

**stichting  
mathematisch  
centrum**



---

AFDELING INFORMATICA  
(DEPARTMENT OF COMPUTER SCIENCE)

IN 15/78

JULY

J.C. VAN VLIET

AN IMPLEMENTATION MODEL OF THE ALGOL 68 TRANSPUT  
DRAFT VERSION

---

**2e boerhaavestraat 49 amsterdam**

BIBLIOTHEEK MATHEMATISCH CENTRUM  
—AMSTERDAM—

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

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

---

AMS (MOS) subject classification scheme: 4.22, 4.41

---

ACM-Computer Review-categories: 68A15

An Implementation Model of the ALGOL 68 Transput  
Draft version

by

J.C. van Vliet

ABSTRACT

This report aims at a precise definition of the transput of ALGOL 68, conforming with section 10.3 of the Revised Report on the Algorithmic Language ALGOL 68. Whereas section 10.3 of the Revised Report describes the intention of transput, the emphasis in this report is on implementability. The present report is prepared for the meeting of the Working Party on Transput, to be held in Amsterdam on August 23-24, 1978. It describes the state of the model as of July 1978.

KEY WORDS AND PHRASES: ALGOL 68, transput, portability, runtime system,  
ALGOL 68 implementation.



1. INTRODUCTION.

{Presumably, this introduction is still rather floating. This draft describes the state of the implementation model as of July 1978. Chapters 1-7 are more or less copied from IW90. Chapters 8-11 are almost completely rewritten. The ALGOL-68 texts in these Chapters have been tested quite thoroughly, and behave well on all test cases. Their efficiency is comparable to that of the transput system of the CDC ALGOL-68 implementation.}

This report aims at a precise definition of the transput of ALGOL 68, conforming with section 10.3 of the Revised Report on the Algorithmic Language ALGOL 68 (henceforth the Revised Report). Whereas section 10.3 of the Revised Report describes the intention of transput, the emphasis in this report is on implementability.

A variety of ALGOL 68 implementations exist or are near completion. They all support some kind of transput, although they all differ slightly from each other and from the Revised Report [2-8]. This diversity renders the porting of programs from one implementation to the other very difficult, if not virtually impossible.

The existence of so many different transput systems may to some extent be due to the fact that the description as given in the Revised Report does not really facilitate implementation of the transput. Each implementer again has to struggle his way through the transput section and locate the problems with the particular operating system. It is hoped that the definition in this report will solve most of these problems once and for all. THOMSON & BROUGHTON [9,10] and R.FISKER of Manchester have been working on transput systems of a similar structure.

The present report is written on request of the Working Party on Transput, which has been set up by the Subcommittee for ALGOL 68 Support of Working Group 2.1 of IFIP. A preliminary version appeared in [16]; that version was discussed at the first meeting of the Working Party in Oxford, December 14-15, 1977. The main objections against [16] concerned its efficiency (most notably string processing and formatted transput). These objections, and many other minor remarks, have been taken care of by now.

{A first implementation of the transput as defined in [16] is by now available on a TR 440 at the University of Bochum (West-Germany). This implementation does not support formatted transput. It was finished within 4 months, solely from the description in [16], which places some confidence in its implementability. At this moment also, work is progressing at the Mathematical Centre on an implementation on a CDC Cyber under the NOS-BE operating system.}

The approach taken is similar to the one in the Revised Report: the transput is described in pseudo-ALGOL 68. The pseudo part can be considered as a language extension which is reasonably implementable. The primitives underlying the model are not defined in ALGOL 68. Instead, their semantics are given in some kind of formalized English, resembling the way in which the semantics in the Revised Report is defined. One advantage of a description in pseudo-ALGOL 68 is that it can largely be tested mechanically. It might even be possible to feed this description of the transput into the compiler, thus automatically creating part of the runtime

environment.

Care has been taken to stick to the Revised Report as closely as possible so far as the meaning is concerned. So the full transput is defined, even those features whose value may be questioned (like, e.g., the elaboration of dynamic replicators upon staticizing a picture). At a few places however, a different meaning is assigned to certain features. These differences are clearly indicated as such in the sections headed 'DIFFERENCES'.

At its first meeting in December 1977, the Working Party on Transput also discussed a host of reported errors and problems in the area of transput. Most of these are collected in [F4]. The Working Party is preparing a document containing solutions and answers to all these problems. This document will be submitted for formal approval to the Subcommittee on ALGOL 68 Support. The changes caused by this document have already been taken care of in the present report.

The main differences between the transput section presented here and the one from the Revised Report are:

- i) Books are considered to form part of the operating-system interface; as such, the mode BOOK is not specified;
- ii) On many systems, not all of the text of a file is available at each instant of time. This has been made explicit in the present model by starting from a "buffer" concept. {The consequences of this approach permeate through the whole transput section!};
- iii) The numerous calls of 'undefined' in the Revised Report have been assigned meanings. Hidden kinds of undefined actions like SKIP, UP gremlins and UP bfileprotect have been paid due attention;
- iv) The number of tests that is performed for each transput operation is minimized. For many routines, pre- and postconditions have been chosen carefully, so as to achieve security with a minimum of tests;
- v) By choosing a different structure for the mode FORMAT, remarkable simplifications and optimizations have been made in the section on formatted transput.

The differences between the present model and the definition in the Revised Report have been collected in the sections headed 'DIFFERENCES'. The following classes are distinguished:

- i) Differences in the descriptive method being used. Only major changes in the descriptive method are listed, such as the change in the internal representation of formats. They are marked {D};
- ii) Definition of 'undefined'. Wherever the Revised Report uses 'undefined', the present model prescribes a more precise action, such as for instance the emission of some error message. These are marked {U};
- iii) Violation of the semantics as given in the Revised Report. There are a few places where the semantics of the Revised Report are felt to be unreasonable. Care has been taken to keep the number of such changes minimal, since they really affect the behavior of user programs. These differences are marked with an {S};
- iv) Fixing admitted bugs. The most important bugs in the transput section of the Revised Report are listed under {B}, although the user should consult [17] for a complete list of them;
- v) Extensions. At a few places also, extensions to the definition as given in the Revised Report are defined. They are marked {E}.

Care has been taken to ensure reasonable efficiency of the whole system. Of course, efficiency is most important for heavily used procedures like 'put' or 'whole'. It might well be worthwhile to tune such routines to a specific operating environment. At places where efficiency is felt to be essential, indications are given that may help the implementer.

Acknowledgements. The author has benefited much from his correspondence and discussions with H. Wupper of the Ruhr-University at Bochum, B. Leverett of Carnegie-Mellon University, C. Cheney of Cambridge, R. Fisker and Ch. Lindsey of Manchester, A.N. Maslov of Moscow, L. Meertens, H. Boom and D. Grune of the Mathematical Centre, and last but not least Section 10.3 of the Revised Report itself.

## 2. UNDERLYING PRIMITIVES.

A model that is intended to be easily implementable on a variety of machines, must be described in such a way that it is clear which aspects are machine dependent, and which are not. There must be a clearly defined set of primitives underlying the transput, and this set must in some sense be small. These primitives then form the operating-system interface. The "meaning" of these primitives must also be defined.

The primitive that lies at the very heart of the present model is the "buffer". In the Revised Report, both the book and the file contain the "text", which is a reference to a three-dimensional character array. For most files on most systems, not all of the three-dimensional character array will be available at any instant of time. This restriction is made explicit in the present model; the piece of text that is available is called the "buffer". Preferably, a buffer corresponds to one line of the text. It is however anticipated that there will be files containing only one page which consists of one huge line (just think of a papertape). In that case, the buffer will probably correspond to a much smaller piece of text. The same holds for files that are used interactively, where the system possibly transputs units of information of the size of a number, say. There also exist systems where the buffer has a fixed size (typically 512 "words"), regardless of the file being used. One immediate consequence of this is that the routine 'backspace', as defined in the Revised Report, cannot be implemented on such a file. The present model deviates from the Revised Report in that an enquiry 'backspace possible' is defined for files, with a function very similar to that of the enquiries 'set possible' and the like.

If buffers are used as units of transput information, there must be ways to write a buffer and read the next buffer. Machine-dependent primitives are needed for this. They are discussed at full length in later sections.

As the internal structure of the buffer is not specified either, primitives for reading and writing single characters from or to the buffer are needed also.

This scheme offers attractive possibilities with regard to "conversion keys": The Revised Report specifies characters to be converted from internal form to external form (and vice versa) before they are actually transput. This conversion is done per character; not only is control given back to the main program if a character cannot be converted, but conversion keys may also be changed per character. It is however not at all sure that characters may be punched alternately in ASCII and EBCDIC, say. It is to be expected that on many devices there will be only limited possibilities to change the conversion key. Maybe another conversion key can be provided only if the file is positioned at the very beginning of a book, or at the beginning of a line. Probably, there will also be channels on which a change of conversion key is altogether impossible. On the other hand, there exist line-printers on which a change of conversion key can be achieved on a line to line basis with one hardware instruction. The present model is flexible enough to allow these very different implementations of the conversion-key concept: having primitives both to transput one buffer and to transput single characters to and from the buffer, the implementer is free to do the conversion in either of these. He may even decide to do no conversion at all, or to do it differently on different channels. Changing the conversion key then amounts to changing the above set of primitives.



(Note that, if there are no conversion keys, the buffer need not be made primitive, but can be defined instead as

```
MODE BUFFER = REF [] CHAR.
```

This obviates the need for primitives to transput characters to and from the buffer. If there are conversion keys, the buffer cannot be defined so simply, since there is no reason to expect that internal characters still look like characters after conversion! Also, it may be convenient and efficient to keep the buffer in a machine-oriented form rather than as an ALGOL 68 row-of-character, even if there is no conversion key.) For further details on the buffer and its primitives, see section 4.2.2.1.

It should be emphasized here that the Working Party on Transput strongly dissuades the use of conversion keys. Conversion keys are felt to be not very useful, whilst they strongly influence the efficiency of the system. The consequences of the absence of conversion keys are clearly indicated in subsequent Chapters.

Almost all machine dependencies in the present model are incorporated in the channel. A channel is a set of attributes that is common to some set of devices. It is anticipated that most machine dependencies will differ for files opened on different channels, but will be the same for files opened on one and the same channel. {Resetting the position of a file will probably be different for tape and disk, but will probably be identical for two files that are both opened on disk.} It is thus advantageous to incorporate these primitives in the channel. This approach is also taken in the ALGOL68S implementation at Carnegie Mellon University [11,12] and at Manchester [13]. (The ALGOL68S implementation at Carnegie Mellon also uses a buffer concept, albeit a slightly different one [12].)

A full list of the primitives that have been incorporated into the channel is:

- read buffer, write buffer, init buffer;
- get char, put char, get bin char, put bin char;
- newpage, newline;
- (part of) set, (part of) reset.

## 3. POSITIONS

Positions within the text are indicated by a page number, a line number and a character number. Two positions are of importance during transput:

- the "logical end", i.e., the position up to which the book has been filled with information;
- the "current position", i.e., the position where the next transput operation will (normally) operate on.

Before any actual transput operation may take place, the validity of the current position has to be ensured. Whether a given position is "valid" depends on the kind of operation that is desired. {If a newpage is to be given, only the page number has to be within its bounds; if a character is written, both the line number and the character number must be within their respective bounds as well.} If one of the position entities need not be within its bounds, it may be off by one at the upper end; in that case, the line, page or book is said to have "overflowed".

{If  $p$ ,  $l$  and  $c$  denote the page number, line number and character number, then a typical text may look as follows:

		c=	1	2	3	4	5	6	7	8	
p=1	l=1	.	.	.	.	.	.	.	.	.	.
	2	.	.	.	.	.	.	.	.	.	.
	3	.	.	.	.	.	.	.	.	.	.
	4	.									.

  

	l=1	.	.	.	.	.	.	.	.	.	.
p=2	2	.	.	.	.	.	.	.	.	.	.
	3	.									.
	4	.	.	.	.	.	.	.	.	.	.
	5	.									.

  

p=3    l=1    .

Possible positions are indicated by ".", although information can only be present at positions within a box. For the dots that are not placed within a box, the condition "line ended" holds. At  $\langle 1, 4, 1 \rangle$ ,  $\langle 2, 5, 1 \rangle$  and  $\langle 3, 1, 1 \rangle$  the condition "page ended" holds too, while at  $\langle 3, 1, 1 \rangle$  the condition "physical file ended" holds as well.}

{Note that if the page has overflowed, the current line is empty (so the line has overflowed too), and, if the text has overflowed, the current page and line are both empty (so the line and page have both overflowed).}

If the current position has overflowed the line, page or book, then it is said to be outside the "physical file". The position where the book has overflowed is termed the "physical file end". (There is only one such position!)

If, on reading, the current position is at the logical end, then it is said to be outside the "logical file".

{Note that, if " $\leq$ " denotes the intuitive lexical ordering, then the invariant

current position <= logical end <= physical file end  
always holds.}

If the current position is outside the (physical or logical) file, the next transport operation will call an "event routine" (except possibly when 'set', 'reset', 'set char number' or 'backspace' is called). Event routines are provided by default, but may be changed by the user (for more details, see 4.3.2). The event routines for the overflow conditions correspond to the following "on routines":

- on logical file end;
- on physical file end;
- on page end;
- on line end.

If an overflow condition occurs, the event routine corresponding to the appropriate "on routine" is called. Since more than one overflow condition may occur at one and the same position (e.g., if the page is ended, the line is also ended), it is important to know which event routine is called in each situation. Obviously, only one out of "physical file ended" and "logical file ended" can occur: on reading the logical end of the file may be reached, on writing the physical end may be reached. The ordering of the calling of event routines is such that physical (logical) file end has preference over page end, while page end has preference over line end.

{One striking difference between the physical file end and the logical file end is that the logical end may occur at each valid position (each position with a dot in the above picture), while the physical end may only occur at the first position on a new page. This will certainly lead to difficulties if straightforwardly implemented: it necessitates knowledge about the size of the current page (which may be compressible) and knowledge about the space that is still available for the text. On many systems, a record that is written to a tape may or may not fit, and this is not known until it is actually tried! If it does not fit, the operating system will usually tell so. It is thus implementation dependent when exactly the event routine corresponding to "on physical file end" is called, and the event routine may have to be called from within the routine that writes a buffer to the book.}

The transport system has to administrate both the logical end and the current position. On reading this is relatively simple: for each character that is read, the current position is advanced over one position until the logical end is reached. Writing is much more complicated. The effect of a write operation depends on:

- whether the current position is before or at the logical end;
- whether the file is random or sequential access;
- whether the file is compressible or not;
- whether it is a layout routine that is called.

It is usually a combination of some of the above possibilities that leads to special effects. These effects are all treated in full detail in later sections.

Writing at the logical end leads to advancing the logical end along with the current position. The logical position ("lpos") will then keep pace with the current position ("cpos"). This is expected to be the normal every-day situation if files are written to. The Revised Report maintains the most important invariant that connects 'cpos' and 'lpos':

I: cpos ≤ lpos

by statements like

{I} cpos += 1; (cpos > lpos | lpos += 1) {I}.

In the present model, a new pair of variables (cpos1, lpos1) is introduced. This pair is related to (cpos, lpos) through:

cpos = cpos1  
lpos = MAX (cpos1, lpos1).

The same invariant I is now ensured by

{I} cpos1 += 1 {I}.

Care has to be taken if the current position is set back (which may occur through a call of 'set', 'reset', 'set char number' or 'backspace'). In that case the value of 'cpos1' must first be assigned to 'lpos1' iff 'lpos1' < 'cpos1' (the routine 'mind logical pos' serves this purpose). 'cpos1' and 'lpos1' are again called 'cpos' and 'lpos' in the text. It is expected that a considerable gain in efficiency will result from this change of representation.

#### 4. BOOKS, CHANNELS AND FILES.

{ "Books", "channels" and "files" model the transput devices of the physical machine used in the implementation. }

##### 4.1. BOOKS

###### 4.1.1. DEVIATIONS FROM THE REVISED REPORT

- {D} The various fields of the mode BOOK are not specified. Instead, routines are provided to construct books, or change certain fields of a book.
- {D} It is not prescribed that books be kept in two chains. Books somehow reside in the system and are searched for in some unspecified way if asked for one. In the Revised Report, multiple references are needed to books that allow multiple access. Each time a file is opened on such a book, one of these references is removed from the chain, and a counter in the book is increased. {Note that a book may only be opened on more than one file simultaneously if writing is not possible on any of them.} It is not defined in the present system how this administration is exactly performed. In the Revised Report, critical sections of routines that access the chains of books are protected by the semaphore 'bfileprotect'. In the present system, all these critical sections are behind the curtain, and so is the semaphore.
- {U} The default identification for files which are opened via 'create' is not left undefined. Rather, an operating-system dependent string is assumed, which may differ from case to case.

###### 4.1.2. NEW DEFINITION

Books model the actual devices on which the transput takes place. All information within the system is to be found in a number of such books. The information that is kept in the book is operating-system dependent information, such as the identification string. Via the book, the text may be reached; this text contains the actual information. The text has a variable number of pages, each of which may have a variable number of lines, each of which may have a variable number of characters. Positions within the text are indicated by a page number, a line number and a character number. The book includes a field which either indicates the "logical end" of the book, i.e., the position up to which it has been filled with information, or it indicates that this position is not known yet. Except for 'set', no routine needs exact knowledge of the logical end of the book, as long as it is not in the current line.

Books are either searched for in the system, or constructed according to certain user-defined requisites. Books are searched for by means of the routine 'book in system'. A book is constructed by one of the routines 'construct book' and 'pseudo book'.

a) MODE ? BOOK =

C An actual declarer specifying a mode which contains at least the following operating-system dependent information:

- the identification string;
- some reference to the actual information;
- the logical end of the book {, which may however be unknown};
- information that tells how many other processes have access to this book, and whether some process is writing to the book. C;

b) MODE ? POS = STRUCT(INT p, l, c);

c) PRIO ? EXCEEDS = 5,

OP EXCEEDS = (POS a, b) BOOL:

p OF a > p OF b OR l OF a > l OF b OR c OF a > c OF b;

d) PRIO ? BEYOND = 5,

OP BEYOND = (POS a, b) BOOL:

IF p OF a < p OF b THEN FALSE

ELIF p OF a > p OF b THEN TRUE

ELIF l OF a < l OF b THEN FALSE

ELIF l OF a > l OF b THEN TRUE

ELSE c OF a > c OF b

FI;

{Positions are indicated by values of the mode POS. They are compared by the operators EXCEEDS and BEYOND.}

e) PROC ? book in system =

(STRING idf, CHANNEL chan, REF REF BOOK book, REF REF BUFFER buffer)

INT:

C The pool of available books is searched for a book with the following properties:

- the book may be identified by 'idf';
- the book may be legitimately accessed through 'chan';
- opening is not inhibited by other users of the book.

{A book may only be opened on more than one file simultaneously if putting is not possible on any of them.}

If such a book is found, it is assigned to 'book', space for a buffer is made available to which 'buffer' is made to refer, and the routine yields 0. Otherwise, the routine yields some positive integer that corresponds to the appropriate error code. (For a list of these error codes, see section 5.2.) C;

f) PROC ? construct book =

(INT p, l, c, STRING idf, CHANNEL chan, REF REF BUFFER buffer)

REF BOOK:

C A book is constructed with a text of the size indicated by (p, l, c), with an identification string 'idf', that may be written to. The book is to be accessed via 'chan'. The routine allocates space for a buffer to which 'buffer' is made to refer; the buffer is not initialized. The logical end of the book is set to (l, l, l). A reference to the book is yielded as result. The validity of the parameters has already been checked by the routine 'establish', which calls 'construct book'. C;

- g) PROC ? pseudo book = (INT p) REF BOOK:  
C A book is constructed whose only interesting information is its logical end, which is set to (p + 1, 1, 1). The routine is called by 'associate'(see 5.2.d). C;
- h) PROC ? default idf = STRING:  
C A default identification string for files which are opened via 'create' C;
- i) PROC ? set logical pos = (REF FILE f) VOID:  
BEGIN  
C The logical end of the book of 'f' is set to the current position  
C;  
REF INT(c of lpos OF f):= c OF cpos OF f;  
status OF f ANDAB logical file ended  
END;

{This routine replaces the pseudo comment in 'put char'.}

## 4.2. CHANNELS

### 4.2.1. DEVIATIONS FROM THE REVISED REPORT

- {S,E} A field 'backspace' is added, which determines whether the routine 'backspace', as defined in section 10.3.1.6.b of the Revised Report, is possible on the channel.
- {D} The conversion key is understood to be a set of routines that define how buffers are transput to and from the book, and how characters are transput to and from the buffer.
- {D} The primitives 'do newline', 'do newpage', 'do set' and 'do reset' have been incorporated in the channel.

### 4.2.2. NEW DEFINITION

A "channel" is a collection of attributes that is common to some set of devices. A channel is a structured value and has two kinds of fields:

- routines returning truth values which determine the available methods of access to a book linked via that channel;
- primitives for device-class dependent actions.

Since the methods of access to a book may well depend on the book as well as on the channel, most of these routines depend on both the channel and the book. The access properties may be examined by use of the environment enquiries for files (4.3.2). Two environment enquiries are provided for channels. These are:

- 'estab possible', which returns true if another file may be "established" (5.2.a) on the channel;
- 'standconv', which may be used to obtain the default "conversion key".

The fields that are primitive actions all depend on the file, since they may have side effects on specific fields of the file {e.g., 'do newpage' will result in a new value for the current position}.

A "conversion key" is a value of the mode specified by CONV, which is used to convert characters to and from the values as stored in "internal" form and as stored in "external" form in a book. A conversion key consists of two parts:

- routines to transput buffers to and from the book and a routine to initialize the buffer;
- routines to transput characters to and from the buffer.

It is not prescribed in the present model whether the actual conversion (if any) is done per line or per character. {This may differ from one implementation to another, or even for two different conversion keys used on one and the same channel.} The implementation may provide additional conversion keys in its library-prelude. After linking a file to a book via some channel, the buffer must be initialized. For files which are "opened" (5.2), this initialization is not done until the first transput operation is initiated, upon which an explicit call of 'init buffer' takes place.

The routines 'set', 'reset' and 'newpage' may cause the current position to be moved outside the current line. {Nevertheless, they may to a great extent be written in proper ALGOL 68. For instance, a routine 'newpage' may be written that searches for an end-of-page condition, in the meantime skipping lines that have already been filled, and filling empty lines with blanks.} It is expected that on most systems a newpage is only possible on





## 4.2.2.1. SEMANTICS OF THE CHANNEL PRIMITIVES

In this section, both the semantics of the routines that comprise the conversion key, and the primitives that can move the current position outside the current buffer ('do newline', 'do newpage', 'do reset' and 'do set') are given.

In the routines given below, the buffer is supposed to have a (boolean) field 'changed' that tells whether the contents of the buffer have changed since its initialization. This information is used by 'do newline': the contents of the buffer of a random-access file is not written back if there is no need to. If this field is not supported, the value of 'put possible(f)' may be used instead. (This field is also inspected by the routines 'close', 'lock' and 'scratch' (5.2.1,m,n).)

