameter part is closing in the text.
	f is a comma (cf. table 1B):;
S5aL15: SOm(d); SOm(e); <u>go</u> <u>to</u> S5L9;	Then there is in L: st[SO] = by-word e, and st[SO + 1] = main word d of parameter key, and the next actual parameter is read.
	S5aL14;

	comment
S5aL16: SO(0);	Then, in the object programme, st[P - 1] = 0, st[P] becoming the by-word of the parameter which is the last element of the parameter list.
	Transport cycle:
	S5aL22;
S5aL17: **if** d < 0 **then** **go** **to** S5aL21;	Then parameter is a non-trivial expression and the key is ready (cf. S5aL25);
S5aL18: g : = P + 1;	The key main word is to be stored on address g;
c : = 8191 **or** d;	Then c is either = 0, or the address where the required contra-declaration is to be listed in I, or the address where the object programme of the string begins;
d : = d - c;	
S5aL19: **if** d = 0 **then** **go** **to** S5aL20;	Then parameter is either a single identifier or a string.
	Parameter is a constant:;
d : = d + P;	Then the key of the parameter looks as follows:
	st[P]= constant e,
	st[P + 1] = d = P + $\overline{0\ 111111t01\ 0...0}$;
g : = g + 2↑32;	which = P + 1 + 2^{32}. If c ≠ 0, then constant e may occur as a label (cf. table 2A). ×
	S5aL19;
S5aL20: **if** c > T **then** st[c - 1]: = g;	which is the internal equivalent of the contra-identifier stored in I on S5aL13. It refers to the main word of the parameter key which is to be formed or modified, when the translator will have arrived on S3bL23.
	S5aL17;

	comment
S5aL21: SO(e); SO(d);	Thus there has been added to the object programme: st[P - 2] = by-word e, and st[P - 1] = main word d of the parameter key;
S5aL22: e : = st[SO]; d : = st[SO+1];	Thus next parameter key is extracted from L;
SO : = SO + 2;	
<u>if</u> d ≠ 0 <u>then</u> <u>go</u> <u>to</u> S5aL17;	Then the next key is transported.
	All keys have been stored in the object programme;
S5aL23: S : = S - 2; SOa;	Then d is the address denoted with st[S - 4] on S5L9;
st[d] : = P;	which is the key address of the function designator or procedure statement;
<u>go</u> <u>to</u> S5aL4;	S5aL3;
S5aL24: <u>if</u> mark ≠ 0 <u>then</u> <u>go</u> <u>to</u> S5aL27;	Then mark = 1, the parameter having the form i[E], in which identifier i = st[S]. i is expected to be either an array- or switch identifier.
	Parameter is a non-trivial expression differing from i[E]:;
SO(return);	cf. table 1E. In the object programme of a parameter i[E] which is a subscripted variable or switch designator, the last word = 0 instead of return. +
	S5aL27;
S5aL25: e : = c + EO;	cf. table 1E; +
d : = 2↑23 × 1017;	d, e is the key of an actual parameter which is a non-trivial expression; (cf. table 3); ×
S5aL26: <u>go</u> <u>to</u> S5aL14;	S5aL24;
S5aL27: P : = P - 1;	Then st [P] is either an ar2 instruction or 0 (cf. S6aL61);

```
                                comment
S0s;                            The mentioned identifier  i  is contra-declared
                                because the ar2 instruction might, and a zero must
                                be replaced later by a jump instruction.
                                A zero is added to the parameter's object programme;
go to S5aL25
end S5a;
```

	comment
begin	Compound S5b.
	Action of delimiter **lsq** (cf. table 1B);
S5bL1: S0a;	Then d = delimiter st[S - 2], the parenthesis having the value 320;
if d ≠ 320 **or** c = 0 **then** **go** **to** S5bL2;	Otherwise the string beginning in the text is an actual parameter;
st[S - 3] : = - c;	Thus the address P listed in L on S5L8 is replaced by its complement;
go **to** input1;	Then the string is read. For storing the successive words, procedure S0 is recommended, because it also pays attention to the position of the list L. After skipping delimiter **rsq**, input1 goes to S1L28;
S5bL2: wrong;	This string is no actual parameter thus perhaps a piece of code programme to be included in the object programme?
end S5b;	

