# CP-6 LISP

## USERS GUIDE

by
Mats Nordstrom, June 1978.

Mailing address:
Datalogilaboratoriet
Sturegatan 2b
S-752 23 UPPSALA
S W E D E N

The original work was supported by the Swedish Board for Technical Development (STU) no 76-4253.

Revised May 1980 for CP-6 LISP
by Nick Briggs and Andrew Gullen,
Academic Support Group,
Carleton University Computing Centre,
Ottawa, Ontario,
CANADA.
K1S 5B6

TABLE OF CONTENTS

PREFACE

1. Primary datatypes.
2. Internal representations.
3. Predefined atoms.
4. I/O - handling.
5. Error handling, Break and Interrupts
6. Garbage Collection.
7. Edit.
8. Miscellaneous.

Appendix A: Functions in CP-6 LISP .
Appendix B: References.

PREFACE

CP-6 LISP is based on LISP F3, a LISP system written in FORTRAN. Except in the areas of memory management, break key handling and I/O, CP-6 LISP is almost identical functionally to the FORTRAN version. The first version (LISP F1) was written 1970-71 and has by now been delivered to about 100 different computer installations around the world.

LISP F1 was a LISP 1.5 system (with some extensions) but has now been almost completely rewritten into INTERLISP standard. CP-6 LISP is (almost) a subset of INTERLISP as defined in Te 74 or Ha 75. In addition it is about 3 - 10 times more efficient that LISP F1 and is easier to implement (as it is coded in a more structured style).

As a users manual, Ha 75 is referred to (and delivered together with the system) and in this guide only differences from INTERLISP are reported.

Some of the functions in CP-6 LISP are coded in LISP and in the following it is assumed, that all those LISP-packages are included in your system. (Check with your installation manager!).

LISP may be invoked with
!LISP and the library (which contains many of the functions described in this manual) may be read in with
(INPUT (OPENFILE 'LIBRARY.LISP)) This file will close itself when finished. Once the library has been read in, you may save a clean LISP memory image, and avoid having to reread the library.

# CHAPTER 1

## PRIMARY DATATYPES

| | |
|---|---|
| small integers | range -10239,10239 a value which is dependent on the memory occupied by the code portion of the LISP interpreter. |
| large integers | a full word signed integer quantity. |
| alfanum atom | Maximum number of characters is 65535. |
| strings | given as "THIS IS A STRING", with a maximum of 65535 characters. |
| floating numbers | DO NOT EXIST. |
| lists | given as (A B (C D)) etc. |

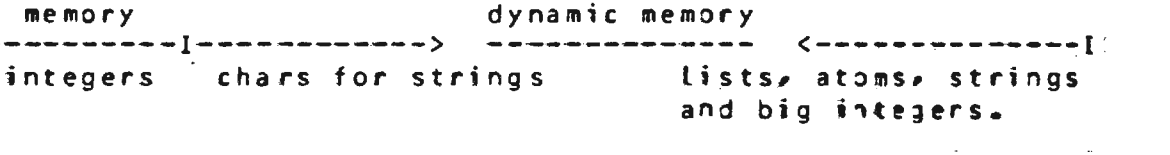
# CHAPTER 2

## INTERNAL REPRESENTATIONS.

A more complete description of the internal representations is given in the implementation guide. Here we only give the information needed for a complete knowledge and useage of CP-6 LISP from the user's point of view.

### a) The ADDRESS SPACE.

The address space is shown by the following figure:




The dynamic area of memory grows and shrinks as necessary to accomodate the users programs/data. The total size is limited by the size of the instruction segment, and user or installation defined memory limits.

### b) Literal ATOMS.

ATOMS are represented as a "three-pointer record" with some additional information which will not generally interest the CP-6 LISP user.

The global value of an atom is stored in CAR(atom). (EVAL checks for a bound value BEFORE a global value - as in INTERLISP but in contradiction to LISP 1.5). A global value may be set either by SET/SETQ at the top level, or directly by RPLACA. If a global value has not been assigned, CAR(atom) points to the atom NOBIND.

### PROPERTY LISTS

The property list of an atom is stored in CDR(atom). This can be any LISP object. Properties are normally accessed by the

functions GETP and PUT.

PRINT NAMES

The PRINT NAME of an atom is pointed to by a (non LISP) pointer in the atom record, in a manner similar to the storage of a string.

FUNCTION DEFINITIONS

In INTERLISP each atom-record also has a "function field" called the function cell (Ha 75 page 4). In CP-6 LISP user defined functions are stored as LAMBDA or NLAMBDA expressesions on the FNCELL. A SUBR or FSUBR is recognized by 'branding' the atom-pointer itself but in order to simulate the facility of making use of "free function indicators", GETD is defined to return (SUBR . FOO) if FOO is a hand coded SUBR, and (FSUBR . FOO) if it is an FSUBR.

The forms (SUBR . FOO) and (FSUBR . FOO) are simulated function indicators and legal function arguments to APPLY.

Example:

```
(DE KAR(X) ((SUBR . CAR) X]
```

This definition of KAR causes KAR to behave exactly as CAR independently of whether CAR has been redefined to something else.

c) STRINGS and SUBSTRINGS.

STRINGS and SUBSTRINGS are represented in two parts, the string header, which contains a length and a (non LISP) pointer to the actual characters which make up the string.

- Two different strings may have the same print name.
- A string's value is always itself.
- A SUBSTRING cannot be distinguished from a normal string.
- A SUBSTRING shares characters with the parent string.

d) The SYMBOL TABLE.

The function
(OBLIST)
returns the actual OBLIST, as a list of lists - each of which is one of the non-NIL buckets in the hashed symbol table. As this is not a copy, care must be taken when doing operations on this list.

e) The ALIST.

Variable bindings are stored in an association list (as in LISP 1.5) but this list simulates a push down stack (as in INTERLISP) and is implicitly given to EVAL, APPLY and EVLIS.

The function
(ALIST)
returns the actual association list.
If evaluation is to be performed in some special variable environment use

(EVALA s assoc) - as (EVAL s) but uses assoc as the push down stack

(APPLYA fn l assoc) as (APPLY fn l) -"-

Ex.: A "safe" definition of GETPQ may look like

```
(DF GETPQ(A IND)(GETP (EVALA A (CDDR (ALIST))) IND]
```

I.e. the rebinding of A and IND, here done by GETPQ, is not seen inside the evaluation of A.

f) LISTS.