If the buffer corresponds to one line of the text, the semantics are somewhat easier to describe than in the case where a buffer corresponds to a smaller unit of information. Therefore, this case is treated first.

```
a) PROC init buffer = (REF FILE f) VOID:
    # The precondition of 'init buffer' is:
        . opened,
        . either read mood or write mood,
        . NOT physical file ended. #
BEGIN status OF f ORAB buffer filled;
    IF get possible(f) AND status OF f SAYS logical pos ok
    THEN changed OF buffer OF f:= FALSE;
        IF C the page of the book of f has overflowed C
        THEN status OF f ANDAB page end;
            REF INT(char bound OF f):= 0
        ELSE
            buffer OF f:= C the next buffer from the book of f, possibly
                after conversion C;
            REF INT(char bound OF f):=
                C the length of the buffer just filled C;
            test line end(f);
            IF C the logical end is in the buffer just read C
            THEN status OF f ANDAB lfe in current line;
                REF INT(c of lpos OF f):= C the position of the logical end C;
                (status OF f SAYS read mood ! test logical file end(f))
            FI
        FI
    ELSE changed OF buffer OF f:= NOT get possible(f);
        IF C the page of the book of f has overflowed C
        THEN status OF f ANDAB page end;
            REF INT(char bound OF f):= 0
        ELSE REF INT(char bound OF f):=
            C the maximum length of this buffer C;
            test line end(f)
        FI
    FI
END;
```

- b) PROC read buffer = (REF FILE f) VOID:  
 # The precondition of 'read buffer' is:  
 . opened,  
 . get possible(f),  
 . NOT logical file end in current line,  
 . NOT page ended.#  
 BEGIN REF POS(cpos OF f):= (p OF cpos OF f, 1 OF cpos OF f + 1, 1);  
 (init buffer OF f)(f)  
 END;
- c) PROC write buffer = (REF FILE f) VOID:  
 # The precondition of 'write buffer' is:  
 . opened,  
 . write mood,  
 . NOT physical file ended,  
 . NOT page ended.#  
 BEGIN C The contents of the buffer of f (up to the position indicated  
 by 'c OF cpos OF f') is, possibly after conversion, written to  
 the book of 'f' C;  
 IF C this fails to succeed (i.e., the physical end of the book is  
 reached while writing this buffer) C  
 THEN status OF f ANDAB physical file end;  
 ensure physical file(f, status OF f);  
 # Note that, since we are writing to the book, this call can never  
 fail; either the situation is mended, or the program is aborted  
 after a suitable error message has been issued. #  
 (write buffer OF f)(f)  
 ELSE REF POS(cpos OF f):= (p OF cpos OF f, 1 OF cpos OF f + 1, 1);  
 IF status OF f SUGGESTS lfe in current line  
 THEN set logical pos(f)  
 FI;  
 (init buffer OF f)(f)  
 FI  
 END;
- d) PROC put char = (REF FILE f, CHAR char) VOID:  
 # The precondition of 'put char' is:  
 . line ok (see 7.2). #  
 BEGIN C The character in 'char' is (possibly after conversion) written  
 to the buffer of f at the position indicated by 'c OF cpos OF  
 f' C;  
 c OF cpos OF f += 1;  
 changed OF buffer OF f:= TRUE  
 END;
- e) PROC put bin char = (REF FILE f, BINCHAR char) VOID:  
 # The precondition of 'put bin char' is:  
 . line ok. #  
 BEGIN C The binary character in 'char' is written to the buffer of f at  
 the position indicated by 'c Of cpos OF f' C;  
 c OF cpos OF f += 1;  
 changed OF buffer OF f:= TRUE  
 END;

```

f) PROC get char = (REF FILE f, REF CHAR char) BOOL:
    # The precondition of 'get char' is:
    . line ok. #
    BEGIN CHAR c = C the character read (and possibly converted) from the
        buffer of f at the position indicated by 'c OF cpos OF
        f' C;
        c OF cpos OF f += 1;
        IF C the conversion succeeds, or no conversion takes place C
        THEN char:= c; TRUE
        ELSE FALSE
        FI
    END;

g) PROC get bin char = (REF FILE f, REF BINCHAR char) VOID:
    # The precondition of 'get bin char' is:
    . line ok. #
    BEGIN char:= C The binary character read from the buffer of f at the
        position indicated by 'c OF cpos OF f' C;
        c OF cpos OF f += 1
    END;

h) PROC do newline = (REF FILE f) VOID:
    # The precondition of 'do newline' is:
    . page ok (see 7.2). #
    IF NOT (status OF f SAYS buffer filled)
    THEN C skip over the current line C;
        IF C this causes the physical end to be reached C
        THEN (status OF f ORAB buffer filled) ANDAB
            physical file end
        ELIF C it causes the logical end to be reached C
        THEN (init buffer OF f)(f)
        ELIF C the page has overflowed C
        THEN status OF f ANDAB page end;
            REF INT(char bound OF f):= 0
        FI
    ELIF status OF f SUGGESTS lfe in current line AND
        status OF f SAYS read mood
    THEN REF INT(c OF cpos OF f):= c of lpos OF f;
        test logical file end(f); newline(f)
    ELSE
        IF status OF f SUGGESTS lfe in current line
        THEN REF INT(c OF cpos OF f):= c of lpos OF f;
            C fill the rest of the buffer with spaces if the file is not
            compressible C
        FI;
        IF changed OF buffer OF f
        THEN (write buffer OF f)(f)
        ELSE (read buffer OF f)(f)
        FI
    FI;

```

```

i) PROC do newpage = (REF FILE f) VOID:
  # The precondition of 'do newpage' is:
    . physical file ok (see 7.2). #
  BEGIN
    WHILE NOT (status OF f SAYS page end)
    DO (do newline OF chan OF f)(f) OD;
    REF POS(cpos OF f):= (p OF cpos OF f + 1, 1, 1);
    IF C this causes the physical file to be ended C
    THEN (status OF f ORAB buffer filled) ANDAB physical file end
    ELSE (init buffer OF f)(f)
    FI
  END;

```

{This routine is incorporated in the channel for optimization purposes only: it is expected that more efficient implementations will in general be available.}

```

j) PROC do reset = (REF FILE f) VOID:
  # The precondition of 'do reset' is:
    . opened,
    . reset possible. #
  BEGIN REF POS(cpos OF f):= (1, 1, 1);
    C the book is physically reset C;
    (status OF f ANDAB NOT buffer filled) ORAB open status
  END;

```

```

i) PROC do set = (REF FILE f, INT p, 1, c) VOID:
  # The precondition of 'do set' is:
    . opened,
    . set possible. #
  IF POS(1, 1, 1) EXCEEDS POS(p, 1, c)
  THEN error(posmin); abort
  ELIF C The line indicated by 'p' and '1' is searched for. Note that the
    current position is updated while searching. Searching may stop
    at the following positions:
    - at the physical file end if 'p' exceeds the number of pages in
      the book of 'f';
    - just beyond the page indicated by 'p' if '1' exceeds the
      number of lines in that page;
    - at the first position of the last (logical) line if the
      position indicated by 'p', '1' and 'c' is beyond that
      position;
    - at the first position of the line indicated by 'p' and '1'
      otherwise. C
    POS(p, 1, 1) EXCEEDS cpos OF f OR c > char bound OF f + 1
  THEN error(posmax); abort
  ELIF (init buffer OF f)(f);
    status OF f SUGGESTS lfe in current line
  THEN STATUS reading = state(f);
    IF c > c of lpos OF f
    THEN c OF cpos OF f:= c of lpos OF f;
      (reading SAYS read mood | test logical file end(f));
      BOOL mended = (logical file mended OF f)(f);
      ensure state(f, reading);
      (NOT mended | error(wrongset); abort)
    ELSE c OF cpos OF f:= c; test line end(f);

```

```
(reading SAYS read mood | test logical file end(f))
FI
ELIF c > 1 AND NOT get possible(f)
THEN error(setmiddle); abort
ELSE c OF cpos OF f:= c;
    test line end(f); test logical file end(f)
FI;
```

In case the buffer does not correspond to one line of the text, the routines essentially remain the same. The value of the current position will in that case not directly lead to a position in the buffer, but some offset is needed (which must be properly set by 'init buffer'); this offset is assumed to be incorporated in the buffer. Normal transput which takes place via 'put char OF f' and 'get char OF f' then assumes that a next buffer is automatically initiated as soon as the current buffer has overflowed.

### 4.3. FILES

#### 4.3.1. DEVIATIONS

- {D} The file includes a field 'buffer', which contains a reference to the current buffer. Transput is performed on this buffer, rather than on the text of the book (but see section 5.2 on associated files). The field 'char bound' indicates the maximum number of characters that the current buffer can contain.
- {D} The file includes a field 'status', which contains the status information. This status is inspected before, and updated after, each transput operation.
- {D} A field 'c of lpos' is included in the file. If the logical end is in the current line (which can be derived from the status information), this field indicates the position of the logical file end (and otherwise it is left undefined).
- {S,E} An enquiry 'backspace possible' is provided, which returns true if the file may be "backspaced" (7.2).

#### 4.3.2. NEW DEFINITION

A "file" is the means of communication between a particular-program and a book which has been opened on that file via some channel. It is the most heavily used concept in the transput section. In the present model, it is considered to be largely machine independent.

A file is a structured value which includes a reference to the book to which it has been linked (5.2). The file includes a reference to the text, which is used for associated files (5.2) and a reference to the buffer, which is used for non-associated files. The file also contains information necessary for the transput routines to work with the book, including its current position "cpos" in the text, its current "status", a reference to its current "collection list" (11.2) and the channel on which it has been opened.

The "status" of a file contains the following information:

- whether or not the file has been opened;
- whether or not the buffer has been initialized;
- whether or not the line has overflowed;
- whether or not the page has overflowed;
- whether or not the physical file has overflowed;
- whether or not the logical file is ended;
- whether or not the logical file end is in the current line. (If the logical file end is in the current line, the 'c of lpos'-field points to this logical file end.);
- the "mood" of the file, which is determined by four booleans:
  - 'read mood', which is true if the file is being used for input;
  - 'write mood', which is true if the file is being used for output;
  - 'char mood', which is true if the file is being used for character transput;
  - 'bin mood', which is true if the file is being used for binary transput;
- whether or not the file may be set.

{The present model supposes that it is always possible to determine whether a given file has been opened. So, after a declaration

```
FILE f,
```

or even

```
REF FILE f = SKIP,
```

it must be possible to detect that 'f' is not opened. To this end, all user-callable routines may be assumed to start with an implicit test for the 'status'-field being available. It is not defined here how this can be done in an actual implementation.}

The file also contains the current conversion key, i.e., the set of routines that is currently being used to transput buffers to and from the book and characters to and from the buffer. After opening a file, the conversion key from the channel on which it is opened is provided by default. Some other conversion key may be provided by the programmer by means of a call of 'make conv' (m).

{Note that changing the conversion key may depend on the book, the channel, the current position, both the current and the new conversion key, and other environmental factors not defined by this model.}

The routine 'make term' is used to associate a string with a file. This string is used when inputting a variable number of characters, any of its characters serving as a terminator. {For efficient use of this feature, implementation through a bit table seems natural. A similar feature can profitably be used in 'get' (10.2.c) and 'getf' (11.2.p). To this end, the mode CHARBAG is introduced, together with an operator STRINGTOBAG and a routine 'char in bag' (10.2.f).}

The available methods of access to a book which has been opened on a file may be discovered by calls of the following routines (note that the yield of such a call may be a function of both the book and the channel):

- 'get possible', which returns true if the file may be used for input;
- 'put possible', which returns true if the file may be used for output;
- 'bin possible', which returns true if the file may be used for binary transput;
- 'compressible', which returns true if lines and pages will be compressed (7.2) during output, in which case the book is said to be "compressible";
- 'reset possible', which returns true if the file may be reset, i.e., its current position set to (1, 1, 1);
- 'set possible', which returns true if the file may be set, i.e., the current position changed to some specified value; the book is then said to be a "random access" book and, otherwise, a "sequential access" book. For optimization reasons, this information is also incorporated in the status of the file;
- 'backspace possible', which returns true if the file may be backspaced, i.e., the current position set back over one position if it remains within the current line; backspacing will always be possible if the buffer of the file corresponds to one line of the text {but may for instance not be possible for files that are used interactively};
- 'feidf possible', which returns true if the 'idf' field of the book may be changed;
- 'chan', which returns the channel on which the file has been opened (this may be used, for example, by a routine assigned by 'on physical file end', in order to open another file on the same channel).



{Not all combinations of the above set are sensible. For instance, it is expected that at least one of 'put possible' and 'get possible' holds. Most likely also, 'reset possible' returns true if 'set possible' does and 'establish possible' implies 'put possible'.}

A file includes some "event routines", which are called when certain conditions arise during transput. After opening a file, the event routines provided by default return false when called, but the user may provide other event routines. The event routines are always given a reference to the file as a parameter. If the calling of an event routine is terminated (by a jump), then the transput routine which called it can take no further action; otherwise, if it returns true, then it is assumed that the condition has been mended in some way, and, if possible, transput continues, but if it returns false, then the system continues with its default action. The "on" routines are:

- 'on logical file end'. The corresponding event routine is called when, during input from a book or as a result of calling 'set' (see 7.2), the logical end of the book is reached.
- 'on physical file end'. The corresponding event routine is called when the current page number of the file exceeds the number of pages in the book and further transput is attempted (see 7.2).
- 'on page end'. The corresponding event routine is called when the current line number of the file exceeds the number of lines in the current page and further transput is attempted (see 7.2).
- 'on line end'. The corresponding event routine is called when the current character number of the file exceeds the number of characters in the current line and further transput is attempted (see 7.2).
- 'on char error'. The corresponding event routine is called when, during input, a character conversion was unsuccessful or a character is read which was not "expected" (section 10.3.4.1.11 of the Revised Report). The event routine is called with a reference to a character suggested as a replacement. The event routine provided by the user may assign some character other than the suggested one. If the event routine returns true, then that suggested character as possibly modified is used.
- 'on value error'. The corresponding event routine is called when:
  - i) during formatted transput an attempt is made to transput a value under the control of a "picture" with which it is incompatible, or when the number of "frames" is insufficient. If the routine returns true, then the current value and picture are skipped and transput continues; if the routine returns false, then first, on output, the value is output by 'put', and next the program is aborted;
  - ii) during input it is impossible to convert a string to a value of some given mode (this would occur if, for example, an attempt were made to read an integer larger than 'max int'). If the routine returns true, transput continues (although no value is assigned to the item being read in); if the routine returns false, an error message is given and the program is aborted.
- 'on format end'. The corresponding event routine is called when, during formatted transput, the format is exhausted while some value still remains to be transput. If the routine returns true, then the program is aborted if a new format has not been provided for the file by the routine; otherwise, the current format is repeated.

- a) MODE FILE = STRUCT(  
 REF BOOK ? book,  
 CHANNEL ? chan,  
 REF REF FORMATLIST ? piece,  
 REF BUFFER ? buffer,  
 REF [[]] CHAR ? text, # for associated files only #  
 REF POS ? cpos,  
 REF INT ? c of lpos,  
 CHARBAG ? term,  
 PROC (REF FILE) VOID ? read buffer, ? write buffer, ? init buffer,  
 PROC (REF FILE, CHAR) VOID ? put char,  
 PROC (REF FILE, BINCHAR) VOID ? put bin char,  
 PROC (REF FILE, REF CHAR) BOOL ? get char,  
 PROC (REF FILE, REF BINCHAR) VOID ? get bin char,  
 REF STATUS ? status,  
 REF INT ? char bound,  
 PROC (REF FILE) BOOL ? logical file mended, ? physical file mended,  
 ? page mended, ? line mended, ? format mended,  
 ? value error mended,  
 PROC (REF FILE, REF CHAR) BOOL ? char error mended);
- b) MODE ? BUFFER = C some mode C;
- c) PROC get possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (get OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- d) PROC put possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (put OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- e) PROC bin possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (bin OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- f) PROC compressible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (compress OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- g) PROC reset possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (reset OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;

- h) PROC set possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN NOT (status OF f SAYS not set poss)  
 ELSE error(notopen); abort  
 FI;
- i) PROC backspace possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (backspace OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- j) PROC reidf possible = (REF FILE f) BOOL:  
 IF status OF f SAYS opened  
 THEN (reidf OF chan OF f)(book OF f)  
 ELSE error(notopen); abort  
 FI;
- k) PROC chan = (REF FILE f) CHANNEL:  
 IF status OF f SAYS opened  
 THEN chan OF f  
 ELSE error(notopen); abort  
 FI;
- l) PROC make conv = (REF FILE f, PROC (REF BOOK) CONV c) VOID:  
 IF status OF f SAYS opened  
 THEN  
   C Some implementation-dependent tests will probably be needed here:  
   whether the conversion key may be changed might depend on the  
   current and the newly given conversion key, the book, the channel,  
   and other environmental factors. If the conversion key may be  
   changed, the routines in the file that comprise the mode CONV have  
   to be exchanged. One must take care that conversion keys from  
   associated files (5.2.d) do not get used as parameter to this  
   routine; also, the conversion key of such a file may not be  
   changed. C  
 ELSE error(notopen); abort  
 FI;
- m) PROC make term = (REF FILE f, STRING t) VOID:  
 term OF f:= STRINGTOBAG t;
- n) PROC on logical file end =  
 (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
 logical file mended OF f:= p;
- o) PROC on physical file end =  
 (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
 physical file mended OF f:= p;
- p) PROC on page end =  
 (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
 page mended OF f:= p;

- q) PROC on line end =  
    (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
    line mended OF f:= p;
- r) PROC on format end =  
    (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
    format mended OF f:= p;
- s) PROC on value error =  
    (REF FILE f, PROC (REF FILE) BOOL p) VOID:  
    value error mended OF f:= p;
- t) PROC on char error =  
    (REF FILE f, PROC (REF FILE; REF CHAR) BOOL p) VOID:  
    char error mended OF f:= p;
- u) PROC reidf = (REF FILE f, STRING idf) VOID:  
    IF status OF f SAYS opened & reidf possible(f) & idf ok(idf)  
    THEN  
        C The identification string of the book of 'f' is made to be 'idf'. C  
    FI;

## 5. OPENING AND CLOSING FILES

### 5.1. DEVIATIONS

- {U} The numerous calls of 'undefined' in the Revised Report have been assigned meanings. Hidden kinds of undefined actions like SKIP, UP gremlins and UP bfileprotect have been paid due attention. At places where 'undefined' was called in the Revised Report, the present system issues an error message. In case no sensible continuation is possible, the elaboration of the particular program is aborted by means of a jump to the label 'abort'. Following this label, all buffers still need to be emptied, and all files are subsequently closed. The "gremlins" are not activated in the present system either; rather, opening a file is treated as a special kind of assignment.
- {U} If opening is not successful, non zero error codes are returned, and 'undefined' is not called.
- {B} The validity check for the parameters 'p', 'l' and 'c' in 'establish' is performed differently, since the operator BEYOND used in the Revised Report does not trap all erroneous combinations of 'p', 'l' and 'c'.
- {D} The routines 'close', 'lock' and 'scratch' are considered primitive actions.
- {E} Contrari to what is stated in the Revised Report, this model allows all upper bounds of the multiple value in 'associate' to be less than 1.

### 5.2. NEW DEFINITION

A book is "linked" with a file by means of 'establish' (a), 'create' (b) or 'open' (c). The linkage may be terminated by means of 'close' (l), 'lock' (m) or 'scratch' (n).

When a file is "established" on a channel, then a book is constructed (4.1.2.f) with a 'text' of the given size and the given identification string, that may be written to. The logical end of the book is set to (1, 1, 1). {An implementation may require (f) that the characters forming the identification string should be taken from a limited set and that the string should be limited in length. It may also prevent two books from having the same string.} If the establishing is completed successfully, then the value 0 is returned; otherwise, some non zero integer is returned, which indicates why the file was not established successfully. A list of these error codes is given below.

When a file is "created" on a channel, then a file is established with a book whose text has the default size for the channel, and whose identification string is a default identification string (4.1.2.h) for the implementation.

When a file is "opened", then the pool of available books is searched for a book with the following properties:

- The book may be identified by the given identification string;
- The book may be legitimately accessed through the given channel;

- Opening is not inhibited by other users of the book.

If such a book cannot be found, an appropriate error message is given and a non zero integer is returned. Otherwise, the file variable is initialized properly, and the value 0 is returned.

The routine 'associate' may be used to "associate" a file with a value of the mode specified by either REF CHAR, REF [] CHAR, REF [][] CHAR or REF [][][] CHAR, thus enabling such variables to be used as the book of a file. All lower bounds of the multiple value referred to must be 1, and all upper bounds should be at least 1. Note that the scope of the multiple value must not be newer than the scope of the given file 'f'.

For associated files, the buffer mechanism cannot be used: here it is necessary to transput directly to or from the associated multiple value, since the user also has direct access to it. It is for this reason that a separate reference to a 'text' is included in the file. The various primitives that have been incorporated in the channel also directly access the text in this case.

As a consequence of linking a book with a file, the fields of the file are initialized as follows:

- 'book'.

In 'establish' and 'create', a book is constructed (4.1.2.f) according to the requirements of the user. In 'open', the pool of available books is searched for a matching one (4.1.2.e). In 'associate', a pseudo book (4.1.2.g) is used;

- 'chan'.

In 'establish', 'create' and 'open', the channel is provided by the user. In 'associate', a channel value is constructed

- i) whose fields selected by 'reset', 'set', 'get', 'put' and 'backspace' always return true,
- ii) whose fields selected by 'bin', 'compress', 'reidf' and 'estab' always return false,
- iii) whose field selected by 'max pos' returns a value of mode POS whose fields selected by 'p', 'l' and 'c' return 'max int',
- iv) whose field selected by 'standconv' is such that transput takes place directly on the multiple value, without any intermediate conversion,
- v) whose fields selected by 'do newline', 'do newpage' and 'do reset' simply move the current position and update the status information accordingly,
- vi) whose field selected by 'do set' tests whether the user-supplied position is valid (possibly after calling the event routine corresponding to 'on logical file end'), and, if so, moves the current position to the supplied one, but if not, gives an appropriate error message, whereupon the elaboration of the particular program is aborted (see also 4.2.2.1.i),
- vii) whose field selected by 'channel number' is left undefined;

- 'plist'.

The field selected by 'plist' is assigned a nil name;

- 'buffer'.

As a result of calling 'establish' or 'create', a reference to the buffer is yielded by the routine that constructs the book; this buffer is initialized. When 'open' is called, the buffer is yielded by 'book in system' along with the book. It is not initialized; initialization has to await the first transput operation. The field 'buffer' is left

- undefined (and is not used) for associated files;
- 'text'.  
The field 'text' is only used for associated files. It then contains a reference to the associated multiple value; otherwise, it is left undefined;
  - 'cpos'.  
The current position is always initialized to (1, 1, 1);
  - 'c of lpos'.  
The 'c of lpos' field is linked up with the logical end of the book. If the logical end is in the current (i.e., 1st) line, 'c of lpos' points to the position where the logical end is. Otherwise it is left undefined. Whether the logical end is in the current line can be derived from the status information via the 'lfe in current line' flag. If a file is established or created, the logical file end is at (1, 1, 1) initially; when a file is opened it depends on the book and the read/write mood; in this case initialization has to await the first transput operation ('init buffer' will then take care of it). When a file is associated with a multiple value, the logical end is just beyond the extreme end of the associated multiple value;
  - 'term'.  
Initially, the terminator string is the empty string;
  - 'read buffer', 'write buffer', 'init buffer', 'put char', 'put bin char', 'get char', 'get bin char'.  
For associated files, these routines are built by the routine 'associate'. They do not make use of an intermediate buffer, nor do they perform any conversion. For the other opening routines they are copied from the given channel;
  - 'status'.  
In all cases, the initial information is such that the file is opened, and the line, page and physical file are not ended. The logical file end is at the current position if a file is established or created. So in that case the 'lfe in current line' flag is raised. {The 'logical file ended' flag is not raised since the write mood will be true.} For associated files they are both not raised, since in that case the text is assumed to have at least one page. If 'open' is called, then setting these flags has to await the first transput operation. The read mood is set to true if a file is linked to a book via 'open', and writing is not possible on the given channel; otherwise it is set to false. The write mood is set to true if a file is linked to a book via 'establish' or 'create', or via 'open' in case reading is not possible on the given channel; otherwise it is set to false. The char mood is set to true if a file is linked to a book, and binary transput is not possible on the given channel; otherwise it is set to false. Lastly, the bin mood is always initialized to false. The 'set poss' flag indicates whether the file is random access or not. It is incorporated in the status information to simplify some of the tests;
  - 'char bound'.  
The length of the current buffer differs from case to case:
    - it is given if 'establish' is called (the parameter 'c'),
    - it is the maximum buffer length of the given channel if 'create' is called;
    - it depends on the book if 'open' is called;
    - it depends on the associated multiple value if 'associate' is called (then it is equal to UPB sss[1][1]);
  - the 'on routines'.  
The 'on routines' are all initialized to routines returning false.

Files may be "closed", "locked" or "scratched". For associated files, these all just amount to updating the status information such that the file is no longer said to be opened. For files that are "opened", "created" or "established", the resulting action is different for each routine. If a file is closed, the book is put back into the pool of available books, so that the book may be re-opened at some later stage (e.g., if the channel corresponds to a tape-unit, the tape is not removed yet). If a book is locked, it cannot be re-opened until some subsequent system-task has put the book back into the pool (in this case, the tape is removed and put on the shelf). If a file is scratched, the book is disposed of by the system in some way (i.e., its information generally gets lost). In all cases, the current buffer may still have to be transput to the book, possibly resulting in an end-of-page or end-of-file condition and the like. The status information is updated such that the file is no longer said to be opened.

