TLISP
TOPS-20 Native-mode Yale/Rutgers/UCI LISP

Nathaniel Mishkin

8 January 1981

. Introduction

LISP is a version of Yale/Rutgers/UCI LISP which runs entirely in native mode
n TOPS-20; i.e. it has no dependencies on the TOPS-10 monitor or the TOPS-20
ompatibility Package (PA1050). This document will describe the differences
etween TLISP and the LISP described in The New UCI LISP Manual.

LISP is most closely related to "new" UCI LISP (hereafter, UCILISP) although a
ew features of UCILISP are not in TLISP and TLISP has some features not in
CILISP. The "history" of TLISP is:

```
Stanford -> UCI -> Rutgers -> Yale LISP -> "new" UCI LISP
                                  !
                                  v
                              Yale TLISP
```

. New Functions and Features

OOP

```
(LOOP (INITIAL v1 e1 v2 e2...) INITIALialize variable vi to ei
      (BEFORE e1 e2 ...)       executed before loop, after init.
      (STEP v e1 e2 [e3])      step v by increments of e3 (1 if
                                  omitted) from e1 to e2
      (WHILE e)                continue loop only if e
      (DO e1 e2 ...)           body of loop
      (NEXT v1 e1 v2 e2...)    after body of loop set the vi to ei
      (UNTIL e)                stop loop if e
      (AFTER e1 e2 e3 ...)     executed after loop, before return.
      (RESULT e))              when loop ends, return e
```

All key-phrases are optional. For clarity, the INITIAL and RESULT clauses
should be at the beginning and end, respectively and BEFORE shoul
immediately follow INITIAL and AFTER should immediately precede RESULT; th
others can appear in any order and any number of times. Note that WHILE,
UNTIL and RESULT take only one expression.

There may be only one STEP clause per LOOP. The increment/decrement occurs
at the very end of the loop. The exit condition for STEP is for v > e2,
for positive e3 and v < e2 for negative e3. e2 is evaluated only once in
the LOOP.

LOOP translates into a PROG body of the following form:

```
    (PROG (variables)                            from INITIAL
          (SETQ vi ei)...                        from INITIAL
          e1 e2 ..                               from BEFORE
     LOOP body                                   from DO, WHILE, UNTIL & NEXT
          (GO LOOP)
     EXIT e1 e2 ...                              from AFTER
          (RETURN return expression))            from RESULT
```

The body of the PROG loop is built by translating the LOOP clauses, in the order which they appear, thus:

```
        (OR (while expression) (GO EXIT))        from WHILE
        expressions...                           from DO
        (SETQ vi ei)...                          from NEXT
        (AND (until expression) (GO EXIT)        from UNTIL
```

NOTE: (LOOP (INITIAL ...) ...) is not the same as (LET (...) ...). (LET (
1 Y X) ...) sets Y to the old value of X and X to 1; (LOOP (INITIAL X 1
X) ...) sets both X and Y to 1.

NOTE: (LOOP ...) loops forever if there is no WHILE or UNTIL (even if there is a RESULT).

NOTE: since the WHILE, DO and UNTIL clauses can be put anywhere, they are taken as they come and the expressions they imply are appended together in between the INITIAL and the RESULT expressions.

OR

```
    (FOR -(v IN l)-
         (WHEN -exps-)
         (DO | FILTER | SPLICE | SAVE | -exps-)
         | (STOP)
     )
```

Builds a mapping function with LAMBDA variables and arguments taken fro
the (v IN l)s. At least one (v IN l) must appear and one of the set DO,
FILTER, SPLICE, SAVE, STOP. DO means no results are saved, SAVE means kee
the results in a list. FILTER means keep the non-NIL results in a list,
SPLICE means NCONC the results together into a list. WHEN is optional. The
DO, etc is done only if the WHEN test is non-NIL. A missing WHEN is
effectively the same as (WHEN T). WHEN-STOP produces a SOME expression.

NOTE: certain forms are optimized -- e.g., like (FOR (X IN L) (DO (FOO X)))
to (MAPC 'FOO L). This is incorrect if FOO is a macro. FOR will not
optimize such functions if MACRO or *MACRO properties are non-NIL.

