Lisp: Basically Speaking—Part I

by Randy Beer

Interested in a language that uses objects instead of numbers? Lisp is a symbol-manipulation language that uses lists of objects.

Lisp is a programming language, usually considered to fall somewhere between machine language and higher-level languages such as Basic, Pascal, APL, or Fortran. Its syntax and data structures differ from more traditional languages. Much of today’s research in symbolic math systems, natural language interfaces, and artificial intelligence is being done in Lisp, or in a higher-level language based on Lisp.

However, writing a program in Lisp no more guarantees that it will be intelligent than having a truckload of materials guarantees that you will be able to build a house. The basic building blocks appear to be there, but more work is necessary to even begin to assemble them into programs that exhibit intelligent behavior.

Perhaps because of its association with such abstract things as artificial intelligence, a stigma of complexity has been associated with Lisp. People who have seen a Lisp program without understanding it remember only the seemingly confusing syntax and endless parentheses. These things that tend to confuse the uninitiated are what makes Lisp powerful in the hands of an experienced programmer.

Understanding Lisp

Lisp is a symbol-manipulation language. Where many languages work with numbers, Lisp works with objects such as “chair” and “block.” Relations between objects are represented as lists; hence, it is a list processor (from which Lisp gets its name). An example of a relationship between a chair and a block would be shown as: (ON BLOCK CHAIR).

These words or objects are called atoms. Numbers are also atoms. Symbolic atoms, however, cannot begin with a number, but can contain one. Thus FACT, ARG1, ONE, 12 and $-3.14159$ are all atoms; FACT, ARG1, and ONE are symbolic atoms; and 12 and $-3.14159$ are numbers. Two special atoms come predefined in every Lisp system; they are the atoms T and NIL, and can usually be thought of as logical true and false, respectively.

Lists are built out of atoms and other lists, with a left parenthesis to mark the beginning of a list and a right parenthesis to mark the end. (A B C), (A (B (C D) E) F) G, and () are all examples of lists.

The atom NIL serves a dual purpose in that it is also used to represent the empty list. NIL and () are equivalent in all respects.

Lisp works with symbolic or s-expressions composed of atoms and lists. Thus, anything that’s an atom or a list is also an s-expression. In the eyes of a Lisp interpreter, programs and data are nearly identical. This fact contributes greatly to the power of Lisp, because it allows one program to write...

Fig. 1. Typing Error Correction

$\text{(MUL 22)}$

$\text{NIL}$

$\text{MUL 2 2}$

$\text{(ADD 1 2.5 1)}$

$\text{(DIV 23)}$

$\text{POWER 2 3}$

$\text{MUL 2 2}$

$\text{DIV 22 7}$

$\text{POWER 2 3}$

$\text{MUL 2 2}$

$\text{POWER 2 3}$

$\text{MUL 2 2}$

$\text{DIV 22 7}$

The Key Box

Model I, II/16, and III
16K RAM, Cassette Basic
32K RAM, Disk Basic
64K RAM, Model II/16

176 • 80 Micro, March 1983
another program and then execute it.

**Using Basic Lisp**

It has been proven time and again that the best way to teach almost anything is to let the student get his hands dirty from the beginning. A Lisp interpreter written in Basic is included in Listing 1. Type in the program and try all of the examples given in this series and any other ideas you may have. Though it may seem contradictory to write an interpreter for such a symbolic, recursive language in Basic, it may help make Lisp available to as many people as possible.

Basic Lisp is only a subset of a full blown Lisp system, but it should prove useful in teaching basic aspects of Lisp programming. All the examples in this series will be geared toward Basic Lisp, but important differences between it and more standardized versions will be pointed out along the way.

Typing an expression to the interpreter is easy. After entering a statement like (ADD 1), there is no need to hit return. As soon as you close all the open parentheses, the expression is evaluated and answered. In this case, a 2 is returned. One important thing to remember is that atoms must be separated by a space or a carriage return, so that (ADD 11) is not at all the same as (ADD 1 1).

Since Lisp is a more highly interactive language than Basic (it actually processes some of your input as you’re typing it in) and since Basic Lisp is an interpreter written in another language, speed types must beware! Trying to type too fast will only get you into trouble. A moderate, steady pace is best. Note that this speed problem stems from the fact that Basic Lisp is written in Basic, and is not a problem inherent in Lisp itself.

When you make a typo, it is best to delete it immediately to avoid filling up the interpreter’s internal memory with mistakes. A special function is provided in Basic Lisp to make these deletions. You should immediately close the remaining open parentheses. (Backspacing will not work.) When the prompt returns (usually after an error message warning you that a mistake has been made), type (%) and the mistake is deleted. Figure 1 shows an example of the complete routine. Again, a more sophisticated Lisp system supports far easier methods of correcting mistakes.