	comment
begin	Compound statement S6.
	Pre-action of the opening symbol [.
S6L1: f := 321;	cf. [in table 1B;
if D < 0 then go to S6L10;	Then a subscript list is beginning in the text.
	- bound pair list:;
S6L2: S0a;	Then the following assignments are performed:
	c = value, variable T had, when the 1st declarator
	of the block was read (cf. S3aL5),
	d = value 322 of the opening symbol begin,
	e = address, reserved for the pre-value [a] of the
	1st array, called a in the explanations (cf. S4L14
	and S4L16);
if e = 0 then	
S6L2a: wrong;	Then a list of array identifiers has been omitted;
S6L3: D := D + 2↑32;	for D must be negative during the translation of the
	expressions of the bound pairs. Then the tests on
	S6L1 and S4L1 succeed;
S6L4: if D or 2↑29 ≠ 0 then go to	Then the arrays are not own.
S6L6;	Arrays are own:;
S6L5: e := P;	which = 1 + address where pre-value [c] of c, the
	last array of the list, is to be stored later
	(cf. S6aL41);
go to S6L7;	S6L4;
S6L6: e := q;	That address is reserved for the internal variable u.
	Thus address e + 1 is reserved for pre-value [c] of c,
	the last array of the list;
T1 := c;	for, within the bounds, there only occur non-local
	variables.
	S6L5;

	comment
S6L7: SOc(e);	
S6L8: e : = e + R + 2↑23 - 2 +	cf. ar1 in table 1D;
2↑26 × 126;	×
SOe(2↑24 × 62);	In the case of an own declaration, the internal × equivalent of the factor identifier $2^{24} \times 62$ is not e and will be inserted later on S6aL38;
S6L9: go to S3L3;	Then, for translating a bound pair list, there is listed in L: st[S - 5] = address reserved for the pre-value [a] of a = 1^{st} array of the list mentioned in the explanations, st[S - 4] = ∓ 1 + address reserved for the pre-value [c] of c = last array of the list, st[S - 3] = -1 + number of bound pairs already translated, st[S - 2] = value 321 of [, st[S - 1] = 0. S6L2;
S6L10: SOh;	for a procedure statement with actual parameters can be the 1st statement in the compound tail of a block (cf. S3L2);
S6L11: if g ≠ 0 then go to S6L15;	Then [is preceded by an array identifier;
S6L12: if e = 0 then go to S5L4;	Then there is beginning an expression enclosed within brackets, which is no subscript. [is preceded by either a formal parameter or a switch identifier:;
S6L13: if e or (2↑32 + 2↑23) ≠ 0 then	cf. ar2 in table 1D.
e : = 2↑26 × (63 - 125);	Then [is not preceded by a formal parameter. Thus × it is preceded by a switch identifier whose declaration may, of course, occur later in the text;

