PENULTIMATE DRAFT REPORT
ON THE ALGORITHMIC LANGUAGE
ALGOL 68

CHAPTERS 10, 11 AND 12

A. van Wijngaarden (Editor),
B.J. Mailloux, J.E.L. Peck

MR 99
October 1968
10. Standard declarations

a) A "standard declaration" is one of the constituent declarations of the standard-prelude \( \{2.1.1\} \) \{; it is either an "environment enquiry", supplying information concerning a specific property of the implementation \( \{2.3.1\} \), a "standard priority" or "standard operation", a "standard mathematical constant or function", a "synchronization operation" or a "transput declaration"\}.

b) A representation of the standard-prelude is obtained by altering each form in 10.1, 10.2, 10.3, 10.4 and 10.5 in the following steps:

Step 1: Each sequence of symbols between \( \{ \) and \( \} \) in a given form is altered in the following steps:

Step 1.1: If \( D \) occurs in the given sequence of symbols, then the given sequence is replaced by a chain of a sufficient number of sequences separated by comma-symbols; the first new sequence is a copy of the given sequence in which copy \( D \) is deleted; the \( n \)-th new sequence, \( n > 1 \), is a copy of the given sequence in which copy \( D \) is replaced by a sub-symbol followed by \( n-2 \) comma-symbols followed by a sub-symbol;

Step 1.2: If, in the given sequence of symbols, as possibly modified in Step 1.1, \( \text{int} \) (\( \text{real} \) or \( \text{compl} \)) occurs, then that sequence is replaced by a chain of \( \text{int lengths} \) \( \{10.1.a\} \) (\( \text{real lengths} \) \( \{10.1.c\} \)) sequences separated by comma-symbols, the \( n \)-th new sequence being a copy of the given sequence in which copy each occurrence of \( L(L) \) has been replaced by \( (n-1) \) times \( \text{long} (\text{Long}) \);

Step 2: Each occurrence of \( \{ \) and \( \} \) in a given form, as possibly modified in Step 1, is deleted;

Step 3: If, in a given form, as possibly modified in Steps 1 and 2, \( \text{int} \) (\( \text{real} \) or \( \text{compl} \), \( \text{bits} \), \( \text{bytes} \), both \( \text{int} \) and \( \text{real} \) or both \( \text{int} \) and \( \text{compl} \) \) occurs, then the form is replaced by a sequence of \( \text{int lengths} \) \( \{10.1.a\} \) (\( \text{real lengths} \) \( \{10.1.c\} \), \( \text{bits widths} \) \( \{10.1.f\} \), \( \text{bytes widths} \) \( \{10.1.j\} \), the minimum of \( \text{int lengths} \) and \( \text{real lengths} \) new forms; the \( n \)-th new form is a copy of the given form in which copy each occurrence of \( L(L, K, S) \) is replaced by \( (n-1) \) times \( \text{long} (\text{Long, Long, short}) \);
10. continued

Step 4: If $Q$ occurs in a given form, as possibly modified or made in the
Steps above, then the form is replaced by four new forms obtained by
replacing $Q$ consistently throughout the form by either $-$ or $+$ or $\times$ or $/$.
Step 5: If $Q$ occurs in a given form, as possibly modified or made in the
Steps above, then the form is replaced by four new forms obtained by
replacing $Q$ consistently throughout the form by either $\text{minus}$ or $\text{plus}$
or $\text{times}$ or $\text{over}$.
Step 6: If $R$ occurs in a given form, as possibly modified or made in the
Steps above, then the form is replaced by six new forms obtained by
replacing $R$ consistently throughout the form by either $<$ or $\leq$ or $=$ or
$\neq$ or $\geq$ or $>$.
Step 7: Each occurrence of $F$ in any form, as possibly modified or made
in the Steps above, is replaced by a representation of the
letter-aleph-symbol $\{1.1.1.n, 5.5.8\}$.
Step 8: If, in some form, as possibly modified or made in the Steps above,
% occurs followed by the representation of an identifier (field-selector,
indication), then that occurrence of % is deleted and each occurrence
of the representation of that identifier (field-selector, indication)
in any form is replaced by the representation of one same identifier
(field-selector, indication) which does not occur elsewhere in a form,
and Step 8 is taken.
Step 9: If a representation of a comment occurs in any form, as possibly
modified or made in the Steps above, then this representation is
replaced by a representation of an actual-declarer or closed-clause
suggested by the comment.
Step 10: If, in any form, as possibly modified or made in the Steps above,
a representation of a routine-denotation occurs whose elaboration in-
volves the manipulation of real numbers, then this denotation may
be replaced by any other denotation whose elaboration has approximately
the same effect {; the degree of approximation is left undefined in this
Report (see also 2.2.3.1.c) }.
Step 11: The standard-prelude is that declaration-prelude whose represent-
ation is the same as the sequence of all the forms, as possibly modified
or made in the Steps above.

{The declarations in this Chapter are intended to describe their effect
clearly. The effect may very well be obtained by a more efficient method.}
10.1. Environment enquiries

a) \( \text{int int lengths} = a \) the number of different lengths of integers \( a \);
b) \( \text{int L max int} = a \) the largest \( L \) integral value \( a \);
c) \( \text{int real lengths} = \)
\( a \) the number of different lengths of real numbers \( a \);
d) \( \text{L real L max real} = a \) the largest \( L \) real value \( a \);
e) \( \text{L real L small real} = a \) the smallest \( L \) real value such that both
\( L1 + L \text{ small real} \geq L1 \) and \( L1 - L \text{ small real} < L1 \);
f) \( \text{int bits widths} = \)
\( a \) the number of different widths of bits \( a \);
g) \( \text{int L bits width} = \)
\( a \) the number of elements in \( L \) bits; see \( L \text{ bits} \) (10.2.8.a) \( a \);
h) \( \text{op abs} = (\text{char a} \text{ int}) : \)
\( a \) the integral equivalent of the character 'a' \( a \);
i) \( \text{op repr} = (\text{int a} \text{ char}) : \)
\( a \) that character 'a', if it exists, for which \( \text{abs} x = a \);
j) \( \text{int bytes widths} = \)
\( a \) the number of different widths of bytes \( a \);
k) \( \text{int L bytes width} = \)
\( a \) the number of elements in \( L \) bytes; see \( L \text{ bytes} \) (10.2.9.a) \( a \);
l) \( \text{char null character} = a \) some character \( a \);

10.2. Standard priorities and operations

10.2.0. Standard priorities

a) \( \text{priority minus} = 1, \text{plus} = 1, \text{times} = 1, \text{over} = 1, \text{modb} = 1, \text{prus} = 1, \)
\( v = 2, \wedge = 3, = = 4, \neq = 4, < = 5, \leq = 5, \geq = 5, > = 5, \ldots = 6, \)
\( + = 6, \times = 7, \div = 7, : = 7, /= 7, \text{elem} = 7, + = 8, \text{lob} = 9, \text{upb} = 8, \)
\( \text{lws} = 8, \text{ups} = 8, \downarrow = 9 ; \)

10.2.1. Rows and associated operations

a) \( \text{mode \% rows} = a \) an actual-declarer specifying a mode united from
\( '{L, 4.3.a} \text{ all modes beginning with 'row of' a} ; \)
10.2.1. continued

b) \texttt{op lwb} = (\texttt{int n, rowe a})\texttt{int} : a \text{ the lower bound in the } n\text{-th quintuple of the descriptor of the value of 'a', if that quintuple exists } a ;

c) \texttt{op upb} = (\texttt{int n, rowe a})\texttt{int} : a \text{ the upper bound in the } n\text{-th quintuple of the descriptor of the value of 'a', if that quintuple exists } a ;

d) \texttt{op lws} = (\texttt{int n, rowe a})\texttt{bool} : a \text{ true (false) if the lower state in the } n\text{-th quintuple of the descriptor of the value of 'a' equals } 1(0), \text{ if that quintuple exists } a ;

e) \texttt{op ups} = (\texttt{int n, rowe a})\texttt{bool} : a \text{ true (false) if the upper state in the } n\text{-th quintuple of the descriptor of the value of 'a' equals } 1(0), \text{ if that quintuple exists } a ;

f) \texttt{op lwb} = (\texttt{rowe a})\texttt{int} : 1 \texttt{lwb} a ;
g) \texttt{op upb} = (\texttt{rowe a})\texttt{int} : 1 \texttt{upb} a ;
h) \texttt{op lws} = (\texttt{rowe a})\texttt{bool} : 1 \texttt{lws} a ;
i) \texttt{op ups} = (\texttt{rowe a})\texttt{bool} : 1 \texttt{ups} a ;

10.2.2. Operations on boolean operands

a) \texttt{op v} = (\texttt{bool a, b})\texttt{bool} : (a \text{ true } b) ;
b) \texttt{op ^} = (\texttt{bool a, b})\texttt{bool} : (a \text{ b false}) ;
c) \texttt{op !} = (\texttt{bool a})\texttt{bool} : (a \text{ false true}) ;
d) \texttt{op =} = (\texttt{bool a, b})\texttt{bool} : (a \text{ b } \lor (\neg a \land \neg b)) ;
e) \texttt{op !} = (\texttt{bool a, b})\texttt{bool} : \neg (a = b) ;
f) \texttt{op abs} = (\texttt{bool a})\texttt{int} : (a \text{ 1 0}) ;

10.2.3. Operations on integral operands

a) \texttt{op <} = (\texttt{L int a, b})\texttt{bool} : a \text{ true if the value of 'a' is smaller than that of 'b' and false otherwise } a ; \{2.2.3.1.c\}
b) \texttt{op <=} = (\texttt{L int a, b})\texttt{bool} : \neg (b < a) ;
c) \texttt{op =} = (\texttt{L int a, b})\texttt{bool} : a \leq b \land b \leq a ;
d) \texttt{op !} = (\texttt{L int a, b})\texttt{bool} : \neg (a = b) ;
e) \texttt{op >=} = (\texttt{L int a, b})\texttt{bool} : b \leq a ;
f) \texttt{op >} = (\texttt{L int a, b})\texttt{bool} : b < a ;
g) \texttt{op -} = (\texttt{L int a, b})\texttt{L int} : a \text{ the value of 'a' minus that of 'b' } a ; \{2.2.3.1.c\}
10.2.3. continued

h) \( op - = (\text{int } a) \text{ int} : L_0 - a ; \)
i) \( op + = (\text{int } a, b) \text{ int} : a - - b ; \)
j) \( op + = (\text{int } a) \text{ int} : a ; \)
k) \( op \text{ abs } = (\text{int } a) \text{ int} : (a < L_0 \mid -a \mid a) ; \)
l) \( op \times = (\text{int } a, b) \text{ int} : (\text{int } s := L_0, t := \text{ abs } b ; \)
while \( i \geq L_1 \text{ do}(s := s + a ; i := i - L_1) ; (b < L_0 \mid -s \mid s) ) ; \)
m) \( op \div = (\text{int } a, b) \text{ int} : \)
\( (b \neq L_0 \mid \text{ int } q := L_0, r := \text{ abs } a ; \)
while \( r := r - \text{ abs } b) \geq L_0 \text{ do } q := q + L_1 ; \)
\( (a < L_0 \land b \geq L_0 \lor a \geq L_0 \land b < L_0 \mid -q \mid q)) ; \)
n) \( op \mod = (\text{int } a, b) \text{ int} : a - a \div b \times b + (a < 0 \mid \text{ abs } b \mid 0) ; \)
o) \( op \div = (\text{int } a, b) \text{ real} : (\text{real } c = a, d = b ; c / d) ; \)
p) \( op \div = (\text{int } a, b) \text{ int} : \)
\( (b \geq 0 \mid \text{ int } p := L_1 ; \text{ to } b \text{ do } p := p \times a ; p) ; \)
q) \( op \text{ leng } = (\text{int } a) \text{ long } \text{ int} : a \text{ the long } L \text{ integral value equivalent} \)
to the value of 'a' \( a ; \) \( \{2.2.3.1.d\} \)
r) \( op \text{ short } = (\text{long } \text{ int } a) \text{ int} : a \text{ the } L \text{ integral value, if it exists,} \)
equivalent to the value of 'a' \( a ; \) \( \{2.2.3.1.d\} \)
s) \( op \text{ odd } = (\text{int } a) \text{ bool} : \text{ abs } a \mod L_0 \mid 2 = L_1 ; \)
t) \( op \text{ sign } = (\text{int } a) \text{ int} : (a > L_0 \mid 1 \mid a < L_0 \mid -1 \mid 0) ; \)
u) \( op \text{ int } = (\text{int } a, b) \text{ compl} : (a, b) ; \)

10.2.4. Operations on real operands

a) \( op < = (\text{real } a, b) \text{ bool} : a \text{ true if the value of } 'a' \text{ is} \)
smaller than that of 'b' and false otherwise \( a ; \) \( \{2.2.3.1.c\} \)
b) \( op \leq = (\text{real } a, b) \text{ bool} : \neg(b < a) ; \)
c) \( op = = (\text{real } a, b) \text{ bool} : a \leq b \land b \leq a ; \)
d) \( op \neq = (\text{real } a, b) \text{ bool} : \neg(a = b) ; \)
e) \( op \leq = (\text{real } a, b) \text{ bool} : b \leq a ; \)
f) \( op \geq = (\text{real } a, b) \text{ bool} : b > a ; \)
g) \( op \div = (\text{real } a, b) \text{ real} : a \text{ the value of } 'a' \text{ minus that of } 'b' \)
\( \{2.2.3.1.c\} ; \)
h) \( op \div = (\text{real } a) \text{ real} : L_0 - a ; \)
10.2.4. continued

i) \( \text{op} + = (L \text{ real } a, b) \ L \text{ real } : a - - b \);

j) \( \text{op} + = (L \text{ real } a) \ L \text{ real } : a \);

k) \( \text{op abs} = (L \text{ real } a) \ L \text{ real } : (a < L0 \mid -a \mid a) \);

l) \( \text{op } \times = (L \text{ real } a, b) \ L \text{ real } : \text{ the value of 'a' times that of 'b' } \);

m) \( \text{op} / = (L \text{ real } a, b) \ L \text{ real } : \text{ the value of 'a' divided by that of 'b' } \);

n) \( \text{op} \text{ long} = (L \text{ real } a) \ \text{long } L \text{ real } : \text{ the long L real value equivalent to the value of 'a' } \);

o) \( \text{op short} = (L \text{ real } a) \ L \text{ real } : a \text{ the L real value, if it exists, equivalent to the value of 'a' } \);

p) \( \text{op round} = (L \text{ real } a) \ L \text{ int } : a \text{ an integral value, if one exists, equivalent to a L real value differing by not more than one-half from the value of 'a' } \);

q) \( \text{op sign} = (L \text{ real } a) \ L \text{ int } : (a > L0 \mid 1 \mid a < L0 \mid -1 \mid 0) \);

r) \( \text{op enter} = (L \text{ real } a) \ L \text{ int } : (L \text{ int } j := L0 ; (j < a \mid e : j := j + L1 ; (j < a \mid e \mid j = L1)) | (j > a \mid s : j := j - L1 ; (j > a \mid s \mid j = L1)) \)

s) \( \text{op } \vdash = (L \text{ real } a, b) \ L \text{ comp } : (a, b) \);

