Introduction to ELisp

and also random Emacs stuff

j4nk_*

September 22, 2022

About Emacs

- “Eight megabytes and constantly swapping”
- “Editor MACroS”
- Family of “text editors”
  - Most famous implementation is GNU Emacs
  - Second most famous is XEmacs (“modern” fork from GNU Emacs in the 90s)
  - Beware of “Ersatz Emacs”
- Nowadays, an ELisp runtime that just happens to have text-editing functionalities

Why not Emacs?

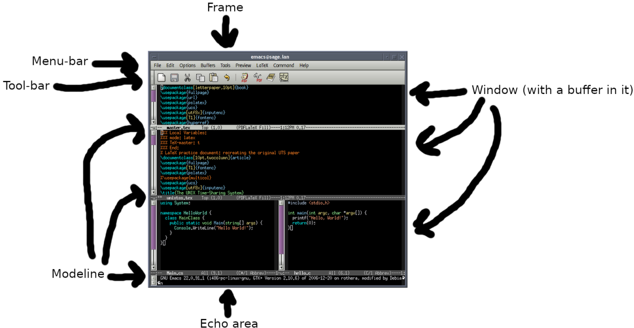
- Objection 1: Emacs is bloat
  - Not if you consider that Emacs is not a text editor, but rather an Emacs Lisp runtime
- Objection 2: Emacs doesn't follow the UNIX philosophy
  - See answer to objection 1
- Objection 3: Emacs can not be run in systems with limited resources, unlike vi/vim
  - Use TRAMP to edit over SSH/telnet/ftp/other protocols
- Objection 4: Emacs pinkie
  - Rebind capslock to ctrl, or even better use foot pedals for ctrl and alt
- Objection 5: Emacs is slow, runs in single thread
  - Emacs allows compiling scripts to a binary form which speeds up execution, groundwork for a parallelized Emacs was laid in a recent version of Emacs

Why Emacs?

- Extremely low barrier to entry
- Running composable functions inside a common ELisp runtime is theoretically faster than composing programs in a shell
- ELisp is very easy to learn and understand
- ELisp scripts downloaded elsewhere are really easy to edit, provided they aren't compiled to binary form
- Help system is very good
- Macros help speed up development immensely
- Emacs daemon allows consistent editing experience throughout a network
- Entire configuration fits in a .emacs file and .emacs.d directory

Common Emacs notation

- Keyboard combinations
  - C - ctrl
  - M - alt (meta)
  - RET - enter/return
  - e.g. M-x = alt+x
  - e.g. C-x C-s = ctrl+x, followed by ctrl+s
- GUI



Emacs as a ELisp runtime environment

- All editing components are just ELisp functions (AKA lambdas)
- Emacs is just a bunch of ELisp lambdas bound to certain key combinations
- ELisp is capable of anything, not just editing

```
(defun factorial (n)
(if (equal n 1)
1
(* n (factorial (- n 1)))))
```

Introduction to ELisp

- ELisp is an approximation of a functional programming language
  - Oriented around functions, not objects
  - A program is a composition of functions, rather than a sequence of instructions
  - Unlike pure functional programming languages, ELisp allows its functions to have side-effects
- Standard data types: bool, int, double, string, etc.
  - Only important for debugging, ELisp is weakly typed
  - nil is a special datatype, denotes end of list and causes errors when dereferenced
- One important data structure: the list

Writing ELisp code

- File extension for ELisp script is .el
- File extension for Elisp compiled code is .elc
- *scratch* buffer present at Emacs startup is automatically in ELisp mode, write temporary code there and use C-x C-e to execute code
  - Note that you can write ELisp code in any buffer, execute it, then delete it
  - Buffers in ELisp mode give you prettifying functions, syntax highlighting and best practices tips
- Put `defun`s directly in .emacs
- Put `defun`s in a .el file and put `(load file.el)` in .emacs
- Package them and use `(use-package)`

Functions in ELisp

- Calling a function: `(FX_NAME arg1 arg2 ... argN)`
- Functions are composable: `(* 5 (+ 1 2))`
- Use `defun` to declare a function with global scope
- Use `cl-labels` to declare functions with local scope (to not clutter the global scope)
- No iterative functions, only recursive
- Return value of a function is the return value of the last item in the BODY list

Types of functions

- Noninteractive functions
  - Functions that are designed to not be directly called by the user
  - Usually called by interactive functions
  - Can be called directly with M-:
- Interactive functions
  - Functions designed to be called by the user
  - Run with M-x FX-NAME RET
  - Have `(interactive)` as the first instruction in the `defun`
  - `interactive` has optional argument format string, examples present in help page for interactive (C-h f interactive RET)

Lists in ELisp

- A list is a linked list of “cons”-es
- A “cons” consists of a “car” and “cdr”
  - car - First item in a list
  - cdr - Rest of the items in a list
- Last item in a list is the cons with a cdr of nil
- The car of a list is, itself, allowed to be a list (i.e. a list of lists)
- Can treat a list as a set, run `(delete-dups list)` beforehand