	comment
S6L14: st[S - 1] : = e + 2↑26 × (126 - 63);	That is either the ar1 instruction referring to a formal parameter, or a positive and harmless constant; ×
<u>go</u> <u>to</u> S6L18;	S6L11;
S6L15: <u>if</u> g > 0 <u>then</u>	
S6L15a: wrong;	Then [is preceded by a number;
S6L16: st[S - 1] : = g;	which is the ar1 instruction listed in I on S6L8 and referring to the 1st factor of the array declaration;
S6L17: e : = U + 2↑31;	Then bit $e_1 = 1$ as it is in the case that e is the internal equivalent of a formal parameter. S6L14;
S6L18: S : = S + 2;	
<u>if</u> S + 1 > T <u>then</u> S0g;	as happens in procedure S0c;
st[S - 1] : = e + 2↑26 × (125 - 63);	×
S0c (- mark);	as is indicated in table 4C;
st[S - 1] : = 0;	
S6L19: <u>go</u> <u>to</u> S3L5;	Then, for translating a subscript list, there is listed in L: st[S - 7] = an ar1 instruction or a positive constant, st[S - 6] = the array- or switch identifier or the formal parameter, preceding [in the text, st[S - 5] = an ar2 instruction of 0, st[S - 4] = minus the value of mark which was present when [was read,

```
                                    comment
                              st[S - 3] = - 1 + number of subscripts already
                              translated,
                              st[S - 2] = value 321 of [ ,
                              st[S - 1] = 0
        end S6;
```

```
                                           comment
    begin                                  Compound statement S6a.
                                           After-actions of the opening symbol [ .
                                           Delimiter  f  read last is either a comma or a colon
                                           or ] . On S1L1 (or S1L18) the value of st[S - 3] as left
                                           by compound S6, has already been assigned to variable  c.
                                           Progressive:
                                           In the text, delimiter  f  is preceded by a trivial
                                           expression  p (a constant or identifier).
                                           S6L1;
S6aL1: d : = st[S - 4];                    Compare S6L9 with S6L19;
S6aL2: if d ≤ 0 then go to S6aL43;         Then  p  is a subscript.
                                           p  is a bound:;
S6aL3: if D or 2↑29 = 0 then go to         Then declaration is own.
       S6aL27;                             Declaration is not own:;
S6aL4: SOk;                                Thus  p  is translated.
                                           Regressive:
                                           expression  p  is not trivial and is already
                                           translated.
                                           S1L21;
S6aL5: d : = st[S - 4];                    cf. S6aL1;
S6aL6: if d ≤ 0 then go to S6aL53;         Then  p  is a subscript.
                                           p  is a bound:;
S6aL7: if D or 2↑29 = 0 then                                                          x
S6aL7a:wrong;                              Please only constant bounds in an own array
                                           declarations;
S6aL8: if c ≠ 0 then d : = d + 1;          Thus, in the case of the 1st bound pair, d is
                                           the address of variable  u, otherwise  d  is the
                                           address of variable  v = [c], as is indicated in
                                           the explanations;
```

	comment	
S6aL9: if f ≠ 331 then go to S6aL12;	Then delimiter f is no colon.	
	colon:;	
S6aL10: SO(d+R+2↑23+2↑26×117);	cf. store accu in table 1D.	×
	Thus u : = accu or v : = accu is translated.	
	entry, S3bL5, S7aL9, S7aL17, S6aL19;	
S6aL11: Saccent : = 8191;		
go to S1L27;	As long as S' has the value 8191, the tests on S1L7, S5L4 and S6aL44 cannot succeed.	
	S6aL9;	
S6aL12: if c = 0 then q : = q - 1;	the latter being the same as q : = d - 1.	
	Then q is the address to be reserved for the variable H_k, the value of which is the number of array elements;	
S6aL13: SO(d+R+2↑23+2↑26×74);	cf. - in table 1A.	×
	Thus accu : = accu - u or accu : = accu - v is translated;	
S6aL14: SO(q+R+2↑23+2↑26×115);	cf. store factor in table 1B;	×
S6aL15: if c = 0 then e : = 0 else		
e : = c + 2;	else e : = d - q may be written instead;	
S6aL16: SO(e);	Thus the constant 0 or k - i + 2 (i < k, cf. explanation) is subsequent to the store factor instruction;	
S6aL17: q : = q - 1;	That address is reserved for the variable to be introduced next which is possibly another factor h_{i-1}.	
	S6aL36, S6aL58;	
S6aL18: st[S - 3]: = c : = c + 1;	Thus the number of bound pairs or subscripts is counted;	
S6aL19: if f = 330 then go to S6aL11;	Then f is a comma;	

