Draft 5
TM-2260/001/00
Clark Weissman

LISP II PROJECT

Memo No. 16

LISP II Input/Output

ABSTRACT

This document specifies the philosophy and mechanics of LISP II I/O. It supersedes LISP II Project Memos 5 and 8.

INDEX

INDEX (Continued)

primitive to Change OPEN Parameters that are legal i.e. protect.

1. INTRODUCTION

This memo specifies the philosophy and mechanics of input/output in LISP II. The design is derived from ideas proposed in earlier memos, but principally from implemented ideas contained in the Q-32 LISP 1.5 I/O package.

1.1 DESIGN OBJECTIVES

LISP II is designed for maximum on-line interactive operation within a time-sharing system environment. The principal communication and control device will be a reactive typewriter or on-line keyboard. However, to enable practical operation with quantities of information than can be conveniently accommodated by keyboard devices, disc, tape, and other bulk memory stores must be accessable to LISP II users, and in a uniform and convenient form. Furthermore, the language implementation must not be biased toward any particular I/O configuration by preempting, a priori, valuable core storage for particular device buffering. Finally, I/O must be machine independent from the user's point of view, thereby allowing program compatibility between various LISP II implementations.

1.2 DESIGN CHARACTERISTICS

In keeping with the functional logic of LISP, I/O will be performed by evaluation of pseudo functions and LAP primitives; but evaluation for effect rather than for value. The effect will be to configure or manipulate internal I/O data structures. Time-sharing systems recognize the organizational elegance of blocking data into files, and they provide extensive I/O facilities for I/O file management. LISP II I/O will capitalize on these facilities by representing its I/O data as files. By so interfacing with the monitoring environment, LISP II gains the advantage of simplified I/O mechanics, and standardization of I/O functions.

1.2.1 Files

A file is the principal data structure for addressing LISP II input and output data. The user, through the provided I/O functions, will be able to create, delete, position, select, and read or print files dynamically, at run-time,

for the complete spectrum of physical devices available to his system. A file is device-dependent, but direction-independent, and may be used as both an input and an output file; and with caution, both simultaneously.

1.2.2 Records

Files are blocked into records that are themselves blocked into lines. Records and lines are of variable size and number, depending on the user's choice and the physical device being addressed. Only one record for each different file can be in core at any one time to reduce buffer storage overhead. One may consider the data record as a "cursor" positioned appropriately over the file. For symbolic data files the record is structured internally as a LISP II string. Binary data files are structured as octal arrays.

1.2.3 Lines

Blocking of symbolic records into lines is a necessary concession for format compatibility with printing hardware. Thus, all symbolic data files will be formatted and can be listed, subject to time-sharing monitor limitations. Line length is fixed for each file when it is created. The number of characters may be specified by the user, within meaningful limits for the device in question, or by default will be assumed to be 72, the maximum number compatible with all printer devices.

Since line formatting is usually controlled by special characters imbedded in the data, and since there are no standards for these characters for different devices at a given installation, line formatting will be controlled internally by LISP II with an appropriate collection of control words stored in an integer array associated with each file. These control words will be used to remove or insert control characters of the appropriate type for the device being accessed at such times as the string representation of the data record is transferred into or out of core. We call this "post string processing" and it is the only I/O area dependent upon machine, monitor, or device configuration. Post-string-processing permits us to represent data records as strings that

contain no control characters, and thus are device independent. By segregating control from data, device independent record-strings provide a high degree of processing freedom necessary for uniform and powerful I/O.

1.2.4 Words

Binary records are not blocked. Such records are, however, composed of machine words. By equating words with lines, file control may be likened to that for symbolic records. Externally, binary data records do not contain any control information.

2. FILE CREATION: OPEN

A file may be created at any time by evaluating the function OPEN. OPEN establishes all necessary communication linkages between LISP II and the time-sharing monitor. In particular it does the following things, though not necessarily in the order given:

1. It creates an internal STRING, of size sufficient to contain one data record for the designated file.

2. It creates an INTEGER array and sets its contents with format control information for the file.

3. It creates a FORMAL array and sets its contents with necessary primitives for formatting and post-string-processing.

4. It declares information to the time-sharing monitor for it to allocate and establish communication linkages with the external storage medium designated for the file.

5. It appends to the property list of the file's name, under the property "I/O," a list of descriptive information about the file.

6. It maintains a list of all OPENed file names, as the value of FLUID variable FILES., which it returns as the value of OPEN and uses to check for redundant or conflicting file names.

OPEN is a function of two arguments and has the form,

(OPEN file name, file description)

2.1 FILE NAME

The first argument of OPEN is the name of the file being created. This name must be a quoted LISP II identifier. The first 6 characters of the name will be used by LISP II as the internal name for the file in establishing its communication linkages with the time-sharing monitor. Therefore, the first 6 characters must be unique among all previously OPENed files. For file names less than 6 characters, OPEN will still use a 6 character name by filling the remaining character positions with blanks.

2.2 FILE DESCRIPTION

The second argument of OPEN is a list of flags and dotted pairs of attributes and values, in property list format, completely describing the file in all its dimensions. A file's dimensions are given by its Unit, Form, Connection, Protection, Identification, and Format.

Except for special, non-standard I/O operations, the user need not concern himself with the construction of a file description list, for he may use one of a set of reserved I/O identifiers for the second argument of OPEN. There is one identifier for each type of I/O unit available to the system, and each evaluates to a preset file description list for that unit type. This file description list is sufficient for most standard I/O operations.

The current reserved I/O identifiers are:

TTY.
DISC.
TAPE.
CORE.
CRT.

