Purpose

To translate BridgeNet XML into a display of the hands, travelers, score cards, and rankings in HTML format.

Rough Process

1) Read xml file as a string into a variable
2) Use xmlminidom to process the string into “node”s
3) Iterate through all board nodes for each of these generate HTML
    1. Make boards with travelers
    2. Make boards without travelers
    3. Make personal score card
    4. Make rankings
4) Output html headers with style sheets, results from 3, and footers to files.

Data Structures

The only specialized data structures are for usage with xmlminidom:

```
(mv nodename attributes children) |  (mv 'text nil value)        ;node and text
attributes = (cons attribute attributes) | nil                   ;attributes
attribute = (mv attributename value)                             ;attribute
nodes = (cons node nodes) | nil                                  ;nodes
children = nodes                                                 ;children
```

Interfaces and Contracts

xmlminidom:

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

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

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

Given any input xml-readnode will either return a structure of type xml-isnode or nil.

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

xml-getdeepnodes (node nodename) → returns descendants of node with type nodename, searching recursively using DFS with node as root.

Assuming (xml-isnode node) is true, xml-getdeepnodes will return:

- something of type xml-isnodelist
- every node with name nodename.

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

Assuming (xml-isnode node) is true, xml-getdeepnode will return:

- something of type xml-isnode
- every node with name nodename.

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

Assuming (xml-isnode node) will return a string (empty string if not found)

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

Assuming (xml-isnode node) will return a string (empty string if no 'text elements)

xml-bfsfindnodes (nodes nodename) → returns a list of children nodes of type nodename using BFS search that are at the shallowest depth.

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

xml-isattributelist (attributes) → returns true iff attributes is nil or a list of mv's of length 2 with both elements of each mv being strings

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

xml-isnodelist (nodes) → returns true iff nodes is a list of nodes or nil

board:

serializedresults (xmlnodes) → returns a string consisting of the concatenation of results table rows from each "Result" node

getseparateresults (xmlnodes boardnum ns1 ew1) → grabs the results for one board separately from the board information without html serialization where xmlnodes is the list of results for a board, boardnum is the boardnum, ns1 and ew1 are the initial lists. It returns a structure like:

```
(mv ns ew)
where
 ns = (list
       (cons (mv pairns section)
            (list
              (mv boardnum pairew totalscore pointsns))
              ...)
       ...)
 ew = (list
       (cons (mv pairew section)
            (list
              (mv boardnum pairns totalscore pointsns))
              ...)
       ...)
```

getallseparateresults (boardnodes) → converts boardnodes, a list of Board nodes, to a sequence like:

```
(mv ns ew)
where
 ns = (list
       (cons (mv pairns section)
             (list
               (mv boardnum pairew totalscore pointsns))
               ...)
       ...)
 ew = (list
       (cons (mv pairew section)
             (list
               (mv boardnum pairns totalscore pointsns))
               ...)
       ...)
```

serializedboards (xmlnodes) → returns appended "board" class divs with their "results" tables from the xmlnode "Board" and "results" formatted to be rendered with the deal and results as required by description.

serializehands (xmlnodes vulnerable dealer) → serialize hands into divs, where the class is the hand direction derived from xmlnodes, the "vulnerable" and "dealer" divs are added inside as necessary, and the cards are added to each hand via serializehandcards.
xmlnodes conforms to the minidom structure and is of type hand.
vulnerable is the text content of the Vulnerable node
dealer is the text content of the Dealer node

serializehandcards (xmlnodes) → serialize the hand cards from xmlnodes where xmlnodes is a list of node structures representing "Suit"s.
For...

```
<Suit symbol="S">832</Suit>
<Suit symbol="H">QT42</Suit>
<Suit symbol="D">A865</Suit>
<Suit symbol="C">A9</Suit>
```

You will get something like...

```
&spades;832<br/>
&hearts;QT42<br/>
&diams;A865<br/>
&clubs;A9<br/>
```

psc:

getpsc (xmlnodes) → Given xmlnodes, this returns a string HTML table for each PSC
Notes: Recurses for every pairid, parsing out each pair's score card

getnameforid (pairid data) → The string pairid needs to define the direction (E-W). section (A, B),

and Number (1,2,3) returns string list (list NameOfPerson1 NameOfPerson2)
Notes: Pulls a pair's names from the nodes data

getboardsforpair (pairid results) → Given string pairid, where PairID needs to define the direction (E-W). section (A, B), and Number (1,2,3), and node results returns a string Boards-HTML which is a HTML table with rows for each match
Board, Direction, Versus, Score, Matchpoints are the columns
Notes: Iterates over the list returned from getseperateresults to pull one pair's board results

rankings:

getrankings (rankingnodes) → Given list of ranking nodes, returns a string HTML will be some header information then a table for each Section/Direction pair
Columns: Pair No. (section, pair ID, and direction, e.g., "A1 N-S"); Players; Strat; an Overall Rank mv for Strats A, B; and C, Section Rank mv for Strats A, B, and C; Matchpoint Score; Percentage Score; Masterpoint Award
Notes: Parses out the rankings data into html tables

getcontestants (section dir id sections) → Find the Contestants node with ID id, in the Section in sections (a list of Section nodes), where Section has Direction dir and SectionLabel section.

getcontestantsnames (contestants) → For the Contestants node given by contestants, extract the names from the text contents of the Player nodes.

io:

main(bridgeXML state)→Given a duplicate bridge XML file, extracts appropriate information to creates four HTML pages that link together: boards, boards with travelers, rankings, and personal score card webpages.

boards-no-trav(bridgeXML state)→Given a duplicate bridge XML file, gets the following information from each board listed in the XML file and creates an HTML page with this information formatted into tables:
Board numbers, dealer, vulnerable, and hands for each direction

boards-trav(bridgeXML state)→ Given a duplicate bridge XML file, gets the same information as boards-no-trav as well as the travelers information for each board, which includes the total score and matchpoints for all pairs at that board.

Creates HTML page with this information with the board information and travelers information in separate tables.

rankings(bridgeXML state)→ Given a duplicate bridge XML file, creates a rankings table in an HTML file including each pairs ranking and various other stats such as matchpoint and percentage score.

personal-score-cards(bridgeXML state)→Given a duplicate bridge XML file, creates an HTML file containing personal score cards for each pair, which includes information about each match they played an against whom.

PROBE Software Size Estimate:

Reused Functions	*LOC*
xml-gettext	*9*
xml-getattribute	*8*
xml-getnode	*2*
xml-getnodes	*10*
xml-readnode	*5*
xml-readnodes	*53*
xml-skipdontcares	*20*
xml-readnodeproperties	*24*
xml-unescape	*27*
splitoff-prefix-mv	*3*
split-at-delimiter-mv	*3*
span-mv	*3*
split-on-token-mv	*3*
serializeboards	*32*
getresults	*35*
serializehands	*62*
serializehandcards	*32*
suit-list?	*8*
suit?	*38*

Total Reused LOC: 377

Historical Data Table for PROBE

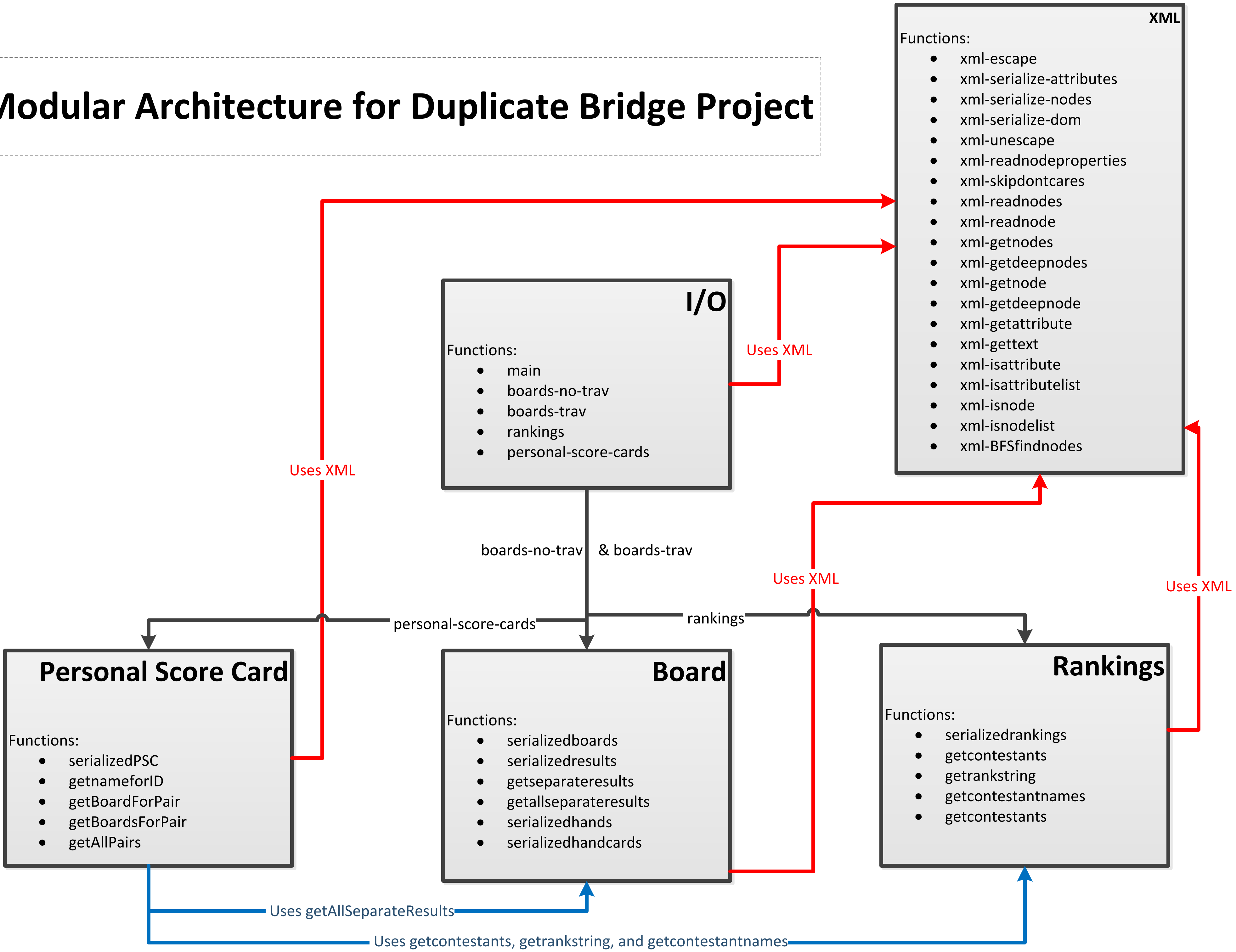
	Tiny(7%)	*Small(24%)*	*Medium(38%)*	*Large(24%)*	*Huge(7%)*
Avg. LOC	*1.69*	*3.82*	*7.34*	*16.31*	*41.71*

Predicted Functions	*Predicted Size*
xml-getdeepnodes	*Medium*
serializeseparateresults	*Large*
serializeresults	*Huge*
getPSC	*Large*
getnameforID	*Medium*
getboardsforpair	*Medium*
getrankings	*Medium*
getcontestant	*Medium*
boards-no-trav	*Small*
boards-trav	*Small*
rankings	*Small*
personal-score-cards	*Medium*
main	*Large*