If opening is not successful, a non zero error code is returned. These error codes consist of two parts: a "general" part and a "specific" part. The general part can in a loose way be associated with the various validity tests; the specific parts depend on the particular operating environment, and they are not further specified in this report. (This strategy is borrowed from [11]). The general error codes used have been given the following symbolic names:

- (1) 'badidf' - The string argument to 'open' or 'establish' is wrong; in 'establish', the call of 'idf ok' returns false, in 'open' no matching book is found.
- (2) 'nowrite' - Writing cannot be done.
- (3) 'notavail' - 'file available' returns false. (This may be a temporary problem which can be handled by re-trying the calling of 'open' or 'establish' - for instance, there is no room in the directory system for a new file.)
- (4) 'noestab' - 'estab OF chan' returns false: files simply cannot be established on this channel.
- (5) 'posmax' - One of the dimension arguments to 'establish' or 'set' is too large.
- (6) 'posmin' - One of the dimension arguments to 'establish' or 'set' is too small (i.e.,  $\leq 0$ ).
- (7) 'notopen' - A transput routine is called with a file as parameter which is not opened.
- (8) 'noread' - Reading cannot be done.
- (9) 'noset' - An attempt is made to set a file for which 'set possible' returns false.
- (10) 'noreset' - An attempt is made to reset a file for which 'reset possible' returns false.
- (11) 'nobackspace' - An attempt is made to backspace on a file for which 'backspace possible' returns false.
- (12) 'noshift' - An attempt is made to shift from 'bin mood' to 'char mood' or vice versa on a sequential-access file.
- (13) 'nobin' - Binary transput is not possible.
- (14) 'noalter' - An attempt is made to shift from 'read mood' to 'write mood' or vice versa on sequential-access files in 'bin mood'.
- (15) 'nomood' - Transput is attempted while no mood has been set yet.
- (16) 'wrongmult' - 'associate' is called with a reference to a multiple value as parameter whose bounds are incorrect.



- (17) 'wrongset' - 'set' is called with a position as parameter which is beyond the logical end, and this situation is not mended by the user.
- (18) 'nocharpos' - No "good" character position can be ensured.
- (19) 'noline' - No "good" line can be ensured.
- (20) 'nopage' - No "good" page can be ensured.
- (21) 'wrongpos' - 'set char number' is called with a wrong character position as parameter.
- (22) 'wrongchar' - 'get' or 'getf' encounters a character which is not expected, or one which cannot be converted, and this situation is not mended by the user.
- (23) 'wrongval' - In 'get', it is impossible to convert a string to a value of some given mode, or during formatted transput an attempt is made to transput a value under the control of a picture with which it is incompatible, or whose number of frames is insufficient, and this situation is not mended by the user.
- (24) 'setmiddle' - An attempt is made to set a write-only file to a position which is not at the beginning of a line (see also section 7.1).
- (25) 'wrongbacksp' - Backspace is called with the current position at the beginning of a line.
- (26) 'smallline' - An attempt is made to output a number to a line which is too small to contain that number.
- (27) 'noformat' - 'putf' or 'getf' is called while no format is associated with the given file, or the user has not provided a new format while the event routine corresponding to 'on format end' returns true.
- (28) 'wrongformat' - During the elaboration of the first insertion of a 'collitem', the user has changed the current format of the file.
- (29) 'wrongbin' - During binary transput, it is not possible to input a value of some given mode.

If more than one validity test needs to be performed, they are performed one after the other; if one test fails, the corresponding error code is returned and the remaining tests are not performed.

Files can be categorized according to various different properties:

- a file may be random access or sequential;
- a file may be read-only, write-only, or both may be possible;
- a file may be used for both character and binary transput, or for character transput only.

Below, a state diagram is given in which the various possible changes in the mood are depicted for both sequential and random-access files (binary transput is supposed to be possible here).

	sequential		random-access	
	char mood	bin mood	char mood	bin mood
read only			↔	
write only			↔	
both possible/reading			↔	
both possible/writing	↑ ↓	↑ ↓	↑ ↓ ↔	↑ ↓
both possible/none	↑ ↓	↑ ↓	↑ ↓	↑ ↓

{There is one striking irregularity in the above picture: reading and writing may not be alternated on sequential-access files if the file is used for binary transput.}

{Note that arrows going downwards into the category "both possible/none" correspond to transitions caused by a call of 'reset'.}

a) PROC establish =

```

    (REF FILE f, STRING idf, CHANNEL chan, INT p, l, c) INT:
    IF NOT file available(chan) THEN error(notavail)
    ELIF NOT estab OF chan THEN error(noestab)
    ELIF POS(p, l, c) EXCEEDS max pos OF chan THEN error(posmax)
    ELIF POS(l, l, l) EXCEEDS POS(p, l, c) THEN error(posmin)
    ELIF NOT idf ok(idf) THEN error(badidf)
    ELSE
    REF BUFFER buffer;
    REF BOOK book:=
        construct book(p, l, c, idf, chan, buffer);
    IF NOT (put OF chan)(book) THEN error(nowrite)
    ELSE
    CONV cc = (standconv OF chan)(book);
    STATUS st:= establish status OR write mood;
    (NOT (bin OF chan)(book) | st ORAB char mood);
    (NOT (set OF chan)(book) | st ORAB not set poss);
    f:=
    (book, chan, REF REF FORMATLIST(NIL), buffer, SKIP,
    HEAP POS:= (l, l, l), HEAP INT:= l,
    STRINGTOTERM "",
    read buffer OF cc, write buffer OF cc, init buffer OF cc,
    put char OF cc, put bin char OF cc,
    get char OF cc, get bin char OF cc,
    HEAP BITS:= st,
        #i.e., opened & lfe in current (lst) line &
        the buffer is ready for writing #
    HEAP INT:= c,
    false, false, false, false, false, false,
    (REF FILE f, REF CHAR c) BOOL: FALSE);

```

```

FI
FI;

```

```

b) PROC create = (REF FILE f, CHANNEL chan) INT:
  BEGIN POS max pos = max pos OF chan;
    establish(f, default idf, chan, p OF max pos, l OF max pos,
      c OF max pos)
  END;

c) PROC open = (REF FILE f, STRING idf, CHANNEL chan) INT:
  IF NOT file available(chan) THEN error(notavail)
  ELIF REF BOOK book, REF BUFFER buffer;
    INT er = book in system(idf, chan, book, buffer);
    er ≠ 0
  THEN er
  ELSE CONV c = (standconv OF chan)(book);
    STATUS st:= open status;
    (NOT (put OF chan)(book) | st ORAB read mood);
    (NOT (get OF chan)(book) | st ORAB write mood);
    (NOT (bin OF chan)(book) | st ORAB char mood);
    (NOT (set OF chan)(book) | st ORAB not set poss);
    f:=
      (book, chan, REF REF FORMATLIST(NIL), buffer, SKIP,
        HEAP POS:= (1, 1, 1), HEAP INT:= SKIP,
        STRINGTOTERM "",
        read buffer OF c, write buffer OF c, init buffer OF c,
        put char OF c, put bin char OF c,
        get char OF c, get bin char OF c,
        HEAP BITS:= st,
        HEAP INT:= SKIP,
        false, false, false, false, false, false,
        (REF FILE f, REF CHAR c) BOOL: FALSE);
    0
  FI;

d) PROC associate = (REF FILE file, REF [][][] CHAR sss) VOID:
  IF INT lp = LWB sss, up = UPB sss;
    INT ll = (up > 0 | LWB sss[1] | 1),
      ul = (up > 0 | UPB sss[1] | 0);
    INT lc = (ul > 0 | LWB sss[1][1] | 1),
      uc = (ul > 0 | UPB sss[1][1] | 0);
    lp ≠ 1 OR ll ≠ 1 OR lc ≠ 1
  THEN error(wrongmult); abort
  ELSE STATUS st:= associate status OR char mood;
    ( up ≤ 0 | st ORAB (physical file end AND logical file ended)
    |: ul ≤ 0 | st ORAB page end
    |: uc ≤ 0 | st ORAB line end);

  PROC t = (REF BOOK a) BOOL: TRUE;
  PROC f = (REF BOOK a) BOOL: FALSE;

  PROC new buffer = (REF FILE f) VOID:
    # page is ok #
    (INT l = 1 OF cpos OF f += 1;
      c OF cpos OF f:= 1;
      IF l > UPB (text OF f)[1]

```

```

THEN status OF f ANDAB page end
FI);

CONV c = (
  # read buffer #
  new buffer,
  # write buffer #
  new buffer,
  # init buffer #
  (REF FILE f) VOID:
    REF STATUS(status OF f):= associate status OR char mood,
  # put char #
  (REF FILE f, CHAR char) VOID:
    BEGIN REF POS cpos = cpos OF f;
      (text OF f)[p OF cpos][1 OF cpos][c OF cpos]:= char;
      c OF cpos += 1; TRUE
    END,
  # put bin char #
  SKIP,
  # get char #
  (REF FILE f, REF CHAR char) BOOL:
    BEGIN REF POS cpos = cpos OF f;
      char:= (text OF f)[p OF cpos][1 OF cpos][c OF cpos];
      c OF cpos += 1; TRUE
    END,
  # get bin char #
  SKIP);

CHANNEL chan =
  (t, t, t, t, f, f, f, t, BOOL: FALSE,
  POS: (max int, max int, max int),
  # newline #
  new buffer,
  # newpage #
  (REF FILE f) VOID:
    (REF POS(cpos OF f):= (p OF cpos OF f + 1, 1, 1);
    STATUS reading = state(f);
    REF STATUS(status OF f):=
      IF p OF cpos OF f = UPB text OF f + 1
      THEN REF INT(c of lpos OF f):= 1;
        associate status AND physical file end AND
        logical file ended
      ELSE associate status
        FI OR reading),
  # reset #
  (REF FILE f) VOID:
    (REF POS(cpos OF f):= (1, 1, 1);
    REF INT(c of lpos OF f):= SKIP;
    REF STATUS(status OF f):= associate status
      OR char mood),

```

```

# set #
(REF FILE f, INT p, l, c) VOID:
  IF INT up = UPB text OF f + 1;
    POS lpos = (up, l, l), STATUS reading = state(f);
    POS(p, l, c) BEYOND lpos
  THEN REF POS(cpos OF f):= lpos;
    (reading SAYS read mood | test logical file end(f));
    BOOL mended = (logical file mended OF f)(f);
    ensure state(f, reading);
    (NOT mended | error(wrongset); abort)
  ELIF
    INT ul = UPB (text OF f)[1] + 1,
        uc = UPB (text OF f)[1][1] + 1;
    BOOL fne = p < up;
    BOOL pne = (fne | l < ul | FALSE);
    BOOL lne = (pne | c < uc | FALSE);
    BOOL lfe = NOT fne AND reading SAYS read mood;
    POS(p, l, c) EXCEEDS
      POS(up, (fne | ul | l), (pne | uc | l))
  THEN error(posmax); abort
  ELIF POS(l, l, l) EXCEEDS POS(p, l, c)
  THEN error(posmin); abort
  ELSE REF POS(cpos OF f):= (p, l, c);
    REF INT(c of lpos OF f):= (fne | SKIP | l);
    REF STATUS(status OF f):=
      bits pack((TRUE, TRUE, lne, pne, fne, lfe, fne,
                FALSE, FALSE, FALSE, FALSE, FALSE))
      OR reading
  FI,
(REF BOOK b) CONV: c,
SKIP);

file:=
(pseudo book(UPB sss), chan, REF REF FORMATLIST(NIL), SKIP, sss,
HEAP POS:= (l, l, l), HEAP INT:= SKIP,
STRINGTOTERM "",
read buffer OF c, write buffer OF c, init buffer OF c,
put char OF c, put bin char OF c,
get char OF c, get bin char OF c,
HEAP STATUS:= associate status OR char mood,
HEAP INT:= UPB sss[1][1],
false, false, false, false, false, false,
(REF FILE f, REF CHAR c) BOOL: FALSE)
FI;

```

- e) PROC ? file available = (CHANNEL chan) BOOL:  
 C true if another file, at this instant of time, may be "opened",  
 "established" or "created" on 'chan', and false otherwise C;
- f) PROC ? idf ok = (STRING idf) BOOL:  
 C true if 'idf' is acceptable to the implementation as the  
 'identification of a new book and false otherwise C;
- g) PROC ? false = (REF FILE f) BOOL: FALSE;

- h) PROC ? set write mood = (REF FILE f) VOID:  
 # opened and (in general) NOT write mood #  
 IF NOT (put OF chan OF f)(book OF f)  
 THEN error(nowrite); abort  
 ELIF status OF f SAYS read to write not possible  
 THEN error(noalter); abort  
 ELSE status OF f ANDAB NOT read mood;  
   status OF f ORAB (write mood OR logical pos ok)  
 FI # opened & write mood #;
- i) PROC ? set read mood = (REF FILE f) VOID:  
 # opened and (in general) NOT read mood #  
 IF NOT (get OF chan OF f)(book OF f)  
 THEN error(noread); abort  
 ELIF status OF f SAYS write to read not possible  
 THEN error(noalter); abort  
 ELSE status OF f ORAB read mood;  
   IF status OF f SAYS write mood  
   THEN status OF f ANDAB NOT write mood;  
     mind logical pos(f)  
 FI  
 FI # opened & read mood #;
- j) PROC ? set char mood = (REF FILE f) VOID:  
 # opened and (in general) NOT char mood #  
 IF status OF f SAYS bin to char not possible  
 THEN error(noshift); abort  
 ELSE status OF f ORAB char mood;  
   status OF f ANDAB NOT bin mood  
 FI # opened & char mood #;
- k) PROC ? set bin mood = (REF FILE f) VOID:  
 # opened and (in general) NOT bin mood #  
 IF NOT (bin OF chan OF f)(book OF f)  
 THEN error(nobin); abort  
 ELIF status OF f SAYS char to bin not possible  
 THEN error(noshift); abort  
 ELSE status OF f ORAB bin mood;  
   status OF f ANDAB NOT char mood  
 FI # opened & bin mood #;
- l) PROC close = (REF FILE f) VOID:  
 BEGIN  
   IF changed OF buffer OF f  
   THEN (write buffer OF f)(f)  
   FI;  
   status OF f ANDAB closed;  
   C The information on the number of users is updated. Some system-task  
     is activated to actually close the book; in this case, the book may  
     be re-opened. C  
 END;

m) PROC lock = (REF FILE f) VOID:

BEGIN

IF changed OF buffer OF f  
THEN (write buffer OF f)(f)

FI;

status OF f ANDAB closed;

C The information on the number of users is updated. Some system-task is activated to actually lock the book; in this case, it is not possible to re-open the book (except possibly after some subsequent system-task). C

END;

n) PROC scratch = (REF FILE f) VOID:

BEGIN

IF changed OF buffer OF f  
THEN (write buffer OF f)(f)

FI;

status OF f ANDAB closed;

C The information on the number of users is updated. The book is disposed of in some way by the system. C

END;

## 6. POSITION ENQUIRIES

### 6.1. DEVIATIONS

- {D} The present model makes no use of the routines 'current pos', 'book bounds', 'line ended', 'page ended', 'physical file ended' and 'logical file ended'. In the present model, the status of the file is inspected rather than its current position.

### 6.2. NEW DEFINITION

The current position of a book opened on a given file is the value referred to by the 'cpos' field of that file. It is advanced by each transput operation in accordance with the number of characters written or read.

If  $c$  is the current character number and  $lb$  is the (maximum) length of the current line, then at all times  $1 \leq c \leq lb+1$ .  $c=1$  implies that the next transput operation will be on the first position of the line, and  $c = lb+1$  implies that the line has overflowed and that the next transput operation will call an event routine. If  $lb = 0$ , then the line is empty and is therefore always in the overflowed state. Corresponding restrictions apply to the current line and page numbers. Note that, if the page has overflowed, the current line is empty, and, if the book has overflowed, the current page and line are both empty.

The user may determine the current position by means of the routines 'char number', 'line number' and 'page number' (a, b, c).

The "status" of a file contains the following information:

- whether or not the file has been opened;
- whether or not the buffer has been initialized;
- whether or not the line has overflowed;
- whether or not the page has overflowed;
- whether or not the physical file has overflowed;
- whether or not the logical file is ended;
- whether or not the logical file end is in the current line;
- whether or not the file is being used for input;
- whether or not the file is being used for output;
- whether or not the file is being used for character transput;
- whether or not the file is being used for binary transput;
- whether or not the file is random access.

In order to achieve an efficient implementation, the status is not defined as a set of separate booleans. Rather, the status is defined to be of the mode BITS. (On a machine where 'bits width' is less than 12, some trivial modifications have to be made in the definitions given below (d, e and f).)

As a consequence of a call of one of the routines 'set', 'reset', 'set char number' and 'backspace', the current position may be set back. Following the philosophy sketched in Chapter 2, the current position 'cpos' and the logical position 'lpos' are related through

$$\begin{aligned} \text{cpos} &= \text{cpos1} \\ \text{lpos} &= \text{MAX}(\text{cpos1}, \text{lpos1}), \end{aligned} \quad (*)$$



where 'cposl' and 'lposl' are the variables maintained by the system described in this report ('cposl' and 'lposl' are termed 'cpos' and 'lpos' in the text again). If one of the above mentioned routines is called, relation (\*) may no longer be valid if  $lposl < cposl$ . For this purpose, the routine 'mind logical pos' has been provided. Since the logical position is of importance when reading and writing is alternated, 'mind logical pos' is also called from within 'set read mood'.

The status is inspected before each transput operation. This inspection generally proceeds in two steps:

- i) an overall test (which depends on the transput operation); the "normal" situation will be detected by this test, so that transput may often be continued after one single test;
- ii) in case the overall test fails, a chain of tests is activated to detect the specific condition that fails to hold.

After each transput operation, the status of the file is updated. Routines are provided to update the 'line end' and 'logical file end' information. Updating 'buffer filled', 'page end', 'physical file end' and 'logical file end in current line' information obviously is one of the tasks of the routines 'read buffer', 'write buffer' and 'init buffer'.

a) PROC char number = (REF FILE f) INT:

```
IF status OF f SAYS opened
THEN c OF cpos OF f
ELSE error(notopen); abort
FI;
```

b) PROC line number = (REF FILE f) INT:

```
IF status OF f SAYS opened
THEN l OF cpos OF f
ELSE error(notopen); abort
FI;
```

c) PROC page number = (REF FILE f) INT:

```
IF status OF f SAYS opened
THEN p OF cpos OF f
ELSE error(notopen); abort
FI;
```

d) MODE ? STATUS = BITS;

# The bits in the status have the following meaning  
(they are numbered from left to right):

```
bit 1 = 1 <=> the file is opened;
bit 2 = 1 <=> the buffer is initialized;
bit 3 = 1 <=> NOT line ended;
bit 4 = 1 <=> NOT page ended;
bit 5 = 1 <=> NOT physical file ended;
bit 6 = 1 <=> NOT logical file ended;
bit 7 = 1 <=> NOT lfe in current line;
bit 8 = 1 <=> read mood;
bit 9 = 1 <=> write mood;
bit 10 = 1 <=> char mood;
bit 11 = 1 <=> bin mood;
bit 12 = 1 <=> NOT set possible. #
```

```

e) PRIO 7 ORAB = 1,
   OP ORAB = (REF STATUS s, STATUS t) REF STATUS: s:= s OR t;

   PRIO 7 ANDAB = 1,
   OP ANDAB = (REF STATUS s, STATUS t) REF STATUS: s:= s AND t;

   PRIO 7 SAYS = 6,
   OP SAYS = (STATUS s, t) BOOL: s  $\geq$  t;

   PRIO 7 SUGGESTS = 6,
   OP SUGGESTS = (STATUS s, t) BOOL: s  $\leq$  t;

```

{Sometimes one wants to know whether certain bits are on (then SAYS is used), sometimes whether they are off (then SUGGESTS is used).}

```

f) # Some constant-declarations. #
   STATUS 7 put char status      = 2r 1 0 000 00 0110 0,
   7 get char status            = 2r 1 0 000 00 1010 0,
   7 put bin status             = 2r 1 0 000 00 0101 0,
   7 get bin status             = 2r 1 0 000 00 1001 0,

   7 line ok                    = 2r 1 1 111 10 0000 0,
   7 page ok                    = 2r 1 0 011 10 0000 0,
   7 physical file ok          = 2r 1 0 001 10 0000 0,

   7 logical pos ok            = 2r 1 0 000 10 0000 0,
   7 logical pos not ok        = 2r 0 0 000 01 0000 1,
   7 logical file ended        = 2r 1 1 111 00 1111 1,

   7 opened                    = 2r 1 0 000 00 0000 0,
   7 closed                    = 2r 0 0 000 00 0000 0,
   7 buffer filled             = 2r 0 1 000 00 0000 0,
   7 not lfe in current line    = 2r 0 0 000 01 0000 0,
   7 lfe in current line       = 2r 1 1 111 10 1111 1,
   7 line end                  = 2r 1 1 011 11 1111 1,
   7 page end                  = 2r 1 1 001 11 1111 1,
   7 physical file end         = 2r 1 1 000 11 1111 1,
   7 associate end             = 2r 1 1 000 11 0000 1,

   7 not set poss.             = 2r 0 0 000 00 0000 1,

   7 establish status          = 2r 1 1 111 10 0000 0,
   7 open status               = 2r 1 0 111 11 0000 0,
   7 associate status          = 2r 1 1 111 11 0000 0,

   7 read mood                 = 2r 0 0 000 00 1000 0,
   7 write mood                = 2r 0 0 000 00 0100 0,
   7 char mood                 = 2r 0 0 000 00 0010 0,
   7 bin mood                  = 2r 0 0 000 00 0001 0,
   7 read or write mood        = 2r 0 0 000 00 1100 0,
   7 mood part                 = 2r 0 0 000 00 1111 0,

   7 read to write not possible = 2r 1 0 000 00 1001 1,
   7 write to read not possible = 2r 1 0 000 00 0101 1,
   7 bin to char not possible  = 2r 1 0 000 00 0001 1,
   7 char to bin not possible  = 2r 1 0 000 00 0010 1;

```

- g) PROC ? mind logical pos = (REF FILE f) VOID:  
IF status OF f SAYS not lfe in current line  
THEN SKIP  
ELSE  
IF c OF cpos OF f > c of lpos OF f  
THEN REF INT(c of lpos OF f):= c OF cpos OF f  
FI;  
test logical file end(f)  
FI;
- h) PROC ? test line end = (REF FILE f) VOID:  
IF c OF cpos OF f > char bound OF f  
THEN status OF f ANDAB line end  
FI;
- i) PROC ? test logical file end = (REF FILE f) VOID:  
IF c OF cpos OF f = c of lpos OF f  
THEN status OF f ANDAB logical file ended  
FI;

### 6.3. EFFICIENCY

The efficiency of the transput system critically depends on the way the operators SAYS and SUGGESTS are implemented. For instance, it will very often not be necessary for these operators to actually yield a value of the mode specified by BOOL. In case of an enquiry in an if-statement, the result of comparing two status values might be kept in a condition register, subsequently to be tested by that if-statement.

Also, the simple routines 'test line end' and 'test logical file end' may profitably be expanded inline.

## 7. LAYOUT ROUTINES

### 7.1. DEVIATIONS

- {S} The Revised Report specifies that 'space', 'newline' and 'newpage' act as skip operations if the logical file end is not yet reached (is not in the current line or page, respectively). This requirement will be difficult to fulfil on write-only files as far as 'space' is concerned. In the present model, blanks are written in that case.
- {S} A write-only file can only be set to a position which is at the beginning of a line (so 'c' equals 1); it is not expected that the system can start to write in the middle of a line that cannot be read (see also the description of 'do set' in section 4.2.2.1).
- {S} 'backspace' is not assumed to be possible on all files. For files having very long lines, the buffer may correspond to a smaller piece of information. In that case, 'backspace' is not allowed in the present model. The user may discover whether backspacing is possible through the environment enquiry 'backspace possible' (4.3.2.j). If 'backspace' is called with a file as parameter for which backspacing is not possible, an error message is given and the program is aborted.
- {D} The 'get good'-routines from the Revised Report are termed 'ensure'-routines in the present model. For clarity's sake, they are written (right-) recursively rather than with the aid of a while loop. {Of course, this can easily be optimized out.}
- {D} The routine 'get good file' in the Revised Report serves a twofold purpose: on reading, it is tested whether the current position is still within the logical file; on writing, the current position is checked against the physical file end. This task has been split up into two separate routines: 'ensure logical file' and 'ensure physical file'.
- {S} The routine 'check pos' as given below differs from the one given in the Revised Report in some obscure circumstances. The Working Party still has to decide on this matter.