2.2.1 Unit

A file is unit dependent, as LISP II uses the unit type of a file for establishing the proper communication linkages with the time-sharing monitor, and for setting up the correct post-string-processing for the file. Thus, one element of the file description list must be a dotted pair designating the unit type. For example, the dotted pair

(UNIT . TTY)

will designate the on-line typewriter or Teletype as the unit for a particular file by the presence of the identifier TTY as the CDR of the dotted pair. This dotted pair may exist as any top-level element of the file description list because a search is performed on the elements of file description lists for an element whose CAR is the identifier UNIT . Similarly,

(UNIT . DISC)
(UNIT . TAPE)
(UNIT . CORE)
(UNIT . CRT)

will designate disc, tape, core, and CRT (SCOPE display), respectively, as the unit for a file.

The ability to specify core as a unit enriches LISP II I/O. With this capability it is possible to print input files, read output files, copy I/O files, and perform text formatting completely within LISP II source language, to name but a few of the possible applications. (It is also possible to perform these techniques without core as a unit; however, the flexibility is provided with a minimum of cost.)

2.2.2 Form

Symbolic data will be represented internally in LISP II as 8-bit ASCII characters. However, not all external media use this standard and conversion will be required. These conversions will be performed by primitives as part of post-string-processing.

For Q-32 LISP II, we will need at least two types of symbolic converters, ~~and one for binary data~~. These converters are designated by the value of the attribute FORM in the file description list. These values may be ASCII, BCD, or BINARY, which specify the FORM of the data on the external device. Q-32 Teletypes use a 12-bit representation of ASCII 8-bit code. The dotted pair

(FORM . ASCII)

will be used to call forth a 12-bit to 8-bit (for input) or a 8-bit to 12-bit (for output) converter. Similarly,

(FORM . BCD)

will be used to call forth a 6-bit to 8-bit, or 8-bit to 6-bit converter. The specific conversions are given in section 8.

(FORM . BINARY)

specifies a binary file, and no conversion is necessary, as a simple binary record will be transferred to and from the file.

2.2.3 Connection

When OPEN establishes communication linkages with the time-sharing monitor, it is necessary for LISP to tell the monitor how to "connect" with the external file. If the file is a new file, that is, one being created by LISP, the monitor will allocate storage for the file on the requested external device, and connect LISP to the file so created. On the other hand, if the file already exists and is in the monitor's file inventory, it is an old file, and LISP must be connected to that particular file. For LISP II to make known its connection intent to the monitor, a flag is optionally placed on the file description list. The flag is the quoted identifier NEW or OLD. If no such flag (or both) is found, NEW is assumed by default.

2.2.4 Protection

File security is a fierce problem in time-sharing systems and data files must be protected by the monitor from inadvertent and malicious acquisition by unauthorized persons. In LISP II, the presence of the dotted pair

(PROTECT . X)

in the file description list, is used to convey necessary "keys" to lock

(READ WRITE)

or unlock various protected files. The nature of the variable χ is dependent upon the protection schemes provided by the monitor.

The Q-32 time-sharing monitor does not, currently, possess any protection mechanisms, and so the dotted pair designating file protection will be ignored in Q-32 LISP II. For other LISP II implementations, the variable χ can designate a password, a protection code, an executable protection function, a change of protection code, or combinations of these, as permitted by the monitor.

2.2.5 Identification

This parameter is optional, and used where it is desired to identify a specific physical unit. The dotted pair is of the form

(y . z)

where y and z may take on the following values for Q-32 LISP II.

Unit	y	z	Comment
TAPE	REEL	$n \leq 9999$	Physical reel number.
CRT	SCOPE	$1 \leq n \leq 6$	Physical scope number.
DISC	NAME	identifier	Disc file name where different from the first argument of OPEN, will be used as the name of the file.

For other LISP II implementations, y and z may take on other values and meanings as necessary.

2.2.6 Format

The last dimension of a file to be considered is its format. By format, we mean the external organization of the file, particularly its blocked structure and its printed structure. Within the physical limitations of the hardware, the user may, optionally, control these formats; otherwise by default, the

system will set the necessary parameters.

2.2.6.1 RECORD. The first format parameter, RECORD, specifies the number of lines to be blocked in each record. This parameter can not be changed over the life of a file. The dotted pair is of the form

(RECORD . n)

where n is the integer number of lines.

For Q-32 LISP II, and most time-sharing systems in general, it is desirable to read or write maximum sized records for faster I/O. The following table specifies the upper bound on n for Q-32 LISP II units.

Unit	n Max	n Default	Comment
TTY	1	1	Record ≡ 1 line of 72 characters.
TAPE	30	30	Line ≡ card image of 72 characters.
	20		Line ≡ 120 characters.
	optional		Line ≡ binary word.
DISC	50	50	Line ≡ card image of 72 characters.
	30		Line ≡ 120 characters.
CRT	680 n	680	Line ≡ binary word; 1 ≤ n ≤ 3
CORE	optional	1	Default will consider a CORE file ≡ 1 record ≡ 1 line of optional character size

2.2.6.2 Page Format. For symbolic files only, the format can stipulate the structure of a printed page. The page is logically considered to have two directions--horizontal measured in columns from 1 to 120, and vertical measured in lines from 1 to 50--with three control points in each direction as shown in Figure 1.