Total Predicted LOC: 146.14

Total Estimated LOC: 523

Modular Architecture for Duplicate Bridge Project



```
;; 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")
(require "../interfaces/Ibasiclex.lisp")
(require "../modules/Mbasiclex.lisp")

(module Tbasiclex
  (import Ibasiclex)

  (include-book "list-utilities" :dir :teachpacks)
  (include-book "testing" :dir :teachpacks)

  (check-expect
   (split-on-token "abc" (str->chrs "123abc456"))
   (list (str->chrs "123") (str->chrs "ABC") (str->chrs "456")))
)

(link Test
      (Mbasiclex Tbasiclex))

(invoke Test)
```

```
#| Team Steele
   Software Engineering I
   Txmlminidom

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

(require "../interfaces/iBoard.lisp")
(require "../modules/mBoard.lisp")
(require "../modules/Mbasiclex.lisp")

(module tBoard
  (import iBoard)

  (include-book "testing" :dir :teachpacks)
;=======================SANITY CHECKS==============================;
; serializedresults (xmlnodes prefix postfix)
  (check-expect (serializedresults nil *tablehead* *tabletail*)
              (string-append *tablehead* *tabletail*))

  (check-expect
   (serializedresults
    (list
     (mv
      "Result"
      (list
       (mv "SectionLabel" "A")
       (mv "PairID-NS" "3")
       (mv "PairID-EW" "5")
       )
      (list
       (mv "TotalScore"
           (list (mv "Direction" "N-S"))
           (list (mv 'text nil "5.0"))
           )
       (mv "MatchpointsNS" nil (list (mv 'text nil "120.0")))
       (mv "MatchpointsEW" nil (list (mv 'text nil "120.0")))
       )
      ))
    *tablehead*
    *tabletail*)
   (stringlist-append
    (list
     *tablehead*
     "<tr>"
     "<td>A3</td>"
     "<td>A5</td>"
     "<td>5.0</td><td>&nbsp;</td>"
     "<td>120.0</td>"
     "<td>120.0</td>"
     "</tr>"
     *tabletail*)))

  (check-expect
   (serializedresults
    (list
     (mv
      "Result"
      (list
       (mv "SectionLabel" "A")
       (mv "PairID-NS" "3")
       (mv "PairID-EW" "5")
       )
      (list
       (mv "TotalScore"
           (list (mv "Direction" "E-W"))
           (list (mv 'text nil "7.0"))
```

```
          )
      (mv "MatchpointsNS" nil (list (mv 'text nil "120.0")))
      (mv "MatchpointsEW" nil (list (mv 'text nil "120.0")))
      )
     ))
   *tablehead*
   *tabletail*)
  (stringlist-append
   (list
    *tablehead*
    "<tr>"
    "<td>A3</td>"
    "<td>A5</td>"
    "<td>&nbsp;</td><td>7.0</td>"
    "<td>120.0</td>"
    "<td>120.0</td>"
    "</tr>"
    *tabletail*)))


 ; serializedboards (xmlnode)
 (check-expect (serializedboards nil) "")
 (check-expect
  (serializedboards
   (list
    (mv
     "Board"
     nil
     (list
      (mv "Vulnerable" nil (list (mv 'text nil "NS")))
      (mv "Dealer" nil (list (mv 'text nil "N")))
      (mv "BoardNo" nil (list (mv 'text nil "33")))
      (mv "Deal" nil
          (list
           (mv
            "Hand"
            (list (mv "direction" "N"))
            (list
             (mv "Suit"
                 (list (mv "symbol" "S"))
                 (list (mv 'text nil "234")))
             ))
         )))))
   )
  (stringlist-append (list
                      "<div class=\"board\">"
                      "<div class=\"boardnum\">Board: 33</div>\n"
                      "<div class=\"N\">"
                      "<div class=\"dealer\">Dealer</div>\n"
                      "<div class=\"vulnerable\">Vulnerable</div>\n"
                      "&spades;234<br />\n"
                      "</div>\n"
                      "</div>\n"
                      *tablehead*
                      *tabletail*)))

 (check-expect
  (serializedboards
   (list
    (mv
     "Board"
     nil
     (list
      (mv "Vulnerable" nil (list (mv 'text nil "NS")))
      (mv "Dealer" nil (list (mv 'text nil "N")))
      (mv "BoardNo" nil (list (mv 'text nil "33")))
      (mv "Deal" nil
           (list
            (mv
             "Hand"
             (list (mv "direction" "N"))
             (list
              (mv "Suit"
                  (list (mv "symbol" "S"))
                  (list (mv 'text nil "234")))
              ))
            ))
       (mv
        "Result"
        (list
         (mv "SectionLabel" "A")
         (mv "PairID-NS" "3")
         (mv "PairID-EW" "5")
         )
        (list
         (mv "TotalScore"
             (list (mv "Direction" "N-S"))
             (list (mv 'text nil "5.0"))
             )
         (mv "MatchpointsNS" nil (list (mv 'text nil "120.0")))
         (mv "MatchpointsEW" nil (list (mv 'text nil "120.0")))
         )
        )
       )))
     )
    (stringlist-append (list
                        "<div class=\"board\">"
                        "<div class=\"boardnum\">Board: 33</div>\n"
                        "<div class=\"N\">"
                        "<div class=\"dealer\">Dealer</div>\n"
                        "<div class=\"vulnerable\">Vulnerable</div>\n"
                        "&spades;234<br />\n"
                        "</div>\n"
                        "</div>\n"
                        *tablehead*
                        "<tr>"
                        "<td>A3</td>"
                        "<td>A5</td>"
                        "<td>5.0</td><td>&nbsp;</td>"
                        "<td>120.0</td>"
                        "<td>120.0</td>"
                        "</tr>"
                        *tabletail*)))
;======================================================================;


;=======================PREDICATE TESTS================================;
  (include-book "doublecheck" :dir :teachpacks)

  (defproperty serializedresults-nil=string-append-prefix-postfix-tst
    :repeat 100
    (prefix :value (random-string)
            postfix :value (random-string))
    (implies (and (stringp prefix) (stringp postfix))
             (string-equal (serializedresults nil prefix postfix)
                           (string-append prefix postfix)))
    )

  (defproperty serializedresults-results-proportional-to-input-tst
    :repeat 100
    (n     :value (random-between 1 200)
           nodes :value (random-list-of
                 (mv
                  "Result"
                  (list
                   (mv "SectionLabel" "A")
                   (mv "PairID-NS" "3")
                   (mv "PairID-EW" "5")
                   )
                  (list
                   (mv "TotalScore"
                       (list (mv "Direction" "N-S"))
                       (list (mv 'text nil "5.0"))
                       )
                   (mv "MatchpointsNS" nil (list (mv 'text nil "120.0")))
                   (mv "MatchpointsEW" nil (list (mv 'text nil "120.0")))
                   )
                  )
                 :size n))
    (implies (> n 0)
           (< (length (serializedresults (cdr nodes) "" ""))
              (length (serializedresults nodes "" ""))))
    )
;======================================================================;

  )

  (link Test
        (Mbasiclex mBoard tBoard))

  (invoke Test)
```

```
;; 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")

(require "../modules/Mio.lisp")
(require "../modules/Mbasiclex.lisp")
(require "../modules/Mboard.lisp")
(require "../modules/Mpsc.lisp")
(require "../modules/Mrankings.lisp")
(require "../modules/Mxmlminidom.lisp")


(link Test
      (Mbasiclex Mxmlminidom Mboard Mrankings Mpsc Mio))

(invoke Test)

(set-state-ok t)

;(main "051115A" state)
```

```
;; 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")

(require "../modules/Mbasiclex.lisp")
(require "../modules/Mrankings.lisp")
(require "../modules/Mxmlminidom.lisp")

(module Trankings
  (import Ixmlminidom)
  (import Irankings)



(include-book "testing" :dir :teachpacks)

(check-expect (getmatchpointtotal nil) 0))

(link Test
      (Mbasiclex Mxmlminidom Mrankings Trankings))

(invoke Test)
```

```lisp
;; 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
   Txmlminidom

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

(require "../interfaces/Ixmlminidom.lisp")
(require "../modules/Mxmlminidom.lisp")
(require "../modules/Mbasiclex.lisp")

(module Txmlminidom
  (import Ixmlminidom)

  (include-book "testing" :dir :teachpacks)
  (include-book "doublecheck" :dir :teachpacks)
  (include-book "audio" :dir :teachpacks)

  (play-wav "ragtime.wav" t)

  (defconst
    *Oface*
    "<bob><slidell id=\"porter\">12 &amp; 3<jumptoconclusions /></slidell></bob>")

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ; Functions to generate an xmlminidom tree
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  (defun normalize-text (nodes)
    (if (< (length nodes) 2)
        nodes
        (let ((a (car nodes))
              (b (cadr nodes))
              (rest (cddr nodes)))
          (if (and
               (equal (car a) 'text)
               (equal (car b) 'text))
              (normalize-text
               (cons (mv 'text nil
                         (string-append (caddr a) (caddr b))) rest))
              (cons (if (equal (car a) 'text)
                        a
                        (mv (car a) (cadr a) (normalize-text (caddr a))))
                    (normalize-text (cons b rest)))))))

  (defrandom randomxmltext (min)
    (if (equal min 0)
        ""
        (string-append
         (coerce (list (code-char
                        (random-case
                         (random-between 33 46)
                         (random-between 48 60)
                         62
                         (random-between 64 126)
                         ))) 'string)
         (randomxmltext (- min 1)))))

  (defrandom randomattribute ()
    (mv (randomxmltext (random-between 1 30)) (random-string)))

  (defrandom randomnode (maxdepth)
    (random-case
     (mv 'text nil (randomxmltext (random-between 1 30)))
     (mv
      (randomxmltext (random-between 1 30))
      (random-list-of (randomattribute) :size (random-between 0 10))
      (normalize-text (random-list-of
       (randomnode (- maxdepth 1)) :size (random-between 0 maxdepth))))))

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ; Property to test if xml is invertible!
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  (defproperty xml-readnode-serialize-dom-invertible-property :repeat 500
    (x :value (randomnode 5))
    (equal x (xml-readnode (xml-serialize-dom x))))

  (check-expect
   (xml-getattribute
    (xml-getnode
     (xml-readnode *Oface*)
     "slidell") "id")
   "porter")

  (check-expect
   (xml-gettext
    (xml-readnode *Oface*))
   "12 & 3")

  (check-expect
   (xml-escape (coerce "Bob & Jane's xml quote was, \"<hello />\"" 'list))
   "Bob &amp; Jane&apos;s xml quote was, &quot;&lt;hello /&gt;&quot;")

  (check-expect
   (xml-serialize-attributes
    (list
     (mv "name" "bob")
     (mv "age" "80")
     (mv "quote" "Bob & Jane's xml quote was, \"<hello />\"")))
   (concatenate 'string
                " name=\"bob\""
                " age=\"80\""
                " quote=\"Bob &amp; Jane&apos;s xml quote was, "
                "&quot;&lt;hello /&gt;&quot;\""))

  (check-expect
   (xml-serialize-dom (mv "Bob" nil nil))
   "<?xml version=\"1.0\"?><Bob/>")

  (check-expect
   (xml-serialize-dom
    (mv "Bob" nil
        (list
         (mv "Joe" nil nil)
         (mv 'text nil " hello ")
         (mv "Poop"
             (list
              (mv "type" "runny")
              (mv "where" "toilet")) nil))))
   (concatenate 'string
    "<?xml version=\"1.0\"?><Bob><Joe/> hello "
    "<Poop type=\"runny\" where=\"toilet\"/></Bob>"))

  (defconst *t1* (mv "bob" (list (mv "a" "b")) nil))
  (check-expect (xml-readnode (xml-serialize-dom *t1*)) *t1*)

  )

(link Test
      (Mbasiclex Mxmlminidom Txmlminidom))
```

```
(invoke Test)
```

```
;; 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
   Rbridge

   Main Linking and Invoking File
|#

(require "modules/Mbasiclex.lisp")
(require "modules/Mxmlminidom.lisp")
(require "modules/Mpsc.lisp")
(require "modules/Mboard.lisp")
(require "modules/Mio.lisp")
(require "modules/Mrankings.lisp")

(link Rbridge
      (Mbasiclex Mxmlminidom Mboard Mrankings Mpsc Mio))

(invoke Rbridge)

(set-state-ok t)

(main "testfiles/051115A" state)
```

```lisp
;; 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))))


  )
```

```lisp
;; 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
   iBoard

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

(defconst *menu*
  (concatenate 'string
               "<table  cellspacing=\"25\""
               "text-align: center; width: 100%;>"
               "<tr>"
               "<th><a href=\"boards-no-trav.htm\">Boards Without Travelers</a></th>"
               "<th><a href=\"boards-trav.htm\">Boards With Travelers</a></th>"
               "<th><a href=\"rnk.htm\">Rankings</a></th>"
               "<th><a href=\"psc.htm\">Personal Score Cards</a></th>"
               "</tr>"
               "</table>"
               "<hr/>"
               ))
(defconst *div-open-1* "<div class=\"")
(defconst *div-open-2* "\">")
(defconst *div-close* "</div>\n")
(defconst *br* "<br />\n")
(defconst *htmlhd1*
  (concatenate 'string
               "<html><head><title>"))
(defconst *htmlhd2*
  (concatenate 'string
               "</title><style>"
               "body {background-color: white; color: black;}"
               ".board {border-collapse: collapse; clear: both;"
               "position: relative; top: 0px; "
               "left: 0px; width: 100%; height: 30em;"
               "border: solid; border-width: 1px; "
               "margin: 1em 0em 0em; background-color: #d8d8d8;}"
               ".boardnum {position: absolute; left: 0em; top: 0em;}"
               ".N {position: absolute; height: 33%; width: 34%; "
               "left: 33%;top: 0%; background-color: white;}"
               ".S {position: absolute; height: 33%; width: 34%;"
               "left: 33%;top: 67%; background-color: white;}"
               ".E {position: absolute; height: 34%; width: 33%;"
               "left: 67%;top: 33%; background-color: white;}"
               ".W {position: absolute; height: 34%; width: 33%;"
               "left: 0%;top: 33%; background-color: white;}"
               ".dealer {text-align: center; font-weight: bold; color: blue;}"
               ".vulnerable {text-align: center; "
               "font-weight: bold; color: red;}"
               ".results {border-collapse:collapse; clear: left; "
               "width: 100%; margin: 2px 1px 0em;}"
               ".results tr td {font-size: 11pt; text-align: center; "
               "border: 1px solid black; margin: 1px;}"
               ".results tr th {font-size: 11pt; text-align: center; "
               "border: 1px solid black; margin: 1px;}"
               ".th {border: 1px; border-collapse:collapse; "
               "border-color: black; border-style: solid; border-width: 1px;} "
               ".main {margin: 0px auto; width: 45em;}"
               "</style></head><body>"
               "<div class=\"main\">"))
(defconst *htmltail* "</div></body></html>")
(defconst *tablehead*
  (concatenate 'string
               "<table class=\"results\"><tr>"
               "<th colspan=\"2\">Pairs</th>"
               "<th colspan=\"2\">Total Score</th>"
               "<th colspan=\"2\">Match Points</th>"
               "</tr><tr>"
               "<th>NS</th>"
               "<th>EW</th>"
               "<th>NS</th>"
               "<th>EW</th>"
               "<th>NS</th>"
               "<th>EW</th>"
               "</tr>"
               ))
(defconst *tabletail* "</table>\n")


(interface Iboard


  ;(include-book "basiclex")
  ;(include-book "xmlminidom")
  ;(include-book "stringutils")

  ; gethandcards (xmlnodes) → returns concatenated list of strings composed