### 7.2. NEW DEFINITION

A book input from an external medium by some system-task may contain lines and pages not all of the same length. Contrariwise, the lines and pages of a book which has been established (5.2.a) are all initially of the size specified by the user. However if, during output to a compressible book (4.3.2), 'newline' ('newpage') is called with the current position in the same line (page) as the logical end of the book, then that line (the page containing that line) is shortened to the character number (line number) of the logical end. Thus 'print(("abcde", newline))' could cause the current line to be reduced to 5 characters in length. Note that it is perfectly meaningful for a line to contain no characters and for a page to contain no lines.

The routines 'space' (a), 'newline' (c) and 'newpage' (d) serve to advance the current position to the next character, line or page, respectively. They do not alter the contents of the positions skipped over,

except during output with the current position at the logical end of the book.

If, during character output with the current position at the logical end of the book, 'space' is called, then a space character is written (similar action being taken in the case of 'newline' and 'newpage' if the book is not compressible). Note that on sequential-access files, 'space' is only treated as a skip operation if the logical end is in the current line! Thus, 'print(("a", backspace, space))' has a different effect from 'print(("a", backspace, blank))', while 'print(space)' may well have the same effect as 'print(blank)', even if the current position is not at the logical end of the book.

The current position may be altered also by calls of 'set char number' (o) and, on appropriate channels, of 'backspace' (b), 'set' (m) and 'reset' (n).

A call of 'set' which attempts to leave the current position beyond the logical end results in an error message, after which execution of the particular program is aborted. There is thus no defined way in which the current position can be made to be beyond the logical end, nor in which any character within the logical file can remain in its initial undefined state.

The mood of the file (4.3.2) controls some effects of the layout routines. If the read/write mood is reading, the effect of 'space', 'newline' and 'newpage', upon attempting to pass the logical end, is to call the event routine corresponding to 'on logical file end'; the default action then is to give an error message and abort the elaboration of the particular program. If the read/write mood is writing, the effect is to output spaces (or, in bin mood, to write some undefined character) or to compress the current line or page. If the read/write mood is not determined on entry to a layout routine, an error message is given and the program is aborted. On exit, the read/write mood present on entry is restored.

A reading or writing operation, or a call of 'space', 'newline', 'newpage', 'set' or 'set char number', may bring the current position outside the physical or logical file (6.2, 3), but this has no immediate consequence. However, before any further transport is attempted, or a further call of 'space', 'newline' or 'newpage' (but not of 'set' or 'set char number') is made, the current position must be brought to a "good" position. It is ensured that the position is "good" by calling one of the "ensure" routines: 'ensure logical file' (g), 'ensure physical file' (h), 'ensure page' (i) and 'ensure line' (j). On writing, the page is "good" if the (physical) file has not overflowed (6.2, 3); on reading, the page is "good" if the logical file has not overflowed (6.2, 3). The line (character position) is "good" if the page (line) has not overflowed (6.2, 3). The event routine corresponding to 'on physical file end', 'on logical file end', 'on page end' or 'on line end' is therefore called as appropriate. Except in the case of formatted transport (which makes use of the routine 'check pos', see below), the default action, if the event routine returns false, is to give an error message and stop execution of the program in the first two cases, or to call 'newpage' or 'newline', respectively. After this (or if the event routine returns true), the position is tested again (recursively).

The routines 'next pos' (k) and 'check pos' (l) also belong in the category of "ensure" routines. 'Next pos' is very similar to 'ensure line'; instead of returning false if the position can not be ensured, an error message is issued and the program is aborted. It is mainly used in the section on unformatted transput. The routine 'check pos' differs from 'ensure line' in that no default line mending is performed (although, by default, 'newpage' may well be called). It is used during unformatted input of strings or numbers and during formatted transput. An important characteristic of 'check pos' is that as soon as an event routine returns false, no other event routine is called.

Most routines in the transput section obey certain well-defined pre- and post conditions. The various conditions that may hold can be summarized as follows:

- opened: the file has been opened.
- mood ok: the file has been opened and the read/write mood is correct (in general, this means that the read/write mood is as on entry).
- physical file ok: the file has been opened, the read/write mood is correct, and the book has not overflowed (i.e., the page number is within its bounds).
- logical file ok: the file has been opened, the read/write mood is correct, and, on reading, the current position is within the logical file.
- page ok: the file has been opened, the read/write mood is correct, and both the book and the current page have not overflowed.
- line ok: the file has been opened, the read/write mood is correct, and the book, the current page and the current line have not overflowed (i.e., the current position is within the physical file).

{The following inclusions may be observed:

.line ok => page ok => physical file ok => mood ok => opened,

and

logical file ok => physical file ok.

}

In terms of the above conditions, the pre- and post-conditions (if the routine does not fail) of the ensure routines are:

	PRE	POST
ensure state	----	mood ok
ensure logical file	mood ok	logical file ok
ensure physical file	mood ok	physical file ok
ensure page	mood ok	page ok
ensure line	mood ok	line ok
next pos	mood ok	line ok
check pos	mood ok	line ok

```

a) PROC space = (REF FILE f) VOID:
  IF STATUS reading = state(f);
    IF status OF f SAYS line ok
    THEN TRUE
    ELSE ensure line(f, reading)
    FI
  THEN # line ok #
    REF INT c = c OF cpos OF f;
    IF status OF f SAYS not lfe in current line
    THEN c += 1; test line end(f)
    ELIF c of lpos OF f > c
    THEN c += 1; test line end(f); test logical file end(f)
    ELIF status OF f SAYS bin mood
    THEN (put bin char OF f)(f, SKIP)
    ELSE
      IF NOT (status OF f SAYS char mood)
      THEN set char mood(f)
      FI;
      put char(f, " ")
    FI
  ELSE error(nocharpos); abort
  FI;

b) PROC backspace = (REF FILE f) VOID:
  IF NOT backspace possible(f)
  THEN error(nobackspace); abort
  ELSE mind logical pos(f);
  REF INT c = c OF cpos OF f;
  (c > 1 | c -= 1 | error(wrongbacksp); abort);
  status OF f ORAB line ok # logical file ok & NOT line ended #
  FI;

```



```

c) PROC newline = (REF FILE f) VOID:
  IF STATUS reading = state(f);
    IF status OF f SAYS page ok
      THEN TRUE
    ELSE ensure page(f, reading)
    FI
  THEN # page ok #
    (do newline OF chan OF f)(f)
  ELSE error(noline); abort
  FI;

d) PROC newpage = (REF FILE f) VOID:
  IF STATUS reading = state(f);
    IF status OF f SAYS physical file ok
      THEN TRUE
    ELSE ensure physical file(f, reading)
    FI
  THEN # physical file ok #
    (do newpage OF chan OF f)(f)
  ELSE error(nopage); abort
  FI;

e) PROC ? state = (REF FILE f) STATUS:
  IF NOT (status OF f SAYS opened)
    THEN error(notopen); abort
  ELIF status OF f SUGGESTS NOT read or write mood
    THEN error(nomood); abort
  ELSE status OF f AND mood part
  FI;

f) PROC ? ensure state = (REF FILE f, STATUS reading) VOID:
  IF NOT (status OF f SAYS opened)
    THEN error(notopen); abort
  ELSE
    IF reading SAYS read mood
      THEN set read mood(f)
    ELSE set write mood(f)
    FI;
    IF reading SAYS char mood
      THEN set char mood(f)
    ELSE set bin mood(f)
    FI
  FI;

g) PROC ? ensure logical file = (REF FILE f, STATUS reading) BOOL:
  BEGIN # logical file ended #
    BOOL mended = (logical file mended OF f)(f);
    ensure state(f, reading);
    IF status OF f SAYS logical pos ok
      THEN TRUE
    ELIF mended
      THEN ensure logical file(f, reading)
    ELSE FALSE
    FI
  END;

```

h) PROC ? ensure physical file = (REF FILE f, STATUS reading) BOOL:  
 # The mood is correct, the file generally not #

```

IF
  IF status OF f SAYS logical pos ok
    OR reading SAYS write mood
  THEN TRUE
  ELSE ensure logical file(f, reading)
  FI
THEN # logical file ok #
  IF status OF f SAYS physical file ok
  THEN TRUE
  ELSE # physical file ended #
    BOOL mended = (physical file mended OF f)(f);
    ensure state(f, reading);
    IF mended
      THEN ensure physical file(f, reading)
      ELSE error(nopage); abort
    FI
  FI
ELSE FALSE
FI;
```

i) PROC ? ensure page = (REF FILE f, STATUS reading) BOOL:  
 # The mood is ok, the page generally not #

```

IF
  IF status OF f SAYS physical file ok
  THEN TRUE
  ELSE ensure physical file(f, reading)
  FI
THEN # physical file ok #
  IF status OF f SAYS page ok
  THEN TRUE
  ELSE # page ended #
    BOOL mended = (page mended OF f)(f);
    ensure state(f, reading); (NOT mended | newpage(f));
    ensure page(f, reading)
  FI
ELSE FALSE
FI;
```

j) PROC ? ensure line = (REF FILE f, STATUS reading) BOOL:  
 # The mood is ok, the line generally not #

```

IF
  IF status OF f SAYS page ok
  THEN TRUE
  ELSE ensure page(f, reading)
  FI
THEN # page ok #
  IF NOT (status OF f SAYS buffer filled)
  THEN (init buffer OF f)(f); ensure line(f, reading)
  ELIF status OF f SAYS line ok
  THEN TRUE
  ELSE # line ended #
    BOOL mended = (line mended OF f)(f);
    ensure state(f, reading); (NOT mended | newline(f));
    ensure line(f, reading)
  FI
FI;
```

```

    FI
  ELSE FALSE
  FI;

```

```

k) PROC ? next pos = (REF FILE f) VOID:
  IF NOT ensure line(f, status OF f)
  THEN error(nocharpos); abort
  FI;

```

```

l) PROC ? check pos = (REF FILE f) BOOL:
  # the mood is ok, the line generally not #
  IF STATUS reading = status OF f;
  IF status OF f SAYS page ok
  THEN TRUE
  ELSE ensure page(f, reading)
  FI
  THEN # page ok #
  IF NOT (status OF f SAYS buffer filled)
  THEN (init buffer OF f)(f); check pos(f)
  ELIF status OF f SAYS line ok
  THEN TRUE
  ELSE # line ended #
    BOOL mended = (line mended OF f)(f);
    ensure state(f, reading);
    (mended | check pos(f) | FALSE)
  FI
  ELSE FALSE
  FI;

```

```

m) PROC set = (REF FILE f, INT p, l, c) VOID:
  IF NOT (status OF f SAYS opened)
  THEN error(notopen); abort
  ELIF status OF f SAYS not set poss
  THEN error(noset); abort
  ELSE mind logical pos(f);
    (do set OF chan OF f)(f, p, l, c)
  FI;

```

```

n) PROC reset = (REF FILE f) VOID:
  IF NOT reset possible(f)
  THEN error(noreset); abort
  ELSE mind logical pos(f);
    REF STATUS st = status OF f;
    st ANDAB NOT mood part;
    (NOT (put OF chan OF f)(book OF f) | st ORAB read mood);
    (NOT (get OF chan OF f)(book OF f) | st ORAB write mood);
    (NOT (bin OF chan OF f)(book OF f) | st ORAB char mood);
    (do reset OF chan OF f)(f)
  FI;

```

```
o) PROC set char number = (REF FILE f, INT c) VOID:
  IF NOT (status OF f SAYS opened)
  THEN error(notopen); abort
  ELIF c < 1 OR c > char bound OF f + 1
  THEN error(wrongpos); abort
  ELSE
    WHILE c OF cpos OF f ≠ c
    DO
      IF c > c OF cpos OF f
      THEN space(f)
      ELSE backspace(f)
      FI
    OD
  FI;
```

## 8. CONVERSION ROUTINES

### 8.1. DIFFERENCES

- {U} In the set of conversion routines given in the Revised Report, it is not always clear exactly when 'undefined' is called. It seems to be the intention to call 'undefined' only when, whatever the value of the first parameter, no string can be delivered satisfying the constraints set by the other parameters. However, when 'x' and 'i' are of the mode REAL and INT, respectively, 'whole(x, 1)' calls 'undefined', while 'whole(i, 1)' does not. Similarly, 'float' may call 'undefined' due to its recursive description. In the present model, 'undefined' is never called; rather, a string filled with 'errorchar' is just returned.
- {D} No use has been made of the routine 'L standardize'. In general, the number of places where real arithmetic comes in is kept minimal: only the routines 'subfixed' and 'log10', and a few lines in 'string to L real' use real arithmetic.
- {D} The routine 'char dig' does not replace spaces by zeroes, since 'indit string' caters for that.

### 8.2. NEW DEFINITION

{An earlier version of this Chapter appeared in the ALGOL Bulletin [15].}

From the routines 'whole', 'fixed' and 'float' given below, the following may be observed:

- The routines do not make use of real arithmetic. All real arithmetic is delegated to 'subfixed' and 'log10'. Unless the exponent in 'float' may be of the order of magnitude of 'max int', the integral arithmetic presents no trouble;
- Arithmetic values are first converted to strings of sufficient length, after which the rounding is performed on the strings. This seems to be the only reasonable way to ensure that values like 'L max real' may be converted using 'fixed' or 'float';
- The routines are written non-recursively;
- Care has been taken to avoid string operations. (Building up a string of n characters one by one using the standard operations PLUSTO and PLUSAB will probably result in allocation of about  $(n^2)/2$  storage cells, most of which are garbage.) Therefore, the routines start by establishing a reasonable upperbound on the width of the converted value, and a reference-to-row-of-character of that length is used thereafter.

The routines 'whole', 'fixed' and 'float' have the following parameters:

- 'v', the value to be converted,
- 'width', whose absolute value specifies the length of the string that is produced,
- 'after', whose value specifies the number of digits desired after the decimal point (for 'fixed' and 'float' only), and
- 'exp', whose absolute value specifies the desired width of the exponent (for 'float' only).

The routine 'whole' is intended to convert integer values. Leading zeroes are replaced by spaces and a sign is normally included. The user may specify that a sign is to be included only for negative values by specifying a negative or zero width. If the width specified is zero, then the shortest possible string is returned. The routine uses 'subwhole' for the actual conversion.

Examples:

'whole(i, -4)' might yield " 0", " 99", " -99", "9999", or, if 'i' were greater than 9999, "\*\*\*\*", where "\*" is the yield of 'errorchar';  
 'whole(i, 4)' would yield " +99" rather than " 99";  
 'whole(i, 0)' might yield "0", "99", "-99", "9999" or "99999".

The routine 'fixed' may be used to convert real values to fixed point form (i.e., without an exponent part). It has an 'after' parameter to specify the number of digits required after the decimal point. From the value of the 'width' and 'after' parameter, the amount of space left in front of the decimal point may be calculated. If the space left in front of the decimal point is not sufficient to contain the integral part of the value being converted, digits after the decimal point are sacrificed. If the number of digits after the decimal point is reduced to zero and the value still does not fit, a string filled with 'errorchar' is returned. The routine uses 'subfixed' for the actual conversion, and 'log10' to estimate the width of the integral part of the value submitted.

Examples:

'fixed(x, -6, 3)' might yield " 2.718", "27.183", "271.83" (in which one place after the decimal point has been sacrificed in order to fit the number in), "2718.3", " 27183" or "271828" (in the last two examples, all positions after the decimal point have been sacrificed);  
 'fixed(x, 0, 3)' might yield "2.718", "27.183" or "271.828".

The routine 'float' may be used to convert real values into floating point form. It has an 'exp' parameter to specify the width of the exponent. A sign is normally included in both the mantissa and the exponent. Just as in the case of the 'width' parameter, the sign of the 'exp' parameter specifies whether or not a plus sign is to be included. If the value of the 'exp' parameter is zero, the exponent is converted to a string of minimal length. The value of the 'width' parameter, however, may not be zero in this case. Note that 'float' always leaves the first position for a sign; if no sign is to be included, a space will be given. From the value of 'width', 'after' and 'exp' it follows how much space is left in front of the decimal point. From this, the value of the exponent follows; this exponent has to fit in a string whose length is bounded by the width specified by the 'exp' parameter. If this is not possible, the digits after the decimal point are sacrificed one by one; if there are no more digits left after the decimal point and the exponent still does not fit, digits in front of the decimal point are sacrificed too. (Note that this has repercussions on the value of the exponent, and thus possibly on the width of the exponent.) At each step during this process, a check must be made to ensure that some positions for digits remain; if this is not the case, a string filled with 'errorchar' is returned. The routine uses 'subfixed' for the actual conversion, and the operator 'EXPLENGTH' to determine the width needed for the exponent.

Examples:

'float(x, 9, 3, 2)' might yield "-2.718e+0", "+2.72e+11" (in which one place after the decimal point has been sacrificed in order to make room for the exponent);

'float(x, 6, 1, 0)' might yield "-256e1", "+26e12" or "+1e -9" (in case 'x' has the value 0.996e-9).

The routine 'subfixed' performs the actual conversion from numbers to strings. The parameter 's' receives the representation of the value submitted; the digits are placed from position 1 onwards. Position 0 is always filled with the character "0", while the positions numbered 'LWB s' up to -1 are filled with spaces. When called from 'fixed', the routine delivers all digits from the integral part, followed by ".", followed by 'after+1' digits from the fractional part. When called from 'float', the first 'after+1' significant digits are delivered and the decimal point is placed after the i-th digit, where i is the initial value of the parameter 'p'. In both cases, the last digit is truncated, and not rounded. (The rounding is done later on, and rounding the number twice may give wrong results.) The parameter 'p' receives the position of the decimal point, where the position to the right of the first digit is counted positive and to the left zero or negative. The routine returns an indication of the sign of the value submitted (true iff negative).

The routine 'subfixed' should to the best possible adhere to the principles of numerical analysis. Rather than an ALGOL 68 routine, a semantic definition is given below.

The (hidden) routines 'power10' and 'round' are used for rounding. For 'power10', the parameter 's' refers to the string to be rounded, the parameter 'last' is the position where the rounding process starts, and the parameter 'rp' will obtain as value the index P in 's' such that 's[P+1]' ... 's[last]' are all equal to "9" (or "."). The routine delivers true if the rounding causes a carry out of the leftmost digit of the number in 's'. At a subsequent stage, 'round' is called with the same parameters 's', 'rp' and 'last'. This routine performs the actual rounding. If 'rp' > 'last', no action is performed since the number may just be truncated at the position indicated by 'last'. Otherwise, 's[rp]' gets as value the character representing the next higher digit, and 's[rp+1]' ... 's[last]' all get the value "0" (except at the position of the decimal point). The rounding is done in two steps, since 'float' may necessitate several calls of 'power10' before the actual rounding may take place.

```

a) MODE 7 NUMBER = UNION({L REAL}, {L INT});

b) PROC whole = (NUMBER v, INT width) STRING:
CASE v IN
  {L INT x):
    (INT abs width = ABS width;
     INT upb = abs width MAX L int width;
     INT lwb:= upb - abs width + 1;
     INT first:= lwb, [0 : upb] CHAR s;
     IF BOOL neg = subwhole(x, first, upb, s); neg OR width > 0
     THEN s[first -:= 1]:= (neg | "-" | "+")
     FI;
     (width = 0 | lwb:= first); # no leading spaces needed #
     IF first ≥ lwb THEN s[lwb : ]
       ELSE abs width * errorchar
     FI);
OUT fixed(x, width, 0)
ESAC;

c) PROC fixed = (NUMBER v, INT width, after) STRING:
IF INT abs width = ABS width,
  BOOL poswidth = width > 0, zerowidth = width = 0;
  INT point:= log10(v) - 1, length:= abs width - ABS poswidth;
  after < 0 OR
  NOT zerowidth AND (after ≥ length OR point > length)
THEN 1 MAX abs width * errorchar
ELIF INT aft = (zerowidth | after | after MIN (length - point));
  # Now aft equals the maximum number of digits delivered after the
  decimal point. #
  INT upb = point + aft + 3;
  [0 MIN (upb - abs width) - 2 : upb] CHAR s;
  BOOL neg = subfixed(v, aft, point, s, FALSE);
  INT last:= point + aft, first:= 1, rp;
  IF zerowidth THEN length:= last
  ELSE length:= abs width - ABS(neg OR poswidth);
    last:= last MIN length
  FI;
  IF power10(s, rp, last)
  THEN first:= 0;
    (NOT zerowidth AND last = length | last -:= 1)
  FI;
  (last = point | last -:= 1);
  # The result should not end with ".". #
  (point = 1 AND last < length | first:= 0);
  # Deliver "0" or "0.xxx". #
  point > last + 1 OR last < first
  THEN abs width * errorchar
  ELSE s[first - 1]:= (neg | "-" | poswidth | "+" | " ");
    round(s, rp, last);
    s[(zerowidth | first - ABS neg | last - abs width + 1) : last]
  FI;

```



```

d) PROC float = (NUMBER v, INT width, after, exp) STRING:
  IF INT abs width = ABS width, sign after = SIGN after,
    INT exp places:= ABS exp;
    INT last:= abs width - exp places - 2;
    INT before:= last - after - sign after;
    SIGN before + sign after <= 0
  THEN | MAX abs width * errorchar
  ELIF INT first:= 1, exponent:= before, rp:= last + 1;
    [-1 : abs width - sign after] CHAR s, BOOL exp sign = exp > 0;
    BOOL neg:= subfixed(v, after, exponent, s, TRUE);
    exponent -= before;
    WHILE
      IF rp > last
        THEN
          IF power10(s, rp, last)
            THEN first:= 0; s[before]:= ".";
              s[before + 1]:= (before = 0 | rp:= 1; "0" | "9");
              before -= 1; last -= 1; exponent += 1
            FI
          FI;
          exp sign EXPLENGTH exponent > exp places AND last >= first
        DO last -= 1; exp places += 1;
          CASE SIGN(last - before - 1) + 2 IN
            (before -= 1; exponent += 1),
            (before += 1; exponent -= 1);
            REF CHAR sb1 = s[before + 1]; s[before]:= sb1; sb1:= "."
          ESAC
        OD;
        last < first
      THEN abs width * errorchar
    ELSE round(s, rp, last);
      s[first -= 1]:= (neg | "-" | width > 0 | "+" | ".");
      s[last += 1]:= "e"; INT l = last + exp places;
      neg:= subwhole(exponent, last += 1, l, s);
      (neg OR exp sign | s[last - 1]:= (neg | "-" | "+"));
      s[first : l]
    FI;
  FI;

```

```

e) PROC ? subwhole =
  (UNION({L INT} x, REF INT first, INT upb, REF [] CHAR s) BOOL:
    # The digits of 'x' are placed in 's' (right justified to the
    position with index 'upb'); the result will be a boolean indicating
    the sign of 'x'. As a result, 'first' will point to the first digit
    of 'x' in 's'. Leading spaces will be placed from position F
    onwards, where F is the initial value of 'first'. #
  CASE x IN
    {L INT x}:
      BEGIN L INT n:= ABS x, INT f = first; first:= upb + 1;
        WHILE s[first -= 1]:= dig char(S(n MOD L 10));
          n OVERAB L 10; n <= L 0
        DO SKIP OD;
        FOR i FROM f TO first - 1 DO s[i]:= "." OD;
        x < 0
      END}
  ESAC;

```

f) PROC 7 subfixed =

(NUMBER v, INT after, REF INT p, REF [] CHAR s, BOOL floating) BOOL:  
 C A unit which, given values V, AFTER and FLOATING (where AFTER is at least zero), yields a value B and makes 'p' and 's' refer to values P and S, respectively, such that:

- B is true if V is negative, and false otherwise;
- Case A: FLOATING is false:
  - it maximizes

$$M = \sum_{i=1}^{P-1} c_i * 10^{P-1-i} + \sum_{i=p+1}^U c_i * 10^{P-i}$$

under the following constraints:

- If  $V \geq 1.0$ , then  $P = \text{ENTIER}(\log_{10}(V)) + 2$ , and, otherwise,  $P=1$ ;
- $U = P + \text{AFTER} + 1$ ;
- $M \leq |V|$ .

{This simply amounts to the following: 's' contains all digits from the fractional part of V at positions 1 to P-1, a decimal point at position P, and 'after+1' digits from the fractional part of V at positions P+1 to U. Furthermore, if we convert the contents of V back to some value M, this M should be "close" (in the sense of numerical analysis) to the original value V.}

Case B: FLOATING is true:

- it maximizes

$$M = \sum_{i=1}^Q c_i * 10^{Q-i} + \sum_{i=Q+2}^U c_i * 10^{Q+1-i}$$

under the following constraints:

- Q is the initial value of 'p' {i.e., the number of digits desired before the decimal point};
- If  $V = 0$ , then  $P = 1$ , and, otherwise,  $P = \text{ENTIER}(\log_{10}(V)) + 1$  {i.e.,  $V = (\text{appr.}) 10^P * x$ , where x is normalized such that  $0.1 \leq x < 1.0$ };
- $U = Q + \text{after} + 2$ ;
- $M \leq |V| * 10^{Q-P}$ .