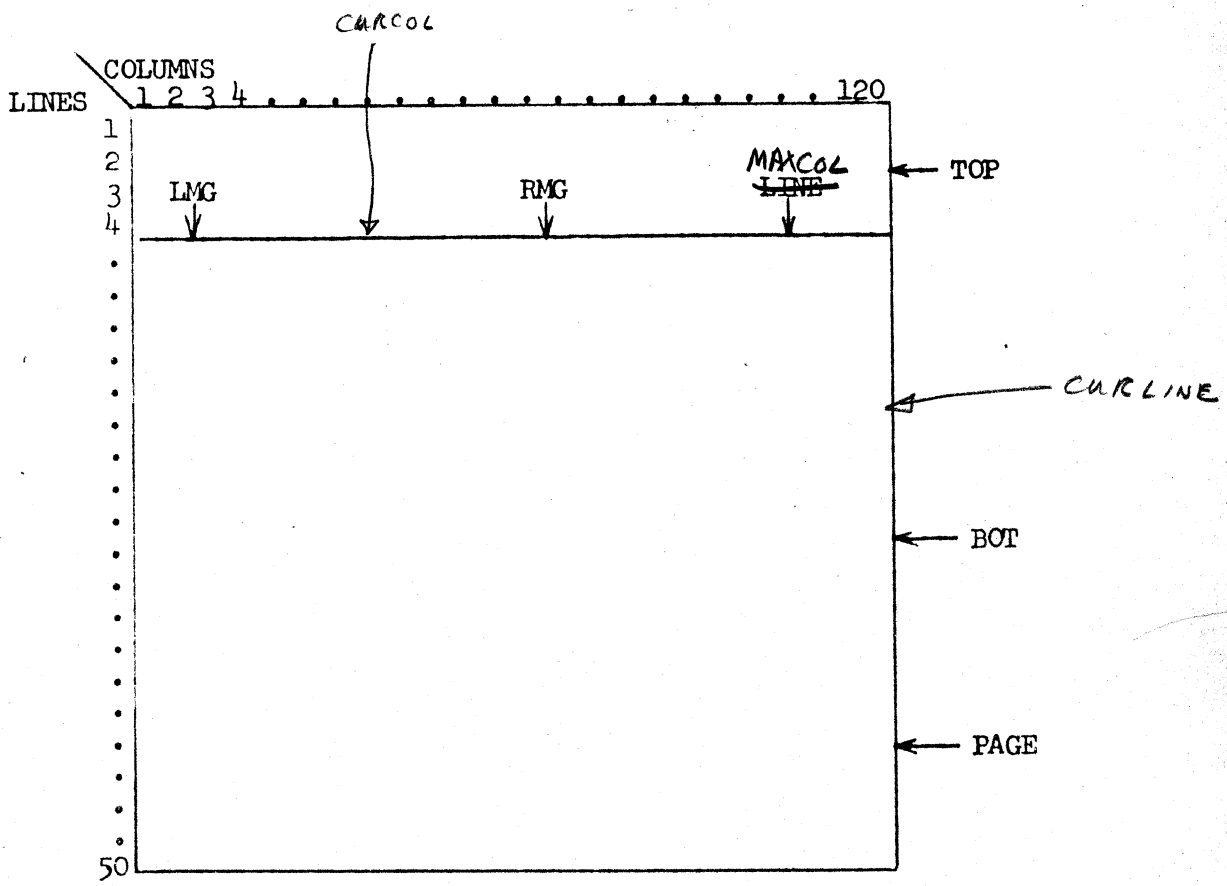


Figure 1. Page Format Controls

In the horizontal direction these control points are (1) the left margin (LMG), (2) the right margin (RMG), (3) and the maximum ~~character~~ column (~~LINE~~ MAXCOL). The vertical direction can be considered symmetric with the horizontal direction and the controls are TOP, BOT, and PAGE, respectively. In all cases, these controls are integer quantities which refer to column or line numbers. Though, apparently more valuable for formatting output, these controls are also effective for reading formatted input information.

2.2.6.3 Horizontal Control. LMG specifies the left most character position for each line, and all tabbing operations, as specified by primitive TAB below, are relative to this margin. It is, therefore, possible to format a line of text on output and position that format, left justified, at any column on the page by appropriate setting of LMG.

RMG acts like the bell on a typewriter and warns the user that he is nearing the maximum column of the line. The user may respond to such a RMG overflow condition by taking remedial actions such as hypenation, restoring the carriage, entering further characters until a blank occurs, etc. The procedure for responding to RMG overflow is discussed in paragraph 2.2.6.5 below.

MAXCOL
~~LINE~~ specifies the highest column in a line, and thus gives the maximum number of characters per line. Whereas LMG and RMG may vary to achieve various formatting effects, MAXCOL ~~LINE~~ as with RECORD will be constant over the life of a file, as these two parameters are used by LISP for structuring the internal string-representation of a file.

To specify these parameters, the dotted pair has the form

(HORIZONTAL . (χ y z))

where χ, y, and z are the integer values for LMG, RMG, and MAXCOL etc → ~~LINE~~, respectively.

2.2.6.4 Vertical Control. TOP specifies the first line of a page, and may be used to position a group of lines of text any where on the page. The primitive ROLL, as noted below, will advance the line controls relative to TOP.

BOT designates the last line of a page, in a fashion analogous to RMG. Upon BOT overflow the user may desire to extend the number of lines, or he may desire to advance to the next page. He may desire to print a header on the next page or a trailer on the current page prior to completing the printing that induced the overflow. (See paragraph 2.2.6.5.)

PAGE designates the highest line of a page and thus the maximum number of lines per page. PAGE overflow can occur and is treated like BOT overflow.

To specify these parameters, the dotted pair has the form

(VERTICAL . (χ y z))

where χ, y, and z are the integer values for TOP, BOT, and PAGE, respectively.

2.2.6.5 Overflow. There are five overflow conditions: RMG overflow, LINE overflow, BOT overflow, PAGE overflow, and RECORD overflow. Except for LINE and RECORD overflow, which are LISP's responsibility, overflow responses are, optionally, under the user's control by specification of procedures to be evaluated at overflow time. The form for this specification is

(OVERFLOW . (χ y z))

where χ, y, and z are FORMAL parameters corresponding to the procedures for RMG overflow, BOT overflow, and PAGE overflow, respectively.