  ; of the concatenation of suite symbol in HTML and card characters from
  ; xmlnodes where xmlnodes is a list of Suite xml nodes
  (sig serializedhandcards (xmlnodes))

  ; gethands (xmlnodes vulnerable dealer) → returns concatenated list of divs
  ; with class set to hand direction from xmlnodes, where xmlnodes is a list
  ; of xmlnode, of type hand, adds “vulnerable” and “dealer” divs inside the
  ; divs as necessary, and adds the cards to each hand
  (sig serializedhands (xmlnodes vulnerable dealer))

  ;grabs the results for one board separately from the board information
  ;without html serialization
  (sig getseparateresults (xmlnodes boardnum ns1 ew1))

  ;getseparateresults (xmlnodes) → serializes xmlnodes to a
  ;sequence of HTML tables corresponding to the seperate results
  ;for each player
  (sig getallseparateresults (xmlnodes))

  ;getresults (xmlnodes prefix postfix) → returns a string consisting of
  ; the concatenation of prefix, results table rows from each “Result” node,
  ; and postfix
  (sig serializedresults (xmlnodes))

  ;getboards (xmlnodes) → returns appended “board” class divs with their
  ; “results” tables from the xmlnode “Board” and “results” formatted to
  ; be rendered with the deal and results as required by description
  (sig serializedboards (xmlnodes trav-flag))

  (sig getgamestring (gamenode))

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;Contracts
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  (con serializedresults-nil=string-append-prefix-postfix-thm
       (implies (and (stringp prefix) (stringp postfix))
                (string-equal (serializedresults nil prefix postfix)
                              (string-append prefix postfix))))
  (con serializedresults-nil-returns-a-string-thm
       (implies (and (stringp prefix) (stringp postfix))
                (stringp (serializedresults nil prefix postfix))))
  )
```

```
;; 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
   iio

|#


(interface Iio

  ;main(bridgeXML state) Given a duplicate bridge XML file, extracts
  ;appropriate information to create four
  ;HTML pages that link together: boards, boards with travelers, rankings,
  ;and personal score card webpages.
  (sig main (bridgeXML state))

  ;boards-no-trav(bridgeXML state) Given a duplicate bridge XML file, gets
  ;the following information from each board listed in the XML file and
  ;creates an HTML page with this information formatted into tables:
  ;Board numbers, dealer, vulnerable, and hands for each direction
  (sig boards-no-trav (bridgeXML state))


  ;boards-trav(bridgeXML state) Given a duplicate bridge XML file, gets the
  ;same information as boards-no-trav as well as the travelers information
  ;for each board, which includes the total score and matchpoints for all
  ;pairs at that board. Creates HTML page with this information with the
  ;board information and travelers information in separate tables.
  (sig boards-trav (bridgeXML state))


  ;rankings(bridgeXML state) Given a duplicate bridge XML file, creates a
  ;rankings table in an HTML file including each pairs ranking and various
  ;other stats such as matchpoint and percentage score.
  (sig rankings (bridgeXML state))


  ;personal-score-cards(bridgeXML state) Given a duplicate bridge XML file,
  ;creates an HTML file containing personal score cards for each pair, which
  ;includes information about each match they played an against whom.
  (sig personal-score-cards (bridgeXML state))

  )
```

```
;; 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
   Ipsc

   Interface for the Personal Score Card Module mPSC
|#

(require "Ixmlminidom.lisp")

(defconst *psctableheadpre* "<table id=\"")
(defconst *psctableheadpost* "\" class=\"results\">")
(defconst *psctableheadcontents*
    (concatenate 'string
      "<tr>"
      "<th>Brd</th>"
      "<th colspan=\"2\">Versus</th>"
      "<th>Score</th>"
      "<th>Match Points</th>"
      "</tr>"))
(defconst *psctabletail* "</table>\n")

(interface Ipsc

  (include Ixmlminidom)


  ;Pulls the Personal Score Card data for all players, and put's them all
  ;into html table format
  ;XMLnodes format: Nodes format XXX update this to two args
  ;Output format: String, HTML formatted text comprising the score card
  ;    for one player pair
  (sig serializedPSC (gamenode boardnodes))

  ;Pulls the Name Strings for a given Pair ID
  ;PairID format: (String Direction, String SectionNumber)
  ;Data format: Nodes format
  ;Output format: (String String), Names of the two players
  (sig getNameForID (pairid data))

  ;;;
  ;;;
  (sig getBoardForPair(rbrds sections sectionlabel dir))

  ;Pulls the match results for a given Pair ID
  ;PairID format: (String Direction, String SectionNumber)
  ;Results format: ?
  ;XXX
  ;Output format: String, HTML formatted text comprising all the boards
  ;    for one player pair
  (sig getBoardsForPair (pairid sectionlabel results sections dir))

  ;;;
  ;;;
  (sig getAllPairs (results sections gamestring average top gamenode))
  #|
  (con getPSC-con1
       (implies ()
                ()))

  (con getNameForID-Gives-Two-Strings
       (implies (and (listp pairid)
                     (stringp (car pairid))
                     (stringp (cadr pairid))
```

```
                   (xml-isnodelist data))
              (let (output (getNameForID pairid data))
                (and (listp output)
                     (equal (len output) 2)
                     (stringp (car output))
                     (stringp (cadr output))))))

(con getBoardsForPair-con1
     (implies ()
              ())))
|#
)
```

```
;; 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")
(require "Ixmlminidom.lisp")

(defconst *rktablehead*
  (concatenate 'string
  "<table class=\"results\">"
  "<tr>"
  "<th rowspan=\"2\">Pair No.</th>"
  "<th rowspan=\"2\">Players</th>"
  "<th rowspan=\"2\">Strat / Flight</th>"
  "<th colspan=\"3\">Section Rank</th>"
  "<th colspan=\"3\">Overall Rank</th>"
  "<th rowspan=\"2\">Matchpoint Score</th>"
  "<th rowspan=\"2\">Percentage Score</th>"
  "<th rowspan=\"2\">Masterpoint Award</th>"
  "</tr>"
  "<tr>"
  "<th>A</th>"
  "<th>B</th>"
  "<th>C</th>"
  "<th>A</th>"
  "<th>B</th>"
  "<th>C</th>"
  "</tr>"
  ))
(defconst *rktabletail*
  "</table>")

(interface Irankings
  (include Ixmlminidom)
  (sig getmatchpointtotal (node))
  (sig sortcontestants (unsortedcontestants))
  (sig sortmvranks (ranks mvs))
  (sig serializedrankings (rankingnodes))
  (sig serializedrankingsheader (gamenodes))
  (sig getcontestants (section dir id sections))
  (sig getcontestantsnames (contestants))
  (sig getrankstring (ranktype contestants))

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;Contracts
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  (con getmatchpointtotal-returns-real
    (implies (xml-isnode mpnode)
             (real (getmatchpointtotal mpnode))))
  (con sortcontestants-returns-list
    (implies (xml-isnodelist rankingnodes)
             (listp (sortcontestants rankingnodes))))
  (con sortmvranks-returns-three
    (implies (xml-isnodelist rankingnodes)
             (eq 3 (len (sortmvrankings rankingnodes (mv nil nil nil))))))
  (con serializedrankings-delivers-string
    (implies (xml-isnodelist rankingnodes)
             (stringp (serializedrankings rankingnodes))))
  (con getcontestants-delivers-xml-node
    (implies (and (stringp section)
                  (stringp dir)
                  (stringp id)
                  (xml-isnodelist sections))
             (xml-isnode (getcontestants section dir id sections))))
  (con getcontestantsnames-delivers-string
    (implies (xml-isnode contestants)
             (stringp (getcontestantsnames contestants))))
  (con getrankstring-delivers-string
    (implies (and (xml-isnode contestants)
             (stringp ranktype)
             (or (string-equal ranktype "Section")
                 (string-equal ranktype "Overall")))
             (stringp (getrankstring ranktype contestants)))))
```

```
;; 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)))))
)
```

```
;; 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")
(require "../interfaces/Ibasiclex.lisp")

(module Mbasiclex
  (include-book "list-utilities" :dir :teachpacks)

  ; ds = delimiters to look for (list)
  ; xs = object to search in    (list)
  ; (split-at-delimiter ds xs) = (before at+)
  ;   where before = longest prefix of xs with no values from ds (list)
  ;         at+    = rest of xs (list)
  (defun split-at-delimiter (ds xs)
    (if (or (endp xs) (member-equal (car xs) ds))
        (list nil xs)
        (let* ((cdr-split (split-at-delimiter ds (cdr xs))))
          (list (cons (car xs) (car cdr-split))
                (cadr cdr-split)))))

  ; ps = list of signals to pass by (no constraints on signals)
  ; xs = list of signals (no constraints on signals)
  ; (span ps xs) = list of two elements
  ;  1 longest prefix of xs containing only signals from ps
  ;  2 rest of xs
  (defun span (ps xs)
    (if (or (endp xs) (not (member-equal (car xs) ps)))
        (list nil xs)
        (let* ((cdr-span (span ps (cdr xs))))
          (list (cons (car xs) (car cdr-span))
                (cadr cdr-span)))))

  ; ps = prefix to look for  (list)
  ; xs = object to search in (list)
  ; (splitoff-prefix ps xs) = (ps-matching ps-af-match xs-af-match)
  ;  where ps-matching = longest ps prefix matching xs prefix (list)
  ;        ps-af-match = rest of ps                           (list)
  ;        xs-af-match = non-matching suffix of xs            (list)
  ; Note: ps-af-match = nil means ps is a prefix of xs
  (defun splitoff-prefix (ps xs)
    (if (and (consp ps)
             (consp xs)
             (equal (car ps) (car xs)))
        (let* ((3way (splitoff-prefix (cdr ps) (cdr xs)))
               (ps-matching (car   3way))
               (ps-af-match (cadr  3way))
               (xs-af-match (caddr 3way)))
          (list (cons (car ps) ps-matching) ps-af-match xs-af-match))
        (list nil ps xs)))

  ; ps = prefix to look for  (list of standard, upper-case characters)
  ; xs = object to search in (list)
  ; (splitoff-prefix-upr ps xs) = (ps-matching ps-af-match xs-af-match)
  ;  where ps-matching = longest ps prefix matching xs prefix (list)
  ;        ps-af-match = rest of ps                           (list)
  ;        xs-af-match = non-matching suffix of xs            (list)
  ; Notes: 1. search is not sensitive to case of letters in xs
  ;        2. ps-af-match = nil means ps is a prefix of xs
  ; Implementation issue: combining general and case-insensitive search
  ;   in one function simplifies maintenance, but complicates
  ;   formulation of software properties and their proofs
  (defun splitoff-prefix-upr (ps xs)
    (if (and (consp ps)
             (consp xs)
             (equal (car ps)
                    (let* ((x (car xs)))
                      (if (standard-char-p x)
                          (char-upcase x)
                          x))))
        (let* ((3way (splitoff-prefix-upr (cdr ps) (cdr xs)))
               (ps-matching (car   3way))
               (ps-af-match (cadr  3way))
               (xs-af-match (caddr 3way)))
          (list (cons (car ps) ps-matching) ps-af-match xs-af-match))
        (list nil ps xs)))

  ; tok-str = characters to look for (string, standard characters)
  ; chrs    = object to search in
  (defun splitoff-prefix-chr (tok-str xs)
    (splitoff-prefix-upr (str->chrs(string-upcase tok-str)) xs))

  ; tok = object to search for (list)
  ; xs  = object of search     (list)
  ; (split-on-token-gen tok xs) = (prefix match suffix)
  ;   where
  ;   prefix = elems of xs before 1st sublist matching tok (list)
  ;          = xs  if no match
  ;   match  = tok if match                                (list)
  ;          = nil if no match
  ;   suffix = elems of xs after 1st sublist matching tok  (list)
  ;          = nil if no match
  (defun split-on-token-gen (tok xs)
    (if (endp xs)
        (list nil nil nil)
        (let* ((splitoff-3way (splitoff-prefix tok xs))
               (matching?     (null (cadr splitoff-3way)))
               (aftr-tok      (caddr splitoff-3way)))
          (if matching?
              (list nil tok aftr-tok)
              (let* ((3way (split-on-token-gen tok (cdr xs)))
                     (bfor-tok (car   3way))
                     (at-tok   (cadr  3way))
                     (aftr-tok (caddr 3way)))
                (list (cons (car xs) bfor-tok)
                      at-tok
                      aftr-tok))))))

  ; tok = object to search for (list of upper-case standard characters)
  ; xs  = object to search in  (list containing no non-standard chars)
  ; Note: matching is not case-sensitive
  ; (split-on-token-chr tok xs) = (prefix match suffix)
  ;   where
  ;   prefix = elems of xs before 1st sublist matching tok (list)
  ;          = xs  if no match
  ;   match  = tok if match                                (list)
  ;          = nil if no match
  ;   suffix = elems of xs after 1st sublist matching tok  (list)
  ;          = nil if no match
  (defun split-on-token-chr (tok xs)
    (if (endp xs)
        (list nil nil nil)
        (let* ((splitoff-3way (splitoff-prefix-upr tok xs))
               (matching?     (null (cadr splitoff-3way)))
               (aftr-tok      (caddr splitoff-3way)))
          (if matching?
              (list nil tok aftr-tok)
              (let* ((3way (split-on-token-chr tok (cdr xs)))
                     (bfor-tok (car   3way))
                     (at-tok   (cadr  3way))
                     (aftr-tok (caddr 3way)))
                (list (cons (car xs) bfor-tok)
                      at-tok
                      aftr-tok))))))
```

```
; tok = object to search for (string or list)
; xs  = object to search in (list, no non-standard chars if tok is string)
; Note: search is not case-sensitive if tok is a string
; Warning! Neither tok nor xs may contain non-standard characters
;          if tok is a string
; Implementation issue: combining general and case-insensitive search
;   in one function simplifies maintenance, but complicates
;   formulation of software properties and their proofs
(defun split-on-token (tok xs)
  (if (stringp tok)
      (split-on-token-chr (str->chrs(string-upcase tok)) xs)
      (split-on-token-gen tok xs)))

(export Ibasiclex))
```

```
;; 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
   mBoard

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

(require "../interfaces/Iboard.lisp")
(require "../interfaces/Ibasiclex.lisp")
(require "../interfaces/Ixmlminidom.lisp")
(module Mboard

  (import Ibasiclex)
  (import Ixmlminidom)

  (include-book "io-utilities" :dir :teachpacks)
  (include-book "list-utilities" :dir :teachpacks)

  ; serializedhandcards (xmlnodes) → returns concatenated list of strings
  ; composed of the concatenation of suit symbol in HTML and card
  ; characters from xmlnodes where xmlnodes is a list of Suit xml nodes
  (defun serializedhandcards (xmlnodes)
    (if (null xmlnodes)
        ""
        (let* ((suit (car xmlnodes))
               (rest (cdr xmlnodes))
               (suitsymbol (xml-getattribute suit "symbol"))
               (suithtml (if (string-equal suitsymbol "S")
                             "&spades;"
                             (if (string-equal suitsymbol "C")
                                 "&clubs;"
                                 (if (string-equal suitsymbol "D")
                                     "&diams;"
                                     "&hearts;"))))
               (cards (xml-gettext suit)))
          (concatenate 'string
                       suithtml
                       cards *br*
                       (serializedhandcards rest)))))