A LIST is represented as a two pointer record, the first field being the CAR, the second being the CDR.

g) NUMBERS.

The value of a small integer is the value of the pointer with a proper offset subtracted. The value of a big integer is stored in a full word hidden from the user (but found through its pointer value).

# CHAPTER 3

## PREDEFINED ATOMS

Here is a list of those atoms which may be of interest for the CP-6 LISP user.

| | |
|---|---|
| NIL,T | These atoms can not be destroyed by any functions such as RPLACA etc. |
| NOBIND | is stored in car of undefined atoms. |
| ADVISEDFNS | List of advised functions. |
| *BACKTRACEFLAG | if true, eval-apply will store forms under execution. This is needed to perform the command BT (backtrace) inside a break. |
| *BACKTRACE | List of forms under execution if *BACKTRACEFLAG = T. |
| BROKENFNS | List of broken functions. |
| USERFNS | List of those functions which have been defined before the first time (CURFILE file) was performed. |
| CURFILE | name of the current file (used by the MAKEFILE package). |
| *PRINTLEVEL | The printlevel used by TRACE. |
| *LASTERRORN | The most recent error number (NIL if no errors have occurred). |
| HELPFLAG | If NIL, the break package will not be called on errors. |

# CHAPTER 4

## I/O HANDLING

Though CP-6 LISP was designed to be as true a subset of INTERLISP as possible, there do exist some minor differences. Most of them have to do with I/O.

a) Input characters of special meaning.

. for dotted pairs. Must be separated by blanks!
A '.' which can not be interpreted as 'a dotted pair' is read as an atom.
% escape character
' QUOTE character
" " string character
[] super brackets

All these characters work in the same way as they do in INTERLISP.

\ 'rescue character'. When this character is seen in an s-expr by the read routine, LISP F3 will enter BREAK with FORM = 's-expr. (Useful for user-replacement at read-time).

b) Changing the meaning of special characters.

The "meaning" of all characters are stored in a table which is accessible by the function

| | |
|---|---|
| (CHTAB x) | Read the type of x. |
| (CHTAB n) | Return the current representative of class n. |
| (CHTAB x n) | Change the type of x to n. Returns old type. |

CHTAB uses the first character of the atom x.

The following character table is standard.

| type | means |
|---|---|
| 1 | space |
| 2 | ( |
| 3 | ) |
| 4 | [ |
| 5 | ] |
| 6 | " |
| 7 | ' |
| 8 | user break |
| 9 | . |
| 10 | alphanumeric |
| 11 | + |
| 12 | - |
| 13-22 | 0-9 |
| 23 | % |
| 24 | rescue character |

Ex.: If you want to have $ as a super bracket, and ] as an ordinary letter do:

```
(SETQ TYPE (CHTAB '%] (CHTAB 'A]
(CHTAB '$ TYPE)
```

and if you want to have * as a break character do

```
(CHTAB '* 8)
```

after which A*B will be read as the three atoms A * B separately.

Note that when LISP is printing or making a string, and wishes to print a character class 2-7, 9 or 23 it will use the last character to be defined as this class. Thus, if one makes $ a right super bracket, it will become the "current representative" of the right super bracket class, and will be used in printing immediately thenceforth, even if ] remains a super bracket. If the current representative of a class has its class changed, LISP will hunt for another, and replace the current representative if it can. If there is no replacement available, it will use a blank.

c) File Operations:

CP-6 LISP provides access to the basic CP-6 operations on sequential files (devices are treated as sequential files); that is, opening, closing, reading and writing. The functions provided are similar those of INTERLISP except that file version numbers are not supported and I/O is record-oriented rather than character-oriented. Both these INTERLISP characteristics could be simulated with LISP functions using the primitives supplied. However, since we are dealing with sequential files, INTERLISP character-addressable files are not implemented and only characters within the current record may be referred to.

CP-6 LISP maintains a special file whose name is T (actually two files, both open to device ME; one with access INPUT, one with access OUTPUT). This file cannot be closed or reassigned. An EOF or any other I/O error on T results in an exit to IBEX.

As in INTERLISP, there exist two special files, the primary input and primary output files, which are initially assigned to T. All I/O operations, unless otherwise specified, take place on these files.

The high-level read and write functions use a "current position" in the file buffers, this being the column position of the next character to be read or written, and refers only to its position within the current record. Functions are available to manipulate these. As well, the print buffer has adjustable left and right margins (such that 0 < left margin < right margin <= buffer length).

(FULLNAME file recog)

Returns the fullname for the file whose name is the printname of file (a fullname is the full fid for a file or device). The parameter recog may have value NEW or OLD (OLDEST is provided only for INTERLISP compatibility and is equivalent to OLD). If recog is NEW, any syntactically valid CP-6 fid will do for file; if OLD, a TEST open will be performed for disk files, an LDEV sought for streams, and a validity check made for devices. NIL will be returned if the check fails. The default for recog is OLD.

Examples :

```
-(FULLNAME 'TESTFILE)
DP#DISK/TESTFILE.PROJECT
-(FULLNAME 'LP@ 'NEW)
LP01@LOCAL
-(FULLNAME 'XX01#)
XX01#
-(FULLNAME 'JE)
JE#
```

(OPENFILE file access recog bytesize reclength)

Opens file if (FULLNAME file recog) succeeds, in a mode specified by access. Legal values and resulting CP-6 file access modes are :

| access | INPUT | OUTPUT | APPEND | BOTH |
|---|---|---|---|---|
| CP-6 mode | FUN=IN | FUN=CREATE, EXIST=NEWFILE | FUN=CREATE, EXIST=OLDFILE | FUN=UPDATE |

If NIL, the default is INPUT.

The parameter bytesize is provided only for INTERLISP compatibility, and although ignored, must be either NIL or a positive integer.

The parameter reclength specifies the size of buffer to be allocated for the file, and thus the maximum size of record which may be read or written. The default is 256 characters.

OPENFILE returns the fullname of the file opened.

Examples:

```
-(OPENFILE 'TESTFILE 'BOTH 'OLD)
DP#DISK/TESTFILE.PROJECT
-(OPENFILE 'L1 'OUTPUT)
LP01#
```

(CLOSEF file disp)

Closes file (which must be open); if file is NIL, then it will close the primary input file if this is not T, else it will close the primary output file if this is not T, else it simply returns NIL. If the primary input or output files are closed, either by default as above or explicitly, the primary file is set to T.

