```
;; The first four lines of this file were added by Dracula.
;; They tell DrScheme that this is a Dracula Modular ACL2 program.
;; Leave these lines unchanged so that DrScheme can properly load this file.
#reader(planet "reader.ss" ("cce" "dracula.plt") "modular" "lang")
(interface Ibasiclex
  (sig split-at-delimiter (ds xs))
  (sig span (ps xs))
  (sig splitoff-prefix (ps xs))
  (sig splitoff-prefix-upr (ps xs))
  (sig splitoff-prefix-chr (tok-str xs))
  (sig split-on-token-gen (tok xs))
  (sig split-on-token-chr (tok xs))
  (sig split-on-token (tok xs))

  (con split-at-delimiter-conserves-elements
       (implies (and (true-listp ds)
                     (true-listp xs))
                (let* ((bfaf (split-at-delimiter ds xs))
                       (bf (car bfaf))
                       (af (cadr bfaf)))
                  (equal (append bf af) xs))))
  (con split-at-delimiter-prefix-contains-no-delimiters
       (implies (and (true-listp ds)
                     (true-listp xs)
                     (member-equal d ds))
                (not (member-equal d (car (split-at-delimiter ds xs))))))
  (con split-at-delimiter-suffix-not-empty-means-xs-not-empty
       (implies (and (true-listp ds)
                     (true-listp xs)
                     (consp (cadr (split-at-delimiter ds xs))))
                (consp xs)))
  (con split-at-delimiter-suffix-starts-with-delim-if-possible
       (implies (and (true-listp ds)
                     (true-listp xs)
                     (consp (cadr (split-at-delimiter ds xs))))
                (member-equal (car (cadr (split-at-delimiter ds xs))) ds)))
  (con split-at-delimiter-delivers-shorter-list
       (implies (and (true-listp ds)
                     (true-listp xs))
                (<= (len (cadr (split-at-delimiter ds xs)))
                    (len xs))))

  (con splitoff-delivers-shorter-list
       (implies (and (true-listp ps)
                     (true-listp xs)
                     (consp ps)
                     (null (cadr (splitoff-prefix ps xs))))
                (< (len (caddr (splitoff-prefix ps xs)))
                   (len xs))))

  (con splitoff-upr-delivers-shorter-list
       (implies (and (true-listp ps)
                     (true-listp xs)
                     (consp ps)
                     (null (cadr (splitoff-prefix-upr ps xs))))
                (< (len (caddr (splitoff-prefix-upr ps xs)))
                   (len xs))))

  (con split-on-token-gen-delivers-shorter-list
       (implies (and (true-listp tok)
                     (consp tok)
                     (true-listp xs)
                     (consp xs))
                (< (len (caddr (split-on-token-gen tok xs)))
                   (len xs)))
       :hints (("Goal"
                :induct (split-on-token-gen tok xs)))) ;hint needed
```

```
(con split-on-token-chr-delivers-shorter-list
     (implies (and (true-listp tok)
                   (consp tok)
                   (true-listp xs)
                   (consp xs))
              (< (len (caddr (split-on-token-chr tok xs)))
                 (len xs)))
     :hints (("Goal"
              :induct (split-on-token-chr tok xs)))) ;hint needed

(con split-on-token-delivers-shorter-list
     (implies (and (or (and (true-listp tok)
                            (consp tok))
                       (and (stringp tok)
                            (> (len tok) 0)))
                   (true-listp xs)
                   (consp xs))
              (< (len (caddr (split-on-token tok xs)))
                 (len xs))))


)
```