  ; serializedhands (xmlnodes vulnerable dealer) → returns concatenated
  ; list of divs with class set to hand direction from xmlnodes, where
  ; xmlnodes is a list of xmlnode, of type hand, adds “vulnerable” and
  ; “dealer” divs inside the divs as necessary, and adds the cards to each
  ; hand
  (defun serializedhands (xmlnodes vulnerable dealer)
    (if (null xmlnodes)
        ""
        (let* ((hand (car xmlnodes))
               (rest (cdr xmlnodes))
               (direction (xml-getattribute hand "direction"))
               (suits (xml-getnodes hand "Suit"))
               (dealerhtml (if (string-equal dealer direction)
                               (concatenate 'string
                                            *div-open-1*
                                            "dealer"
                                            *div-open-2*
                                            "Dealer"
                                            *div-close*)
                               ""))
               (vulnerablehtml (if (or (string-equal vulnerable "Both")
                                       (and (string-equal vulnerable "NS")
                                            (or (string-equal direction
                                                              "N")
                                                (string-equal direction
                                                              "S")))
                                       (and (string-equal vulnerable "EW")
                                            (or (string-equal direction
                                                              "E")
                                                (string-equal direction
                                                              "W"))))
                                   (concatenate 'string
                                                *div-open-1*
                                                "vulnerable"
                                                *div-open-2*
                                                "Vulnerable"
                                                *div-close*)
                                   "")))
          ; For your reading pleasure, this is the let* body
          (concatenate 'string
                       *div-open-1*
                       direction
                       *div-open-2*
                       dealerhtml
                       vulnerablehtml
                       (serializedhandcards suits)
                       *div-close*
                       (serializedhands rest vulnerable dealer)))))

  ; getseparateresults (xmlnodes boardnum ns1 ew1) → grabs the results for
  ; one board separately from the board information without html
  ; serialization where xmlnodes is the list of results for a board,
  ; boardnum is the boardnum,  ns1 and ew1 are the initial lists. It
  ; returns a structure like:
  ; (mv ns ew)
  ; where
  ;  ns = (list
  ;         (cons (mv pairns section)
  ;               (list
  ;                 (mv boardnum pairew totalscore pointsns))
  ;                 ...)
  ;         ...)
  ;  ew = (list
  ;         (cons (mv pairew section)
  ;               (list
  ;                 (mv boardnum pairns totalscore pointsns))
  ;                 ...)
  ;         ...)
  (defun getseparateresults (xmlnodes boardnum ns1 ew1)
    (if (null xmlnodes)
        (mv ns1 ew1)
        (let* ((result (car xmlnodes))
               (rest (cdr xmlnodes))
               (section (xml-getattribute result "SectionLabel"))
               (pairns (xml-getattribute result "PairID-NS"))
               (pairew (xml-getattribute result "PairID-EW"))
               (totalscorenode (xml-getnode result "TotalScore"))
               (totaldir (xml-getattribute totalscorenode "direction"))
               (totalscore (xml-gettext totalscorenode))
               (pointsns (xml-gettext (xml-getnode result
                                                   "MatchpointsNS")))
               (pointsew (xml-gettext (xml-getnode result
                                                   "MatchpointsEW"))))
          (mv-let (ns ew)
                  (getseparateresults rest boardnum ns1 ew1)
            (let ((nskv (assoc-equal (mv pairns section) ns))
                  (ewkv (assoc-equal (mv pairew section) ew)))
                  (mv (cons (cons (mv pairns section)
                                  (cons (mv boardnum
                                            pairew
```

```
                                              (if (string-equal totaldir
                                                                "N-S")
                                                  totalscore
                                                 (string-append "-"
                                                                totalscore))
                                              pointsns)
                                          (cdr nskv)))
                              (remove-equal nskv ns))
                        (cons (cons (mv pairew section)
                                    (cons (mv boardnum
                                              pairns
                                              (if (string-equal totaldir
                                                                "E-W")
                                                  totalscore
                                                  (string-append "-"
                                                                 totalscore))
                                              pointsew)
                                           (cdr ewkv)))
                              (remove-equal ewkv ew))))))))

; getallseparateresults (boardnodes) → converts boardnodes, a list of
; Board nodes, to a sequence like:
; (mv ns ew)
; where
;  ns = (list
;         (cons (mv pairns section)
;               (list
;                 (mv boardnum pairew totalscore pointsns))
;                 ...)
;         ...)
;  ew = (list
;         (cons (mv pairew section)
;               (list
;                 (mv boardnum pairns totalscore pointsns))
;                 ...)
;         ...)
(defun getallseparateresults (boardnodes)
  (if (null boardnodes)
      (mv nil nil)
      (let* ((board (car boardnodes))
             (rest (cdr boardnodes))
             (boardnum (xml-gettext (xml-getnode board "BoardNo")))
             (results (xml-getnodes board "Result")))
        (mv-let (ns ew)
                (getallseparateresults rest)
          (getseparateresults results boardnum ns ew)))))

; serializedresults (xmlnodes) → returns a string consisting of
; the concatenation of results table rows from each “Result” node
(defun serializedresults (xmlnodes)
    (if (null xmlnodes)
        ""
        (let* ((result (car xmlnodes))
               (rest (cdr xmlnodes))
               (section (xml-getattribute result "SectionLabel"))
               (pairns (xml-getattribute result "PairID-NS"))
               (pairew (xml-getattribute result "PairID-EW"))
               (totalscorenode (xml-getnode result "TotalScore"))
               (totaldir (xml-getattribute totalscorenode "direction"))
               (totalscore (xml-gettext totalscorenode))
               (pointsns (xml-gettext (xml-getnode result
                                                   "MatchpointsNS")))
               (pointsew (xml-gettext (xml-getnode result
                                                   "MatchpointsEW"))))
          (concatenate 'string
                       "<tr>"
                       "<td><a href=\"psc.htm#N-S" pairns section "\">"
```

```
                         section pairns "</a></td>"
                         "<td><a href=\"psc.htm#E-W" pairew section "\">"
                         section pairew "</td>"
                         "<td>" (if (string-equal totaldir "N-S")
                                    totalscore
                                    "&nbsp;")
                         "</td>"
                         "<td>" (if (string-equal totaldir "E-W")
                                    totalscore
                                    "&nbsp;")
                         "</td>"
                         "<td>" pointsns "</td>"
                         "<td>" pointsew "</td>"
                         "</tr>"
                         (serializedresults rest)))))

; serializedboards (xmlnodes) → returns appended “board” class divs with
; their “results” tables from the xmlnode “Board” and “results” formatted
; to be rendered with the deal and results as required by description
(defun serializedboards (xmlnodes trav-flag)
  (if (null xmlnodes)
      ""
      (let* ((board (car xmlnodes))
             (rest (cdr xmlnodes))
             (vulnerable (xml-gettext (xml-getnode board "Vulnerable")))
             (dealer (xml-gettext (xml-getnode board "Dealer")))
             (boardnum (xml-gettext (xml-getnode board "BoardNo")))
             (hands (xml-getnodes (xml-getnode board "Deal") "Hand"))
             (results (xml-getnodes board "Result")))
        (concatenate 'string
                     *div-open-1*
                     "board\" id=\"" boardnum
                     *div-open-2*
                     *div-open-1*
                     "boardnum"
                     *div-open-2*
                     "Board: " boardnum "<br/><a href=\""
                     (if (equal trav-flag 1)
                         (concatenate
                          'string
                          "boards-no-trav.htm#"
                          boardnum "\"><small>(hide travelers)</small></a>")
                         (concatenate
                          'string
                          "boards-trav.htm#"
                          boardnum "\"><small>(show travelers)</small></a>")
                         )
                     *div-close*
                     (serializedhands hands vulnerable dealer)
                     *div-close*
                     (if (equal trav-flag 1)
                         (concatenate 'string
                                      *tablehead*
                                      (serializedresults results)
                                      *tabletail*)
                         "")
                     (serializedboards rest trav-flag)))))

(defun getgamestring (gamenode)
  (let* ((clubgame (xml-gettext (xml-getnode gamenode "ClubGame")))
         (eventname (xml-gettext (xml-getnode (xml-getnode gamenode "Event")
                                              "EventName")))
         (session (xml-gettext (xml-getnode gamenode "Session")))
         (date (xml-gettext (xml-getnode gamenode "Date"))))
    (concatenate 'string
                 clubgame " " eventname ", " session ", " date)))
  (export Iboard))
```

```
;; 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")
(require "../interfaces/Iio.lisp")
(require "../interfaces/Iboard.lisp")
(require "../interfaces/Ipsc.lisp")
(require "../interfaces/Irankings.lisp")
(require "../interfaces/Ixmlminidom.lisp")

(module Mio
  (import Ixmlminidom)
  (import Iboard)
  (import Irankings)
  (import Ipsc)

  (include-book "io-utilities" :dir :teachpacks)

  ; boards-no-trav (bntXML state) Given a duplicate bridge XML file, gets
  ; the following information from each board listed in the XML file and
  ; creates an HTML page with this information formatted into tables:
  ; Board numbers, dealer, vulnerable, and hands for each direction
  (defun boards-no-trav (bntXML state)
    (mv-let (status state)
            (string-list->file (string-append "boards-no-trav" ".htm")
                               (list *htmlhd1*
                                     "Boards without Travelers"
                                     *htmlhd2*
                                     *menu*
                                     (serializedboards
                                       (xml-getnodes (xml-getnode
                                                       (xml-getnode bntXML
                                                         "Game")
                                                       "HandRecords")
                                                     "Board")
                                       0)
                                     *htmltail*)
                               state)
      (if (null status)
          (mv 'ok state)
          (mv 'error state))))

  ; boards-trav (btXML state) Given a duplicate bridge XML file, gets the
  ; same information as boards-no-trav as well as the travelers information
  ; for each board, which includes the total score and matchpoints for all
  ; pairs at that board. Creates HTML page with this information with the
  ; board information and travelers information in separate tables.
  (defun boards-trav (btXML state)
    (let* ((boardnode (xml-getnodes (xml-getnode (xml-getnode btXML "Game")
                                                 "HandRecords")
                                    "Board")))
      (mv-let (status state)
              (string-list->file (string-append "boards-trav" ".htm")
                                 (list *htmlhd1*
                                       "Boards with Travelers"
                                       *htmlhd2*
                                       *menu*
                                       (serializedboards boardnode 1)
                                       *htmltail*)
                                 state)
        (if (null status)
            (mv 'ok state)
            (mv 'error state)))))

  ; rankings (rnkXML state) Given a duplicate bridge XML file, creates a
  ; rankings table in an HTML file including each pairs ranking and various
  ; other stats such as matchpoint and percentage score.
  (defun rankings (rnkXML state)
    (let* ((games (xml-getnode rnkXML "Game"))
           (sections (xml-getnodes games "Section")))
      (mv-let (status state)
              (string-list->file (string-append "rnk" ".htm")
                                 (list *htmlhd1*
                                       "Rankings"
                                       *htmlhd2*
                                       *menu*
                                       (serializedRankingsHeader games)
                                       (serializedRankings sections)
                                       *htmltail*)
                                 state)
        (if (null status)
          (mv 'ok state)
          (mv 'error state)))))


  ; personal-score-cards (pscXML state) Given a duplicate bridge XML file,
  ; creates an HTML file containing personal score cards for each pair,
  ; which includes information about each match they played an against
  ; whom.
  (defun personal-score-cards (pscXML state)
    (let* ((gamenode (xml-getnode pscXML "Game"))
           (boardnodes (xml-getnodes (xml-getnode gamenode "HandRecords")
                                     "Board")))
      (mv-let (status state)
              (string-list->file (string-append "psc" ".htm")
                                 (list *htmlhd1*
                                       "Personal Score Cards"
                                       *htmlhd2*
                                       *menu*
                                       (serializedPSC gamenode boardnodes)
                                       *htmltail*)
                                 state)
        (if (null status)
          (mv 'ok state)
          (mv 'error state)))))

  ; main (bridgeXML state) Given a duplicate bridge XML file, extracts
  ; appropriate information to create four HTML pages that link together:
  ; boards, boards with travelers, rankings, and personal score card
