MEMORANDUM

To: SDS 940 LISP Users

From: D. G. Bobrow, L. P. Deutsch, D. L. Murphy

Subject: General Structure of LISP 1.69

Date: 23 March 1967

I. INTRODUCTION

This is a preliminary memo describing the BBN LISP 1.69 system for the SDS 940 computer. It is a description of how the system is working now, except for those places clearly noted in the text below. Any difference between the descriptions given and actual operation found should be reported, in writing, to the authors. At the end of this memo there is a copy of the index to function descriptions in the document, "The BBN LISP SYSTEM (revised October 1966)." Those functions marked with an * have been changed from the description in that manual. Those marked with ** have been omitted from the current system. Some functions not marked are no longer in the library, but are built-in subr's. Since these changes are unimportant in terms of use of the system, they will not be noted here. The report mentioned is in short supply. If you do not have one, and would like one, please send a note to Rita Doherty. When the manual is reissued we will send you one. This will probably not be for a period of at least one month. The description below is intended to summarize the differences between 940 LISP and both the BBN PDP-1 system, and the 7090 LISP 1.5 system.

II. FUNCTION TYPES

There are basically eight function types in the BBN LISP System. These eight types are characterized by three dichotomies. A function may independently have:

1. its arguments evaluated or unevaluated,
2. a fixed number of arguments to be spread (paired with argument names),
3. be defined by a LISP expression or by machine code.

The latter class of functions, defined by machine code, were written in the system as subr's, or were compiled from expressions. The basic distinction between these two subtypes of the code class is that subr's are always in core, while compiled function code is loaded as needed from the drum. The three major distinctions will be testable by three functions, evalp[fn], spreadp[fn], and exprp[fn]; the value of each of these predicates will be T for ordinary uncompiled LAMBDA expressions with a list of arguments.

Expressions used to define functions must start with either LAMBDA, or NLAMBDA; indicating that the arguments of this function are to be evaluated, or not evaluated, respectively. Following the LAMBDA or NLAMBDA is any atom (except NIL) or a list of atoms (possibly empty). If there is a list of atoms, each atom in the list is the name of an argument for the function defined by the expression. Arguments for the function will be evaluated or unevaluated, as dictated by LAMBDA or NLAMBDA, and paired with these argument names. If an atom follows the LAMBDA or NLAMBDA, this function has an

indefinite number of arguments. If it is an NLAMBDA expression, then the atom is paired to the list of arguments (unevaluated) of the function; that is, to CDR of the form in which this function name was CAR.

The case where an atom follows a LAMBDA is not yet implemented, but will be shortly. It is our intention to evaluate each of the arguments in turn, spread them out on the push-down list, and bind to the atom following the LAMBDA the number of arguments, n, that have been placed on the push-down list. It is proposed that there exists a built in function arg [m] which will extract the mth argument of this function from the push-down list. This latter function will be undefined for m<Ø, will be equal to the number of arguments, n, for m = 0, and will be undefined for m>n.

A standard feature of the BBN LISP system is that no error occurs if a function is called with too many or too few arguments. If a function is called with too many arguments, the extra arguments are evaluated but ignored. If a function is called with too few arguments, the unsupplied ones will be delivered as NIL. This applies to both built in and defined functions.

There is a function PROGN of an arbitrary number of arguments, which evaluates the arguments in order and returns the value of the last (i.e., it resembles and is an extension of PROG2).

The conditional expression has been generalized so that instead of doublets it accepts n+1-tuplets which will be interpreted in the following manner:

```
(COND
   (P1 E11 E12 E13)
   (P2 E21 E22)
   (P3)
   (P4 E41))
```

will be taken as equivalent to (in LISP 1.5):

```
(COND
   (P1 (PROGN E11 E12 E13))
   (P2 (PROGN E21 E22))
   (P3 P3)
   (P4 E41) (T NIL))
```

This is not exactly true, but only because P3 is not evaluated a second time, if needed in the third item in the second conditional expression. Thus, a list in a COND with only a predicate and no following expressions causes the value of the predicate itself to be returned. Note also that NIL is returned if all the predicates have value NIL. No error is invoked.