Some sample FOR expansions:

```
(FOR (X IN L) (DO (FOO X)))
(MAPC (FUNCTION FOO) L)

(FOR (X IN L) (SAVE (FOO X)))
(MAPCAR (FUNCTION FOO) L)

(FOR (X IN L) (SPLICE (FOO X)))
(MAPCAN (FUNCTION FOO) L)

(FOR (X IN L) (FILTER (FOO X)))
(MAPCAN (FUNCTION
          (LAMBDA (X) ((LAMBDA (X) (AND X (NCONS X))) (FOO X))))
         L)

(FOR (X IN L) (FILTER X))
(MAPCAN (FUNCTION (LAMBDA (X) (AND X (NCONS X)))) L)

(FOR (X IN L) (WHEN (FOO X)) (STOP))
(SOME (FUNCTION FOO) L)

(FOR (X IN L) (WHEN (FOO X)) (SAVE X))
(SUBSET (FUNCTION FOO) L)

(FOR (X IN L) (WHEN (FOO X)) (SPLICE X))
(MAPCAN (FUNCTION (LAMBDA (X) (COND ((FOO X) X)))) L)

(FOR (X IN L) (WHEN (FOO X)) (FILTER X))
(MAPCAN (FUNCTION (LAMBDA (X)
          (COND ((FOO X) (AND X (NCONS X))))))
         L)

(FOR (X IN L) (WHEN (FOO X)) (DO (BAZ X)))
(MAPC (FUNCTION (LAMBDA (X) (COND ((FOO X) (BAZ X))))) L)

(FOR (X IN L) (WHEN (FOO X)) (SAVE (BAZ X)))
(MAPCAN (FUNCTION (LAMBDA (X) (COND ((FOO X) (NCONS (BAZ X)))))) L)

(FOR (X IN L) (WHEN (FOO X)) (SPLICE (BAZ X)))
(MAPCAN (FUNCTION (LAMBDA (X) (COND ((FOO X) (BAZ X))))) L)

(FOR (X IN L) (WHEN (FOO X)) (FILTER (BAZ X)))
(MAPCAN (FUNCTION (LAMBDA (X)
          (COND ((FOO X)((LAMBDA (X) (AND X (NCONS X)))(BAZ X))))))
         L)
```

If there is more than one expression in the WHEN clause, PROGN is used.

INCHAN OUTCHAN
(INCHAN FILSPEC) opens the file given by FILSPEC for input. (OUTCHA
FILSPEC) opens the file given by FILSPEC for output. FILSPEC shoul
evaluate to (DEV DIR FILE). DEV and DIR are optional. Useful as an EX
version of INPUT and OUTPUT. If FILSPEC is atomic, then it will be treat
as (FILSPEC) for convenience.

ALTNAME
(ALTNAME fn1 fn2) puts fn1's function definition under fn2

LET

```
(LET (L1 V1 L2 V2...) X1 X2...) =>
     ((LAMBDA (L1 L2...) X1 X2...) V1 V2...)

(LET FOO (L1 V1 L2 V2...) X1 X2...) =>
     ((LABEL FOO (LAMBDA (L1 L2...) X1 X2...)) V1 V2...)
```

LET is useful for allocating and initially assigning local variables and i
much cleaner than PROGs.

OUTAPP
Like OUTPUT except appends to existing file instead of writing to a ne
file.

DSKAPP
Version of DSKOUT which uses OUTAPP instead of OUTPUT.

^N PRINTPROPS
PRINTPROPS prettyprints all the important properties of atom X -- even i
the property value fits on one line, it is SPRINTed in order to ge
printmacros (Mark Burstein's idea) -- returns X ^NATOM => (PRINTPRO
'ATOM) (^N = Datamedia "PRINT" key)

EXPRPDL EXPSPDL
REALLOC functions for the regular and special pushdown lists

APPEND1
(APPEND1 L X) APPENDs X to the end of L, but it doesn't change L

DEX DFX DMX
Like DE, DF and DM except all macros in the function body are expanded a
define time.

