EEN 594 P - Alternative Paradigms of Programming - Spring 2012

"Syllabus"

Part 1: Lisp

- Learn how to program in Lisp
- Do some simple programming exercises
- Do some trickier programming exercises
- See what the real point of Lisp is
- Do some interesting programming exercises
- Learn how Lisp works
- Make your own Lisp

Part 2: Prolog

- Learn how to program in Prolog
- Do some programming exercises
- See what the real point of Prolog is
- Do some interesting programming exercises
- Learn how Prolog works
- Make your own Prolog in Lisp

Part 3: Smalltalk

- Learn how to program in Smalltalk
- Program in Smalltalk
- See Object Orientation in a slightly different way
- Find out how Smalltalk works

EMACS commands

C-x means “control x”
M-x means “meta x”: just like the control key, hold down ALT while typing x.
Unlike the control key, it is case sensitive. M-X is different from M-x.
M-C-x means type x while holding both the control and alt keys.
F-n refers to one of the function keys F1, F2, ... at the top of the keyboard.

The usual keys such as delete, backspace, the arrows, etc., have their usual effects.

C-g	Cancel a command that you haven’t finished yet.
C-_	Undo
C-x u	Also undo
C-x C-f	“Find File” open file if it exists, create it if it doesn’t.
C-x C-v	Open a another file instead of the current one
C-x C-s	Save current buffer
C-x C-c	Exit
C-v	Scroll up
M-v	Scroll down
M-<	Go to beginning of file
M->	Go to end of file
C-s	Start a search (incremental) press enter when at right place
C-r	Start a search in the reverse direction
C-k	Kill line
C-@	Mark
C-w	Cut - Do “mark” at the beginning, “cut” at the end, all text between is cut out.
C-y	“Yank back”: paste previously cut text.
M-w	Copy without cutting
M-y	Replace the paste that you just did with text that was cut even earlier
C-x 2	Split window into two windows
C-x 1	Close all windows except this one.
C-x o	Go to the other window
M-C-v	Scroll *other* window down one page.
C-x C-b	List other open buffers
F-3	Start recording keyboard macro
F-4	(if you’re recording a keyboard macro) End recording
F-4	(otherwise) Replay last-recorded macro

LISP'S BASIC THINGS

Constants

Integers A bunch of decimal digits
Optionally preceded by + or −, and optionally followed by a decimal point
or `#b` followed by a bunch of binary digits, `#x` for hexadecimal, `#o` for octal
or (for example) `#7r` followed by a bunch of base-seven digits

Ratios Typed as Integer/Integer with no spaces, for example `3/7`

Reals As usual, but there must be at least one digit after the decimal point

Complex written as `#C`(Number Number), for example `#C(3 2.5)`

String Just about anything inside double quotes. `\` serves almost its normal purpose.

Characters Examples: `#\a  #\A  #\*  #\X  #\space`

Specials:
`T` The Boolean value "true"
`NIL` The Boolean value "false" and the NULL pointer value

If you type in a constant, lisp will type it back (maybe tidied up).

Symbols

Are a cross between variables and variable names.
Symbols can have values and can be used as variables, but they don't have to.
A symbol can just be what it is. A symbol is also a value in itself.

A symbol can consist of just about any sequence of characters that is not already defined to mean something else. These are all perfectly valid symbols and variable names:

```
hello   x1    1x   123y45     cat      tom+jerry
*       <pa>  -    b^2-4*a*c  $29.99   2+2=5
```

You can even do crazy things like having spaces inside symbols if you use the `\` character like you would in a string.

If you type in a symbol, lisp will assume you are using it as a variable and type its value. If you haven't given it a value, that's an error.

Cons Cells

Cons Cells almost mere linked-list links. They are just a little bit more general.
A Cons Cell is an object that contains two values and gives you easy access to both of them.
The values in a cons cell may be anything, including other cons cells.

The simplest way to write a cons cell is to write the two values separated by a dot and surrounded by round brackets (parentheses). Often the dot needs to have spaces around it because dots can be parts of symbol names.

The following are valid cons cell representations:

```
(12 . 34)
(T . "waa")
(1 . (2 . 3))
((1 . 2) . 3)
((1 . 2) . (3 . 4))
```

But all of them would cause an error if you typed them. When you type a cons cell, lisp expects the first value in it to represent a function, and it will try to apply that function in a special way to the second value.

The most common thing made out of cons cells is a linked list. In lisp, a true linked list is one of two things. A linked list is either

the value `NIL`, which represents the empty list, and can also be written as `()`

or

a cons cell whose first value can be anything, but whose second value is a proper linked list.

Because linked lists are so important to lisp, there is a special convenient notation for them. Just put any bunch of values inside round brackets.

```
(1 2 3 4 5)
```

is exactly equivalent to

```
(1 . (2 . (3 . (4 . (5 . NIL)))))
```

if you enter something in the latter form, it will be printed in the former.

By extension, this

```
((1 2 3) (4 5 6))
```

is equivalent to

```
((1 . (2 . (3 . NIL))) . (4 . (5 . (6 . NIL))))
```

Of course, you are not restricted to using only proper linked lists. You can use cons cells for anything, perhaps a binary tree:

```
((((1 . 2) . (3 . 4)) . (5 . 6)) . (((7 . 8) . (9 . 10)) . 11))
```