10.2.5. Operations on arithmetic operands

a) \( \text{op } \text{ P} = (L \text{ real } a, b) \ L \text{ real } : (L \text{ real } c := b ; c \ P \ a) \);

b) \( \text{op } \text{ P} = (L \text{ int } a, L \text{ real } b) \ L \text{ real } : (L \text{ real } c := a ; c \ P \ b) \);

c) \( \text{op } \text{ P} = (L \text{ real } a, L \text{ int } b) \ \text{bool} : (L \text{ real } c := b ; c \ P \ a) \);

d) \( \text{op } \text{ P} = (L \text{ int } a, L \text{ real } b) \ \text{bool} : (L \text{ real } c := a ; c \ P \ b) \);

e) \( \text{op } \text{ P} = (L \text{ real } a, L \text{ int } b) \ L \text{ compl } : (L \text{ real } c := b ; a \ L \ c) \);

f) \( \text{op } \text{ P} = (L \text{ int } a, L \text{ real } b) \ L \text{ compl } : (L \text{ real } c := a ; c \ L \ b) \);

g) \( \text{op } \text{ P} = (L \text{ real } a, \ \text{int} \ b) \ L \text{ real } : (L \text{ real } p := L1 ;
能使 \text{abs b do } p := p \times a ; (b \geq 0 \mid p \mid L1 / p) \)

10.2.6. Operations on character operands

a) \( \text{op } \text{ P} = (\text{char } a, b) \ \text{bool} : \text{ abs a < abs b} \) \" \text{(10.1.b)} \);

b) \( \text{op } \text{ P} = (\text{char } a, b) \ \text{bool} : \neg(b < a) \);

c) \( \text{op } \text{ P} = (\text{char } a, b) \ \text{bool} : a \leq b \ \text{and} \ b \leq a \);

d) \( \text{op } \text{ P} = (\text{char } a, b) \ \text{bool} : \neg(a = b) \).
10.2.6. continued

e) \texttt{op} \geq = (\texttt{char} \ a, \ b) \ \texttt{bool} : \ b \leq a ;
f) \texttt{op} > = (\texttt{char} \ a, \ b) \ \texttt{bool} : \ b < a ;

10.2.7. Complex structures and associated operations

a) \texttt{struct} \ L \ \texttt{compl} = (L \ \texttt{real} \ \texttt{re}, \ \texttt{im}) ;
b) \texttt{op} \ \texttt{re} = (L \ \texttt{compl} \ a) \ L \ \texttt{real} : \ \texttt{re} \ \texttt{of} \ a ;
c) \texttt{op} \ \texttt{im} = (L \ \texttt{compl} \ a) \ L \ \texttt{real} : \ \texttt{im} \ \texttt{of} \ a ;
d) \texttt{op} \ \texttt{abs} = (L \ \texttt{compl} \ a) \ L \ \texttt{real} : \ \texttt{sqrt}(\texttt{re} \ a + 2 \ \texttt{im} \ a + 2) ;
e) \texttt{op} \ \texttt{conj} = (L \ \texttt{compl} \ a) \ L \ \texttt{compl} : \ \texttt{re} a \perp - \ \texttt{im} a ;
f) \texttt{op} = = (L \ \texttt{compl} \ a, \ b) \ \texttt{bool} : \ \texttt{re} a = \texttt{re} b \land \texttt{im} a = \texttt{im} b ;
g) \texttt{op} \ne = (L \ \texttt{compl} \ a, \ b) \ \texttt{bool} : - (a = b) ;
h) \texttt{op} + = (L \ \texttt{compl} \ a) \ L \ \texttt{compl} : \ a ;
i) \texttt{op} - = (L \ \texttt{compl} \ a) \ L \ \texttt{compl} : - \ \texttt{re} a \perp - \ \texttt{im} a ;
j) \texttt{op} + = (L \ \texttt{compl} \ a, \ b) \ L \ \texttt{compl} : (\texttt{re} a + \texttt{re} b) \perp (\texttt{im} a + \texttt{im} b) ;
k) \texttt{op} - = (L \ \texttt{compl} \ a, \ b) \ L \ \texttt{compl} : (\texttt{re} a - \texttt{re} b) \perp (\texttt{im} a - \texttt{im} b) ;
l) \texttt{op} \times = (L \ \texttt{compl} \ a, \ b) \ L \ \texttt{compl} :
\begin{align*}
& (\texttt{re} a \times \texttt{re} b - \texttt{im} a \times \texttt{im} b) \perp (\texttt{re} a \times \texttt{im} b + \texttt{im} a \times \texttt{re} b) ;
\end{align*}
m) \texttt{op} / = (L \ \texttt{compl} \ a, \ b) \ L \ \texttt{compl} :
\begin{align*}
& (L \ \texttt{real} \ d = \texttt{re}(b \times \texttt{conj} \ b) ; L \ \texttt{compl} \ n = a \times \texttt{conj} \ b ;
& \ (\texttt{re} n / d) \perp (\texttt{im} n / d)) ;
\end{align*}

n) \texttt{op} \ \texttt{leng} = (L \ \texttt{compl} \ a) \ \texttt{long} L \ \texttt{compl} : \ \texttt{leng} \ \texttt{re} a \perp \ \texttt{leng} \ \texttt{im} a ;
o) \texttt{op} \ \texttt{short} = (\texttt{long} L \ \texttt{compl} \ a) L \ \texttt{compl} : \ \texttt{short} \ \texttt{re} a \perp \ \texttt{short} \ \texttt{im} a ;
p) \texttt{op} \ \texttt{P} = (L \ \texttt{compl} \ a, L \ \texttt{int} b) L \ \texttt{compl} : (L \ \texttt{compl} \ c = b ; a \ \texttt{P} a) ;
q) \texttt{op} \ \texttt{P} = (L \ \texttt{compl} \ a, L \ \texttt{real} b) L \ \texttt{compl} : (L \ \texttt{compl} \ c = b ; a \ \texttt{P} a) ;
r) \texttt{op} \ \texttt{P} = (L \ \texttt{int} a, L \ \texttt{compl} b) L \ \texttt{compl} : (L \ \texttt{compl} \ c = a ; a \ \texttt{P} b) ;
s) \texttt{op} \ \texttt{P} = (L \ \texttt{real} a, L \ \texttt{compl} b) L \ \texttt{compl} : (L \ \texttt{compl} \ c = a ; a \ \texttt{P} b) ;
t) \texttt{op} + = (L \ \texttt{compl} \ a, int b) L \ \texttt{compl} : (L \ \texttt{compl} \ p := L L ;
\begin{align*}
& \texttt{to abs} \ b \ \texttt{do} \ p := p \times a ; (b \geq 0 \ | \ p \ | L L / p)) ;
\end{align*}

10.2.8. Bits structures and associated operations

a) \texttt{struct} \ L \ \texttt{bits} = ([1 : L \ \texttt{bits width}] \ \texttt{bool} L F) ; \{\texttt{See 10.1.g}\}
\begin{itemize}
\item The field-selector is hidden from the user in order that he may not break open the structure; in particular, he may not subscript the field.
\end{itemize}
10.2.8. continued

b) \( \text{op} = (L \text{ bits } a, b) \text{ bool } : \)
   \[
   \text{true} \quad \text{if } (L \text{ of } a)[i] \neq (L \text{ of } b)[i] \quad \text{for } i \text{ to } L \text{ bits width do } \]
   \[
   \text{false} \quad \text{otherwise} \; ; \; a \]

c) \( \text{op} \neq (L \text{ bits } a, b) \text{ bool } : -(a = b) \; ; \)

\[ \]

d) \( \text{op} \lor (L \text{ bits } a, b) \text{ L bits } : \)
   \[
   \text{true} \quad \text{for } i \text{ to } L \text{ bits width do } \]
   \[
   (L \text{ of } a)[i] \; \lor \; (L \text{ of } b)[i] \; ; \; a \]

e) \( \text{op} \land (L \text{ bits } a, b) \text{ L bits } : \)
   \[
   \text{true} \quad \text{for } i \text{ to } L \text{ bits width do } \]
   \[
   (L \text{ of } a)[i] \; \land \; (L \text{ of } b)[i] \; ; \; a \]

f) \( \text{op} \leq (L \text{ bits } a, b) \text{ bool } : (a \lor b) = b \; ; \)

g) \( \text{op} \geq (L \text{ bits } a, b) \text{ bool } : b \leq a \; ; \)

h) \( \text{op} + (L \text{ bits } a, \text{ int } b) \text{ L bits } : \)
   \[
   (\text{abs } b \leq L \text{ bits width } \text{ then } \text{bits } c := a \; ; \; \text{to } \text{abs } b \text{ do } \]
   \[
   (b > 0 \; \text{ for } i \text{ from } 1 \text{ to } L \text{ bits width do } (L \text{ of } a)[i-1] := \]
   \[
   (L \text{ of } a)[i] \; ; \; (L \text{ of } a)[L \text{ bits width}] := \text{false} \; \text{ for } i \text{ from } L \text{ bits width by } -1 \text{ to } 2 \text{ do } (L \text{ of } a)[i] := \]
   \[
   (L \text{ of } a)[i-1] \; ; \; (L \text{ of } a)[1] := \text{false} \; ; \; a \; \text{fi} \; ; \]

i) \( \text{op abs } (L \text{ bits } a) \text{ L int } : \)
   \[
   (L \text{ int } a := L0 \; \text{ for } i \text{ to } L \text{ bits width do } \]
   \[
   c := L2 \times c + \text{abs}(L \text{ of } a)[i] \; ; \; a \; \text{fi} \; ; \]

j) \( \text{op bin } (L \text{ int } a) \text{ L bits } : \text{if } a > 0 \text{ then } \)
   \[
   L \text{ int } b := a \; ; \; L \text{ bits } a \; \text{ for } i \text{ from } L \text{ bits width by } -1 \text{ to } 1 \text{ do } \]
   \[
   ((L \text{ of } a)[i] := \text{odd } b \; \text{ for } i \text{ from } L \text{ bits width by } -1 \text{ to } 1 \text{ do } \]
   \[
   a \; \text{fi} \; ; \]

k) \( \text{op elem } (\text{int } n, L \text{ bits } b) \text{ bool } : \text{L } \text{ of } b[a] \; ; \)

l) \( \text{op L bb } ([1:] \text{ bool } a) \text{ L bits } : \)
   \[
   (\text{int } n = \text{upb } a \; \text{ for } i \text{ to } L \text{ bits width do } \]
   \[
   (L \text{ of } a)[i] := \text{false} \; \text{ for } i \text{ to } L \text{ bits width do } \]

10.2.9. Bytes and associated operations

a) \( \text{struct } L \text{ bytes } = ([1:] \text{ L bytes width } \text{ char } LF) \; ; \; \text{See } 10.2.8.a \text{ and } \)

b) \( \text{op } \text{le } (L \text{ bytes } a, b) \text{ bool } : (\text{string } := a) < (\text{string } := b) \; ; \)

c) \( \text{op } \text{le } (L \text{ bytes } a, b) \text{ bool } : -(b < a) \; ; \)

d) \( \text{op } = (L \text{ bytes } a, b) \text{ bool } : a \leq b \land b \leq a \; ; \)

10.2.9. continued

e) \textit{op \&} = (\text{L bytes } a, b) \text{ bool } : \neg (a = b) ;
f) \textit{op \&} = (\text{L bytes } a, b) \text{ bool } : b \leq a ;
g) \textit{op \&} = (\text{L bytes } a, b) \text{ bool } : b < a ;
h) \textit{op \&} = (\text{int } a, \text{ L bytes } b) \text{ char } : (\text{L F of } b)[a] ;
i) \textit{op \&} = (\text{string } a) \text{ L bytes } :
   \begin{aligned}
   (\text{int } n = \text{upb } a ; (n \leq \text{L bytes width } \text{L bytes } c) ;
   \text{for } i \text{ to } \text{L bytes width } \text{do } (\text{L F of } c)[i] :=
   (i \leq n \mid a[i] \text{ null character}) ;
   ) ;
   \end{aligned}

10.2.10. Strings and associated operations

a) \textit{mode string } = [l : \text{flex } ] \text{ char } ;
b) \textit{op < } = (\text{string } a, b) \text{ bool } : \neg (b < a ) ;
c) \textit{op \&} = (\text{string } a, b) \text{ bool } : a \leq b \wedge b \leq a ;
d) \textit{op \&} = (\text{string } a, b) \text{ bool } : a \leq b ;
e) \textit{op \&} = (\text{string } a, b) \text{ bool } : a = b ;
f) \textit{op > } = (\text{string } a, b) \text{ bool } : b < a ;
g) \textit{op > } = (\text{string } a, b) \text{ bool } : b < a ;
h) \textit{op R } = (\text{string } a, \text{ char } b) \text{ bool } : (\text{string } c = b ; a \text{ R } b) ;
i) \textit{op R } = (\text{char } a, \text{ string } b) \text{ bool } : (\text{string } c = a ; a \text{ R } b) ;
j) \textit{op + } = (\text{string } a, b) \text{ string } :
   \begin{aligned}
   (\text{int } m = \text{upb } a , n = \text{upb } b ; [m \wedge m + n ] \text{ char } c ;
   a[l : m] := a ; c[m + 1 : m + n] := b ; c) ;
   \end{aligned}
k) \textit{op + } = (\text{string } a, \text{ char } b) \text{ string } : (\text{string } s = b ; a + c) ;
l) \textit{op + } = (\text{char } a, \text{ string } b) \text{ string } : (\text{string } s = a ; s + b) ;

\{\text{The operations defined in } b, h \text{ and } i \text{ imply that if } \text{abs "a"} < \text{abs "b"},
\text{then "" < "a" ; "a" < "b" ; "aa" < "ab" ; "aa" < "ba" ; "ab" < "b". }\}

10.2.11. Operations combined with assignments

a) \textit{op minus } = (\text{ref } \text{L int } a, \text{L int } b) \text{ ref } \text{L int } : a := a - b ;
b) \textit{op minus } = (\text{ref } \text{L real } a, \text{L real } b) \text{ ref } \text{L real } : a := a - b ;
c) \textit{op minus } = (\text{ref } \text{L compl } a, \text{L compl } b) \text{ ref } \text{L compl } : a := a - b ;
10.2.11. continued

d) \text{op plus} = (\text{ref L int } a, \text{ L int } b) \text{ ref L int } : a := a + b ;
e) \text{op plus} = (\text{ref L real } a, \text{ L real } b) \text{ ref L real } : a := a + b ;
f) \text{op plus} = (\text{ref L compl } a, \text{ L compl } b) \text{ ref L compl } : a := a + b ;
g) \text{op times} = (\text{ref L int } a, \text{ L int } b) \text{ ref L int } : a := a \times b ;
h) \text{op times} = (\text{ref L real } a, \text{ L real } b) \text{ ref L real } : a := a \times b ;
i) \text{op times} = (\text{ref L compl } a, \text{ L compl } b) \text{ ref L compl } : a := a \times b ;
j) \text{op over} = (\text{ref L int } a, \text{ L int } b) \text{ ref L int } : a := a / b ;
k) \text{op mod} = (\text{ref L int } a, \text{ L int } b) \text{ ref L int } : a := a \mod b ;
l) \text{op over} = (\text{ref L real } a, \text{ L real } b) \text{ ref L real } : a := a / b ;
m) \text{op over} = (\text{ref L compl } a, \text{ L compl } b) \text{ ref L compl } : a := a / b ;
n) \text{op Q} = (\text{ref L real } a, \text{ L int } b) \text{ ref L real } : a \ Q (\text{L real } b) ;
o) \text{op Q} = (\text{ref L compl } a, \text{ L int } b) \text{ ref L compl } : a \ Q (\text{L compl } b) ;
p) \text{op Q} = (\text{ref L compl } a, \text{ L real } b) \text{ ref L compl } : a \ Q (\text{L compl } b) ;
q) \text{op plus} = (\text{ref string } a, \text{ string } b) \text{ ref string } : a := a + b ;
r) \text{op plus} = (\text{ref string } a, \text{ char } b) \text{ ref string } : a := a + b ;
s) \text{op plus} = (\text{ref string } a, \text{ char } b) \text{ ref string } : a := b + a ;
t) \text{op plus} = (\text{ref string } a, \text{ char } b) \text{ ref string } : a := b + a ;