The parameter disp is the disposition to be used, legal values are REL or SAVE (default is no specification, use system defaults). The fullname of the file is returned.

Examples:
```
-(CLOSEF)
DP#DISK/LISP_OUT.PROJECT
-(CLOSEF 'TESTFILE 'REL)
DP#DISK/TESTFILE.PROJECT
```

(OPENP file access recog)

If file is non-NIL, it will be tested to determine if it is open with the access and recog modes specified (if NIL these will not be taken into account). The fullname is returned if open, otherwise NIL.

If file is NIL, a list of files open with the access and recog specified is returned (again, if NIL, these will not be checked).

Example:
```
-(OPENP)
(DP#DISK/TESTFILE.PROJECT LPO1@LOCAL DP#SYS/STANDARD.LIBRARY)
-(OPENP 'LP 'INPUT)
NIL
-(OPENP NIL 'INPUT 'OLD)
(DP#SYS/STANDARD.LIBRARY)
```

(INPUT file) and (OUTPUT file)

These change the primary input or output file to file, which must be open. If file is NIL, the setting is not altered. The old value is returned.

(READPOS n) and (PRINTPOS n)

Sets the "current position" within the buffer for the primary input or output file respectively to n (note once again that this is a position within the current record). If n is NIL, the value is not altered. The old value is returned.

(LEFTMARGIN n) and (RIGHTMARGIN n)

Set the left or right margin of the current primary output file to n. If n is NIL, the setting is not altered. The old setting is returned.

(PRINTDEPTH n) and (PRINTLENGTH n)

Set the maximum depth in a list to which the print routines will go before printing "...", and the maximum length of a list which will printed before terminating with "---". This parameter applies only to T, and will be ignored for other files. If n is NIL, the value is not altered. The old value is returned.

(BUFSIZE file)

Returns the buffer size of file; if file is NIL, then a list of the buffer sizes for the primary input and output files is returned. Since T has two buffers, file=T returns a list of the input and output buffers.

(GETREC file)

The file must be open in access mode INPUT or BOTH. The next record is read into the file buffer, destroying the previous contents. GETREC returns a substring of the buffer which is the record read. If this substring, or any part of it, is to be kept, it should be copied, as future I/O operations will usually

overwrite it.

An error will occur if EOF is encountered, see ERRORSET for user handling of errors. If file is NIL, the current primary input file is used.

(PUTREC file string)

The file must be open in access mode OUTPUT, APPEND or BOTH. The string replaces the contents of the file buffer, which is then written out to the file. If file is NIL, the current primary output file will be used.

The string written is returned.

(REWIND file)

The file is positioned at the beginning (including APPEND files). The buffer position will be set to one for INPUT and BOTH files, to LEFTMARGIN for OUTPUT and APPEND files.

(PRINTLEVEL carn cdrn) or (PRINTLEVEL '(carn . cdrn))

This works as in Interlisp, and sets PRINTDEPTH and/or PRINTLENGTH to carn and cdrn respectively.

(INFILE file) and (OUTFILE file)

The file is opened for input or output and made the primary input or output file. The old primary file is returned.

(INFILEP file) and (OUTFILEP file)

These test the eligibility of the file for INFILE and OUTFILE. NOTE - INFILEP does a syntax check and TEST open, thus 'file busy' will not be detected; OUTFILEP does a syntax check only.

(CLOSEALL disp)

Closes all files with disposition disp.

d) Changing standard behavior of CP-6 LISP.

The functions GCREPORT, PRETTYPRINT, PRETTYBREAK, PRINTQUOTE, and PRINTESCAPES are used to set flags which control the behaviour of the interpreter in the following ways :

```
(GCREPORT T)      - Report garbage collections.
(PRETTYPRINT T)   - Prettyprint output.
(PRETTYBREAK T)   - Begin a new line whenever (... is found,
                    unless it is the first (or sometimes second)
                    sub-expression.
(PRINTESCAPES T)  - Use " and % where necessary to permit output
                    to be read back in correctly (print vs print1).
(PRINTQUOTE T)    - Print (QUOTE x) as opposed to 'x.
```

e) Basic I/O functions

The following functions work as in INTERLISP except that they do not have a file argument.

```
(READ)        (RATOM)          (READC)
(PRINT x)     (PRIN1 x)        (PRIN2 x)      (TERPRI)
(EJECT)       (SPACES n)
```

(One minor difference for READ and RATOM though: If a number is too large to be handled as an integer, an alphanumeric atom is returned.)

As a matter of fact, PRINT, PRIN1 and PRIN2 are also defined in LISP using the one and only printing function PRIN0 which is defined as:

(PRIN0 x printtype flag1 flag2)

x value to be printed (No TERPRI before or after!).
printtype must be 0, 1, 2 or NIL (If NIL, 0 will be used).
Meanings are as follows (ref Sect 4d above) :
0 - ordinary print (fast print).
1 - prettyprint (no prettybreak).
2 - prettyprint and prettybreak.
if flag1 then print % or " when so necessary to read atoms back.
if flag2 then print (QUOTE s-expr) instead of 's-expr.

Note that if there are no atoms too long to fit between the margins then printtypes 1 and 2 (which move the left margin) will still produce output which can be read back in; however, only printtype 0, with flag1 T, is guaranteed to do so. Also, since an escape-requiring character in the middle of a pname requires one character for the escape, one for the character and one for the continuation escape, a minimum width of three characters is required (less makes it impossible to get any of the pname printed on a line, and infinite looping will result.

(PRINTL s1 s2 ...) and (PRINTL-SP s1 s2 ...)

perform PRIN1 on s1 s2 etc. followed by a TERPRI. (PRINTL-SP inserts spaces between each s-expr).

(NEWLINE)

will TERPRI if anything has been printed on the current line.

Also, the function

(PROMPTREAD prompt)

where prompt is a string or atom, will perform a (READ) using the pname of prompt as a prompt. It operates with interrupts inhibited so that the user cannot break out and leave the prompt

incorrectly set.

f) Save/Restore of the memory image.

The IBEX commands !SAVE and !GET may be used to save a core image of LISP. Care should be taken to close all files before SAVEing. The SAVE command may be embedded in a LISP function through the use of the LISP IBEX function. When the memory image is restored with !GET, execution will continue immediately after the call to IBEX. For example, the function SAVE is defined as

```
(LAMBDA (FILE EXPR)
        (AND (NULL (OPENP)
             (IBEX (CONCAT "SAVE OVER " FILE))
             (EVAL EXPR)))]
```

is used to save a memory image on FILE if no files are open and EVAL the s-expr EXPR when the image is restored. NOTE : If the save is successful, the value of EXPR is returned, otherwise NIL.

g) The makefile package.