{Thus, 's' contains the first Q significant digits of V, followed by a decimal point, followed by the next 'after+1' digits of V. If the contents of 's' is converted back to some value M, this M should be close to the (scaled) original value V.}

- For all i from 1 to U, if  $s[i] \neq "."$ , then  $0 \leq c_i \leq 9$ , where  $c_i = \text{'char dig}(s[i])'$ .

C;

g) PROC ? log10 = (NUMBER v) INT:

C A number P such that  $P = 0$  if  $v < 1.0$ , and, if  $v \geq 1.0$ , then P is such that

$$\text{ENTIER}(\log_{10}(v)) + 1 \leq P \leq \text{ENTIER}(\log_{10}(v)) + 2.$$

Thus, P is an estimate of the number of digits in the fractional part of 'v'; this estimate may be at most 1 too large. This definition should allow efficient computation of P; e.g., using the normal floating point representation of 'v' with a mantissa M and base 2

exponent E ( $E = \text{ENTIER}(\log_2 v)$ ), P may be given the value

$\max(\text{ENTIER}(\log_{10} 2 * (E+1))+1, 0)$ . Obviously also, 'log10' is a typical candidate for inline expansion. C;

h) PROC ? EXPLENGTH = 9;

OP EXPLENGTH = (BOOL sign, INT exp) INT:

C The smallest E such that 'whole(exp, ABS sign \* E)' succeeds. This operator is used to estimate the length needed to convert the exponent in 'float'. This is probably easier and faster than actually converting the exponent and subsequently testing its width, since on most implementations the exponent will be a relatively small integer ( $\leq 322$ , say). C;

i) PROC ? power10 = (REF [] CHAR s, REF INT rp, INT last) BOOL:

IF rp := last + 1; CHAR c := s[rp];  
char dig((c = "." | s[rp+1] | c))  $\geq 5$

THEN

WHILE c := s[rp -:= 1]; c = "9" OR c = "." DO SKIP OD;

rp = 0

ELSE FALSE

FI;

j) PROC ? round = (REF [] CHAR s, INT rp, last) VOID:

IF rp  $\leq$  last

THEN REF CHAR srp = sr[p]; srp := dig char(char dig(srp)+1);

FOR i FROM rp + 1 TO last

DO REF CHAR si = s[i]; (si  $\neq$  "." | si := "0") OD

FI;

k) PROC ? dig char = (INT x) CHAR:

"0123456789abcdef"[x+1];

l) PROC ? MAX = 9;

OP MAX = (INT a, b) INT: (a > b | a | b);

m) PROC ? MIN = 9;

OP MIN = (INT a, b) INT: (a < b | a | b);

Strings are converted to arithmetic values by the operator 'ADD' and the routine 'string to L real'. 'ADD' is used to construct an integer value. Since getting an integer is relatively simple, no intermediate string is constructed; rather, each digit read is directly appended to the partially constructed number. For real values, a string S is first built which contains the (at most) 'L real width+1' most significant digits. Subsequently, an integer exponent EXP is computed such that, if  $d_1, \dots, d_n$  are the digits corresponding to the elements of S, then the number

$$d_1 d_2 \dots d_n * 10^{\text{EXP}}$$

is a close approximation of (the absolute value of) the number read. This construction is performed by 'string to L real', which also gets the sign of the result as parameter. Obviously, a semantic definition of this routine is given below.

n) PROC ? ADD = 1;

```
OP ADD = (REF L INT a, INT d) BOOL:
  IF L INT amax = L max int OVER L 10,
    dmax = L max int MOD L 10;
    a > amax OR a = amax AND K d > dmax
  THEN FALSE
  ELSE a := L 10 * a + K d; TRUE
  FI;
```

o) PROC string to L real =

```
(REF [] CHAR s, INT exp, BOOL neg, REF L REAL x) BOOL:
  C A unit which, given values S, EXP and NEG, yields a value B such
  that:
```

• Let M be equal to  $\left( \sum_{i=\text{LWB } s}^{\text{UPB } s} c_i * 10^{\text{UPB } s - i} \right) * 10^{\text{EXP}}$

• IF  $M \leq L \text{ max real}$ , then:

• 'x' is made to refer to a value X, where  $X = M * (\text{neg} \mid -1 \mid 1)$ ;

• B is true;

Otherwise,

• B is false {, and 'x' is left unchanged}.

C;

p) PROC ? char dig = (CHAR x) INT:

```
(INT i; char in string(x, i, "0123456789abcdef"); i-1);
```

q) PROC char in string = (CHAR c, REF INT i, STRING s) BOOL:

```
(BOOL found:= FALSE;
  FOR k FROM LWB s TO UPB s WHILE NOT found
  DO (c = s[k] | i:= k; found:= TRUE) OD;
  found);
```

r) INT L int width =

# the smallest integral value such that 'L max int' may be converted without error using the pattern  $n(L \text{ int width})d \#$

```
(INT c:= 1;
  WHILE L 10 ** (c-1) < L max int OVER L 10 DO c += 1 OD;
  c);
```

s) INT L real width =

C the smallest integral value such that different values yield different strings using the pattern  $d.n(L \text{ real width} - 1)d C$ ;

t) INT L exp width =

C the smallest integral value such that 'L max real' may be converted without error using the pattern  $d.n(L \text{ real width} - 1)d e n(L \text{ exp width})d C$ ;

### 8.3. EFFICIENCY

The conversion routines are likely to be used quite heavily. Therefore, the efficiency of their implementation is of crucial importance. Care has already been taken to make the ALGOL-68 text as efficient as possible. Most notably, all string processing has been avoided. The following machine-dependent optimizations that will speed up the code considerably, are recommended (although it may in practice turn out to be advantageous to wholly rewrite the section on conversion routines in machine-code):

- i) Since the character arrays that are used are simple, one-dimensional arrays with stride 1, an efficient and simple array-subscripting mechanism can be used on most machines.
- ii) The routines 'dig char' and 'char.dig' can often be implemented much more efficiently using knowledge about the internal ordering of the characters.

## 9. TRANSPUT MODES AND STRAIGHTENING

## 9.1. DEVIATIONS

- {B} The way INTYPE is defined in the Revised Report, 'flexible row of character' may only occur immediately after 'reference to', and may not be stowed. Thus, names of the mode specified by REF STRUCT(BOOL b, STRING s), say, cannot be input to. This restriction is very much against the way in which stowing is normally handled; also, straightening is well capable of handling this task. This unintended restriction has been removed.

## 9.2. NEW DEFINITION

- a) MODE ? SIMPLOUT = UNION(† L INT †, † L REAL †, † L COMPL †, BOOL, † L BITS †, CHAR, [] CHAR);
  - b) {Here, uppercase stands for metanotions.}
    - OUTTYPE:: union of OUTTYPER mode.
    - OUTTYPER:: OUTTYPER; OUTTYPER OUTTYPER.
    - OUTTYPER:: PLAIN;
      - structured with OUTTAGS mode;
      - ROWS of OUTTYPER.
    - OUTTAGS:: OUTTYPER field TAG; OUTTYPER field TAG OUTTAGS.
  - c) MODE ? SIMPLIN = UNION(† REF L INT †, † REF L REAL †, † REF L COMPL †, REF BOOL, † REF L BITS †, REF CHAR, REF [] CHAR, REF STRING);
  - d) INTYPE:: union OF INTYPERS mode.
    - INTYPERS:: reference to INTYPER; reference to INTYPER INTYPERS.
    - INTYPER:: PLAIN;
      - flexible row of character;
      - structured with INTAGS mode;
      - ROWS of INTYPER.
    - INTAGS:: INTYPER field TAG; INTYPER field TAG INTAGS.
  - e) OP ? STRAIGHTOUT = (OUTTYPE x) [] SIMPLOUT:
    - C the result of "straightening" 'x' C;
  - f) OP ? STRAIGHTIN = (INTYPE x) [] SIMPLIN:
    - C the result of "straightening" 'x' C;
- {Straightening is defined in the Revised Report in 10.3.2.3.c,d.}

## 10. FORMATLESS TRANSPUT

### 10.1. DIFFERENCES

- {S} In the Revised Report, the routines 'put' and 'get' start with a test for the file being opened. This test is only needed for the case where the second parameter is an empty row, as in 'put(f, ())'. In all other cases, the file is tested each time around the loop. In the present model, this test has been omitted; it can now be stated that the test for the file being opened is performed only once for each item to be output except when control is given back to the user in between.
- {S} In the Revised Report, an empty string written at the end of a page cannot be read back. To achieve compatibility between getting and putting strings, 'ensure page' is called explicitly before a row of character is output. Note however that this introduces an incompatibility between putting and getting rows of characters! Therefore, 'ensure page' is also called before a row of characters is input using 'get'. This makes no difference as long as the row of characters is not empty; for an empty row, a good page will be found, however.
- {S} In the present model it is assumed that internal characters can always be converted to external ones. Thus, 'char error mended' is never called by 'put char'. As a consequence, the test for the file being opened and the current position being good can be omitted from 'put char'.
- {S,E} As the present model does not assume that backspacing is possible on each file, the character that may have been read ahead by 'get' must be restored in a different way. A primitive 'back char' is introduced for this purpose.
- {S} If a complex number is read, no value is assigned to the complex variable if either conversion to a real number fails. In the Revised Report, a value may in that case be assigned to one of the subnames of the complex variable.

### 10.2. NEW DEFINITION

In formatless transput, the elements of a "data list" are transput, one after the other, via the specified file. Each element of the data list is either a routine of the mode specified by PROC (REF FILE) VOID or a value of the mode specified by OUTTYPE (on output) or INTYPE (on input). On encountering a routine in the data list, that routine is called with the specified file as parameter. Other values in the data list are first straightened (9.2) and the resulting values are then transput via the given file one after the other.

Transput normally takes place at the current position but, if there is no room on the current line, then first, the event routine corresponding to 'on line end' (or, where appropriate, to 'on page end' or 'on physical file end' or 'on logical file end') is called, and next, if this returns false, the next "good" character position of the book is found, viz., the first character position of the next nonempty line.



For formatless output, 'put' (a) and 'print' (or 'write') (section 10.5.1 of the Revised Report) may be used. Each straightened value V from the data list is output as follows:

If the mode of V is specified by L INT, L REAL or L COMPL, output has to fit on one and the same line. Moreover, if output does not take place at the beginning of a line, a space is given first. The length of the string that is output is such that 'L max int' ('L max real', ('L max real', 'L max real')) is output without error if the mode of V is specified by L INT (L REAL, L COMPL). So, if the current position is at the beginning of a line, the length of these strings is:

L int width + 1,  
 L real width + L exp width + 4, or  
 2 \* (L real width + L exp width + 4) + 2,

respectively, and one more otherwise. If the length of the string happens to be greater than the length of the current line (i.e., the string would not fit even if the line were empty), an error message is given and the program is aborted. Otherwise, if there is not enough room for a string of this length on the current line, then a good position is found on a subsequent line, and the test is repeated until the number will fit. Then, when not at the beginning of a line, a space is given and V is output as if under the picture

n(L int width-1)z + d,  
 +d.n(L real width-1)d e n(L exp width-1)z + d, or  
 +d.n(L real width-1)d e n(L exp width-1)z + d " " i  
 +d.n(L real width-1)d e n(L exp width-1)z + d,

respectively.

If the mode of V is specified by BOOL, then first, if the current line is full, a good position is found on a subsequent line; next, if V is true (false), the character yielded by 'flip' ('flop') is output (with no intervening space).

If the mode of V is specified by L BITS, then the elements of the only field of V are output (as if of the mode specified by BOOL) one after the other (with no intervening spaces, and with new lines being taken as required).

If the mode of V is specified by CHAR, then first, if the current line is full, a good position is found on a subsequent line; next V is output (with no intervening space).

If the mode of V is specified by [] CHAR, then first a good page is found; next the elements of V are output (as above) one after the other (with no intervening spaces, and with new lines being taken as required).

```

a) PROC put = (REF FILE f, [] UNION(OUTTYPE, PROC (REF FILE) VOID) x) VOID:
  FOR i TO UPB x
  DO
    IF NOT (status OF f SAYS put char status)
    THEN ensure state(f, put char status)
    FI;
    CASE x[i] IN
      (PROC (REF FILE) VOID pf): pf(f),
      (OUTTYPE ot):
        BEGIN [] SIMPLOUT y = STRAIGHTOUT ot;

          {PROC L real conv = (REF [] CHAR s, L REAL x) VOID:
            # This routine converts 'x' into
            s[0 : L real width + L exp width + 3] #
            BEGIN INT exponent:= 1, rp, last:= L real width + 1;
              BOOL neg =
                subfixed(x, L real width - 1, exponent, s, TRUE);
              exponent -= 1;
              IF power10(s, rp, last)
              THEN s[1]:= "1"; s[2]:= ".";
                FOR i FROM 3 TO last DO s[i]:= "0" OD;
                  exponent += 1
              ELSE round(s, rp, last)
              FI;
              s[0]:= (neg | "-" | "+"); s[last += 1]:= "e"; last += 1;
              BOOL expneg =
                subwhole(exponent, last, last + L exp width, s);
              s[last - 1]:= (expneg | "-" | "+")
            END};

          FOR j TO UPB y
          DO
            CASE y[j] IN
              UNION(NUMBER, {L COMPL}) nc):
                BEGIN
                  INT upb =
                    CASE nc IN
                      {(L INT): L int width},
                      {(L REAL): L real width + L exp width + 3},
                      {(L COMPL): 2 * L real width +
                        2 * L exp width + 9}
                    ESAC;
                  [0 : upb] CHAR s;
                  CASE nc IN
                    {(L INT k):
                      BEGIN INT first:= 0;
                        BOOL neg = subwhole(k, first, upb, s);
                        s[first - 1]:= (neg | "-" | "+")
                      END},
                    {(L REAL r): L real conv(s, r)},
                    {(L COMPL z):
                      BEGIN L real conv(s, re OF z);
                        INT istart = L real width + L exp width + 4;
                        s[istart]:= " "; s[istart + 1]:= "i";
                        L real conv(s[istart + 2 : @ 0], im OF z)
                      END}
                END
            END
          DO
        END
      END
    END
  DO

```

```

ESAC;
WHILE
  IF NOT (status OF f SAYS line ok)
  THEN next pos(f)
  FI;
  (upb  $\geq$  char bound OF f | error(smallline); abort);
  c OF cpos OF f + upb + SIGN(c OF cpos OF f - 1) >
  char bound OF f
DO BOOL mended = (line mended OF f)(f);
  ensure state(f, put char status);
  (NOT mended | newline(f))
OD;
(c OF cpos OF f  $\neq$  1 | (put char OF f)(f, " "));
FOR k FROM 0 TO upb
DO (put char OF f)(f, s[k]) OD;
IF status OF f SAYS logical pos not ok
THEN set logical pos(f)
FI;
test line end(f)
END # numeric #,
(BOOL b):
(IF NOT (status OF f SAYS line ok)
THEN next pos(f)
FI;
put char(f, (b | flip | flop))),
‡(L BITS lb):
FOR k TO L bits width
DO
  IF NOT (status OF f SAYS line ok)
  THEN next pos(f)
  FI;
  put char(f, (k ELEM lb | flip | flop))
OD‡,
(CHAR k):
(IF NOT (status OF f SAYS line ok)
THEN next pos(f)
FI;
put char(f, k)),
([] CHAR ss):
(IF NOT (status OF f SAYS page ok)
THEN ensure page(f, put char status)
FI;
FOR k FROM LWB ss TO UPB ss
DO
  IF NOT (status OF f SAYS line ok)
  THEN next pos(f)
  FI;
  put char(f, ss[k])
OD)
ESAC
OD
END
ESAC
OD;

```

```

b) PROC ? put char = (REF FILE f, CHAR char) VOID:
    BEGIN (put char OF f)(f, char);
        IF status OF f SAYS logical pos not ok
        THEN set logical pos(f)
        FI;
        test line end(f)
    END;

```

For formatless input, 'get' (a) and 'read' (section 10.5.1 of the Revised Report) may be used. Values from the book are assigned to each straightened name N from the data list as follows:

If the mode of N is specified by REF L INT, then first, the book is searched for the first character that is not a space (finding good positions on subsequent lines as necessary); next, the largest string is read from the book that could be "indited" (section 10.3.4.1.1.kk of the Revised Report) under the control of some picture of the form +n(k1)" "n(k2)dd or n(k2)dd (where k1 and k2 yield arbitrary nonnegative integers); this string is converted to an integer and assigned to N; if the conversion is unsuccessful, the event routine corresponding to 'on value error' is called.

If the mode of N is specified by REF L REAL, then first, the book is searched for the first character that is not a space (finding good positions on subsequent lines as necessary); next, the largest string is read from the book that could be "indited" (section 10.3.4.1.1.kk of the Revised Report) under the control of some picture of the form +n(k1)" "n(k2)d or n(k2)d followed by .n(k3)dd or by ds., possibly followed again by en(k4)" "+n(k5)" "n(k6)dd or by en(k5)" "n(k6)dd; this string is converted to a real number and assigned to N; if the conversion is unsuccessful, the event routine corresponding to 'on value error' is called.

If the mode of N is specified by REF L COMPL, then first, a real number is input (as above); next, the book is searched for the first character that is not a space; next, a character is input and, if it is not "l" or "i", then the event routine corresponding to 'on char error' is called, the suggestion being "l"; finally, a second real number is input. Note that if either conversion to a real number is unsuccessful, the event routine corresponding to 'on value error' is called. In that case, no value is assigned to either subname of N. If conversion is successful, the first (second) number read is assigned to the first (second) subname of N.

{Numbers are input using a finite-state machine that largely follows the syntax of numbers as given in sections 8.1.1 and 8.1.2 of the Revised Report. However, spaces within numbers are explicitly allowed after a sign and after a 'times-ten-to-the-power-symbol' (although no good positions are found on a subsequent line).}

If the mode of N is specified by REF BOOL, then first, the book is searched for the first character that is not a space (finding good positions on subsequent lines as necessary); next, a character is read; if this character is the same as that yielded by 'flip' ('flop'), then true (false) is assigned to N; otherwise, the event routine corresponding to 'on char error' is called, the suggestion being 'flop'.

If the mode of N is specified by REF L BITS, then input takes place (as for booleans, see above) to the subnames of N one after the other (with new

lines being taken as required).

If the mode of N is specified by REF CHAR, then first, if the current line is exhausted, a good position is found on a subsequent line; next, a character is read and assigned to N.

If the mode of N is specified by REF [] CHAR, then input takes place (as above) to the subnames of N one after the other (with new lines being taken as required).

If the mode of N is specified by REF STRING, then characters are read until either

- i) a character is encountered which is contained in the string associated with the file by a call of the routine 'make term', or
- ii) the current line is exhausted, whereupon the event routine corresponding to 'on line end' (or, where appropriate, to 'on page end' or 'on logical file end') is called; if the event routine moves the current position to a good position, then input of characters is resumed. Note that, if the page has overflowed, a new page is given by default, but, if the line has overflowed, no default action is taken.

The string consisting of the characters read is assigned to N (note that, if the current line has already been exhausted, or if the current position is at the start of an empty line or outside the logical file, then an empty string is assigned to N).

```
c) PROC get = (REF FILE f, [] UNION(INTYPE, PROC (REF FILE) VOID) x) VOID:
  FOR i TO UPB x
  DO
    IF NOT (status OF f SAYS get char status)
    THEN ensure state(f, get char status)
    FI;
    CASE x[i] IN
      (PROC (REF FILE) VOID pf): pf(f),
      (INTYPE it):
        BEGIN [] SIMPLIN y = STRAIGHTIN it, CHAR k;

          PRIO ! = 8;
          OP ! = (CHARBAG s, CHAR sugg) VOID:
            IF k:= sugg;
            BOOL ok =
              IF (char error mended OF f)(f, k)
              THEN char in bag(k, s)
              ELSE FALSE
            FI;
            ensure state(f, get char status); NOT ok
            THEN error(wrongchar); k:= sugg
            FI;

          PROC skip initial spaces = VOID:
            WHILE
              IF NOT(status OF f SAYS line ok)
              THEN next pos(f)
              FI;
              get char(f, k); k = "."
            DO SKIP OD;
```

```

PROC skip spaces = VOID:
  WHILE
    IF status OF f SAYS line ok
    THEN get char(f, k); k = "."
    ELSE (check pos(f) | get char(f, k)
          | error(nocharpos); abort);
    FALSE
  FI
DO SKIP OD;

OP NODIGIT = (CHAR c) BOOL:
  NOT char in bag(c, radix10digit); # inline code #

PROC read L integer = (REF L INT i) BOOL:
  BEGIN BOOL ok:= TRUE, BOOL neg = k = "-";
    (neg OR k = "+" | skip spaces);
    (NODIGIT k | radix10digit ! "0");
    L INT j:= K chardig(k);
  WHILE
    IF status OF f SAYS line ok
    THEN get char(f, k);
      IF NODIGIT k
      THEN backchar(f); FALSE
      ELSE (ok | ok:= j ADD chardig(k); TRUE)
    FI
    ELSE FALSE
  FI
DO SKIP OD;
IF ok THEN i:= (neg | -j | j) FI;
ok
END;

PROC read L real = (REF L REAL r) BOOL:
  BEGIN BOOL ok:= TRUE, BOOL neg = k = "-";
    (neg OR k = "+" | skip spaces);
    [0 : L real width] CHAR s, INT index:= -1, exp:= 0,
    BOOL sig:= FALSE;

  PROC read digits = (BOOL after) VOID:
    BEGIN INT i = ABS after;
      (NODIGIT k | radix10digit ! "0");
    WHILE
      IF NODIGIT k
      THEN FALSE
      ELIF
        IF sig:= sig OR k ≠ "0"
        THEN (index < L real width | s[index += 1]:= k
              | exp += 1)
        FI;
        exp -= i; status OF f SAYS line ok
      THEN get char(f, k); TRUE
      ELSE FALSE
    FI
    DO SKIP OD
  END;

  END;

```

```

IF k ≠ "." THEN read digits(FALSE) FI;
IF k = "."
THEN
  IF (status OF f SAYS line ok | TRUE | check pos(f))
  THEN get char(f, k)
  ELSE error(nocharpos); abort
  FI;
  read digits(TRUE)
FI;
IF char in bag(k, times ten to the power)
THEN INT e; skip spaces; ok:= read integer(e);
  IF ok:= ok AND
    (SIGN e ≠ SIGN exp OR ABS e ≤ max int - ABS exp)
  THEN exp += e
  FI
ELSE back char(f)
FI;
IF NOT ok THEN FALSE
ELIF L REAL x; string to L real(s[:index], exp, neg, x)
THEN r:= x; TRUE
ELSE FALSE
FI
END‡;

FOR j TO UPB y
DO
  CASE y[j] IN
    (UNION( ‡ REF L INT ‡, ‡ REF L ‡ REAL ‡,
      ‡ REF L COMPL ‡) irc):
      BEGIN skip initial spaces;
        IF NOT
          CASE irc IN
            ‡(REF L INT ii): read L integer(ii)‡,
            ‡(REF L REAL rr): read L real(rr)‡,
            ‡(REF L COMPL zz):
              (L COMPL z;
                BOOL ok:= read L real(re OF z); skip spaces;
                IF NOT char in bag(k, plus i times)
                THEN plus i times ! "i"
                FI;
                skip spaces;
                (ok:= ok AND read L real(im OF z) | zz:= z); ok)‡
          ESAC
        THEN BOOL mended = (value error mended OF f)(f);
          ensure state(f, get char status);
          (NOT mended | error(wrongval); abort)
        FI
      END,
    (REF BOOL bb):
      (skip initial spaces;
        IF NOT char in bag(k, flipflop)
        THEN flipflop ! flop
        FI;
        bb:= k = flip),