10.3. Standard mathematical constants and functions

a) \text{L real } \pi = \text{a a L real value close to } \pi ; \text{ see Math. of Comp. v. 16, 1962, pp. 80-99 a ;}
b) \text{proc L sqrt } = (L \text{ real } x) \text{ L real } : a \text{ if } x > 0, \text{ a L real value close to the square root of } 'x' a ;
c) \text{proc L exp } = (L \text{ real } x) \text{ L real } : a \text{ a L real value, if one exists, close to the exponential function of } 'x' a ;
d) \text{proc L ln } = (L \text{ real } x) \text{ L real } : a \text{ a L real value, if one exists, close to the natural logarithm of } 'x' a ;
e) \text{proc L cos } = (L \text{ real } x) \text{ L real } : a \text{ a L real value close to the cosine of } 'x' a ;
f) \text{proc L arccos } = (L \text{ real } x) \text{ L real } : a \text{ if abs } x \leq 1, \text{ a L real value close to the inverse cosine of } 'x' , \text{ L arccos}(x) \leq L \pi a ;
g) \text{proc L sin } = (L \text{ real } x) \text{ L real } : a \text{ a L real value close to the sine of } 'x' a ;
10.3. continued

h) \textit{proc} \( L \) \( \text{arasin} = (L \text{real } x) \) \( L \text{real} : a \) if \( \text{abs } x \leq L1 \), a \( L \text{real} \) value
   close to the inverse sine of 'x', \( \text{abs } L \text{arasin}(x) \leq L \pi / L2 a \);

i) \textit{proc} \( L \) \( \text{tan} = (L \text{real } x) \) \( L \text{real} : a \) a \( L \text{real} \) value, if one exists, close to the tangent of 'x';

j) \textit{proc} \( L \) \( \text{aratan} = (L \text{real } x) \) \( L \text{real} : a \) a \( L \text{real} \) value close to the inverse tangent of 'x',
   \( \text{abs } L \text{aratan}(x) \leq L \pi / L2 a \);

k) \textit{proc} \( L \) \( \text{random} = L \text{real} \text{expr} a \) the next pseudo-random \( L \text{real} \) value
   from a uniformly distributed sequence on the interval \((L0, L1) a\);

l) \textit{proc} \( L \) \( \text{set random} = (L \text{real } x) : a \) the next call of \( L \text{random} \) is
   made to deliver the value of 'x' \( a ; L \text{random} ) ;

10.4. Synchronization operations

a) \( \text{op down} = (\text{ref int } \text{dijkstra}) : (\text{do}(\text{if } \text{dijkstra} \geq 1 \text{ then}
   \text{dijkstra minus } 1 ; l \text{ else } a \text{ if the closed-statement replacing this}
   \text{comment is contained in a unitary-phrase which is a constituent}
   \text{unitary-phrase of the smallest collateral-phrase, if any, beginning}
   \text{with a parallel-symbol and containing this closed-statement, then}
   \text{the elaboration of that unitary-phrase is halted \{6.0.2.c\} ;}
   \text{otherwise, the further elaboration is undefined } a \text{ fi} ) ; l : \text{skip} ) ;

b) \( \text{op up} = (\text{ref int } \text{dijkstra}) : (\text{dijkstra plus } 1 ; a \text{ the elaboration}
   \text{is resumed of all phrases whose elaboration is not terminated but}
   \text{is halted because the name possessed by 'dijkstra' referred to a}
   \text{value smaller than one } a ) ;

\{\text{See 2.2.5; for insight into the use of down and up, see E.W. Dijkstra,}
\text{Cooperating Sequential Processes, EWD123, Tech. Univ. Eindhoven, 1965,}
\text{and also 11.13. }\}
10.5. Transput declarations {"So it does!" said Pooh. "It goes in!"
"So it does!" said Piglet. "And it comes out!"
"Doesn't it?" said Eeyore. "It goes in
and out like anything."}
Winnie-the-Pooh, A.A. Milne.)

10.5.0. Transput modes and straightening

10.5.0.1. Transput modes

a) \texttt{mode \% simplout = union(\* int, \* real, \* compl,}
\texttt{ bool, char, string);}

b) \texttt{mode \% outtype = union(\* D L int, \* D L real, \* D bool,}
\texttt{ \* D char, \* D outstruct);}

c) \texttt{mode \% outstruct = a actual-declarer specifying a mode united}
\texttt{ from \{4.4.3.a\} all modes, except that specified by tarrof,}
\texttt{ which are structured from \{2.2.4.1.d\} only modes from which the}
\texttt{ mode specified by outtype is united a;}

d) \texttt{mode \% tarrof = struct(string \*}); \texttt{(See the remarks under 5.5.8);} e) \texttt{mode \% intype = union(\* ref D L int, \* ref D L real,}
\texttt{ \* ref D bool, \* ref D char, \* ref D outstruct);}

10.5.0.2. Straightening

a) \texttt{op \% straightout = (outtype x) \[] simplout :}
\texttt{ a the result of "straightening" 'x' \a ;}

b) \texttt{op \% straightin = (intype x) \[] ref simplout :}
\texttt{ a the result of straightening 'x' \a ;}

The result of straightening a given value is a multiple value obtained
in the following steps:

Step 1: If the given value is (refers to) a value from whose mode that
specified by simplout is united, then the result is a new instance of
a multiple value composed of a descriptor consisting of an offset
1 and one quintuple (1, 1, 1, 1, 1) and the given value as its only
element, and Step 4 is taken;

Step 2: If the given value is (refers to) a multiple value, then, letting
\texttt{n} stand for the number of elements of that value, and \texttt{y_i}
for the result of straightening its \texttt{i}-th element, Step 3 is taken; otherwise,
letting \texttt{n} stand for the number of fields of the (of the value referred
to by the) given value, and \texttt{y_i}
for the result of straightening its \texttt{i}-th
field, Step 3 is taken;
10.5.0.2. continued

Step 3: If the given value is not (is) a name, then, letting $m_i$ stand for the number of elements of $y_i$, the result is a new instance of a multiple value composed of a descriptor consisting of an offset 1 and one quintuple $(1, m_1 + \ldots + m_n, 1, 1, 1)$ and elements, the $l$-th of which, where $l = m_1 + \ldots + m_{k-1} + j$, is the (is the name referring to the) $j$-th element of $y_k$ for $k = 1, \ldots, n$ and $j = 1, \ldots, m_k$.

Step 4: If the given value is not (is) a name, then the mode of the result is 'row of' (row of reference to') followed by the mode specified by $simplout$.

10.5.1. Channels and files

(aa) "Channels", "backfiles" and files model the transput devices of the physical machine used in the implementation.

(bb) A channel corresponds to a device, e.g. a card reader or punch, a magnetic drum or disc, to part of a device, e.g. a piece of core memory, the keyboard of a teleprinter, or to a number of devices, e.g. a bank of tape units or even a set-up in nuclear physics the results of which are collected by the computer. A channel has certain properties (10.5.1.1.d: 10.5.1.1.n).

A "random access" channel is one for which set possible (10.5.1.1.e) is true, and a "sequential acces" channel is one for which set possible is false.

The transput devices of some physical machine may be seen in more than one way as channels with properties. The choice made in an implement­ation is a matter for individual taste. Some possible choices are given in table I.

(cc) All information on a given channel is to be found in a number of backfiles. A backfile (10.5.1.1.b) comprises a three-dimensional array of integers (bytes of information), the book of the backfile; the lower bounds of the book are all one, the upperbounds are nonnegative integers, the maxpage, maxline and maxchar of the backfile; furthermore, the back­file comprizes the position of the "end of file", i.e. the page number, line number and char number up to which the backfile is filled with in­formation, the current position and the "identification-string" of the backfile.
<table>
<thead>
<tr>
<th>properties</th>
<th>card reader</th>
<th>card punch</th>
<th>magnetic tape unit</th>
<th>line printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset possible</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>set possible</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>get possible</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>put possible</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>bin possible</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>idf possible</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>max page</td>
<td>1</td>
<td>1</td>
<td>very large</td>
<td>very large</td>
</tr>
<tr>
<td>max line</td>
<td>large</td>
<td>very large</td>
<td>16</td>
<td>large</td>
</tr>
<tr>
<td>max char</td>
<td>72</td>
<td>80</td>
<td>84</td>
<td>60</td>
</tr>
<tr>
<td>stand conv</td>
<td>a 48- or 64-character code</td>
<td>64-char code</td>
<td>some code</td>
<td>line-pr code</td>
</tr>
<tr>
<td>max mb files</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>properties</th>
<th>magnetic disc</th>
<th>magnetic drum</th>
<th>paper tape reader</th>
<th>tape punch</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset possible</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>set possible</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>get possible</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>put possible</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>bin possible</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>idf possible</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>max page</td>
<td>200</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>max line</td>
<td>16</td>
<td>1</td>
<td>256</td>
<td>very large</td>
</tr>
<tr>
<td>max char</td>
<td>128</td>
<td>524288</td>
<td>256</td>
<td>80</td>
</tr>
<tr>
<td>stand conv</td>
<td>some code</td>
<td>some code</td>
<td>some code</td>
<td>5-hole code</td>
</tr>
<tr>
<td>max mb files</td>
<td>10</td>
<td>4</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>
10.5.1. continued

dd) After the elaboration of the declaration of `chainbfile' (10.5.1.1.c), all backfiles form the chains of backfiles referenced by `chainbfile', each backfile chained to the next one by its field `next'.

Examples:

a) In a certain implementation, channel six is a line printer. It has no input information, `chainbfile [6]` is initialized to refer to a backfile the book of which is an integer array with upper bounds 2000, 60 and 144 (2000 pages of continuous stationery), with both the current position and the end of file at (1, 1, 1) and `next` equal to nil. All elements of the book are left undefined.

b) Channel four is a drum, divided into 32 segments each being one page of 256 lines of 256 bytes. It has 32 backfiles of input information (the previous contents of the drum), so `chainbfile [4]` is initialized to refer to the first backfile of a chain of 32 backfiles, the last one having `next` equal to nil. Each of those backfiles has an end of file at position (2, 1, 1).

c) Channel twenty is a tape unit. It can accommodate one tape at a time; one input tape is mounted and another tape laid in readiness. Here, `chainbfile [20]` is initialized to refer to a chain of two backfiles. Since it is part of the standard declarations, all input is part of the program, though not of the particular-program.

ee) A file (10.5.1.2.a) is a structured value which comprises a reference to a backfile, and the information necessary for the transput routines to work with that backfile. A backfile is associated with a file by means of `open` (10.5.1.2.b), `create` (10.5.1.2.c) or `establish` (10.5.1.2.d).

A given channel can accommodate a certain number (10.5.1.1.n) of backfiles at any stage of the elaboration. The association is ended by means of `scratch` (10.5.1.2.u), `close` (10.5.1.2.s) or `look` (10.5.1.2.t).

ff) When a file is "opened" on a channel for which `idf possible` is false, then the first backfile is taken from the chain of bfiles for that channel, and is made the `bfile` of the file, obliterating the previous backfile, if any, of the file.

When a file is opened on a channel for which `idf possible` is true, then the first backfile which has the given identification string is taken from the chain of backfiles for the channel; this backfile is made the `bfile` of the file.
10.5.1. continued 2

6g) When a file is "established" on a channel, then a backfile is generated (8.5) with a book of the given size, the given identification-string and both the current position and the end of file at (1, 1, 1); when a file is "created" on a channel, then a file is established with a backfile the book of which has the maximum size for the channel and an empty string as its identification string.

hh) When a file is "scratched", then its associated backfile is obliterated.

ii) When a file is "closed", then first, if the given identification string is not empty, then the identification of the backfile of the file is replaced by that string; next, it is attached to the chain referenced by chainbfile of the channel. Another file may now be opened with this backfile by a suitable call of open.

jj) When a file is "locked", then first, if the given identification string is not empty, then the identification of the backfile of the file is replaced by that string; next, it is attached to the chain referenced by lockedbfile of the channel. No file can now be opened with this backfile.

kk) A file comprizes some fields of the mode 'procedure-boolean', 'procedure-with-reference-to-character-parameter-boolean' or 'procedure-with-integral-parameter-boolean', routines which are called when in transput certain error situations arise. After opening or creating a file, the routines provided yield the value false when called, but the programmer may assign other routines to those fields. If the elaboration of such a routine is terminated, then the transput routine which called it can take no further action; otherwise, if it yields the value true, then it is assumed that the error situation has been remedied in some way, and, if possible, transput goes on, but if it yields the value false, then undefined is called, i.e., some sensible system action is taken (rr).

These routines are:

a) logical file end, which is called when during input from a file on a sequential channel the end of file of its backfile is passed. If the routine yields the value true, then transput goes on, and if it yields false, then some sensible action is taken.
10.5.1. continued 3

Example:
The programmer wishes to count the number of integers on his input-
tape. The file \texttt{intape} was opened in a surrounding range. If he writes
\begin{verbatim}
begin int n := 0 ; logical file end of intape := goto f ;
do(get(intape, loc int) ; n plus 1) ; f : print (n)
end,
\end{verbatim}
then the assignment to the field of \texttt{intape} violates the scope restrict-
ions (; the scope of the routine (: goto f) is smaller than the scope
of \texttt{intape}), so he has to write
\begin{verbatim}
begin int n := 0 ; file auxin := intape ;
logical file end of auxin := goto f ;
do(get(auxin, loc int) ; n plus 1) ; f : print (n)
end.
\end{verbatim}

b) \texttt{physical file end}, which is called when the \texttt{maxpage}, the \texttt{maxline} or
the \texttt{maxchar} of the backfile of a file is exceeded.
If the routine yields the value true, then transput goes on, and if
it yields false, then some sensible action is taken.

Example:
The programmer wishes automatically to give a new line at the end of
a line and a new page at the end of a page on his file \texttt{f}:
\begin{verbatim}
proc new line page = bool : [(line ended (f) | new line (f)) ;
(page ended (f) | new page (f)) ; true) ;
\end{verbatim}

c) \texttt{char error}, which is called when during input a character is read
which does not agree with the frame specifying it, with as its
actual-parameter a suggested character to replace it(5.5.1.mm).
If the frame is a digit- (zero-, point-, exponent-, complex-) frame
and if the character read is not a digit (digit or space, point-symbol,
times-ten-to-the-power-symbol, plus-i-times-symbol) then the routine
is called with a digit-zero- (digit-zero-, point-, times-ten-to-the-
power-, plus-i-times-) symbol. The routine provided by the programmer
may give some other character instead of the suggested one. If the
routine yields true, then that suggested character as possibly modified
by the routine is used, and, if it yields false, then the original
suggested character is used.
Example:
The programmer wishes to print a list of all such disagreements. He assigns to the field `char error` of his file `f`

```c
((ref char sugg) bool : 
  char k ; backspace (f) ; int p = page number (f),
  l = line number (f), c = char number (f) ; get (f, k) ;
  print ((new line, "at", p, l, c, "present", "", k,
         "", _ suggested "", sugg, """" ) ) ; false)
```

da) `value error`, which is called when during formatted transput an attempt is made to transput a value under control of a picture with which it is not compatible, or when the number of frames is not sufficient. If the routine yields true, then the current value and picture are skipped, i.e., transput goes on at 5.5.1.dd Step 5; if the routine yields false, then first, on output, the value is output by `put`, and next some sensible action is taken.

d) `format end`, which is called when during formatted transput the format is exhausted while still some value remains to be transput.

