L I S P F 3

IMPLEMENTATION GUIDE AND SYSTEM DESCRIPTION.

by
Mats Nordstrom

June 1978

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

This work has been supported by the Swedish Board for Technical Development (STU) no 76-4253.

ABSTRACT.

A LISP interpreter is written in FORTRAN IV. The LISP dialect used is a subset of INTERLISP. This paper consists mainly of

- implementation guide.
- program description.
- Advices for changing and adding new FORTRAN code.

All facts of how to use the LISP F3 system are found in No78 (LISP F3 - Users guide).

TABLE OF CONTENTS.

CHAPTER 1

HOW TO IMPLEMENT LISP F3 ON YOUR COMPUTER.

The LISPF3 system consists mainly of three parts:
A. The interpreter written in FORTRAN 4.
B. 'SYS-ATOMS' A file which will be read by LISPF3 at initiation.
C. Additional functions, written in LISP.
Before compiling the interpreter, the following steps must be checked in order to fit the system into your computer.

1.
If your compiler requires it, insert a PROGRAM statement at the beginning of the FORTRAN code.

2.
Observe, that the COMMON block is separated from the rest of the FORTRAN code and replaced by something like:

```
/INC F3COMMON
```

instead. This makes it easy to change the common block, but if you do not have any source editor which performs "including" you must write a trivial preprocessor to take care of this.

3. Half word integers.

The system is delivered with some arrays declared as INTEGER*2. If you do not have this facility (or if you do not want it), change INTEGER*2 to INTEGER in the "COMMON" block and in the routines DMPIN2 and DMPOU2.

4. Non-standard FORTRAN routines.

The routines GETCH and PUTCH must be coded in assembler. They are used for moving bytes (characters) to/from an array. Calling format:

```
CALL PUTCH(VEC,CH,I)
CALL GETCH(VEC,CH,I)
```

Move from/to place I in VEC to/from CH. The character in CH is left justified (with space padding) Characters in VEC are numbered 1,2,3 ... (1 = the leftmost one).

The routine MSLEFT which returns the number of milliseconds left in the job. Replace with a dummy(return 0) if you do not have any similar time-scheduling routine.

5. Changes of arrays etc.

In order to make LISP F3 as machine independent as possible, quite a large number of constants must be preset in routine INIT1.

NAME	REMARK	CORR. ARRAY IN COMMON
NATOM= n	Nr of atoms.	PNP(n+1)
NFREET=f	f > 2*n. Nr of atoms + cons cells.	CAR(f), CDR(f)
NSTACK=s	s>500	STACK(s)
NHTAB= h	h == 1.5*n == means "about"	HTAB(h)
NPNAME=p	p == 2*n	PNAME(p+1)
BYTES= b	Nr of bytes in an INTEGER.	
MAXPN= mp	Length of the longest atom <= MARG below.	ABUFF,BUFF,RDBUFF, PRBUFF(2*mp)
NBYTES=nb	nb = 2**x where x = nr of bits in a byte (normally x = 6 for BCD, 7 for ASCII and 8 for EBCDIC).	CHTAB(nb)
MAXMES	Number of messages.	
NBMESS	Max nr of characters in a message. Change it only when you add more (and longer) messages to the system.	
	Size of IMESS: im = MAXMESS*NBMESS/BYTES	IMESS(im)
NCHTYP=c	Nr of different "character types" such as ( ) < > etc. c = size of COMMON /CHARS/.	
MAXREC=r	Used as "buffer size" for binary I/O. (The routines DMPIN/OUT, DMPIN2/OU2).	
MARG= m	Right margin in I/O buffers.	
LUNIN= li	Logical input.	
LUNUT= lu	Logical output.	
LUNSYS=ls	Logical unit for the SYS-atoms.	
CHDIV=d	Used for calculating an index from a character. If CH is an INTEGER, holding a character left justified the value of i=ABS(CH/d)+1 must be in the range (1,nb). CHDIV is only used in routines GETCHT and SETCHT. Make sure that they work! If they do not, LISPF3 will fail while reading the SYS-atoms and consequently never reach the LISP top-loop.	
MAXBIG	The largest positive integer that fits in a full word.	
MAXINT	The largest positive integer that fits in the word size used for CAR and CDR. (If you don't use half word integers, MAXINT = MAXBIG).	
MAXLUN	The largest logical unit number allowed.	
IOBUFF	Size of ABUFF BUFF RDBUFF PRBUFF.	

6.
Depending on how many bytes an INTEGER can hold you may have to change the FORMAT(..,A4) in RDA4 and WRA4 to FORMAT(...,A5) (or whatever wordsize you have).

7.
It is now time to compile and run the system. Assign your input(LISP-CODE) to logical unit li (normally teletype), and your output to lu (normally teletype). The file SYS-atoms should be assigned to logical unit ls. If all is OK so far, the system starts (after reading the SYS-atoms) with a message

```
    LISP F3, LATEST UPDATE =
    ...
```

8.
Read those LISP-packages you want to use.
To change standard input in the "scratch" system, use
   (IOTAB 1 unit)
You may choose between:

```
  BASIC1          (this one is necessary)
  BASIC2
  IO1
  FUNC1         DE,DF etc
  DEBUG1        BREAK1,error
  DEBUG2        BREAK,ADVISE, TRACE
  MAKEF         MAKEFILE
  EDIT          STRUCTURE EDITOR
```

BASIC2 and IO1 are almost necessary to do something meaningful.

9.
For convenience you may now save your system by doing
   (ROLLOUT unit)
10.
Next time you enter LISPF3 start with (ROLLIN unit).

11.
Or, if you want to, replace the subroutine INIT2 by CALL ROLLIN(unit) and you have not to care about the SYS-atoms any longer.

DEC-10 Users only: If you have got LISPF3 on a DEC-tape, this tape also contains:
- necessary MACRO10 routines (extension MAC)
- LISPF3.CMD A Commandfile for compiling etc.

To compile, do EXEC \LISPF3

IBM Users only:
Your tape also contains GETCH and PUTCH written in Assembler.

Additional hints:

- The print routine uses FORMAT(1X,150A1) for writing. If you write on a non-printer file, and want to get rid of the first blank character, change in the routine WRA1.

- The range for small integers (MAXINT - BIGNUM)/2 should not be too small. Make it > 1000.