	comment	
if f ≠ 321 then		
S6aL19a:wrong;	Then a subscript, or a bound of an array declaration, is not followed by the required comma,], or colon;	
S6aL20: D : = D + 2↑32;	which compensates for S6L3;	×
S6aL21: S : = S - 2;		
S6aL22: if D or 2↑29 = 0 then go to S6aL37;	for own;	×
S6aL23: S0a;	Then is (cf. S6L9): c = address reserved for variable [a], d = address reserved for u;	
S6aL24: S0(c+R+2↑23+2↑26×114);	cf. store pre-value in table 1D;	×
S6aL25: S0(d - c);	In the object programme, the store pre-value instruction is followed by the constant which is the difference (address of u minus that of [a]);	
S6aL26: go to S5aL5;	The brackets are exhausted. S6aL3;	
S6aL27: if g = 0 then go to S6aL7a;	Then a variable bound occurs within a declaration of own arrays;	
S6aL28: if g + 2↑25 × 127 < 0 then go to S6aL7a;	Then the bound has not the integer representation;	
S6aL29: if f ≠ 331 then go to S6aL32;		
S6aL30: if c ≠ 0 then v : = e else u : = e;		
S6aL31: go to S1L28	S6aL29;	
S6aL32: if c ≠ 0 then e : = e - v else e : = e - u;		
S6aL33: if e < 0 then		
S6aL33a:wrong;	Then lower bound is greater than upper bound;	
e : = e + 1;		

	comment
S6aL34: SOm(e);	In the store, the calculated constants must appear in the same order as have the factor variables of the arrays being not <u>own</u>. Storing in L through procedure SOm is a means of reversing the order;
S6aL35: <u>if</u> c = 0 <u>then</u> <u>go</u> <u>to</u> S6aL36;	
st[SO + c]: = st[SO + c] × e;	which is the next value of H_k;
u : = u × e + v;	which is the next value of u.
	S6aL35;
S6aL36: <u>go</u> <u>to</u> S6aL18;	S6aL22, S6aL37;
S6aL37: g : = st[SO]; SO(g);	
SO : = SO + 1;	
c : = c − 1;	
<u>if</u> c ≠ 0 <u>then</u> <u>go</u> <u>to</u> S6aL37;	Thus the constants h_1, ... h_{k-1}, and H_k are listed in the object programme, g being = H_k at the end;
S6aL38: st[T+1]: = P+2↑23+2↑26×126−2;	which is the correction announced on S6L8; ×
S6aL39: SO(u);	Constant u is stored;
S6aL40: SOa;	cf. S6aL23. Addresses c and d−1 are reserved for the first and last pre-value constants. S6aL41;
S6aL41: st[c]: = P − u; P : = P − g;	
c : = c + 1;	
<u>if</u> c ≠ d <u>then</u> <u>go</u> <u>to</u> S6aL41;	Thus space for own arrays is reserved in object programme;
S6aL42: <u>if</u> P>SO−2 <u>then</u> SOg;	
<u>go</u> <u>to</u> S5aL5;	S6aL2;
S6aL43: <u>if</u> c ≠ 0 <u>then</u> <u>go</u> <u>to</u> S6aL45;	Then the identifier or constant p to be translated is not the first subscript of the list. First subscript:;

	comment	
S6aL44: if Saccent + 2 < S then SOd;	as happens also on S1L7;	
SOk;	Thus 1st subscript is translated;	
go to S6aL54;	S6aL43;	
S6aL45: P : = P - 1;	Then st[P] = instruction partres, stored at the previous action of this compound S6a on S6aL57;	
S6aL46: if g ≠ 0 then go to S6aL50;	Then g indicates the representation of e, e being a constant subscript	
S6aL47: if e < 0 then go to S6aL52;	Then subscript p is a procedure identifier namely no simple variable and no formal parameter.	
	Simple variable or formal parameter;	
S6aL48: e : = e + 2↑26 × (72 - 63);	cf. + in table 1A;	×
S6aL49: go to S6aL51;	S6aL46;	
S6aL50: SO(g + 2↑26 × 72);	cf. S6aL48.	×
	S6aL49;	
S6aL51: SO(e);		
go to S6aL54;	not: to 53.	
	S6aL47;	
S6aL52: P : = P + 1;	Thus instruction partres remains in the object programme;	
SOp;	Thus a contra-declaration of function name p is made. The test on S6aL53 succeeds.	
	S6aL6;	
S6aL53: if c ≠ 0 then	cf. + in table 1A. This instruction	
SO(2↑23 × 3 + 2↑26 × 72);	accu : = accu + (partial result) corresponds to the instruction partres, added to the object programme during the previous action of this compound S6a on S6aL57.	×
	S6aL44, S6aL51;	
S6aL54: if f=321 then go to S6aL59;	cf.] in table 1B. Then the last subscript has been translated.	

