THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

by

JANA TAYLOR GOODMAN

B. S., Kansas State University, 1973
MS Ed., University of Southern California, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

Major Professor

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

Table of Contents

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

PREFACE

The reader need not have expert knowledge of the LISP programming language in order to benefit from reading this document. However, it is assumed that the reader has at least a basic acquaintance with LISP structures, functions, and terminology.

It is particularly important that the reader be familiar with property lists; as LEDIT is especially designed to help the user edit these LISP structures.

An excellent reference the reader may use to learn about or review LISP is Let's Talk LISP , by Laurent Siklossky {1}. Anatomy of LISP, by John Allen {2}, is also highly recommended, especially for the more seasoned LISP programmer.

KEYWORDS: S-expression, atom, property list, editor, interactive, real-time, workspace, FEXPR, EXPR, LISP interpreter, pointer.

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

Part 1: Motivation for the Design and Implementation of LEDIT

One of the most attractive features of the programming language LISP is that it fosters close and frequent interaction between the user and the computer. Despite LISP's extremely dynamic programming environment, editing programs or data is a considerably less dynamic activity on many LISP systems. Most often, editing function definitions or the data base requires leaving the LISP interpreter to use a conventional text editor. This is somewhat akin to using a draft horse to pull a Ferrari!

During a typical online programming session using a LISP interpreter, the user receives feedback from the machine-- often and immediately. Normal procedure is to create small modules and test them as they are created.

In contrast, sessions employing other widely-used programming languages (especially batch-oriented systems) often result in a minimum of interaction between user and machine. Typically, the programmer codes the program, and the machine compiles it. Then the program is executed, and (eventually) the user receives the output.

Unfortunately, even in many LISP systems, the user is unable to sustain this continuous feedback relationship with the interpreter when the LISP program or database needs editing. Usually the programmer must leave the interpreter and use a system text editor. Leaving LISP is one problem; using a conventional text editor for LISP code presents another problem. Text editors are designed for

the basic structure composed of characters and lines. The basic structures of LISP are S-expressions, which have no relationship to characters and lines. So editing LISP with a text editor is somewhat artificial and inconsistent.

Since the need for editing is bound to arise, this drawback demands attention. The solution presented here is to introduce a LISP editor composed of LISP functions.

Thus, the fundamental purpose of this project was to design and implement such a real-time, workspace LISP editor. The main benefit expected was improvement (in terms of convenience and consistency) in the programming environment for the user of the LISP facilities available on the Itel ASR5 here at KSU.

1.1 DISADVANTAGES OF THE CURRENT LISP SYSTEM

Without a workspace editor, the user has two choices when she wishes to make changes in the function definitions or the data base in her current workspace:

(1) The user can retype the definition or property list with the desired changes. Certainly, this is a poor solution for the user--especially when long portions of code must be reentered.

(2) If the code has been stored in a file, another possibility for the user is to leave the LISP interpreter, use a conventional text editor to make corrections to the LISP file, recall the interpreter, and reload the altered file. In addition, once back in LISP mode, it may be necessary for the user to read the stream from a number of files and repeat some initialization to return to the