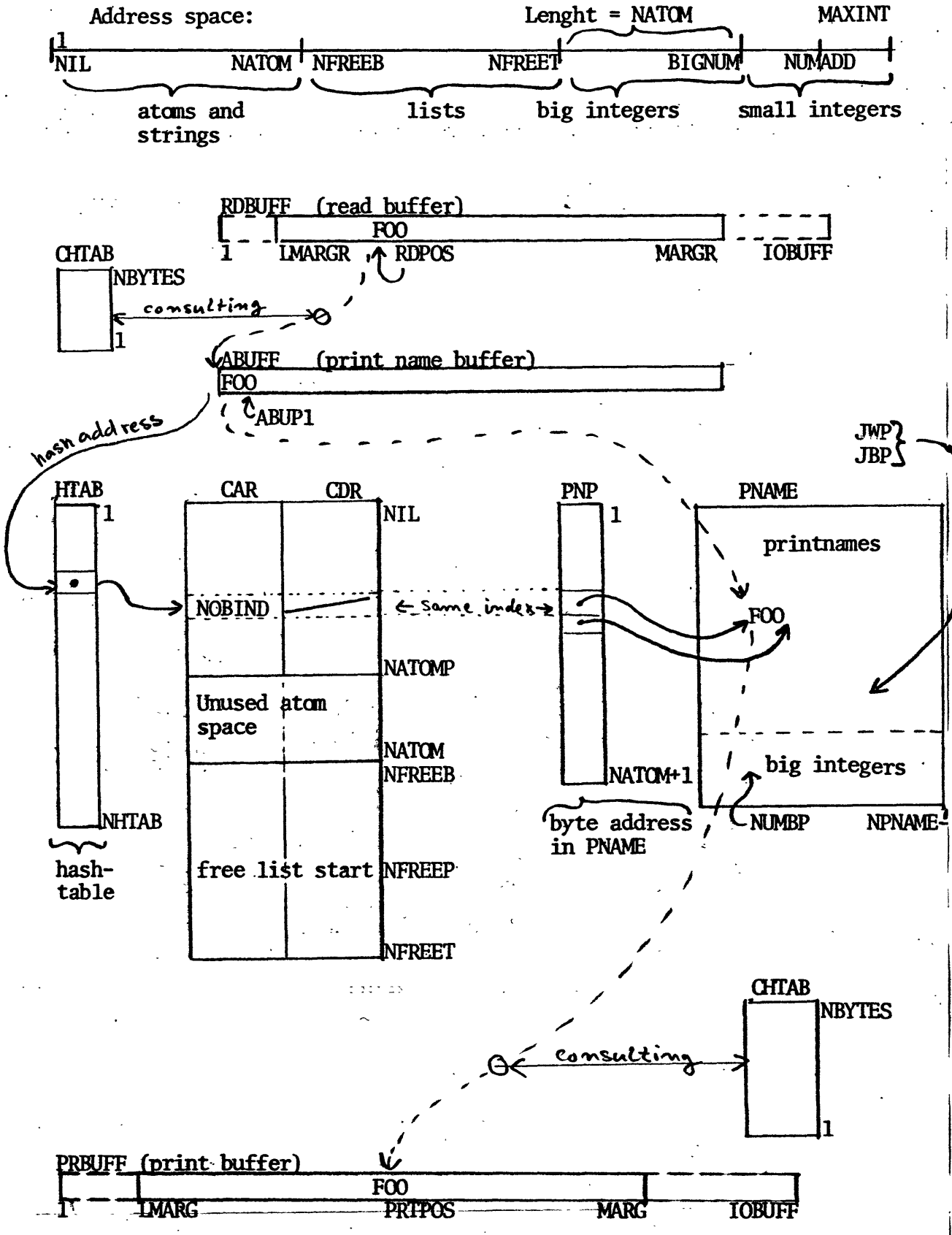
- The routine GARB may call LISPF3 "recursively" in case of error. This is done only in order to offer the user a nice error handling, but if your OS checks (and prohibits) "recursive" calls in FORTRAN you must either do it indirectly through an assembler routine or replace the call by an "error-return" by locking the stacks and placing an error code at the top of FSTACK. (See chapter 7.7!).

CHAPTER 2

INTERNAL REPRESENTATIONS.

On the next page there is a picture over the most important areas and pointers used by the system. Pointers are marked with straight arrows (------->) and the flow of characters are marked with sparse arrows (- - - - ->).

Alpha numerical atoms:
Let us follow what will happen when the atom FOO is read and printed. FOO is stored in RDBUFF by a previous FORTRAN-READ (with A1 format). RDPOS points to "the next character to be read". FOO is now inserted to ABUFF and escape characters are moved. Before being moved the character is used as an index in CHTAB to determine its type. A hash address is calculated and used as entry in HTAB. Suppose that FOO has not been read before. After finding an empty place in HTAB, this place is updated to point to NATOMP and NATOMP is increased by one (new atoms are stored consecutively). Suppose that NATOMP had the old value i. The printname FOO is now stored in PNAME and JWP,JBP are updated to point to the next empty character position in PNAME. (Also printnames are stored consecutively starting from the bottom). The starting address for the printname is stored in PNP(i) and the ending address+1 is stored in PNP(i+1). A pointer to the atom NOBIND is placed in CAR(i) and CDR(i) is set to NIL. Finally the pointer i is returned to the caller.
If later on FOO is to be printed, the print routine recognizes FOO as an atom (the pointer value i is below NATOM). The printname is fetched using PNP(i) and PNP(i+1) and moved the PRBUFF at position PRTPOS. While moving CHTAB is consulted to check if escape characters (%) are needed. When PRBUFF is filled up (PRTPOS > MARG) or if terpri() is called, the line is printed using FORTRAN-WRITE with format (1X,150A1).


Address space:
Lenght = NATOM
MAXINT
1
NIL
NATOM
NFREEB
NFREET
BIGNUM
NUMADD
atoms and strings
lists
big integers
small integers
RDBUFF (read buffer)
FOO
CHTAB
1
LMARGR
RDPOS
MARGR
IOBUFF
NBYTES
consulting
1
ABUFF (print name buffer)
FOO
ABUP1
hash address
JWP
JBP
HTAB
1
CAR
CDR
NIL
PNP
1
PNAME
printnames
NOBIND
← Same index →
FOO
NATOMP
Unused atom space
NATOM
NFREEB
NATOM+1
big integers
byte address in PNAME
NUMBP
NPNAME
NHTAB
hash-table
free list start
NFREEP
NFREET
CHTAB
NBYTES
consulting
1
PRBUFF (print buffer)
FOO
1
LMARG
PRTPOS
MARG
IOBUFF

Small integers:
The address space NFREET+NATOM - MAXINT is reserved for small integers, and they are stored in the list structures as pointers. The numerical value of a small integer is pointer-NUMADD.

Big integers:
Big integers are represented as pointers in the address space NFREET+1 - BIGNUM. The length of this range is NATOM. The integers themselves are stored consecutively as full word integers in PNAME starting from the top and using NUMBP as "free integer space pointer". When no more space for big numbers is left, a big number garbage collection is activated.

Strings:
Strings are treated the same way as literal atoms except for that they do not have an entry in HTAB and that a pointer to the atom STRING is stored in CAR(i). ("" are removed from the printname by the read routine and added if asked for by the print routine).

Substrings:
Substrings are represented as pointers in the same range as those representing atoms and strings (1,NATOM).
CAR(substring) = SUBSTR
CDR(substring) = (sourcestring start . length)
Substrings do not have a printname of it's own (and thus do not occupy space in HTAB or PNAME).