	comment
	Subscript list is not yet exhausted:;
S6aL55: d : = st[S - 7];	That is either an ar1 instruction referring to an array identifier or formal parameter, or positive;
<u>if</u> d > 0 <u>then</u>	
S6aL55a:wrong;	Then 2 or more subscripts are attached to an identifier which is no array identifier and no formal parameter;
S6aL56: <u>if</u> d - 2↑23 - 2↑26×126 < 0 <u>then</u> SO(d) <u>else</u> c : = d - c; SO(c);	Thus the ar1 instruction just stored either refers itself to the factor h_i (i < k) required, or it refers to the key of a formal parameter which refers to the factor h_{k-1} of the array represented by the parameter, while ar1 is followed, in the object programme, by the constant k - i - 1;
S6aL57: SO(partres);	cf. table 1E This instruction partres corresponds to the instruction accu : = accu + (partial result) which will be added to the object programme during the next action of this compound (cf. S6aL53). However, often both instructions are dispensable;
S6aL58: <u>go</u> <u>to</u> S6aL18;	S6aL54;
S6aL59: mark : = - d;	Thus the value of mark, listed in L on S6L19, is restored;
S6aL60: Saccent : = S : = S - 6;	Compare this with S5aL5;
S6aL61: SO(st[S + 1]);	Then st[P - 1]is the ar2 instruction or 0, mentioned on S6L19.

```
                              comment
                         st[S] is the identifier to which the subscripts
                         are attached in the text;
go to S1L28
end S6a;
```

	comment
<u>begin</u>	Compound statement S7.
	Pre-action of the opening symbol <u>for</u>
S7L1: SOh;	cf. S3L2;
S7L2: st[S - 1] : = 0;	Thus an identifier or constant preceding opening symbol <u>for</u> is skipped;
SOc(0);	
st[S - 1] : = - P;	
f : = 324;	
<u>go</u> <u>to</u> S3L4;	Then, for translating a <u>for</u> statement, there is listed in L: st[S - 5] = 0, st[S - 4] = 0 to be replaced by the internal equivalent of the controlled variable v on S9L4, st [S - 3] = minus address P' where the object programme of the <u>for</u> list element to be read next is to begin, st [S - 2] = value 324 of the opening symbol <u>for</u>, st [S - 1] = 0
<u>end</u> S7;	

	comment
begin	Compound statement S7a. After-actions of the opening symbol for . Delimiter f read last may be a separation symbol step, until, while, comma, or do, or the closing symbol of the for statement. Progressive: Expression p preceding delimiter f in the text is trivial (an identifier or constant). S1L6;
S7aL1: S0k;	for translation of p.
	Regressive: Expression p is not trivial. S1L21;
S7aL2: S := S - 2;	
S0a;	Thus internal equivalent of controlled variable is assigned to variable d (cf. st[S - 4] in compound S7);
if d ≤ 0 then	
S7aL2a: wrong;	Then either controlled variable is no simple variable and no formal parameter, or even := has been omitted;
S7aL3: if c < 0 then go to S7aL18;	Then after-action of for with do has already taken place so that f is the closing symbol (compare st[S - 5] in S7aL16 with that in S7aL15 or S7L2);
S7aL4: S := S + 2;	Thus previous value of S is restored;
if f ≠ 324 then go to S7aL10;	Then f is either comma or do (cf. 324 in table 1B);
S7aL5: if e < 0 then go to S7aL7;	Then delimiter f is either step or while.
	Delimiter f = until:;
S7aL6: st[e] := d+2↑26 ×(123-63);	cf. for1 in table 1D; ×