*FOPEN
Primitive file opening routine. (*FOPEN file GTJFN-bits OPENF-bits) gets
JFN and opens "file" (which must be a string) using as the flags bits fo
the GTJFN and OPENF calls the 2nd and 3rd arguments respectively. I
OPENF-bits is NIL, the JFN is gotten but the file is not opened. Return
the JFN returned by the GTJFN. A more useful file open function, FOPEN, i
documented below.

OPEN

Function for obtaining and opening JFNs. (FOPEN file access size) gets a JFN and opens "file" (which must be a string) using the access information specified in the string "access". The file is opened for byte size "size". If "file" is NIL then the file name will be read from the primary input using the recognition feature

FOPEN returns a JFN for file or NIL if call fails.

"Access" is a string containing single character codes:

R - Read access to the file
W - Write access to the file
A - Append access to the file
U - Read, write access to the file

One or more of the above must be specified. All of the remaining letters are optional.

C - Allow concurrent access by other jobs even if they want to write on the file.
X - Exclusive access to the file.
G - Use next higher generation of this file.
N - This must be a new file.
E - This must be an existing file.
T - This is a temporary file.
D - Consider files marked as deleted when searching the directory.
B - Block this process until the requested access is granted.
P - Don't update the access dates of this file.
L - Don't check this file for line numbers.
I - Allow only one process to access the JFN associated with the file
i - Don't search search lists
* - Allow wildcards in the file name.
! - Get a JFN for file, but don't open the file

HELP

On line help function. (HELP function-name) prints out some documentation on "function-name".

SEXIT

The only way to exit TLISP leaving things in a state for being saved as a .EXE file. (SEXIT NIL) unmaps the shared high segment. (SEXIT T) keeps it around.

URRCOL
CURRCOL is an alternate name for CHRPOS.

loating point output
The printing of floating point numbers has been improved (and corrected) The user now has the ability to control the format of floating poin numbers by setting *FP-FORMAT to the format control word used by the FLOU JSYS (see the TOPS-20 Monitor Calls Reference Manual if you really want t mess with this). *FP-FORMAT is initially zero which produces free form (and generally acceptable) output. FPFORM.LSP contains some usefu functions for munging with the format word.

Window Package
TLISP has hooks in it to facility a display window facility. These hook may be ignored. See <F.LISP>WW.LSP.

3. TLISP Functions Which Work Differently from their UCILISP Counterparts

File Names
Directory names may be specified in file names; e.g. <MISHKIN.LISP>FOO.L would be specified as (INPUT T (MISHKIN LISP) (FOO . LSP)), <TOOLS>BAZ a (INPUT T (TOOLS) BAZ). .LSP is assumed as a default extension in INPUT i a file with no extension is specified and the file without the extensi does not exist. File names are not limited to six characters and thr characters for extensions.

Also, a valid TOPS-20 file name may be specified as a string (e.g (INPUT "<TOOLS>BAZ")), however the .LSP default extension logic will not apply i this form is used.

INPUT OUTPUT
The I/O system has been rewritten. INPUT and OUTPUT now take only one fi name as an argument. I/O functions return JFNs (which are numbers) inste of "channels". These changes should be mostly invisible. The channel na argument to INPUT and OUTPUT which was formerly optional is now mandator if it is not T it will be set to the JFN assigned and returned by INPUT a OUTPUT.

DSKIN
Calling DSKIN with no arguments will result in the user being prompted f a list of files, one per line, the list being terminated with a blank lin The TOPS-20 file recognition feature can be used.

INC OUTC
INC and OUTC now assume their first argument will evaluate to a numbe which is a JFN. This change is to make the changes to INPUT and OUTP transparent to the user.

INITFL
    Only one INITFL file may be supplied and it must be specified as a LISP string. Initially (INITFL "INIT.LSP") for TLISP and (INITFL "INITC.LSP") for TLISPC.

%RENAME RENAME
    The arguments to the rename functions must be complete file names. No fields of the rename will be defaulted.

GET
    GET checks to see if its first argument is a LITATOM.

SORT
    SORT is compatible with the old SORT but is much faster.

DTIME
    DTIME is only accurate to the nearest second.