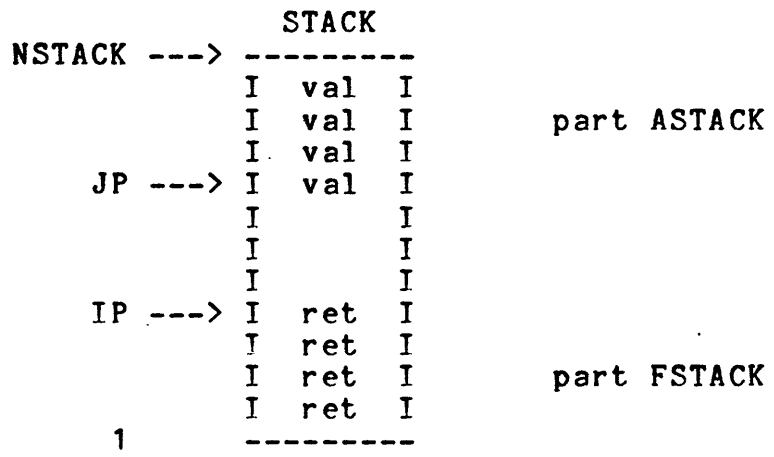
Lists:
List structures are represented as pointers in the range (NFREEB,NFREET). The variable NFREEP points to the free list. After a compacting garbage collection the free list is a consecutive list of cons cells starting from NFREEP and growing backwards down to NFREEB.

CHAPTER 3

RECURSIVE PROGRAMMING IN FORTRAN.

The LISP F3 interpreter is an almost direct translation of the definition given in appendix B. That definition is highly recursive and in this chapter we will explain how recursion has been programmed in FORTRAN.

The stacks:
There are two stacks for recursive calls of sub functions. One stack (named ASTACK) is used for saving values which are to be saved during a recursive call. The other stack (named FSTACK) is used to hold return jump indicators, here represented as integers which are used in a computed GOTO-statement. Both ASTACK and FSTACK are physically stored in the same vector STACK using IP and JP as stack-top pointers.



Pushing and popping are done by the routines

```
APUSH APUSH2 APUSH3
APOP APOP2 APOP3
FPUSH
```

and sometimes (for efficiency) in line as in

```
      998 I = STACK(IP)
          IP=IP - 1
```

Recursive calls and returns:

All recursive functions are coded in the FORTRAN subroutine LISPF3. That means that they are not subroutines themselves put just pieces of FORTRAN code.
A call is performed by saving necessary values with APUSH (or APUSH2, APUSH3) and by saving a return jump indicator with FPUSH.
After all this pushing follows an unconditional GOTO to the "function" and then follows (normally) a statement with the same statement number as indicated by IND, there the execution is to continue at return from the called function. Saved values are then fetched by a proper number of calls to APOP.
A return from a recursive function is done by GOTO 998 where the indicator saved on FSTACK is popped and used in a computed GOTO-statement leaving the program control to the caller.

N.b.
In order to give the user a chance in case of stack overflow, we have done the following:
Whenever the difference JP-IP becomes less than a preset value MIDDL, MIDDL is divided by 2 and SYSERROR is called (which in turn normally calls the break package). When MIDDL has become too small RESET is performed and MIDDL is reinitialized to its original value.

CHAPTER 4

THE EVAL-APPLY SYSTEM.

Eval-apply and all SUBR's and FSUBR's are handled by (or called from) the subroutine LISPF3 which is "the heart of the interpreter". Calling format is

```
CALL LISPF3(IND)
```

where IND = 1 means "this is the first call to LISPF3".
= 2 means "restart the interpreter"
< 0 means "call SYSERROR with error number -IND.

The top level loop is defined as

```
lispx();
error();
reset();
```

where lispx() is a SUBR defined as

```
loop: print(eval(read()));  go loop;
```

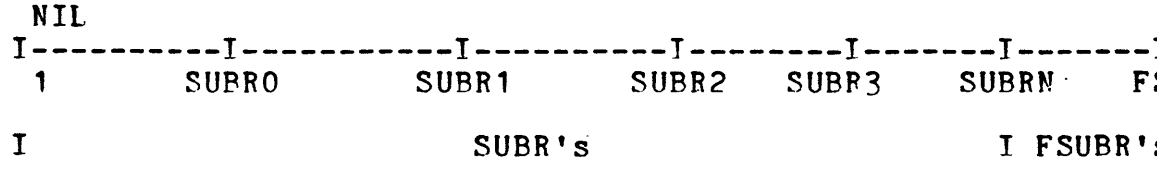
but may be redefined by the user.

Transmission of arguments:
The variables ARG ARG2 ARG3 are reserved to hold the first three arguments to SUBR's.
In case of SUBRN (see below) and FSUBR arguments are pushed onto ASTACK and the number of arguments are held in the variable IARGS.
The value of a function is assigned to IRES before returning and IRES is EQUIVALENCE:d to ARG. (Sometimes an argument just passes through).
In addition the variable FORM holds the form currently under execution.

(In the definition in appendix B, evlis is used for evaluating arguments also to a SUBR. In practice arguments to SUBR's are pushed onto ASTACK and if necessary spread to ARG ARG2 and ARG3 afterwards.)

Variable bindings:
LAMBDA, NLAMBDA and PROG variables are pushed onto an association list (the variable ALIST) in traditional manner. This list should be thought of as a simulated variable stack (and is used implicitly by eval, apply setq etc.).

Representation of SUBR's and FSUBR's etc.
The type of a function (if not LAMBDA or NLAMBDA) is indicated by the pointer value of the atom itself as seen in the following picture:



SUBR's with no arguments are numbered in the range 1-SUBR0 and so on for SUBR1, SUBR2 and SUBR3. In the range SUBR3+1 - SUBR we have SUBR's with an indefinite number of arguments (as PLUS). FSUBR's are numbered in SUBR+1 - FSUBR.
These conventions make it easy for apply to determine the proper argument actions and then jump to the corresponding code using the pointer value of the atom in a computed GOTO.
From the user's points of view apply uses the LISP function getd(f) to determine the type of f. Still as seen from the user getd(f) returns (SUBR . f) or (FSUBR . f) in the case of FORTRAN coded lisp functions.
In practice getd(f) is used inline in eval-apply and does not construct the list (SUBR . f), but as (SUBR . f) is a legal function form, cases like ((SUBR . f) .....) are also taken care of.

Error handling:
Two classes of errors may occur in the system.

1. Hard errors - reset() is called.
These errors jump to the reset point (statement nr 1) in LISPF3 (if detected inside LISPF3) or perform CALL LISPF3(2) if detected by an other subroutine.
Typical errors are stack overflow or very little space left in free memory.

2. Soft errors - syserror() is called.
All these errors jump to a place where a call to

```
syserror(errnr, fn, args, form)
```

is built up and sent to apply for further action.
SYSERROR is defined as a SUBR which just prints a message and jumps to RESET. Normally SYSERROR is redefined in lisp to make use of the break package after the message.

Removing of recursions:
Though eval-apply works as given in appendix B not all help functions are called but placed inline. Moreover it does not recursively call eval when a straight jump to eval is as good. This situation arises when the form to be evaluated is the last one in
- PROGN
- a LAMBDA body.
- a COND or SELECT clause.

Especially as (LAMBDA (..) S) is quite a common expression this little trick saves a lot of good stack storage when the recursion digs down.

CHAPTER 5

HOW TO ADD NEW SUBR'S AND FSUBR'S.

There are two ways of adding new lisp-functions in the interpreter. The simplest way is to use xcall(nr,args), the other way is to add code inline in the same fashion as other SUBRS etc.

1. How to use xcall.
The SUBR xcall is defined just to return NIL, but given to LISPF3 users as a handle where you may add calls to other FORTRAN routines. It is recommended to add your pieces of code inside the FORTRAN function XCALL(nr,args) (for example some FORTRAN call statements) and use nr as a "code selector" and args (a list of arguments) as the arguments to the specific new routines. Proper calling functions may then be defined as LISP functions using xcall.