```

```

‡(REF L BITS lb):
  ([1 : L bits width] BOOL b;
  FOR i TO L bits width
  DO skip initial spaces;
    IF NOT char in bag(k, flipflop)
    THEN flipflop ! flop
    FI;
    b[i]:= k = flip
  OD;
  lb:= L bits pack(b))‡,
(REF CHAR cc):
  (IF NOT (status OF f SAYS line ok)
  THEN next pos(f)
  FI;
  get char(f, cc)),
(REF [] CHAR ss):
  (IF NOT (status OF f SAYS page ok)
  THEN ensure page(f, get char status)
  FI;
  FOR i FROM LWB ss TO UPB ss
  DO
    IF NOT (status OF f SAYS line ok)
    THEN next pos(f)
    FI;
    get char(f, ss[i])
  OD),
(REF STRING ss):
  BEGIN INT index:= 0,
    upbs:= 80 MIN (char bound OF f - c OF cpos OF f + 1);
  REF STRING s:= HEAP [1 : upbs] CHAR;
  WHILE
    IF (status OF f SAYS line ok | TRUE | check pos(f))
    THEN get char(f, k);
      IF char in bag(k, term OF f)
      THEN back char(f); FALSE
      ELSE TRUE
      FI
    ELSE FALSE
    FI
  DO
    IF index = upbs
    THEN REF [] CHAR t = s;
      s:= HEAP [1 : upbs +:=
        80 MIN (char bound OF f - c OF cpos OF f + 2)
        ] CHAR;
      s[: index]:= t
    FI; # extend s #
    s[index +:= 1]:= k
  OD;
  ss:= s[: index]
END
ESAC
OD
END
ESAC
OD;

```



d) PROC ? get char = (REF FILE f, REF CHAR char) VOID:

```

char:=
  IF CHAR k; BOOL conv ok = (get char OF f)(f, k);
  test line end(f);
  IF status OF f SUGGESTS lfe in current line
  THEN test logical file end(f)
  FI;
  conv ok
  THEN k
  ELIF CHAR sugg:= ".";
  BOOL mended = (char error mended OF f)(f, sugg);
  ensure state(f, get char status); mended
  THEN sugg
  ELSE error(wrongchar); "."
  FI;

```

e) PROC ? back char = (REF FILE f) VOID:

C If the current position is not at the beginning of a buffer, it is set back over one position; otherwise, an error message is given and the elaboration of the particular program is aborted. (This can only be caused by a call of the event routine corresponding to 'on char error' while reading a string; this call must then have caused the current position to be moved to the first position of a new buffer, while it returned a character from the terminator string of 'f'. This case is assumed to be exceedingly rare.) C;

f) MODE ? CHARBAG =

C Some mode which allows efficient retrieval of information on the presence or absence of a given character in a given set. C;

OP ? STRINGTOBAG = (STRING s) CHARBAG:

C The string in 's' is converted to a corresponding value of the mode specified by CHARBAG. C;

PROC ? char in bag = (CHAR k, CHARBAG s) BOOL:

C This routine returns true if the character 'k' is contained in 's', and false otherwise. C;

OP + = (CHARBAG s, t) CHARBAG:

C A value K of the mode CHARBAG is delivered, such that, for any character C, C is in K iff C is in s and/or t. C;

```

CHARBAG ? radix10digit = STRINGTOBAG "0123456789",
? radix 2digit = STRINGTOBAG "01",
? radix 4digit = STRINGTOBAG "0123",
? radix 8digit = STRINGTOBAG "01234567",
? radix16digit = STRINGTOBAG "0123456789abcdef",
? times ten to the power = STRINGTOBAG "\e",
? flipflop = STRINGTOBAG (flip + flop),
? plus i times = STRINGTOBAG "|i",
? xylpkq = STRINGTOBAG "xylpkq";

```

### 10.3. EFFICIENCY

The efficiency of formatless transput critically depends on the efficiency of the routines 'put char' and 'get char'. It is recommended that these be not straightforwardly implemented as procedures; rather, inline code should be generated here. (This also holds for the calls of 'test line end' inside 'put char' and 'get char'.) When outputting numbers, there is no need to test for the line being ended after each character. Consequently, the corresponding piece of code in 'put' has been optimized. If an implementation offers a fast way to output strings of characters, this may profitably be applied at this spot.

Various small operators and routines that are used in 'get', such as 'NODIGIT' and 'char in bag', also offer attractive possibilities for optimization. The mode 'CHARBAG' is incorporated just to encourage efficient implementation; if efficiency is not important, the mode 'STRING' may be substituted instead. In that case, 'STRINGTOBAG' becomes a dummy operation, while 'char in bag' may be replaced by 'char in string'. {Note that the routine 'char in bag' is also heavily used in formatted transput ('indit string', 11.2.3.o.).}

## 11. FORMATTED TRANSPUT

### 11.1. DIFFERENCES

- {D} A different structure has been chosen for the mode specified by FORMAT. In the Revised Report, the mode specified by FORMAT almost exactly mirrors the structure of 'format-texts'. This mode FORMAT serves two purposes: it contains a description of some format (which does not change), and it contains some administration on how this format is currently being used (which does change). These two parts have been separated in the present model. Here, the mode specified by FORMAT only contains a description of the format; thus, formats need no longer be copied upon assignment. The dynamic information is collected in the 'piece' field of the file to which the format is associated. This change results in remarkable simplifications in the routines 'get next picture', 'do fpattern' and 'associate format'.
- {D} Formatted transput is rather complicated, and slowed down, by the possibility of using "dynamic" replicators, i.e., replicators whose value is not known until runtime. In the Revised Report, both dynamic replicators and non-dynamic replicators (i.e., replicators that are plain integers) are treated in a uniform way; they both give rise to a routine returning an integer. For example, the replicator '10' gives rise to a routine composed from 'INT: 10'; the dynamic replicator 'n(lim-1)' gives rise to 'INT: (lim-1)'. For output, as a consequence, the frames that a pattern is composed of are traversed three times:
  - First, the frames are traversed to evaluate all replicators (i.e., each PROC INT is turned into an INT);
  - Second, the frames are traversed to extract the necessary information to structure the string to be output (the position of the decimal point, the width of the exponent, etc., are determined);
  - Third, after building the string, this string is matched against the frames character by character, to determine the actual output (insertions may have to be placed in between, zeroes may be replaced by spaces, etc.).

For input, only the first and (the reverse of) the third kind of traversal is needed.

In the case where a pattern contains only non-dynamic replicators (which is probably the common case), a much simpler scheme is possible: in this case, there is no need to construct routines returning an integer; rather, the integers themselves may be incorporated in the pattern. Also, the compiler is quite capable of extracting the information needed to structure the string in that case. (Alternatively, this may be done upon associating the format with the file.) In this way, the frames need only be traversed once.

In the present model, two kinds of pictures are distinguished: simple, static pictures (whose mode is SPICT), and dynamic pictures. A static picture may only arise from an 'integral-', 'real-', 'boolean-', 'complex-', 'string-' or 'bits-pattern'. The reasons therefore are twofold:

- These kinds of pictures will probably occur most often;
- The treatment of choice-patterns and format-patterns is complicated by the fact that their insertions are evaluated at moments which

are different from those of other patterns.

Outputting a value W using a picture P now proceeds as follows:

- If the mode of P is DPICT, then P is staticized, thereby yielding a static picture;
  - Subsequently, the picture is either static, or it is a dynamic picture whose pattern is a 'choice-', 'format-', 'general-' or 'void-pattern'. Dynamic pictures are handled in the same way as they are in the Revised Report. Static pictures can now be handled quite efficiently: the picture contains all information needed to build the string to be output. After having done this, 'edit string' may be called.
- {D} There still is one further optimization possible. In the Revised Report, a format contains quite a lot of redundant information in the form of default values for insertions, replicators and the like. In the definition given below, these default values are not explicitly stored in the data structures that are used. Thus, a "frame" is either a routine (returning an integer), a character, or a fixed name referring to a string. This will save both space and time.

## 11.2 NEW DEFINITION

In formatted transput, each straightened value from a data list (cf section 10.2) is matched against a constituent 'picture' of a 'format-text' provided by the user. A 'picture' specifies how a value is to be converted to or from a sequence of characters and prescribes the layout of those characters in the book. Features which may be specified include the number of digits, the position of the decimal point and of the sign, if any, suppression of zeroes and the insertion of arbitrary strings. For example, using the 'picture' '-d.3d "." 3d "." e.z+d', the value '1234.567' would be transput as the string ".1.234.567.e.+3".

A "format" is a multiple value (i.e., an internal object) of mode 'FORMAT', which mirrors the structure of a 'format-text' (which is an external object). For a description of the syntax of 'format-texts', the reader is referred back to sections 10.3.4.1.1 (about 'collections' and 'pictures') and 10.3.4.2 - 10.3.4.10 (about 'patterns') of the Revised Report. Below, only the semantics are described for obtaining the corresponding format from a 'format-text'. This is necessary since the internal representation of a 'format-text' in this model is rather different from the representation used in the Revised Report. The only deviation in the semantics is that, whereas this model prescribes sequential elaboration of replicators, the Revised Report uses collateral elaboration.

A "piece" is brought into being by means of associating a format with a file. A piece is a structured value composed of a reference to the collection list currently in use, a 'count' field which tells how many times this collection list should be used, the index of the current collection, and a reference to a piece to be used after the current collection list is exhausted. Upon "associating" (...) a format with a file, the piece will contain a reference to the collection list of the format. Upon selection of a picture from this collection list that is itself a replicated "collection", the current collection list is "suspended", and this new collection list is made to be the current collection list. Subsequent transput now uses this collection list until it has been exhausted, after

which the suspended collection list is again made to be the current one. Something similar happens when other formats are invoked by means of 'format-patterns'. (section 10.3.4.9.1 of the Revised Report).

Although a 'format-text' may contain 'ENCLOSED-clauses' (in 'replicators' and 'format-patterns') or 'units' (in 'general-patterns'), these are not elaborated at this stage but are, rather, turned into routines for subsequent calling as and when they are encountered during formatted transput. Note however that, if a data picture contains only non-dynamic replicators, i.e., replicators that are plain integers, these replicators are elaborated directly, thus giving rise to a so-called static picture. {The term "data picture" is used as an abbreviation for a picture whose pattern is an 'integral-', 'real-', 'boolean-', 'complex-', 'string-' or 'bits-pattern'.} It still remains true that the elaboration of a 'format-text' does not result in any actions of any significance to the user.

If, at runtime, a data picture is encountered that does contain dynamic replicators, then this data picture is "staticized", thereby yielding a static picture, which is used when actually transputting values. Note that the compiler is not forced to construct any static pictures; if it does not, the result will still be the same. An implementer may also choose to staticize such pictures upon associating the format with the file.

For ease of description, the semantics given below are such that a static picture is the result of staticizing a dynamic data picture; this intermediate dynamic data picture is of course not necessarily constructed in an actual implementation.

#### 11.2.1. Mode declarations

- a) MODE FORMAT = REF [] COLLECTION;
- b) MODE ? COLLECTION = UNION(PICTURE, COLLITEM);
- c) MODE ? COLLITEM =  
     STRUCT(INSERTION i1,  
         PROC INT rep, # replicator #  
         FORMAT p,  
         INSERTION i2);
- d) MODE ? PICTURE =  
     UNION(SPICT, DPICT, CPICT, FPICT, GPICT, VOIDPICT);
- e) MODE ? SPICT =  
     STRUCT(UNION(INTPATTERN, REALPATTERN, BOOLPATTERN, COMPLPATTERN,  
         STRINGPATTERN, BITSPATTERN) p,  
         REF [] SFRAME sframes);
- f) MODE ? DPICT =  
     STRUCT(INT type,  
         REF [] DFRAME frames);

- g) MODE 7 CPICT =  
 STRUCT(INSERTION i1,  
 INT type,  
 REF [] INSERTION c,  
 INSERTION i2);
- h) MODE 7 FPICT =  
 STRUCT(INSERTION i1,  
 PROC FORMAT pf,  
 INSERTION i2);
- i) MODE 7 GPICT =  
 STRUCT(INSERTION i1,  
 FLEX [1:0] PROC INT spec,  
 INSERTION i2);
- j) MODE 7 VOIDPICT = INSERTION;
- k) MODE 7 INSERTION = REF [] DFRAME;
- l) MODE 7 DFRAME =  
 UNION(PROC INT, REF [] CHAR, CHAR);
- m) MODE 7 SFRAME =  
 UNION(INT, REF [] CHAR, CHAR);
- n) MODE 7 SINSERT = SFRAME;
- o) MODE 7 INTPATTERN = REF STRUCT(INT width, sign);  
 # 'width': The length of the string controlled by the integral pattern;  
 'sign' : The absolute value of 'sign' is the length of the string controlled by the sign mould of the pattern. If 'sign' < 0 (> 0), then the sign mould contains a descendent minus-symbol (plus-symbol). If 'sign' = 0, then the pattern contains no sign mould. Note that because of this way of coding there is no need for u- or v-frames, since the 'sign' field contains the necessary information. #
- p) MODE 7 REALPATTERN = REF STRUCT(INT b, s1, a, e, s2, point);  
 # 'b' : The length of the string controlled by the first integral mould of the stagnant part of the pattern.  
 'a' : The length of the string controlled by the second integral mould of the stagnant part.  
 'e' : The length of the string controlled by the integral pattern of the exponent part.  
 's1' : The length of the string controlled by the sign mould of the stagnant part, coded in the same way as the 'sign' field of the integral pattern.  
 's2' : Idem for the sign mould of the exponent.  
 'point': 'point' = 1 if the pattern contains a point frame, and 0 otherwise. #
- q) MODE 7 COMPLPATTERN = REF STRUCT(REALPATTERN re, im);
- r) MODE 7 BOOLPATTERN = VOID;

- s) MODE ? STRINGPATTERN = INT;  
 # The length of the string controlled by the pattern. #
- t) MODE ? BITSPATTERN = REF STRUCT(INT width, radix);  
 # 'width': The length of the string controlled by the pattern.  
 'radix': The radix of the radix frame. #

{The corresponding revised metaproduction rules are not given here since they follow trivially from the above mode declarations.}

### 11.2.2. Semantics

{This section replaces sections 10.3.4.1.2, 10.3.4.8.2, 10.3.4.9.2 and 10.3.4.10.2 of the Revised Report.}

{The yield of a 'format-text' is that of its 'collection-list', by way of pre-elaboration.}

- a) The yield N of a 'collection-list' C, in an environ E, is determined as follows:
- N is a newly created name {whose mode is 'FORMAT'};
  - N is equal in scope to the environ necessary for C in E;
  - N is made to refer to a value V {whose mode is 'row of COLLECTION'}, having a descriptor ((l, m)), where m is the number of constituent 'collections' of C, and elements determined as follows:  
 For j = 1, ..., m, let C<sub>j</sub> be the j-th constituent 'collection' of C.  
 Case A: The direct descendents of C<sub>j</sub> include a 'picture' P:  
    - P is elaborated in E;
    - the j-th element of V is the yield of P;
 Case B: The direct descendents of C<sub>j</sub> include a first 'insertion' I<sub>1</sub>, a 'replicator' REP, a 'collection-list-pack' P and a second 'insertion' I<sub>2</sub>:  
    - the j-th element of V is a structured value whose mode is 'COLLITEM' and whose fields, taken in order, are the yields of
      - {i1} I<sub>1</sub>,
      - {rep} REP,
      - {p} {the 'collection-list' of} P,
      - {i2} I<sub>2</sub>.
- b) The yield N, in an environ E, of a 'picture' PICT is a value W whose mode is 'PICTURE', determined as follows:
- Let V be the yield of the elaboration of the constituent 'pattern' P together with the constituent 'insertion' I of PICT in E (c, d, e, f);
  - If the constituent 'pattern' of PICT is an 'integral-', 'real-', 'boolean-', 'complex-', 'string-' or 'bits-pattern' which does not contain any dynamic replicators, then
    - W is the result of "staticizing" (k) V in E;
 Otherwise,
    - W is V.

c) The yield, in an environ E, of an 'integral-', 'real-', 'boolean-', 'complex-', 'string-' or 'bits-pattern' P {sections 10.3.4.2.1.a, 10.3.4.3.1.a, ..., 10.3.4.7.1.a of the Revised Report}, together with an 'insertion' I, is a structured value W whose mode is 'DPICT', whose fields, taken in order, are:

- {type} 1 (2, 3, 4, 5) if P is an 'integral-' ('real-', 'boolean-', 'complex-', 'string-') '-pattern' and 6 (8, 12, 20) if P is a 'bits-pattern' whose constituent 'RADIX' is a 'radix-two' ('-four', '-eight', '-sixteen');
- {frames} a newly created name, equal in scope to the environ necessary for P in E, which is made to refer to a value F, whose mode is 'row of FRAME', having a descriptor ((1, n)) and n elements determined as follows:
  - a counter i is set to 0;
  - the constituent frames of P, in order, are "transformed" (h) in E into F, using i;
  - I is "transformed" (i) in E into F.

d) The yield, in an environ E, of a 'choice-pattern' P, together with an 'insertion' I, is a structured value W whose mode is 'CPICT', determined as follows:

- let n be the number of constituent 'NEST-literals' of the 'praglit-list-pack' of P;
- let  $S_i$ ,  $i = 1, \dots, n$ , be a 'NEST-insertion' akin to the i-th of those constituent 'NEST-literals';
- the insertion  $I_i$  of P, all of  $S_1, \dots, S_n$ , and the insertion I are elaborated in E;
- the fields of W, in order, are:
  - {i1} the yield of  $I_i$ ;
  - {type} 1 (2) if P is a 'boolean-' ('integral-') '-choice-pattern';
  - {c} a newly created name, equal in scope to the environ necessary for P in E, which is made to refer to a value F whose mode is 'row-of-INSERTION', having a descriptor ((1, n)), and n elements, that selected by (i),  $i = 1, \dots, n$ , being the yield of  $S_i$ ;
  - {i2} the yield of I.

e) The yield, in an environ E, of a 'NEST-format-pattern' P, together with an 'insertion' I, is a structured value whose mode is 'FPICT' and whose fields, taken in order, are:

- {i1} the yield of the insertion of P;
- {pf} a routine whose mode is 'procedure yielding FORMAT', composed of a 'procedure-yielding-FORMAT-NEST-routine-text' whose unit U is a new unit akin to the 'meek-FORMAT-ENCLOSED-clause' of P, together with the environ necessary for U in E;
- {i2} the yield of I.

f) The yield, in an environ E, of a 'NEST-general-pattern' P, together with an 'insertion' I, is a structured value whose mode is 'GPICT' and whose fields, taken in order, are:

- {i1} the yield of the insertion of P;
- {spec} a multiple value W whose mode is 'row of procedure yielding integral', having a descriptor ((1, n)), where n is the number of constituent 'meek-integral-units' of the 'width-specification-option' of P, and n elements determined as follows:



- For  $i = 1, \dots, n$ ,
- ⊙ the  $i$ -th element of  $W$  is a routine, whose mode is 'procedure yielding integral', composed of a 'procedure-yielding-integral-NEST-routine-text' whose unit  $U$  is a new unit akin to the  $i$ -th of those 'meek-integral-units', together with the environ necessary for  $U$  in  $E$ ;
  - ⊙ {i2} the yield of  $I$ .
- g) The yield  $N$ , in an environ  $E$ , of an 'insertion'  $I$  {section 10.3.4.1.1.d of the Revised Report} is determined as follows:
- ⊙  $N$  is a newly created name {whose mode is INSERTION};
  - ⊙  $N$  is equal in scope to the environ necessary for  $I$  in  $E$ ;
  - ⊙  $N$  is made to refer to a value  $W$  whose mode is 'row of FRAME', having a descriptor  $((1, n))$  and  $n$  elements, determined as follows:
    - ⊙ a counter  $i$  is set to 0;
    - ⊙  $I$  is "transformed" ( $i$ ) in  $E$  into  $W$ , using  $i$ .
- h) A 'frame'  $V$  is "transformed" in an environ  $E$  into a multiple value  $F$  whose mode is 'row of FRAME', using a counter  $i$ , as follows:
- ⊙ the constituent 'insertion' of  $V$  is "transformed" ( $i$ ) in  $E$  into  $F$ , using  $i$ ;
  - ⊙ the constituent 'replicator' of  $V$  is "transformed" ( $j$ ) in  $E$  into  $F$ , using  $i$ ;
  - ⊙ if the constituent 'UNSUPPRESSETY-suppression' of  $V$  contains a 'letter-s-symbol', then
    - ⊙  $i$  is increased by 1;
    - ⊙ the element of  $F$  selected by ( $i$ ) is a 'letter-s-symbol';
    - ⊙  $i$  is increased by 1;
    - ⊙ the element of  $F$  selected by ( $i$ ) is the intrinsic value of the constituent 'symbol' of the marker of  $V$ .
- i) An 'insertion'  $I$  is "transformed" in an environ  $E$  into a multiple value  $F$  whose mode is 'row of FRAME', using a counter  $i$ , as follows:
- ⊙ Let  $U_1, \dots, U_n$ , be the constituent 'UNSUPPRESSETY-replicators' of  $I$ , and let  $A_j, j = 1, \dots, n$  be the 'denoter-coercee' or 'alignment-code' {immediately} following  $U_j$ ;
  - ⊙ For  $j = 1, \dots, n$ :
    - ⊙  $U_j$  is "transformed" ( $j$ ) in  $E$  into  $F$ , using  $i$ ;
    - ⊙  $i$  is increased by 1;
    - ⊙ the element of  $F$  selected by ( $i$ ) is determined as follows:
      - Case A:  $A_j$  is an 'alignment-code':
        - ⊙ it is the {character which is the} intrinsic value of the 'LETTER-symbol' of  $A_j$ ;
      - Case B:  $A_j$  is a 'denoter-coercee':
        - ⊙ it is a newly created name, equal in scope to the environ necessary for  $F$  in  $E$ , which is made to refer to the yield of  $A_j$ .
- j) A 'NEST-UNSUPPRESSETY-replicator'  $R$  is "transformed" in an environ  $E$  into a multiple value  $F$  whose mode is 'row of FRAME', using a counter  $i$ , as follows:
- If  $R$  is not invisible, then:
- ⊙  $i$  is increased by 1;
  - ⊙ the element of  $F$  selected by ( $i$ ) is a routine whose mode is 'procedure yielding integral', composed of a 'procedure-yielding-integral-NEST-routine-text' whose 'unit' is  $U$ , together with the environ necessary for  $U$  in  $E$ , where  $U$  is determined as follows:

Case A: R contains a 'meek-integral-ENCLOSED-clause' C:

- U is a new 'unit' akin to C;

Case B: R contains a 'fixed-point-numeral' D, but no 'ENCLOSED-clause':

- U is a new 'unit' akin to D;

Otherwise,

- no action is taken.

k) A 'data picture' P is "staticized" in an environ E, yielding a value V of mode 'SPICT', as follows:

- Let F be the value referred to by the 'frames' field of P;
- Let  $V_i$ ,  $i = 1, \dots, n$ , be the yield of "staticizing" (1) the  $i$ -th element of F, where  $((1, n))$  is the descriptor of F;
- Let X be the "structure description" (m) determined by the type of P and  $V_1, \dots, V_n$ ;
- The fields of V, taken in order, are:
  - {p} X;
  - {sframes} a newly created name, equal in scope to the primal environ, which is made to refer to a value F, whose mode is 'row of SFRAME', having a descriptor  $((1, n))$  and n elements, that selected by (i),  $i = 1, \dots, n$ , being  $V_i$ .

l) A 'frame' V is "staticized" in an environ E, yielding a value S of mode 'SFRAME', as follows:

- If V is a routine, then
  - S is the yield of the calling of V in E;
- Otherwise,
  - S is V.

m) The "structure description" determined by a type I and sframes  $V_1, \dots, V_n$  is a value X, determined as follows:

Case A: I equals 1:

- Let  $i$  be the "position" (n) of the "+" or "-" in  $V_1, \dots, V_n$ , if any, and 0 otherwise;
- Let  $j$  be the length controlled by  $V_1, \dots, V_i$ ;
- X is a newly created name, equal in scope to the primal environ, and is made to refer to a value of the mode 'STRUCT(INT width, sign)', whose fields, taken in order, are:
  - {width} the length "controlled" (o) by  $V_1, \dots, V_n$ ;
  - {sign}  $j$  if  $i = 0$  or  $V_i = "+"$ , and  $-j$  otherwise.