	comment	
SO(for3);	cf. table 1E;	+
c : = 0;		
go to S7aL8;	S7aL5;	
S7aL7: c : = P;		
SO (-e);	Thus address where object programme of while- or step element begins, is retained in object programme itself on address c;	
SO(d + 2↑26 × (117 - 63));	cf. store accu in table 1D. S7aL6, S7aL15;	×
S7aL8: st[S - 3] : = c;		
S7aL9: go to S6aL11;	st[S - 2] = for remains in L, f disappears. S7aL4;	
S7aL10: if e > 0 then go to S7aL12;	Then value of e has been listed in L during previous action of this compound on S7aL7-8. It is a while element whose object programme is to be completed;	
S7aL11: if e ≠ 0 then		
begin	element is an expression	
SO(d + 2↑26 × (117 - 63));		×
b : = for2;	cf. table 1E	+
end		
else b : = d + 2↑26 × (74-63);	cf. in table 1A. It is a step element	×
SO(b);		
go to S7aL14;	Object programme of step- or expression element is ready. S7aL10;	
S7aL12: b : = st [e];	That is the address c of S7aL7. Element has the form: F while G. S7aL13;	

```
                                                   comment
S7aL13: st[e] : = st [e - 1]:
        e : = e - 1;
        if e ≠ b then go to S7aL13;        Thus object programme of expression  F  is shifted
                                           over one place;
        st [b] : = for2;                   cf. table 1E.
                                           Thus instruction for2 precedes object programme of  F.
                                           S7aL11;
S7aL14: if f = 323 then go to S7aL16;      cf. do in table 1B;
        if f ≠ 330 then
S7aL14a:wrong;                             Then for statement contains wrong delimiter;
S7aL15: st[S - 5] : = P;                   That is the address where to insert a  for instruction
                                           later (cf. S7aL16);
        SO(c);                             On address st[S - 5] the previous address where to
                                           insert  a  for instruction, is stored;

        c : = - P;
        go to S7aL8;                       Thus  c  is the complemented address where the object
                                           programme of the next for list element is to begin.
                                           S7aL14;
S7aL16: e : = P+2+2↑23+2↑26×124;           cf. for  in table  1D;                          ×
        SO(e);                             Thus last  for  instruction is stored;
        st[S - 5]: = - P;                  That is the complemented address where to store a
                                           pass instruction later (cf. S7aL18);
        SO(0);                             For the time being,  0  is inserted.
                                           S7aL17;
S7aL17: if c = 0 then go to S6aL11;        Then all for instructions have been inserted;
        b : = c;  c : = st [c];
        st[b] : = e;                       Thus next for instruction is inserted on address  b;
        go to S7aL17;                      S7aL3;
```

```
                                          comment
S7aL18: SO(for0);                         cf. table 1E.                                  +
                                          Thus object programme of  for statement is concluded
                                          by adding the word  for0;
        st[-c]: = P + 2↑23 + 2↑26 × 121;  Thus a bridging pass instruction (cf. table 1D) is
                                          inserted;
        go to S1L17;                      There variable S is again (cf. S7aL2) decreased
                                          with 2, and the delimiter preceding  for  in L, is
                                          going to after-act with delimiter  f
        end S7a
```

	comment
begin	Compound statement S8.
	Pre-action of the opening symbol **go** **to**;
S8L1: S0h;	cf. S3L2;
S0c(325);	Then st [S - 2] is the value 325 of **go** **to**
	(cf. table 1B)
	S8bL6, S9L3;
S8L2: mark : = 2;	cf. table 4C;
Saccent : = 8191;	as happens also on S6aL11;
go **to** S1L28	
end S8;	

```
                                        comment
    begin                               Compound statement S8a.
                                        After-actions of the opening symbol go to .
                                        Regressive (cf. S8bL1):
                                        S1L21;

S8aL1: if mark = 0 then go to S8aL3;
       P : = P - 1;                     Progressive (cf. S8bL2):
                                        S1L6;
S8aL2: SC1;                             S2aL32, S8aL1, S8bL11;
S8aL3: mark : = 0;
       go to S1L17;                     Delimiter st[S - 4] is going to after-act with
                                        delimiter f
       end S8a;
```