Note:
If your routine makes use of some other SUBR you must write the code inline in LISPF3 to make use of the recursive calling conventions.

2. How to add new SUBR's etc but not using xcall.
In this case you have to do:

a) Insert the function name in the file SYSATOMS. The location of the name is important! First select the corresponding list (the first one is all SUBR's with 0 arguments, second one is all SUBR's with 1 numerical argument, third one is all SUBR's with 1 nonnumerical argument. Then comes those with 2,3, indefinite numbers and finally all FSUBR's).
Second, place the name preferably in alphabetical order in the selected list and remember the number of the location.

b) Change the corresponding GOTO-statement determined by the type of the function. Insert the statement number refering to your own piece of code so that the order in the selected list from a) still corresponds to the GOTO statement. It is also recommended to change the comment card telling which function refers to which statement number.

c) Put in your piece of code somewhere.
At entrance the arguments are held in ARG ARG2 ARG3 (or in the stack ASTACK in which case NARGS = the number of arguments).
Normal return is then done by assigning IRES to the result value and

then doing GOTO 998. (In fact ARG and IRES are equivalent). If your code is a SUBRN or FSUBR you also have to reset ASTACK before returning. The variable EJP holds the old value of JP as before pushing arguments onto the stack, so just reset JP to EJP before returning.

Warning:
If you make use of CONS (or MKNUM or MATOM) explicitly or implicitly, variables which earlier have been given lists as a value must be saved in case of a garbage collection. To be sure that new structures are recognized as active storage by the garbage collector (if they can not be reached in a normal way) the three variables TEMP1, TEMP2 and TEMP3 are given and should be used for temporary pointers to "lists under construction".

CHAPTER 6

ADVICES FOR SAVING SPACE IF YOUR COMPUTER IS A MINI.

- Do not use double buffering in I/O.

  Some operating systems gives a choice between single and double buffering as an option. Use single!

- Rewrite some routines in assembler.

  Especially FORTRAN I/O is used in a very trivial manner, and is very easy to recode in assembler. In many cases this will save a lot of program storage. (See chapter 7 where those routines which make use of I/O are explained).

- Overlaying.
  The following routines never call each other directly or indirectly and should thus be possible to overlay on most computers:
  LISPF3 vs. INIT1 INIT2. (INIT2 ought to be replaced by a call to ROLLIN anyhow).
  (GARB MARKL REHASH ROLLIN ROLLOU MOVE) vs.
  (LSPEX NCHARS EQUAL GET).
  But as the last list consists of very small routines only it is probably not worth overlaying. However this list can be extended by the routines (IPRINT PRIN1 PRINAT TERPRI) if you remove the call to IPRINT and TERPRI from the routine GARB, rename GARB to GARB1 and define a small routine GARB which calls GARB1 and then does the IPRINT call.
  See also the cross reference listing given in appendix A.

- Removing code.

  a) In GARB. Remove the parts that perform atom - bignumber and compacting gbc. Also remove the recursive (inline) code for list marking and use MARKL only.
  Remove REHASH.
  This implies that ROLLIN/ROLLOU must read/write the entire list space and the hash table HTAB.

  b) In PRIN1. Remove the parts that are active during pretty-print only.

CHAPTER 7

COMMENTS ABOUT THE FORTRAN ROUTINES USED.

The FORTRAN routines are grouped in the following manner. (See also appendix A, where a cross reference listing is given).

1. The "main" subroutine.
   LISPF3

2. Input routines.
   IREAD RATOM MATOM MKNUM SHIFT

3. Output routines.
   IPRINT PRIN1 PRINAT TERPRI

4. ROLLIN/OUT.
   ROLLIN ROLLOU MOVE

5. Garbage collection.
   GARB MARKL REHASH

6. Help routines to LISPF3
   COMPPN CONS EQUAL GET GETNUM GETPN LSPEX MESS MSLEFT NCHARS SUBST XCALL

7. Push and pop.
   APUSH APUSH2 APUSH3 APOP APOP2 APOP3 FPUSH

8. Other helproutines.
   GETCH PUTCH GETCHT SETCHT

9. Fortran I/O.
   RDA1 WRA1 RDA4 WRA4 DMPOUT DMPIN DMPOU2 DMPIN2 EJECT REW

10. Initiation routines.
    INIT1 INIT2

1.1 SUBROUTINE LISPF3(IREE)

The eval-apply system and all FORTRAN coded LISP functions. See chapter 4 for details.

2.1 FUNCTION IREAD(N)

(N is a dummy).
IREAD reads an S-expression into internal form. (See also appendix B where IREAD is defined in LISP).
It uses RATOM to read separate tokens such as atoms numbers parenthesises etc.
The value is a pointer to the constructed S-expression.

2.2 INTEGER FUNCTION RATOM(A,IOP)

Used by IREAD to read the next token from the input buffer RDBUFF. (IOP = 1).
RATOM is also called from LISPF3 (IOP = 0).
RATOM classifies the token and returns a type (and a value in the parameter A) as follows:

token	returned value	returned in A
atom	1	atom
(	2	NIL
)	3	NIL
'	4	NIL
.	5	the atom .

RATOM keeps < and > in a separate bracket stack and (if called from IREAD) < or > are never seen but returned as (or as a proper number of)'s.

2.3 FUNCTION MATOM(L)

At entry abs(L) characters are stored in ABUFF. If L > 0 MATOM creates an atom. Otherwise a string is constructed.
If an atom with the same printname already exists, that atom is returned as the value of MATOM; otherwise a new atom is created. If no more space for atoms are left a compacting atom garbage collection is performed.

2.4 FUNCTION MKNUM(N)

N is a full word integer.
Returns a small integer or big integer depending on the numerical value of N.

2.5 SUBROUTINE SHIFT(I)

Reads the next character from the input buffer RDBUFF (and reads a new line into RDBUFF if necessary). The character is returned in CHR and the type of the character in CHT. The table CHTAB is accessed with the character value (range 1 - NBYTES) to determine the type. (A list of different types is given in LISPF3 - Users guide).

If I = 1 at entry the previous CHR is stored in ABUFF (the buffer for a printname given to MATOM).

If I = 3 at entry, the previous CHR belongs to a string under construction and is stored in PNAME.

The escape character % is never returned by SHIFT but signals the next character to be treated as a letter, digit, + or - sign (depending on its type).
Normally SHIFT is called from RATOM (the flag IFLG2 = NIL) but sometimes also from the code for unpack(x) in LISPF3 (IFLG2 = T). If called from unpack, characters are read from PRBUFF instead and all characters are treated as if they were prefixed by a %.

3.1 SUBROUTINE IPRINT(I)

```
      CALL PRIN1(I)
      CALL TERPRI
      RETURN
      END
```

3.2 SUBROUTINE PRIN1(S)

This is the routine which prints a S-expression. The format of the output is directed by SYSFLAG(i) as follows:

SYSFLAG nr	value	means
2	NIL	fast printing
	T	pretty printing
3	NIL	(QUOTE s) prints as (QUOTE s)
	T	(QUOTE s) prints as 's
5	NIL	Do not add % or "
	T	Add % or " when so needed for a correct read back. (This flag is checked by PRINAT).
7		Only used during pretty-print.
	NIL	Do not begin a new line if the current expression will fit on line.
	T	Begin a new line whenever a sublist is found unless it is the first (or sometimes second) sub-expression.