RUN
    RUN is a runtime function. See description of the runtime functions below

PUSH
    The argument order is reversed from the old PUSH. (PUSH X L) is (SETQ L (CONS X L)). Returns the element pushed onto the list.

SETSYS
    SETSYS takes one argument which is a string file name of the file to ge the TLISP sharable high segment from.

SETCHR
    SETCHR takes a character (i.e. one character atom) instead of a number.

PATH
    PATH is the CD path function. TLISP doesn't have the I/O PATH function.

4. UCILISP Functions not in TLISP
TLISP does not have the any of the functions associated with the the loader. Th following functions are also not in TLISP:

LOAD	EXCISE	UFDINP	SCAN	RDFILE
FILBAK	%FILEXT	PUTSYM	GETSYM	RPUTSYM
IRF	HGHIN	TYILIST	TYOLIST	<
*GETSYM	*PUTSYM	DSKLOG	NEQUAL	
RDFILENAME	CHRTAB	LISPSCAN	COMMENTBEGIN	COMMENTEND
STRINGBEGIN	STRINGEND	SLASHIFY	LETTER	LETTERP
DELLIMP	IGNORE	IGNOREP	UUO	DELLIM
EXARRAY	MYPPN	DELETE	RGETSYM	>
DEFAULTEXT				

The LISP structure editor is not built into TLISP but may be loaded fro <F.LISP>EDIT.LAP. It requires about 4500 words of binary program space to loa The FILPRO and FILBAK features are not in TLISP.

. Environmental Differences

AUTOEXP
    If *AUTOEXP is non-NIL then TLISP will attempt to do a directory expunge if it gets a quota exceeded error on output. Otherwise, TLISP will halt. *AUTOEXP is initially T.

LLFNS
    Function atoms are not saved under ALLFNS unless (GET 'ALLFNS 'ALLFNS) is non-NIL.

NOEXIT
    When *NOEXIT is T, the functions EXIT and SEXIT and the ^C^X and ^C^Z features will not cause TLISP to halt. These operations become no-ops. The *NOEXIT feature is useful for controlling the situations under which TLISP can be exited.

C
    ^C^P ^C^A do not exist in TLISP.

aving core images
    Saved core images must have been exited with SEXIT. Saving a core image exited with EXIT will not work.

NUM range
    The INUM range in TLISP is much smaller than in regular LISP. In TLISP the range is [-37777Q..37777Q] ([-16383..16383]). In UCILISP the range is [-177777Q..177777Q] ([-65535..65535]).

.. TOPS-20 Runtime Support

he runtime functions are a set of LISP functions designed to provide an nterface between LISP and the DEC-20 operating system. The runtime library is n two parts: the part built into TLISP and the part NOT built into TLISP. The ource for the first class is RUNTIM.LSP and for the second is RUN2.LSP.

ote about arguments: some of the functions take file names or other strings as rguments. A simple file name, such as FOO, can be typed directly, if quoted ith a single quote mark ('). More complex names, with extensions or irectories, should be typed inside string quotes ("), in which case no quote is eeded. Anywhere a directory may be given as an argument, a logical name may be iven instead. Most of the functions are EXPRs or SUBRs.

.1. Runtime Functions of General Interest

.1.1. Runtime Functions Built into TLISP

(C "file"). This is equivalent to typing "C file" to the EXEC, except that when you exit C with <control-C> or <control-Z>, you return to the LISP core image, not the EXEC.

```
    EXAMPLES:
    (C) ~ edit the most recently edited file
    (C 'FOO) ~ edit FOO
    (C "FOO.LSP") ~ edit FOO.LSP
```

(Z "file"). Run the Z editor. Just like C function except since the Z editor remembers its files better the Z function will not reload Z on each call.

Z, CC

(ZZ function1 function2 ...). This prettyprints the functions on a temporary file, edits that file and loads the file when you exit. Handy for patching functions. ZZ is an FEXPR.

```
    EXAMPLE:
    (ZZ FOO BAZ)          ~ edit the functions FOO and BAZ
```

CC is like ZZ except the C editor is invoked.

XEC

(EXEC). Calls the system EXEC from within LISP. To return to LISP, type POP.

