LISP FOR THE M6800

BY FRITS VAN DER WATEREN
van't Hoffstraat 140
Haarlem, NL 2014 RK
The Netherlands

LISP is an interactive list processing language and very suitable for microprocessors. LISP is very easy to learn and has a very simple syntax. It is very easy to write recursive programs and functions, because the system will stack all returns and intermediate results. When you want to define, for instance, the function for N!, in LISP this would look like:

```
(DEF FAC (N)(COND ((EQ N 0) 1)
                  (T (TIMES (N (FAC (MINUS N 1)))))
          ) )
```

"BASIC minded" people would find this an ugly notation, but in fact it is very well structured.

As you see, we define the function FAC of a variable N. The function-body is a CONDitional expression, where N is tested to be EQual to 0, then the result is 1; else (T) the result is the product of N and the 'FAC' of N − 1.

As you see, we recursively call the function FAC again, where N is assigned to N − 1 and so on. As I said, the system will stack all the intermediate results. This is done on a so-called Association list, because all structures in LISP are lists. Not only data is represented as a list structure, but programs and functions are also lists. This is one of the powers of LISP. And by means of the list notation you can directly see the internal structure of a list.

This interpreter recognizes "inner list" notation only, but it is very easy to write a program, in terms of LISP, which recognizes both "inner-" and "outer lisp" notation. We could write:

```
(DEF LISP2( ) (PROG (A)
 START (TERPRI)
       (TEREAD)
       (PRINT(QUOTE :))
       (PRINT(COND((ATOM(SETQ A (READ)))(APPLY A (READ) NIL))
                  (T (EVAL A NIL)) ))
       (GO START) ))
```

When we now call: (LISP2). We are in the just defined higher level interpreter, which has ":" as a prompt (the input routine READ will still print "*", so we get :* as a prompt now) and the system will now evaluate both:

```
CAR((A B))              which gives A and
(CAR(QUOTE (A B)))      also giving A.
```

In outer lisp notation the system does the qouting for you.

For completeness I shall give the definition of DEF:

```
(PUTPROP (QUOTE DEF)
         (QUOTE (LAMBDA (L A)
          (PUTPROP (CAR L) (CONS LAMBDA  (CDR L)) EXPR )))
         FEXPR)
```

We can now define a function by:

(DEF function-name (list of parameters) (function body))

The basic LISP interpreter consists of a READ. EVALuate and PRINT cycle, and will therefore directly execute the expression, the user has typed, and print the result. When you type, for instance:

(TIMES −3 4)

The system directly responds with: −12.

It is not my intention to give a complete description of the language, because a good description is too long for just an article. But there are some good books about LISP (see References).

This implementation of LISP is a version which is called in the literature LISP 1.5. That is standard LISP, or LISP 1, with two extensions called the PROG-feature and the FUNARG-mechanism.

Features of This Implementation

- The interpreter uses 4.5K bytes, including the initial OBLIST, and runs very well in a system with 8K bytes.
- Non compacting garbage collector.
- Advanced I/O capability.
  Implemented handlers are:
  TTY handler with an ACIA.
  READER handler with a PIA (A).
  PUNCH handler with a PIA (B).
  And it is very easy to extend this by patching the device table with an address to user written handlers.
- Very free input format.
- Easy error recovery by means of BACKSPACE, DELETE and CANCEL.
- Automatic storage allocation.
- Extended error messages.
- The interpreter recognizes 'inner lisp' notation.

Brief Function Description

Elementary functions

(CAR X)	value is the first element of list X.
(CDR X)	value is the remainder of list X.
(CONS X Y)	value is the list: (X . Y)
(QUOTE X)	value is X literally
(RPLACA X Y)	replace the 'CAR-part' of X by Y; value is X.
(RPLACD X Y)	replace the 'CDR-part' of X by Y; value is X

I/O functions

(READCH DEV)	read one single character from 'DEV'; value is this character.
(READ1 DEV)	read an atom; value is this atom.
(READ DEV)	read an S-expression; value is this expression.
(TEREAD)	flush the input buffer; value is NIL.
(PRIN1 DEV)	print atom X; value is NIL.
(PRINT X DEV)	print S-expression X; value is NIL.
(TERPRI DEV)	print a CR and LF on DEV; value is NIL.
(OPEN DEV NAME)	open a file on DEV with NAME as file name.
(CLOSE DEV)	close file on DEV.

Predicates

(ATOM X)	if X is an atom, value is T else NIL.
(NUMBER X)	if X is a number, value is T else NIL.
(NULL X)	if X is NIL, value is T else NIL.
(EQ X Y)	if X is the same as Y, value is T else NIL.
(GREATER X Y)	if X is greater than Y, value is T else NIL.
(COND (A B) (X Y))	If A is not NIL then B is evaluated; else if X is not NIL then Y is evaluated; else NIL.
(LIST A B C. . . Z)	The result is a list of all its evaluated arguments. The number of arguments may be infinite.
(EVAL X Y)	Evaluate X with Y as association-list. In fact this is the LISP-interpreter itself.
(APPLY X Y Z)	Apply the argument Y to the function with Z as initial association-list.
(PROG (A B. . . .) L (statement 1) (statement 2))	With this function we are capable of writing programs in LISP. The first argument of PROG is a list of variables used inside the PROG, and the remainder is a list of labels and statements.
(GO L)	Goto label L (literally)
(RETURN X)	Return from a PROG with the value of X.
(FUNCTION X)	This is a quotation for functions.
(ALIST)	return the current systems association-list.

Arithmetic functions

(PLUS X Y)	value is X + Y
(MINUS X Y)	value is X − Y
(TIMES X Y)	value is X * Y
(QUOTIENT X Y)	value is X/Y

Some miscellaneous functions

(SETQ X Y)	X is set to the value of Y; this is also the value of this function.
(PUTPROP X Y Z)	The property-list of atom X is extended by property Y under an indicator Z. If the indicator already exists, then its property is altered; value is Y.
(GET X Y)	Get the property saved under the indicator Y from the property-list of atom X.
(SASSOC X Y)	Lookup X on the association-list Y, when found return its value; else NIL.

How to Obtain This Implementation

Send an International Money Order to:

F. v. d. Wateren
van't Hoffstraat 140
Haarlem, NL 2014 RK
The Netherlands.

Costs:

- $10. – for an object papertape in S1-FORMAT complete documentation, and a listing of the I/O package of LISP.
- $25. – for an assembler listing with symbol table and the macro generation for the OBLIST (in GPM)
- $30. – for the above mentioned two items.
- $12.50 for cassette tape, Kansas City standard, 300 baud, including description and I/O listing.
- $32.50 for cassette and complete listing.

References

1. Berkeley, Edmund C., and Daniel B. Bobrow, *The Programming Language LISP: Its Operation and Applications*, MIT Press: Cambridge, MA, 1966.
2. Friedman, D.P., *The Little LISPer*, Science Research Associates: Palo Alto, CA, 1974.
3. Siklosky, Laurent, *Let's Talk LISP*, Prentice-Hall: Englewood Cliffs, NJ, 1976.
4. van der Poel, W.L., *The Programming Languages LISP and TRAC*, T.H. Delft: The Netherlands, 1972.
5. Waite, W.M., *Implementing Software for Nonnumeric Applications*, Prentice-Hall: Englewood Cliffs, NJ, 1972.
6. Weissman, Clark, *LISP 1.5 Primer*, Dickenson: Belmont, CA, 1976.