A definition of PRIN1 in LISP is given in appendix B.

PRIN1 may also have been called from NCHARS in order to generate a printname. In this case TERPRI is responsible for not writing the output buffer PRBUFF on the physical output unit.

3.3 SUBROUTINE PRINAT(X,NKW,JPOLD)

Prints an atom X or ... (if X = -1) or --- (if X = -2).
NKW is the number of '-s to be printed.
JPOLD is a saved stack pointer which is used to reset the stack in case of error.
The printname is first stored in PRBUFF, then checked if it fits. (That is why the size of the print buffer is twice maximum of a printname).
If overflow has occurred TERPRI prints the line up to the old print position and then the printname for the atom is moved to the proper position in PRBUFF.
For each character sent to PRBUFF a test by the routine GETCHT is performed to check whether a % is needed or not. This check is not done if SYSFLAG(5) = NIL.

3.4 SUBROUTINE TERPRI

Writes PRBUFF on logical unit LUNUT and resets PRTPOS (the print position) to LMARG (current left margin). If called from NCHARS (the flag IFLG1 is not = NIL) output is not printed but the numbers of characters (= value of PRTPOS-1) is accumulated to IFLG1.

4.1 SUBROUTINE ROLLOU(LUN)

The routine saves a binary pattern of the LISP memory to be used later on (read by ROLLIN).
The following parts of the memory are written on logical unit LUN. (The values of im,c and nb below are explained in chapter 1.4)

CINF(1-14)	The 14 first words in COMMON /A/. Used by ROLLIN to detect if rollin is possible.
IMESS(1-im)	All messages.
AREA(1-NAREA)	COMMON /B/ up to (and including) DREG(7).
PNAME(..)	Printnames. Only used upper and lower parts.
PNP(..)	Printname pointers. Only used lower part.
CAR,CDR(..)	Only used lower(atoms) and upper(lists) parts. The free list is not written.
BCOM(1-c)	COMMON /CHARS/. (Character variables).
CHTAB(1-nb)	The character translation table.

Before writing a compacting garbage collection as called in order to save a lot of space and time by not writing the free-list.
CINF(1-6) consists of those variables which must not be changed until next ROLLIN.
CINF(7-14) are those local pointers which may be updated if the sizes of arrays have been changed.

Arrays which may have been declared as short integers are written by DMPOU2, others with DMPOUT.
ROLLOU rewinds LUN before returning.

4.2 INTEGER FUNCTION ROLLIN(LUN)

Reads a file produced by ROLLOU from logical unit LUN.
If rollin is not possible NIL is returned otherwise the value is LUN.
If the size of any array has changed since the last ROLLOU a lot of pointers must be updated. This is done by the routine MOVE.
After reading, a new free list is constructed in the empty list space and the atoms are "rehashed" using the routine REHASH in order to establish a correct hash table.
Finally read and write buffers are cleared, REWIND LUN is performed and LUN is returned as the value of ROLLIN.

SUBROUTINE MOVE(DIFF,MIN,MAX)

Used by ROLLIN to add DIFF to pointers in the range MIN < p <= MAX. Pointers p are taken from CAR,CDR and ARGS(1-10) where ARGS is equivalent to ARG ARG1 ARG2 etc. in COMMON /B/.

5.1 INTEGER FUNCTION GARB(GBCTYP)

This is the routine responsible for garbage collection. The kind of gbc is indicated by GBCTYP:

GBCTYP	action	called from
0	Normal gbc. List cells only.	CONS
1	List compacting.	ROLLOU
2	Big numbers.	MKNUM
3	Big numbers and atoms.	MATOM, MKNUM

GARB may also have been called from LISPF3 (the LISP function reclaim(gbctyp)).
Here is a short description of the working behavior of the garbage collector:

Step 1:
Mark active cells by negating CDR. If big number gbc mark in the vector PNP. If atom gbc mark also in CDR of atoms (by temporarily setting NFREEB = T+1).

Step 2:
Select proper action depending on GBCTYP.

0 goto step 7
1 goto step 3
2 goto step 5
3 goto step 5

Step 3:
List compacting (GBCTYP = 1).
Move active CAR CDR to the top of free storage and unmark CDR. Leave new address in CDR of moved cell. Goto step 6.

Step 4:
Atom gbc (GBCTYP = 3).
Move active atoms to the lower part of atom space (lower CAR,CDR). Leave new address in (negative) HTAB and unmark CDR. Goto step 6.

Step 5:
Big numbers (GBCTYP = 2,3).
Move active numbers (marked in PNP) to the top of PNAME. Leave new address in old place in PNAME. Reset PNP. .If GBCTYP = 2 then goto 6, otherwise goto step 4.

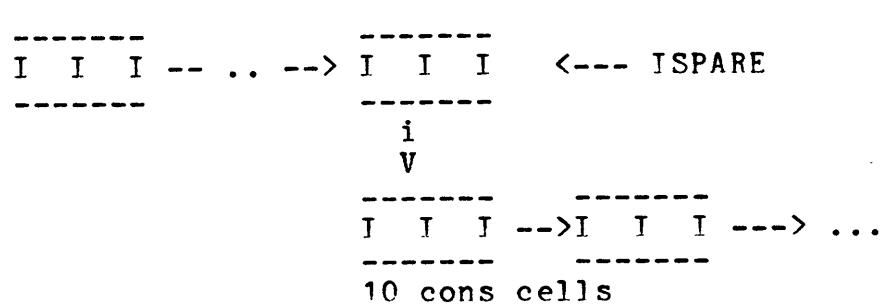
Step 6: Restore moved pointers (GBCTYP = 1,2,3).
Check all list pointers and change to new value if they have moved.

Step 7:
Clear memory.
GBCTYP = 0,1 Construct a free list and return.
2 Return.
3 Rehash all saved atoms and return.

Marking active atoms and cells starts from:
- The COMMON variables ARG ARG2 ARG3 ALIST FORM TEMP1 TEMP2 TEMP3 I1CONS I2CONS
- Current ASTACK (temporarily saved values).
- CAR and CDR of defined atoms.

Marking is performed by changing the sign of CDR. This is done inline by a local recursive piece of code. If however stack overflow occurs a non recursive routine MARKL takes over.

The free list which is built up at step 7 is equipped with a "spare tank" as follows:



If less than 10 cons cells are found no spare tank is constructed (and ISPARE = NIL), but RESET is called (CALL LISPF3(2)).
If less than 15 cons cells are found the spare tank is switched on by doing CDR(ISPARE) = CAR(ISPARE) and RESET is called.
In both cases the user still has a chance to release memory before the

space is definitely exhausted.
Actually the user has in these cases been warned before as the CONS routine checks if there is a reasonable amount of space left after a gbc. If not, SYSERROR is called which in turn normally goes into a break.

5.2 SUBROUTINE MARKL
This is a non recursive marking routine used by GARB if local stack overflow has occurred. It is somewhat slower than the corresponding inline code in GARB as it looks upon each cons cell twice. But in case of program space problems, you may use only MARKL and remove the marking code in GARB.
The algorithm is described in Sc 67.