	comment
begin	Compound statement S8b. After-action of the opening symbol switch . f = delimiter read last is either the separation symbol comma or the closing symbol semicolon of the switch declaration. Regressive: The designational expression p preceding delimiter f in the text, is not a single label. S1L21;
S8bL1: if mark = 0 then go to S8bL3;	Then p contains more than a switch designator and has already been translated and contra-declared. p is a switch designator i[E]. Translation of E is ready. st[P - 1] = 0 or an ar2 instruction (cf. S6aL61) which is to be replaced by a jump instruction later. Identifier i is available as st[S] and g = 0;
P : = P - 1;	Progressive: p is a label. S1L6;
S8bL2: SO1;	Thus there is made a contra-declaration of the label or switch identifier, referring to address P - 1 where a jump instruction will be inserted later (cf. S3bL39). S3bL1;
S8bL3: S : = S - 2; SOa;	Now d is the address where the designational expression preceding delimiter f in the text, has it object programme beginning (cf. S8bL6 and S4L39);
S8bL4: c ; = d + 2↑23 + 2↑26 × 121;	which is the pass instruction referring to d;

```
                                            comment
S8bL5: if f ≠ 330 then go to S8bL7;        Then  f  is a semicolon.
                                            f  is a comma:;
S8bL6: S : = S + 2;                         Thus previous value of  S  is restored;
       st[S - 4] : = P;
       SOm(c);                              For the time being, pass instruction is listed in L;
       go to S8L2;                          for reading next switch list element.
                                            S8bL5;
S8bL7: d : = 0;                             for counting the entries of the switch list i.e.
                                            pass instructions preliminarily listed in L.
                                            S8bL8;
S8bL8: d : = d + 1;
       SO(c); SO : = SO + 1;
       c : = st[SO - 1];
       if c ≠ 0 then go to S8bL8;           Then the next pass instruction is stored in the
                                            object programme.
                                            The list of pass instructions has been stored in the
                                            object programme:;
S8bL9: q : = q + 1;                         cf. S4L38;
S8bL10: st[e - 1] : = P+R+2↑23×127;         Thus internal equivalent (table 2) of switch          ×
                                            identifier is listed in  I;
S8bL11: SO(q + R + 2↑23 - 1 + 2↑26×112);cf. switch in table 1D;
        SO(d);                              Thus, in object programme, the switch instruction
                                            is followed by the number of entries;

        go to S8aL3;
        end S8b;
```

	comment	
<u>begin</u>	Compound statement S8d.	
	After-actions of the opening symbol <u>procedure</u>.	
	Progressive:	
	S1L6;	
S8dL0: wrong;	Procedure body is a dummy or a single identifier or constant.	
	Regressive:	
	S1L21;	
S8dL1: <u>if</u> D > 0 <u>then</u> <u>go</u> <u>to</u> S8dL8;	Then body must still be read.	
	Body has been translated:;	
<u>if</u> f ≠ 332 <u>then</u>		
S8dL1a:wrong;	Then procedure body is not followed by a semi-colon;	
S8dL2: S : = S - 2; S0a;	Then c = q', d = P' + 2 and e = T' are the values mentioned on S4L31;	
S8dL3: R : = R - 2↑13;	for compensating the action of S4L26;	×
S8dL4: <u>if</u> (st[e - 1] <u>or</u> 2↑24) = 0 <u>then</u>	cf. table 1E.	+
S0(extract procedure);	Thus instruction extract procedure only appears in the object programme of a type procedure;	
S8dL5: S0(Y);	table 1E.	+
	Thus, in object programme of any procedure, instruction Y is the last word;	
S8dL6: <u>if</u> st[d - 1] < 0 <u>then</u>	Thus, in object programme of a procedure having no	
st[d - 2] : = X1;	parameters, instruction X1 of table 1E is 1st word;	+
S8dL7: e : = e - 2; <u>go</u> <u>to</u> S3bL12;	S8dL1;	
S8dL8: <u>if</u> f = 332 <u>then</u> <u>go</u> <u>to</u> S8dL9;	for semi-colon;	
<u>if</u> (D <u>or</u> 2↑23) = 0 <u>then</u> <u>go</u> <u>to</u>		
S5L2;	for reading the next formal parameter;	
<u>go</u> <u>to</u> S1L28;	for reading parameter to be next specified.	
	S8dL8;	

```
                                        comment
S8dL9: T1 : = T; D : = 2↑23 × 543;     Table 2;                ×
       go to S6aL11
       end S8d;
```