  ; webpages.
  (defun main (bridgeXML state)
    (mv-let (contents status state)
          (file->string (string-append bridgeXML ".xml") state)
          (if (null status)
              (let* ((xml (xml-readnode contents))
                     (bt (boards-trav xml state))
                     (bnt (boards-no-trav xml (cadr bt)))
                     (rnk (rankings xml (cadr bnt)))
                     (psc (personal-score-cards xml (cadr rnk))))
                (mv 'ok (cadr psc)))
              (mv 'error state))))

  (export Iio))
```

```
;; 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
   Mpsc

   Personal Score Card Module
|#
(require "../interfaces/Ipsc.lisp")
(require "../interfaces/Iboard.lisp")
(require "../interfaces/Irankings.lisp")
(require "../interfaces/Ixmlminidom.lisp")
(require "../interfaces/Ibasiclex.lisp")

(module Mpsc
  (import Iboard)
  (import Ixmlminidom)
  (import Irankings)
  (import Ibasiclex)
  (include-book "io-utilities" :dir :teachpacks)
  (include-book "list-utilities" :dir :teachpacks)

  ;Pulls the Name Strings for a given Pair ID
  ;PairID format: (String Direction, String SectionNumber)
  ;Data format: Nodes format
  ;Output format: (String String), Names of the two players
  (defun getNameForID (pairid data) nil)

  ;;;
  ;;;
  (defun getBoardForPair (rbrds sections sectionlabel dir)
    (if (null rbrds)
        ""
        (let* ((sbrd (car rbrds))
               (rest (cdr rbrds))
               (id (second sbrd))
               (contestants (getcontestants sectionlabel dir id sections))
               (players (getcontestantsnames contestants)))
          (concatenate 'string
                       "<tr>"
                       "<td><a href=\"boards-trav.htm#" (first sbrd) "\">"
                            (first sbrd)    "</a></td>"    ; boardnum
                       "<td><a href=\"psc.htm#"
                       dir id sectionlabel "\">"
                       sectionlabel id "</a></td>"    ; vs. info
                       "<td>" players         "</td>"    ; names
                       "<td>" (third sbrd)    "</td>"    ; score
                       "<td>" (fourth sbrd)   "</td>"    ; matchpoints
                       "</tr>"
                       (getBoardForPair rest sections sectionlabel dir)))))

  ;Pulls the match results for a given Pair ID
  ;PairID format: (String Direction, String SectionNumber)
  ;Results format: ?
  ;Output format: String, HTML formatted text comprising all the boards
  ;    for one player pair
  (defun getBoardsForPair (pairid sectionlabel results sections dir)
    (let* ((bforp (assoc-equal (mv pairid sectionlabel) results)))
      (getBoardForPair (cdr bforp) sections sectionlabel dir)))

  ;;;
  ;;;
  (defun getAllPairs (results sections gamestring average top gamenode)
    (let* ((ns (car results))
           (ew (cadr results))
```

```
         (nextns (car ns))
         (keyns (car nextns))
         (nextew (car ew))
         (keyew (car nextew))
         (restns (cdr ns))
         (restew (cdr ew)))
    (if (null nextew)
        ""
        (if (null nextns)
            (let* ((pairid (car keyew))
                   (sectionlabel (cadr keyew)))
              (concatenate 'string
                           *psctableheadpre*
                           "E-W" (car keyew) (cadr keyew)
                           *psctableheadpost*
                           (pscheader sectionlabel "E-W" pairid sections
                                      average top)
                           *psctableheadcontents*
                           ;get info from gamenode
                           (getBoardsForPair pairid
                                             sectionlabel
                                             ew           ; results
                                             sections
                                             "N-S") ; The *opponents'* dir.
                           (pscfooter gamestring)
                           *psctabletail*
                           (getAllPairs (mv ns restew)
                                        sections
                                        gamestring
                                        average
                                        top
                                        gamenode)))
            (let* ((pairid (car keyns))
                   (sectionlabel (cadr keyns)))
              (concatenate 'string
                           *psctableheadpre*
                           "N-S" (car keyns) (cadr keyns)
                           *psctableheadpost*
                           (pscheader sectionlabel "N-S" pairid sections
                                      average top)
                           *psctableheadcontents*
                           (getBoardsForPair pairid
                                             sectionlabel
                                             ns           ; results
                                             sections
                                             "E-W")
                           (pscfooter gamestring)
                           *psctabletail*
                           (getAllPairs (mv restns ew)
                                        sections
                                        gamestring
                                        average
                                        top
                                        gamenode)))))))

(defun pscheader (sectionlabel dir id sections average top)
  (let* ((contestants (getcontestants sectionlabel dir id sections)))
    (concatenate 'string
                 "<tr><td colspan=\"5\">Scorecard for <b>Pair "
                 sectionlabel
                 id
                 " "
                 dir
                 " (strat "
                 (xml-getattribute contestants "Strat")
                 ")</b><br>"
                    (getcontestantsnames contestants)
                 "<br>"
                 " Score: "
                    (xml-gettext (xml-getnode contestants "MatchpointTotal"))
                    " ("
                    (xml-gettext (xml-getnode contestants "Percentage"))
                    "%) "
                 (let* ((award (xml-getnode contestants "Award")))
                   (if award
                       (concatenate 'string
                                    "MP: "
                                    (xml-getattribute award "TypeOfAward")
                                    " "
                                    (xml-gettext award)
                                    " ("
                                    (xml-getattribute award "AwardCategory")
                                    (xml-getattribute award "AwardStrat")
                                    ")")
                       ""))
                 " Ave: "
                 average
                 " Top: "
                 top
                 "<br>"
                 (getrankstring "Section" contestants)
                 " "
                 (getrankstring "Overall" contestants)
                 "</td></tr>")))

(defun pscfooter (gamestring)
  (concatenate 'string "<tr><td colspan=\"5\">" gamestring "</td></tr>"))

;Pulls the Personal Score Card data for all players, and put's them all
;into html table format
;XMLnodes format: Nodes format
;Output format: String, HTML formatted text comprising the score card
;    for one player pair
(defun serializedPSC (gamenode boardnodes)
  (let* ((sections (xml-bfsfindnodes (list gamenode) "Section"))
         (results (getAllSeparateResults boardnodes))
         (gamestring (getgamestring gamenode))
         (movement (xml-getnode gamenode "Movement"))
         (average (xml-gettext (xml-getnode movement "Average")))
         (top (xml-gettext (xml-getnode movement "Top"))))
    (getAllPairs results sections gamestring average top gamenode)))

(export Ipsc))
```

```lisp
;; 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")
(require "../interfaces/Ixmlminidom.lisp")
(require "../interfaces/Irankings.lisp")

(module Mrankings
  (import Ixmlminidom)

  (include-book "io-utilities" :dir :teachpacks)

  ; Gets the matchpointtotal for a contestant as a rational
  (defun getmatchpointtotal (node)
    (str->rat (xml-gettext (xml-getnode node "MatchPointTotal"))))

  ; Insert function adapted from *site*
  (defun insert-sorted (a lst)
    (if (or (endp lst)
            (>= (getmatchpointtotal a)
                (getmatchpointtotal (car lst))))
        (cons a lst)
        (cons (car lst) (insert-sorted a (cdr lst)))))

  ; This function sorts the contestants by MatchPointTotal
  (defun sortcontestants (unsortedcontestants)
    (if (null unsortedcontestants)
        nil
        (let* ((contestant (car unsortedcontestants))
               (rest (cdr unsortedcontestants)))
          (insert-sorted contestant (sortcontestants rest)))))

  (defun sortmvranks (ranks mvs)
    (if (null ranks)
        mvs
        (let ((rank (car ranks))
              (rest (cdr ranks)))
          (mv-let (a b c)
                  mvs
                  (sortmvranks
                   rest
                   (mv
                    (if (equal (xml-getattribute rank "Strat") "A")
                        (xml-gettext rank)
                        a)
                    (if (equal (xml-getattribute rank "Strat") "B")
                        (xml-gettext rank)
                        b)
                    (if (equal (xml-getattribute rank "Strat") "C")
                        (xml-gettext rank)
                        c)))))))

  (defun serializedcontestants (contestants section direction)
    (if (null contestants)
        ""
        (let* ((contestant (car contestants))
               (rest (cdr contestants))
               (pairno (xml-getattribute contestant "ID"))
               (players (xml-getnodes contestant "Player"))
               (player1 (xml-gettext (xml-getnode (car players) "Name")))
               (player2 (xml-gettext (xml-getnode (cadr players) "Name")))
               (strat (xml-getattribute contestant "Strat"))
               (sectionranks (xml-getnodes contestant "SectionRank"))
               (overallranks (xml-getnodes contestant "OverallRank"))
               (matchpoint (xml-gettext
                            (xml-getnode contestant "MatchpointTotal")))
               (percentage (xml-gettext
                            (xml-getnode contestant "Percentage")))
               (mpvalue (xml-gettext (xml-getnode contestant "Award")))
               (masterpoint (if (equal mpvalue "")
                                "&nbsp;"
                                mpvalue)))
          (concatenate 'string
                       "<tr>"
                       "<td><a href=\"psc.htm#" direction pairno section
                       "\">" pairno "</a></td>"
                       "<td>" player1 "<br />" player2 "</td>"
                       "<td>" strat "</td>"
                       (mv-let (a b c)
                               (sortmvranks sectionranks
                                            (mv "&nbsp;" "&nbsp;"
                                                "&nbsp;" ))
                               (concatenate 'string
                                            "<td>" a "</td>"
                                            "<td>" b "</td>"
                                            "<td>" c "</td>"))
                       (mv-let (a b c)
                               (sortmvranks overallranks
                                            (mv "&nbsp;" "&nbsp;" "&nbsp;" ))
                               (concatenate 'string
                                            "<td>" a "</td>"
                                            "<td>" b "</td>"
                                            "<td>" c "</td>"))
                       "<td>" matchpoint "</td>"
                       "<td>" percentage "</td>"
                       "<td>" masterpoint "</td>"
                       "</tr>"
                       (serializedcontestants rest section direction)))))

  ; XXX rankingnodes is a bad misnomer.  rankingnodes should definitely
  ; *not* be a list of Rankings nodes.  At a minimum, we need the Section
  ; nodes, too.
  (defun serializedrankings (rankingnodes)
    (if (null rankingnodes)
        ""
        (let* ((ranking (car rankingnodes))
               (rest (cdr rankingnodes))
               (section (xml-getattribute
                        ranking
                        "SectionLabel"))
               (direction (xml-getattribute
                        ranking
                        "Direction"))
               (unsortedcontestants (xml-getnodes
                                     (xml-getnode ranking "Rankings")
                                     "Contestants"))
               (contestanthtml (serializedcontestants
                                (sortcontestants unsortedcontestants)
                                section direction)))
          (concatenate 'string
                       *rktablehead*
                       "<p align=\"CENTER\">"
                       "Section "
                       section
                       "  --  "
                       direction
                       "</p>"
                       contestanthtml
                       *rktabletail*
                       (serializedrankings rest)))))

  ;Pulls header information from the game node
  (defun serializedRankingsHeader (gamenodes)
    (let* ((Date (xml-gettext (xml-getnode gamenodes "Date")))
           (Club (xml-gettext (xml-getnode gamenodes "ClubGame")))
           (Event (xml-gettext (xml-getnode
                                (xml-getnode gamenodes "Event")
                                "EventName"))))
      (concatenate 'string
                   "<h4 align=\"CENTER\">"
                   "Rankings - "
                   Date
                   "<br />"
                   Club
                   " - "
                   Event
                   "</h4>")))
  ; sectionnodes should be a list of Section nodes
  (defun findspecificsection (sectionnodes label dir)
    (if sectionnodes
        ; linear search!
        (let* ((current (car sectionnodes)))
          (if (and (equal (xml-getattribute current "SectionLabel")
                          label)
                   (equal (xml-getattribute current "Direction")
                          dir))
              current
              (findspecificsection (cdr sectionnodes) label dir)))
        nil))

  ; nodes should be a list of Contestants nodes
  (defun findspecificcontestants (nodes id)
    (if nodes
        (let* ((current (car nodes)))
          (if (equal (xml-getattribute current "ID") id)
              current
              (findspecificcontestants (cdr nodes) id)))
        nil))

  (defun getcontestants (sectionlabel dir id sections)
    (let* ((section (findspecificsection sections sectionlabel dir)))
      (findspecificcontestants
                         (xml-bfsfindnodes (list section) "Contestants")
                         id)))

  ; For the two players in a Contestants element, delivers a string in the
  ; form "Alice - Bob".
  ; Contestants is the Contestants node
  (defun getcontestantsnames (contestants)
    (let* ((players (xml-getnodes contestants "Player")))
      (concatenate 'string (xml-gettext (car players))
                   " - "
                   (xml-gettext (cadr players)))))

  (defun getranks (ranksoftype)
    (if (< 0 (len ranksoftype))
      (let* ((rank (car ranksoftype)))
        (concatenate 'string
                     (xml-gettext rank)
                     "("
                     (xml-getattribute rank "Strat")
                     ")"