5.3 SUBROUTINE REHASH
The routine is used for constructing a new entry in HTAB for each existing atom. REHASH is called from ROLLIN and from GARB (when atom gbc is asked for).

6.1 INTEGER FUNCTION COMPPN(X,Y)

returns:	if:
-2	x is invalid
-1	x < y
0	x = y
1	x > y
2	y is invalid

6.2 INTEGER FUNCTION CONS(I1,I2)
Does cons(I1,I2).
In case of an empty free list garbage collection is activated (GARB). If the number of free cons cells is less than ISPLFT a weak error is triggered (SYSERROR - break) and ISPLFT is divided by 2.
ISPLFT is reset to 400 whenever a RESET is performed.

6.3 INTEGER FUNCTION EQUAL(II,JJ)
Does equal(ii,jj).

6.4 INTEGER FUNCTION GET(J,I)
Does get(j,i).

6.5 INTEGER FUNCTION GETNUM(I)
Returns the FORTRAN integer value of a small or big LISP integer. The caller is responsible for that I is a LISP integer.

6.6 INTEGER FUNCTION GETPN(X,MAIN,JB,IPL)

MAIN= Pointer to main string if substring.
JB= Byte offset in printname.
IPL= Byte length of printname.
GETPN= -1 X is invalid.
0 if X literal atom.
1 if X string or substring.

6.7 SUBROUTINE LSPEX
Exits from LISP F3 after writing statistical messages. The exit is done by the FORTRAN statement STOP.

6.8 SUBROUTINE MESS(I)
If I = 0 then MESS was called from INIT2 and messages are read from LUNSYS. Otherwise a message corresponding to the nr I is printed.

6.9 MSLEFT
The time scheduling routine which is coded in assembler.
MSLEFT is only called from LISPF3, the LISP function clock(). Use your own time routine, or just skip it if you do not care about the clock function. Remember that clock shall return a dotted pair in order to avoid introducing big numbers in the result.

6.10 INTEGER FUNCTION NCHARS(S,IFLG)
This routine is used by the LISP functions concat, pack, unpack, and nchars in order to create a printname representation in PRBUFF. It is done by saving the current print status (buffer and pointer) and calling PRIN1 for each member in S (which is a list of S-expressions).
IFLG = T indicates that % is to be printed by PRIN1 when needed.
The flag IFLG1 is set to NUMADD (not = NIL) to indicate to TERPRI that lines should not be printed but print positions should be added to IFLG1 (in order to count characters).
After calling NCHARS,

- concat creates a string. TERPRI builds the printname from the print buffer.
- nchars returns the number of characters.
- pack creates an atom from the print buffer.
- unpack creates a list of atoms by reading the print buffer character by character.

Before returning they must reset the print status.

6.11 INTEGER FUNCTION SUBST(IX,IY,IS)
Does subst(ix,iy,is)

6.12 INTEGER FUNCTION XCALL(FN,L)
A dummy function (returns NIL) where you may add your own pieces of FORTRAN codes such as calling other subroutines. Use FN as a selector in a computed GOTO and L as the argument list. FN is a FORTRAN

integer at calling.

7. PUSHING and POPPING.
the following subroutines handle ASTACK, the upper part of the array STACK. JP points to the current top.

APUSH(I) APUSH2(I,J) APUSH3(I,J,K)
APOP(I) APOP2(I,J) APOP3(I,J,K)

(Pushing and popping are done from left to right in the argument lists).
The subroutine FPUSH(I) pushes a statement number indicator onto FSTACK, the lower part of STACK. Here IP points to the current top.

If stack over/underflow occurs no action is performed but a reset address is put onto FSTACK (which forces LISPF3 to jump to the reset label at next recursive return.).

Remark: In the eval-apply system pushing and popping are coded inline. At the entrance of eval there is a check whether the stack space left is reasonable. See chapter 3 for further details.

8.1 GETCH and PUTCH.
These are coded in assembler and used for moving bytes from/to an array. Calling format:

CALL PUTCH(VEC,CH,I)
Store a character CH at location I in the array VEC.
CALL GETCH(VEC,CH,I)
Fetch a character from location I in VEC to CH.

The character in CH is left justified with space padding.
The array VEC behaves as a character array numbered 1,2,3... (1 = the leftmost one).

8.2 GETCHT and SETCHT
These routines are used for fetching/storing the internal type of a character. Calling format:

I=GETCHT(IC)
CALL SETCHT(IC,IT)

A character should be placed left justified in IC before calling. The integer value of that character is used as an index in the table CHTAB. The variable CHDIV is used to calculate this index by an integer division.
CHTAB is accessed and its content is returned (GETCHT) or replaced with a new one (IT in SETCHT).
It is recommended to recode GETCHT and SETCHT in assembler.

9. FORTRAN I/O - routines.

The following routines are the only ones which use FORTRAN I/O. Only very simple format codes are used, and it is recommended to recode them in assembler, mainly for the matter of program space.
All these routines are found at the end of the LISP F3 program.

- SUBROUTINE RDA1(LUN,CARD,I1,I2,IEOF)

Reads from logical unit LUN to the array CARD(I1-I2) using A1 format.
If end of file is found, IEOF is set to 2.
Used by SHIFT and INIT2.

- SUBROUTINE WRA1(LUN,LINE,I1,I2)

Writes LINE(I1-I2) on logical unit LUN using format (1X,150A1).
Used by TERPRI.

- SUBROUTINE RDA4(LUN,CARD,I1,I2)

As RDA1 but uses format A4.
Used by MESS.

- SUBROUTINE WRA4(LUN,CARD,I1,I2)

As WRA1 but uses format (1X,100A4).
Used by MESS.

- SUBROUTINE DMPOUT(LUN,AREA,MIN,MAX)

Writes AREA(MIN-MAX) in binary format on logical unit LUN.
Used by ROLLOU.

- SUBROUTINE DMPIN(LUN,AREA,MIN,MAX)

Reads AREA(MIN-MAX) in binary format from logical unit LUN.
Used by ROLLIN.

- SUBROUTINE DMPOU2/DMPIN2(LUN,AREA,MIN,MAX)

As DMPOUT/DMPIN but AREA is declared as a half word integer array.
Used by ROLLOU/ROLLIN.

- SUBROUTINE EJECT(LUN)

Skip to next page by writing format (1H1) on LUN.

- SUBROUTINE REW(LUN)

Does REWIND LUN.

10. Initiation routines.

10.1 INIT1

Here all mashine depended variables are set. See also chapter 1.4.

10.2 INIT2

This routine reads the file SYSATOM and sets up the character table, the symbol table and the list-space memory. Also some variables corresponding to some LISP atoms are defined. It is recommended that INIT2 is replaced by a call to ROLLIN as soon as a working system is generated.

APPENDIX A

CROSS-REFERENCE LIST OF FORTRAN ROUTINES.