This package is coded in LISP and follows the conventions for INTERLISP makefile (See He 76 page 98 for details). The only commands in FILEVARS which are implemented are:

```
        * (P ...) (PROP ...) (E ...)
```

Before doing MAKEFILE (or LOAD) you must however open the file by:

```
    (OPEN symfile io file)
    symfile your symbolic name
    io      I or INPUT for input file
            O or OUTPUT for output files
    file    external file name
```

and if you have no further use of the file you may close it with

```
    (CLOSE symfile)
```

The NOEVAL function

(CURFILE symfile)

defines the "current file" and all new functions defined afterwards belong to this file and will be added to the list symfileFNS. If (CURFILE symfile) is not evaluated, the name of the current symfile is USER, and the function names are saved on USERFNS.

The symbolic file names (including those from the library) are kept in a list on the atom CURLIBS.

Ex.: Define some functions and save them as your file MYFILE on the file SAVEFILE.

```
(OPEN 'MYFILE 'O 'SAVEFILE)
(CURFILE MYFILE)
(DE ..... ]
(DE ..... ] etc
(MAKEFILE 'MYFILE 1 80)
```

A pretty printed version of all functions is now written on SAVEFILE. (Argument nr 2 is used as PRIND's argument nr 2 (fast print - pretty print - pretty break) when the printing is performed. Argument nr 3 is used as the RIGHTMARGIN value during printing - defaults to file buffer length, which is defaulted to OPENFILE default.)

The symbolic file names (including those from the library) are kept in a list on the atom CURLIBS.

# CHAPTER 5

## ERROR HANDLING, BREAK AND INTERRUPTS

a) Error and Break

Almost all errors detected by CP-6 LISP call the function SYSERROR which is a SUBR and which calls RESET after printing a message. SYSERROR is then redefined in one of the standard LISP packages as a LAMBDA function which calls BREAK1 after the message. BREAK1 is the ordinary "break-function" and may therefore also have been called by a user setup break. Inside BREAK1 the following commands exist:

| | |
|---|---|
| !, ', STOP | return to previous break if any, otherwise RESET. |
| OK | continue |
| GO | print value of broken form and continue. |
| EVAL | eval broken form and break afterwards. The value of the form is stored in the atom !VALUE |
| !OK | as OK etc. but the function |
| !GO | is first unbroken |
| !EVAL | then rebroken |
| RETURN x | return the value of x. |
| UB | unbreaks the function. |
| BR | breaks the function. |
| BT | backtrace of function calls (only LAMBDA and NLAMBDA's). This is only possible if you have performed (SETQ *BACKTRACEFLAG T) before evaluation. |
| ALIST | prints the current value-binding stack (except for variables bound in BREAK1 and SYSERROR). |
| ? | prints a list of the break commands. |

any other input is evaluated and value is printed.

In addition to BREAK1, the functions BREAKO BREAK UNBREAK REBREAK TRACE are defined and work as in INTERLISP.

There also exists a function BREAK11, which is a LAMBDA version of BREAK1 (which in turn is a NLAMBDA) and a function UNTRACE.

Each error is associated with a number. The function

(ERRORN)

returns the number for the the most recent error, and

(ERRORMESS n)

prints out a corresponding message.

b) Program Interrupts

On CP-6, we have only one method of obtaining the system's attention during program execution; that is, the <break> key (excluding ctrl-Y, which is left for emergency access to IBEX). This results in some slight deviation in interrupt procedures from those of INTERLISP. One major unfortunate effect is that interrupting a program will cause all typeahead and queued terminal output to be lost.

To avoid confusion with the BREAK package, we will use the term "interrupt" instead of "break", except where, as above, we are referring to the actual key, as <break>.

First, interrupts may be armed or disarmed. If disarmed, then one interrupt, if attempted, will be stored for servicing on re-arming, the rest will be ignored. If armed, the request for attention will be honoured at the next "safe point". Note that a "safe point" is merely one at which CP-6 LISP can be interrupted and kept in a meaningful state, i.e. outside stack operations, garbage collection, etc. It is NOT a "clean point", at which the user's computation can be suspended without damage.

Some interrupt actions can be performed at a safe point without damaging the computation, e.g. requesting CPU time used. Others, however, require a clean point; one may elect to either wait for one, and preserve the computation, or to unwind back to one, and

accept the damage (damage will occur only if side effects have been produced by the interrupted function which will not be reproduced on restarting it). More on this under "interrupt hardness".

Arming and disarming are done with
(INTERRUPTABLE flag)
If flag is NIL, the arm/disarm flag will be set NIL, i.e. interrupts disarmed. Anything else sets the flag to T, for armed. If an interrupt has been stored while disarmed, arming will cause it to fire. Any such interrupt will be soft; thus a function may rearm interrupts as its LAST action, and any interrupt will occur after it has exited.

The status of the arm/disarm flag may be queried with
(INTERRUPTABLEP)
which returns NIL or T.

On honouring a <break>, the system will read a character from the terminal. All characters have an associated "interrupt class", which is an action to be performed on receipt of the character. The classes and their corresponding actions are:

| CLASS | ACTION |
|---|---|
| RESET | - perform a (RESET) |
| ERROR | - unwind to last break (as in "break package "), or to top level if none. |
| ERRORX | - cause "USER BREAK" error. |
| BREAK | - cause "BREAK" error. |
| HELP | - ask for system HELP (non-destructive) |
| CPUTIME | - print CPU time used since the beginning of the current LISP session (non-destructive) |
| NONE | - no action, continue (non-destructive) |

All characters also have an associated "hardflag". This determines whether the associated action will be performed immediately (hard - unwinding the current function if necessary and possibly damaging the computation), or at the next clean point (soft). The clean points are at form evaluation, either through APPLY or EVAL, thus a soft interrupt will not succeed in interrupting an infinitely looping SUBR (e.g. MEMB on an infinite list).

Character interrupt classes may be set with
(SETINTERRUPT char class hardflag)
which sets the interrupt class of char to class with hardness flag hardflag.

