TM-2260/003/00
Produced under ARPA Contract SD-97

LISP II
Memo #5
DRAFT - February 19, 1965

INPUT/OUTPUT FOR LISP II

by

Clark Weissman

Introduction

Presented herein are a number of ideas for LISP II I/O, primarily for BCD data. The salient characteristics include: dynamic (run time) device assignment and creation (and deletion) of I/O buffers, a current-unit concept to simplify input/output procedures for the variety of different I/O devices, and finite state machines for transforming the input character "stream" into appropriate internal language "tokens" (and vice versa). Implementation of these ideas will be achieved by defining procedures in terms of a set of machine language I/O primitives; a necessary, but not complete list of which is suggested herein.

Certain desirable capabilities are omitted in this paper as they probably should not be part of the basic LISP II system. First, is the ability to specify simultaneous I/O operations. This ability is dependent upon the time-sharing environment in which LISP II must operate and hence special procedures will probably be necessary. The other apparent omission is I/O formatting. It is expected that format capability can easily be accommodated within this I/O proposal by extending the set of primitives and by use of character, string and Comit-rule features of the source language.

1.0 Unit Classes

Rather than assign arbitrary numeric codes for unit classes, LISP II source language will include an open set of UNIT IDENTIFIERS. These identifiers are not reserved, (i.e., in the sense that "FOR," "IF," "PROCEDURE" are reserved identifiers) and may be used by the programmer with complete freedom as variables, procedure names, etc. UNIT IDENTIFIERS must be used for those primitives which require specification of the unit class to allow the LISP II system to recognize the device and perform the appropriate internal actions.

1.1 Unit Identifiers

TAPE (magnetic or paper), DISC, SCOPE (CRT and light pen), TTY (teletype) or CONSOLE, PHONE (high-speed data link) or COMPUTER, DRUM, CARD (reader and punch), PRINTER,* RANDT.*

*These identifiers correspond to uni-directional devices which will cause an error condition if used in the incorrect direction.

1.2 Data Structure

Each I/O device will contain data stored in FILES. FILES will be composed of RECORDS, which in turn will consist of LINES. For BCD FILES, LINES will be composed of CHARACTERS. For binary devices LINES and computer words will be synonymous.

The size, format, and code for LINE, RECORD, and FILE will be defined (hopefully parametrically) for each UNIT IDENTIFIER, subject to the peculiarities of the local machine system. End-of-File, End-of-Record, and End-of-Line may be denoted with explicit control symbols, or by implicit counters.

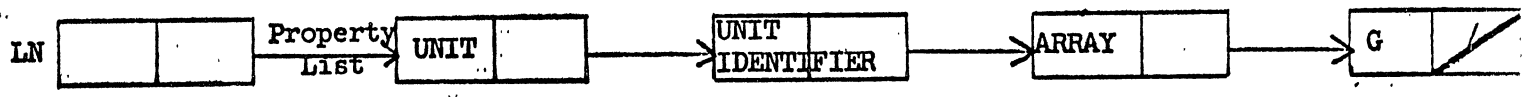
1.3 Character Codes

LISP II will use the 7-bit ASCII character set with certain prohibitions on input (", ?, !). For 6-bit character machines (e.g., 7090, Q-32), some internal LISP II encoding will accept the following 64 characters:

0-9	= < > '	blank
A-Z	+ - * $	shift
. : , ;	/ \ ↑ ←	carriage return
( ) [ ]	# % & @	line feed

1.4 Dynamic I/O Buffers

It will be possible in LISP II to reserve various available I/O devices concurrently at run time in a fashion similar to the CTSS ATTACH and the TSS FILE calls. In addition, it will be necessary for the LISP II user to create a LOGICAL NAME (LN) for these devices. For each device declared, LISP II will create an integer array sufficient to contain one full RECORD of data as defined by the device's UNIT IDENTIFIER. This array will be internally named (e.g., given a GENSYM name) and that name and the UNIT IDENTIFIER will be placed on the property list of the device's LOGICAL NAME.



1.5 Primitives

1. MAKEBUF (U): Creates an integer array sufficient to contain one full RECORD for UNIT IDENTIFIER U. Returns G, the internal array name (a GENSYM) if successful, NIL is unsuccessful (not enough array space).

2. REQUEST (U LN L): LN is the LOGICAL NAME for a device of class UNIT IDENTIFIER U. L is an ordered list of supplementary parameters necessary to reserve the I/O device. (The contents and order of L is particular to the device class and machine system.) Returns LN if successful, NIL if unsuccessful (all units busy or not enough array space).

3. RELEASE (LN): LN is the LOGICAL NAME for a previously reserved device. The device reservation is released, as are the array and array name. The property list of LN is purged of all I/O references. Returns LN.