Iterations in ELisp

- There are no “iterative” functions in ELisp, only recursive
- Macros exist for convenience, but they expand to recursions
  - `(dotimes (idx times)BODY...)`
  - `(dolist (item list)BODY...)`

```
(defun recursion_through_loop_function (the_list)
(if (not (car the_list))
base_case
do something with car the_list, combine with (
    recursion_through_loop_function (cdr the_list)))
```

Inserting/removing text

- `(insert TEXT)` - insert text into current buffer
- `(kill-whole-line)` - delete the current line
- `(delete-char)` - delete the character at point
- `(insert-file-contents filename)` - insert contents of filename into current buffer
- `(with-temp-buffer inst1, ..., instN)` - Spawn a temporary buffer invisible to the user, set it as the current buffer, execute inst1, ..., instN, and delete the buffer afterwards

Navigating a buffer

- `(point-{min,max})` - beginning/end of buffer
- `(line-{beginning,end}-position)` - beginning/end of line
- `(goto-char pos)` - sets point to pos
- `(forward-line n)` - go forward n lines, or backward n lines if n is negative
- `(re-search-{forward,backward} regexp end-pos)` - Updates some internal variable with position of all matches to regexp going forward/backward from point, will not search beyond end-pos; nil if no match found
  - Access matched positions with (match-beginning 0), (match-beginning 1), ...

String operations

- Note: Emacs is optimized to work on buffers, not strings
- `(split-string TEXT)` - split text into a list of strings, with whitespace delimiter, can change delimiter with additional argument
- `(string-replace text rep str)` - replace occurrences of text in str with rep
- `(replace-regexp-in-string regexp rep str)` - replace all substrings of str matching regexp with rep
- `(buffer-substring-no-properties begin_pos end_pos )` - Returns the text in between begin_pos and end_pos in the current buffer as a string, discarding properties
- `(concat str1 ...)` - Concatenates str1...strN to one string

Example 1: C header guard generator

```
(defun insert-c-header-guard()
  (interactive)
  (if buffer-file-name
          (let ((fmt-file-name (replace-regexp-in-string "\\." "_" (upcase (file-name-nondirectory (buffer
                  -file-name))))))
                (insert (concat "#ifndef " fmt-file-name "\n"))
                (insert (concat "#define " fmt-file-name "\n\n\n\n"))
                (insert (concat "#endif "))
                (forward-line -2))))
```

Example 2: Insert code for LaTeX sections

```
(defun lhw-sections (numSections)
  (interactive "nNumber of sections: ")
  (dotimes (i numSections)
    (insert (concat "\\section{Question " (number-to-
        string (+ i 1)) "}\n"))
    )
  )
```

Example 3: Convert a list to a string with a separator, except last element

```
(defun ef—list—to—string—with—separator (list string separator)
"Accumulate LIST as a STRING separated by SEPARATOR except the last element."
(if list
    (if (eq (length list) 1) ;; last element, don't include separator
        (concat string (ef—remove—delimiter—for—special—state (car list)) (ef—list—to—string—with
            —separator (cdr list) string separator))
        (concat string (ef—remove—delimiter—for—special—state (car list)) separator (ef—list—to—string
            —with—separator (cdr list) string separator))) ;; otherwise include separator
  (concat string "")))  ;; list is nil, return string concat with empty string
```

Example 4: Return a list of every other element from list

```
;; note: even elements are (ef—every—other—element—from—list list)
;; while odd elements are (ef—every—other—element—from—list (cdr list))
(defun ef—every—other—element—from—list (list)
  "Return a list consisting of even elements of LIST."
  (if list
          (cons (car list) (ef—every—other—element—from—list (nthcdr 2 list)))
        )
  )
```

Useful resources for Emacs

- FAQ - C-h C-f
- Interactive tutorial - C-h t
- Someone's guide on Emacs ricing - https://github.com/AbdeltwabMF/emacs-for-dev (my current .emacs is based heavily off of this)
- Flycheck - de facto official linter frontend
- Company mode - de facto official autocompleter
- Corfu mode - Minimal, and much more tightly integrated with Emacs competitor to Company
- VHDL mode - widely considered to be the best way to write VHDL
- HOL mode - widely considered to be the best way to write HOL
- Org mode - Emacs evangelists shill this as the ultimate note-taker

Useful resources for ELisp

- Built-in help: `C-h f fun` opens the help file for function fun
- EmacsWiki - poorly organized, but a pretty authoritative source outside of built-in help
- eintr.pdf - the main resource I used for learning ELisp, available at https://www.gnu.org/software/emacs/manual/pdf/eintr.pdf
- Xah Lee's website - well organized source for Emacs and ELisp, http://xahlee.info/emacs/
- MELPA - de facto main repository for Emacs packages
- dashlib - library of useful functions missing from main emacs. Flycheck depends on it so you should probably already have it installed
- cl-lib - Set of functions that implement CommonLisp functionality in ELisp
- Emacs stackexchange