The parameter char must be a string of length=1, an atom with Pname of length 1, or a number char, 0 <= char <= 127 (for strings and atoms, 0 <= CHCON1 [char] <= 127). The class may be any atom except T or NIL, or any list, hardflag may be NIL or T. If class is a system-defined class, then the associated action will take place on receipt of char. If it is a literal atom other than one of these, the GLOBAL value of the atom will be set to T on receipt of char. If class is a list, it will be interpreted as a form to be EVAL'ed on receipt of char. This is done via the call (INTERRUPT fn args char), fn being the CAR of the form and args the CDR; INTERRUPT may be redefined. The interrupt character (as an atom) will be added as the last element of the form, thus the same form may be used for a number of characters and yet still be able to determine which was used.

SETINTERRUPT returns the old class and hardflag of char.

Example :
(SETINTERRUPT 'A '(INTFN 1))
(SETINTERRUPT "B" '(INTFN 1))
(SETINTERRUPT 7 'RESET T) will cause an interrupt character "A" to result in evaluation of the form (INTFN 1 'A); "B" will cause evaluation of (INTFN 1 'B), and an ASCII '007'O (bell) will cause a RESET.

The class and hardflag of a character may be queried with
(GETINTERRUPT char) where char is a string of one character or a number (0 <= char <= 127).

Conversely,
(GETINTERRUPT class hardflag)
returns a list of the characters with interrupt class of class and hardness of hardflag (a NIL value for class signals "don't care", however, NIL is a valid value for hardflag). Note that

(GETINTERRUPT atom) will be interpreted as GETINTERRUPT on a class.

To overcome the necessity of using two keystrokes in situations where the desired action is already known, a further facility is added, that of autointerrupts. Once the full flexibilty of the interrupt services are no longer required (e.g. in a debugged package), the response to the <break> key may be preset with

(SETAUTOINTERRUPT class hardflag)

where class and hardflag are as in SETINTERRUPT, except that NIL and T are allowed for class (see later). On <break>, no read will be performed; the action class will be applied automatically. Note that this should be used with care; if, for instance, the autointerrupt class is CPUTIME, and one gets caught in an infinite loop, the only option will be ctrl-Y, and loss of the workspace. Autointerrupt class of BREAK with hardflag NIL could also be fatal if caught in an infinitely looping SUBR.

If a form has been specified, since no character has been typed, the call to INTERRUPT will be (INTERRUPT fn args), thus the char argument to fn will be NIL. Since in this case one presumably knows what is to be done on <break>, this should not be a problem.

(SETAUTOINTERRUPT NIL)
returns the current autointerrupt class and hardflag, while
(SETAUTOINTERRUPT T)
resets to "no autointerrupt class" status.

INITIAL CHARACTER INTERRUPT CLASSES

| char | class | hardflag |
|---|---|---|
| R,r | RESET | T |
| U,u | ERROR | T |
| E,e | ERRORX | T |
| B,b | BREAK | NIL |
| H,h | BREAK | T |
| T,t | CPUTIME | T |
| ? | HELP | T |
| All others | NONE | T |

Default autointerrupt : class = NIL, hardflag = NIL.

# CHAPTER 6

## GARBAGE COLLECTION

Garbage collection is invoked automatically whenever storage is exhausted for a particular data type, but may also be invoked by the user with the function RECLAIM.

The user may exercise some control over the memory allocation during garbage collection with the function MINFS, which determines how much free space should be available to each data type when garbage collection is completed.

Garbage collection strategy is to collect before allocating a new page (thus minimizing memory utilization). Any time the free space for a type is exhausted all the fixed size objects (atoms, strings, big numbers, and lists) are collected. If the space available for a type is now at least the minfs value for that type (at least 1 if this type caused the garbage collection) then no further memory is allocated. If there is less than the minfs space, a new page will be allocated if possible. If for any reason this is not possible a full compaction will be invoked, and the allocation is retried. The pages are allocated on a "first come first served" basis to pname and string character space (over which the user has no control) then to the atoms, strings, big integers, and lastly lists. No attempt is made to share the available memory around if there is not sufficient space for everyone.

For both RECLAIM and MINFS the types are identified by the predicate functions which recognize that type:

| | |
|---|---|
| LITATOM | Atoms |
| STRINGP | string headers |
| FIXP | big (integer) numbers |
| LISTP | lists |

---- FUNCTIONS ----

(RECLAIM type)

RECLAIM invokes a garbage collection for the type specified by type, which may be any one of the type predicates, or NIL. NIL

is treated as LISTP in this situation. RECLAIM returns as its result the number of free objects for the causing type.

(MINFS type value)

MINFS sets the minimum number of objects of type type that should be available after a garbage collection to value, and returns the last minfs value for that type. If value is negative, 0 is used. If value is NIL, no change is made, and the current value is returned.

# CHAPTER 7

## EDIT

Two edit functions are implemented:

| | |
|---|---|
| (EDITF fn . edcom) | edit a function. Value = NIL. |
| (EDITS s edcom) | edit any s-expr. Value = s. |

edcom = list of edit commands (or NIL). If present, either EDIT will exit on completion.

The following commands are implemented.
Commands explained in HA 76 are accomplished by a page reference.

| | |
|---|---|
| P | Print to level 2 |
| PP | Pretty Print to level 2 |
| ? | Print to level 1000 |
| ?? | Pretty Print to level 1000 |

Note: In INTERLISP the print commands are not exactly as ours.

| | |
|---|---|
| OK | p 52 |
| UP | p 50 |
| F | p 50 |
| E s | p 117 |
| NX | p 114 |
| I or ' | p 49 |
| S x | Set x to the current expression. Used in combination with US. |
| n | p 49 |
| (n) | p 49 |
| (n e1..) | p 49 |
| (-n e1..) | p 49 |
| (N e1..) | p 49 |
| (R x y) | p 50 |
| (BI n m) | p 51 |
| (BO n) | p 51 |
| (LI n) | p 51 |
| (LO n) | p 51 |
| (RI n m) | p 51 |
| (RO n) | p 51 |
| (: e1..) | p 114 |
| (US x commands) | Use a copy of the saved value of x in commands. |
| (MARK X) | Save the current chain in x. |
| (\ x) | Reset the edit chain to x. |

NOTE : S and US can be used in different edit sessions.

Ex.: Move the PROG expression of FOO to be the PROG expression of another function FII.

```
(EDITF FOO)
F PROG S DEF OK
(EDITF FII)
(US DEF (3 DEF)) OK
```

The 3rd element (the prog expression of FII) is replaced by the one stored in DEF.