4. POSITIONR (LN N): The device associated with LOGICAL NAME LN is positioned N RECORDS from its current position. If N is negative, device is backed N RECORDS or to first RECORD of current FILE. If N is positive, device skips N RECORDS or to first RECORD of next FILE. Returns LN.

5. POSITIONF (LN FX): The device associated with LOGICAL NAME LN is positioned to the first RECORD of an externally named FILE FX. FX may also be a signed integer in which case the device backs n FILES for negative n or skips n FILES for positive n from its current position. Returns LN if successful or NIL if unsuccessful, (no FILE found).

6. MOVEIN (LN): Transfers the next RECORD on the device associated with LOGICAL NAME LN into the internal array associated with LN. Returns LN if successful, or NIL if unsuccessful, (no LN declared or end-of-file encountered--in which case device positioned after end-of-file).

7. MOVEOUT (LN): Transfers the internal array associated with LOGICAL NAME LN as the next RECORD on the device associated with LN. Writes an end-of-record mark if necessary. Returns LN if successful, or NIL is unsuccessful, (no LN declared or no storage available on device).

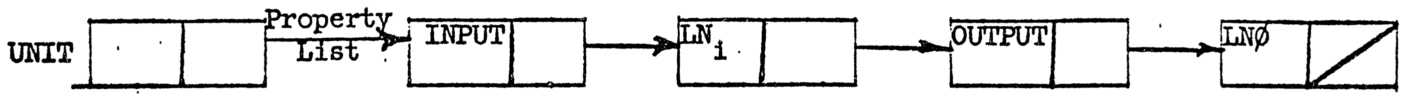
8. WEOF (LN): Writes an end-of-file mark as the next RECORD on device associated with LOGICAL NAME LN. Returns LN.

9. REWIND (LN): If meaningful for the device associated with the LOGICAL NAME LN, that device is positioned to the first RECORD. Returns LN.

2.0 Current-UNIT Approach

For simplicity of input/output specification in LISP II, all I/O procedures will reference the property list of the reserved identifier UNIT to determine the current I/O device. A user may then call I/O procedures without explicitly citing an I/O device each time. It will only be necessary to cite an I/O device when the user wishes to change the current unit. To further simplify matters, the LISP II system will always initialize the current unit to some standard device, probably teletype, at start-up and at any error occurrence.

2.1 Mechanism



There will be two attributes on the property list of UNIT, INPUT and OUTPUT. Each will be associated with the LOGICAL NAME of the current input or output unit defined by a previous REQUEST, from which the necessary information for I/O calls can be retrieved. It will then be possible to read and write from different devices concurrently without changing UNIT's property list. Necessary primitives will be available to change the current unit.

2.2 Primitives

1. UNIT (LNi LNØ):	Set LOGICAL NAME LNi as the value of attribute INPUT and LNØ as the value of attribute OUTPUT on the property list of reserved identifier UNIT. If LNi or LNØ is NIL, the respective attribute value remains unchanged. Returns a list of the prior values of INPUT and OUTPUT respectively.
2. SETPROP (L A V):	The property list of identifier L is searched for the first occurrence of attribute A. If A is found, the next list element is set to the value V. Returns prior value of A if successful, or NIL is unsuccessful, (no property list for L, or A not found).
3. GETPROP (L A):	The property list of identifier L is searched for the first occurrence of attribute A. If found, the next list element is returned as the value of GETPROP. Returns NIL if no property list for L, or A not found.
4. REMPROP (L A):	Analogous to LISP 1.5 REMPROP.

3.0 Finite State Machines

Compilation of source language in LISP II will be essentially a three phase procedure, namely: source language translation to internal language, internal language compiled into LAP, and finally LAP assembled into machine code. For efficiency and flexibility of design a Finite State Machine (FSM) will perform syntax preprocessing in the translation of source language into internal language.

A FSM will be called upon by the syntax translator to read the input character stream and "pass up" the next "token" read on the input device. Tokens are the smallest syntactic elements manipulated by the syntax translator, i.e., they are the symbols of the LISP II syntax. They include: real, integer, octal, and Boolean numbers, strings, identifiers, and reserved identifiers.

FSM's will also be used for output and for other than source language input, including binary. The requisite transforms will be specified for each FSM as they are needed. New FSM's may be added to the system whenever necessary.

3.1 Readers

In reality, a FSM will have two parts, a syntax translator for concatenating characters into tokens (fsm), and a reader (rdr) for reading characters from the input array and passing them to the fsm. This distinction is not capacious for it allows extensive generality. Any fsm can be applied to any input array by use of different rdr's. There will be rdr's for different purposes dictated by the input data structure. For example, there will be binary as well as character rdr's. Also, there will be character rdr's which are "blind" to certain character fields, thereby allowing the reading of data containing external accounting information (such as sequence numbers).