Case B: I equals 2:

- Let  $e$  be the position of the "e" in  $V_1, \dots, V_n$ , if any, and  $n+1$  otherwise;
- Let  $p$  be the position of the "." in  $V_1, \dots, V_n$ , if any, and  $e$  otherwise;
- Let  $x$  be the position of the "+" or "-" in  $V_1, \dots, V_p$ , if any, and 0 otherwise;
- Let  $s_1$  be the length controlled by  $V_1, \dots, V_x$ ;
- Let  $u$  be the position of the "+" or "-" in  $V(e+1), \dots, V_n$ , if any, and 0 otherwise;
- Let  $s_2$  be the length controlled by  $V(e+1), \dots, V_u$ ;
- X is a newly created name, equal in scope to the primal environ, and is made to refer to a value of the mode 'STRUCT(INT b,  $s_1$ , a, e,  $s_2$ , point)', whose fields, taken in order, are:

- ⊙ {b} the length controlled by  $V_1, \dots, V_{(p-1)}$ ;
- ⊙ {s1} s1 if  $x = 0$  or  $V_x = "+"$ , and  $-s1$  otherwise;
- ⊙ {a} the length controlled by  $V_{(p+1)}, \dots, V_{(e-1)}$ ;
- ⊙ {e} the length controlled by  $V_{(e+1)}, \dots, V_n$ ;
- ⊙ {s2} s2 if  $u = 0$  or  $V_u = "+"$ , and  $-s2$  otherwise;
- ⊙ {point} 1 if  $p > 0$ , and 0 otherwise.

Case C: I equals 3:

- ⊙ X is a value of mode 'BOOLPATTERN' {and is equal to empty}.

Case D: I equals 4:

- ⊙ Let i be the position of the "i" in  $V_1, \dots, V_n$ ;
- ⊙ X is a newly created name, equal in scope to the primal environ, and is made to refer to a value of the mode 'STRUCT(REALPATTERN re, im)', whose fields, taken in order, are:
  - ⊙ {re} the structure description determined by 2 and  $V_1, \dots, V_{(i-1)}$ ;
  - ⊙ {im} the structure description determined by 2 and  $V_{(i+1)}, \dots, V_n$ .

Case E: I equals 5:

- ⊙ X is a value of mode 'STRINGPATTERN' and is equal to the length controlled by  $V_1, \dots, V_n$ .

Case F: I equals 6, 8, 12 or 20:

- ⊙ X is a newly created name, equal in scope to the primal environ, and is made to refer to a value of the mode 'STRUCT(INT width, radix)', whose fields, taken in order, are
  - ⊙ {width} the length controlled by  $V_1, \dots, V_n$ ;
  - ⊙ {radix} I-4.

n) The "position" of a character C in sframes  $V_1, \dots, V_n$  is an integer i such that  $V_i$  is C {which, if it exists at all, is unique}.

o) The length "controlled" by sframes  $V_1, \dots, V_n$  is an integer I, determined as follows:

- ⊙ counters i and j are set to 0 and 1, respectively;
- ⊙ while  $j \leq m$ :
  - ⊙ if  $V_j$  is an integer, then rep is set to the maximum of 0 and  $V_j$ , and j is increased by 1, otherwise rep is set to 1;
  - ⊙ if  $V_j$  is an "s", then j is increased by 1;
  - ⊙ if  $V_j$  is a ".", "+", "-", "e", "i", "z", "d" or "a", then i is increased by rep;
  - ⊙ j is increased by 1;
- ⊙ I is i.

During formatted transput, values are transput using the current format of the file. This current format, together with its administration, is incorporated in the 'piece' field of the file. The mode of that field is 'reference to reference to FORMATLIST', and it refers to the following information:

- (count) the number of times the current collection list is to be repeated;
- (cp) the number of the collection to be executed next;
- (p) the current collection list;
- (next) a reference to a chain of (embracing) collection lists with which to continue after the current one is finished.

Upon associating a format with a file, a name W is created, which is (initially) made to refer to a value of the mode specified by FORMATLIST, whose fields are:

- (count) 1 (since the collection list comprising the format is to be repeated once);
- (cp) 1;
- (p) the collection list of the given format;
- (next) a nil name.

Subsequently, the 'piece' subname of 'f' is made to refer to W.

When, during formatted transput to a file 'f', a collection is encountered which itself contains a collection list c, then further transput uses the collections of c, and c is repeated r times, where r is the integer returned by the replicator of the collection containing c. In that case, a new name W is created which is made to refer to a value of the mode specified by FORMATLIST, and whose fields are:

- (count) r;
- (cp) 1;
- (p) the yield of c;
- (next) the {old} 'piece' field of 'f'.

Subsequently, the 'piece' subname of 'f' is made to refer to W.

Something very similar occurs when a format pattern fp is encountered: A new name W is created, which is made to refer to a value of the mode specified by FORMATLIST, whose fields are:

- (count) 1;
- (cp) 1;
- (p) the value {a format} returned by the 'pf' field of 'fp';
- (next) the {old} 'piece' field of 'f'.

Subsequently, the 'piece' field of 'f' is made to refer to W.

In all three cases, a special generator is needed to create W: the newly created name must have a scope which is not older than the scope of the value to which it is made to refer, nor newer than that of the file with which the format is associated. {Here, the scope is taken equal to that of the file. Even if another solution is adopted, once a scope has been adopted upon associating, the scope of the 'piece' field will not change with the other manipulations.}

11.2.3. Formatted transput

```

a) MODE ? FORMATLIST =
    STRUCT(INT count, cp, FORMAT p, REF FORMATLIST next);

b) PROC ? get next picture = (REF FILE f, REF PICTURE picture) VOID:
    IF piece OF f ::= REF REF FORMATLIST(NIL)
    THEN error(nofformat); abort
    ELSE BOOL picture found:= FALSE, STATUS reading = status OF f;
      IF cp OF piece OF f > UPB p OF piece OF f
      THEN update cp(f, FALSE, SKIP)
      FI;
    WHILE NOT picture found
    DO
      IF cp OF piece OF f = 0      # format ended #
      THEN BOOL mended = (format mended OF f)(f);
        ensure state(f, reading);
        IF NOT mended
        THEN cp OF piece OF f:= count OF piece OF f:= 1
        ELIF cp OF piece OF f = 0
        THEN error(nofformat); abort
        FI
      ELSE REF REF FORMATLIST piece = piece OF f;
        CASE (p OF piece)[cp OF piece] IN
          (COLLITEM cl):
            (REF FORMATLIST pl = piece;
              [1 : UPB il OF cl] SINSERT si;
              staticize frames(il OF cl, si);
              INT count = rep OF cl;
              (reading SAYS read mood | get insertion | put insertion)
                (f, si);
              IF pl :: REF FORMATLIST(piece)
              THEN error(wrongformat); abort
              FI;
              piece:= C a newly created name which is made to refer to the
                yield of an actual-formatlist-declarer and whose
                scope is equal to the scope of 'f' C
                := (count, 1, p OF cl, pl);
              IF count <= 0
              THEN picture found:= TRUE;
                picture:= VOIDPICT(HEAP [1:0] DFRAME:= ());
                cp OF piece:= UPB p OF piece + 1
                # This forces the yielding of a void picture.
                Subsequently, the second insertion of the collitem 'cl'
                will be performed. #
              FI),
            (PICTURE pict):
              (picture found:= TRUE; picture:= pict; cp OF piece += 1)
          ESAC
        FI
      OD
    FI;

```

c) PROC ? update cp =

```
(REF FILE f, BOOL perform insertions, STATUS reading) VOID:
BEGIN REF REF FORMATLIST piece = piece OF f;
  WHILE cp OF piece > UPB p OF piece
  DO
    IF (count OF piece --:= 1) > 0
    THEN cp OF piece:= 1 # repeat this piece #
    ELIF REF FORMATLIST next = next OF piece;
      next :=: REF FORMATLIST(NIL)
    THEN cp OF piece:= 0 # format ended #
    ELSE piece:= next;
      IF perform insertions
      THEN INSERTION extra =
        CASE (p OF piece)[cp OF piece] IN
          (COLLITEM cl): i2 OF cl,
          (FPICT fp): i2 OF fp
        ESAC;
        [1 : UPB extra] SINSERT sinsert;
        staticize frames(extra, sinsert);
        (reading SAYS read mood | get insertion | put insertion)
          (f, sinsert)
      FI;
      cp OF piece += 1
    FI
  OD
END;
```

d) PROC ? staticize frames =

```
(REF [] DFRAME frames, REF [] SFRAME sframes) VOID:
FOR i TO UPB frames
DO sframes[i]:=
  CASE frames[i] IN
    (PROC INT n): n,
    (REF [] CHAR s): s,
    (CHAR a): a
  ESAC
OD;
```

e) PROC ? staticize picture = (DPICT p) SPICT:

```
# 'staticize picture' turns a picture containing dynamic replicators
into one containing only simple (integer) replicators. It also
extracts information needed to build up the character string to be
output from the frames of the picture. This information is
collected in the 'p' field of the static picture that is delivered.
#
BEGIN HEAP [1 : UPB frames OF p] SFRAME sf;
staticize frames(frames OF p, sf);
[1 : CASE type OF p IN 2, 6, 0, 12, 1 OUT 1 ESAC] INT t,
INT count:= 0, rep:= 1, info:= 1, point:= 6, sign:= 2;

FOR i TO UPB t DO t[i]:= 0 OD;

FOR i TO UPB sf
DO
  CASE sf[i] IN
    (INT n): rep:= 0 MAX n,
```

```

(CHAR a):
  ( IF a = "a" OR a = "d" OR a = "z" THEN count += rep
  ELIF a = "+" THEN count += 1; t[sign]:= count
  ELIF a = "-" THEN count += 1; t[sign]:= -count
  ELIF a = "." THEN
    t[info]:= count; count:= 0; info += 2; t[point]:= 1
  ELIF a = "e" THEN
    t[info]:= count; count:= 0; info:= point - 2;
    sign:= point - 1
  ELIF a = "i" THEN
    t[info]:= count; count:= 0; info:= 7; sign:= 8;
    point:= 12
  ELSE SKIP
  FI;
  rep:= 1)
OUT rep:= 1
ESAC
OD;
(UPB t > 0 | t[info]:= count);

(CASE type OF p IN
  # integral # HEAP STRUCT(INT width, sign):= (t[1], t[2]),
  # real      # HEAP STRUCT(INT b, s1, a, e, s2, point):=
              (t[1], t[2], t[3], t[4], t[5], t[6]),
  # boolean  # EMPTY,
  # complex  # HEAP STRUCT(REALPATTERN re, im):=
              ((HEAP STRUCT(INT b, s1, a, e, s2, point):=
                 (t[1], t[2], t[3], t[4], t[5], t[6]),
                 HEAP STRUCT(INT b, s1, a, e, s2, point):=
                 (t[7], t[8], t[9], t[10], t[11], t[12]))),
  # string   # t[1]
OUT
  # bits     # HEAP STRUCT(INT width, radix):=
              (t[1], type OF p - 4)
ESAC, sf)
END;

f) PROC ? put insertion = (REF FILE f, REF [] SINSERT sf) VOID:
  BEGIN INT rep:= 1; ensure state(f, put char status);
  FOR sfp FROM LWB sf TO UPB sf
  DO
    CASE sf[sfp] IN
      (INT count): rep:= count,
      (REF [] CHAR s): (put insert string(f, rep, s); rep:= 1),
      (CHAR a): (alignment(f, rep, a); rep:= 1)
    ESAC
  OD
END;

g) PROC ? put insert string = (REF FILE f, INT rep, REF [] CHAR s) VOID:
  TO rep
  DO
    FOR i TO UPB s
    DO (check pos(f) | put char(f, s[i]) | error(nocharpos); abort)
    OD
  OD;

```

- h) PROC ? get insertion = (REF FILE f, REF [] SINSERT sf) VOID:  
 BEGIN INT rep:= 1; ensure state(f, get char status);  
 FOR sfp FROM LWB sf TO UPB sf  
 DO  
 CASE sf[sfk] IN  
 (INT count): rep:= count,  
 (REF [] CHAR s): (get insert string(f, rep, s); rep:= 1),  
 (CHAR a): (alignment(f, rep, a); rep:= 1)  
 ESAC  
 OD  
 END;
- i) PROC ? get insert string = (REF FILE f, INT rep, REF [] CHAR s) VOID:  
 (CHAR c, si;  
 TO rep  
 DO  
 FOR i TO UPB s  
 DO (check pos(f) | get char(f, c) | error(nocharpos); abort);  
 IF c ≠ (si:= s[i])  
 THEN BOOL mended = (char error mended OF f)(f, c:= si);  
 ensure state(f, get char status);  
 (NOT mended | error(wrongchar); abort)  
 FI  
 OD  
 OD);
- j) PROC ? alignment = (REF FILE f, INT r, CHAR a) VOID:  
 IF a = "x" THEN TO r DO space(f) OD  
 ELIF a = "y" THEN TO r DO backspace(f) OD  
 ELIF a = "l" THEN TO r DO newline(f) OD  
 ELIF a = "p" THEN TO r DO newpage(f) OD  
 ELIF a = "k" THEN set char number(f, r)  
 ELIF a = "q"  
 THEN  
 IF status OF f SAYS read mood  
 THEN get insert string  
 ELSE put insert string  
 FI (f, r, LOC [1 : 1] CHAR:= blank)  
 FI;
- k) PROC ? do fpict = (REF FILE f, FPICT fpict) VOID:  
 BEGIN [1 : UPB i OF fpict] SINSERT si;  
 REF FORMATLIST pl = piece OF f, STATUS reading = status OF f;  
 staticize frames(il OF fpict, si);  
 FORMAT pf = pf OF fpict;  
 (reading SAYS read mood | get insertion | put insertion)(f, si)  
 IF pl ≠: REF FORMATLIST(piece OF f)  
 THEN error(wrongformat); abort  
 FI;  
 REF REF FORMATLIST(piece OF f):= C a newly created name which is made  
 to refer to the yield of an actual-formatlist-declarer and whose  
 scope is equal to the scope of 'f' C  
 := (1, 1, pf, pl)  
 END;



```

1) PROC 7 associate format = (REF FILE f, FORMAT format) VOID:
    piece OF f:= C a newly created name which is made to refer to the yield
        of an actual-reference-to-formatlist-declarer and whose
        scope is equal to the scope of 'f' C
    := C a newly created name which is made to refer to the yield
        of an actual-formatlist-declarer and whose scope is
        equal to the scope of 'f' C
    := (1, 1, format, NIL);

m) PROC 7 edit string =
    (REF FILE f, REF [] SFRAME sf, REF INT sfp, REF [] CHAR s,
    BOOL end) VOID:
    BEGIN INT rep:= 1, j:= LWB s - 1, INT last = UPB s, CHAR k,
    BOOL supp:= FALSE, zs:= TRUE;

    PROC copy = (CHAR c) VOID:
        IF
            IF status OF f SAYS line ok
            THEN TRUE
            ELSE check pos(f)
            FI
        THEN put char(f, c)
        ELSE error(nocharpos); abort
        FI;

    WHILE j < last OR (end AND sfp ≤ UPB sf)
    DO
        CASE sf[sfp] IN
            (INT count): rep:= count,
            (REF [] CHAR s): (put insert string(f, rep, s); rep:= 1),
            (CHAR a):
                IF a = "s" THEN supp:= TRUE
                ELSE
                    IF a = "d" THEN zs:= TRUE;
                    IF supp THEN j += rep
                    ELSE
                        TO rep
                        DO k:= s[j += 1]; copy((k = "." | "0" | k)) OD
                    FI
                    ELIF a = "z" THEN
                        TO rep
                        DO k:= s[j += 1];
                        (zs | (k = "0" | k:= "." | : k ≠ "." | zs:= FALSE));
                        (NOT supp | copy(k))
                        OD
                    ELIF a = "a" THEN
                        IF supp THEN j += rep
                        ELSE TO rep DO copy(s[j += 1]) OD
                        FI
                    ELIF a = "+" OR a = "-" THEN k:= s[j += 1];
                        (zs | (k = "0" | k:= "."
                            | : k ≠ "+" AND k ≠ "-" AND k ≠ "." | zs:= FALSE));
                        copy(k)
                    ELIF a = "." THEN (NOT supp | copy(".")); j += 1
                    ELIF a = "e" OR a = "i" THEN
                        (NOT supp | copy(a)); zs:= TRUE
                
```

```

        ELIF a = "b" THEN copy(s[j +:= 1])
        ELIF a = "r" THEN SKIP
        ELSE alignment(f, rep, a)
        FI;
        supp:= FALSE; rep:= 1
    FI
    ESAC;
    sfp +:= 1
OD
END;

```

```

n) PROC putf = (REF FILE f, [] UNION(OUTTYPE, FORMAT) x) VOID:
FOR k TO UPB x
DO
    IF NOT (status OF f SAYS put char status)
    THEN ensure state(f, put char status)
    FI;
    CASE x[k] IN
        (FORMAT format): associate format(f, format),
        (OUTTYPE ot):
            BEGIN INT j:= 0, PICTURE picture, [] SIMPLOUT y = STRAIGHTOUT ot;
            WHILE (j +:= 1) ≤ UPB y
            DO BOOL incomp:= FALSE;
            get next picture(f, picture);

            INT n =
            CASE picture IN
                (DPICT dp):
                    (picture:= staticize picture(dp);
                    ensure state(f, put char status); 0),
                (CPICT cp): UPB i2 OF cp,
                (GPICT gp): UPB i2 OF gp,
                (VOIDPICT vp): UPB vp
            OUT 0
            ESAC;

            REF [] SINSERT sinsert:= LOC [1 : n] SINSERT;

            CASE picture IN
                (SPICT sp):
                    BEGIN INT sfp:= 1, REF [] SFRAME sf = sframes OF sp;

                    PROC convert L real =
                    (REALPATTERN rp, REF [] CHAR s, REF INT first, last,
                     L REAL x) BOOL:
                    IF INT sign1 = ABS s1 OF rp;
                    INT before = b OF rp - SIGN sign1;
                    e OF rp > 0
                    THEN INT exp:= before, rplace;
                    BOOL neg1 = subfixed(x, a OF rp, exp, s, TRUE);
                    last:= a OF rp + before + point OF rp;
                    first:=
                    IF power10(s, rplace, last)
                    THEN exp +:= 1; s[before]:= ".";
                    s[before+1]:= (before = 0 | rplace:= 1; "0" | "9");
                    last -= 1; 0

```

```

ELSE I
  FI;
round(s, rplace, last);
IF sign1 ≠ 0
THEN s[first -:= 1] :=
      (neg1 | "-" |: s1 OF rp > 0 | "+" | ".");
FI;
# now s[first:last] contains the stagnant part of x #
exp -:= before;
# note that "e" is not explicitly stored in 's' #
INT f := last + 1;
BOOL neg2 = subwhole(exp, f, last + e OF rp, s);
INT sign2 = ABS s2 OF rp;
IF last + SIGN sign2 ≥ f OR
  (sign2 = 0 AND neg2) OR (sign1 = 0 AND neg1)
THEN FALSE
ELSE
  (sign2 ≠ 0
   | s[(sign2 + last) MIN (f - 1)] :=
     (neg2 | "-" |: s2 OF rp > 0 | "+" | ".");
   last +:= e OF rp; TRUE
  )
FI
ELSE # e OF rp = 0 #
  INT bb, rplace;
  BOOL neg1 = subfixed(x, a OF rp, bb, s, FALSE);
  last := bb + a OF rp + point OF rp - 1;
  first := (power10(s, rplace, last) | bb +:= 1; 0 | 1);
  round(s, rplace, last);
  IF INT p = bb - 1 - b OF rp;
    p > 0 OR (sign1 = 0 AND neg1)
  THEN FALSE
  ELSE
    (sign1 ≠ 0
     | s[(sign1 + p) MIN (first - 1)] :=
       (neg1 | "-" |: s1 OF rp > 0 | "+" | ".");
     first := p + 1; TRUE
    )
  FI
FI};

PROC edit L real = (L REAL x, REALPATTERN rp) VOID:
  IF INT u, v;
    IF e OF rp > 0
      THEN u := -1;
        v := b OF rp + a OF rp + 2 + L exp width MAX e OF rp
      ELSE INT b = log10(x);
        u := 0 MIN (b - b OF rp) - 1;
        v := b + a OF rp + 2
      FI;
    [u : v] CHAR s, INT first, last;
    convert L real(rp, s, first, last, x)
  THEN edit.string(f, sf, sfp, s[first : last], TRUE)
  ELSE incomp := TRUE
  FI};

```

```

PROC edit L compl = (L COMPL z, COMLPATTERN cp) VOID:
  IF INT u1, v1, u2, v2;
    IF e OF re OF cp > 0
      THEN u1 := -1;
        v1 := b OF re OF cp + a OF re OF cp + 2 +
              L exp width MAX e OF re OF cp
      ELSE INT b = log10(re OF z);
        u1 := 0 MIN (b - b OF re OF cp) - 1;
        v1 := b + a OF re OF cp + 2
      FI;
    IF e OF im OF cp > 0
      THEN u2 := -1;
        v2 := b OF im OF cp + a OF im OF cp + 2 +
              L exp width MAX e OF im OF cp
      ELSE INT b = log10(im OF z);
        u2 := 0 MIN (b - b OF im OF cp) - 1;
        v2 := b + a OF im OF cp + 2
      FI;
    [u1 : v1] CHAR s re, [u2 : v2] CHAR s im,
    INT f re, l re, f im, l im;
    convert L real(re OF cp, s re, f re, l re, re OF z) AND
    convert L real(im OF cp, s im, f im, l im, im OF z)
  THEN edit string(f, sf, sfp, s re[f re : l re], FALSE);
    edit string(f, sf, sfp, s im[f im : l im], TRUE)
  ELSE incomp := TRUE
  FI;

```

CASE p OF sp IN

```

(INTPATTERN ip):
  (y[j]
  | (L INT i):
    IF INT upbs = L int width MAX width OF ip;
      [0 : upbs] CHAR s, INT first := 0;
      BOOL neg = subwhole(i, first, upbs, s);
      INT p = upbs - width OF ip,
          abssign = ABS sign OF ip;
      p + SIGN abssign ≥ first OR (abssign = 0 AND neg)
    THEN incomp := TRUE
    ELSE
      (abssign ≠ 0
      | s[(abssign + p) MIN (first - 1)] :=
        (neg | "-" | : sign OF ip > 0 | "+" | "."));
      edit string(f, sf, sfp, s[p + 1 : ], TRUE)
    FI;
  | incomp := TRUE),

(REALPATTERN rp):
  (y[j]
  | (L REAL r): edit L real(r, rp);
  | (L INT i): edit L real(i, rp);
  | incomp := TRUE),

```

```

(BOOLPATTERN bp):
  (y[j]
  | (BOOL b):
      edit string(f, sf, sfp,
                  LOC [1:1] CHAR:= (b | flip | flop),
                  TRUE)
  | incomp:= TRUE),

(COMPLPATTERN cp):
  (y[j]
  | { (L COMPL z): edit L compl(z, cp) },
  | { (L REAL r): edit L compl(r, cp) },
  | { (L INT i): edit L compl(i, cp) }
  | incomp:= TRUE),

(STRINGPATTERN stp):
  (y[j]
  | (CHAR c):
      IF stp = 1
      THEN edit string(f, sf, sfp,
                      LOC [1:1] CHAR:= c, TRUE)
      ELSE incomp:= TRUE
      FI,
  | ([ CHAR t):
      IF stp = UPB t - LWB t + 1
      THEN edit string(f, sf, sfp,
                      LOC [1:stp] CHAR:= t, TRUE)
      ELSE incomp:= TRUE
      FI
  | incomp:= TRUE),

(BITSPATTERN bp):
  (y[j]
  | { (L BITS lb):
      IF INT upbs = L bits width MAX width OF bp;
      [1 : upbs] CHAR s, L INT n:= ABS lb, INT first:= upbs;
      WHILE s[first]:= dig char(S(n MOD K radix OF bp));
      n OVERAB K radix OF bp; n ≠ L 0
      DO first -= 1 OD;
      INT p = upbs - width OF bp + 1;
      p > first
      THEN incomp:= TRUE
      ELSE
      WHILE p < first DO s[first -= 1]:= "." OD;
      edit string(f, sf, sfp, s[p : ], TRUE)
      FI }
  | incomp:= TRUE)
ESAC;

```

```

IF incomp
THEN sfp:= UPB sf;
  WHILE
    CASE sf[sfp] IN
      (CHAR a): char in bag(a, xylpkq)
    OUT TRUE
    ESAC
  DO sfp -= 1 OD;
  sinsert:= sf[sfp + 1 : ]
  # the last insertion of 'sf' is searched for #
FI
END,