A FORMAL parameter for overflow ~~must point to a procedure that is a predicate~~ must be a function of no arguments and no value with side effects. The side effects are the effects the user is after, such as changing the margins, or printing a header or trailer. ~~The value TRUE for the predicate is interpreted by the I/O logic as "truncate" the token being printed; i.e., do not enter any more characters from the print name of the token that induced the overflow and continue with the calling function. The value FALSE for the predicate is interpreted as a NOP, and printing of the token that induced the overflow will continue using the current state~~ of all control parameters. The state of these parameters will be unchanged unless they have been modified by such primitives as TAB, ROLL, ENDOUT, ENDIN, ENDOUTR, ENDINR, PRINCH, READCH, etc., evaluated as the side effect of the overflow procedure.

The primitives HORIZONTAL and VERTICAL, as described later, permit dynamic setting of a file's format controls, including the overflow procedures.

2.2.6.6 Format Defaults. If some or all format parameters are absent, default values will be set automatically by OPEN as noted below.

Item	Default Value
RECORD	See paragraph 2.2.6.1
LMG	1
RMG	~~LINE~~ 73
LINE	72
TOP	1
BOT	~~PAGE~~ 51
PAGE	50
RMG overflow	~~RMGO~~ NOP
BOT overflow	~~BOTO~~ NOP
PAGE overflow	~~BOTO~~ NOP

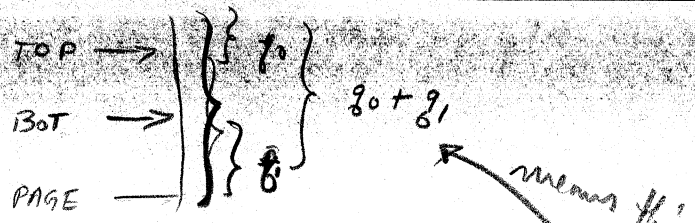
The overflow functions RMGO and BOTO access and adjust four system parameters that control the internal column, line, and record logic. These parameters are:

CURCOL	current column number
CURLINE	current line number of this page
SUMLINE	current line number of this record
LINELOC	location current line of this record

The actions of RMGO and BOTO can be described in English as follows:

RMGO:
1. Set CURCOL equal to the left margin (LMG).
2. Increment CURLINE by one.
3. Increment SUMLINE by one.

IRMGO ≡ ENDIN + CURLINE ← CURLINE+1, Return FALSE



4. Increment LINELOC by number of words per line (WPL).*

~~5. Return the Boolean FALSE.~~

BOTO: 1. Set CURLINE equal to the top of the page (TOP).

2. Increment LINELOC to the top of the next page, i.e., LINELOC + (WPL)(PAGE-BOT+TOP)

~~3. Return the Boolean FALSE.~~

The system actions for LINE and RECORD overflow are similar to RMGO and BOTO except that necessary mechanisms for loading or dumping of records from or to the external medium must be activated conditionally by use of the functions ENDIN, ENDOUT, ENDINR, and ENDOUTR.

2.3 RESERVED I/O IDENTIFIERS

This paragraph gives the complete structure of the file description list (second argument of OPEN) for each reserved I/O identifier.

2.3.1 TTY.

```
((UNIT . TTY) (FORM . ASCII) (RECORD . 1)
 (HORIZONTAL . (1 72 72)))
```

2.3.2 DISC.

```
((UNIT . DISC) (FORM . ASCII) (RECORD . 50)
 (HORIZONTAL . (1 72 72)) (VERTICAL . (1 50 50))
 (OVERFLOW . (RMGO  BOTO  BOTO )))
```

---

*WPL is a file parameter set to 1 for binary files and computed once by OPEN for symbolic files as $\left(\frac{\text{LINE}-1}{\text{CPW}}\right) + 1$, where CPW (characters per word) is a system constant equal to 6 for Q-32 LISP II.

2.3.3 TAPE.

```
((UNIT . TAPE) (FORM . ASCII) (RECORD . 30)
 (HORIZONTAL . (1 72 72)) (VERTICAL . (1 50 50))
 (OVERFLOW . (RMGO  BOTO  BOTO )))
```

2.3.4 CORE.

```
((UNIT . CORE) (FORM . ASCII) (RECORD . 1))
```

NOTE: CORE may be treated like other I/O units; however, considering that all internal files are a string of record size, CORE as an I/O unit, if restricted to a file of one record in size, with the record being only one line of characters, can be used to create internal strings for special formatting purposes. Thus CORE. contains no HORIZONTAL, VERTICAL, or OVERFLOW parameters. The last two items would be meaningless for a one-line file; however, HORIZONTAL is necessary to specify the parameter LINE. We shall discuss how this and other parameters can be appended to the reserved I/O identifier in paragraph 2.3.6.

2.3.5 CRT.

```
((UNIT . CRT) (FORM . BINARY) (RECORD . 680))
```

2.3.6 Extensions

The reserved I/O identifiers do not necessarily satisfy all the required parameters of a file description. For example, if the UNIT of a file to be OPENed is TAPE, a REEL parameter is required. Similarly, if the file is CORE, LINE must be given as part of the parameter HORIZONTAL. These extensions can be provided by CONSing them to the value of the reserved I/O identifier when OPEN is called. For the above examples

```
(OPEN 'TAPE01) (CONS '(REEL . 1234) TAPE.))
```

will OPEN a file named TAPE01 from physical reel numbered 1234. Similarly,

```
(OPEN 'PRETTYP) (CONS '(HORIZONTAL . (1  360 360)) CORE.))
```

will open a core file named PRETTYP that is a line (a character string) of 360 characters.