The actual operation of a Lisp interpreter is simple. It reads and evaluates an s-expression and prints the result (also an s-expression).

An s-expression is evaluated using a few simple rules. The value of T is T, the value of NIL is NIL, and the value of any number is itself. The value of any other atom is the s-expression it is bound to (bound and unbound atoms will be explained shortly). Type in some atoms and let the interpreter evaluate them for you.

When a list (ADD 1) is evaluated, the first atom is treated as a function and the rest of the elements of the list are treated as arguments to that function. This is known as prefix nota-
$\text{SET 'BROTHERS '(RALPH JOHN)}$

$\text{(RALPH JOHN)}$

$\text{SETQ SISTERS '(SHERRY BETTY)}$

$\text{(SHERRY BETTY)}$

$\text{BROTHERS}$

$\text{(RALPH JOHN)}$

$\text{SISTERS}$

$\text{(SHERRY BETTY)}$

$\text{SISTERS}$

$\text{(SHERRY BETTY)}$

$\text{SETQ GIRLS 'SISTERS}$

$\text{SISTERS}$

$\text{GIRLS}$

$\text{(SHERRY BETTY)}$

$\text{SETQ GIRLS 'SISTERS}$

$\text{SISTERS}$

$\text{SISTERS}$

$\text{SISTERS}$

$\text{SHERRY UNBOUND ATOM}$

$\text{(SETQ ONE ON TWO THREE 3)}$

$\text{ONE}$

$\text{3}$

$\text{Fig. 3. SET and SETQ}$
fect, you’re declaring a constant. Thus, `(MUL 2 3)` is a function call resulting in a 6 and `(MUL 2 3)` is just a list of three atoms: MUL, 2, and 3. The apostrophe is actually a shorthand for the QUOTE function and `(MUL 2 3)` is represented internally as (QUOTE (MUL 2 3)). The two notations are identical in all respects, and either can be used. The single quote mark is more common because of the increased clarity.

Much like Basic variables, Lisp atoms can have values. Atoms that have been assigned a value are called bound atoms. Atoms that haven’t yet received a value are called unbound atoms. Unlike regular variables, the value of an atom can be any Lisp object: a list, a number, or even another atom.

There are no "string" atoms and "integer" atoms. A single atom can hold either value at different times. As mentioned earlier, the value of T, NIL, or any number is simply itself. The values of predefined function names like MUL and ADD are unprintable machine code and are actually pointers to the basic subroutines that perform the functions. One atom that comes predefined in Basic Lisp is the atom FREE. Its value at any time is the amount of
two things. First, they assign the value of their second argument to their first. Second, they return the value of their last arguments. An assignment is known as a side-effect, because something has been permanently changed. Almost all Basic Lisp functions return a value, but only a few have side-effects. An example of a function without a side-effect is (ADD 1 1). This function call returns a 2, but changes nothing. Sometimes a function is used for its returned value, or for its side-effects (if any), and sometimes for both. The (SETQ B 'C) in (SETQ A (SETQ B 'C)) assigns C to B and returns C.

\[ \begin{align*}
\text{CAR} & \quad ((A \ B \ (C \ D))) \\
& \quad (A \ B) \\
\text{CDR} & \quad ((A \ B \ (C \ D))) \\
& \quad ((C \ D)) \\
\text{NIL} & \quad \text{CDR} (\text{A}) \\
\text{NIL} & \quad \text{CAR} (\text{NIL}) \\
\text{NIL} & \quad \text{CDR} (\text{NIL}) \\
\text{SHERRY} & \quad \text{CAR} (\text{CDR} ((\text{RALPH} \ \text{SHERRY} \ \text{JOHN} \ \text{BETTY}))) \\
\text{CDR} & \quad \text{CAR} (\text{CDR} (\text{A} \ \text{B} \ \text{C})) \\
\end{align*} \]

**Fig. 4.** CAR and CDR

---

**COPY MODEL III, SCRIPSIT/VISICALC®**

**Only $20**

Works with all model III Visicalc/Scripsit and allows unlimited backups of master disks. Works even if you're already backed up twice. Simple, 17 line program.

And look at these other outstanding values!

**APPARATICS**

**NEWDOS/80® , 2.0**

$129.95

Convert Model I Scrypt and Visicalc to Model III, plus. Renumber program lines - move program lines - move blocks of program lines - duplicate program lines - selective variable deletion - program single stepping - memory sort multi-dimensional arrays - swap variables - read and write to model I disks - reference keywords - spoof printing - change disk speeds - disable BREAK and CLEAR - Much, much more!