If the routine yields true, then transput goes on (so the routine must have provided a new current format of the file), and, if the routine yields false, then the current format is repeated, i.e., its first picture again is made to be the current picture of the file.

e) `other error`, which is called with some actual-integral-parameter, when during transput, some other error situation arises.

No call of this routine occurs explicitly in the standard-prelude, and neither the meaning of its actual-parameter nor that of the value yielded, is defined in this Report.

This routine may, in some implementation, be called when an incorrigible hardware error occurs which makes transput involving this file impossible. (The programmer may provide a routine which then closes the file and opens it on some other channel.)

e) The `conv` of a file is used in conversion; after opening, creating or establishing a file, `stand conv` of the channel is used, but an other "conversion key" may be provided by the programmer.
On output, the given character is converted to that integer, if any, in the conversion key, whose ordinal number is the integral equivalent of that character;

on input, the given integer is converted to that character, if any, whose integral equivalent is the lowest ordinal number for which the element of the conversion key is equal to that given integer.

What action is taken when an attempt is made to convert a character with an integral equivalent exceeding the upper bound of the conversion key, or to convert an integer which is not equal to some element of the conversion key is left undefined (parity error, nonexistent code).

mm) The term of a file is used in reading strings of a variable number of characters, where either the end of line or any of the characters of term serves as a terminator (see 5.5.1.jj and 10.5.2.2.dd). This terminator string may be provided by that programmer.

nn) On a channel for which reset possible is true, a file may be "reset", causing its position to be (1, 1, 1). On a sequential access file the end of file remains at the position up to which the backfile contains information, but when after resetting any output is done, the end of file is first set at the current position.

oo) On a random access channel a file may be "set", causing its position to be the given position.

pp) On files opened on a sequential access channel, binary and nonbinary transput may not be alternated, i.e. after opening, creating or resetting such a file, either is possible, but, once one has taken place on the file, the other may not until the file has been reset again.

qq) On files opened on a sequential access channel for which put possible and get possible both are true, nonbinary input and output may be alternated, but it is not allowed to read past the end of file.

rr) When in transput something happens which is left undefined, for instance by an explicit call of undefined (10.5.1.2.y), this does not imply that the elaboration is catastrophically and immediately terminated, but only that some sensible action is taken which is not or cannot be described by this Report alone, and is generally implementation dependent. For instance, in some implementation it may be possible to set a tape unit to any position within the logical file, even if set possible is false (oo).
Example:

\begin{verbatim}
begin file f1, f2 ; [1 : 10000] int x ; int n ;
open ( f1, "my input", channel 2) ;
f2 := f1 ; a now f1 and f2 can be used interchangeably a
conv of f1 := flexocode ; a flexocode is a string,
defined in the library declarations for this implementation a
conv of f2 := telexcode ;
a now f1 and f2 use different codes a
reset (f1) ; a consequently f2 is reset too a
for i while- logical file ended (f1) do
  (n := i ; get (f1, x[i])) ;
a too bad if there are more than 10000 integers in the input a
reset (f1) ;
for i to n do put (f2, x[i]) ;
reset (f2) ; close (f2, "my output")
a f1 is now closed too a
\end{verbatim}
10.5.1.1. Channels

a) \( \text{int mb channels} = \) an integral-clause indicating the number of transput channels in the implementation;

b) \( \text{struct \% bfile} = ([,\text{int book, int lpage, lline, lchar, page, line, char, npage, nline, nchar, string idf, ref bfile next]);} \)

c) \([1:mb channels] \text{ref bfile \% chainbfile := a some appropriate initialization (see 10.5.1.6a)} \)

d) \([1:mb channels] \text{bool reset possible = a row-of-boolean-clause, indicating which of the physical devices corresponding to the channels allow resetting (e.g. rewinding of a magnetic tape)} \)

e) \([1:mb channels] \text{bool set possible = a row-of-boolean-clause, indicating which devices can be accessed at random} \)

f) \([1:mb channels] \text{bool get possible = a row-of-boolean-clause, indicating which devices can be used for input} \)

g) \([1:mb channels] \text{bool put possible = a row-of-boolean-clause, indicating which devices can be used for output} \)

h) \([1:mb channels] \text{bool bin possible = a row-of-boolean-clause, indicating which devices can be used for binary transput} \)

i) \([1:mb channels] \text{bool idf possible = a row-of-boolean-clause, indicating on which devices backfiles have an identification} \)

j) \([1:mb channels] \text{int max page = a row-of-integral-clause, giving the maximum number of pages per file for the channels} \)

k) \([1:mb channels] \text{int max line = a row-of-integral-clause, giving the maximum number of lines per page} \)

l) \([1:mb channels] \text{int max char = a row-of-integral-clause, giving the maximum number of characters per line} \)

m) \([1:mb channels] \text{ref \[] \text{int \% stand conv = a row-of-reference-to-row-of-integral-clause giving the standard conversion keys for the channels}} \)

n) \([1:mb channels] \text{int max mb files = a row-of-integral-clause, giving the maximum numbers of files the channels can accommodate} \)

o) \([1:mb channels] \text{int \% mb opened files; for i to mb channels do mb opened files [i] := 0;} \)

p) \([1:mb channels] \text{ref bfile \% lookedbfile; for i to mb channels do lookedbfile [i] := nil;} \)
10.5.1.1. continued

q) **proc file available** = (int channel) bool :
   rmb opened files [channel] < max rmb files [channel];

10.5.1.2. Files

a) **struct file** = (ref bfile % bfile, ref int % chan, % forl,
   ref bool % state def, % state get, % state bin, % opened,
   ref string % format, string term, [1:] int com,
   proc bool logical file end, physical file end, format end,
   value error, proc (ref char) bool char error, proc (int) bool
   other error);

b) **proc open** = (ref file file, string idf, int ch) :
   if file available (ch)
   then ref bfile bf := chainbfile [ch], obf := nil ;
      while bf :#: nil do.
         (idf of bf = idf < idf possible [ch] | l | obf := bf ;
          bf := next of bf) ; undefined.
   l : file := (bfile := bf, int := ch, int := 0, bool := false,
      bool, bool, bool := true, nil, stand conv [ch], false,
      false, false, false, (ref char a) bool : false), skip) ;
   (obf := nil | chainbfile [ch] | va3 next of obf) := next of bf ;
   numb opened files [ch] plus 1 ;
   else undefined fi;

c) **proc create** = (ref file file, int ch) :
   establish (file, max page [ch], max line [ch], max char [ch], ch) ;

d) **proc establish** = (ref file file, string idf, int mp, ml, ma, ch) :
   if file available (ch) ^ mp < max page [ch] ^
      ml < max line [ch] ^ ma < max char [ch]
   then bfile bf = ([1 : mp, 1 : ml, 1 : ma] int, 1, 1, 1, 1, 1, 1,
      mp, ml, ma, idf, nil) ;
   file := (bfile := bf, int := ch, int := 0, bool := false,
      bool, bool, bool := true, nil, stand conv[ch], false, false,
      false, false, (ref char a) bool : false), skip) ;
   , rmb opened files [ch] plus 1 ;
   else undefined fi;
10.5.1.2. continued

e) proc set = (file file, int p, l, a):
   if set possible [ chan of file ] \& opened of file
      then page of bfile of file := p ; line of bfile of file := l ;
         char of bfile of file := a ; check plo (file) ;
   else undefined fi;

f) proc reset = (file file) :
   if reset possible [ chan of file ] \& opened of file
      then page of bfile of file := 1 ; line of bfile of file := 1 ;
         char of bfile of file := 1 ; state def of file := false ;
   else undefined fi;

g) proc % check plo = (file file) ; if opened of file
   then (not (logical file ended file) \& logical file end of file |:
             line ended (file) \& page ended (file) \& file ended (file)
             \& physical file end of file | true) \& undefined)
   else undefined fi;

h) proc line ended = (file file) bool : (opened of file |:
   int a = char of bfile of file ; a \leq 0 \vee a > max char of bfile of
   file);

i) proc page ended = (file file) bool : (opened of file |:
   int l = line of bfile of file ; l \leq 0 \vee l > max line of bfile of
   file);

j) proc file ended = (file file) bool : (opened of file |:
   int p = page of bfile of file ; p \leq 0 \vee p > max page of bfile of
   file);

k) proc logical file ended = (file file) bool : (opened of file |:
   set possible [ chan of file ] \& state def of file \& state get of file |:
   bfile b = bfile of file ;
   int p = page of b, lp = page of b, l = line of b, ll = line of b,
   c = char of b, lc = char of b ;
   (p \leq lp | false | p > lp | true | l \leq ll | false | l > ll |
   true | a \leq lc) | false);

l) proc % get string = (file file, ref string s) :
   if get possible [ chan of file ] \& opened of file,
   then ref int p = page of bfile of file, l = line of bfile of file,
      a = char of bfile of file ;
if ¬ set possible [char of file] then state def of file
then (state bin of file | undefined) fi;
state def of file := state get of file := true;
state bin of file := false;
for i to upb s do
  (check plo (file) ; for j to upb conv of file do
    (conv of file)[j] = book of bfile of file [p, l, o] | s[i] :=
    repr j ; e) ; undefined.
e : (e plus 1)
else undefined fi;
m) proc % put string = (file file, string s) : if put possible [char of file] ^ opened of file
then ref int p = page of bfile of file, l = line of bfile of file,
  o = char of bfile of file ;
if ¬ set possible [ch] then state def of file
then (state bin of file | undefined) fi;
state get of file := state bin of file := false;
state def of file := true;
for i to upb s do
  (check plo (file) ; book of bfile of file [p, l, o] :=
    conv of file [abs s[i]] ; o plus 1 ;
    (p = page of bfile of file \& l = line of bfile of file
     | (o > lchar of bfile of file \& lchar of bfile of file := o)
     | lpage of bfile of file := p ; lline of bfile of file := l ;
       lchar of bfile of file := o))
else undefined fi;

n) proc char in string = (char c, ref int i, string s) bool :
  (for k to upb s do (c = s[k] \& i := k ; false, l : true) ;
o) proc space = (file file) :
  (char of bfile of file plus 1 ; check plo (file)).
p) proc backspace = (file file) :
  (char of bfile of file minus 1 ; check plo (file)).
q) proc new line = (file file) :
  (line of bfile of file plus 1 ; char of bfile of file := 1 ;
    check plo (file)).
10.5.1.2. continued 3

r) **procedure new page** = (file file) :
   (page of bfile of file plus 1 ; line of bfile of file := char of bfile of file := 1 ; check plo (file));
s) **procedure close** = (file file, string idf) :
   (opened of file | int oh = chan of file ; (idf != "" | idf of bfile of file := idf) ;
    next of bfile of file := chainbfile[oh] ;
    chainbfile[oh] := bfile := bfile of file ;
    opened of file := false ; mb opened files[oh] minus 1) ;
t) **procedure look** = (file file, string idf) :
   (opened of file | int oh = chan of file ; (idf != "" | idf of bfile of file := idf) ;
    next of bfile of file := lockedfile[oh] ;
    lockedfile[oh] := bfile := bfile of files ;
    opened of file := false ; mb opened files[oh] minus 1) ;
u) **procedure scratch** = (file file) :
   (opened of file | opened of file := false ;
    mb opened files[ohan of file] minus 1) ;
v) **procedure char number** = (file f) int : (opened of f | char of bfile of f) ;
w) **procedure line number** = (file f) int : (opened of f | line of bfile of f) ;
x) **procedure page number** = (file f) int : (opened of f | page of bfile of f) ;
y) **procedure % undefined** = ((false | true) | skip) ; {10.5.1.rr}.

10.5.1.3. Standard channels and files

a) **int stand in channel** = o an integral-clause such that get possible
   [stand in channel] is true and idf possible [stand in channel] is
   false o ;
b) **int stand out channel** = o an integral-clause such that put possible
   [stand out channel] is true and idf possible [stand out channel] is
   false o ;
c) **int stand back channel** = o an integral-clause such that reset possible
   [stand back channel], set possible [stand back channel], get possible
   [stand back channel], put possible [stand back channel] and bin possible
   [stand back channel] are true and idf possible [stand back channel] is
   false o ;
10.5.1.3. continued

da) \texttt{file} \% \texttt{f}; \texttt{open (f, stand in channel)};
\texttt{file stand in} = \texttt{f};
e) \texttt{open (f, stand out channel)};
\texttt{file stand out} = \texttt{f};
f) \texttt{open (f, stand back channel)};
\texttt{file stand back} = \texttt{f};

(Certain "standard files" (d, e, f) need not (and cannot) be opened by the programmer, but are opened for him in the standard declarations; \texttt{print} (10.5.2.1.a) can be used for output on \texttt{stand out}, \texttt{read} (10.5.2.2.a) for input from \texttt{stand in}, and \texttt{write bin} (10.5.4.1.a) and \texttt{read bin} (10.5.4.2.a) for transput involving \texttt{stand back}.}

10.5.2. Formatless transput

10.5.2.1. Formatless output

(For formatless output, \texttt{print} and \texttt{put} can be used. The elements of the given value of the mode specified by \texttt{union (outtype, proc (file))} are treated one after the other; if an element is of the mode specified by \texttt{proc (file)} (i.e. a "layout procedure"), then it is called with the file as its parameter; otherwise, it is straightened (10.5.0.2), and the resulting values are output on the given file one after the other, as follows:

aa) If the mode of the value is specified by \texttt{L int}, then first, if there is not enough room on the line for \texttt{L int width} + 2 characters, then this room is made by giving a new line and, if the page is full, giving a new page; next, when not at the beginning of a line, a space is given and the value is edited as if under control of the picture \texttt{n(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+dn(L int width - 1)d+

bb) If the mode of the value is specified by \texttt{L real}, then, first, if there is not enough room on the line for \texttt{L real width} + \texttt{L expwidth} + 5 characters, then this room is made; next, when not at the beginning of a line, a space is given, and the value is edited as if under control of the picture \texttt{+d.n(L real width - 1)den(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)d+dn(L expwidth - 1)
10.5.2.1. continued

cc) If the mode of the value is specified by \texttt{L comp}, then first, if there is not enough room on the line for \(2 \times (L \text{ real width} + L \text{ exp width} + 5) + 2\) characters, then this room is made; next, when not at the beginning of a new line a space is given, and the value is edited as if under control of the picture \(+d.n(L \text{ real width} - 1)\) \(\text{den}(L \text{ expwidth} - 1)z+d\) \text{ " } i + d.n(L \text{ real width} - 1)\text{den}(L \text{ expwidth} - 1)z+d.

dd) If the mode of the value is specified by \texttt{[]} then its elements are written one after the other.

e) If the mode of the value is specified by \texttt{char} then, first if the line is full room is made; then the character is written.

