```
;;;; SPLIT-SEQUENCE
;;;
;;; This code was based on Arthur Lemmens' in
;;; <URL:http://groups.google.com/groups?as_umsgid=39F36F1A.B8F19D20%40simplex.nl>;
;;;
;;; changes include:
;;;
;;; * altering the behaviour of the :from-end keyword argument to
;;; return the subsequences in original order, for consistency with
;;; CL:REMOVE, CL:SUBSTITUTE et al. (:from-end being non-NIL only
;;; affects the answer if :count is less than the number of
;;; subsequences, by analogy with the above-referenced functions).
;;;
;;; * changing the :maximum keyword argument to :count, by analogy
;;; with CL:REMOVE, CL:SUBSTITUTE, and so on.
;;;
;;; * naming the function SPLIT-SEQUENCE rather than PARTITION rather
;;; than SPLIT.
;;;
;;; * adding SPLIT-SEQUENCE-IF and SPLIT-SEQUENCE-IF-NOT.
;;;
;;; * The second return value is now an index rather than a copy of a
;;; portion of the sequence; this index is the `right' one to feed to
;;; CL:SUBSEQ for continued processing.

;;; There's a certain amount of code duplication here, which is kept
;;; to illustrate the relationship between the SPLIT-SEQUENCE
;;; functions and the CL:POSITION functions.

;;; Examples:
;;;
;;; * (split-sequence #\; "a;;b;c")
;;; -> ("a" "" "b" "c"), 6
;;;
;;; * (split-sequence #\; "a;;b;c" :from-end t)
;;; -> ("a" "" "b" "c"), 0
;;;
;;; * (split-sequence #\; "a;;b;c" :from-end t :count 1)
;;; -> ("c"), 4
;;;
;;; * (split-sequence #\; "a;;b;c" :remove-empty-subseqs t)
;;; -> ("a" "b" "c"), 6
;;;
;;; * (split-sequence-if (lambda (x) (member x '(#\a #\b))) "abracadabra")
;;; -> ("" "" "r" "c" "d" "" "r" ""), 11
;;;
;;; * (split-sequence-if-not (lambda (x) (member x '(#\a #\b))) "abracadabra")
;;; -> ("ab" "a" "a" "ab" "a"), 11
;;;
;;; * (split-sequence #\; ";oo;bar;ba;" :start 1 :end 9)
;;; -> ("oo" "bar" "b"), 9

(defpackage "SPLIT-SEQUENCE"
  (:use "CL")
  (:nicknames "PARTITION")
  (:export "SPLIT-SEQUENCE" "SPLIT-SEQUENCE-IF" "SPLIT-SEQUENCE-IF-NOT"
           "PARTITION" "PARTITION-IF" "PARTITION-IF-NOT"))

(in-package "SPLIT-SEQUENCE")
```

```
(defun split-sequence (delimiter seq &key (count nil) (remove-empty-subseqs nil) (from-end nil) (start 0) (end nil) (test nil test-supplied) (test-not
  "Return a list of subsequences in seq delimited by delimiter.

If :remove-empty-subseqs is NIL, empty subsequences will be included
in the result; otherwise they will be discarded.  All other keywords
work analogously to those for CL:SUBSTITUTE.  In particular, the
behaviour of :from-end is possibly different from other versions of
this function; :from-end values of NIL and T are equivalent unless
:count is supplied. The second return value is an index suitable as an
argument to CL:SUBSEQ into the sequence indicating where processing
stopped."
  (let ((len (length seq))
        (other-keys (nconc (when test-supplied
                             (list :test test))
                           (when test-not-supplied
                             (list :test-not test-not))
                           (when key-supplied
                             (list :key key)))))
    (unless end (setq end len))
    (if from-end
      (loop for right = end then left
            for left = (max (or (apply #'position delimiter seq
                                       :end right
                                       :from-end t
                                       other-keys)
                                -1)
                            (1- start))
            unless (and (= right (1+ left))
                        remove-empty-subseqs) ; empty subseq we don't want
            if (and count (>= nr-elts count))
            ;; We can't take any more. Return now.
            return (values (nreverse subseqs) right)
            else
            collect (subseq seq (1+ left) right) into subseqs
            and sum 1 into nr-elts
            until (< left start)
            finally (return (values (nreverse subseqs) (1+ left))))
      (loop for left = start then (+ right 1)
            for right = (min (or (apply #'position delimiter seq
                                        :start left
                                        other-keys)
                                 len)
                             end)
            unless (and (= right left)
                        remove-empty-subseqs) ; empty subseq we don't want
            if (and count (>= nr-elts count))
            ;; We can't take any more. Return now.
            return (values subseqs left)
            else
            collect (subseq seq left right) into subseqs
            and sum 1 into nr-elts
            until (>= right end)
            finally (return (values subseqs right))))))

(defun split-sequence-if (predicate seq &key (count nil) (remove-empty-subseqs nil) (from-end nil) (start 0) (end nil) (key nil key-supplied))
  "Return a list of subsequences in seq delimited by items satisfying
predicate.
