L I S P

Programmer's Manual

MIT Artificial Intelligence Project

MODIFICATIONS

1.	cons(a,d)	3/3/59
2.	consw(w)	3/3/59
3.	copy(L)	3/3/59
4.	equal(L1,L2)	3/3/59
5.	eralis(L)	3/3/59
6.	erase(L)	3/3/59
7.	maplist(L,f)	3/3/59
8.	Open Subroutines	3/3/59
9.	search(L,p,f,u)	3/3/59
10.	apply	3/3/59

CONS (a,d)

cons (a,d) puts comb(a,d) into a register taken from free storage, and returns with the location of this register as its value. It may be written as:

cons (a,d) = consw(comb(a,d))

```
CONS    STQ T1
        ARS 18
        ADD T1
        SXD T1,4
        LXD FREE,4
        TXH *+4,4,0
        SXD FROUT,4
        TSX FROUT+1,4
        LXD FROUT,4
        LDQ 0,4
        STQ FREE
        STO 0,4
        PXD 0,4
        LXD T1,4
        TRA 1,4
         1
T1      PZE
```

Status: Checked out.

March 3, 1959
Author: J. McCarthy

Modification number 1
Makes obsolete:

CONSW (w)

consw (w) takes the first word in the free storage list, puts w in it, and returns with the location of the word as the value of consw(w). consw(w) may be called by the instruction

```
                TSX CONSW,4
                 .
                 .
                 .
      CONSW     SYN CONS + 3
```

(See cons(a,d)))

Status: Checked out.

March 3, 1959
Author: J. McCarthy

Modification number 2
Makes obsolete

COPY (L)

The list structure starting in (L) is copied into free storage and the value of copy (L) is the location of the lead word of the copied structure.

copy (L) = (L = 0→0,car(L) =-1→L,1→cons(copy(car(L)), copy(cdr(L))))

Status: copy (L) is available as a debugged SAP language subroutine.

March 3, 1959
Author: J. McCarthy

Modification number 3
Makes obsolete:

equal (L1,L2) compares the list structures starting at L1 and L2, and the result is 1 if the structures agree both as to forms and as to the identities of the objects in corresponding places.

$$\text{equal}(L1,L2) = (L1=L2 \rightarrow 1,\ \text{car}(L1) = -1 \vee \text{car}(L2) = -1 \rightarrow 0,\ 1 \rightarrow \text{equals}(\text{car}(L1),\text{car}(L2)) \wedge \text{equals}(\text{cdr}(L1),\text{cdr}(L2)))$$

Status: equal(L1,L2) is available as a debugged SAP language subroutine.

March 3, 1959
Author: K. Maling

Modification number 4
Makes obsolete:

ERALIS (L)

eralis(L) erases the list structure starting in L.

```
subroutine (eralis(L))
/ L = 0∨car(L) =-1→return
M = erase(L)
eralis (add(M))
eralis (dec(M))
\return
```

Status: Checked out.

March 3, 1959

Modification number 5

Author: J. McCarthy

Makes obsolete:

ERASE (L)

erase (L) returns the word in location L to free storage, and has as its value, the former contents of the erased word.

```
ERASE     SXD T1,4
          PDX 0,4
          CLA 0,4
          LDQ FREE
          STQ 0,4
          SXD FREE,4
          LXD T1,4
          TRA 1,4
           .
           .
T1
```

Status: Checked out.

March 3, 1959
Author: J. McCarthy

Modification number 6
Makes obsolete:

<u>maplist (L,f)</u> constructs a list in free storage whose elements are in 1-1 correspondence with the elements of the list L. The address portion of the element of the new list at J, corresponding to the element at L contains f(car(L)). The value of maplist is the address of the new list.

a) "fast" maplist

```
maplist(L,f)=/L=0→return(0)
              maplist = cons(f(L),0)
              M = maplist
           al L = cdr (L)
              cdr(M) = cons(f(L),0)
              cdr(L) = 0→return(maplist)
              M = cdr(M)
             \go(al)
```

b) "slow" maplist

```
maplist(L,f) =(L=0→0,1→cons(f(L),maplist(cdr(L),f)))
```

Status: Both maplists have been checked out. In compiling, the fast maplist is used, as it saves about 1.3 milliseconds per list element of L. (75 % saving)

March 3, 1959
Author: J. McCarthy

Modification number 7
Makes obsolete:

