```lisp
(defpackage :runr (:use :cl))
(in-package :runr)

(defvar *egs* nil)
(defvar *settings* nil)
(defvar *help* "
runr: simple lisp
(c)2032 Tim Menzies <timm@ieee.org> BSD-2

USAGE: lisp runr.lisp [OPTIONS]

OPTIONS:
  -h  help    show help         = nil
  -g  action  start up action   = none
  -s  seed    random number seed = 10019")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  _ _ _    
; | (_) |__ 
; | | | '_ \
; |_|_|_.__/

(defun l() (load "runr"))

(defmacro aif (test then &optional else)
  `(let ((it ,test))
     (if it ,then ,else)))

(defmacro ? (s x &rest xs)
  "recursive slot-value access"
  (if (null xs) `(slot-value ,s ',x) `(? (slot-value ,s ',x) ,@xs)))

(defmacro ! (s) `(first (cdr (assoc ',s *settings*))))

(defun charn (s c &optional (n 0))
  "is `s` a string holding `c` at position `n`?"
  (if (stringp s)
    (if (< n 0)
      (charn s c (+ (length s) n))
      (and (>= n 0) (< n (length s)) (eql c (char s n))))))

(defun trim (s)
 "kill leading,trailing whitespace"
  (string-trim '(#\Space #\Tab #\Newline) s))

(defun thing (s &aux (s1 (trim s)))
  "coerce `s` into a number or string or t or nil or #\?"
  (cond ((equal s1 "?") #\?)
        ((equal s1 "t") t)
        ((equal s1 "nil") nil)
        (t (let ((n (read-from-string s1 nil nil)))
             (if (numberp n) n s1)))))

(defun words (s &optional (char #\,) (filter #'thing) (here 0))
  "generate words, filtered, from`s`"
  (let* ((there (position char s :start here))
         (word  (funcall filter (subseq s here there))))
    (labels ((tail () (if there (words s char filter (1+ there)))))
      (if (equal word "") (tail) (cons word (tail))))))

(defun with-lines (file fun)
  "Call `fun` for each line in `file`"
  (with-open-file (s file)
    (loop (funcall fun (or (read-line s nil) (return))))))

(defvar *seed* 10013)
(defun rand (&optional (n 2))
  "Random float 0.. < n"
  (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
  (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))

(defun randi (&optional (n 2) &aux (base 10000000000.0))
  "Random int 0..n-1"
  (floor (* n (/ (rand base) base))))

(defmacro defstruct+ (x doco &body body)
  "Creates %x for constructor, enables pretty print, hides slots with '_' prefix."
  (let* ((slots (mapcar    (lambda (x) (if (consp x) (car x) x))          body))
         (show  (remove-if (lambda (x) (eq #\_ (char (symbol-name x) 0))) slots)))
    `(progn
       (defstruct (,x (:constructor ,(intern (format nil "%MAKE-~a" x)))) ,doco ,@body)
       (defmethod print-object ((self ,x) str)
         (labels ((fun (y) (format nil ":~(~a~) ~a" y (slot-value self y))))
           (format str "~a" (cons ',x (mapcar #'fun ',show))))))))

(defun settings (s)
  "for lines like '  -Key Flag ..... Default', return (KEY flag (thing Default))"
  (loop :for (flag key . rest)
        :in  (words s #\NewLine (lambda (s1) (words s1 #\Space #'trim)))
        :if  (charn flag #\-)
        :collect (list (intern (string-upcase key)) (thing (car (last rest))) flag)))

(defun cli (settings &optional (args #+clisp ext:*args*
                                    #+sbcl sb-ext:*posix-argv*))
  (dolist (setting settings settings)
    (aif (member (third setting) args :test 'equal)
         (let ((b4  (second it)))
           (setf (second setting) (cond ((eq b4 t) nil)
                                        ((eq b4 nil) t)
                                        (t (thing (second it)))))))))

(defun about ()
  "show help"
  (format t "~a~%~%ACTIONS:~%" *help*)
  (dolist (three *egs*)
    (format t " ~10a : ~a~%" (getf three :name) (getf three :doc))))

(defmacro eg (what doc &rest src)
  "define a example"
  `(push (list :name ',what :doc ',doc :fun (lambda nil ,@src)) *egs*))

(defun egs ()
  "run 'all' actions or just the (! action) action
 (resetting random seed and other setting before each action)"
  (let ((fails 0)
        (b4 (copy-list *settings*)))
    (dolist (eg *egs*)
      (when (member (! action) (list (getf eg :name) "all") :test 'equal)
        (setf *settings* b4
              *seed* (! seed))
        (unless (funcall (getf eg :fun))
          (incf fails)
          (format t "FAIL âM-^]M-^L ~a~%" (getf eg :key)))))
    #+clisp (ext:exit fails)
    #+sbcl  (sb-ext:exit :code fails)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     _      _          
;  __| |__ _| |_ __ _ ___
; / _` / _` |  _/ _` (_-<
; \__,_\__,_|\__\__,_/__/

(defun isNum    (s) (and (> (length s) 1) (upper-case-p (char s 0))))
(defun isGoal   (s) (or (isKlass s) (isLess s) (isMore s)))
(defun isIgnore (s) (charn s #\X -1))
(defun isKlass  (s) (charn s #\! -1))
(defun isLess   (s) (charn s #\- -1))
(defun isMore   (s) (charn s #\+ -1))

(defstruct+ num (at 0) (txt "") (n 0) (mu 0) (m2 0) (w 1) (lo 1E31) (hi -1E21))

(defun make-num (&optional (at 0) (txt ""))
  "summarizes streams of numbers"
  (%make-num :at at :txt txt :w (if (isLess txt) -1 1)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;     _                 
;  __| |___ _ __  ___ ___
; / _` / -_) '  \/ _ (_-<
; \__,_\___|_|_|_\___/__/

(eg "my" "show options" (print 2) t)
(eg "ls" "show options" (print *settings*) t)
(eg "num" "test number" (print (make-num 10 "Aaas-")) t)

(setf *settings* (cli (settings *help*)))
(if (! help) (about) (egs))
```