Comp210 Exam #1, Fall 2003

Name ___________________________________

Honor Code pledge (write and sign please!):

This is an in-class, closed-book, closed-notes exam.

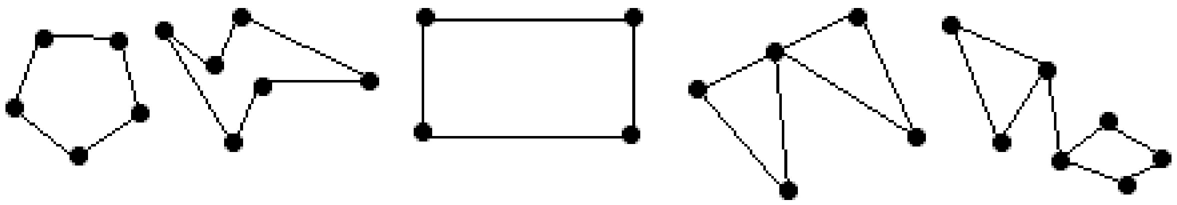
Scores (to be filled in by graders):

```
    1:       /  10

    2:       /  10

    3:       /  10

    4:       /  10

    5:       /  20

    6:       /  30

    7:       /  10
       -----

Total:       / 100
```

While the problems all follow a single "storyline" of development,you should be able to do them in almost any order. So, if you have trouble with a problem, move on to another. Each problem indicates which parts of the design recipe you should include.

Comp210 Exam #1, Fall 2003

Through the following problems, we will develop code to represent arbitrary polygons. The following are valid polygons for our purposes where all the straight lines are single line segments:



(The rightmost polygon has two line segments on top of each other.)

A polygon will be represented by a list of its border line segments.
Each line segment is a pair of 2-dimensional Cartesian points.

As in the textbook, the data definition for 2D points is as follows:

```
;A posn represents an (x,y) point.
(define-struct posn (x y))
```

The following definition is used in the supplied test cases. `squarePoly` is a valid polygon with the supplied vertices below. `createPolygon` simply returns a `Polygon` given a list of `posn`s--don't worry about its implementation as you won't need it.

```
(define squarePoly (createPolygon
                  (list (make-posn 0 0)
                        (make-posn 1 0)
                        (make-posn 1 1)
                        (make-posn 0 1))))
```

Problem 1 (10 points):

Provide the data definitions of "`Polygon`" and "`LineSeg`", as described by the above English text.

Comp210 Exam #1, Fall 2003

Problem 2 (10 points):

As in the textbook, the template for functions on a "`posn`" is

```
     ;a-posn-fun : posn -> ...
     ;(define (a-posn-fun a-posn)
     ;   ...(posn-x a-posn)...
     ;   ...(posn-y a-posn)...)
```

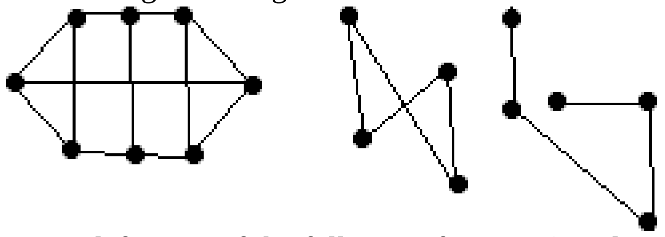
Finish the following templates for functions on a line segment and functions on a polygon.

```
;a-lineseg-fun : LineSeg -> ...
;(define (a-lineseg-fun a-lineseg)
;   ...)
```

```
;a-polygon-fun : Polygon -> ...
;(define (a-polygon-fun a-polygon)
;  ...)
```

Comp210 Exam #1, Fall 2003

A polygon data structure only truly models a mathematical polygon if the `LineSegs` *don't intersect each other*, and *they form a closed shape*. We will want to test for these properties. Here are some invalid polygons, where all the straight lines shown are single line segments:



First, assume that you are GIVEN a definition of the following function (*i.e.* don't write it yourself):

```
;intersect? : LineSeg LineSeg -> boolean
;Returns whether the two lines intersect. If one LineSeg is fully
;contained within the other, or if one LineSeg shares only its endpoint
;with the other, they are NOT considered intersecting.
```