YS:

(SYS: "program-name" "arguments"). This is equivalent to typing "program-name arguments" to the EXEC, except that when the program finishes, you return to the LISP core image, not the EXEC. SYS_: is handy for defining your own functions to call system programs. C and <control-A> are defined with SYS_:.

```
    EXAMPLES:
    (SYS: 'MM)        ~ run MM
    (DE HELP (X) (SYS: 'HELP X))
                          ~ define HELP to call the system program HELP
                          ~     with X as an argument.
```

RUN

(RUN "directory" "program-name" "arguments"). This is what SYS: uses. It can be used to run a program from an arbitrary directory.

```
    EXAMPLE:
    (DE TDO (FILE) (RUN "<F.TOOLS>" "DO" FILE))
                                          ~ run DO program in <F.TOOLS> area.
```

ERUN
(RERUN "directory" "program-name" "arguments"). Like RUN except keeps around the fork created to run the program. Successive calls to RERUN with the same program will cause the program to be continued (like the EXEC' CONTINUE command) rather than reloaded and started. The Z and EXEC functions use RERUN.

MDSCN
(CMDSCN). The first time called, this returns a list of the items that appeared on the command line that called LISP. After the first time, NIL is returned. This can be used by functions in your INIT.LSP file to start LISP with different switches.

```
EXAMPLES:
If you started LISP with LISP, then (CMDSCN) returns (LISP).
If you started LISP with LISP 20, then (CMDSCN) returns (LISP 20).
If you started LISP with LISP -A -B, then (CMDSCN)
    returns (LISP -A -B).
```

.1.2. Runtime Function NOT built into TLISP
he following functions ARE NOT BUILT INTO TLISP. They may be used by DSKINing UN2 LSP or RUN2.LAP from the TLISP source directory.

A
(control-A)program-name arguments. This is a readmacro which makes it easy to call system programs. The line is read and made into a string so no string quotes are needed.

```
EXAMPLES:
^AMM             ~ run MM
^AHELP PHOTO     ~ ask for HELP about PHOTO
^AC FOO.LSP      ~ edit FOO.LSP
```

ELETE-FILE
(DELETE-FILE "file" expunge-flag). Deletes "file" and expunges just that file if expunge-flag is not NIL.

AGELEN
(PAGELEN size). Sets the terminal page length. Like the EXEC TERMINA PAGE command. (PAGELEN 0) is useful for when a program is about to generate a lot of output since it stops the monitor from pausing output to the terminal but leaves the cntl-S and cntl-Q functions on in case you want to manually stop output. Returns old page length.

STUFF
(STUFF "string"). This is like the EXEC DO command (simple version). The characters in the string are jammed into your TTY input buffer just as if you had typed them directly. Only the first 120 characters will fit.

.2. Runtime Functions of Interest to Hacker's

ne functions described here are the ones most likely to be useful to system ackers. They are all built into TLISP.

XECL

(EXECL directory file arg). This creates a fork under LISP, loads file.EXE from the specified directory into the fork and starts it with arg (if non-NIL) as part of the command line. EXECL returns a process handle (as a LISP number) if it succeeds, otherwise NIL. Any function calling EXECL should save the process handle, WAIT for it to terminate, then KILL it (see the definition of SYS_: in RUNTIM.LSP). Otherwise you will end up with a pile of unused forks. SYS_: and EXEC call EXECL.

ILL

(KILL process-handle). Takes a LISP number and kills the process with that number, if any.

AIT

(WAIT process-handle). Takes a LISP number and waits for the process with that number, if any, to terminate.

JSYS

(XJSYS jsys ac1 ac2 ac3 ac4). This executes the JSYS with the four accumulators set as given (0 if no argument or NIL is given). A JSYS is an atom with its SYM property set to the right instruction value (i.e., a 36 bit number with 104000 base 8 in the left half). If the JSYS executes correctly then the content of ac1 (converted back to a LISP number) is returned. If not, then NIL is returned. In either case, the global variable *VJSYS* is set to a list of the four accumulators (converted into LISP numbers).

```
EXAMPLE:
(XJSYS 'PBOUT 101Q)        ~ prints A if PBOUT has 104000000074Q
                           ~ under SYM and sets *VJSYS* to (65 0 0 0).
```

XJSYSF is just like XJSYS except it doesn't set *VJSYS* (and hence is a bit faster). Good for inside loops.