This technique of CONSing parameters to the reserved I/O identifier is also quite useful for OPENing non-standard dimensioned files. For example

```
(OPEN 'DISCFILE (CONS 'OLD   DISC.))
```

will OPEN a disc file named DISCFILE by connecting to an existing permanent disc file in the Time-Sharing System inventory, as the flag OLD designates the OLD connection mode. The absence of a connection mode flag in DISC. and other reserved I/O identifiers allows them to be used for OPENing new files, as NEW is implied by default. Other examples such as those below are also possible.

```
(OPEN 'TAPEB (CONS '(FORM . BINARY) TAPE.))
(OPEN 'TTYFILE (CONS '(HORIZONTAL . (10 60 72)) TTY.))
```

3. FILE PURGING: SHUT

A file created by OPEN may be purged from LISP II by evaluating the function SHUT. SHUT breaks all the communication linkages and deletes all internal structures--arrays, strings, and variables--dynamically established by OPEN. Before purging the file, SHUT communicates to the time-sharing monitor the disposition of the file; i.e., add file to permanent inventory, delete all file references, change protection mode of file, change file name, etc. The value of SHUT is the value of the FLUID variable FILES., which is a list of the names of all currently OPENed files.

3.1 FUNCTION CALL

SHUT is a function of two arguments, and has the form

(SHUT file-name, file-disposition)

The file name must be a quoted LISP II identifier naming an actively OPENed file; i.e., a file name previously used with OPEN. If

(MEMBER file-name, FILES.) = FALSE

SHUT acts as a NOP.

3.2 FILE DISPOSITION

The second argument of SHUT is a list of dotted pairs of attributes and values specifying the disposition of the file.

3.2.1 FILE

The dotted pair

(FILE . X)

is used to communicate to the time-sharing monitor the inventory action desired for the file. The action desired is specified by variable X. For Q-32 LISP II, X may take on the values SAVE or DELETE . SAVEd files are inventoried and may be accessed at a later date by the use of the flag OLD as the connection mode of an OPEN call. DELETEd files will not be inventoried and will disappear. If the CDR of the FILE pair is DELETE, no further disposition parameters are meaningful.

3.2.2 NAME

The dotted pair

(NAME . file name)

may be used to change the name of the SAVEd file in the Time-Sharing System inventory. If this parameter is absent, the file name under which the file was OPENed, will be used. In either case, SHUT may be forced to query the on-line user for a new file name if the file name (a name local to LISP) conflicts with a file already existant in the inventory.

3.2.3 PROTECT

The dotted pair

(PROTECT . y)

may be used to change the file protection of the SAVEd file. If this parameter is absent, the protection mode under which the file was OPENed will be used.

For Q-32 LISP II, this parameter will be ignored until such time as file protection mechanisms are installed in the time-sharing monitor.

3.2.4 LOG

The dotted pair

(LOG . z)

may be used to log a one line message on the operator's console. The variable z is that message, in the form of an S-expression. LOG is useful for giving directions, labels to tapes, etc. to console operators when necessary.

4. FILE SELECTION: INPUT, OUTPUT

Once a file has been OPENed it may be selected as the current input or output file by evaluating one of the functions INPUT or OUTPUT. Once a file is selected, all I/O primitives act only on that file. Thus it is possible to compose a LISP II program that is unit, form, format, etc. independent by supplying the name of a file as the argument for the program. Such a capability is quite powerful for debugging purposes where the checkout is simplified with an on-line Teletype file for a program that ultimately requires a tape or disc file.

4.1 FUNCTION CALLS

The file selection calls are of the form

(INPUT file name)

(OUTPUT file name)

where file name is the name of the file used in a previous OPEN call. INPUT is the function used for selecting input files for reading. OUTPUT is the function used for selecting output files for printing. The value of INPUT or OUTPUT is the file name previously selected. By using INPUT or OUTPUT as the right part of an assignment statement, one can save the name of the prior selected file for subsequent reselection; e.g.,

```
———
———
———
X ←— (OUTPUT 'TTYFILE)
———
———
———
(OUTPUT X)
———
———
```

4.2 SELECTION MECHANICS

When a file is OPENed, three internal arrays are created and initialized by OPEN and placed on the property list of the file name. They are:

1. The data record string
2. An INTEGER array of format control variables
3. A FORMAL array of overflow, and post-string processing functions.

When a file is selected, INPUT or OUTPUT retrieves the locations of these arrays from the property list of the file, and binds various locations within these arrays to a family of FLUID, LOCATIVE variables. Subsequent use of read and print functions chain through these FLUID, LOCATIVE variables to reference and change the stored data values. In this way no extraneous data copying is necessary, and yet all control variables are effected at a common location relative to each file, i.e., the internal arrays.

Since all modified control variables are uniformly effected at one point, selection and de-selection of files has no effect itself on the state of the file; i.e., the state of its control parameters. This is required if de-selection of files is to be permitted freely in the middle of lines and records. Recall, that a partial line or record exists only by the state of the file's control parameters.

Finally, there exist a number of I/O functions that allow the user to adjust a file's control parameters explicitly; e.g., TAB, ROLL, HORIZONTAL, VERTICAL, etc. These functions affect the file's internal arrays directly and may be used without concern as to whether the file is or is not selected.

5. TERMINATOR FUNCTIONS: ENDIN, ENDOUT, ENDINR, ENDOUTR

There are four I/O functions--ENDIN, ENDOUT, ENDINR, ENDOUTR--whose sole purpose is to keep I/O data flowing through the system. They relate to line and record control for input and output of the selected file.

	input	output
line	ENDIN	ENDOUT
record	ENDINR	ENDOUTR

In effect, they are similar to the actions of TERPRI and TEREAD of LISP 1.5. They are functions of no arguments and no values, and are evaluated for their side effects only. When evaluated they chain through the FLUID, LOCATIVE variables set up by INPUT or OUTPUT to housekeep the three primary file control parameters, CURCOL, SUMLINE, and LINELOC.