                     (getranks (cdr ranksoftype))))

      ""))

  (defun getrankstring (ranktype contestants)
    (let* ((ranks (getranks (xml-getnodes contestants
                                          (concatenate 'string
                                                       ranktype
                                                       "Rank")))))
      (if (< 0 (length ranks))
         (concatenate 'string ranktype " Rank: " ranks)
         "")))

  (export Irankings))
```

```lisp
;; 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
   Mxmlminidom

   Software that creates a document object model from XML input.
|#
(in-package "ACL2")

(require "../interfaces/Ixmlminidom.lisp")
(require "../interfaces/Ibasiclex.lisp")

;(defconst *whitespace* '(#\Space #\Return #\Newline #\Linefeed #\Tab ))
;(defconst *endtagname* (cons #\> (cons #\/ *whitespace*)))
;(defconst *endattrname* (cons #\= *whitespace*))

(module Mxmlminidom
  (import Ibasiclex)

  (include-book "list-utilities" :dir :teachpacks)

  ;These functions just wrap the basiclex functions of the same name
  ;returning mv's instead of lists because they are a guaranteed size
  ;and I like using mv-let.
  (defun split-on-token-mv (tok xs)
    (let* ((res (split-on-token tok xs)))
      (mv (car res) (cadr res) (caddr res))))
  ;See above
  (defun span-mv (ps xs)
    (let* ((res (span ps xs)))
      (mv (car res) (cadr res))))
  ;See above
  (defun split-at-delimiter-mv (ds xs)
    (let* ((res (split-at-delimiter ds xs)))
      (mv (car res) (cadr res))))
  ;See above
  (defun splitoff-prefix-mv (ps xs)
    (let* ((res (splitoff-prefix ps xs)))
      (mv (car res) (cadr res) (caddr res))))

  ;xml-escape (unescapedchars) → returns string with bad chars replaced
  ;    with entities
  (defun xml-escape (unescapedchars)
    (if (null unescapedchars)
        ""
        (let ((char (car unescapedchars))
              (rest (cdr unescapedchars)))
          (string-append
           (cond
            ((equal char #\&) "&amp;")
            ((equal char #\<) "&lt;")
            ((equal char #\>) "&gt;")
            ((equal char #\') "&apos;")
            ((equal char #\") "&quot;")
            (t (chrs->str (list char))))
           (xml-escape rest)
           ))))

  ;xml-serialize-attributes (attributes) → Returns a string that is
  ;    xml that represents the passed in attribute list.
  (defun xml-serialize-attributes (attributes)
    (if (null attributes)
        ""
        (let ((attribute (car attributes))
              (rest (cdr attributes)))
          (concatenate
           'string
           " "
           (xml-escape (str->chrs (car attribute)))
           "=\""
           (xml-escape (str->chrs (cadr attribute)))
           "\""
           (xml-serialize-attributes rest)))))


;xml-serizlize-nodes (xmlnodes) → Returns a string containing xml
;    nodes that represents the node list, xmlnodes.
(defun xml-serialize-nodes (xmlnodes)
  (if (null xmlnodes)
      ""
      (let ((node (car xmlnodes))
            (rest (cdr xmlnodes)))
        (mv-let (nodename attributes children)
                node
                (string-append
                 (if (equal nodename 'text)
                     (xml-escape (str->chrs children))
                     (concatenate 'string
                      "<"
                      (xml-escape (str->chrs nodename))
                      (xml-serialize-attributes attributes)
                      (if (null children)
                          "/>"
                          (concatenate 'string
                           ">"
                           (xml-serialize-nodes children)
                           "</"
                           (xml-escape (str->chrs nodename))
                           ">"))))
                 (xml-serialize-nodes rest))))))

;xml-serizlize-dom (xmlnode) → Returns a string containing an xml
;    document that represents the dom passed in through xmlnode.
(defun xml-serialize-dom (xmlnode)
      (string-append
       "<?xml version=\"1.0\"?>"
       (xml-serialize-nodes (list xmlnode)))
  )

;xml-unescape (escapedchars) → string with entities replaced
(defun xml-unescape (escapedchars)
  (if (and
       (consp escapedchars)
       (standard-char-listp escapedchars))
      (let* ((sot (split-on-token '(#\& ) escapedchars))
             (pre (car sot))
             (post (caddr sot))
             (sot1 (split-on-token '(#\; ) post))
             (thechar (car sot1))
             (theend (caddr sot1))
             (thecharstr (chrs->str thechar)))
            (concatenate 'string
             (chrs->str pre)
              (if (string-equal "amp" thecharstr)
                  "&"
                  (if (string-equal "lt" thecharstr)
                      "<"
                      (if (string-equal "gt" thecharstr)
                          ">"
                          (if (string-equal "quot" thecharstr)
                              "\""
```

```
                                (if (string-equal "apos" thecharstr)
                                    "'"
                                    "")))))
                (xml-unescape theend)))
        ""))

;xml-readnodeproperties (xmlchars) →
; returns (mv attributes remainingxmlstring)
(defun xml-readnodeproperties (xmlchars)
  (mv-let (ws1 att1)
          (span-mv *whitespace* xmlchars)
          (if (null att1)
              (mv nil nil)
              (if (member (car att1) *endtagname*)
                  (mv nil att1)
                  (mv-let
                   (propname r1)
                   (split-at-delimiter-mv *endattrname* att1)
                   (mv-let
                    (r2 r3 propvstart)
                    (split-on-token-mv '(#\" ) r1)
                    (mv-let
                     (propvalue r4 rest)
                     (split-on-token-mv '(#\" ) propvstart)
                     (mv-let
                      (props rrest)
                      (xml-readnodeproperties rest)
                      (mv (cons (mv
                                 (xml-unescape propname)
                                 (xml-unescape propvalue))
                                props) rrest)))))
                  ))))

;xml-skipdontcares (xmlchars) → returns next xmlchars sans don't cares
(defun xml-skipdontcares (xmlchars)
  (mv-let (ws1 tag1)
          (span-mv *whitespace* xmlchars)
          (if (null tag1)
              nil
              (if (equal (car tag1) #\< )
                  (if (equal (cadr tag1) #\? )
                      (mv-let (dontcare dctok therest)
                              (split-on-token-mv '(#\? #\> )
                                                 (cddr tag1))
                              (xml-skipdontcares therest))
                      (mv-let (matched dontcare comment)
                              (splitoff-prefix-mv '(#\! #\- #\- )
                                                  (cdr tag1))
                              (if (= (length matched) 3)
                                  (xml-skipdontcares
                                   (caddr
                                    (split-on-token
                                     '(#\- #\- #\> ) comment)))
                                  tag1)))
                  xmlchars))))

;xml-readnodes (xmlchars) → returns (mv nodes remainingxmlstring)
(defun xml-readnodes (xmlchars)
  (mv-let (ws content)
          (span-mv *whitespace* (xml-skipdontcares xmlchars))
          (if (null content)
              (mv nil nil)
              (if (equal (car content) #\< )
                  (mv-let (ws1 tag1)
                          (span-mv *whitespace* (cdr content))
                          (if (null tag1)
                              (mv nil nil)
                              (if (equal (car tag1) #\/ )
                                  (mv-let (dontcare dctok therest)
                                          (split-on-token-mv
                                           '(#\> ) tag1)
                                          (mv nil therest))
                                  (mv-let (tagname tagattrs)
                                          (split-at-delimiter-mv
                                           *endtagname* tag1)
                                          (mv-let
                                           (attribs tagend)
                                           (xml-readnodeproperties
                                            tagattrs)
                                           (mv-let
                                            (dc1 dc2 inner)
                                            (split-on-token-mv
                                             '(#\> ) tagend)
                                            (mv-let
                                             (nodes rest)
                                             (if (equal
                                                  (car tagend)
                                                  #\/ )
                                                 (mv nil inner)
                                                 (xml-readnodes
                                                  inner))
                                             (mv-let
                                              (morenodes morerest)
                                              (xml-readnodes rest)
                                              (mv
                                               (cons
                                                (mv
                                                 (xml-unescape
                                                  tagname)
                                                 attribs
                                                 nodes) morenodes)
                                               morerest)))))))))
                  (mv-let (thetext therest)
                          (split-at-delimiter-mv '(#\< ) content)
                          (if (null thetext)
                              (xml-readnodes therest)
                              (mv-let (nodelist restoftext)
                                      (xml-readnodes therest)
                                      (mv
                                       (cons
                                        (mv 'text nil
                                            (xml-unescape
                                             (append ws thetext)))
                                        nodelist)
                                       restoftext)))))))

;xml-readnode (xmlchars) → returns the root node from xmlstring
(defun xml-readnode (xmlchars)
  (let ((result (xml-readnodes (str->chrs xmlchars))))
    (if (null (car result))
        nil
        (caar result))))

;xml-getnodes (node nodename) → returns children of node with type
;  nodename
(defun xml-getnodes (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getnodes (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                (cons curnode (xml-getnodes (mv nil nil rest) nodename))
                (xml-getnodes (mv nil nil rest) nodename))))))

;xml-getdeepnodes (node nodename) → returns children of node with type
;  nodename searching recursively using DFS with node as root.
(defun xml-getdeepnodes (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getdeepnodes (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                (cons curnode
                      (xml-getdeepnodes
                       (mv
                        nil
                        nil
                        (concatenate 'list (caddr curnode) rest))
                       nodename))
                (xml-getdeepnodes
                 (mv
                  nil
                  nil
                  (concatenate 'list (caddr curnode) rest))
                 nodename))))))

;xml-getnode (node nodename) → returns first child node with type
;  nodename
(defun xml-getnode (node nodename)
  (car (xml-getnodes node nodename)))

;xml-getdeepnode (node nodename) → returns first child node with type
;  nodename searching recursively using DFS with node as root.
(defun xml-getdeepnode (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getdeepnode (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                curnode
                (xml-getdeepnode
                 (mv
                  nil
                  nil
                  (concatenate 'list (caddr curnode) rest))
                 nodename))))))

;xml-getattribute (node attributename) → returns the value of node's
; attribute with name attributename
(defun xml-getattribute (node attributename)
  (if (= (length (cadr node)) 0)
      ""
      (let ((curattr (car (cadr node)))
            (rest (cdr (cadr node))))
        (if (string-equal (car curattr) attributename)
            (cadr curattr)
            (xml-getattribute (mv nil rest nil) attributename)))))


;xml-gettext (node) → returns the composite of all text inside of a node
(defun xml-gettext (node)
  (if (null node)
      ""
      (if (= (length (caddr node)) 0)
          ""
          (let ((curnode (car (caddr node)))
                (rest (cdr (caddr node))))
            (if (equal (car curnode) 'text)
                (string-append (caddr curnode)
                               (xml-gettext (mv nil nil rest)))
                (string-append (xml-gettext curnode)
                               (xml-gettext (mv nil nil rest))))))))

;xml-isattribute (attribute) → returns true iff attribute is an mv of
;  length 2 with both elements of the mv being strings
(defun xml-isattribute (attribute)
  (and
   (true-listp attribute)
   (equal (length attribute) 2)
   (stringp (car attribute))
   (stringp (cadr attribute))))

;xml-isattributelist (attributes) → returns true iff attributes
;  is nil or a list of attributes
(defun xml-isattributelist (attributes)
  (or
   (null attributes)
   (and
    (true-listp attributes)
    (xml-isattribute (car attributes))
    (xml-isattributelist (cdr attributes)))))

(mutual-recursion
 ;xml-isnode (node) → returns true iff node is actually a node
 (defun xml-isnode (node)
   (and
    (true-listp node)
    (equal (length node) 3)
    (or
     (and
      (equal 'text (car node))
      (null (cadr node))
      (stringp (caddr node)))
     (and
      (stringp (car node))
      (xml-isattributelist (cadr node))
      (xml-isnodelist (caddr node))))))

 ;xml-isnodelist (nodes) → returns true iff nodes is a list of nodes
 (defun xml-isnodelist (nodes)
   (and
    (true-listp nodes)
    (or
     (null nodes)
     (and
      (xml-isnode (car nodes))
      (xml-isnodelist (cdr nodes))))))

 )

;gluekids (nodes)→ Given a list of nodes, glue all nodes' children
;   together in one big list; i.e., if the nodes are rooted in some tree
;   where they're at depth k, then take all of the nodes at depth k+1
;   (cousins or siblings to one another), and put them into a list
;   together.
(defun gluekids (nodes)
  (if (consp nodes)
    (mv-let (nodename atts kids)
            (car nodes)
            (concatenate 'list kids (gluekids (cdr nodes))))
    nil))
```

```
;xml-bfsfindnodes (nodes nodename) → returns a list of children nodes of type
;    nodename using BFS search that are at the shallowest depth.
(defun xml-bfsfindnodes (nodes nodename)
  ; Build a dummy node that looks like xmlminidom, so we can act like
  ; nodes are rooted there as children, and we can just use xml-getnodes
  ; on it.
  (let* ((dummyroot (mv "dummyroot" nil nodes))
         (maybes (xml-getnodes dummyroot nodename)))
      (if maybes
          ; We found them
          maybes
          ; There were no nodes.  Look deeper.
          (xml-bfsfindnodes (gluekids nodes) nodename))))

(export Ixmlminidom))
```