OPEN SUBROUTINES

1. add(w) extracts the 15 bit address of the word w.
2. car(n). The value of car(n) is the 15 bit contents of the address part of the register in location n.
3. cdr(n). The value of cdr(n) is the 15 bit contents of the decrement part of the register in location n.
4. comb(a,d) combines tow 15 bit quantities to make a 36 bit word.
5. cwr(n) is the 36 bit contents of the register n.
6. dec(n) extracts the 15 bit decrement of the word w.
7. replaca(j,k) replaces the address of k with the 15 bit word j.
8. replacd(j,k) replaces the decrement of ~~k~~ j with the 15 bit word ~~j~~. k

March 3, 1959

Modification number 8
Makes obsolete:

search(L,p,f,u) examines the list L for an element satisfying the condition p, and if it finds one, it exits with f of that element; if the search is unsuccessful, search(L,p,f,u) exits with the value of the expression u.

$$\text{search}(L,p,f,u) = (L=0 \rightarrow u, p(L) \rightarrow f(L), 1 \rightarrow \text{search}(\text{cdr}(L), p,f,u))$$

Status: Checked out.

March 3, 1959
Author J. McCarthy

Modification number 9
Makes obsolete:

The Universal function - APPLY

WRITING LISP FOR APPLY AND EVAL

APPLY and EVAL understand lists whose first elements are function names or symbols to denote certain special expressions built into EVAL. The succeeding elements are taken to be the arguments of the functions, or part of the expression.

Since we do not yet have an input program to read infixes or rearrange parenthesis we must write the above lists in restricted external notation. Thus, the following translation hold between our usual notation and the notation appropriate for read-in to apply and eval.

Usual Notation	Restricted Notation
x	x
car(x)	(car,x)
cons(x,y)	(cons,x,r)
cons(car(x),cdr(y))	(cons,(car,x),(cdr,y))

EVAL

Eval(E,A) is a function that evaluates the lisp expression E using the list of pairs A to determine the values of variables.

If E is a variable name, it searches A for the value paired with E and takes this value. If E is a function it evaluates the arguments of the function and then uses apply to evaluate the function. In addition, it recognized certain special expressions described below.

The expression (const,C) indicates that the symbol C is a constant and not to be looked up on the A list.

(sub,E) does a sublist on the result of evaluating the expression E, using the entire list A as a list of substitutions.

(cond,$(p_1,e_1),(p_2,e_2)$---(p_k,e_k)) is the conditional expression.The p_1 are evaluated successively until one is found with a value of 1. The corresponding e_1 is evaluated. If none of the p_1 are 1, error is enter

(vare,B)(variable expression) causes the evaluation of the expression paired with the object B on the A list.

(varc,C)(variable constant) causes the item paired with C on the A-list to be the value of eval.

(intv,N) causes the integer value of N to be looked up on the property list of N. At present only 0,1, and MINUS1 are allowable as N

The following are examples of statements in our usual notation and in the restricted notation necessary for eval:

Usual	Eval
x	x
cons(x,y)	(cons,x,y)
$(p_1 \ \ e_1, p_2 \ \ e_2 --- p_k \ \ e_k)$	$(cond,(p_1e_1),(p_2,e_2),---(p_k,e_k)$
L=1	(EQUAL,L,(INTV,1))

APPLY

Apply(F,L,A) is a function that evaluates a function F for the arguments given in the list L. In addition the values of previously bound variables and some function definitions are given by the list of pairs, A.

If the function F is an object, it may be basic (car,cdr,cons), in which case it is built into part of apply, it may be defined on its property list by either a 704 program or a lisp expression, or it may be paired on its A list with a lisp expression.

If the definition of F is an expression, it must be the name of a function object, or else begin with lambda or label, followed by lambda.

If F is a subexpression it may have the same form described above, or it may be an expression, that when evaluated defines the function in a manner acceptable to apply.

The list of arguments must have the same number of elements as F has arguments, or an error may result.
(Certain built-in functions: e.g. list, may have an arbitrary number of arguments.)
The first element of L is interpreted as the first argument, the second element as the second argument, etc.

The expression (label, name,(F)) when it appears as a function is treated exactly as F would be, except that the name is paired with F and put on the A list given to all lower level uses of apply and eval. This permits writing recursions within a statement. For example the definition of the "slow" maplist using label is:

```
(LABEL,MAPLIST,(LAMBDA,(L,F),(COND,((EQUAL(INTV,O),L),
   (INTV,O)),  ((INTV,1),(CONS,(F,L),(MAPLIST,(CDR,L),F))))))
```

The functions built in to Apply are (car,cdr,cons,list,) and there are also two predicates null and atom. This value of null is 1 only if its argument is the null list, O, and the value of atom is 1 only if its argument is an object.

