1. Introduction. This is a `CLIM` implementation of plot lib demonstrated in the book *Common LISP Modules* by Mark Watson. So finally, I'm starting to learn how to do Artificial Intelligence in Common Lisp.

2. Front matters and license issue. This file and generated files are in public domain. You can do what you want including but not limited to copying, modifying and distributing it without prior consent from the author.

3. Begin our prelude. Shadow conflicted symbols. And this is a load script.

```
⟨ load.lisp 3 ⟩ ≡
(quicklisp-client:quickload :mcclim)
(defpackage #:mig
  (:use #:common-lisp #:clim)
  (:shadowing-import-from
   #:common-lisp
   #:interactive-stream-p)
  (:export
   #:init-plot
   #:plot-fill-rect
   #:plot-size-rect
   #:clear-plot
   #:pen-width
   #:plot-frame-rect
   #:plot-line
   #:show-plot
   #:plot-string))
(load #P"mig")
⟨ load patch 16 ⟩
```

4.

```
(in-package #:mig)
```

5. Since there's a delay before a frame get initialized, we use a symbol macro for **w**, which refers to the global variable **frame**.

```
(define-symbol-macro *w* (get-frame-pane *frame* 'display))
(defvar *frame* nil)
```

6. We define the basic plot window here.

```
(define-application-frame plot ()
  ()
  (:panes
   (display :application))
  (:layouts
   (default display)))
```

7. Define *init-plot*. Further portability issues for *process-run-function*.

```
(defun init-plot (&optional (title "Plot␣Window") (xsize 250) (ysize 250))
  "Initializes␣a␣standard␣plot␣window"
  (flet ((run ()
           (let ((frame (make-application-frame
                         'plot
                         :width xsize :height ysize :pretty-name title)))
             (setq *frame* frame)
             (run-frame-top-level frame))))
    (clim-sys:make-process #'run :name "plot")))
```

8. Define plot function for rectangles.

```
(defun plot-fill-rect (x y xsize ysize pattern
                        &aux (pattern (truncate pattern)))
  "Fills␣in␣a␣rectangle␣with␣one␣of␣five␣gray-scale␣values"
  (draw-rectangle* *w* x y (+ x xsize) (+ y ysize)
                   :ink (cond ((< pattern 1) +white+)
                              ((equal pattern 1)
                               +grey70+)
                              ((equal pattern 2)
                               +grey50+)
                              ((equal pattern 3)
                               +grey30+)
                              (t +black+))
                   :filled t))
```

9. Makes a black rectangle of size proportional to val. This is an alternative to using function *plot-fill-rect* for showing graphically the value of a number.

```
(defun plot-size-rect (x y xsize ysize val &aux (val (min val xsize)))
  (draw-rectangle* *w* x y (+ x xsize) (+ y ysize)
                   :ink +white+
                   :filled t)
  (draw-rectangle* *w* x y (+ x val) (+ y val)
                   :ink +black+
                   :filled t))
```

10. Clears (erases) the plot window.

```
(defun clear-plot ()
  (window-clear *w*))
```

11. Since `CLIM` doesn't memorize line thickness, we create a variable for it.

```
(defvar *thickness* 1)
(defun pen-width (nibs)
  "Sets␣the␣drawing␣size␣for␣the␣pen"
  (setf *thickness* nibs))
```

12. Frames a rectangle of size (xsize ysize) at position (x y).

```
(defun plot-frame-rect (x y xsize ysize)
  (let ((x2 (+ x xsize))
        (y2 (+ y ysize)))
    (draw-rectangle* *w* x y x2 y2 :line-thickness *thickness* :filled nil)))
```

13. Draws a line between two points.

```
(defun plot-line (x1 y1 x2 y2)
  (draw-line* *w* x1 y1 x2 y2 :line-thickness *thickness*))
```

14. This function is just a stub.

```
(defun show-plot ()
  nil)
```

15. Draw text.

```
(defun plot-string (x y str &optional (size 10))
  (draw-text* *w* str x y :text-size size :text-face :roman))
(defun plot-string-bold (x y str &optional (size 12))
  (draw-text* *w* str x y :text-size size :text-face :bold))
(defun plot-string-italic (x y str &optional (size 12))
  (draw-text* *w* str x y :text-size size :text-face :italic))
```

16. We have to override loading font functions for CCL since it by default doesn't allow sharing stream across threads.

⟨ load patch 16 ⟩ ≡

```
#+:ccl
(load #P"ccl-patch")
```

This code is used in section 3.

⟨ load patch 16 ⟩ Used in section 3.

MIG