3.2 Mechanism

Figure 1 shows schematically the flow of data through the I/O system. A FSM is created by attaching a fsm to a rdr, and a rdr to an I/O buffer previously created by a REQUEST. The current rdr name is stored as an OWN variable within the fsm. For BCD data, and after appropriate initialization the fsm asks for a character. The rdr accesses the current LINE of the attached I/O buffer via reference to the LOGICAL NAME for the current unit--(All I/O buffer references must be made through the LOGICAL NAME because of array relocatability.) (The relative address of the current LINE and the current unit (LOGICAL NAME) are stored as OWN variables within the rdr)--and "explodes" the current LINE into its constituent characters, simultaneously converting the characters into internal LISP II code by table lookup and storing the converted characters, one per word into an OWN array within the rdr; the OWN array being large enough to store a LINE of characters. One more OWN variable is required by the rdr, namely the current character pointer. This pointer is an index into the exploded line. The rdr passes this pointer up to the fsm which uses it as an index into parameter tables of character classes needed by the fsm for processing the syntax of tokens. (fsm with different token syntax equations can be easily accommodated in LISP II by using different parameter tables.) After processing that character the fsm requests another character from its rdr. If the rdr has exhausted the characters in its exploded buffer, it will explode another LINE. If the I/O buffer is also exhausted, the rdr will internally call MOVEIN to fill the I/O buffer from the next RECORD on the current unit. (The current unit is gotten by chaining down the property list of UNIT.) If there are no remaining RECORDS on the current unit, and error condition exists. fsm calls rdr repeatedly until a token is formed. At that time the FSM exits, its value being a pointer to the formed token, and a token class code, (e.g., string, identifier, octal number, etc.) These items are used by the higher level syntax translator for its devious purposes.

3.3 Further Comment

To allow concurrent use of various fsm's and rdr's on different I/O buffers without changing FSM's to push down-store-automatons, copying rdr's will be required.

Concurrent use of various fsm's and rdr's also creates a problem regarding reinitialization of FSM's (i.e., clearing rdr OWN array and variables) by TEREAD for example since there is no connection between I/O buffers and FSM's. This problem can be resolved by setting the rdr's name on the property list of the current-unit when FSM is first called. (LOGICAL NAME is gotten from the property list of UNIT.) RELEASE must also reinitialize the FSM attached to the LOGICAL NAME.

It should be noted that reinitialization will require Tailing to allow access to all OWN parameters.

Finally, not much has been said concerning FSM's for output or binary since these FSM's are relatively simple compared to the ones discussed.

3.4 Primitives

1.	RDRC	: RDRC reads the next character an device LN and is a psuedo function since it transmitts its value internally. LN is the LOGICAL NAME of an I/O unit previously defined by REQUEST from which the name of the I/O buffer is retrieved for reading characters. RDRC uses three OWN variables, a CHARACTER pointer a LINE pointer, and unit pointer (LN), and one OWN array.
2.	RDRCS	: Like RDRC, but it is "blind" to the last m characters of every LINE. Since sequence numbers are usually placed in columns 73-80 of each LINE, RDRCS can be used to skip these fields.
3.	RDRB	: RDRB reads the next binary word on device LN and is also a psuedo function. It has one OWN variable, word pointer and no OWN array. Otherwise it is like RDRC.
4.	FSMC	: FSMC is a finite state machine for creating character identifier tokens. It takes no arguments, since its single OWN variable, the name of a rdr, is set by SETFSM. Returns a character on unit LN (see RDRC).