tImplTeamSteele

Personal Software Process Summary

Project Essentials

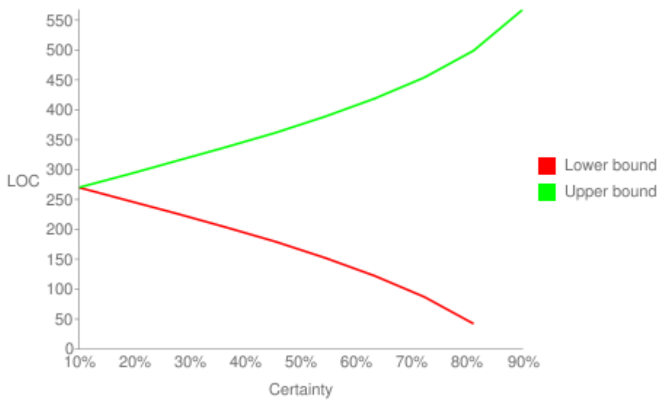
Name: Zach Bartlett
Instructor: Dr. Page
Date: Nov 11, 2010
Language: ---

Lines of Code

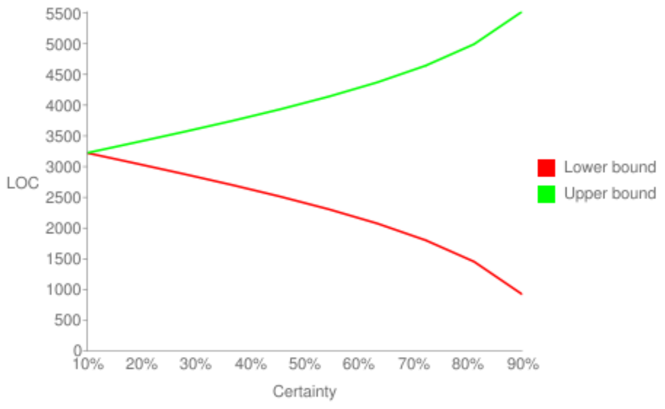
Type	Prediction by user	Actual
Added	327	1515
Base	377	394
Modified	0	183
Removed	0	10

