Emacs and eev, or: how to automate almost everything

Eduardo Ochs - http://angg.twu.net/

august 20, 2005

I heard it through the grapevine...

That Emacs is a text editor

- that is huge,
- complicated,
- uses a lot of memory,
- has horrible keybindings

But Emacs is the simplest *extensible editor* possible.

Another definition for Emacs

Emacs is a *Lisp environment*
(created in the 1970s)
and Lisp is a programming language
(created in the 1950s).

Eeeek, that's *old!*
Geometry, integer numbers and fractions have thousands of years.
The notion of number that we use nowadays is from 1858.[1]

What is a computer?
How do we make a machine that executes operations in sequence?

- 1822: Difference Engine (Charles Babbage; mechanic)
- 1936: Turing Machine (Alan Turing; theoretical)
- 1940+: electronic computers

[1] defined by "Dedekind cuts":
$0.999999999\ldots = 1.0000000000\ldots = 1$

What is a computer?

= Memory + processor

The processor reads a few bytes from the memory and
"executes an instruction" —
the memory (the "state" of the computer) is modified.

A part of the memory corresponds to the pixels of the screen.
Some positions of the memory correspond to the keyboard.

A computer like that is similar to a Turing machine.
It is easy to build but hard to program.

What is a program?

(We're in 1950).
How do we express a procedure as a series of steps?
How do we make a machine that executes those steps for us?

Consider: *the people that are inventing computers and programming languages in 1950 are aware of the main ideas in 2500 years of discussions.*

1950: Lisp. Is is more natural (for humans) to express algorithms in Lisp than in "machine language".

$$\text{Problem} \xrightarrow{\qquad\qquad\text{programmer}\qquad\qquad} \text{machine language}$$

$$\text{Problem} \xrightarrow{\text{programmer}} \text{program in Lisp}$$

Objects in Lisp

Objects in Lisp can be of the following kinds:

- Numbers: 0, 1, 2, -20, 3.14
- Strings: "", "AbC", " Foo bar", "(+ 1 2)"
- Symbols: a, b, nil, t, square, +, *
- Lists: (), (2 "two"),
  (* (+ 1 2) (+ 3 4)),
  (concat "abc" "def")

Trick: ALL the complex things will be implemented using lists, and lists are constructed using 'cons'es.

How a traditional Lisp environment works

```
LISP> (+ 1 2)
3
LISP> (* (+ 1 2) (+ 3 4))
21
LISP> (setq a 5)
5
LISP> a
5
LISP> (* a a)
25
```

$$\text{"(+ 1 2)"} \xmapsto{\text{read}} (+\ 1\ 2) \xmapsto{\text{eval}} 3 \xmapsto{\text{print}} \text{"3"} \qquad \text{(in the screen)}$$

How ‘eval’ works

$$\begin{array}{lccccc} (* & 3 & 7) & \longmapsto & 21 \\ (* & (+\ 1\ 2) & (+\ 3\ 4)) & \longmapsto & 21 \end{array}$$

$$\begin{array}{lcccc} (\text{setq} & \text{a} & 22) & \longmapsto & 22 \\ & \text{a} & & \longmapsto & 22 \end{array}$$

$$\begin{array}{lcl} 22 & \longmapsto & 22 \\ \text{"abc"} & \longmapsto & \text{"abc"} \\ \text{t} & \longmapsto & \text{t} \\ \text{nil} & \longmapsto & \text{nil} \\ () & \longmapsto & \text{nil} \end{array}$$

$$\begin{array}{lcll} (<\ 1\ 2) & \longmapsto & \text{t} & (\text{true}) \\ (<\ 2\ 0) & \longmapsto & \text{nil} & (\text{false}) \end{array}$$

$$(\text{if}\ (<\ 1\ 2)\ \text{"yes"}\ \text{"no"}) \quad \longmapsto \quad \text{"yes"}$$

Every thing in a Lisp system is a Lisp object

Functions (programs):

$$\begin{array}{ccc} 5 & \xmapsto{\text{square}} & 25 \\ x & \longmapsto & (* \ x \ x) \end{array}$$

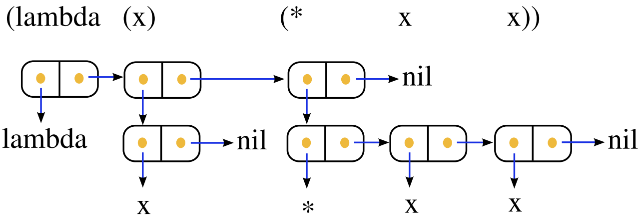
square = (lambda (x) (* x x))

(setq a 22)
(setq square 33)
(defun square (x) (* x x))

"a"	22

"square"	33
	(lambda (x) (* x x))

How lists are built using "cons"



The state of the system is determined by the values of symbols.

It is possible to explain all Lisp in its entirety (*all* details) in one hour or less — the hardest part is 'eval'.
We don't have all that time now, but we don't need all the details of how Lisp works — and it's better to start by Emacs anyway.

Extensible text editors

The kernel of Emacs is the simplest
extensible text editor possible.

Early 1970s: TECO (the first extensible editor)
Bad programming language, extensions were usually incompatible
— but people loved it anyway.
How can we make the best extensible editor possible?

Idea: take the most flexible programming language of all — Lisp
— and add functions to make it into a text editor.

Another important idea: the editor would be developed *by the community*.

Emacs = community (1)

From <http://www.gnu.org/software/emacs/emacs-paper.html> (an article by Richard Stallman written in 1979):

The programmable editor is an outstanding opportunity to learn to program! A beginner can see the effect of his simple program on the text he is editing; this feedback is fast and in an easily understood form. Educators have found display programming to be very suited for children experimenting with programming, for just this reason (see LOGO).

Programming editor commands has the additional advantage that a program need not be very large to be tangibly useful in editing. A first project can be very simple. One can thus slide very smoothly from using the editor to edit into learning to program with it.

(continues)

Emacs = community (2)

(Continuing...)

When large numbers of nontechnical workers are using a programmable editor, they will he tempted constantly to begin programming in the course of their day-to-day lives. This should contribute greatly to computer literacy, especially because many of the people thus exposed will be secretaries taught by society that they are incapable of doing mathematics, and unable to imagine for a moment that they can learn to program. But that won't stop them from learning it if they don't know that it is programming that they are learning! According to Bernard Greenberg, this is already happening with Multics EMACS.

Emacs = community (3)

How is this done?
How do people learn Lisp?

(Just some ideas; I haven't written the rest yet)
(The Emacs loop: C-x C-f /tmp/foo $\longrightarrow$ (find-file "/tmp/foo") $\xrightarrow{\text{eval}}$ ...; refresh screen)
(Windows, buffers, frames, keys, keymaps, modes)
(The Lisp interpreter is easy to invoke: C-x C-e)
(The source code is readily accesible, and quite readable)
(Extensions that are well-written enough and considered important enough are included in Emacs, and become available to everyone "by default")
(Documentation is considered extremely important: the manuals)
(Example 1: (+ 1 2))
(Example 2: source code with hyperlinks)