ff) If the mode of the value is specified by \texttt{bool} then, if the value is true (false) a flip (flop) is output as in ee.

a) \texttt{proc print} = ([]) \texttt{union (outtype, proc (file)) x} :
\begin{verbatim}
put (stand out, x);
\end{verbatim}

b) \texttt{proc put} = (file file, [1 :] \texttt{union (outtype, proc (file)) x}) :
\begin{verbatim}
begin outtype ot ; proc (file) pf ;
for i to upb x do
 (ot := x[i]; pf := x[i] \mid pf (file))
[1 :] simpout y = straightout ot ;
for j to upb y do
 (string s ; bool b ; char c ;
(\$ (L int i ; (i := y[j] \mid
 s := L int string (i, L int width + 1, 10) ;
 sign supp zero (s, 1, L int width + 1)) \$) ;
(\$ (L real x ; (x := y[j] \mid s := L real conv (x))) \$) ;
(\$ (L comp z ; (z := y[j] \mid s := L real conv (re z)
 + \text{ " } i + L real conv (im z))) \$) ;
(b := y[j] \mid s := (b \mid \text{ " } t \mid \text{ " } w)) ;
(c := y[j] \mid nextpla (file) ; put string (file, c) ; end) ;
(s := y[j] \mid putstring (file, s) ; end) ;
ref int c = char of bfile of file ; int a 1 = c, n = upb s ;
c plus (a 1 = 1 \mid n \mid n + 1) ;
(line ended (file) \mid next pla (file) \mid c := c 1) ;
put string (file, (c 1 \mid s \mid \text{ " } i + s)) ;
end : skip))
end ;
\end{verbatim}
c) \texttt{proc L int string = (L int x, int w, r) string : (r > 1 \land r < 1? | string c := ; L int n := abs x ; L int lr = Kr ; for i to w - 1 do (a \texttt{proc dig char (S (n \ast: lr)) ; n over lr) ; (n = L0 | (x \geq L0 | "+" | "-" + c)) ;

\texttt{d) proc L real string = (L real x, int w, d, e) string : (d \geq 0 \land e > 0 \land d + e + 4 \leq w | L real g = L10 + (w - d - e - 4) ; L real h = g \times L1 ; L real y := abs x ; int p := 0 ; while y \geq g do (y times L1 ; p plus 1) ; (y + L \times L, + d \geq g | y := h ; p plus 1) ; L \texttt{dec string (x \geq 0 | y \land -y), w - e - 2, d) + "10" + int string (p, e + 1, 10) ;

\texttt{e) proc L dec string = (L real x, int w, d) string : (abs x < L10 + (w - d - 2) \land d \geq 0 \land d + 2 \leq w | string s := ; L real y := (abs x + L.5 \times L, + d) \times L1 + (w - d - 2) ; for i to w - 2 do s plus dig char (int a = entier S (ytimes L10) ; y minus K a ; a)) ; (x \geq 0 | "+" | "-" + s[l : w - d - 2] + "." + s[w - d - 1 : ] ;

\texttt{f) proc \% dig char = (int a) char : ("0123456789abcdef" [x + 1]) ;

{In connection with 10.5.2.1.c,d,e, see Table II.}

\texttt{g) proc \% sign supp zero = (ref string a, int l, u) : for i from l + 1 to u while a[i] = "0" do (a[i] := a[i - 1] ; a[i - 1] := ",") ;

\texttt{h) int L int width = (int a := 1 ; while L10 + (a - 1) < L1 \times L max int do a plus 1 ; a) ;

\texttt{i) int L real width = entier S (L ln (L small real) / L ln (L10)) ;

\texttt{j) int L exp width = 1 + entier S
(L ln (L ln (L max real) / L ln (L10)) / L ln (L10)) ;

\texttt{k) proc \% L real conv = (L real x) string :
(string s := L real string (x, L real width + L exp width + 4, L real width - 1, L exp width) ; sign supp zero (s, L real width + 4, L real width + L exp width + 3) ; s) ;

\texttt{l) proc \% nextpic = (file file) : (opened of file | (line ended (file) | new line (file)) ; (page ended (file) | new page (file)) ;}
\textbf{TABLE II:} Display of the values of $L$ \textit{int string}, $L$ \textit{dec string} and $L$ \textit{real string}

\begin{center}
\begin{tabular}{llll}
\hline
\textbf{frame} & \textbf{type} (1 = integer, 2 = real fixed, 3 = real floating, & \\
& 4 = complex fixed, 5 = complex floating, 6 = string, & \\
& 7 = integer choice, 8 = boolean) & \\
\hline
\textbf{radix} (2, 4, 8, 10 or 16) & \\
\textbf{sign} (0 = no sign frame, 1 = sign frame '+', 2 = sign frame '-') & \\
\textbf{number of digits before point}; if type = 1 then $w-7$, else if & \\
\textbf{type} = 2 or 4 then $w-d-2$ else if type = 3 or 5 then $w-d-e-4$, or, & \\
\textbf{if type} = 6, then number of characters in string (0, when variable) & \\
\textbf{number of digits after point}; if type = 2, 3, 4 or 5 then $d$ & \\
\textbf{sign of exponent}; if type = 3 or 5 then as [3] & \\
\textbf{number of digits of exponent}; if type = 3 or 5 then $e$ & \\
[8], ..., [14] as [7], ..., [7] when $\text{frame}[7] = 4$ or 5 & \\
\hline
\end{tabular}
\end{center}

\textbf{TABLE III:} Significance of the elements of \textit{frame}
10.5.2.2. Formatless input

(For formatless input, read and get can be used. The elements of the
given value of the mode specified by [ ] union (intype, proc (file)) are
treated one after the other; if an element is a layout procedure, then it
is called with the file as its parameter; otherwise, it is straightened
(10.5.0.2), and to the resulting names values are assigned, input from
the given file as follows:

aa) If the name refers to a value whose mode is specified by L int, then,
first the file is searched for the first character that is not a space
(giving new lines and pages as necessary); next the largest string is
read from the file that could be indited under control of some
picture or the form n(k2)d or +n(k1)"n(k2)d; this string is converted
to an integer by L string int.

bb) If the name refers to a value whose mode is specified by L real, then,
first the file is searched for the first character that is not a space;
next the largest string is read from the file that could be in­
dited under control of a picture of the form +n(k1)"n(k2)d or n(k2)d
followed by .n(k3)d or s. possible followed by n(k4)"en(k5)" +
n(k6)"n(k7)d or n(k4)"en(k6)"n(k7)d; this string is converted to
a real number by L string real.

cc) If the name refers to a value whose mode is specified by L comp, then,
first a real number is input as in bb and assigned to the real part;
next the file is searched for the first character that is not a space;
next a plus-i-times is expected; finally, a real number is input and
assigned to the imaginary part.

dd) If the name refers to a value whose mode is specified by [] char, then,
if both upper- and lowerstate of the value are one then as many characters
are read as the value has elements; if not both states are one, then
characters are read from the line under control of the terminatorstring
referenced by the file (5.5.1.jj, 10.5.1.mm); the string with those char­
acters as its elements is then the resulting value.

e) If the name refers to a value whose mode is specified by char, then,
first, if the line is full a new line is given, and, if the page is full,
a new page is given; next the character is read from the file.
10.5.2.2. continued

ff) If the name refers to a value whose mode is specified by \texttt{bool}, then, first the file is searched for the first character that is not a space; then a character is read; if this character is flip (flop), then the resulting value is true (false); if the character is neither flip nor flop, then the further elaboration is undefined. }

\begin{verbatim}
a) \texttt{proc read} = ([] union (intype, \texttt{proc (file)}) \texttt{x}) :
  get (\texttt{stand in, x}) ;
  \texttt{proc get} = (file file, [1:] union (intype, \texttt{proc (file)}) \texttt{x}) :
    begin intype it ; \texttt{proc (file)} pf ; \texttt{char k ; priority} \texttt{'} = \texttt{8} ;
    for \texttt{i} to upb \texttt{x} do
      (it \texttt{:=} x[\texttt{i}] ; pf \texttt{:=} x[\texttt{i}] | pf (file) | [1:] \texttt{ref \ simpout} y = straightin it ;
      op \texttt{?} = (string \texttt{a}) \texttt{bool} :
        (outside (file) | \texttt{false} | get string (file, k) ;
        \texttt{char} in string (k, loc int, e) | \texttt{true} | backspace (file) ; \texttt{false}) ;
      op \texttt{!} = (string \texttt{a}, \texttt{char} \texttt{o}) \texttt{char} :
        (get string (file, k) ; \texttt{char} in string (k, loc int, e) | k | \texttt{char} sugg \texttt{:=} \texttt{o} ; (\texttt{char} error \texttt{of} file (sugg) \texttt{| sugg | o))) ;
      \texttt{proc skip spaces} = : (while (nextpos (file) ; ? ")") \texttt{do skip} ;
      \texttt{proc read dig} = \texttt{string} :
        (string \texttt{t} := ; while ? "0123456789" do \texttt{t} plus \texttt{k} ; \texttt{t}) ;
    \texttt{proc read num} = \texttt{string} :
      (char \texttt{t} := (?"+-" \texttt{| k} \texttt{| ")") ; while ? "-" \texttt{do skip} ;
       ".10123456789":"0" \texttt{+ read dig}) ;
    \texttt{proc read real} = \texttt{string} :
      (string \texttt{t} := (skip spaces ; read num) ; (? "." \texttt{| t} \texttt{plus ".") + read dig ; (? "10" \texttt{| t} \texttt{plus ",10" + read num}) ; \texttt{t})
      for \texttt{j} to upb \texttt{y} do
        (ref \texttt{bool} bb ; ref \texttt{char} cc ; ref \texttt{string} ss ;
         (* \texttt{ref \ int} it ; (it \texttt{:=} y[j] |
           \texttt{val} \texttt{it} := \texttt{L string int (read num, 10)}) \texttt{*)) ;
        (* \texttt{ref \ real} xx ; (xx \texttt{:=} y[j]
           \texttt{val} \texttt{xx} := \texttt{L string real (read real)}) \texttt{*)} ;
\end{verbatim}


10.5.2.2. continued 2

\[(ref L compL as ; (as := y[j] | get (file, re of as)) ;
    skip spaces ; "]|.|" ; get (file, im of as)))] ;

(bb := y[j] | skip spaces ; val bb :=
    (? "\" | true | "\" | "\" = "\") ;

(cc := y[j] | nextplo (file) ; get string (file, cc)) ;

(ss := y[j] | lus ss \ ves ss | get string (file, ss) |
    string t := ; while (line ended (file) | false) :
    ? term of file | backspace (file) ; false | true) do t plus k ;
    val ss := t : at (lws ss | lws ss | lws t - upb t + upb ss)] ) ;
end : skip) end ;

\)  

\(\text{c) proa L string int = ( [1 : } \right) \text{ char x, int r) L int :}
\)
\(\text{at } (r > 1 \land r < 17 \land L int n := L0 ; L int lr = kr ; \text{int w = upb x ;}
\)
\(\text{for } i \text{ from 2 to } w \text{ do } n := n \times lr + k \text{ (int } d = \text{ char dig (x[i]) ;}
\)
\((d < r \land d)) ; (x[1] = "+" | n | : x[1] = "-" | -n)) ;

\(\text{\)  

\(\text{d) proa L string real = (string x) L real :}
\)
\(\text{at e ; (char in string ("10", e, x) |}
\)
\(\text{L string dec (x[1 : e - 1]) \times L10 \times string int (x[e + 1:], 10) |}
\)
\(\text{L string dec (x))} ;

\(\text{\)  

\(\text{e) proa L string dec = ([1 : } \right) \text{ char x, L real : (int w = upb x ;}
\)
\(\text{L real r := L0 ; int p ; (char in string ("", p, x) |}
\)
\(\text{[1 : w - 2] char s = x[2 : p - 1] + x[p + 1:] ;}
\)
\(\text{for } i \text{ to } w - 2 \text{ do } r := L w \times r +
\)
\(\text{K (int } d = \text{ char dig (s[i]) ; (d < 10 \land d)) ;}
\)
\((x[1] = "+" | r | : x[1] = "-" | -r) \times L.1 + (w - p) |}
\(\text{L string dec (x + ".")}) ;

\(\text{\)  

\(\text{f) proa % char dig = (char x) int :}
\)
\(\text{(int i ; (char in string (x, i, "0123456789abedef") | i - 1)) ;
\)

\(\text{\)  

\(\text{g) proa % outside = (file f) bool : line ended (f) \lor page ended (f) \lor}
\)
\(\text{file ended (f) ;
\)

10.5.3. Formatted transfut

\(\text{(For the significance of formats see format-denotations (5.5).)}\)

\(\text{\)  

\(\text{a) proa format = (file file, tamrof tamrof) :}
\)
\(\text{(forp of file := 1 ; format of file := collection list pack}
\)
\((\"(" + F F of tamrof + ")", Loc int := 1)) ;\)
10.5.3. continued

b) \texttt{proc \% collection list pack = (string s, ref int p) string :}
   \begin{verbatim}
   (string t := collection (s, p);
    while s[p] = "," do t plus "," + collection (s, p);
    p plus 1; t);
   \end{verbatim}

c) \texttt{proc \% collection = (string s, ref int p) string :}
   \begin{verbatim}
   (int n, q; string f := (p plus 1; insertion (s, p));
    q := p; replicator (s, p, n);
    (s[p] = "(" | string t = collection list packs (s, p);
    to n do f plus t | p := q; f plus picture (s, p, loc[1 : 14] int));
    f + insertion (s, p));
   \end{verbatim}

d) \texttt{proc \% insertion = (strings, ref int p) string :}
   \begin{verbatim}
   (int q = p; skip insertion (s, p); s[q : q + p - 1]);
   \end{verbatim}

e) \texttt{proc \% skip insertion = (string s, ref int p) :}
   \begin{verbatim}
   while (p > upb s | false | skip align (s, p) | true |
   skip lit (s, p)) do skip ;
   \end{verbatim}

f) \texttt{proc \% skip align = (string s, ref int p) bool :}
   \begin{verbatim}
   (int q = p; replicator (s, p, loc int);
    (char in string (s[p], loc int, "x y p l k") |
    p plus 1; true | p := q; false));
   \end{verbatim}

g) \texttt{proc \% replicator = (string s, ref int p, n) :}
   \begin{verbatim}
   (string t :=; while char in string
    (s[p], loc int, "0123456789") do (t plus s[p]; p plus 1); n := (t = "" | 1 | string int ("+" + t, 10)));
   \end{verbatim}

h) \texttt{proc \% skip lit = (string s, ref int p) bool :}
   \begin{verbatim}
   (int q = p; replicator (s, p, loc int);
    (s[p] = "" | while (s[p plus 1] = "" | s[p plus 1] = "" |
    true) do skip ; true | p := q; false));
   \end{verbatim}

i) \texttt{proc \% picture = (string format, ref int p,}
   \texttt{ ref[\] int frame) string :}
   \begin{verbatim}
   begin int n; int po = p;
    op ? = (string s) bool :
     (skip insertion (format, p); p > upb format | false |
      int q = p; replicator (format, p, n);
      (format[p] = "e" | p plus 1);
      (char in string (format [p], loc int, s) |
      p plus 1; true | p := q; false));
   \end{verbatim}
10.5.3 continued 2