Problem 3 (10 points):

Define the following function. You have previously given the appropriate data analysis, and you do not need to show any test cases. You do not need to rewrite the contract and purpose.

```
;intersect-any? : LineSeg Polygon -> boolean
;Returns whether the line segment intersects any of those in the polygon.
```

```
"intersect-any? test cases:"
(boolean=? false (intersect-any? (first squarePoly) squarePoly))
(boolean=? true (intersect-any? (make-LineSeg (make-posn -2 -1)
                                              (make-posn 4 3))
                                squarePoly)
```

Problem 4 (10 points):

Define the following function. You do not need to show any test cases. You do not need to rewrite the contract and purpose.

```
;any-intersect-any? : Polygon -> boolean
;Returns whether any line in the polygon intersects any other.
```

Problem 5 (20 points):

A representation of a polygon only makes sense if it is closed, i.e., the line segments should touch end-to-end in a single continuous whole. (See the previous drawing.)

For simplicity, let's further restrict our representation. We will require that the 2nd endpoint of each line segment is the same as the 1st endpoint of the next line segment. Also, we want the 2nd endpoint of the last line segment to be the same as the 1st endpoint of the first line segment. First, consider the following definition:

```
;last : list-of-any --> any or string
;Returns the last element in the list, if any.
; Otherwise, “No last element”
(define (last a-list)
  (cond
    [(empty? a-list) 'No_last_element]
    [(cons? a-list)  (last_helper (rest a-list)(first a-list))]))

;last_helper : list-of-any any --> list-of-any
;Returns the last element in the list or prev, if the list is empty
(define (last_helper a-list prev)
  (cond
    [(empty? a-list) prev]
    [(cons? a-list)  (last_helper (rest a-list) (first a-list))]))
```

Continued →

Given `last` above, one possible definition of `closed?` is as follows.

```
;closed? : Polygon --> boolean
;Returns whether the polygon is closed.
(define (closed? poly)
  (cond
    [(empty? poly) false]  ;A polygon of no line segments is not closed.
    [(cons? poly) (and (equal? (LineSeg-p2 (last poly))
                               (LineSeg-p1 (first poly)))
                       (all-end-to-end? poly))]))

;all-end-to-end? : Polygon -> boolean
;Given a list of the remaining line segments, return whether these line
;segments all meet endpoint to endpoint.
(define (all-end-to-end? poly)
  (cond
    [(empty? poly) false]  ;A polygon of no line segments is not closed.
    [(cons? poly)
     (cond
       [(empty? (rest poly)) true] ; A single line segment is end-to-end
       [(cons? (rest poly)) (and (equal? (LineSeg-p2 (first poly))
                                          (LineSeg-p1 (first (rest poly))))
                                 (all-end-to-end? (rest poly)))])]))
```

This definition works correctly, but it has two distinct stylistic problems. What are they?

(NOTE that the algorithm could be made somewhat more efficient by integrating the list traversal of "`last`" into that of "`closed?`". This is NOT the kind of answer we are looking for. Exclude "`last`" from your consideration of problems.)

Comp210 Exam #1, Fall 2003

Problem 6 (30 points):

Provide a better definition of `closed?` that fixes those two problems. You may use any of the code in the previous definition without rewriting it, but be clear about what you are deleting or changing. You do NOT need to repeat any unchanged contracts and purposes. You do not need to provide test cases.

Hint: Fix the two problems separately. One is easy to fix, the other takes some thought.

```
"closed? test cases:"
(boolean=? true (closed? squarePoly))
(boolean=? false (closed? (cons (make-LineSeg (make-posn 2 3) (make-posn -1 2)) squarePoly))
(boolean=? false (closed? (cons (make-LineSeg (make-posn 2 3) (make-posn 0 0)) squarePoly))
```

Problem 7 (10 points):

Once again, a Polygon truly represents a polygon *only if it is closed* **and** *doesn't intersect itself*.
Define the following function. You do not need to show any test cases or repeat the contract and purpose.

```
;valid? : Polygon -> boolean
;Returns whether the polygon is a proper representation.
```

```
"valid? test cases:"
(boolean=? true (valid? squarePoly))
```