LAMBDA and NLAMBDA expressions also have implicit PROGN'S; thus for example

```
(LAMBDA (V1 V2) (F1 V1) (F2 V2) NIL)
```

is interpreted as

```
(LAMBDA (V1 V2) (PROGN (F1 V1) (F2 V2) NIL))
```

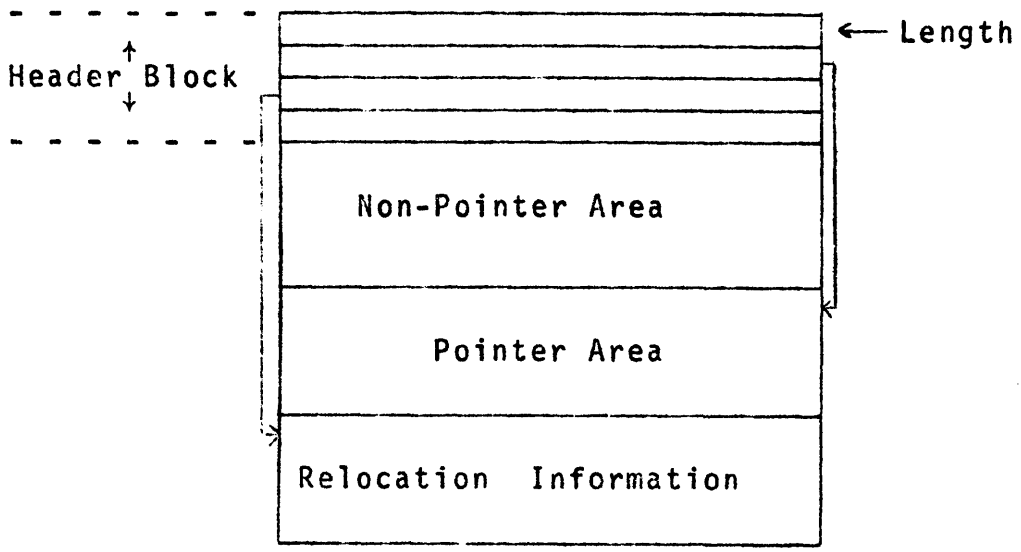
The value of the last expression following LAMBDA (or NLAMBDA) is returned as the value of the expression.

III. DATA TYPES

At the current time BBN 940 LISP has as data types literal atoms, list cells, small integers, and large integers. It does not have floating point numbers, although these will be added in the not too distant future.

A literal atom is known to be an atom by its address. Therefore it has no need to have a special symbol (such as -1) in the CAR of the atom. Literal atoms have four cells associated with them. These are called respectively the CAR, CDR, PNAME cell and function cell of the atom. CAR of the atom always contains the top level binding of the atom. If the atom has not yet been bound, the value cell contains the special atom NOBIND. CDR of the atom is a pointer to the atom property list, initially NIL. The PNAME cell contains a pointer to a packed character table which contains the print name of the atom. The function cell contains NIL until a function by that name is defined. There is no check in the functions CAR and CDR for atoms (except numbers); thus, one can access directly the top level value and the property list of an atom with CAR and CDR. One implication not immediately obvious is that CAR [NIL] = NIL, and CDR [NIL] = NIL. These latter two values are a significant convenience in programming.

Arrays are also available in BBN LISP. Arrays are identifiable as such by their position in the address space.



Typical Array

Every array is divided as shown in the figure. The HEADER BLOCK is four cells long and contains:

Cell: Ø Length of entire array.
1 Relative address of first word of protected pointers.
2 Relative address of first word of relocation information.
3 Not presently used.

An array may contain both pointer and non-pointer data, separated as shown. Pointer data is assumed to be one of the standard LISP types, and the pointer data cells in all

arrays are used as base cells for tracing during garbage collection. The non-pointer data, beginning in the fifth cell of the array, is of unrestricted type, and will not be used as trace pointers during garbage collection.