\texttt{proc intreal pattern = (ref[1 : 7] int frame) bool :}

\begin{verbatim}
  (num mould (frame[2 : 4]) | frame[1] := 1 ; l) ;
  (? "." | num mould (frame[3 : 5]) | frame[1] := 2 ; l) ;
  (? "e" | num mould (frame[5 : 7]) | frame[1] := 3 ; l) ;
  false. l : true) ;
\end{verbatim}

\texttt{proc num mould = (ref[1 : 3] int frame) bool :}

\begin{verbatim}
   while ? "as" do frame[3] plus n ;
   format[p] = ",," v format[p] = "i" v format[p] = ")") ;
\end{verbatim}

\texttt{proc string mould = (ref[] int frame) bool : (? "t" | true |}

\begin{verbatim}
  while ? "a" do frame[4] plus n ; format[p] = ",," v
  format[p] = ")") ;
\end{verbatim}

\begin{verbatim}
  for i to 14 do frame[i] := 0 ; frame[8] := 10 ;
  (intreal pattern (frame[1 : 7]) | (? "i" |
   frame[1] plus 2 ; intreal pattern (frame[8 : 14])) ; end) ;
  (string mould (frame) | frame[1] := 6 ; end) ;
  frame[1] := 0 ; end) ;
  (format[p] = "(" |
   while ? "(," do skip lit (format, p) ; p plus 1) ;
  end; skip insertion (format, p) ; format [po : p - 1]
\end{verbatim}

\texttt{end ;}

{In connection with 10.5.3.i. see Table III.}

10.5.3.1 Formatted output

\texttt{a) proc outf = (file file, tamrof tamrof,[] outtype w) :}

\begin{verbatim}
  (format (file, tamrof) ; out (file, w)) ;
\end{verbatim}

\texttt{b) proc out = (file file, [1 :] outtype w) :}

\begin{verbatim}
  begin string format = format of file; ref int p = forp of file ;
  for k to upb w do
  ([1 :] simplout y = straightout w[k] ; int q, j := 0 ;
   [1 : 14]int frame ;
\end{verbatim}

\begin{verbatim}
  \texttt{end ; skip insertion (format, p) ; format [po : p - 1]
\end{verbatim}
10.5.3.1. continued

rep : j plus 1; step :
while (do insertion (file, format, p) ; p > upb format | falsé | format[p] = ",") do p plus 1; (j > upb y | end) ; (p > upb format | (format end of file | p := 1) ; step) ;
q := p ; picture (format, q, frame) ;
(frame[1] | int, real, real, compl, compl, string, intch, bool) ;

int: (* (L int i ; (i ::= y[j] | edit L int (file, i, format, p, frame) ; rep)) *) ; incomp ;
real: (* (L real x ; (x ::= y[j] | edit L real (file, x, format, p, frame) ; rep)) *) ;
(int L int i ; (i ::= y[j] | edit L real (file, i, format, p, frame) ; rep)) *) ; incomp ;
compl: (* (L compl z ; (z ::= y[j] | edit L compl (file, z, format, p, frame) ; rep)) *) ;
(* (L real x ; (x ::= y[j] | edit L compl (file, x, format, p, frame) ; rep)) *) ;
(* (L int i ; (i ::= y[j] | edit L compl (file, i, format, p, frame) ; rep)) *) ; incomp ;
string: ([L : frame[4]] char s ; char a ;
(s ::= y[j] | frame[4] = 0 | put (file, s)) |
edit string (file, s, format, p, frame) ; rep) ;
(a ::= y[j] |
edit string (file, a, format, p, frame) ; rep)) ; incomp ;
intch: (int i ; (i ::= y[j] |
edit choice (file, i, format, p) ; rep)) ; incomp ;
bool: (bool b ; b ::= y[j] |
edit bool (file, b, format, p) ; rep)) ;
incomp: (value error of file | rep | put (file, y[j]) ; undefined) ;
end : skip)
end ;
c) proc % edit L int = (file f, L int i, string format, ref int p, [] int fr) :
edit string (f, L int string (i, fr[4] + 1, fr[2]), format, p, fr) ;
d) proc % edit L real = (file f, L real x, string format, ref int p, [] int fr) :
edit string (f, stringed L real (x, fr), format, p, fr) ;
e) proc % stringed real = (L real x, [] int fr) string :

f) proc % edit L compl = (file f, L compl z, [] int fr) :
    edit string (f, ([1 : 14] int g := fr ; g [1] minus 2 ;
    stringed L real (re z, g[1 : 7]) + "|" + stringed L real
    (im z, g[8 : 14])), format, p, fr) ;

g) proc % edit string = (file f, string x, format,
    ref int p, [] int frame) :
    begin int p1 := 1, n ; bool supp ; string s := x ;
    op ? = (string s) bool:
        (do insertion (file, format, p) ; p > upb format |
        false | int q = p ; replicator (format, p, n) ;
        (supp := format[p] = "s" | p plus 1) ;
        (char in string (format[p], loc int, s) |
        p plus 1 ; true | p := q ; false)) ;
    proc copy = : ((supp | put string (f, s[p1])) ; p1 plus 1) ;
    proc intreal mould = :
        (? "r" ; sign mould (frame[3]) ; int mould ;
        (? "n" | copy ; int mould |: s[p1] = "." | p1 plus 1) ;
        (? "e" | copy ; sign mould (frame[6]) ; int mould)) ;
    proc sign mould = (int sign) : (sign = 0 | p1 plus 1 |
    s[p1] := (s[p1] = "+" | (sign | "+", ",") | ",") ;
    (? "z" | sign supp zero (s, p1, p1 + n) | n := 0) ;
    to n + 1 do copy ; p plus 1) ;
    proc int mould = :
        (? ? "a" | bool as := true ; to n do
        (s[p1] = "0" ^ as | put string (file, "_")) ;
        p1 plus 1 | as := false ; copy) ; l) ;
        (? "d" | to n do copy ; l)) ;
    proc string mould = :
        while ? "a" do to n do copy ;
    tes: (frame[7] = 6 | string mould |: intreal mould ;
    frame[7] > 3 | p plus 1 ; copy ; intreal mould)
    end ;
10.5.3.1. continued 3

h) proc % edit choice = (file f, int a, string format, ref int p) :
   (a > 0 | do insertion (f, format, p) ; p plus 2 ;
   to a - 1 do (skip lit (format p) ; format[p] = ",";
   p plus 1 | undefined) ;
   do lit. (f, format, p) ;
   while format[p] "," do (p plus 1 ; skip lit (format, p)) ;
   p plus 1 | undefined) ;

i) proc % edit bool = (file f, bool b, string format, ref int p) :
   (do insertion (f, format, p) ; (format[p + 1] = "(" |
   p plus 2 ; (b | do lit (f, format, p) ; p plus 1 ; skip lit
   (format, p) | skip lit (format, p) ; p plus 1 ; do lit (f, format, p) |
   put string (f, (b | "(" | ")")) ; p plus 1) ;

j) proc % do insertion = (file f, string s, ref int p) :
   while (p > upb s | false | do align (f, s, p) | true |
   do lit (f, s, p)) do skip ;

k) proc % do align = (file f, string s, ref int p) bool : 
   (int q = p ; int n ; replicator (a, p, n) ;
   (s[p] = "a" | to n do space (f) ; l |:
   s[p] = "y" | to n do backspace (f) ; l |:
   s[p] = "p" | to n do new page (f) ; l |:
   s[p] = "k" | char of bfile of f := n ; l) ; p := q ; false.
   l : p plus 1 ; true) ;

l) proc % do lit = (file f, string s, ref int p) bool : 
   (int q = p ; int n ; replicator (a, p, n) ; (s[p] = "\n"
   while (s[p] plus 1) = "\n" | s[p plus 1] = "\n" | true) do
   put string (f, s[p]) ; true | p := q ; false)) ;

10.5.3.2. Formatted input

a) proc inf = (file file, tamrof tamrof, [] intype x) :
   (format (file, tamrof) ; in (file, x)) ;

b) proc in = (file file, [1 :] intype x) :
   begin string format = format of file ; ref int p = forp of file ;
   for k to upb x do
   ([1 :) ref simplout y = straightin x[k] ; int q, j := 0 ;
   [1 : 14] int frame ;

}
10.5.3.2. continued

```plaintext
rep : j plus 1 ; step :
while ( \exp\text{insertion} (file, format, p) ; p > upb format |
false | format[p] = "," ) do p plus 1 ; (j = upb y | end) ;
(p > upb format | (format end of file | p := 1) ; step) ;
q := p ; picture (format, q, frame) ;
(frame[1] | int, real, real, compl, compl, string, intah, bool) ;
int: ( \{ (ref L int ii ; (ii := y[j] | 
indit L int (file, ii, format, p, frame) ; rep)) \} ; incomp ;
real: ( \{ (ref L real xx ; (xx := y[j] | 
indit L real (file, xx, format, p, frame) ; rep)) \} ; incomp ;
compl: ( \{ (ref L compl zz ; (zz := y[j] | 
indit L compl (file, zz, format, p, frame) ; rep)) \} ; incomp ;
string: (ref string ss ; ref char aa ; [1 : frame[4]] char t ;
(frame[4] = 0 | : ss := y[j] | get (file, ss) ; rep | incomp ;
indit string (file, t, format, p, frame) ;
(ss := y[j] | val ss := t ; rep) ; aa := y[j] | 
val aa := t[1] ; rep)) ; incomp ;
intah: (ref int ii ; (ii := y[j] | 
indit choice (file, ii, format, p) ; rep)) ; incomp ;
bool: (ref bool bb ; (bb := y[j] | 
indit bool (file, bb, format, p) ; rep)) ;
incomp: (value error of file | rep | undefined) ;
end : skip )
end ;
c) \text{proc} % \text{indit L int} =
(file f, ref L int i, string format, ref int p, [] int fr) :
(string t ; indit string (f, t, format, p, fr) ;
i := L string int (t, fr[2]));
d) \text{proc} % \text{indit L real} =
(file f, ref L real x, string format, ref int p, [] int fr) :
(string t ; indit string (f, t, format, p, fr) ;
x := L string real (t));
e) \text{proc} % \text{indit L compl} =
(file f, ref L compl a, string format, ref int p, [] int fr) :
(string t ; int i ; indit string (f, t, format, p, fr) ;
a := (char in string ("\", i, t) | 
(L string real (t[1 : i - 1]) | L string real (t[i + 1 : ])));
```
f) proc % init string =
   (file f, ref string t, string format, ref int p, [] int frame) :
   begin int n ; bool supp ; char k ; string x := ;
   op ? = (string s) bool :
      (exp insertion (format, p) ; p > upb format \ false |
      int q = p ; replicator (format, p, n) ;
      (supp := format[p] = "s" | p plus 1) ;
      (char in string (format[p], loc int, s) |
      p plus 1 ; true | p := q ; false)) ;
   priority ! = 8 ;
   op ! = (string s, char a) string :
      (char in string (next, loc int, s) | (supp | "" | k) |
      char supp := a ; (char error (supp) | supp | a)) ;
   proc next = char : (get string (f, k) ; k) ;
   proc intreal mould = :
      (? "n" ; sign mould (frame[3]) ; int mould ;
      (? "." | x plus "." ! "." ; int mould ;
      (? "a" | x plus "10" ! "10" ! sign mould (frame[6]) ;
      int mould)) ;
   proc sign mould = (int sign) : (sign = 0 | x plus "+" |
   int j ; (? ? "z" | n := 0) ; for i to n + 1
   while next = "," do j := i ;
   x plus (sign = 1 | " +" ! "+" | : k = "-" | k |:
   backspace (f) ; j > 0 | j minus 1 ; "+" ! "+" ! "10") ;
   for i from j + 1 to n + 1 do x plus "0123456789" ! "0")
   proc int mould = : (f :
      (? "a" | int j ; for i to n while next = "," do j := i ;
   backspace (f) ;
   from j to n do x plus "0123456789" ! "0" ; l) ;
   (? "d" | to n do x plus "0123456789" ! "0" ; l)) ;
   proc string mould = while? "a" do to n do x plus
   (supp | "," | next) ;
   "|" ! "|" ; intreal mould) ;
   t := x ;
end ;
10.5.3.2. continued 3

g) \textit{proc} \texttt{\%} \textit{indit choice} = 
\begin{verbatim}
(file f, ref int c, string format, ref int p) :
(exp insertion (f, format, p) ; p plus 2 ; c := 1 ;
while ask lit (f, format, p) do
(c plus 1 ; format[p] = ";" | p plus 1 | undefined) ;
while format[p] \neq ")" do (p plus 1 ; skip lit (format, p)) ;
p plus 1 ; exp insertion (f, format, p)) ;
\end{verbatim}

h) \textit{proc} \texttt{\%} \textit{indit bool} = 
\begin{verbatim}
(file f, ref bool b, string format, ref int p) :
(exp insertion (f, format, p) ; (format[p + 1] = "(" | p plus 2 ;
(b := ask lit (f, format, p) | p plus 1 ; skip lit (format, p) |:
p plus 1 ; ask lit (f, format, p) | undefined) |
char k ; get string (f, k) ; b := (k = "\" | true |
k = "\" | false)) ;
p plus 1 ; exp insertion (f, format, p)) ;
\end{verbatim}

i) \textit{proc} \texttt{\%} \textit{exp insertion} = (file f, string s, ref int p) :
\begin{verbatim}
while (p > upb s | false) : do align (f, s, p) | true |
exp lit (f, s, p)) do skip ;
\end{verbatim}

j) \textit{proc} \texttt{\%} \textit{exp lit} = (file f, string s, ref int p) bool :
\begin{verbatim}
(int q = p ; int n ; replicator (s, p, n) ;
(s[p] = """" | int r = p ; to n do (p := r ;
while (s[p plus 1] = """" | s[p plus 1] = """" | true) do
(char k ; get string (f, k) ; b := (k \neq s[p] | undefined)) ; true |
p := q ; false)) ;
\end{verbatim}

k) \textit{proc} \texttt{\%} \textit{ask lit} = (file f, string s, ref int p) bool :
\begin{verbatim}
(int c = char of f ; int n ; replicator (s, p, n) ;
(s[p] = """" | int r = p ; to n do (p := r ;
while (s[p plus 1] = """" | s[p plus 1] = """" | true) do
(char k ; get string (f, k) ; k \neq s[p] | b)) ; true.
L : while (s[p plus 1] = """" | s[p plus 1] = """" | true) do skip ;
char of f := c ; false)) ;
\end{verbatim}
10.5.4. Binary transput

a) \texttt{proc} \% to bin = (file f, simplout x) [] int:
\hspace{1cm} a value of mode 'row-of-integral' whose lower bound is one,
\hspace{1cm} and whose upper bound depends on the value of 'f' and on the
\hspace{1cm} mode of the value of 'x'; furthermore,
\hspace{1cm} x = from bin (f, x, to bin (f, x)) a ;

b) \texttt{proc} \% from bin = (file f, simplout v, [] int y) simplout:
\hspace{1cm} a value, if one exists, of the mode of the actual parameter
\hspace{1cm} corresponding to v, such that
\hspace{1cm} y = to bin (f, from bin (f, v, y)) a ;

(On some channels a more straightforward way of transput is available. Some properties of this binary transput depend on the particular implementation, others can be deduced from 10.5.4.}

