Misc lisp: (i) regular expressions

- basic pattern matching for regular expressions provided

  `(si::string-match pattern str)`
- returns position of first match (-1 if no match)
- ^ and $ match string start/end, . matches any char
- ( ) to enclose a pattern, + * and ? to repeat patterns
- [ ] to specify any one of a set of chars, ^ to negate
- \ is the escape for special chars, e.g. \]
- in a normal string “\\” means the char \, so string-match needs a pair of those to represent pattern \, i.e. needs string “\\\\”
- `(si::re-quote-string str)` inserts the extra \’s for you, e.g.
- `(si::re-quote-string “a slash \\ string”)` inserts two more \\’s

packages

- Similar to the idea of a namespace in C++
- When you bind/use a symbol it uses current package by default, can refer to one in a different package using `pname::varname`
- Can specify you want to be able to use all the names from another package `(use package pname)`
- Current package name is in variable `si::*package*`
- Can switch packages using `(in-package pname)`
- Can get list of all packages `(list-all-packages)`
- Create new package `(make-package ‘pname :use ‘(common-lisp))`

timing/sleep

- Current time/date `(get-universal-time)`
- Internal clock time `(get-internal-run-time)`
- See how long something takes to execute
  `(time (whatever))`
- Pause a program for N seconds `(sleep N)`

Random number generator

- Seed random number generator first

```
(setf *random-state* (make-random-state t))
```

- For a random integer in range 0..N-1

```
(random N) ; N must be positive int
```

- For a random float between 0 and N

```
(random N) ; N  must be positive float (not an integer)
```

Compiling lisp files

- Can compile a lisp file into an executable (large)
- Do not include the #! line in your file
- Have a main function where execution will begin and identify the name of that function using

```
(defun si::top-level () (main))
In the interpretter, run the following
(compile-file “filename.cl”)  ; creates filename.o
(load “filename.o”)
(si::save-system “exename”) ; saves exe with given name
```

Compiling lisp functions

- Can compile an individual function

```
(compile ‘funcname)
```

- Can compile and run lambda functions

```
(defvar f (compile nil ‘(lambda ......)))
(funcall f whatever)
```

- Can load other compiled lisp files and call their functions

```
(load “fname.o”)
```

- see asm for compiled functions

```
(disassemble ‘fname)
```

catch/throw

- some exception handling through throw and catch
- define a catch block, can throw exceptions from inside
- throw exits a specified block with a chosen return value

```
(setf myBlockResult (catch ‘myblock
    ....do regular stuff ...
    (if (somecondition) (throw ‘myblock value))
    ... do more regular stuff ... ))
```

- `myBlockResult` now holds either the normal result of the block or the value that was thrown

gotos (tagbody/go)

- Lisp does actually support a form of goto, allowing you to jump to any label within a tagbody block, e.g.

```
(tagbody
    .... do regular stuff ...
   MyLabel
    ... more regular stuff ...
    (if (somecondition) go MyLabel))
    ... and more regular stuff ...)
```

Recording lisp session (dribble)

- Can start/stop recording a lisp session using the dribble function, putting the recorded i/o in a file

```
(dribble filename)
.. all std i/o gets recorded ..
... end the session with
(dribble)
```