C211 Exam 1 – Fall 2015

Name: _____

Student Id (last 4 digits): _____

- You may use the usual primitives and expression forms from Intermediate Student; for everything else, define it.
- You may write `c` → `e` for `(check-expect c e)`.
- We expect data definitions to appear in your answers, unless they're given to you. We expect you to follow the design recipe on all problems. In a number of cases, defining helper functions will be useful.
- If they are given to you, you do not need to repeat the signature and purpose statements in your implementations. Likewise, you do not need to repeat any test cases given to you, but you should add tests wherever appropriate.
- You do not need to write separate templates unless the problem explicitly asks for them. We expect that the functions you write follow the appropriate template, however.
- Some basic test taking advice: Before you start answering any problems, read *every* problem, so your brain can be thinking about the harder problems in background while you knock off the easy ones.

Problem	Points	/out of
1		/ 22
2		/ 11
3		/ 20
4		/ 10
5		/ 11
Total		/ **74**

Good luck!

Problem 1 A *Vertex* data structure contains 2 numbers: x and y. Write a data definition for *Vertex*, and two examples of *Vertex*. Throughout this problem, you can assume that a *Vertex* never stores negative numbers. Use the following structure definition:

22 POINTS

```
(define-struct vertex (x y)).
```

Write the data definition for *ListofVertex*. Write two examples of *ListofVertex*.

Design the function `make-small`, which consumes two *Vertex*s and produces a new *Vertex* whose x coordinate is the smaller of the two x coordinates of the given *Vertex*s, and whose y coordinate is the smaller of the two y coordinates of the given *Vertex*s. Here is an example:

```
(check-expect (make-small (make-vertex 0 5) (make-vertex 5 0))
              (make-vertex 0 0))
```

You may find the `min` function useful.

A *bounding box* for a *ListofVertex* is a box that fully encompasses all vertices in the list. A bounding box is defined via a pair of *Vertex*s: the min and the max. The min vertex is a vertex with an x coordinate is less than every x coordinate in the list, its y coordinate is less than every y coordinate in the list. Similarly for the max vertex, its coordinates are greater than every corresponding coordinate in the list.

Design a function that accepts a *ListofVertex* and computes the min *Vertex* of the list. If there are no elements in the list, both coordinates should be 0.

Design a function `inside?` that given the min and max points which define a bounding box, and another vertex, determine if that point is within that bounding box (that is, if the x coordinate is in between the min x coordinate and the max x coordinate, and the same for the y coordinate).

Problem 2 Recall the data definition for a list of numbers: 11 POINTS

```
;; A ListOfNumber is one of:
;; - empty
;; - (cons Number ListOfNumber)
```

A sorted list of numbers is defined such that each number in the list is at least as big as the previous number, i.e. `(list 1 3 6 10 34 225 337)` is a sorted list. Also note that an empty list is also a sorted list.

Design the `insert` function, which accepts a sorted list and a number and inserts that number in the appropriate position such that the list is still sorted. As the list is already sorted, all you need to do is search through the list until you find a number that is less than or equal to the given number, and insert the number immediately after this location. Some examples of using your `insert` function would be:

```
(insert (list 1 3 7 9) 4)  -> (list 1 3 4 7 9)
(insert (list 1 3 7 9) 11) -> (list 1 3 7 9 11)
(insert (list 1 3 7 9) 0)  -> (list 0 1 3 7 9)
(insert empty 5)           -> (list 5)
```

[Here is some more space for the previous problem.]

Design a function `sort` which is given a list of numbers and produces a sorted list of numbers. You will use the your previously defined `insert` function to perform this. Note, that by definition an empty list is a sorted list and may be given to your `insert` function along with a number to produce a sorted list.

Hint: Follow the template for this function. Trust your purpose statements.

Problem 3 Design a data definition to represent a *Person*. A *Person* can either have no children, or can have two children, both of which are also *Person*s. A *Person* also has an eye color and height. The eye color should be represented by a string, and height is represented by a number of inches.

20 POINTS

Design a function which given a *Person*, counts the number of *Person*s among them and all their children who have eyes that are `"blue"`.

Design a function which, given a *Person*, determines if that *Person* and all of their children are over 60 inches tall.

[Here is some more space for the previous problem.]

Problem 4 Here is the data and structure definition for a *FootballTeam*. 10 POINTS

```
(define-struct team (name wins losses))
;; A FootballTeam is (make-team String Number Number)
```

List the functions defined by this structure definition. Then write an example.

Design the function `playoffs?`, which determines if the given *FootballTeam* is currently in position to make the playoffs. A team is in position to make the playoffs if its winning percentage is at least as good as if it had 10 wins and 6 losses. For example, a team with 5 wins and 3 losses *is* in position to make the playoffs.

Problem 5 Consider the following data definition: 11 POINTS

```
;; A Set is one of:
;; - (make-empty-set)
;; - (make-one-set Number)
;; - (make-two-set Number Number)
;; - (make-more Number Set)
```

Design the function `member-of`, which takes a *Number* and a *Set* and determines if the given *Number* is in the *Set*. Your function should produce a *Boolean* as the answer.

Design the function `add-to`, which takes a *Number* and a *Set* and produces a new set with the *Number* in the new *Set*.

Consider the following program:

```
(member-of n (add-to s n))
```

Can you come up with definitions of `s` and `n` which make this expression produce `#false`? If so, give the definitions. If not, why not?