10.5.4.1. Binary output

a) \texttt{proc} write bin = ([] outtype x) : put bin (stand back, x) ;

b) \texttt{proc} put bin = (file file, [1 :] outtype x) :
\hspace{1cm} if bin possible[chan of file] \wedge opened of file
\hspace{1cm} then if set possible[chan of file] then state def of file
\hspace{1cm} else state def of file := state bin of file := true ;
\hspace{1cm} state get of file := false
\hspace{1cm} fi ;
\hspace{1cm} for k to upb x do
\hspace{1cm} ([1 :) simplout y = straightout x[k]) ;
\hspace{1cm} for j to upb y do
\hspace{1cm} ([1 :) int bin = to bin (file, y(j)); bfile b = bfile of file;
\hspace{1cm} ref int p = page of b, l = line of b, c = char of b;
\hspace{1cm} for i to upb bin do (next p = (file) ;
\hspace{1cm} book of b[p, l, c] := bin[i] ; a plus 1 ;
\hspace{1cm} (p = lpage of b \wedge l = llane of b | (c > lchar of b \wedge lchar of b := c))
\hspace{1cm} else undefined
\hspace{1cm} fi ;
10.5.4.2. Binary input

a) \texttt{proc read bin = ([] intype x) : get bin (stand back, x) ;}

b) \texttt{proc get bin = (file file, [1 :] intype x) :}

\texttt{if bin possible[chan of file] \& opened of file}
\texttt{then if - set possible[chan of file] thef state def of file}
\texttt{then (~state get of file \textasciitilde state bin of file | undefined)}
\texttt{else state def of file := state bin of file :=}
\texttt{state get of file := true}

\texttt{fi ;}

\texttt{for k to upb x do}
\texttt{([1 :) ref simplout y = straightin x[k] ;}
\texttt{for j to upb y do}
\texttt{([1 :) int bin := to bin (file, y[j]); bfile b = bfile of file ;}
\texttt{for i to upb bin do (next pic (file) ; check pic (file) ;
bin[i] := bool of b[page of b, line of b, char of b] ;
char of b plus 1) ;}
\texttt{(k (ref \& int ii ; (ii := y[j] | }
\texttt{val ii := from bin (file, ii, bin))) \}) ;}
\texttt{(k (ref \& real xx ; (xx := y[j] | }
\texttt{val xx := from bin (file, xx, bin))) \}) ;}
\texttt{(k (ref \& compl ss ; (ss := y[j] | }
\texttt{val ss := from bin (file, ss, bin))) \}) ;}
\texttt{(ref string ss ; (ss := y[j] |}
\texttt{val ss := from bin (file, ss, bin))) ;}
\texttt{(ref char cc ; (cc := y[j] |}
\texttt{val cc := from bin (file, cc, bin))) ;}
\texttt{(ref bool bb ; (bb := y[j] |}
\texttt{val bb := from bin (file, bb, bin))) })
\texttt{else undefined}
\texttt{fi ;}

\{But Eeyore wasn't listening. He was
taking the balloon out, and putting it
back again, as happy as could be. \ldots

Winnie-the-Pooh, \quad A.A. Milne. \}
11. Examples

11.1. Complex square root

A declaration in which \texttt{compsqrt} is a procedure-with-[complex]-parameter-[complex]-mode-identifier (here [complex] stands for structured-with-real-field-letter-r-letter-e-and-real-field-letter-i-letter-m.):

\begin{itemize}
  \item[a)] \texttt{proc compsqrt} = (\texttt{compl} \ z) \ \texttt{compl} : a the square root whose real part is nonnegative of the complex number \ z \ a
  \item[b)] \texttt{begin real} \ x = \texttt{re} \ z, \ y = \texttt{im} \ z ;
  \item[c)] \texttt{real rp} = \texttt{sqrt} \ ((\texttt{abs} \ x + \sqrt{x^2 + y^2}) / 2) ;
  \item[d)] \texttt{real ip} = (\texttt{rp} = 0 \ | 0 \ | y / (2 \times \texttt{rp})) ;
  \item[e)] (x \geq 0 \ | \texttt{rp} \ \texttt{ip} \ \texttt{abs} \ \texttt{ip} \ | (y \geq 0 \ | \texttt{rp} \ | -\texttt{rp}))
  \item[f)] \texttt{end}
\end{itemize}

[complex]-calls \{8.6.2\} using \texttt{compsqrt}:

\begin{itemize}
  \item[g)] \texttt{compsqrt} (w)
  \item[h)] \texttt{compsqrt} (-3.14)
  \item[i)] \texttt{compsqrt} (-1)
\end{itemize}
11.2. Innerproduct1

A declaration in which innerproduct1 is a procedure-with-integral-parameter-and-procedure-with-integral-parameter-real-parameter-and-procedure-with-integral-parameter-real-parameter-real-mode-identifier:

a) \texttt{proc innerproduct1 = (int n, proc (int) real x, y) real :} \\
\hspace{1em} \texttt{comment the innerproduct of two vectors, each with n components, } \\
\hspace{1em} \texttt{x(i), y(i), i = 1, \ldots, n, where x and y are arbitrary mappings } \\
\hspace{1em} \texttt{from integer to real number comment} \\
b) \texttt{begin long real s := long 0 ;} \\
c) \texttt{for i to n do s plus long x(i) \times y(i) ;} \\
d) \texttt{short s} \\
e) \texttt{end} \\

Real-calls (8.6.2) using innerproduct1:

f) innerproduct1 (m, (int j) real : x[j], (int j) real : y[j]) \\
g) innerproduct1 (n, nstn, nnoe) \\

11.3. Innerproduct2

A declaration in which innerproduct2 is a procedure-with-reference-to-row-of-real-parameter-and-reference-to-row-of-real-parameter-real-mode-identifier:

a) \texttt{proc innerproduct2 = (ref[1 :] real a ; ref[1 : upb a] real b) real) :} \\
\hspace{1em} \texttt{a the innerproduct of two vectors a and b with equal number of } \\
\hspace{1em} \texttt{elements \_} \\
b) \texttt{begin long real s := long 0 ;} \\
c) \texttt{for i to upb a do s plus long a[i] \times b[i] ;} \\
d) \texttt{short s} \\
e) \texttt{end} \\

Real-calls using innerproduct2:

f) innerproduct2 (x1, y1) \\
g) innerproduct2 (y2[2], y2[., o])
11.4. Innerproduct3

A declaration in which innerproduct3 is a procedure-with-reference-to-integral-parameter-and-integral-parameter-and-procedure-real-parameter-and-procedure-real-parameter-real-mode-identifier:

a) \texttt{proc innerproduct3 = (ref int i, int n, proc real xi, yi) real :}
   \texttt{comment the innerproduct of two vectors whose n elements are the}
   \texttt{values of the expressions xi and yi and which depend, in general,}
   \texttt{on the value of i comment}

b) \texttt{begin long real s := long 0 ;}

c) \texttt{for k to n do (i := k ; s plus long xi \times long yi) ;}

d) \texttt{short s}

e) \texttt{end}

A real-call using innerproduct3:

f) \texttt{innerproduct3 (j, 8, x1[j], y1[j + 1])}

11.5. Largest element


a) \texttt{proc absmax = (ref[1 : : 1] real a,}

b) \texttt{a result a ref real y, a subscripts a ref int i, k) :}
   \texttt{comment the absolute value of the element of greatest absolute value}
   \texttt{of the matrix a is assigned to y, and the subscripts of this element}
   \texttt{to i and k comment}

c) \texttt{begin y := -1 ;}

d) \texttt{for p to 1 upb a do for q to 2 upb a do}

e) \texttt{if abs a[p, q] > y then y := abs a[i := p, k := q] fi}

f) \texttt{end}

Void-calls \{8.6.2\} using absmax:

g) \texttt{absmax (x2, x, i, j)}

h) \texttt{absmax (x2, x, loc int, loc int)}
11.6. Euler summation

a) **proc euler** = (proc (int) real f, real eps, int tim) real:
   comment the sum for i from 1 to infinity of f(i), computed by means of a suitably refined Euler transformation. The summation is terminated when the absolute values of the terms of the transformed series are found to be less than eps tim times in succession. This transformation is particularly efficient in the case of a slowly convergent or divergent alternating series comment

b) begin int n := 1, t; real mn, ds := eps; [1 : 16] real m;
c) real sum := (m[1] := f(1))/2;
d) for i from 2 while (t := (abs ds < eps | t + 1 | 1)) <= tim do
   begin mn := f(i)
   for k to n do begin mn := ((ds := mn) + m[k])/2;
   m[k] := ds end;
   sum plus (ds := (abs mn < abs m[n] | n < 16 | n plus 1; m[n] := mn; mn/2 | mn))
e) end;
f) end sum;
g) end

A call using euler:
m) euler ((int i) real : (odd i | -1 < i | 1 / i), 10-8, 2)

11.7. The norm of a vector

a) **proc norm** = (ref[1 :] real a) real :
   the euclidean norm of the vector a a
b) (long real s := long 0;
c) for k to upb a do s plus long a[k] + 2;
d) short long sqrt(s))

For a use of norm as a call, see 11.8.a.
11.8. Determinant of a matrix

a) \( \text{proc det} = (\text{ref[1 : 1]} \, \text{real} \, a, \text{ref[1 : upb a]} \, \text{int} \, p) \, \text{real} : \)

b) \( \text{if upb} \, a = 2 \, \text{upb} \, a \)

c) \( \text{then int} \, n = \text{upb} \, a \);

comment the determinant of the square matrix \( a \) of order \( n \) by the method of Crout with row interchanges: \( a \) is replaced by its triangular decomposition \( l \times u \) with all \( u[k, k] = 1 \). The vector \( p \) gives as output the pivotal row indices; the \( k \)-th pivot is chosen in the \( k \)-th column of \( l \) such that \( \text{abs} \, l[i, k] / \text{row norm} \) is maximal comment

d) \( [1 : n] \, \text{real} \, v ; \, \text{real} \, d := 1, r := -1, s, \text{pivot} ; \)
e) \( \text{for} \, i \, \text{to} \, n \, \text{do} \, v[i] := \text{norm} \, (a[i]) ; \)
f) \( \text{for} \, k \, \text{to} \, n \, \text{do} \)
g) \( \text{begin int} \, k1 = k - 1 ; \, \text{ref int} \, pk = p[k] ; \)
h) \( \text{ref[1 :]} \, \text{real} \, a1 = a[1 : 1] , \, au = a[1 : k1] ; \)
i) \( \text{ref[1 :]} \, \text{real} \, ak = a[k] , \, ka = a[1 , k] , \, apk = a[pk] , \)
j) \( \text{alk} = a[k] , \, ka1 = au[1 , k] ; \)
k) \( \text{for} \, i \, \text{from} \, k \, \text{to} \, n \, \text{do} \)
\( \text{begin} \, \text{ref real} \, aik = ka[i] ; \)
l) \( \text{if} \, (s := \text{abs} \, (aik \, \text{minus innerproduct2} \, (a[i] , \, ka1)) / \, v[i]) > r \)
\( \text{then} \, r := s ; \, pk := i \, \text{fi} \)
\( \text{end} ; \)
m) \( v[pk] := v[k] ; \, \text{pivot} := ka[pk] ; \)
\( \text{for} \, j \, \text{to} \, n \, \text{do} \)
\( \text{begin} \, \text{ref real} \, akj = a[k , j] , \, apkj = apk[j] ; \)
\( \text{r} := akj ; \, akj := \text{if} \, j < k \, \text{then} \, \text{apkj} \)
\( \text{else} \, (\text{apkj} - \, \text{innerproduct2} \, (alk , \, au[j]) \) / \, \text{pivot} \, \text{fi} ; \)
\( \text{if} \, pk \neq k \, \text{then} \, \text{apkj} := -r \, \text{fi} \)
\( \text{end} ; \)
\( d \, \text{times} \, \text{pivot} \)
\( \text{end} ; \)
\( y \)
\( d \)
\( z \)
\( \text{fi} \)

A call using \( \text{det} \):

aa) \( \text{det} \, (y2 , \, i1) \)
11.9. Greatest common divisor

An example of a recursive procedure:

a) \( \text{proc god} = (\text{int} \ a, b) \ \text{int} : \)
   \( \text{a the greatest common divisor of two integers a} \)

b) \( (b = 0 \ \text{abs} \ a \ \text{god} (b, a +: b)) \)

A call using \( \text{god} \):

c) \( \text{god} (n, 124) \)

11.10. Continued fraction

An example of a recursive operation:

a) \( \text{op} \ / = ([1 :] \ \text{real} \ a ; [1 : \text{upb} \ a] \ \text{real} \ b) \ \text{real} : \)
   \( \text{comment the value of a/b is that of the continued fraction} \)
   \( a_1 / (b_1 + a_2 / (b_2 + \ldots a_n / b_n) \ldots) \text{ comment} \)

b) \( (\text{upb} \ a = 1 \ | \ a[1] / b[1] \ | \ a[1] / (b[1] + a[2 :] / b[2 : 1])) \)

A formula using \( \text{/} \):

c) \( x_1 / y_1 \)

(The use of recursion may often be elegant rather than efficient as in 11.9 and 11.10. See, however, 11.11 and 11.14 for examples in which recursion is of the essence.)
11.11. Formula manipulation

a) begin union form = (ref const, ref var, ref triple, ref call);
b) struct const = (real value);
c) struct var = (string name, real value);
d) struct triple = (form left operand, int operator, form right operand);
e) struct function = (ref var bound var, form body);
f) struct call = (ref function function name, form parameter);
g) int plus = 1, minus = 2, times = 3, by = 4, to = 5;
h) const zero, one; value of zero := 0; value of one = 1;
i) op = (form a, ref const b) bool:
   (ref const ea; (ec := a | val ea :=: b | false));
j) op + = (form a, b) form:
   (a = zero | b | b = zero | a | triple := (a, plus, b));
k) op - = (form a, b) form:
   (b = zero | a | triple := (a, minus, b));
l) op * = (form a, b) form:
   (a = zero \ b = zero | zero | a = one | b | b = one | a | triple := (a, times, b));
m) op / = (form a, b) form:
   (a = zero \ b = zero | zero | b = one | a | triple := (a, by, b));
n) op \ = (form a, ref const b) form:
   (a = one \ (b :=: zero) | one | b :=: one | a | triple := (a, to, b));
o) proc derivative of = (form e, a with respect to a ref var a) form;
p) begin ref const ec; ref var ev; ref triple et; ref call ef;
q) case ec, ev, et, ef :=: e in
r) zero comma
s) (val ev :=: a | one | zero) comma
t) (form u = left operand of et, v = right operand of et,
u) udash = derivative of (u, a with respect to a x),
v) vdash = derivative of (v, a with respect to a x);
w) case operator of et in
x) udash + vdash, udash - vdash,
y) u \ vdash \ udash \ v, (udash - et \ vdash) / v,
z) (ec :=: v | v \ u \ (const c;
ua) value of c := value of ec - 1; c) \ udash) ecall comma
11.11. continued