```

(CPICT choice):

```

BEGIN [1 : UPB il OF choice] SINSERT si;
  staticize frames(il OF choice, si);
  put insertion(f, si);
  INT l = CASE type OF choice IN
    # boolean #
    (y[j] | (BOOL b): (b | 1 | 2)
      | incomp:= TRUE; SKIP),
    # integral #
    (y[j] | (INT i): i
      | incomp:= TRUE; SKIP)
  ESAC;
  IF NOT (incomp:= incomp OR l <= 0
          OR l > UPB c OF choice)
  THEN INSERTION cl = (c OF choice)[l];
    [1 : UPB cl] SINSERT ci;
    staticize frames(cl, ci);
    put insertion(f, ci)
  FI;
  staticize frames(i2 OF choice, sinsert)
END,

```

(FPICT fpict): do fpict(f, fpict),

(GPICT gpict):

```

BEGIN [1 : UPB il OF gpict] SINSERT si;
  staticize frames(il OF gpict, si);
  staticize frames(i2 OF gpict, sinsert);
  INT n = UPB spec OF gpict; [1 : n] INT s;
  FOR i TO n DO s[i]:= (spec OF gpict)[i] OD;
  put insertion(f, si);
  IF n = 0 THEN put(f, y[j])
  ELSE
    NUMBER yj = (y[j]
      | {L INT i}: i,
      | {L REAL r}: r
      | incomp:= TRUE; SKIP);
  
```

```

IF NOT incomp
THEN
  CASE n IN
    put(f, whole(yj, s[1])),
    put(f, fixed(yj, s[1], s[2])),
    put(f, float(yj, s[1], s[2], s[3]))
    # For optimization purposes, one might want to
    generate different code here. #
  ESAC
  FI
  FI
  END,

  (VOIDPICT v):
  (j -= 1; staticize frames(v, sinsert))
  ESAC;

  IF incomp
  THEN ensure state(f, put char status);
    BOOL mended = (value error mended OF f)(f);
    ensure state(f, put char status);
    (NOT mended | put(f, y[j]); error(wrongval); abort)
  FI;
  put insertion(f, sinsert);
  IF cp OF piece OF f > UPB p OF piece OF f
  THEN update cp(f, TRUE, put char status)
  FI
  OD
  END
  ESAC
  OD;

```

```

o) PROC indit string =
  (REF FILE f, REF [] SFRAME sf, REF INT sfp, REF [] CHAR s,
   INT sign, radix, BOOL end) VOID:
  BEGIN
    CHARBAG digits = ( radix = 10 | radix10digit
                      |: radix = 2 | radix 2digit
                      |: radix = 4 | radix 4digit
                      |: radix = 8 | radix 8digit
                      | radix16digit);
    CHARBAG digits and space = digits + STRINGTOBAG("."),
      digits and signspace = digits + STRINGTOBAG("+-.");

    PRIO ! = 8;
    OP ! = (CHARBAG s, CHAR c) CHAR:
      IF CHAR k;
        IF (status OF f SAYS line ok | TRUE | check pos(f))
          THEN get char(f, k)
          ELSE error(nocharpos); abort
          FI;
        char in bag(k, s)
      THEN k
      ELSE k:= c; BOOL mended = (char error mended OF f)(f, k);
        ensure state(f, get char status);
        IF (mended | char in bag(k, s) | FALSE)
          THEN k
          ELSE error(wrongchar); c
          FI
      FI;

    INT index:= ABS sign, rep:= 1,
    BOOL sign found:= FALSE, first space:= FALSE, supp:= FALSE;
    CHARBAG allowed:= CASE sign + 2 IN
      # "-"-frame # (first space:= TRUE; signspace),
      # no frame # (sign found:= TRUE; digits and space),
      # "+"-frame # signspace
      ESAC;

    WHILE index < UPB s OR (end AND sfp ≤ UPB sf)
    DO
      CASE sf[sfp] IN
        (INT count): rep:= count,
        (REF [] CHAR s): (get insert string(f, rep, s); rep:= 1),
        (CHAR a):
          IF a = "s" THEN supp:= TRUE
          ELSE
            IF a = "d" THEN
              TO rep
              DO s[index +:= 1]:= (supp | "0" | digits ! "0") OD;
              allowed:= digits and space
            ELIF a = "z" OR a = "+" OR a = "-" THEN
              TO rep
              DO
                IF sign found
                THEN s[index +:= 1]:=
                  (supp | "0"
                   | CHAR c = allowed ! "0");

```



```

                                (c ≠ "." | allowed:= digits; c | "0")
ELSE CHAR c:= (a = "+" | plusminus | allowed) ! "+";
  IF c = "." AND a = "z"
  THEN (first space | allowed:= digits and signspace;
        first space:= FALSE);
        c:= "0"
  ELSE sign found:= TRUE; allowed:= digits;
        (c = "." | c:= "+")
  FI;
  (c = "+" OR c = "-" | s[1]:= c | s[index += 1]:= c)
FI
OD
ELIF a = "." THEN
  (NOT supp | point ! ".")
ELIF a = "e" THEN
  (NOT supp | times ten to the power ! "e")
ELIF a = "i" THEN
  (NOT supp | plus i times ! "i")
ELIF a = "r" THEN SKIP
ELIF a = "b" THEN
  s[index += 1]:= flipflop ! flop
ELIF a = "a" THEN
  TO rep
  DO s[index += 1]:=
      (supp | "."
       | CHAR c;
        (check pos(f) | get char(f, c)
         | error(nocharpos); abort);
        c)
  OD
  ELSE alignment(f, rep, a)
  FI;
  rep:= 1; supp:= FALSE
FI
ESAC;
sfp += 1
OD
END;

```

```

p) PROC getf = (REF FILE f, [] UNION(INTYPE, FORMAT) x) VOID:
  FOR k TO UPB x
  DO
    IF NOT (status OF f SAYS get char status)
    THEN ensure state(f, get char status)
    FI;
    CASE x[k] IN
      (FORMAT format): associate format(f, format),
      (INTYPE it):
        BEGIN INT j:= 0, PICTURE picture, [] SIMPLIN y = STRAIGHTIN it;
          WHILE (j += 1) ≤ UPB y
          DO BOOL incomp:= FALSE;
            get next picture(f, picture);

```

```

INT n =
CASE picture IN
  (DPICT dp): (picture:= staticize picture(dp);
              ensure state(f, get char status); 0),
  (CPICT cp): UPB i2 OF cp,
  (GPICT gp): UPB i2 OF gp,
  (VOIDPICT vp): UPB vp
OUT 0
ESAC;

REF [] SINSERT sinsert:= LOC [1 : n] SINSERT;

CASE picture IN
(SPICT sp):
  BEGIN INT sfp:= 1, REF [] SFRAME sf = sframes OF sp;

  {PROC convert L real =
    (REALPATTERN rp, REF L REAL rr, BOOL end) BOOL:
  BEGIN INT upbs = a OF rp + b OF rp, exp width = e OF rp;
    [1 : upbs] CHAR s;
    indit string(f, sf, sfp, s, SIGN s1 OF rp, 10,
                exp width = 0 AND end);
    BOOL ok:= TRUE, INT first:= (s1 OF rp = 0 | 1 | 2);
    WHILE first < upbs AND s[first] = "0"
    DO first += 1 OD;
    INT last = (first + L real width) MIN upbs;
    INT exp:= b OF rp - last;
    IF exp width > 0
    THEN [1 : exp width] CHAR s;
      indit string(f, sf, sfp, s, SIGN s2 OF rp, 10, end);
      INT e:= 0;
      FOR i FROM (s2 OF rp = 0 | 1 | 2) TO exp width
      WHILE ok
      DO ok:= e ADD char dig(s[i]) OD;
      IF ok:= ok AND ABS exp <= max int - e
      THEN exp += (s[i] = "-" | -e | e)
      FI
    FI;
    IF ok
    THEN ok:=
      string to L real(s[first:last], exp, s[i] = "-", rr)
    FI;
    ok
  END};

CASE p OF sp IN

  (INTPATTERN ip):
    (y[j]
    | {(REF L INT ii):
      BEGIN [1 : width OF ip] CHAR s;
        indit string(f, sf, sfp, s, sign OF ip, 10, TRUE);
        BOOL ok:= TRUE, L INT j:= 0;
        FOR i FROM (sign OF ip = 0 | 1 | 2)
        TO width OF ip WHILE ok
        DO ok:= j ADD char dig(s[i]) OD;

```

```

        (ok | ii:= (s[l] = "-" | -j | j));
        incomp:= NOT ok
    END‡
| incomp:= TRUE),

(REALPATTERN rp):
    (y[j]
    | ‡(REF L REAL rr):
        incomp:= NOT convert L real(rp, rr, TRUE)‡
    | incomp:= TRUE),

(BOOLPATTERN bp):
    (y[j]
    | (REF BOOL bb):
        BEGIN [l : l] CHAR s;
            indit string(f, sf, sfp, s, 0, 0, TRUE);
            bb:= s[l] = flip
        END
    | incomp:= TRUE),

(COMPLPATTERN cp):
    (y[j]
    | ‡(REF L COMPL zz):
        (L COMPL z; BOOL ok:=
            convert L real(re OF cp, re OF z, FALSE);
            ok:= ok AND
            convert L real(im OF cp, im OF z, TRUE);
            (ok | zz:= z); incomp:= NOT ok)‡
    | incomp:= TRUE),

(STRINGPATTERN stp):
    (y[j]
    | (REF CHAR cc):
        IF stp = l
        THEN indit string(f, sf, sfp, REF [] CHAR(cc),
            0, 0, TRUE)

        ELSE incomp:= TRUE
        FI,
    (REF [] CHAR ss):
        IF UPB ss - LWB ss + 1 = stp
        THEN indit string(f, sf, sfp, s, 0, 0, TRUE)
        ELSE incomp:= TRUE
        FI,
    (REF STRING ss):
        BEGIN [l : stp] CHAR s;
            indit string(f, sf, sfp, s, 0, 0, TRUE);
            ss:= s
        END
    | incomp:= TRUE)

```

```

(BITSPATTERN bp):
  (y[j]
  | { (REF L BITS lb):
    BEGIN [l : width OF bp] CHAR s,
      INT radix = radix OF bp;
      indit string(f, sf, sfp, s, 0, radix, TRUE);
      INT r = (radix = 2 | 1 | : radix = 4 | 2
              | : radix = 8 | 3 | 4),
      INT w:= width OF bp, n:= 0, d,
      [l : L bits width] BOOL b;
      FOR i FROM L bits width BY -1 TO 1
      DO
        IF n = 0
          THEN d:= (w > 0 | char dig(s[w]) | 0);
            w -= 1; n:= r
          FI;
          b[i]:= ODD d; d:= d OVER 2; n -= 1
        OD;
        lb:= L bits pack(b)
      END;
  | incomp:= TRUE)
ESAC;
END,

(CPICT choice):
  BEGIN [l : UPB il OF choice] SINSERT si;
    staticize frames(il OF choice, si);
    get insertion(f, si);
    INT c = c OF cpos OF f, CHAR kk,
    INT k:= 0, BOOL found:= FALSE;
    WHILE k < UPB c OF choice AND NOT found
    DO INSERTION ck = (c OF choice)[k += 1];
      [l : UPB ck] SINSERT si;
      staticize frames(ck, si);
      ensure state(f, get char status);
      BOOL bool:= TRUE, INT rep:= 1;
      FOR i TO UPB si WHILE bool
      DO
        CASE si[i] IN
          (INT count): rep:= count,
          (REF [] CHAR ss):
            (FOR j TO rep WHILE bool
            DO
              FOR l TO UPB ss
                WHILE bool:= bool AND status OF f SAYS line ok
                DO get char(f, kk); bool:= kk = s[l] OD
              OD;
              rep:= 1)
            ESAC
          OD;
        (NOT (found:= bool) | set char number(f, c))
      OD;

```

```

IF NOT found THEN incomp:= TRUE
ELSE
  CASE type OF choice IN
  # boolean #
    (y[j]
     | (REF BOOL b): b:= k = 1
     | incomp:= TRUE),
  # integral #
    (y[j]
     | (REF INT i) i:= k
     | incomp:= TRUE)
  ESAC
FI;
staticize frames(i2 OF choice, sinsert)
END,

(FPICT fpict): do fpict(f, fpict),

(GPICT gpict):
BEGIN [1 : UPB i1 OF gpict] SINSERT si;
  staticize frames(i1 OF gpict, si);
  staticize frames(i2 OF gpict, sinsert);
  get insertion(f, si); get(f, y[j])
END,

(VOIDPICT v):
(j -:= 1; staticize frames(v, sinsert))

ESAC;

IF incomp
THEN ensure state(f, get char status);
  BOOL mended = (value error mended OF f)(f);
  ensure state(f, get char status);
  (NOT mended | error(wrongval); abort)
FI;
get insertion(f, sinsert);
IF cp OF piece OF f > UPB p OF piece OF f
THEN update cp(f, TRUE, get char status)
FI
OD
END
ESAC
OD;

```

### 11.3. EFFICIENCY

Considerable attention has been paid to the efficiency of formatted transput. The data structures have been chosen with great care, so as to minimize both space and time. The efficiency may be further increased by carefully rewriting the routines 'edit string' and 'indit string' in machine code. {This may well speed up things by a factor of 2!}

## 12. BINARY TRANSPUT

### 12.1. DEVIATIONS

- {D} Rather than assuming that binary transput goes via elementary values of the mode CHAR, a special mode BINCHAR is used as a primitive in this model.
- {D} In the Revised Report, the number of characters to be input to a name N is determined as follows:
  - Let 'yj' be the value referred to by N;
  - The number of characters that is input is equal to the number of characters output by 'put bin(f, yj)' {i.e., 'UPB to bin(f, yj)'}. It is anticipated that in an actual implementation there will be smarter ways to determine that number. Therefore, a separate routine 'bin length' (d) has been introduced.

### 12.2. NEW DEFINITION

In binary transput, the values obtained by straightening the elements of a data list are transput, via the specified file, one after the other. The manner in which such a value is stored is defined only to the extent that a value of mode M (being some mode from which that specified by SIMPLOUT is united) output at a given position may subsequently be re-input from that same position to a name of mode 'REFERENCE TO M'. Note that, during input to the name referring to a multiple value, the number of elements read will be the existing number of elements referred to by that name.

The current position is advanced after each value by a suitable amount and, at the end of each line or page, the appropriate event routine is called, and next, if this returns false, the next good character position is found.

For binary output, 'put bin' (e) and 'write bin' (section 10.5.1 of the Revised Report) may be used and, for binary input, 'get bin' (f) and 'read bin' (section 10.5.1 of the Revised Report).

#### a) MODE ? BINCHAR =

C The elementary mode of binary transput; each value to be transput is so via some 'row of BINCHAR', the length of the row being determined by the file on which the transput takes place, the mode of the value to be transput (and its length in case of a multiple value). C;

#### b) PROC ? to bin = (REF FILE f, SIMPLOUT x) [] BINCHAR:

C The lower bound of the resulting multiple value is 1, the upper bound depends on 'f' and on the mode and the value of 'x'; furthermore, x = from bin(f, x, to bin(f, x)). C;

#### c) PROC ? from bin = (REF FILE f, SIMPLOUT y, [] BINCHAR c) SIMPLOUT:

C A value, if one exists, of the mode of the value yielded by 'y', such that c = to bin(f, from bin(f, y, c)). If such a value does not exist, an error message 'wrongbin' is given and the program is aborted. C;

d) PROC ? bin length = (REF FILE f, SIMPLIN y) INT:

C The upper bound of the multiple value which is needed to input a value into 'y'. C;

# the following ALGOL-68 unit will do:

```
(SIMPLOUT yj =
CASE y IN
  †(REF L INT i): i‡,
  †(REF L REAL r): r‡,
  †(REF L COMPL z): z‡,
  (REF BOOL b): b,
  †(REF L BITS lb): lb‡,
  (REF CHAR c): c,
  (REF [] CHAR s): s,
  (REF STRING ss): ss
ESAC;
UPB to bin(f, yj)) #
```

e) PROC put bin = (REF FILE f, [] OUTTYPE ot) VOID:

```
FOR k TO UPB ot
DO
  IF NOT (status OF f SAYS put bin status)
  THEN ensure state(f, put bin status)
  FI;
  [] SIMPLOUT y = STRAIGHTOUT ot[k];
  FOR j TO UPB y
  DO [] BINCHAR bin = to bin(f, y[j]);
    FOR i TO UPB bin
    DO next pos(f); (put bin char OF f)(f, bin[i]);
      IF status OF f SAYS logical pos not ok
      THEN set logical pos(f)
      FI;
      test line end(f)
    OD
  OD
OD;
```

f) PROC get bin = (REF FILE f, [] INTYPE it) VOID:

```
FOR k TO UPB it
DO
  IF NOT (status OF f SAYS get bin status)
  THEN ensure state(f, get bin status)
  FI;
  [] SIMPLIN y = STRAIGHTIN it[k];
  FOR j TO UPB y
  DO [1 : bin length(f, y[j])] BINCHAR bin;
    FOR i TO UPB bin
    DO next pos(f); (get bin char OF f)(f, bin[i]);
      test line end(f);
      IF status OF f SUGGESTS lfe in current line
      THEN test logical file end(f)
      FI
    OD
  OD;
```



```

CASE y[j] IN
  ‡(REF L INT ii):
    (from bin(f, ii, bin) | (L INT i): ii:= i)‡,
  ‡(REF L REAL rr):
    (from bin(f, rr, bin) | (L REAL r): rr:= r)‡,
  ‡(REF L COMPL zz):
    (from bin(f, zz, bin) | (L COMPL z): zz:= z)‡,
  (REF BOOL bb):
    (from bin(f, bb, bin) | (BOOL b): bb:= b),
  ‡(REF L BITS lb):
    (from bin(f, lb, bin) | (L BITS b): lb:= b)‡,
  (REF CHAR cc):
    (from bin(f, cc, bin) | (CHAR c): cc:= c),
  (REF [] CHAR ss):
    (from bin(f, ss, bin) | ([] CHAR s): ss:= s),
  (REF STRING ss):
    (from bin(f, ss, bin) | (STRING s): ss:= s)
ESAC
OD
OD;

```

## REFERENCES

- [1] WIJNGAARDEN, A. VAN, et al (eds.), Revised Report on the Algorithmic Language ALGOL 68, Acta Informatica 5 (1975), pp 1-236.
- [2] WOODWARD, P.M. & S.G. BOND, ALGOL 68-R Users Guide, Royal Radar Establishment, Malvern, England, 1975.
- [3] ALGOL 68 Version I Reference Manual, Control Data Services B.V., Rijswijk, The Netherlands, 1976.
- [4] HILL, U., H. SCHEIDIG & H. WOESSNER, An ALGOL 68 Compiler, Technische Universität München and University of British Columbia, 1972.
- [5] ROBERTSON, A. & G.E. HEDRICK, A Portable Compiler for an ALGOL 68 Subset, in G.E. Hedrick (ed.), Proceedings of the 1975 International Conference on ALGOL 68, Oklahoma State University, Stillwater, Oklahoma, June 10-12, 1975, pp 59-64.
- [6] BERRY, R.D., A practical Implementation of Formatted Transput in ALGOL 68, MS Thesis, Oklahoma State University, Stillwater, Oklahoma, July 1973.
- [7] LEROY, A., et al, On the Adequacy of the ALGOL 68 Environment Compared with an Existing Current Operating System and Problems of I/O Implementation, in G.E. Hedrick (ed.), Proceedings of the 1975 International Conference on ALGOL 68, Oklahoma State University, Stillwater, Oklahoma, June 10-12, 1975, pp 202-220.
- [8] BROUGHTON, C.G. & C.M. THOMSON, Aspects of Implementing an ALGOL 68 Student Compiler, in G.E. Hedrick (ed.), Proceedings of the 1975 International Conference on ALGOL 68, Oklahoma State University, Stillwater, Oklahoma, June 10-12, 1975, pp 23-38.
- [9] THOMSON, C.M., A description of the FLACC Transput System, preliminary version, Edmonton, Canada, August 1977.
- [10] THOMSON, C.M. & C.G. BROUGHTON, A description of a New Transput System, preliminary version, Edmonton, Canada, August 1977.
- [11] CMU ALGOL-68 Transput, Users's Guide, Carnegie Mellon University.
- [12] ALGOL68S implementation -- file system, Carnegie Mellon University.
- [13] FISHER, R., private communication.
- [14] VLIET, J.C. VAN, Compilation of problems and errors in section 10.3 of the Revised Report, Mathematical Centre, June 1977.
- [15] VLIET, J.C. VAN, On the ALGOL 68 Transput Conversion Routines, ALGOL Bulletin 41, July 1977, pp 10-24.
- [16] VLIET, J.C. VAN, Towards an Implementation-Oriented Definition of the ALGOL 68 Transput, Report IW90, Mathematical Centre, October 1977.
- [17] LINDSEY, CH., Draft Commentaries on Transput, CHL 78-06-23.

Alphabetic listing of all defining occurrences of mode-indications and identifiers. If a mode-indication or identifier is prefixed with an \*, this means that it is not (completely) defined in ALGOL 68.

56	ADD	46	ensure line
84	alignment	45	ensure logical file
38	ANDAB	46	ensure page
31	associate	46	ensure physical file
38	associate end	45	ensure state
85	associate format	13	estab possible
38	associate status	30	establish
69	*back char	38	establish status
44	backspace	10	EXCEEDS
23	backspace possible	55	*EXPLENGTH
10	BEYOND	33	false
100	*bin length	22	FILE
38	bin mood	33	*file available
22	bin possible	52	fixed
38	bin to char not possible	69	flipflop
99	*BINCHAR	53	float
75	BITSPATTERN	73	FORMAT
10	*BOOK	81	FORMATLIST
10	*book in system	74	FPICT
74	BOOLPATTERN	99	*from bin
22	*BUFFER	65	get
38	buffer filled	100	get bin
23	chan	16	*get bin char OF f
13	CHANNEL	38	get bin status
56	char dig	69	get char
69	*char in bag	16	*get char OF f
56	char in string	38	get char status
38	char mood	84	get insert string
37	char number	84	get insertion
38	char to bin not possible	81	get next picture
69	*CHARBAG	22	get possible
47	check pos	93	getf
34	*close	74	GPICT
38	closed	33	*idf ok
73	COLLECTION	92	indit string
73	COLLITEM	14	*init buffer OF f
74	COMPLPATTERN	74	INSERTION
22	compressible	74	INTPATTERN
10	*construct book	59	*INTYPE
13	CONV	57	*L exp width
74	CPICT	56	L int width
31	create	57	*L real width
11	*default idf	38	lfe in current line
74	DFRAME	38	line end
55	dig char	37	line number
84	do fpict	38	line ok
16	*do newline OF chan OF f	35	*lock
17	*do newpage OF chan OF f	55	*log10
17	*do reset OF chan OF f	38	logical file ended
17	*do set OF chan OF f	38	logical pos not ok
73	DPICT	38	logical pos ok
85	edit string	23	*make conv

3	make term	47	reset
5	MAX	22	reset possible
.5	MIN	55	round
39	mind logical pos	38	SAYS
38	mood part	35	*scratch
45	newline	47	set
45	newpage	34	set bin mood
47	next pos	34	set char mood
38	not lfe in current line	48	set char number
38	not set poss	11	*set logical pos
52	NUMBER	23	set possible
24	on char error	34	set read mood
24	on format end	34	set write mood
24	on line end	74	SFRAME
23	on logical file end	59	SIMPLIN
23	on page end	59	SIMPLOUT
23	on physical file end	74	SINSERT
24	on value error	44	space
31	open	73	SPICT
38	open status	13	*stand back channel
38	opened	13	*stand in channel
38	ORAB	13	*stand out channel
59	*OUTTYPE	13	standconv
38	page end	45	state
37	page number	82	staticize frames
38	page ok	82	staticize picture
38	physical file end	37	STATUS
38	physical file ok	59	*STRAIGHTIN
73	PICTURE	59	*STRAIGHTOUT
69	plus i times	56	*string to L real
10	POS	75	STRINGPATTERN
55	power10	69	*STRINGTOBAG
11	*pseudo book	54	*subfixed
62	put	53	subwhole
100	put bin	38	SUGGESTS
15	*put bin char OF f	39	test line end
38	put bin status	39	test logical file end
64	put char	69	times ten to the power
15	*put char OF f	99	*to bin
38	put char status	82	update cp
83	put insert string	74	VOIDPICT
83	put insertion	52	whole
22	put possible	15	*write buffer OF f
86	putf	38	write mood
69	radix 2digit	38	write to read not possible
69	radix 4digit	69	xylpkq
69	radix 8digit	69	*+
69	radix10digit		
69	radix16digit		
15	*read buffer OF f		
38	read mood		
38	read or write mood		
38	read to write not possible		
74	REALPATTERN		
24	*reidf		
22	reidf possible		