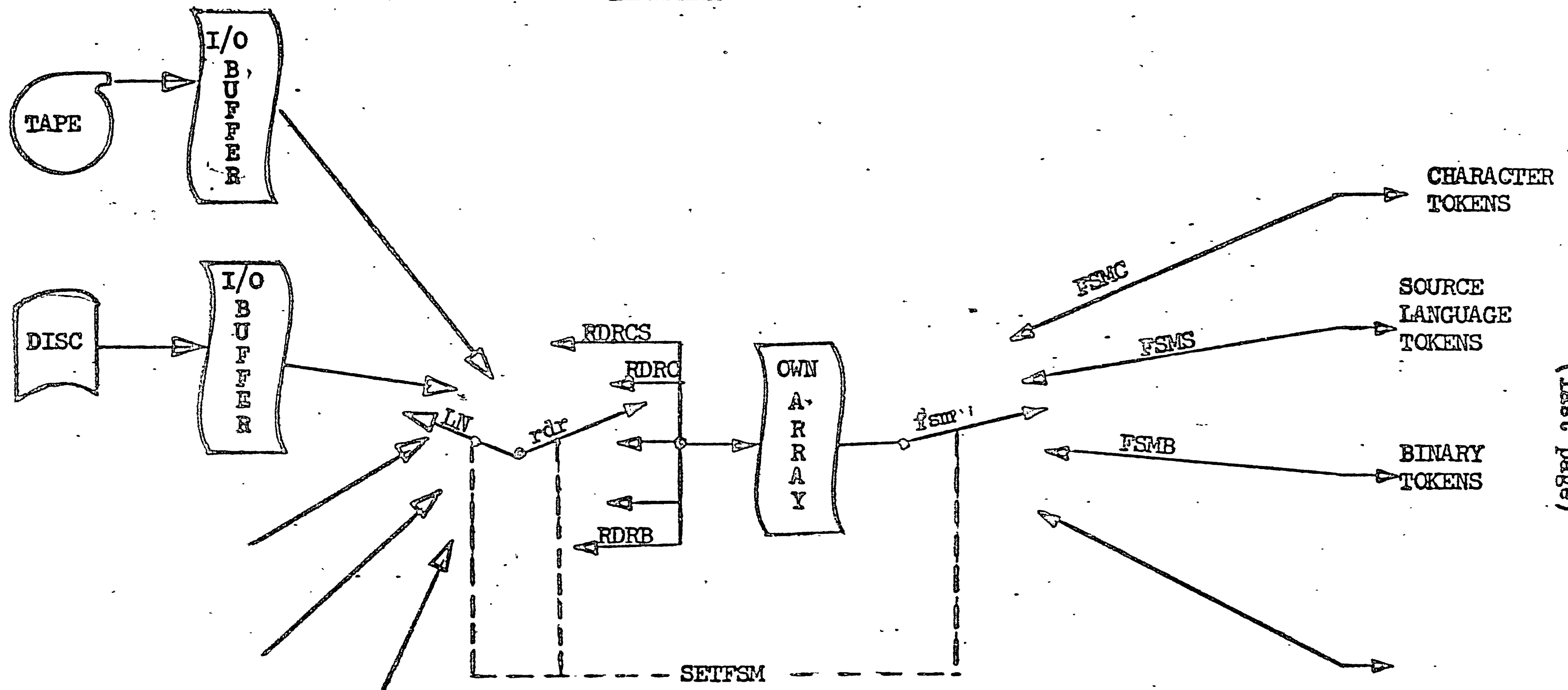
5.	FSMS	: A finite state machine for creating source language tokens,--real, integer, octal, and Boolean numbers, strings, identifiers, and reserved identifiers. Tokens created can also be used by S-expression syntax translator. Otherwise like FSMC. Returns the next token on unit LN.
6.	FSMB:	: Representative of a class of finite state machines used for processing binary input units.
7.	∅FSM(S)	: A finite state machine for printing strings. Input is a strings of characters which are converted to external code and output on current unit LN. Returns NIL
8.	∅FSMB	: Representative of a class of finite state machines used for processing binary output units.
9.	SETFSM(FSM RDR LN)	: Attaches reader RDR to finite state machine FSM by setting FSMs' OWN variable to RDR, and by setting LN as the value of RDR's current unit OWN variables. RDR uses LN to initialize its OWN parameters (see RDRC). Sets RDR as the value of attribute 'RDR' on property list of LN. Returns FSM.
10.	COPYRDR (RDRX RDRY)	: Copies reader RDRX. Name of the copy is RDRY. All OWN parameters of RDRY are identical to those of RDRX. Marks RDRY as copy (see REMRDR). Returns RDRY if copy successful, or NIL if unsuccessful, (no RDRX, or no room for copy).
11.	CLEARDR (RDR)	: Reinitializes all OWN parameters of reader RDR. Removes RDR from property list of LN(LN available as value of an RDR OWN variable). Returns RDR.
12.	REMRDR (RDR)	: If RDR is a copy of an existing reader (COPYRDR marks all copies) it is deleted and RDR is removed from property list of LN if there. If RDR is not a copy, REMRDR does nothing. Returns NIL.

13. TEREAD (LN)	:	Locates value of RDR on LN and evaluates CLEARDR for that value. Returns LN.
14. TERPRI (LN)	:	Evaluates MOVEOUT (LN)and reinitializes finite state machine ØFSMS. Returns LN.

LISP II I/O SYSTEM

SCHEMATIC DIAGRAM

FIGURE 1



KEY

LN - LOGICAL NAME of current unit
rdr - current reader (RDR) may be set to:
    RDRC - read characters
    RDRCS - read characters in a special way
    RDRB - read binary
fsm - current finite state machine (FSM) may be set to:
    FSMC - character token FSM
    FSMS - source language token FSM
    FSMB - binary token FSM
SETFSM - Attaches a FSM to a RDR for a logical unit LN.