# CHAPTER 8

## MISCELLANEOUS

The function GO will search the last PROG entered, no matter how many functions down it is invoked (as in Interlisp), but will only search that one PROG.

A new function GO* is defined as a FSUBR.

```
(GO* lab1 ...)
```

searches through all current PROG's for one of the argument labels. If it is found, a jump is performed. If it is not, the arguments are returned and no other action takes place. GO also takes multiple arguments; in both cases, where a PROG contains more than one of the argument labels, the first (in list order from the beginning of the PROG form) will be found.

GO* is a way of implementing ERRORSET, ERROR!, TRYTOEVALUEATE, FAIL, etc.

In fact, ERROR! is defined as with GO* so as to unwind back to an ERRORSET or the BREAK package, whichever comes first, or RESETs if neither are found.

Ex.:
ERRORSET is defined as:

```
(DE ERRORSET (ERRORFORM ERRFLG)
             (PROG NIL
                   (RETURN (LIST (EVAL ERRORFORM)))
              ERRORSET]
```

and SYSERROR is defined as:

```
(DE SYSERROR (ERRORTYPE FN ARG FORM)
             (SETQ *LASTERRORN ERRORTYPE)
```

```
- print message if ERRORFLG = T -
- exit if error fatal -
(GO* ERRORSET)
(COND (HELPFLAG (BREAK11 FORM T NIL))
      (T (GO* higher break)(RESET]
```

When SYSERROR is called it saves the error number as the value of *LASTERRORN, then tries to jump to the label ERRORSET. If it succeeds (error occurred under errorset) a "big jump" to ERRORSET is performed and the function ERRORSET returns NIL. Otherwise: if HELPFLAG is not NIL, BREAK11 is called; if NIL, it tries to jump to a higher break and failing this, RESETs.

EXIT has been redefined such that if called with NIL as an argument and there files open which would be lost on exit, a warning is printed and RESET performed.

String Functions:

In addition to those explained in Ha 75 (page 108) a new string function is defined:

(STRALLOC n c)

The first character of the literal atom (or /sub/string) c is fetched, and a new string of length n is allocated, and filled with the character from c. If c is NIL then blanks will be used..

Other functions not reported in Ha 75 (but in Te 74) are:

| | |
|---|---|
| (ABS n) | if (MINUSP n) then (MINUS n) else n. |
| (ADDLIST a l) | if memb(a,l) then l else cons(a,l) |
| (ASSOC X L F) | searches down a list L until it finds an element Y such that (F X Y) is non-NIL. A general case function which is an extension of that in Te 74; however, F defaults to EQ which is as in Te 74. |
| (CLOCK) | time in milliseconds. |
| (DSORT l) | Destructive sorting function |
| (EVLIS l) | MAPCAR(l, 'EVAL) |
| (GCREPORT flag) | Print message when GBC (if flag = T) |
| (NTH l n) | Performs CDR n-1 tims on l) |
| (ONLINEP) | T if online, otherwise NIL. |
| (PROMPT s) | changes the prompt to the pname of s, which must be 1-30 characters in length. T resets to default, NIL returns present value. |

| | |
|---|---|
| (RPT n s) | evaluate s n times |
| (RPTQ n s) | as RPT but s is not evaluated at calling time |
| (SASSOC KEY AL) | searches for an element on an association list - a list of dotted pairs of the form (key . value). Defined as (ASSOC KEY AL 'EQUAL). |
| (SIGN n) | 0 or 1 or -1 depending on the sign of n. |
| (XCALL fn 1) | A way of calling FORTRAN routines. Returns NIL in the virgin system. Ask your system implementor if the definition has changed. |

The top-level function, LISPX, is normally "defined" as

```
(LAMBDA NIL
        (PROG (TEMP)
         LOOP (PRINT (EVAL (READ)))
              (GO LOOP)))]
```

However, if LISP is being run in batch or from an XEQ file, and echoing is desired, it may be redefined as

```
(LAMBDA NIL
        (PROG (TEMP)
         LOOP [PRINT (EVAL (PROG1
                             (SETQ TEMP (READ))
                             (PRINO (PROMPT))
                             (PRINT TEMP]
              (GO LOOP)))]
```

In general, LISP primitives may be redefined to produce any behaviour desired.

A note on the mechanics of AND and OR :

These two functions are FSUBRs which take their arguments unevaluated and evaluate them sequentially until the result is determined. The result of AND is initially T; it EVALs its arguments until it finds one whose value in NIL, or it runs out of arguments. The result is the value of the last argument. The result of OR is initially NIL, it EVALs until it finds an

argument with a non-NIL value, or runs out. The result is the value of the last argument.

# APPENDIX A

## LIST OF FUNCTIONS

| NAME(ARGS) | TYPE | CHAPT. | HA 75 | TE 74 |
|---|---|---|---|---|
| ABS(N) | L | 8 | | 13.8 |
| ADD1(N) | S | | 62 | 13.3 |
| ADDLIST(A L) | S | 8 | | |
| ADDPROP(A P V) | L | | 17 | 7.1 |
| ADVISE(FN WHEN WHERE WHAT) | L | | 131 | 19.4-5 |
| ALIST () | S | 2d | | |
| ALPHORDER(A B) | S | | 150 | 6.11 |
| AND L | FS | | 67 | 5.12 |
| APPEND(L1 L2) | S | | 55 | 6.1 |
| APPLY(FN L) | S | | 82 | 8.9 |
| APPLY*(FN . L) | L | | 82 | 8.10 |
| APPLYA(FN L AL) | S | 2d | | |
| ASSOC (X L F) | L | | 58 | 5.13 |
| ATOM(S) | S | | 12 | 5.11 |
| BREAK 'L | NL | 5 | 127 | 15.18 |
| BREAKO(FN WHEN COMS) | L | 5 | 127 | 15.16-19 |
| BREAK1('BRKEXPR 'BRKWHEN 'BRKFN 'BRKCOMS) | NL | 5 | 127 | |
| BREAK11(BRKEXPR BRKWHEN BRKFN BRKCOMS) | L | 5 | | |
| BUFSIZE (FILE) | S | 4c | | |
| CAAAR(S) | S | | 10 | 5.1 |
| CAADR(S) | S | | 10 | 5.1 |
| CAAR(S) | S | | 10 | 5.1 |
| CADAR(S) | S | | 10 | 5.1 |
| CADDR(S) | S | | 10 | 5.1 |
| CADR(S) | S | | 10 | 5.1 |
| CAR(S) | S | | 10 | 5.1 |
| CDAAR(S) | S | | 10 | 5.1 |
| CDADR(S) | S | | 10 | 5.1 |
| CDAR(S) | S | | 10 | 5.1 |
| CDDAR(S) | S | | 10 | 5.1 |
| CDDDR(S) | S | | 10 | 5.1 |