Lisp program for single statement interpreter

```
APPLY(F,L,A)=select(car(F);
        -1,app2(F,L,A);
        lambda,eval(caddr(F),append(pair(cadr(F),L),A));
        label,apply(caddr(F),L,append(pair(cadr(F),caddr
                        (F)),A));
        apply(eval(F,A),L,A))
EVAL(E,A)=select(car(E);
        -1,search(A,λ(J,caar(J)=E),λ(J,cadar(J)),error);
        intv ,search(cadr(E),λ(J,car(J)=int),λ(J,cdadr(J)),
                        error);
        sub,sublis(A,eval(cadr(E),A));
        const,cadr(E);
        label,eval(caddr(E),append(pair(cadr(E),caddr(E)),
                        A));
        varc,search(A,λ(J,cadar(J)=cadr(E)),λ(J,cadar(J)),
                        error);
        care,search(A,λ(J,caar(J)=cadr(E)),λ(J,eval(cadar(J),
                        cdr(J)),error);
        apply(car(E),maplist(cdr(E),λ(J,eval(car(J),A))),A))
APP2(F,L,A)=select(F;car,caar(L);cdr,cdar(L);cons,cons(car(L),cadr(L));
        list,L;null,car(L)=0;atom,caar(L)=-1;
                        search(F,λ(J,car(J)=subr∨expr),
                            λ(J,(car(J)=subr→app3(F,L,
                            1→apply(cadr(J),L,A))),
        search(A,λ(J,caar(J)=F),λ(J,apply(cadar(J),L,A)),
                            error))
evcon(E,A) = (E=0→error,eval(caar(E),A)→eval(cadar(E),A),1→evcon
                            (cdr(E),A))
```

MODIFICATIONS

11.	Function Names	3/20/59	
12.	maplist(L,f)	3/20/59	(replaces no. 7)

FUNCTION NAMES

It was agreed in an Artificial Intelligence Project meeting that the following abbreviations for the elementary functions would be used.

sin	arsin	sinh	asinh
cos	arcos	cosh	acosh
tan	artan	tanh	atanh
cot	arcot	coth	acoth
sec	arsec	sech	asech
csc	arcsc	csch	acsch

$a + b - c$ is written (plus,a,b,(minus,c))

$\frac{a.b}{c}$ is written (times,a,b,(recip,c))

$\frac{a.b}{c.d}$ is written (times,a,b(recip,(times,c,d)))

$a.b.\frac{1}{c}.\frac{1}{d}$ is written (times,a,b,(recip,c),(recip,d))

u^v is written (power,u,v)

$\log_b x$ is written (log,b,x)

Note: The natural logarithm is denoted by (log,e,x)

The symbol <u>ln</u> is not used for this purpose.

March 20, 1959 Modification number 11

Author: N. Rochester Makes obsolete:

MAPLIST (L,f)

maplist (L,f) constructs a list in free storage whose elements are in 1-1 correspondence with the elements of the list L. The address portion of the element of the new list at J, corresponding to the element at L contains f(L). The value of maplist is the address of the new list.

a) "fast" maplist

```
maplist(L,f)=/L=0→return(0)
                 maplist=cons(f(L),0)
                 M=maplist
              a1 L=cdr(L)
                 cdr(M)=cons(f(L),0)
                 cdr(L)=0→return(maplist)
                 M=cdr(M)
                \go(a1)
```

b) "slow maplist"

```
maplist(L,f)=(L=0→0,1→cons(f(L),maplist(cdr(L),f)))
```

Status: Both maplists have been checked out. In compiling, the fast maplist is used, as it saves about 1.3 milliseconds per list element of L. (75% saving)

March 20, 1959 — Modification number 12

Author: J. McCarthy — Makes obsolete: Mod. no. 7

MODIFICATIONS

13.	EQ1(L1,L2)	3/27/59
14.	CP1(L)	3/27/59
15.	PRINT(L)	3/27/59
16.	FLVAL(L)	4/3/59
17.	MAKENU(L)	4/3/59
18.	NUTERN(L)	4/3/59
19.	PRDCT(L,K)	4/3/59
20.	SUM(L,K)	4/3/59

eq1(L1,L2) compares the one level lists at L1 and L2. It's value is 1 if the two lists are identical, and zero otherwise.

```
eq1(L1,L2)=(L1=L2→1,
            L1=0∨L2=0→0,
            1→cwr(car(L1))=cwr(car(L2))∧eq1(
               cdr(L1),cdr(L2)))
```

Status: Available as a debugged SAP subroutine.