ab) \[ \text{ref function } f = \text{function name of } ef ; \]
ac) \[ \text{form } g = \text{parameter of } ef ; \]
ad) \[ \text{ref var } y = \text{bound var of } f ; \]
ae) \[ \text{function } f\text{dash} := (y, \text{derivative of (body of } f, y)) ; \]
af) \[ (\text{call} := (f\text{dash}, g)) \times \text{derivative of } (g, x) \]
ag) \[ \text{esac} \]
ah) \[ \text{end a derivative } a ; \]
ai) \[ \text{proc value of } = (\text{form } e) \text{ real} : \]
aj) \[ \text{begin ref const } ea ; \text{ref var } ev ; \text{ref triple } et ; \text{ref call } ef ; \]
ak) \[ \text{case } ea, ev, et, ef := e \text{ in} \]
al) \[ \text{value of } ea \text{ comma} \]
am) \[ \text{value of } ev \text{ comma} \]
an) \[ (\text{real } u = \text{value of (left operand of } et), \]
ao) \[ v = \text{value of (right operand of } et) ; \]
ap) \[ \text{case operator of } et \text{ in} \]
aq) \[ u + v, u - v, u \times v, u / v, \exp (v \times \ln(u)) \text{ esac} \]
ar) \[ \text{esac} \]
as) \[ \text{ref function } f = \text{function name of } ef ; \]
at) \[ \text{value of bound var of } f := \text{value of (parameter of } ef) ; \]
au) \[ \text{value of (body of } f) \]
sv) \[ \text{esac} \]
sw) \[ \text{end a value of } a ; \]
x) \[ \text{form } f, g ; \text{var } a := ("a", \text{ skip}), b := ("b", \text{ skip}), x := ("x", \text{ skip}) ; \]
xy) \[ \text{start here :} \]
ax) \[ \text{read ((value of } a, \text{ value of } b, \text{ value of } x)) ; \]
ay) \[ \text{f} := a + x / (b + x) ; \text{g} := (f + \text{one}) / (f - \text{one}) ; \]
bz) \[ \text{print ((value of } a, \text{ value of } b, \text{ value of } x, \]
bb) \[ \text{value of (derivative of } (g, a \text{ with respect to } a x))) \]
bc) \text{end}
11.12. Information retrieval

a) **begin a authors and titles enquiry system a**
    
    ```
    struct book = (string title, ref book next),
    auth = (string name, ref auth next, ref book book);
    ```

b) ```ref book book; ref auth auth, first auth := nil, last auth;```

c) ```string name, title; int i; file input, output;```

d) ```open (input, remote in); create (output, remote out);```

e) ```put (output, (reset,```

f) ```"to enter a new author, type "author", a space, and his name", new line,```

g) ```"to enter a new book, type "book", a space, the name of the author, a new line, and the title", new line,```

h) ```"for a listing of the books by an author, type "list", a space, and his name", new line,```

i) ```"to find the author of a book, type "find", a new line, and the title", new line,```

j) ```"to end, type "end"", new line);```

k) ```proc update = expr if val first auth := nil```

l) ```then auth := first auth := last auth := auth := (name, nil, nil)```

m) ```else auth := first auth; while val auth := nil do```

n) ```(name = name of auth | known | auth := next of auth);```

o) ```last auth := next of last auth := auth :=```

p) ```auth := (name, nil, nil);```

q) ```known: skip fi;```

r) ```client: inf (input, fe ("author", "book", "list", "find", "end", ""), a30al, 80al, f, i);```

s) ```case i in author, publ, list, find, end, error esc"c;```

 t) ```publ: in (input, (name, title)); update;```

u) ```if val book of auth := nil```

v) ```then book of auth := book := title, nil)```

w) ```else book := book of auth; while val next of book := nil do```

x) ```(title = title of book | client | book := next of book);```


z) ```(title, nil))```

``` fi; client;```
11.12. continued

aa) list: in (input, name) ; update ;

ab) outf (output, if"author:"30a1f, name) ;

ac) if val (book := book of auth) := nil

ad) then put (output, "no Publications")

ae) else while val book :# nil do

af) begin if line number (output) = max line [remote out]

ag) then outf (output, f"continued.on.next.page"p

"author:"30a4k"continued"1lf, name)

ah) 

ai) book := next of book

aj) 

ak) fi ; client ;

al) find: in (input, {oo string, title}) ; auth := first auth ;

am) while val auth :# nil do

an) begin book := book of auth ; while val book :# nil do

ao) if title = title of book

ap) then outf(output, f"author:"30a2f

"name of auth) ; client

aq) 

ar) book := next of book

as) fi ; auth := next of auth

at) end ; outf (output, f"unknown") ;

au) client ;

av) end: put (output, (new page, "signed_off");

aw) close (input) ; close (output).

ax) error: put (output, (new line, "mistake, try again.");

ay) end
11.13. Cooperating sequential processes

a) \texttt{begin int mmb magazine slots, mmb producers, mmb consumers ;}
b) \texttt{read ((mmb magazine slots, mmb producers, mmb consumers));}
c) \texttt{[1 : mmb producers]file infile, [1 : mmb consumers]file outfile ;}
d) \texttt{for i to mmb producers do}
e) \texttt{open (infile[i], inchannel[i]) ;}
f) \texttt{the multiple values inchannel and outchannel are}
g) \texttt{defined in a surrounding range ;}
h) \texttt{for i to mmb consumers do}
i) \texttt{open (outfile[i], outchannel[i]) ;}
j) \texttt{mode page = [1 : 60, 1 : 132] char ;}
k) \texttt{[1 : mmb magazine slots] ref page magazine ;}
l) \texttt{int a pointers of a cyclic magazine ;}
m) \texttt{index := 1, exdex := 1,}
\texttt{general semaphores ;}
o) \texttt{full slots := 0, free slots := mmb magazine slots,}
\texttt{binary semaphores ;}
p) \texttt{in buffer busy := 1, out buffer busy := 1 ;}
q) \texttt{proc par call = (proc (int) p, int n) :}
r) \texttt{calls n incarnations of p in parallel ;}
s) \texttt{(n > 1 | par (p (n), par call (p, n - 1)) | p (1)) ;}
t) \texttt{proc producer = (int i) :}
u) \texttt{do (page page ; get (infile[i], page)) ;}
v) \texttt{down free slots ; down in buffer busy ;}
w) \texttt{magazine[index] := page ;}
x) \texttt{index modb mmb magazine slots plus 1 ;}
y) \texttt{up full slots ; up in buffer busy ;}
z) \texttt{proc consumer = (int i) :}
aa) \texttt{do (page page ; down full slots ;}
ab) \texttt{down out buffer busy ; page := magazine[exdex] ;}
ac) \texttt{exdex modb mmb magazine slots plus 1 ;}
d) \texttt{up free slots ; up out buffer busy ;}
e) \texttt{put (outfile[i], page)) ;}
f) \texttt{par (par call (producer, mmb producers),}
g) \texttt{par call (consumer, mmb consumers) ;}
ah) \texttt{end}
11.14. Towers of Hanoi

a) \texttt{begin proc } p = \texttt{(int} me, de, ma) :
\texttt{if ma > 0 then} p (me, 6-me-de, ma - 1);
\texttt{out (stand out, (me, de, ma));}
\texttt{a move from peg 'me' to 'peg 'de' piece number 'ma' a}
p (6-me-de, de, ma - 1);
\texttt{end;}
\texttt{for k to 7 do}
\texttt{(outf (stand out, f"k="str(k,
\texttt{n (max int) (2(4(3(2(1)\times)\times)\times)\times)\times)\times) f, k) ;}
p (1,2,k))
\texttt{end}

K = 1

121
K = 2
131 122 321
K = 3
121 132 231 123 311 322 121
K = 4
131 122 321 133 211 232 131 124 321 312 211 323 131
K = 5
121 132 231 123 311 322 121 134 231 212 311 233 121 132 231 125
311 322 121 313 231 212 311 324 121 132 231 123 311 322 121
K = 6
131 122 321 133 211 232 131 124 321 312 211 323 131
211 232 131 213 321 122 322 131 234 131 122 321 133 211 232 131
131 122 321 133 211 232 131 124 321 312 211 323 131
K = 7
121 132 231 123 311 322 121 134 231 212 311 233 121 132 231 125
311 322 121 313 231 212 311 324 121 132 231 123 311 322 121
321 212 311 233 121 132 231 214 311 322 121 313 231 212 311
121 132 231 123 311 322 121 134 231 212 311 233 121 132 231 127
311 322 121 313 231 212 311 324 121 132 231 123 311 322 121
321 212 311 233 121 132 231 214 311 322 121 313 231 212 311
231 212 311 233 121 132 231 214 311 322 121 313 231 212 311
121 132 231 123 311 322 121 134 231 212 311 233 121 132 231 125
311 322 121 313 231 212 311 324 121 132 231 123 311 322 121

12. Glossary

Given below are the locations of the first, and sometimes other, instructive appearances of a number of words which, in Chapters 1 up to 10 of this Report, have a specific technical meaning. A word appearing in different grammatical forms (e.g. "contain", "contains", "contained", "containing") is given once, usually as infinitive (e.g. "contain").

action 2.2, 2.2.5
ALGOL 68 program 4.4
apostrophe 1.1.6.c
applied occurrence 4.1.2.a
appoint 6.0.2.a
a priori value 5.1.0.2.b
arithmetic value 2.2.3.1.a
assign 2.2.2.1, 8.3.1.2.c
asterisk 1.1.2.a
automaton 1.1.1.a
backfile 10.5.1.aa, cc
balance 6
blind alley 1.1.2.d
case clause 9.4.b, c, d, e
channel 10.5.1.aa, bb
character 2.2.3.1.a, f
close a file 10.5.1.ii
coercend 8
collateral 2.2.5, 6.2.2.a
colon 1.1.2.a
comma 1.1.2.a
compatible 5.5.1.dd, nn
compile 2.3.c
component of 2.2.2.h
composite 3.1.2.d
computer 1.1.1.a
constant 5
constituent 1.1.6.e
contain 1.1.6.b
conversion-key 5.5.1.ff
copy 2.2.4.1.a
create a file 10.5.1.gg
defining occurrence 2.2.2.c
4.10.2.a
denote 1.1.6.c
deprocedureing 8.2, 8.2.2
dereferencing 8.2, 8.2.1
describe 2.2.3.3.b
descriptor 2.2.3.3.a
direct constituent 1.1.6.e
direct production 1.1.2.c
divided by 2.2.3.1.c
dec 5.5.1.ii
elaborate collaterally 6.2.2.a
elaboration 1.1.6.h, 6.0.2.a
element 2.2.2.k
elementary 2.2.5
encountered 4.4.4.b
end of file 10.5.1.cc
English language 1.1.1.b
envelope 1.1.6.j
environment enquiry 10.1
equivalent to 2.2.2.h
establish a file 10.5.1.gg
expect 5.5.1.gg
extended language 1.1.1.a
extension 1.1.7
external object 2.2.1
false 2.2.3.1.e
field 2.2.2.k
12 continued

file 5.5.1.aa, 10.5.1, 10.5.1.ee
firm position 8.2
firmly coerced from 4.4.3.a
following 1.1.6.a
formal language 1.1.1.b
format 2.2.3, 2.2.3.4, 5.5
halt 6.0.2.a
hardware language 1.1.8.b
heap 8.5.1
hipping 8.2, 8.2.7
hold 2.2
home 4.1.2.b
human being 1.1.1.a
hyphen 1.1.6.c.iv
identification string 10.5.1.cc
identify 2.2.2.b
implementation 2.3.c
index 2.2.3.3.a
indication-applied occurrence 4.2.2.a
indication-defining occurrence 2.2.2.c, 4.2.2.a
indit 5.5.1.mm
initiate 2.2.2.g, 6.0.2.a
input 5.5.1.aa, 10.5
instance 2.2.1
integer 2.2.3.1.a, b, c, d
integral equivalent 2.2.3.1.f
internal object 2.2.1
interrupt 6.0.2.a
in the reach of 4.4.2.c
in the sense of numerical analysis 2.2.3.1.c
large syntactic marks 1.1.2.a
layout procedure 10.5.2.1
length number 2.2.3.1.b
list of metanotions 1.1.3.c
list of notions 1.1.2.c
literal 5
lock a file 10.5.1.jj
loosely related 4.4.3.c
lower bound 2.2.3.3.b
lower state 2.2.3.3.b
marked 8.6.1.2
meaningful program 4.4
member 1.1.2.d
metalanguage 1.1.3.a
metamember 1.1.3.d
metanotion 1.1.3.a
minus 2.2.3.1.c
mode 1.1.6.i, 2.2.4.1.a
multiple value 2.2.3, 2.2.3.3
name 2.2.2.1, 2.2.3.5
nil 2.2.2.1, 2.2.3.5.a
notion 1.1.2.a
object 2.2
object program 2.3.c
offset 2.2.3.3.b
of the same mode as 2.2.2.h
open a file 10.5.1.jj
operator-applied occurrence 4.3.2.a
operator-defining occurrence 2.2.2.c, 4.3.2.a
original 1.1.6.c
other syntactic marks 1.1.2.a
output 5.5.1.aa, 10.5
paranotion 1.1.6.c
pass on 6
permanent 2.2.2.a
plain value 2.2.3, 2.2.3.1
point 1.1.2.a
portrayal 2.2.4.1.d
position 5.5.1.ff
possess 2.2.2.d
possibly intended 2.3.c
pragmatic 1.3
precede 1.1.6.a
preelaboration 1.1.6.i
premode 1.1.6.i
prescope 1.1.6.i
present 5.5.1.ff
prevalue 1.1.6.i
proceduring 8.2, 8.2.3
production 1.1.2.e
production rule 1.1.2.a
productive 1.1.2.d
program 1.1.1.a
proper program 4.4
protect 6.0.2.d
protonotion 1.1.2.b
publication language 1.1.8.b
quintuple 2.2.3.3.b
random access 10.5.1.bb
reach 4.4.2.a, 4.4.2.c
read 5.5.1.jj
real number 2.2.3.1.a, b, c, d
refer to 2.2.2.h
related 4.4.3.b
relationship 2.2
repetitive statement 9.3
representation 1.1.8.a
representation language 1.1.1.a
reset a file 10.5.1.mm
resume 6.0.2.a
routine 2.2.2.f
rowing 8.2, 8.2.6
scope 1.1.6.i, 2.2.3.5.a
scratch a file 10.5.1.hh
select 2.2.3.1.f
semicolon 1.1.2.a
sequential access 10.5.1.bb
set a file 10.5.1.oo
shield 4.4.4.a
show 4.4.4.b
smaller than 2.2.2.h
small syntactic marks 1.1.2.a
soft position 8.2
sort 6
standard declaration 10.a
standard file 10.5.1.3
standard mathematical constant 10.3
standard mathematical function 10.3
standard operation 10.2
standard priority 10.2.0
straightening 10.5.0.2
strict language 1.1.1.a
stride 2.2.3.3.b
string 5.5.1.bb
strong position 8.2
structured value 2.2.3, 2.2.3.2
structured from 2.2.4.1.d
subvalue 2.2.2.k
successor 6.0.2.a
supersede 8.3.1.2.a
suppress 5.5.1.11
symbol 1.1.2.b
synchronization operation 10.4
syntactic position 8.2
terminal production 1.1.2.f
terminator-string 5.5.1.jj
12 continued

textual order 1.1.6.a
Times 2.2.3.1.c
transput 5.5.1.aa, 10.5
transput declaration 10.5
true 2.2.3.1.e
truth value 2.2.3.1.a, e
united from 4.4.3.a
uniting 8.2, 8.2.4

upper bound 2.2.3.3.b
upper state 2.2.3.3.b
value 1.1.6.i, 2.2.3
voiding 8.2, 8.2.8
weak position 8.2
widening 2.2.3.1.d, 8.2, 8.2.5
write 5.5.1.gg

{Denn eben, wo Begriffe fehlen,
Da stellt ein Wort zur rechten Zeit sich ein.
Faust, J.W. von Goethe.}