```lisp
;;; cl-typesetting copyright 2003-2004 Marc Battyani see license.txt for the details
;;; You can reach me at marc.battyani@fractalconcept.com or marc@battyani.net
;;; The homepage of cl-typesetting is here: http://www.fractalconcept.com/asp/html/cl-
typesetting.html

(in-package typeset)

;;; A lisp pretty printer for Common Lisp
(defparameter *pp-font-size* 9)
(defparameter *pp-default-decoration* '("courier" (0.0 0.0 0.0)))
(defparameter *pp-keyword-decoration* '("courier" (0.8 0.0 0.0)))
(defparameter *pp-common-lisp-decoration* '("courier" (0.0 0.0 0.4)))
(defparameter *pp-string-decoration* '("courier" (0.0 0.5 0.0)))
(defparameter *pp-comment-decoration* '("courier" (0.2 0.2 0.6)))
(defparameter *pp-symbol-decoration-table* (make-hash-table))

(defun add-symbol-decoration (symbol decoration)
  (setf (gethash symbol *pp-symbol-decoration-table*) decoration))

(loop for (symbol . decoration) in '((defvar "courier-bold" (0.0 0.0 0.5))
                                     (defun "courier-bold" (0.0 0.2 0.5))
                                     (defmethod "courier-bold" (0.0 0.2 0.5)))
      do (add-symbol-decoration symbol decoration))

(defun split-comment (line)
  (let ((comment-pos (position #\; line))); line)))
    (if comment-pos
        (values (subseq line 0 comment-pos)(subseq line comment-pos))
        line)))

(defun clean-line (line)
  (setf line (copy-seq line))
  (map-into line #'(lambda (char)
                     (if (find char "()'`#      ")
                         #\Space
                         char))
            line))

(defun process-lisp-line (line)
  (multiple-value-bind (code comment)(split-comment line)
    (let* ((cleaned-line (clean-line code))
           (cl-package (find-package 'common-lisp))
           (decorations '())
           (start 0)
           (trimmed 0)
           (length (length cleaned-line)))
      (iter:iter
       (setf trimmed (position #\Space cleaned-line :start start :test #'char/=))
       (while (and trimmed (< trimmed length)))
       (for (values obj end) = (ignore-errors
                                 (read-from-string
                                  cleaned-line nil nil
                                  :start trimmed :preserve-whitespace t)))
       (unless (numberp end)
         (setf end (position #\Space cleaned-line :start trimmed :test #'char=)))
       (while (and (numberp end) (< end length)))
       (cond ((keywordp obj)
              (push (list* trimmed end *pp-keyword-decoration*) decorations))
             ((stringp obj)
              (push (list* trimmed end *pp-string-decoration*) decorations))
             ((gethash obj *pp-symbol-decoration-table*)
              (push (list* trimmed end (gethash obj *pp-symbol-decoration-table*)) decorations))
             ((and (symbolp obj)(eq (symbol-package obj) cl-package))
              (push (list* trimmed end *pp-common-lisp-decoration*) decorations)))
       (setf start end))
      (setf start 0)
      (loop for (start-tok end-tok font-name color) in (nreverse decorations) do
            (when (/= start start-tok)
              (with-text-compilation
                  (verbatim (subseq line start start-tok))))
            (with-text-compilation
                (with-style (:font font-name :font-size *pp-font-size* :color color)
                  (verbatim (subseq line start-tok end-tok))))
            (setf start end-tok))
      (with-text-compilation
        (when (< start length)
          (verbatim (subseq line start)))
        (with-style (:font (first *pp-comment-decoration*)
                           :font-size *pp-font-size*
                           :color (second *pp-comment-decoration*))
            (verbatim comment)
            (when (zerop length) (verbatim " ")) :eol)))))

(defmethod process-lisp-code ((s stream))
  (with-text-compilation
      (paragraph (:h-align :left :top-margin 10
                  :left-margin 5 :right-margin 5
                  :font "courier" :font-size *pp-font-size*)
                 (loop for line = (read-line s nil)
                       while line
                       do (with-text-compilation
                              (process-lisp-line line)
                            )))))

(defmethod process-lisp-code ((lisp-file pathname))
  (with-open-file (s lisp-file :direction :input)
    (process-lisp-code s)))

(defmethod process-lisp-code ((lisp-string string))
  (with-input-from-string (s lisp-string)
    (process-lisp-code s)))

(defun pprint-lisp-file (lisp-code pdf-file &optional title)
  (with-document ()
    (let* ((margins '(30 50 30 50))
           (print-stamp (multiple-value-bind (second minute hour date month year)
                            (get-decoded-time)
                          (format nil "Printed on ~4D-~2,'0D-~2,'0D ~2,'0D:~2,'0D"
                                  year month date hour minute)))
           (header (compile-text ()
                         (paragraph (:h-align :center
                                     :font "Helvetica-BoldOblique" :font-size 12)
                                    (put-string (cond
                                                  (title title)
                                                  ((pathnamep lisp-code)(namestring lisp-code))
                                                  (t "Lisp Source Code"))))
                         (vspace 1)
                         (hrule :dy 0.5)))
           (footer (lambda (pdf:*page*)
                      (compile-text (:font "Helvetica" :font-size 10)
                           (hrule :dy 1/2)
                           (hbox (:align :center :adjustable-p t)
                                 (verbatim print-stamp)
                                 :hfill
                                 (verbatim
                                  (format nil "Page ~d"
                                          (1+ (position pdf:*page* (pages pdf:*document*))))))))))
           (content (compile-text () (process-lisp-code lisp-code))))
      (draw-pages content :margins margins :header header :footer footer)
      (when pdf:*page* (finalize-page pdf:*page*))
      (pdf:write-document pdf-file))))
```