PSP Projection

LoC Certainty

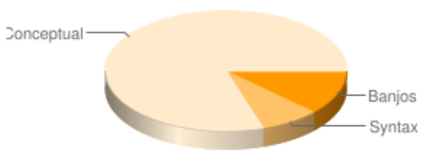


Time Certainty

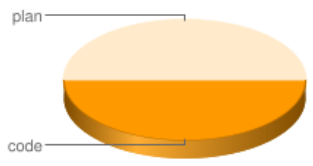


Project Data

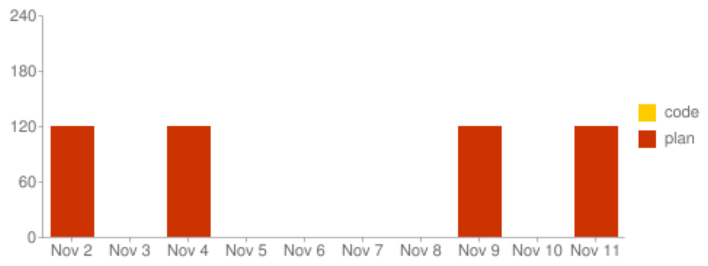
Time Per Defect Type

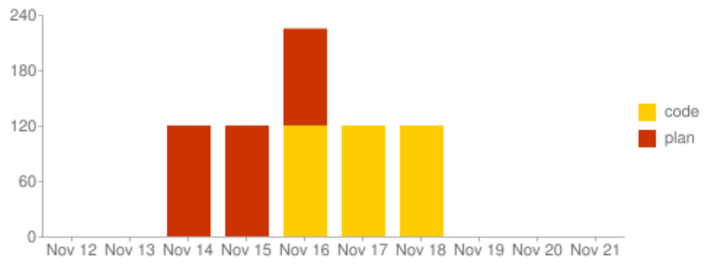


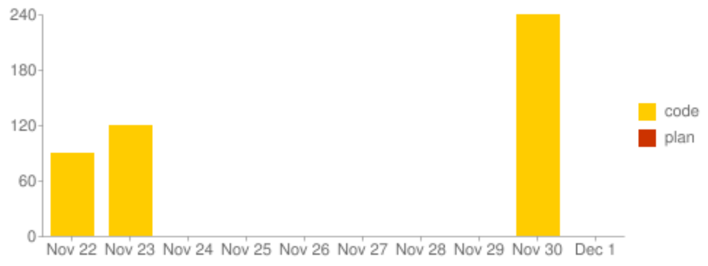
Time Per Phase

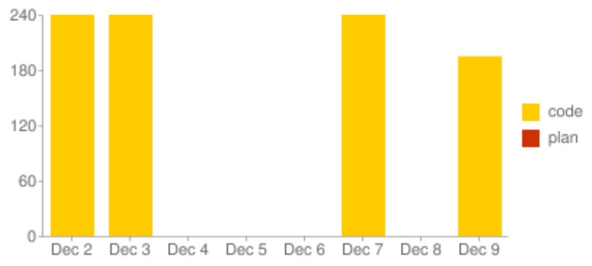


Time by Day



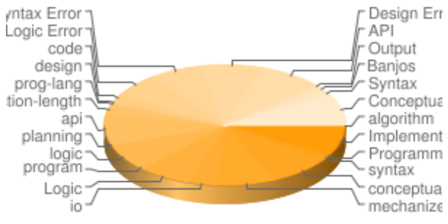




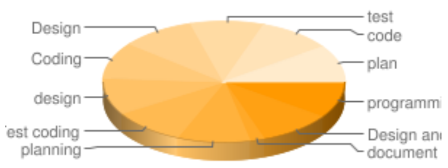


Cumulative Data

Time Per Defect Type



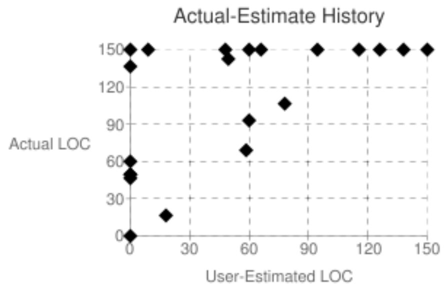
Time Per Phase



Actual vs Estimated LoC

Actual vs Estimated LoC

Project	Estimate	Actual
iEx5	40	62
Individual Exercise 5	77	156



Bridge Hands with Travelers (iEx5)	12	11
iEx5	33	95
iEx5	40	250
iEx4	84	143
Individual Exercise 4	112	101
Bridge Hand Displays (iEx4)	39	46
iEx4	32	244
iEx4	6	175
iEx3	92	145
Individual Exercise 3	52	71
Dealing Instructions from XML data (iEx3)	44	114
iEx3	0	290
iEx3	63	125
Individual Exercise 2	0	91
iEx2	0	0
Project 2	0	33
Individual Exercise 1	0	40
iEx1	0	31
Project 1	0	33

Time Log

Date	Type	Int. Time	Description
Nov 2, 2010, 3:00 PM - 5:00 PM	plan	0	Completed a rough sketch (design) of modules and interfaces. Discussed Project Description and requirements. Discussed previous code implementations to be used in tImpl.
Nov 4, 2010, 12:00 PM - 2:00 PM	plan	0	Worked more on the design, completed a visio diagram of modules and function definitions and assigned tasks to each team member for tDsgn. Looked more at the project description.
Nov 9, 2010, 3:00 PM - 5:00 PM	plan	0	Brought together the function descriptions, decided on github as our distributed version control system. Assigned additional assignments to team members. Looked even more at project description, noting possible difficulties.
Nov 11, 2010, 3:00 PM - 5:00 PM	plan	0	worked on PSP++ report, discussed the project description more to decide how to manage the linkage aspect. More difficult to do score cards and rankings. Set up github accounts, etc.
Nov 14, 2010, 6:00 PM - 8:00 PM	plan	0	Colby, Edward, and Zach worked on Design Review more. Renamed several functions in design to better explain what they should do. Expanded upon annotations of functions especially in psc / rankings.
Nov 15, 2010, 6:00 PM - 8:00 PM	plan	0	Kye, Brett, and Micah worked on Design Review. Updated PROBE data to be more accurate and appended IO module data.

Nov 16, 2010, 11:45 AM - 1:30 PM	plan	0	Group met to perform a final review of tDsgn and tDsgnreview. Looked over function contracts and descriptions, finalized psp data, finalized architecture.
Nov 16, 2010, 3:00 PM - 5:00 PM	code	0	Decided to use Ed's iex5 as a starting point for tImpl. Looked over code and began work on splitting up existing code into modules and interfaces. Realized we needed to repackage xmlminidom and basiclex into modules/interfaces as well.
Nov 17, 2010, 6:00 PM - 8:00 PM	code	0	Ed worked on updating xmlminidom and repackaging it inside a module/interface.Kye worked on repackaging IO. Micah worked on creating module/interface stubs for Rankings. Colby worked on creating module/interface stubs PSC. Zach and Brett worked on repackaging Board.
Nov 18, 2010, 3:00 PM - 5:00 PM	code	0	Discussed tDsgn with group and began to understand, after creating modules / interfaces, what else would be required in modules. Focused primarily on Rankings and PSC modules / interfaces as they would require the most work.
Nov 22, 2010, 11:30 PM - 1:00 PM	code	0	Zach put together a general idea of what getseparateresults should look like. This included a mix of code and psuedo code comments.
Nov 23, 2010, 3:00 PM - 5:00 PM	code	0	Reviewed Mboard for tCodeRvw during class time. Group was able to cover this entire module (around 260 lines).
Nov 30, 2010, 11:45 AM - 3:00 PM	code	0	Met and reviewed code so far. For this session this included going over the implemented functions so far in Mio (including boards-no-trav, boards-trav, and main primarily).
Nov 30, 2010, 3:00 PM - 5:00 PM	code	0	Group met and fixed getseparateresults. Needed a somewhat complicated mixture of mv-let structures and associated lists from these mv-let structures to create an easily searchable list based on the unique id of a direction, id, and section.
Dec 2, 2010, 11:45 AM - 6:00 PM	code	0	Group met and worked on Tmio and PSC. Ed, Colby, Micah, and Zach worked on implementing PSC and Brett and Kye worked on getting all other modules debugged and runnable in Tmio. PSC currently outputs all but Rankings related information.
Dec 2, 2010, 6:00 PM - 7:30 PM	code	0	Colby worked on Rankings module to implement getcontestants and a few helper xml node functions.
Dec 3, 2010, 1:30 PM - 6:00 PM	code	0	Group worked together on finishing up the implementations to Rankings module. Created a few new functions (serializedcontestants, etc.) and renamed functions to fit scheme. Rankings is finished other than creating a list of name ID pairs for PSC.
Dec 7, 2010, 11:45 AM - 3:00 PM	code	0	Colby worked on getcontestants and above, Micah worked on cleaning up the CSS.
Dec 7, 2010, 3:00 PM - 6:00 PM	code	0	Colby and Zach fixed PSC to display names for IDs in the "Versus" column. Group worked on contracts.

Dec 7, 2010, 9:00 PM - 11:00 PM	code	0	Ed worked on Ixmlminidom and created a working property function as well as functions that create "dummy" xml data.
Dec 9, 2010, 11:45 AM - 3:00 PM	code	0	Finished up contracts, cleaned up design, worked on finalizing psp, made final changes to headers for psc cards.

Defect Log

Date	Phase	Fix Time	Description
Nov 4, 2010	Conceptual	15	Realized XML files given as examples had given schemas. Resulted in less work required to differentiate between placement of results tags for boards.
Nov 4, 2010	Conceptual	30	Wrote new xmlminidom (utilities) functions that performs a Depth First Search of nodes of a particular name.
Nov 9, 2010	Conceptual	20	Decided that getseparateresults needed a helper function to send results data that is not appended with html tags.
Nov 14, 2010	Conceptual	30	Decided that board functions required addition specification and new names (serialized...).
Nov 16, 2010	Conceptual	45	Realized the need for functions that returned results in a way that could be easily searched based on a player's ID (direction, id, and section). Additionally this information cannot be serialized in html.
Nov 23, 2010	Conceptual	35	Fixed many syntax errors in Mboard and Mio, such as incorrect sigs and defuns and misplaced parenthesis. Ed, Zach, and Colby focused on Mboard and Brett, Kye, and Micah focused on Mio.
Nov 30, 2010	Conceptual	35	Realized we needed to remove the old mv associated list structure from the list of associated lists we were creating after inserting its old information with our new information from a new board. This way a search on these lists returns a single list.
Nov 30, 2010	Conceptual	15	Realized we need to pass in an initial north-south and east-west list to allow the structure of building the associated mvs we wanted. Called these ns1 and ew1.
Nov 30, 2010	Conceptual	45	Called wrong let in getseparateresults. Wanted TotalScore attribute text and gave function TotalScore node.
Dec 2, 2010	Conceptual	25	Fixed error in getAllPairs where ew list was being iterated needlessly along with ns, causing the function to end on the end of the ns pairs and check only whatever ew pairs whose index was after the length of ns.
Dec 2, 2010	Conceptual	35	Fixed problem in linking in Tmio where the order of the linkage mattered in terms of what modules relied on other modules.

Dec 3, 2010	Conceptual	25	Created function sortmvranks so that we could fix an error where the lack of an A rank would cause an error in serializedrankings not outputting B and C if they existed, likewise with B not existing but C existing.
Dec 3, 2010	Conceptual	10	Replaced nil in sortmvranks with to solve need for separate handling of empty table entries.
Dec 3, 2010	Conceptual	15	Fixed error syntax error in Rankings function sortmvranks, (mvs) should have been mvs, confused compiler.
Dec 3, 2010	Conceptual	10	Fixed infinite loop in rankings, forgot ending condition on serializedrankings after modifing it.
Dec 3, 2010	Conceptual	30	Fixed error where Rankings function sortmvranks was taking in the wrong order of ranks (such as Section / Overall).
Dec 7, 2010	Conceptual	30	Fixed error where Rankings function sortmvranks was taking in the wrong order of ranks (such as Section / Overall).
Dec 8, 2010	Conceptual	10	Moved bfsfindnodes out of rankings and into xmlminidom (personal score cards module now uses and made sense).
Dec 8, 2010	Conceptual	15	Fixed formatting in link header for all files.
Dec 8, 2010	Conceptual	25	Made significant formatting changes to all html, centered output and table width.
Dec 8, 2010	Syntax	10	Improper signatures for new functions
Dec 8, 2010	Conceptual	20	Fixed sigs for many functions that were being modified.
Dec 8, 2010	Banjos	15	Fixed length of ragtime.wav to match property test time.