Routine:	Calling:				
MAIN	INIT1	INIT2/ROLLIN		LISPF3	(LSPEX)
INIT2	IREAD	RDA1	SETCHT	MESS	
LISPF3	All routines but:				
	INIT1	INIT2	MAIN	(These are not called).	
	REHASH	DMPIN	DMPOUT	(These and the following	
	DMPIN2	DMPOU2	RDA1	are only called indirectly).	
	WRA1	RDA4	WRA4		
IREAD	RATOM	CONS	FPUSH	APUSH2	APOP2
RATOM	MATOM	MKNUM	CONS	SHIFT	
MATOM	GARB	PUTCH	GETCH		
MKNUM	GARB	MESS			
SHIFT	LSPEX	MESS	GETCHT	RDA1	MATOM
IPRINT	PRIN1	TERPRI			
PRIN1	PRINAT	TERPRI	APUSH2	APUSH3	
	APOP2	APOP3			
PRINAT	GETNUM	TERPRI	GETCH	GETCHT	GETPN
TERPRI	WRA1	MATOM	PUTCH		
GARB	MARKL	MESS	IPRINT	REHASH	
	GETCH	PUTCH	TERPRI	LISPF3	
REHASH	GETCH				
CONS	GARB				
SUBST	CONS	APUSH	APOP	FPUSH	
EQUAL	GETNUM	APUSH2	APOP2	COMPPN	
NCHARS	PRIN1	MKNUM	APUSH		
MESS	RDA4	WRA4			
ROLLIN	MOVE	REHASH	REW	DMPIN	DMPIN2
ROLLOU	GARB	REW	DMPOUT	DMPOU2	
LSPEX	IPRINT	MKNUM	MESS	TERPRI	
GETPN	GETNUM				
COMPPN	GETPN	GETCH			