5.1 LINE TERMINATORS

The functions ENDIN and ENDOUT are used to terminate reading and printing, respectively, the current line of a file, and advance the control logic to the next line, if it exists. If it does exist, i.e., SUMLINE > RECORD, we have record overflow and the appropriate record terminator--ENDINR or ENDOUTR for input and output, respectively--is invoked.

Realizing that ENDIN chains through FLUID, LOCATIVE variables set up by INPUT, and ENDOUT chains through FLUID, LOCATIVE variables set up by OUTPUT, both perform essentially the following actions:

1. Increment SUMLINE by one
2. If SUMLINE > RECORD, call END{OUT/IN}R, otherwise
3. Set CURCOL=LMG, for symbolic files only
4. Set LINELOC=LINELOC+WPL

5.2 RECORD TERMINATORS

The functions ENDINR and ENDOUTR are seldom called explicitly by the user, but rather, are called by other I/O functions, such as by the line terminators above. When they are evaluated they terminate the file record currently in core. For ENDINR, that means loading another record from the external medium; for ENDOUTR, it means dumping the current core record onto the external medium. To achieve these operations it is necessary for the record terminators to invoke post-string processes that:

(1) read or write a raw external core image from or to external medium,
(2) delete or insert requisit format information, and
(3) translate raw character codes from 12-to-8 bits, 8-to-12 bits, 6-to-8 bits, 8-to-6 bits, etc. from tape, Teletype, and disc standards

The required post-string processing is determined and parameterized by OPEN from the file description, mechanized by INPUT or OUTPUT from the OPEN parameters, and invoked by these record terminators.

Though performed in different order, ENDINR and ENDOUTR perform essentially the same following actions:

1. Clear record-string (to blanks for symbolic files, to zero for binary files)
2. Load or dump external core image
3. Perform post-string processing
4. Set CURCOL=LMG, for symbolic files only
5. Set SUMLINE=1
6. Set LINELOC=record-string location+(TOP*WPL)

Note that the record terminators do not invoke the line terminators, and so any partially filled line will be lost after a record terminator is evaluated.

Also note that the line terminators invoke the record terminators only when record overflow is encountered. When reading blocked records, it is sometimes desirable to explicitly evaluate ENDINR before a record is fully read, thus initializing the file controls to the first line of the next record. When printing blocked records, a final ENDOUTR should be evaluated before quitting for the day, or before SHUT is evaluated, else any data in a partially filled record will be lost.

try to make ENDOUTR / ENDINR } pick up last partial line

6. BASIC PRIMITIVES

All the I/O primitives noted in this section affect only the file selected by INPUT or OUTPUT. Because of this, these primitives themselves are file independent and may be used as conveniently on disc files as on tape or Teletype files.

As a mnemonic aid, all print primitives that drop the "t" in their names, e.g., PRIN, PRINCH, etc., do not evaluate ENDOUT and do not terminate the current line. Those that use the full spelling of "print" in their names do terminate the current line by evaluating a final ENDOUT. In general, all read primitives do not evaluate ENDIN, and do not terminate the current line.

6.1 READCH

READCH is a function of no arguments. It reads the current character of the line and returns that character as a one-character identifier; it also increments CURCOL by one. All format and overflow conditions, as specified for the file selected, will be invoked. If line or record overflow is encountered, the next line or record will be positioned by ENDIN or ENDINR, respectively, but the value of READCH will be FALSE. Thus, READCH can be used as a semi-predicate to test for line or record boundaries. Since both these conditions return FALSE, primitive IOSTATUS, discussed below, can be used to remove the ambiguity, where necessary.

READCH will also return FALSE if it is impossible to read the next character because a physical (tape) or logical (disc) end-of-file was encountered. Again, IOSTATUS can be used to clarify the nature of the read failure.

6.2 PRINCH

PRINCH is a function of one argument, a one-character identifier that is also returned as its value. The print name (a single character) is entered in the line at the current column, and CURCOL is incremented by one. All format and overflow controls are in effect. If line or record overflow is

encountered, ENDOUT or ENDOUTR will be evaluated. For symmetry with READCH, PRINCH will accept FALSE as its argument, that is treated by PRINCH as a line termination, i.e.,

(PRINCH FALSE) ≡ (ENDOUT) .

Note that PRINCH, normally, does not call ENDOUT, except for line overflow and a FALSE argument.

6.3 PRINTOKEN

PRINTOKEN is a function of one argument, a LISP II token (see TM-2260/004/00, LISP II PROJECT MEMO NO. 11, "The Syntax of Tokens") that is also returned as its value. The print name of the token is entered in the line, starting at the current column. All format and overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. PRINTOKEN does not normally call ENDOUT, and after evaluation, CURCOL marks the column of the line following the last character of the token's print name.

Many tokens, such as numbers, Booleans, and special tokens, have no print names. For these cases special primitives, such as TOSTRG and PRINSTRING, will be called to print these tokens. (TOSTRG is a conversion function that converts any token to a string.)

Note that PRINTOKEN does not apply special primitives for tokens with unusual spellings. For such cases, it is the user's responsibility to evaluate TOSTRG and SYMPRIN himself.

6.4 PRINSTRING

PRINSTRING is a function of one argument, a LISP II string, that is also returned as its value. The string is taken literally as its print name and entered in the line starting at the current column,as if each character in the string were printed with PRINCH. All format and overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. PRINSTRING does not normally call ENDOUT, and after evaluation CURCOL marks

the column of the line following the last character of the string.

For users wishing to print a string with unusual spellings so that it has READ symmetry, i.e., can be read back in by LISP as a string, they should use SYMPRIN, which provides this special formatting. For example,

(PRINSTRING '#ABC'D#) prints as ABC'D
whereas
(SYMPRIN '#ABC'D#) prints as #ABC''D#

6.5 PRIN AND PRINT

PRINT is PRIN plus a final ENDOUT. Therefore, we shall need only specify PRIN. PRIN is analogous to Q-32 LISP 1.5 PRINO, except that it handles all legal LISP II data types.