March 27, 1959 Modification number 13

Author: K. Maling

cpl(L) copies the one-level list beginning at L into free storage, and returns with the location of the copied list as its value.

cpl(L)=(L=0→0,

  1→cons(consw(cwr(car(L))),cpl(cdr(L))))

Status: Available as a debugged SAP subroutine.

March 27, 1959 Modification number 14

Author: K. Maling

PRINT(L)

print(L) prints the list at L in restricted external notation, using 119 character lines. print(L) requires the subroutines prin1(L), prin2(L), terpri, MISPH2 (or UASPH2) all headed by P, and save, unsave, error unheaded.

```
print(L)=(car(L)=-1→prin1(L),
        1→(prin2(LPAR2),print(car(L)),
              (cdr(L)=0→prin2(RPAR2),
              1→(prin2(COMMA2),print(cdr(L))))))
```

prin1(L) prints the print-name on the property list.

```
SUBROUTINE (prin1(L))
          /car(L)≠-1  error
        a1 cdr(L)=0  error
           L=cdr(L)
           car(L)≠PNAME  go(a1)
           L=car(cdr(L))
       a2  prin2(cwr(car(L))
           cdr(L)=0  return
           L=cdr(L)
          \ go(a2)
```

prin2 prints up to 6 characters in one word when the characters are justified to the left, followed by the illegal character whose octal form is 77.

Status: print(L) is available as a debugged SAP program.

Author: J. McCarthy

FLVAL (L)

flval(L) finds the address of the floating point representation of the number represented by the property list L. The value of flval(L) is the address of the floating point number.

```
flval(L) = /car(L)≠-1→error
         B1 cdr(L)=0→error
            L=cdr(L)
            car(L)≠FLOAT→go(B1)
           \return(cdar(L))
```

Status: Available as a debugged SAP subroutine.

April 3, 1959 Modification number 16

Author: S. Goldberg

makenu(L) makes an numerical object of the list structure at L, and adds it to the number list. The value of makenu(L) is the address of the constructed object list.

Status: Available as a debugged SAP subroutine.

April 3, 1959

Modification number 17

Author: S. Goldberg

NUTERN(L)

nutern(L) searches the number list for a number equal to the floating point number L. If no number is found on the number list, a new property list is formed, using makenu. The value of the function is the address of a property list which represents the floating point number L.

```
nutern(L)=/val 1=L
          return(search(cdr(nulist),
                        Lambda(J,search(car(J)),
                            Lambda(J,car(J)=FLOAT),
                            Lambda(J,cdar(J)=val 1),
                            Lambda(J,0),
                        Lambda(J,car(J)),
                        Lambda(J,makenu(List(numb,
                            FLOAT,consw(cwr(val 1)))))))
```

Status: Available as a debugged SAP subroutine

April 3, 1959 Modification number 18

Author: S. Goldberg

prdct(L,K) computes the product of two floating point numbers represented on the property lists L and K. Its value is the address of an object containing the product.

Status: Available as a debugged SAP subroutine.

April 3, 1959

Modification number 19

Author: S. Goldberg

sum(L,K) computes the sum of the floating point numbers represented by the object lists L and K. Its value is the address of an object containing the sum.

Status: Available as a debugged SAP subroutine.

April 3, 1959 Modification number 20

Author: S. Goldberg

MODIFICATIONS

21.	desc[u;m]	4/7/59
22.	pick[s;f]	4/7/59
23.	mapcar(L,f)	4/9/59
24.	GREATR(J,K)	4/9/59
25.	format[n;f;v]	4/29/59
26.	SUBSTR(R,S)	4/15/59

desc[u;m] descends a list structure m going in the address or decrement direction according to the list u. Each element of the list u is either A or D.

We have

desc[u;m]=[null[u]→m;atom[m]→error;car[u]=A→desc[cdr[u];car[m]];car[u]=D→desc[cdr[u];cdr[m]];1→error]

As an example

desc[(A,A,D);(((U,V)),W)]=(V)=cdaar[(((U,V)),W)]

desc[u;m] will be used by the functions created by format. Even by itself it will operate faster when used by apply than the corresponding composition of car and cdr.