JSYSERR

(JSYSERR flag). Returns the most recent JSYS error message as a LISP string If flag is non-NIL, the string is also printed.

XWD

(XWD number1 number2). Makes a 36 bit LISP number with number1 in the left half and number2 in the right half.

```
EXAMPLE:
(XWD -1 (PACK string))  -1 in the left half, address in the right
(interpreted as byte pointer by most JSYS's)
```

ACK

(PACK string). Some JSYS's takes ASCIZ strings as arguments. PACK takes LISP string (non-strings are converted) and packs it into ASCIZ form, usin blocks from Binary Program Space. Old blocks are re-used when possible. PACK returns a LISP number which is the address of the BPS string.

ET-BLOCK

(GET-BLOCK size). Allocates a contiguous block of "size" words from binary program space. Returns address of block as a LISP number.

REE-BLOCK

(FREE-BLOCK address). This takes the address of a block built and returne by PACK or GET-BLOCK and marks it as free for use. Until a block is s marked, it cannot be re-used by PACK. Address is returned, so if you nee one string for one-time use by a JSYS you could write (XJSYS ... (XWD -1 (FREE-BLOCK (PACK string))) ...)

NPACK

(UNPACK address). This takes the BPS address (as a LISP number) of a ASCIZ string and returns the LISP string equivalent. It does NOT free u the block addressed.

EFI E-JSYS-LIST

(DEFINE-JSYS-LIST jsys-def-list). Defines symbolic names for TOPS-20 JSYSes.

```
EXAMPLE:
(DEFINE-JSYS-LIST '((CFORK 152Q) (ERSTR 11Q) (GET 200Q)))
```

EEK

(PEEK addr [index]). Returns the the contents of address "addr" optionall indexed by "index" as a LISP number.

```
EXAMPLES:
(PEEK 1000)     ~ get contents of address 1000 octal
(PEEK 1000 I)   ~ get contents of address 1000+I octal
```

OKE

(POKE addr value [index]). Sets the the contents of address "addr" optionally indexed by "index" to "value".

```
EXAMPLES:
(POKE 1000 777Q)     ~ set contents of address 1000 octal to 777Q
(POKE 1000 666Q I)   ~ set contents of address 1000+I octal to 666Q
```

ILENAME

(FILENAME jfn#). Returns the file name associated with "jfn#" as a LISP string.

ETCMD

(GETCMD). Returns the arguments for the current process as a LISP string. These arguments are obtained by looking in the PRARG block and if it is empty, the RSCAN command line buffer. CMDSCN uses GETCMD.

UTCMD

(PUTCMD "string" handle). Sets the arguments for the process indicated by "handle". "String" is placed both in the PRARG block (format is: first word is zero, remaining words contain packed ASCIZ version of "string") and the RSCAN buffer.

. Technical information

ee O-INFO-ON-TOPS20-LISP.YALE for details on the hacks which were made to utgers/UCI LISP to get it to its present state.

Table of Contents

```
 .ROLLINGER

TTT   L          III     SSSS   PPPP
T     L           I     S       P   P
T     L           I     S       P   P
T     L           I      SSS    PPPP
T     L           I         S   P
T     L           I         S   P
T     LLLLL      III    SSSS    P



DD      OOO      CCCC            999
  D   O     O   C               9    9
  D   O     O   C               9    9
  D   O     O   C                9999
  D   O     O   C                    9
  D   O     O   C          ..        9
DD      OOO      CCCC      ..     999


START* Job TLISP Req #948 for A.ROLLINGER     Date 21-Sep-82 10:56:53 Monitor: Ya
le PS:<A.AI-LIBRARY>TLISP.DOC.9, created: 11-Jan-82 17:10:09
       printed: 21-Sep-82 10:56:54
quest created:21-Sep-82 10:56:42 Page limit:54    Forms:NORM01
le parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```