PRIN is a function of one argument, an S-expression, that is also the value of PRIN. Starting at the current column, PRIN enters left and right parentheses, dots (set off with blanks), and print names for all tokens in the S-expression, in list-notation format. Token print names are entered in the line by the primitive PRINTOKEN, thus there is no special action taken for unusually spelled tokens.

All format and overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. PRIN does not terminate with a final ENDOUT, as does PRINT; therefore, after evaluation, CURCOL marks the column of the line following the last character of the S-expression.

6.6 READ

READ is a function of no arguments. Its value is the next S-expression in the file beginning at the current column. If line overflow occurs, READ automatically calls ENDIN. READ does not normally terminate with ENDIN, and therefore CURCOL marks the next column following the last character of the S-expression read. All other format and overflow controls are in effect.

READ operates by CONSing tokens into list structure as directed by the structure of the S-expression seen. READ calls upon a Finite State Machine* to supply these tokens. The Finite State Machine uses READCH to read characters in the file which it converts into LISP II tokens. Thus, READ does not directly concern itself with the processes of searching and maintaining the OBLIST, composing numbers, making strings, and the like. These are more efficiently performed by the Finite State Machine.

6.7 SYMPRIN AND SYMPRINT

SYMPRINT is SYMPRIN plus a final ENDOUT. Therefore, we need only specify SYMPRIN.

SYMPRIN is a symmetric PRINSTRING whose argument and value are a LISP II string. It is symmetric because it prints the string such that the printout, when read back in by LISP, will yield, internally, the identical string. For most strings, SYMPRIN could be defined by the effect of the following:

```
(PRINCH '#)
(PRINSTRING χ)
(PRINCH '#)
```

However, for strings with unusual spellings, i.e., strings containing the characters quote mark ('), fence (#), percent sign (%), or carriage return ((cr)), SYMPRIN must quote these characters to remove syntactic ambiguity. For example,

(PRINSTRING '#A%B#) prints as A%B

and

(SYMPRIN '#A%B#) prints as #A'%B#

---

*Finite State Machine specification will be published as a separate document and is not covered herein.

Now if #A'%B# is read by LISP, it yields the internal string '#A%B#, whereas reading A%B would probably cause an error, since B is not a legal character following the percent sign,which is used as an escape character.

As far as format control and overflow, SYMPRIN works exactly as does PRINSTRING.

6.8 READWORD AND PRINTWORD

These two primitives are to be used with binary files exclusively. ENDIN and ENDOUT for READWORD and PRINTWORD, respectively, are always invoked since a word and a line are equivalent for binary files.

READWORD is a function of no arguments, whose value for Q-32 LISP II is a 16-digit octal number contained at the current word of the octal arrays for the record. READWORD advances control to the next word, after evaluation.

PRINTWORD is a function of one argument, an octal number, that is returned as the value of the function. PRINTWORD enters the octal number into the current word of the octal array for the record and advances control to the next word.

7. FILE CONTROL PRIMITIVES

These primitives permit dynamic file positioning, format initialization and interrogation. By appropriate utilization of these primitives, the user can retain absolute control of his files. He can compose private overflow functions, adjust printed output for various degrees of formality. Though not provided herein, these primitives will be used to create private and public "pretty print" procedures for S-expression, Intermediate Language, and Source Language output.

7.1 FILE POSITIONING: TAB, ROLL, AND POSITION

The structure of a file is the hierarchy of FILE, RECORD, PAGE, LINE, and column. The user has freedom of movement within these structures by use of the three primitives TAB, ROLL, and POSITION. He may use these primitives regardless of whether the specified file is selected as the current file or not.

7.1.1 TAB

TAB is a function of two arguments that positions the current column of the current line. It has the form

(TAB file-name, column)

The first argument, file name, is the name of a previously OPENed file. That file need not be currently selected.

The second argument, column, is a positive or negative decimal integer, relative to the left margin (LMG),that is set as the new value of CURCOL. The negative value is permitted since the TAB operation is always taken relative to the left margin (LMG). The restrictions on column can be expressed by

CURCOL ←(LMG + column), if

$1 \leq (LMG + column) \leq LINE$

Like a typewriter carriage, if (LMG + column) exceeds either upper or lower bound, CURCOL will be set to LINE or 1, respectively.

By permitting negative values for column, the user may enter information to the left of the left margin. High values for column, i.e., (LGM + column) > RMG, allow information to be entered beyond the right margin without inducing right margin overflow. In both these cases, then, TAB can be used as a "margin release" for the current line only, when LMG and RMG and column are properly set at values other than the extremes 1, and MAXCOL respectively.

The value of TAB is a decimal integer corresponding to the value of the current column (relative to LMG) prior to the TAB action. In fact, the value is that returned by the interrogation primitive COLUMN, discussed below. By combining column positioning with interrogation, the user is provided additional format control. If TAB fails, e.g. — file not OPENed, TAB returns a neg integer value

Finally, TAB can be used to backup within a line for various purposes that include over printing, setting variable fields within a preset header, etc.

7.1.2 ROLL

ROLL is a function of two arguments that positions the current line of the page in a fashion analogous to TAB. It has the form

(ROLL file name, line)

The first argument, file name, is the name of a previously OPENed file. The second argument, line, is a positive or negative decimal integer, relative to the top of the page (TOP), that is set as the new value of CURLINE. As with TAB, appropriate values of TOP, BOT, and line can be used to override overflow margin controls.

