Slip into LISP: how to choose a LISP interpreter

A. Narayanan
Department of Computer Science, University of Exeter, UK

Abstract There are currently many different LISP interpreters available on microcomputers and minicomputers, and some cost a few pounds while others a few thousand pounds. Some are good for LISP beginners, others for LISP experts. This review presents 23 questions one should ask of a LISP interpreter before one buys it, and three LISPs — Franz LISP, Golden Common LISP, and BBC LISP — are quizzed using a questionnaire.

Background
LISP is a computer language which is thought, by many researchers in AI, to be ideal for their particular research. The language was devised by John McCarthy — an authority on AI — in 1956. 'LISP' is now a well-known acronym, the letters originating from 'LISt Processing'. The elevation of LISP in the consciousness of the general computer public is nothing short of remarkable. Just 10 years ago, to mention LISP in circles of professional programmers would have caused a puzzled look or a hostile response, the latter reaction probably being expressed in comments such as: "That **** language with all those brackets!" But since the development of AI during the 1970's and 1980's, it has become increasingly common to encounter schoolchildren who have used LISP.

LISP and AI
It is not hard to find reasons why LISP should be a good computer language for AI research.

1 LISP is ideal for symbol-manipulation rather than number-crunching, and symbolic manipulation is at the heart of one of the central research topics in AI — knowledge representation and manipulation.

2 A LISP program is represented within the computer in exactly the same way as data which are to be fed to the program are represented. This separates LISP from nearly all other high-level programming languages. When a programmer constructs a Pascal program, for example, the procedure, main program statements and data remain precisely the same. But in LISP, a programmer can write LISP statements
which tell the computer how to do something, and these LISP statements can be fed as data to other statements in the same LISP program for modification (as any data can be) and then subsequently executed as normal LISP statements. The potential application of LISP in a ‘learning’ system, where the computer program may need to modify itself in the light of previous results and new input, is not hard to appreciate.

3 LISP is an interpreted language. Just as with interpreted BASIC, interpreted LISP allows a beginner quick and easy access to itself. The beginner need not plough through many chapters of a ‘Teach-Yourself LISP’ book in order to discover how to construct a program from top-to-bottom before attempting any communication with a LISP system. Instead, the beginner can start conversing in LISP immediately, even if the communication is at a fairly rudimentary level, and can be guided by the messages coming back from the system. That is, at least, the theory!

Learning LISP

But, of course, learning and using LISP can pose difficulties.

1 The super-abundance of brackets can easily deter the beginner (even if the LISP beginner is an experienced programmer in another language) from attempting to learn the language. However, beginning every LISP statement with an open parenthesis is a habit soon acquired.

2 LISP has pretensions towards being an applicative language. This means that LISP uses functions in order to tell the computer what to do. Nearly all high-level programming languages have subroutines and functions as procedures, but LISP only has functions. The essential difference between a subroutine and a function is that a function always returns a value to the user who, or function which, called it, whereas a subroutine does not. Even for a LISP beginner well-versed in FORTRAN or Pascal, the idea of writing a program in terms of functions which must return values is sometimes difficult. Dijkstra, the world-famous Dutch computer scientist and the father of structured programming, once said: “It is virtually impossible to teach good programming habits to students who have had prior exposure to BASIC: as programmers, they are mentally mutilated beyond hope of regeneration.” I sometimes startle expert Pascal users learning LISP, and who might agree whole-heartedly with the caustic sentiment expressed in the above quote, with my own version of Dijkstra’s comment: “It is virtually impossible to teach good functional programming habits to students who have had prior exposure to Pascal: as functional programmers they are mentally mutilated beyond hope of regeneration.” The point of the above version of the quote is not to point out the difference between unstructured and structured programming languages, as Dijkstra wished to do, but to highlight another fundamental difference, namely, between imperative languages such as BASIC, Pascal and FORTRAN, which achieve their effect mainly by assigning values to variables (i.e. by changing the content of memory), and applicative (functional) languages, such as LISP, which in turn achieve their effect by the use of functions and parameters.

3 The functions will be applied recursively. Recursion is one of those concepts which is widely talked about but even more widely misunderstood. I believe that,