| | | | | |
|---|---|---|---|---|
| CDDR(S) | S | | 10 | 5.1 |
| CDR(S) | S | | 10 | 5.1 |
| CHARACTER(N) | S | | 93 | 10.3 |
| CHCON (S) | L | | 93 | 10.3 |
| CHCON1(A) | S | | | 10.3 |
| CHTAB(A N) | S | 4b | | |
| CLOCK () | S | 8 | 149 | 21.3 |
| CLOSE(FILE) | L | 4g | | |
| CLOSEALL (DISP) | L | 4c | 89 | 14.5 |
| CLOSEF (FILE DISP) | S | 4c | | |
| CONCAT L | S | | 108 | 10.7 |
| COND(...) | FS | | 21 | 5.4 |
| CONS(S1 S2) | S | | 10 | 5.1 |
| COPY(S) | L | | 56 | 6.4 |
| CURFILE('FILE) | NL | 4g | | |
| DE('FN 'ARGS . 'BODY) | NL | | 71 | |
| DEFINEQ 'L | NL | | 73 | 8.7 |
| DEFLIST (L PROP) | L | | 57 | 7.3 |
| DF('FN 'ARGS . 'BODY) | NL | | 71 | |
| DIFFERENCE(N1 N2) | S | | 63 | 13.7 |
| DSORT(L) | L | 8 | | |
| EDITF('FN . 'EDCOM) | NL | 7 | 113 | 9.84 |
| EDITS(S EDCOM) | L | 7 | 113 | |
| EJECT () | S | | 92 | |
| EQ(S1 S2) | S | | 11 | 5.11 |
| EQUAL(S1 S2) | S | | 11 | 5.12 |
| ERROR! () | L | | 121 | 16.12-13 |
| ERRORMESS(N) | S | 5 | | 16.13 |
| ERRORN () | L | 5 | | 16.13 |
| ERRORSET(ERRFORM ERRFLG) | L | 8 | 121 | 16.14 |
| EVAL(S) | S | | 82 | 8.9 |
| EVALA(S AL) | S | 2d | | 8.10 |
| EVLIS(L) | S | 8 | | |
| EXIT () | S | 8 | 29 | |
| FIXP (S) | S | | | 13.4 |
| FULLNAME (FILE RECOG) | S | 4c | | |
| FUNCTION('FN) | FS | | 86 | 11.1 |
| GCREPORT(FLAG) | L | 4d | | 10.15 |
| GENSYM () | S | | 108 | 10.4-5 |

| | | | | |
|---|---|---|---|---|
| GETD(FN) | S | | 73 | 8.3 |
| GETINTERRUPT (CHAR) or (CLASS HARDFLAG) | S | 5b | | |
| GETP(A P) | S | | 17 | 7.3 |
| GETREC (FILE) | S | 4c | | |
| GETWS (FILE) | L | 8 | | |
| GLC (STRING) | S | | 108 | 10.5.8 |
| GNC (STRING) | S | | 108 | 10.5.8 |
| GO L | FS | | 81 | 5.7 |
| GO* L | FS | 8 | | |
| GREATERP(N1 N2) | S | | 63 | 13.8 |
| IBEX | S | 8 | | |
| INFILE (FILE) | L | 4c | | |
| INFILEP (FILE) | L | 4c | | |
| INPUT (FILE) | L | 4c | | |
| INTERRUPT (FN ARGS CHAR) | L | | | 16.2 |
| INTERRUPTABLE (FLAG) | S | 5b | | |
| INTERRUPTABLEP () | S | 5b | | |
| LAST(L) | L | | 57 | 6.7 |
| LEFTMARGIN (N) | S | 4c | | |
| LENGTH(L) | S | | 57 | 6.8 |
| LESSP(N1 N2) | S | | 63 | 13.8 |
| LISPX () | S | | 95 | 22.47 |
| LIST L | S | | 55 | 6.1 |
| LISTP(S) | S | | 55 | 5.11 |
| LITATOM(S) | S | | 54 | 5.11 |
| LOAD(FILE) | L | | 98 | 14.27 |
| MAKEFILE(FILE FLAG) | L | | 98 | 14.45-48 |
| MAP(L FN1 FN2) | S | | 86 | 11.2 |
| MAPC(L FN1 FN2) | S | | 87 | 11.3 |
| MAPCAR(L FN1 FN2) | S | | 86 | 11.3 |
| MAPLIST(L FN1 FN2) | S | | 87 | 11.3 |
| MEMB(A L) | S | | 12 | 5.14 |
| MEMBER(A L) | S | | 55 | 5.14 |
| MINFS (TYPE N) | S | 6 | | |
| MINUS(N) | L | | 63 | 13.7 |
| MINUSP(N) | L | | 62 | 13.6 |
| MKATOM(S) | L | | 109 | 10.5 |
| MKSTRING(S) | L | | 108 | 10.4 |

| | | | | |
|---|---|---|---|---|
| NCHARS(S) | S | | 107 | 10.3 |
| NCONC(L1 L2) | S | | 104 | 6.2-3 |
| NCONC1(L A) | S | | 104 | 6.2-3 |
| NEQ(S1 S2) | S | | 55 | 5.12 |
| NEWLINE () | L | 4e | | |
| NLISTP(S) | S | | 55 | 5.11 |
| NTH(L N) | L | 8 | | 6.8 |
| NULL(S) | S | | 12 | 5.12 |
| NUMBERP(S) | S | | 61 | 5.11 |
| OBLIST () | S | 2c | | |
| ONLINEP () | S | 8 | | |
| OPEN(FILE OPT N) | L | 4g | | |
| OPENFILE (FILE ACCESS RECOG BYTESIZE | S | 4c | | |
| RECLENGTH) | S | 4c | | |
| OPENP (FILE ACCESS DISP) | S | 4c | | |
| OR L | FS | | 67 | 5.13 |
| OUTFILE (FILE) | L | 4c | | |
| OUTFILEP (FILE) | L | 4c | | |
| OUTPUT (FILE) | S | 4c | | |
| PACK(S FLAG) | S | | 107 | 10.2 |
| PLUS L | S | | 63 | 13.7 |
| PP 'L | NL | | 92 | 14.29 |
| PRETTYBREAK (FLAG) | S | 4d | | |
| PRETTYPRINT (FLAG) | S | 4d | | |
| PRIN0 (S SELECTN ESCFLAG QUOTEFLAG) | S | 4e | | |
| PRIN1 (S) | L | 4e | 91 | 14.17 |
| PRIN2 (S) | L | 4e | 91 | 14.17 |
| PRINT(S) | L | 4e | 91 | 14.17 |
| PRINTDEF(S) | L | | 92 | 14.38-39 |
| PRINTDEPTH (N) | S | 4c | | |
| PRINTESCAPES (FLAG) | S | 4d | | |
| PRINTL L | L | 4e | | |
| PRINTL-SP L | S | 4e | | |
| PRINTLENGTH (N) | S | 4c | | |
| PRINTLEVEL(CARN CDRN) | L | 4c | | 14.19 |
| PRINTPOS (N) | L | 4c | | |
| PRINTQUOTE (FLAG) | S | 4d | | |
| PROG(...) | FS | | 80 | 5.6 |
| PROGN L | FS | | 54 | 5.6 |