Relocation information contains the relative addresses of cells in the array which are to be relocated when the array is used as a compiled function, and is placed in core memory.

Examples:

1. Compiled code.
   a. Machine instructions and unboxed numeric literals are in the non-pointer area.
   b. Other literals and variable name pointers are in the pointer area.
   c. Relocation information area addresses all machine instructions whose address is within the same program, e.g., BRANCH instructions.

2. Array of lists.
   All data would be in the pointer area; the other areas would be of length Ø.

3. Array of unboxed numbers.
   All data would be in the non-pointer area; the other areas would be of length Ø.

Arrays are manipulated by three functions:

    array [n]

which creates an array of length n, and whose value is a pointer to the array.

elt [a;i]

which gets the ith element of the array pointed to by a,

seta (a, i, v)

which sets the value of array a, element i, to the value of v. The value of seta is the value of v. Special provision will be made in the compiler for handling arrays efficiently.

IV. INPUT/OUTPUT FUNCTIONS

A. Opening and Closing Files

BBN 940 LISP 1.69 allows the user to have any number of files open at a given time. Restrictions in the time-sharing system currently limit this to a maximum of 2, however. A file is identified by its LISP File Number (abbreviated L.F.N.) while open; actual file names are only used in the opening process and in the 'SYSIN' and 'SYSOUT' functions.

The three basic file manipulation operations are:

infile [name; type]

used to open the file named "name" (which must be of type 'type' (i.e., binary, 2, or symbolic, 3) if 'type' is not NIL) for input: its value is the L.F.N. for the file if it was opened successfully, or NIL otherwise.

outfile [name; type]

opens the file 'name' (which is set to type 'type' if 'type' is not NIL, and otherwise to type 3, symbolic) for output; its value is the L.F.N. or NIL as for infile.

closef [file]

closes a file, where 'file' is the L.F.N. of the file to be closed. Its value is NIL.

At any given time one input and one output file are selected as primary (the exact meaning of this is given below). Normally these are respectively files Ø (teletype input) and 1 (teletype output). The primary input file may be changed by

```
input [file]
```

which sets the file whose L.F.N. is 'file' to the primary input file. Its value is the L.F.N. of the old primary input file. Similarly, the primary output file may be set with

```
output [file]
```

which has the obvious effect. To read the current setting of the primary input or output file, either

```
input []
```

or

```
output []
```

may be used.

Obviously if other devices are added, they can be given standard numbers and specified as above.

B. Input/Output Transmission

Without exception, functions that actually read or write on files may be given an additional argument which is the L.F.N. for the file on which the operation is to take place. In fact, if the additional argument is 'NIL,' the primary file will be used just as though the argument had been omitted. This is a special case of the observation about omitted arguments for functions.

The following functions perform output:

feed [w]

produces 'N' carriage returns and line feeds:

prinl [a]

prints its argument.

prin2 [a]

prints the atom 'A' with double-quote marks inserted where required for it to read back in properly; both prinl and prin2 print lists as well as atoms. Neither print a carriage return upon termination, both have value 'a'.

print [x]	Prints the S-expression 'x'; uses prin2;
spaces [n]	Produces 'n' spaces;
terpri [ ]	Produces a carriage return and line feed;

The following functions perform input:

ratom [] Reads the next atom. The atom is delimited as specified by setsepr and setbrk. At present, setsepr and setbrk can only take numbers as arguments. This will be changed to allow characters. Meanwhile, chcon described below is most useful.

read [ ] Reads the next S-expression.

readc [ ] Reads the next character.