The restrictions on line can be expressed by

CURLINE ← (TOP + line) , if

$1 \leq (TOP + line) \leq PAGE$

CURLINE will be set to the extremes PAGE, or 1 if the value of line forces the quantity (TOP + line) to exceed these limits.

The value of ROLL is a decimal integer corresponding to the value of the current line (relative to TOP) prior to the ROLL action. In fact the value is exactly that returned by LINE, discussed below. A negative value may be returned if file-name is not a currently OPENed file.

7.1.3 POSITION

POSITION is a function of two arguments that positions the current record of the file. It also allows various termination marks, e.g., end-of-file, end-of-tape, to be written in the file. It has the form

(POSITION file-name, action)

where file name is as noted with TAB and ROLL above. The second argument, action, is a positive decimal integer action code for a file operation. The value of POSITION depends on the value of the action desired. The legal action codes, values, and their meanings are given in the following schedule.

For many files, particularly TTY, CRT, and CORE units, POSITION acts as a NOP with a value of NIL. NIL is also returned for illegal action codes. For POSITION, a record corresponds to a record on tape, and a sector on disc. Furthermore, many physical files, separated by an end-of-file mark, may exist on tape; however, only one file is permitted on disc. Therefore, it is impossible to POSITION beyond a disc end-of-file, but you can POSITION past a tape end-of-file,at your own risk.

Action Code	Action	Value
1	Skip to line 1 of next record	• 'EOF, if next record is an end-of-file • n, the number of non-EOF records skipped. • 'EOT, if next record is an end-of-tape • If EOF encountered, tape positioned after EOF mark, disc positioned at EOF mark (can be written over it)
2	Skip to line 1 of next tape file - or to the end-of-file mark of this disc file	
3	Write an end-of-file mark at current line of file.	file name
4	Write an end-of-tape mark at current line of tape file; action 3 for disc files	
5.	Backup to line 1 of file (rewind)	
6	Backup to line 1 of prior record	
7	Back to last line (just before end-of-file) of prior tape file; action 5 for disc files	
$7 < a < 1$	Currently illegal for Q-32 LISP II	NIL

7.2 FORMAT INITIALIZATION: HORIZONTAL, VERTICAL AND CLEAR

As described in section 2.2.6, it is possible for the user to initialize format control parameters at the time the file is OPENed. With the primitives described here, the user may dynamically initialize many of these parameters after the file is OPENed. Furthermore, it is not necessary for the file specified to be the current file selected.

7.2.1 HORIZONTAL

HORIZONTAL is a function of two arguments, that change the horizontal control parameters of the file. The form of the call is

(HORIZONTAL file name, parameter list)

The file name is as previously defined. The format parameters are contained in the parameter list that has the form

(LMG, RMG, RMG overflow function)

Each of these parameters has been previously described. To repeat, LMG is a positive decimal integer specifying the left margin; RMG is a positive decimal integer specifying the right margin; the overflow function is a FORMAL parameter to be invoked by LISP at such times as right margin overflow occurs.

The legal values for LMG and RMG is expressed by

$$1 \leq LMG < RMG$$

If these inequalities are violated, HORIZONTAL acts as a NOP, and returns a value of NIL, otherwise the file name is returned as the function value.

7.2.2 VERTICAL

VERTICAL is a function of two arguments, that change the vertical control parameters of the file. The form of the call is

(VERTICAL file name, parameter list)

where the arguments are similar to those of HORIZONTAL. However, the elements of the parameter list are somewhat different.

(TOP, BOT, PAGE, BOT overflow function, PAGE overflow function) is the form of the parameter list, with the parameters having the same definitions as given previously. Reviewing these, TOP and BOT are positive decimal integers specifying the upper and lower page boundaries, respectively. PAGE is also a positive decimal integer specifying the maximum lines per page. The overflow functions are FORMAL parameters that are invoked at a bottom margin, and maximum line per page overflow.

The legal values for TOP, BOT, and PAGE is expressed by

$1 \leq TOP < BOT \leq PAGE \leq 50$

remove restriction here

remove

If these inequalities are violated, VERTICAL acts as a NOP. ~~and~~ returns ~~a value of NIL, otherwise the file name is returned as the function value.~~

Return old values of parameters

If not opened, error

Note: BOT may be > PAGE, and thus BOT overflow can never occur as PAGE overflow would be encountered first. If BOT=PAGE, both overflow conditions can occur, with BOT occurring before PAGE.

7.2.3 CLEAR

CLEAR is a function of one argument, file name, as above, and has no value. CLEAR clears the internal record of the file named to all blanks for symbolic files, and to all zero words for binary files. CLEAR is provided as a convenience to the user.

7.3 FORMAT INTERROGATION: COLUMN, LINE, IOSTATUS

When performing various format sensitive read or print operations, it is particularly helpful to know where you are now. These primitives allow the user to interrogate his file for this information.

7.3.1 COLUMN

COLUMN is a function of one argument, file name, as above, that returns the value of the current column of the line relative to the left margin. The value returned can be expressed as

(CURCOL - LMG)

~~return -1 if~~ error NO FN

7.3.2 LINE

LINE is a function of one argument, file name as above, that returns the value of the current line of the page, relative to the top of the page. The value returned can be expressed as

(CURLINE - TOP) ~~returns~~ if no FN error.

7.3.3 IOSTATUS

IOSTATUS is a function of no arguments. It returns as its value, a decimal integer code value designating the status of the last I/O action for either the input or output selected file. IOSTATUS must be used immediately following any read or print primitive to test the status of the transfer. This is particularly valuable for reading, as it can resolve the cause of FALSE during READCH, or whether EOF or EOT returned by READ were literal data in the file, or READ's output response to an end-of-file, or end-of-file condition.

The current Q-32 LISP II code values are:

Value	Status Condition
1	end-of-line
2	end-of-file
3	end-of-tape
n > 3	error conditions