```
;; 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")
#| Team Steele
   Software Engineering I
   Ixmlminidom

   Software that creates a document object model from XML input.
|#

(defconst *whitespace*
  (list (code-char 32)
        (code-char 10)
        (code-char 9)
        (code-char 11)
        (code-char 12)
        (code-char 13)
        (code-char 27)))
(defconst *endtagname* (cons #\> (cons #\/ *whitespace*)))
(defconst *endattrname* (cons #\= *whitespace*))

(interface Ixmlminidom
  ;xml-escape (unescapedchars) → returns string with bad chars replaced
  ;    with entities
  (sig xml-escape (unescapedchars))

  ;xml-serialize-attributes (attributes) → Returns a string that is
  ;    xml that represents the passed in attribute list.
  (sig xml-serialize-attributes (attributes))

  ;xml-serizlize-nodes (xmlnodes) → Returns a string containing xml
  ;    nodes that represents the node list, xmlnodes.
  (sig xml-serialize-nodes (xmlnodes))

  ;xml-serizlize-dom (xmlnode) → Returns a string containing an xml
  ;    document that represents the dom passed in through xmlnode.
  (sig xml-serialize-dom (xmlnode))

  ;xml-unescape (escapedchars) → string with entities replaced
  (sig xml-unescape (escapedchars))

  ;xml-readnodeproperties (xmlchars) →
  ; returns (mv attributes remainingxmlstring)
  (sig xml-readnodeproperties (xmlchars))

  ;xml-skipdontcares (xmlchars) → returns next xmlchars sans don't cares
  (sig xml-skipdontcares (xmlchars))

  ;xml-readnodes (xmlchars) → returns (mv nodes remainingxmlstring)
  (sig xml-readnodes (xmlchars))

  ;xml-readnode (xmlchars) → returns the root node from xmlstring
  (sig xml-readnode (xmlchars))

  ;xml-getnodes (node nodename) → returns children of node with type
  ;  nodename
  (sig xml-getnodes (node nodename))

  ;xml-getdeepnodes (node nodename) → returns children of node with type
  ;  nodename searching recursively using DFS with node as root.
  (sig xml-getdeepnodes (node nodename))

  ;xml-getnode (node nodename) → returns first child node with type
  ;  nodename
  (sig xml-getnode (node nodename))
```

```
;xml-getdeepnode (node nodename) → returns first child node with type
;  nodename searching recursively using DFS with node as root.
(sig xml-getdeepnode (node nodename))

;xml-getattribute (node attributename) → returns the value of node's
; attribute with name attributename
(sig xml-getattribute (node attributename))

;xml-gettext (node) → returns the composite of all text inside of a node
(sig xml-gettext (node))

;xml-isattribute (attribute) → returns true iff attribute is an mv of
;  length 2 with both elements of the mv being strings
(sig xml-isattribute (attribute))

;xml-isattributelist (attributes) → returns true iff attributes
;  is nil or a list of attributes
(sig xml-isattributelist (attributes))

;xml-isnode (node) → returns true iff node is actually a node
(sig xml-isnode (node))

;xml-isnodelist (nodes) → returns true iff nodes is a list of nodes
(sig xml-isnodelist (nodes))

(sig xml-bfsfindnodes (nodes nodename))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Contracts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(con xml-unescape-returns-string
     (implies (standard-char-listp x)
              (stringp (xml-unescape x))))
(con xml-skipdontcares-lessthanequal-xmlchars
     (imlies (and (standard-char-listp x)
                  (equal (length x) y))
             (<= (xml-skipdontcares x) y)))
(con xml-getnodes-returns-nodes
     (implies (and
               (stringp y)
               (xml-isnode x))
              (xml-isnodelist (xml-getnodes y x))))
(con xml-getnodes-returns-children
     (implies (and
               (xml-isnode x)
               (> (length (caddr x)) 0)
               (equal node (car (caddr x)))
               (equal y (car node))
               (stringp y)
               )
              (let ((res (xml-getnodes y x)))
              (and
               (> (length res) 0)
               (equal node (car res))))))
(con xml-getdeepnodes-returns-nodes
     (implies (and
               (stringp y)
               (xml-isnode x))
              (xml-isnodelist (xml-getdeepnodes y x))))
(con xml-getnode-returns-node-or-nil
     (implies (and
               (xml-isnode x)
               (stringp y))
              (let ((res (xml-getnode x y)))
                (or
                 (null res)
                 (xml-isnode res)))))
```

```
  (con xml-getdeepnode-returns-node-or-nil
       (implies (and
                 (xml-isnode x)
                 (stringp y))
                (let ((res (xml-getdeepnode x y)))
                  (or
                   (null res)
                   (xml-isnode res)))))
  (con xml-getattribute-returns-string
       (implies (and
                 (xml-isnode x)
                 (stringp y))
                (stringp (xml-getattribute x y))))
  (con xml-gettext-returns-string
       (implies (xml-isnode x))
                (stringp (xml-gettext x)))

  (con xml-readnode-serialize-dom-invertible
       (implies (xml-isnode x)
                (equal x (xml-readnode (xml-serialize-dom x)))))
)
```