| | | | | |
|---|---|---|---|---|
| PROMPT (X) | S | 8 | | |
| PROMPTREAD (X) | L | 8 | | |
| PROG1 L | FS | | 54 | 5.6 |
| PUT(A P V) | S | | 17 | 7.1-2 |
| PUTD(FN S) | L | | 73 | 8.4 |
| PUTREC (FILE STRING) | S | 4c | | |
| QUOTE('S) | FS | | 22 | 5.3 |
| QUOTIENT(N1 N2) | S | | 63 | 13.7 |
| RATOM () | S | 4e | 90 | 14.11-13 |
| READ () | S | 4e | 90 | 14.10-11 |
| READC () | S | 4e | 90 | 14.14 |
| READPOS (N) | S | 4c | | |
| READVISE 'L | NL | | 131 | 19.8 |
| REBREAK 'L | NL | 5 | 127 | 15.22-23 |
| RECLAIM (TYPE) | S | 6 | 149 | 10.11 |
| REMOVE (A L) | L | | 56 | 6.4 |
| REMPROP(A P) | L | | 17 | 7.2 |
| RESET () | S | 5 | 121 | 16.13 |
| RETURN(S) | S | | 81 | 5.7 |
| REVERSE(L) | S | | 56 | 6.4 |
| REWIND(N) | S | 4c | | |
| RIGHTMARGIN (N) | S | 4c | | |
| RPLACA(S1 S2) | S | | 101 | 5.3 |
| RPLACD(S1 S2) | S | | 102 | 5.2 |
| RPLSTRING(S1 N S2) | S | | 109 | 10.5 |
| RPT(N S) | S | 8 | | 8.10-11 |
| RPTQ(N 'S) | NL | 8 | | 8.10-11 |
| SASSOC (KEY ALIST) | L | 8 | 58 | 5.13 |
| SAVEDEF(FN) | L | | 73 | 8.7-8 |
| SAVEWS (FILE S) | L | 8 | | |
| SELECTQ(...) | FS | | 53 | 5.4-5 |
| SET(A S) | S | | 25 | 5.8 |
| SETAUTOINTERRUPT (CLASS HARDFLAG) | S | 5b | | |
| SETINTERRUPT (CHAR CLASS HARDFLAG) | S | 5b | | |
| SETQ('A S) | FS | | 26 | 5.5 |
| SETQQ ('A 'S) | NL | | 54 | 5.5 |
| SIGN(N) | L | 8 | | |
| SMALLP (S) | S | | | 13.4 |
| SPACES(N) | L | 4e | 91 | 14.19 |

| | | | | |
|---|---|---|---|---|
| STRALLOC(N C) | S | 8 | | |
| STREQUAL(S1 S2) | L | | 108 | 10.4 |
| STRINGP(S) | L | | 108 | 5.9,10.4 |
| SUB1(N) | S | | 62 | 13.3 |
| SUBST(A B S) | S | | 56 | 6.5 |
| SUBSTRING(S N1 N2) | S | | 109 | 3.7,10.4,8 |
| SYSERROR(ERRTYPE FN ARG FORM) | L | 5,7 | | |
| TAILP (S1 S2) | L | | | 5.12 |
| TCONC (PTR S) | S | | 104 | 6.2 |
| TERPRI () | S | 4e | 91 | 14.19 |
| TIMES L | S | | 63 | 13.7 |
| TRACE 'L | NL | 5 | 127 | 15.18-19 |
| TYPE (S) | S | 8 | | |
| UNADVISE 'L | NL | | 131 | 19.8 |
| UNBREAK 'L | NL | 5 | 127 | 15.21 |
| UNPACK(S FLAG) | S | | 107 | 10.2-3 |
| UNSAVEDEF(FN) | L | | 73 | 8.8 |
| UNTRACE 'L | NL | 5 | | |
| VIRGINFN(FN) | L | | 129 | 15.23 |
| XCALL(FN L) | S | 8 | | |
| ZEROP(N) | S | | 62 | 13.4 |

# APPENDIX B

## DIFFERENCES BETWEEN LISPF3 AND CP-6 LISP

This appendix is designed to give a brief overview of differences between LISPF3 and CP-6 LISP, for those who have been using the former.

In the area of file management, the use of FORTRAN unit numbers has been replaced by CP-6 file id's, and some INTERLISP features added. See section 4c for details.

In the area of break key control, the break key has been used to implement (as far as possible) INTERLISP interrupts, which provide a comprehensive set of facilities for control and debugging of programs. See section 5b for details.

The memory management has been modified to do "dynamic memory management". This allows CP-6 LISP to grow to a size appropriate to the task that it is asked to perform, and shrink when memory is not required. This will normally be transparent to the user in terms of the LISP language itself.

## APPENDIX C

## REFERENCES


| | | |
|---|---|---|
| Ha 75 | A. Haraldsson: | "LISP-DETAILS. INTERLISP 360/370" DLU 75/9 |
| Mn 71 | M. Nordstrom: | "LISP F1 - A FORTRAN Implementation of Lisp 1.5". DLU 71. |
| Mn 78 | M. Nordstrom: | "LISP F3 - Implementation Guide". DLU 78/3. |
| Te 74 | W. Teitelman: | "INTERLISP REFERENCE MANUAL" XEROX corp. |