```

```
  8       If :remove-empty-subseqs is NIL, empty subsequences will be included
  9       in the result; otherwise they will be discarded.          All other keywords
120       work analogously to those for CL:SUBSTITUTE-IF.  In particular, the
  1       behaviour of :from-end is possibly different from other versions of
  2       this function; :from-end values of NIL and T are equivalent unless
  3       :count is supplied. The second return value is an index suitable as an
  4       argument to CL:SUBSEQ into the sequence indicating where processing
  5       stopped."
  6         (let ((len (length seq))
  7               (other-keys (when key-supplied
  8                             (list :key key))))
  9           (unless end (setq end len))
130           (if from-end
  1               (loop for right = end then left
  2                     for left = (max (or (apply #'position-if predicate seq
  3                                                :end right
  4                                                :from-end t
  5                                                other-keys)
  6                                         -1)
  7                                     (1- start))
  8                     unless (and (= right (1+ left))
  9                                 remove-empty-subseqs) ; empty subseq we don't want
140                     if (and count (>= nr-elts count))
  1                     ;; We can't take any more. Return now.
  2                     return (values (nreverse subseqs) right)
  3                     else
  4                     collect (subseq seq (1+ left) right) into subseqs
  5                     and sum 1 into nr-elts
  6                     until (< left start)
  7                     finally (return (values (nreverse subseqs) (1+ left))))
  8               (loop for left = start then (+ right 1)
  9                     for right = (min (or (apply #'position-if predicate seq
150                                                 :start left
  1                                                 other-keys)
  2                                         len)
  3                                     end)
  4                     unless (and (= right left)
  5                                 remove-empty-subseqs) ; empty subseq we don't want
  6                     if (and count (>= nr-elts count))
  7                     ;; We can't take any more. Return now.
  8                     return (values subseqs left)
  9                     else
160                     collect (subseq seq left right) into subseqs
  1                     and sum 1 into nr-elts
  2                     until (>= right end)
  3                     finally (return (values subseqs right))))))
  4
  5       (defun split-sequence-if-not (predicate seq &key (count nil) (remove-empty-subseqs nil) (from-end nil) (start 0) (end nil) (key nil key-supplied))
  6         "Return a list of subsequences in seq delimited by items satisfying
  7       (CL:COMPLEMENT predicate).
  8
  9       If :remove-empty-subseqs is NIL, empty subsequences will be included
170       in the result; otherwise they will be discarded.          All other keywords
  1       work analogously to those for CL:SUBSTITUTE-IF-NOT.  In particular,
  2       the behaviour of :from-end is possibly different from other versions
  3       of this function; :from-end values of NIL and T are equivalent unless
  4       :count is supplied. The second return value is an index suitable as an
  5       argument to CL:SUBSEQ into the sequence indicating where processing
  6       stopped."
```

```
      (let ((len (length seq))
            (other-keys (when key-supplied
                          (list :key key))))
        (unless end (setq end len))
        (if from-end
            (loop for right = end then left
                  for left = (max (or (apply #'position-if-not predicate seq
                                             :end right
                                             :from-end t
                                             other-keys)
                                      -1)
                                  (1- start))
                  unless (and (= right (1+ left))
                              remove-empty-subseqs) ; empty subseq we don't want
                  if (and count (>= nr-elts count))
                  ;; We can't take any more. Return now.
                  return (values (nreverse subseqs) right)
                  else
                  collect (subseq seq (1+ left) right) into subseqs
                  and sum 1 into nr-elts
                  until (< left start)
                  finally (return (values (nreverse subseqs) (1+ left))))
            (loop for left = start then (+ right 1)
                  for right = (min (or (apply #'position-if-not predicate seq
                                              :start left
                                              other-keys)
                                       len)
                                   end)
                  unless (and (= right left)
                              remove-empty-subseqs) ; empty subseq we don't want
                  if (and count (>= nr-elts count))
                  ;; We can't take any more. Return now.
                  return (values subseqs left)
                  else
                  collect (subseq seq left right) into subseqs
                  and sum 1 into nr-elts
                  until (>= right end)
                  finally (return (values subseqs right))))))

;;; clean deprecation

(defun partition (&rest args)
  (apply #'split-sequence args))

(defun partition-if (&rest args)
  (apply #'split-sequence-if args))

(defun partition-if-not (&rest args)
  (apply #'split-sequence-if-not args))

(define-compiler-macro partition (&whole form &rest args)
  (declare (ignore args))
  (warn "PARTITION is deprecated; use SPLIT-SEQUENCE instead.")
  form)

(define-compiler-macro partition-if (&whole form &rest args)
  (declare (ignore args))
  (warn "PARTITION-IF is deprecated; use SPLIT-SEQUENCE-IF instead.")
  form)
```

```
  6
  7       (define-compiler-macro partition-if-not (&whole form &rest args)
  8         (declare (ignore args))
  9         (warn "PARTITION-IF-NOT is deprecated; use SPLIT-SEQUENCE-IF-NOT instead")
240         form)
  1
  2       (pushnew :split-sequence *features*)
```