Status: SAP routine not yet checked out.

April 7, 1959 Modification number 21

Author: J. McCarthy and K. Maling Makes obsolete

pick[s;f] has as value a list each of whose elements is A or D and which gives the location of the symbol s in the structure f. The value of pick[s;f] can be used by desc to get the element of a structure in a given position.

We have

pick[s;f] = [null[f] → NO;equal[s;f] → Λ;1 → λ[[u];[equal[u;NO] → λ[[v];[equal[v;NO] → NO;1 → cons[D;v]]][pick[s;cdr[f]]];1 → cons[A;u]]][pick[s;car[f]]]]

As an example

pick[V;(((U,V)),W)] = (A,A,D,A)

pick will be used by format.

Status: LISP routine not checked out. There are no plans to write a SAP version but the version for apply will be debugged.

April 7, 1959 — Modification number 22

Authors: J. McCarthy and K. Maling — Makes obsolete mapcar is like maplist except that it does not construct a new list and it has 0 as its value. As as example of the use of mapcar, suppose one wanted to replace with CO the variables in list L.

```
mapcar(L,(var(car(L))→replaca(L,CO),1→0))
mapcar(L,f)=(L=0→0,
       f(L)→0,
       1→mapcar(cdr(L),f))
```

Status: Available as a debugged SAP routine.

April 9, 1959 Modification number 23

Author: N. Rochester

GREATR(J,K)

This is the predicate J>K it takes as arguments two 15 bit numbers and has a one bit quantity as value. It is written in SAP

```
  GREATR          TLQ *+3
                  PXD 0,0
                  TRA 1,4
                  CLA INTV1
                  TRA 1,4
INTV1             HTR ,,1
```

Status: Checked out.

Date: April 9, 1959 Modification number 24

Author: N. Rochester

format[n;f;v]

format[n;f;v] has the value n. n is an object, f is some list structure and v is a list of variables occurring in f. Its execution causes n and the variables of v to become functions which are available to APPLY. This is best explained by an example.

Consider format[SHAKESPEARE;(UNDER,GREENWOOD,TREE);(GREENWOOD,TREE)]

There are two variables involved, GREENWOOD and TREE

Then the execution of format generates three functions to which we could give arguments

```
shakespeare [SPREADING;CHESTNUT]
greenwood   [(BENEATH,SPREADING,CHESTNUT)]
tree        [(BENEATH,SPREADING,CHESTNUT)]
```

Executing these functions in turn gives

```
    (UNDER,SPREADING,CHESTNUT)
     SPREADING
and CHESTNUT            respectively
```

Thus <u>shakespeare</u> has as argument a list u which must contain as many terms as v; and substitutes in f for one occurrence of each variable in v the corresponding variable in u.

<u>greenwood</u> and <u>tree</u> have as argument a list structure g and pick out the element in g which occupies a position corresponding to their's in f.

```
format[n;f;v]=λ[[n;f;v];[λ[[s;t];t][attrib[n;sublis[[[V;v];[F;f]];[P;
     formatp[v]]];(EXPR,(LAMBDA,V,(SUBLIS,(LIST,P),(CONST,F))))]]
          formatq[n;f;v]]]]]
formatp[v]=[null[v]→Λ;T→cons[subst[car[v];X;(LIST,(CONST,X),X)];
     formatp[cdr[v]]]]
formatq [n;f;v]=[null[v]→n;T→λ[[z];[z=NO→error;T→λ[[x;y];y]
      [attrib[car[v];subst[z;R;(EXPR,((LAMBDA,(X),(DESC,R,X))))]]]
      [formatq[n;f;cdr[v]]]]][pick[car[v];f]]]
```

Status: APPLY routine not yet checked out.

Author J. McCarthy and K. Maling Modification No. 25

April 29, 1959

SUBSTR(R,S)

substr(R,S) is the proposition that the list structure S is a substructure of the list structure R.

```
substr(R,S)=EQUAL(R,S)→T
            NULL (R)→F
            ATOM (R)→F
            SUBSTR(CAR(R),S)→T
            T→SUBSTR(CDR (R),S)
```

STATUS: Available as a debugged LISP function for apply.

April 15, 1959

Author: J. Slagle

Modification number 26