C. Input/Output Control Functions

These functions perform a variety of operations on the state of files. Those marked with * do not take the optional extra argument to indicate a file.

clearbuf [ ] Clears the input buffer of the file (not particularly useful for any file other than the teletype):

*control [j] Sets modes for reading with RATOM as follows:

J = T Eliminates LISP'S normal line buffering (and also eliminates automatic detection of control-A and control -Q as line-editing characters on the TTY):

J = NIL	Restores line buffering (normal).
J = 0	Eliminates the echo of the character being deleted by control -A:
J = 1	Restores the echo (normal).
*linelength [N]	Sets the length of the print line for all files. The value is the former setting of the line length;
*position [ ]	Gives the character position on the print line. No guarantees are made about its meaningfulness if output is being done intermittently to more than one file.
*'readp [ ]	Gives 'T' if there is something in the input buffer (either the TSS input buffer or LISP's line buffer) and 'NIL' otherwise.

D. Special Functions

sysout [name]	Dumps the entire state of LISP on the file named. This name should not specify a drum file, since more than 38K of information (the maximum for a sequential drum file) will always be written.

sysin [name] Restores the state of LISP from a sysout file. SYSIN may only be done once after entering LISP.

V. OTHER FUNCTIONS IN BBN 940 LISP

allocate [n] = Allocates n word of an array. Returns a pointer to the address of the first word allocated.

apply [fn; arglist] = as in LISP 1.5 with no A-list.

character [n] = Creates an atom whose print name is the character whose code is n.

chcon [x;j] x atom
Returns a list of numbers representing characters in print name.

j = NIL prin1 representation
= T prin2 representation (not implemented at present)

clock [j] for J = 1, 2 as before. For J=3 it gives number of seconds of compute time exclusive of garbage collection.

closer [a;x] = Legal only if one has immediately preceded it with an openr [T]. Stores 'x' into location 'a'. Both 'x' and 'a' must be numbers.
$a<2^{14}$ actual core location.
$a\geq 2^{14}$ address in virtual address space.

conscount [ ] = Number of conses since LISP started up.

load [f, name] = Loads from file named, and types values if f ≠ NIL.

loc [x] = Makes a number out of x

ilp [x] = Unrestricted car of x with error checks.

logout [ ] = Exits you from LISP

logxor [x_1; ...; x_n] = Logical exclusive or of 'x',,...,'x_n'

max [x_1;...; x_n] = Maximum value of 'x',;...;'x_n'

min [x_1;... x_n] = Minimum value of 'x_1'; ...;'x_n'

openr [a] = Value is number in a as defined in closer. If a = T this allows you to do one closer.

trace [ ] = This is now an NLAMBOA and uses BREAK1. It takes an indefinite number of arguments. It will ask for ARGUMENTS for subr's and compiled functions. Give an atom or argument list as required. Do not trace fsubr's like plus.

VI. USING LISP ON THE 940

In order to use LISP you should have in your file directory an entry which points to the file on tape which is a sysout file of the basic system. This basic LISP system file, called BSCSYS on Bobrow's files, contains a binary image of LISP after it has been initialized and loaded with the library. You do not need a copy of the library if you have this file. If you do not have such an entry consult D. Murphy.

Call LISP by typing LIS; the system will respond P; you type .; when LISP finally responds READY, type the following:

```
SYSIN (BSCSYS)
```

After typing the above, terminated by a carriage return, the system will find that file on the tape. When it has read it successfully, it will respond with a T. At this point you are talking to LISP evalquote (indeed you were talking to evalquote before you did the SYSIN).

When typing into the computer, typing a control-Q will clear the entire input line buffer erasing the entire line up to the last carriage return. Typing control-A erases the last character typed in but will not go beyond the last carriage return. Using these line editing features, the user will get echoing as determined by the function control[j] as defined above.

To abort a type-in completely one can press the RUBOUT key. Pressing it once will just ring a bell; pressing it a second time will cause control to go back to evalquote. Pressing the RUBOUT button while an evaluation is in progress is like pressing the BREAK key on the PDP1 LISP, or on Project MAC 7090 LISP. It causes an untrace unless an errorset is in force. A second RUBOUT will cause control to be immediately returned to evalquote.

VII. Index to Functions Described in AFCRL-66-180 (Revised)

In the following, * indicates the function has been modified as described; ** indicates the function has been omitted from 940 LISP 1.69.