APPENDIX B

DEFINITION OF EVAL APPLY READ AND PRINT.

```
<DEFINEQ
<LISPF3
(LAMBDA (N)
        (PROG (EVALSW ERRFN ALIST *BACKTRACE *BACKTRACEFLG)
        (AND (EQ N 1) (PRINT "LISP F3 etc."))
        (RESET)
        (APPLY 'LISPX NIL)
        (ERROR 25 'LISPX NIL NIL>
<LISPX
(LAMBDA NIL
        (PROG NIL
         LOOP (PRINT (EVAL (READ)))
              (GO LOOP>
<EVAL
(LAMBDA (FORM)
        (COND ((OR (NUMBERP FORM) (STRINGP FORM)) FORM)
              ((ATOM FORM)
               <COND ((FORM ON ALIST?) (USE BOUND VALUE))
                     ((NEQ (CAR FORM) 'NOBIND) (CAR FORM))
                     (T (ERROR 1 'EVAL FORM FORM>)
              (T (SETQ EVALSW T)
                 (SETQ ERRFN 'EVAL)
                 (EAPPLY (CAR FORM) (CDR FORM>
<APPLY
(LAMBDA (FN ARGS)
        (SETQ EVALSW NIL)
        (SETQ ERRFN 'APPLY)
        (SETQ FORM (CONS FN ARGS))
        (AND FN (EAPPLY FN ARGS>
<EAPPLY
(LAMBDA (FN ARGS)
        (COND ((LITATOM FN)

               <COND ((GETD FN) (APPLYFN2 (GETD FN) ARGS))
                     (T (ERROR 2 ERRFN FN FORM>)
              (T (APPLYFN2 FN ARGS>
<APPLYFN1
(LAMBDA (FN ARGS)
        (COND ((SUBRP FN) (AND EVALSW (SETQ ARGS (EVLIS ARGS)))
                          (SAPPLY FN ARGS))
              ((FSUBRP FN) (SAPPLY FN ARGS))
              (T (ERROR 2 ERRFN FN FORM>
```

```
<APPLYFN2
(LAMBDA (FN ARGS)
        (COND ((NLISTP FN) (ERROR 2 ERRFN FN FORM))
              (T (SELECTQ (CAR FN)

                          (LAMBDA (AND EVALSW (SETQ ARGS (EVLIS ARGS)))
                                  (LAPPLY FN ARGS))
                          (NLAMBDA (LAPPLY FN ARGS))
                          ((SUBR FSUBR) (APPLYFN1 (CDR FN) ARGS))
                          (FUNARG (PUSH ALIST)
                                  (SETQ ALIST (CADDR FN))
                                  (SETQ RES (EAPPLY (CADR FN) ARGS))
                                  RES)
                          (ERROR 2 ERRFN FN FORM>
<LAPPLY
(LAMBDA (FN ARGS)
        (AND *BACKTRACEFLG
             (SETQ *BACKTRACE (CONS FORM *BACKTRACE)))
        (PUSH_ARGS_ON_ALIST)
        (SETQ RES (EVLAST (CDDR FN)))
        (AND *BACKTRACEFLG
             (SETQ *BACKTRACE (CDR *BACKTRACE)))
        RES>
<SAPPLY
(LAMBDA (FN ARGS)
        (JUMP_TO_CODE_FOR_FN>
<GETD
(LAMBDA (FN)
        (COND ((GETP FN 'FNCELL))
              ((SUBRP FN) (CONS 'SUBR FN))
              ((FSUBRP FN) (CONS 'FSUBR FN>
<ERROR
(LAMBDA (ERRORN FN ARGS FORM)
        (APPLY 'SYSERROR
               (LIST ERRORN FN ARGS FORM>
<SYSERROR
(LAMBDA (ERRORN FN ARGS FORM)
        (ERRORMESS ERRORN)
        (PRIN1 FN)
        (PRIN1 '-)
        (PRINT ARGS)
        (RESET>
```

```
(PRINT '"LISP F3 READ -- 7 FEB 79")
(PRINT '(VERSION 0))
<DEFINEQ
<BRSTK
<LAMBDA (X)
        (RPLACA BRSTK (PLUS X (CAR BRSTK)))
        (COND ((ZEROP (CAR BRSTK))
               (SETQ FLG NIL)
               (SETQ BRSTK (CDR BRSTK)))
              ((MINUSP (CAR BRSTK)) (SETQ OB NIL>>

<READ-L
(LAMBDA (S1 SN)
        (PROG (X)
         L    (SETQ X (READ-V))
              (AND (EQ X '%)) (GO R))
              (AND (NULL S1)
                   <SETQ S1 (SETQ SN (LIST (READ-S X>
                   (GO L))
              (COND ((NEQ X '%.)
                     (RPLACD SN (LIST (READ-S X)))
                     (SETQ SN (CDR SN))
                     (GO L)))
              (SETQ X (LIST X))
              (SETQ X (READ-L X X))
              (RPLACD SN
                     (COND ((AND (LISTP (CDR X))
                                 (NULL (CDDR X)))
                            (CADR X))
                           (T X)))
         R    (RETURN S1))>

<READ-S
(LAMBDA (X)
        (SELECTQ X
          (%( (READ-L))
          (%) (BRSTK 1) NIL)
          (%' (BRSTK 1)
              (PROG1 (LIST 'QUOTE (READ-S (READ-V)))
                     (BRSTK -1)))
          X))>

<READ-V
(LAMBDA NIL
        (OR FLG (SETQ OB (RATOM)))
        (SELECTQ OB
          (%< (SETQ BRSTK (CONS 1 BRSTK)) '%()
          (%> (SETQ FLG T) (BRSTK -1) '%))
          (%( (BRSTK 1) OB)
          (%) (BRSTK -1) OB)
          OB))>

<*READ
<LAMBDA NIL
        (PROG (FLG OB (BRSTK (LIST 0)))
              (RETURN (READ-S (READ-V>>

>
(SETQQ READFNS (BRSTK READ-L READ-S READ-V *READ))
(SETQQ READCOMS "LISP F3 READ -- 7 FEB 79")
(SETQ READGENNR 0)
STOP
```

```
(PRINT '"LISP F3 PRINT -- 7 FEB 79")
(PRINT '(VERSION 0))
<DEFINEQ
<EDITFLAG
(LAMBDA NIL (SYSFLAG 7))>

<ESCAPEFLAG
(LAMBDA NIL (SYSFLAG 5))>

<EXAMINE
(LAMBDA (S) (PROG1 (UNKWOTE S) (SETQ *NKW *COUNT)))>

<LASTDEPTH
<LAMBDA (S)
        (PROG ((DEPTH LEVEL))
         L    (COND ((OR (GREATERP DEPTH (PRINTLEVEL))
                         (NLISTP S))
                    (RETURN (DIFFERENCE DEPTH LEVEL)))
                   (T (SETQ DEPTH (ADD1 DEPTH))
                      <SETQ S (UNKWOTE (CAR (*LAST S>
                      (GO L>>

<LINEBREAK
<LAMBDA (S)
        (COND ((NULL PPBREAK) (SPACES 1))
              (<AND (LISTP S)
                    (LITATOM (CAR S))
                    (EQUAL '(FSUBR . QUOTE) (GETD (CAR S>
                (TAB (PLUS -19 *RMARG)))
              <(AND (EQ 1ST 'PROG) (GREATERP I 1))
                (TAB (PLUS (LMARG)
                           (COND ((ATOM S) -5) (0>
              ((OR *LBEFORE (LISTP S)) (TAB (LMARG)))
              ((SPACES 1>>

<LMARG
(LAMBDA (X) (IOTAB 7 X))>

<MIN
(LAMBDA (X Y) (COND ((LESSP X Y) X) (T Y)))>

<PPFLAG
(LAMBDA NIL (SYSFLAG 2))>

<PRINT-A
(LAMBDA (A) (RPT *NKW '(PRIN1 "'")) (PRINO A (ESCAPEFLAG)))>
```

```
<PRINT-L
(LAMBDA (S RP)
        (PROG ((I 0)
               (LEVEL (ADD1 LEVEL))
               (LMARG (LMARG))
               (1ST (CAR S))
               PPBREAK X)
              (RPT *NKW '(PRIN1 "'"))
              (PRIN1 (COND (RP "(")
                           ((AND (PPFLAG)
                                 (GREATERP
                                   (SETQ RP (LASTDEPTH S))
                                   *MAXPAR))
                            "<")
                           (T (SETQ RP 0) "(")))
              <COND ((PPFLAG)
                     <SETQ PPBREAK
                           (AND (LESSP LEVEL (PRINTLEVEL))
                                (OR (EDITFLAG)
                                    (GREATERP
                                     (PLUS
                                       (PRINTPOS)
                                       (NCHARS S))
                                      *RMARG>
                           (LMARG (MIN (PLUS -3 *RMARG)
                                       (ADD1 (PRINTPOS>
        L      <SETQ X
                     (COND ((QFLAG) (EXAMINE (CAR S)))
                           ((CAR S>
               (OR (ZEROP I) (LINEBREAK X))
               (PRINT-S X (AND (NULL (SETQ S (CDR S))) RP))
               (SETQ I (ADD1 I))
               <AND (PPFLAG)
                    (EQ I 1)
                    (NLISTP 1ST)
                    (LESSP (PRINTPOS)
                           (WEIGH (LMARG) *RMARG *WEIGHT 1))
                    (LMARG (ADD1 (PRINTPOS>
               (COND ((NULL S))
                     ((NLISTP S) (PRIN1 " . ") (PRINT-A S))
                     ((EQ I (PRINTLENGTH)) (PRIN1 " ---"))
                     ((GO L)))
               (SELECTQ RP
                 (0 (PRIN1 ")"))
                 (1 (PRIN1 ">"))
                 NIL)
               (LMARG LMARG)
               (RETURN S)))>

<PRINT-S
<LAMBDA (S RP)
        (AND (EQ LEVEL (PRINTLEVEL)) (SETQ S '...))
        (COND ((NLISTP S) (PRINT-A S) (SETQ *LBEFORE))
              (T (PRINT-L S
                   (SELECTQ RP ((0 NIL) RP) (SUB1 RP)))
                 (SETQ *LBEFORE T>>
```

```
<QFLAG
(LAMBDA NIL (SYSFLAG 3))>

<RMARG
(LAMBDA NIL (IOTAB 8))>

<TAB
(LAMBDA (X)
        (AND (GREATERP (PRINTPOS) (SUB1 X)) (TERPRI))
        (PRINTPOS X))>

<UNKWOTE
(LAMBDA (S)
        (PROG NIL
              (SETQ *COUNT 0)
         L    (AND (LISTP S)
                   (EQ (CAR S) 'QUOTE)
                   (LISTP (CDR S))
                   (NULL (CDDR S))
                   (SETQ *COUNT (ADD1 *COUNT))
                   (SETQ S (CADR S))
                   (GO L))
              (RETURN S)))>

<WEIGH
(LAMBDA (X1 X2 W1 W2)
        (QUOTIENT
          (PLUS (TIMES X1 W1) (TIMES X2 W2))
          (PLUS W1 W2)))>

<*LAST
<LAMBDA (L)
        (PROG ((I 0))
         L    (RETURN (COND ((NLISTP (CDR L)) L)
                            ((EQ I (PRINTLENGTH)) '(---))
                            (T (SETQ I (ADD1 I))
                               (SETQ L (CDR L))
                               (GO L>>

<*PRINT
(LAMBDA (X)
        (PROG ((*NKW 0)
               (*RMARG (RMARG))
               (LEVEL 0)
               *LBEFORE *COUNT)
              (AND (QFLAG) (SETQ X (EXAMINE X)))
              (PRINT-S X)
              (TERPRI)
              (RETURN X)))>

>
(SETQQ PRINTFNS
       (EDITFLAG ESCAPEFLAG EXAMINE LASTDEPTH LINEBREAK LMARG MIN PPFLAG
         PRINT-A PRINT-L PRINT-S QFLAG RMARG TAB UNKWOTE WEIGH *LAST
         *PRINT))
(SETQQ PRINTVARS (*MAXPAR *WEIGHT))
(SETQQ PRINTCOMS "LISP F3 PRINT -- 7 FEB 79")
(SETQ PRINTGENNR 0)
(SETQQ *MAXPAR 3)
(SETQQ *WEIGHT 8)
STOP
```

APPENDIX C

REFERENCES

Ha 75	A. Haraldsson:	"LISP-DETAILS. INTERLISP 360/370" DLU 75/9.
McC 62	J. McCarhty:	"LISP 1.5 Manual" MIT Press 1962.
No 71	M Nordstrom:	"LISP F1 - A FORTRAN Implementation of LISP 1.5", DLU71/1
No 78	M Nordstrom:	"LISP F3 - Users Guide", DLU 78/4.
Te 74	W.Teitelman:	"Interlisp Reference Manual", Xerox Corp.
Sc 67	H. Schon etc:	"An efficient machine independent procedure for ...." CACM Aug 1967.