	comment	
begin	Compound statement S9.	
	Pre-action of the opening symbol : =;	
S9L1: **if** g ≠ 0 **then**		
S9L1a:wrong;	Then : = is preceded by a constant;	
S9L2: S0h;	as happens also on S3L2. The procedure performed the following assignments:	
	d : = st[S − 2], e : = st [S − 1];	
S9L3: **if** d = 328 **then** **go** **to** S8L2;	Then the delimiter listed last in L is **switch** (cf. S4L39);	
S9L4: **if** d ≠ 324 **then** **go** **to** S9L5;	Otherwise the delimiter listed last in L is **for**:;	
st[S − 4] : = e;	as was announced in compound S7;	
go **to** S1L28;	S9L4;	
S9L5: **if** e < 0 **then** **go** **to** S9L11;	Then : = is preceded in the text by the identifier of a type procedure;	
if e ≠ 0 **then** **go** **to** S9L7;	Then either a simple variable or a formal parameter is precedent to :=	
	A subscripted variable is precedent to : = ;	
if (st[P − 1] **or** 2↑26 × 127) ≠ 2↑26 × 125 **then**		×
S9L5a:wrong;	Then, within an assignment statement, : = is not preceded by a (simple or subscripted) variable or formal parameter or identifier of a type procedure;	
S9L6: e : = − 1;		
go **to** S9L9;	S9L5;	
S9L7: e : = e + 2↑26 × (117 − 63);	cf. store accu in table 1D;	×
S9L8: **if** e − 2↑23 − 2↑26 × 117 > 0 **then** **go** **to** S9L10;	Then a simple variable is precedent to : =	×
	Formal parameter;	

```
                                                comment
        e : = e + 2↑26;                   cf. store address in table 1D.
                                          S9L6;
S9L9:   S0(e);                            Then st [ P - 1 ] is either a store address instruction
                                          or the constant -1;
        e : = extract address;            cf. table 1E.                                          +
                                          S9L8, S9L11;
S9L10;  st[S - 1] : = e;                  Thus either a store accu-or store procedure- or
                                          extract address instruction is listed in L;
        f : = 323;                        cf. : = in table 1B;
        go to S3L4;                       S9L5;
S9L11:  f : = e + 2↑32; S0f;              Thus procedure identifier is again looked up;        ×
        e : = (d or 2↑26-2↑13)+2↑13       cf. store procedure and Q0 in tables 1D and
        + 2↑26×113+3+Q0;                  1E;                                                 + ×
        if (d or 2↑24 + 2↑29) = 0         Then identifier signifies a type procedure;
        then go to S9L10;                                                                      ×
        go to S9L5a
        end S9;
```

```
                                              comment
      begin                                   Compound statement S9a.
                                              After-actions of the opening symbol  : =
                                              On S1L1 or S1L18, the value of st [S - 3] has been
                                              assigned to variable  c.
                                              Progressive:
                                              The expression preceding  delimiter  f  in the text
                                              is a single identifier or constant  p.
                                              S1L6;
S9aL1: S0k;                                   Thus  p  is translated.
                                              Regressive:
                                              p  has already been translated.
                                              S1L21;
S9aL2: S0(c);                                 Thus st[P - 1] is the instruction listed in  L  on
                                              S9L10;
      go to S1L17;                            cf. S8aL3;
      end S9a;

input:
      begin
        :
      end
input1:
      begin

        :
      end

      end  translator
```