**FREE** "Layman's Guide to Newdos 80" included with this purchase.

*Use VISA - Mastercard - Money Orders - Allow 2 weeks for personal checks. Kansas residents add 3.5% sales tax.*

**HAYES SMARTMODEM®**

$249.95

Talk with other computers over regular telephone lines. SMARTMODEM can auto dial, auto answer, repeat, allow for audio monitoring, programmable, is FCC approved, direct connect - full or half duplex capability, 0-300 baud, RS-232C interface, seven status LEDs.

**VERBATIM® DATALIFE DISKETTES**

5" diskettes come in a reusable plastic storage box.

**CALL TOLL FREE!**

1 - 800 - 835-0071

**IN KANSAS:**

316 - 665-3611

**ST80 III®**

$129.95

Smart terminal communications package. Allows file uploading/downloading. Emulates many different terminals. Total operator control. (Written by Lance McKlus.)

**FREE**

Catalog price list of all Sales Data Direct-Order merchandise available upon request. Super prices - Rapid delivery - Superior products.

**DASALES DATA, INC.**

526 E. 4th Hutchinson, Kansas 67501

316/665-3611

Dealer Inquiries Invited.

Phone or write Sales Data, Inc.
Listing continued

Listing continued

Listing continued

which is then assigned to A by the first SETQ. The result of the entire function call is to set both A and B to C.

The function EVAL provides an extra round of evaluation beyond the one already performed. In other words, the result of evaluating the argument is then evaluated again. Figure 5 shows an example of how the function EVAL is used.

You will find it useful to be able to take lists apart. Lisp provides two functions for doing this, CAR and CDR. These functions would probably be more understandable if they had been called FIRST and REST respectively, but you are left with historical convention. CAR returns the first element of a list: (CAR '(A B C)) would return A. CDR returns a list of all but the first element of the list except the first: (CDR '(A B C)) would return (B C). Some examples of the use of these two functions are in Fig. 5.

DELETE is a function that removes parts of a list. In Basic Lisp, DELETE takes an atom and a list as arguments and returns a copy of the list with all top-level occurrences of the atom deleted. Full-blown Lisp systems can delete any s-expression from a list, but Basic Lisp can delete only atoms.
Model II/16 Conversion

BY JESSE W. LAKER

EDIT THE FOLLOWING LINES:

10 20=1:PRINT"ABCDEFGHIJKL":GOSUB990
20 GOSUB1000:PRINT":RETURN
30 FORI=1TO5:PRINT"A":NEXTI:RETURN
40 PRINT"BLAH":RETURN
50 (RETURN)

GET INPUTS FROM USER:

100 NEXT:PRINT":RETURN
200 PRINT":RETURN
300 PRINT":RETURN
400 NEXT:PRINT":RETURN

RETURN VALUES TO USER:

100 PRINT"RETURN":RETURN
200 PRINT":RETURN
300 PRINT":RETURN
400 NEXT:PRINT":RETURN

APPEND NEW ELEMENTS TO LIST:

100 PRINT":RETURN
200 PRINT":RETURN
300 PRINT":RETURN
400 NEXT:PRINT":RETURN

If lists can be taken apart, there should also be ways to put them together. CONS, LIST, and APPEND are three Lisp functions that do just that. CONS takes a list and a new first element for the list and returns a list with the new first element added. LIST makes a list out of all of its arguments. APPEND strings the top-level contents of each list given as an argument into a single list. Figure 5 shows examples of the functions DELETE, CONS, LIST, and APPEND.

You now have a good foundation of basic skills in Lisp programming and have been introduced to most of the functions of Basic Lisp. In Part II, you will put some of these pieces together as you learn how to define your own functions.

Randy Beer, a student of computer engineering, can be reached at 301 Lex-Ontario Road, Mansfield, OH 44903.

$ (SETOQ GIRLS'SISTERS)
SISTERS

$ (SETOQ SISTERS (SHERRY BETTY))
(SHERRY BETTY)

$ GIRLS
SISTERS

$ (EVAL GIRLS)
(SHERRY BETTY)

$ (DELETE 'A '(A B C))
(B C)

$ (SETOQ A-LIST '(A (B) (C)))
(A (B) (C))

$ (DELETE 'A A-LIST)
(A B C)

$ (CONS 'A 'B C))
(A B C)

$ (LIST 'A 'B 'C 'D)
(A (B) (C) (D))

$ (APPEND 'A 'B C 'D)
((A 'B C 'D))

$ (LIST 'MUL 2 3)
(MUL 2 3)

$ (EVAL (LIST 'MUL 2 3))
6