Assignment: 3

Due:	Tuesday, October 4th, 9:00 pm
Language level:	Beginning Student
Files to submit:	`olympics.rkt`, `set.rkt`, `triangle.rkt` (bonus)
Warmup exercises:	6.3.2, 6.4.1, 7.1.2
Practise exercises:	6.3.3, 6.4.2, 6.4.3, 6.5.2, 7.1.3, 7.5.1, 7.5.2, 7.5.3

- Policies from Assignment 2 carry forward.
- Your solutions must be entirely your own work.
- Solutions will be marked for both correctness and good style.
- Good style includes qualities such as meaningful names for identifiers, clear and consistent indentation, appropriate use of helper functions, and documentation (the design recipe).
- **For this and all subsequent assignments, you should include the design recipe as discussed in class (unless otherwise noted, as in question 1).**
- Test data for all questions will always meet the stated assumptions for consumed values.
- You must use *check-expect* for both examples and tests of functions that produce exact values. You must use *check-within* for examples and tests of functions that produce inexact values.
- You may use the **cond** special form. You are **not** allowed to use **if** in any of your solutions.
- **It is very important that the function names match ours.** You must use the basic tests to be sure. In most cases, solutions that do not pass the basic tests will not receive any correctness marks. The names of the functions must be written exactly. The names of the parameters are up to you, but should be meaningful. The order and meaning of the parameters are carefully specified in each problem.
- Any string or symbol constant values must exactly match the descriptions in the questions. Any discrepancies in your solutions may lead to a severe loss of correctness marks. Basic tests results will catch many, but not necessarily all of these types of errors.
- Since each file you submit will contain more than one function, it is very important that the code runs. If your code does not run then none of the functions can be tested for correctness.
- Do not send any code files by email to your instructors or tutors. Course staff will not accept it as an assignment submission. Course staff will not debug code emailed to them.
- You may use examples from the problem description in your own solutions.

Here are the assignment questions you need to submit.

1. In this question, you will perform step-by-step evaluations of Racket programs, by applying substitution rules until you either arrive at a final value or you cannot continue. You will use an online evaluation tool that we have created for this purpose.

   To begin, visit this web page:

   `https://www.student.cs.uwaterloo.ca/~cs135/stepping`

   Note: the use of `https` is important; that is, the system will not work if you omit the `s`. This link is also in the table of contents on the course web page.

   You will need to authenticate yourself using your Quest/WatIAm ID and password. Once you are logged in, try the questions in the "Warmup questions" category under "CS 135 Assignment 3," in order to get used to the system. Note the "Show instructions" link at the bottom of each problem. Read the instructions before attempting a question!

   When you are ready, complete the six stepping problems in the "Assignment questions" category, using the semantics given in class for Beginning Student. You can re-enter a step as many times as necessary until you get it right, so keep trying until you completely finish every question. All you have to do is complete the questions online—we will be recording your answers as you go, and there is no file to submit. The basic tests for this assignment will tell you whether or not we have a record of your completion of the stepper problems. **Note however that you are not done with a question until you see the message** `Question complete!` You should see this once you have arrived at a final value and clicked on "simplest form" (or "Error," depending on the question).

   You are tracing the given expressions assuming that all function and constant definitions that appear above the expression exist. This means that you are not required to do any simplification of the constant definitions as part of the tracing.

   You should **not** use DrRacket's Stepper to help you with this question for several reasons. First, as mentioned in class, DrRacket's evaluation rules are slightly different from the ones presented in class, but we need you to use the evaluation rules presented in class. Second, in an exam situation, you will not have DrRacket's Stepper to help you, and there will definitely be step-by-step evaluation questions on at least one of the exams.

   Note: If you get stuck in the tracing question, **do not post to Piazza requesting the next step in the trace.** This is a violation of the academic integrity policy. Review the substitution rules carefully from Module 3 to try to solve the problem yourself. If you still cannot find your error, then you are encouraged to ask a question in person during office hours. As a last resort, you may make a private post to Piazza describing where you are stuck. Course staff will provide guidance directing you to the next step, but they will not give you the answer. Public post to Piazza related to question 1 of this assignment will be deleted.

2. In the Olympic games, athletes compete on behalf of their countries. Top competitors are awarded medals: gold for first place, silver for second place, and bronze for third place. When reporting the results from the Olympics, news organizations often rank countries based on the number of gold, silver, and bronze medals won.

   This question will be using the following structure and data definition.
   (*define-struct medalcount* (*country gold silver bronze*))
   ;; A MedalCount is a (make-medalcount Str Nat Nat Nat).

   You must include structure definition for *medalcount* and the data definition for `MedalCount` at the beginning of your solution.

   Write a function called *higher-rank?* that consumes two `MedalCount` structures and produces *true* when the first structure ranks higher than the second structure, and *false* otherwise. Note that this means if the rankings are the same, the function should produce *false*.

   The ranking is decided as follows:

   - The country with the higher total number of medals is ranked higher.
   - If the total medal count is the same, then the country with the higher number of gold medals is ranked higher.
   - If the total medal count and total number of golds is the same, then the country with the higher number of silver medals is ranked higher.
   - If the total medal count, total number of golds, and total number of silvers is the same, then the country with the higher number of bronze medals is ranked higher.

   You must **not** use **cond** in your solution. Place your solution in in the file `olympics.rkt`.

3. The following two questions are based on a game called Set™. Set™ is a game with cards that have four features: a shape, a shading, a colour, and a number of shapes. For more information about the game see: `http://en.wikipedia.org/wiki/Set_(game)`. Each card has one of three possible shapes: diamond, squiggle, and oval. Each card has one of three possible shadings: solid, striped, and open. Each card has one of three possible colours: red, green, or purple. Each card has one, two or three of the shapes on it. An example of a card description is: three green ovals with solid shading. Each card in the game is unique. In the game of Set™, you must find combinations of three cards where each of the features on the three cards are either all the same or all different.

For example, this is a set:

- three green ovals with solid shading
- three red ovals with striped shading
- three purple ovals with open shading

because all the cards have three ovals, but each card has a different colour and a different shading.

This is not a set:

- one green squiggle with solid shading
- two green squiggles with solid shading
- three green squiggles with striped shading

because although all the cards have green squiggles and a different number of shapes, two of the three cards have the same shading.

A *setcard* is a structure, of type `SetCard`, with four fields that appear in the following order:

- *number*, a positive integer in the range 1 to 3 inclusive.
- *colour*, a symbol, one of 'red, 'green, or 'purple.
- *shape*, a symbol, one of 'diamond, 'oval, or 'squiggle.
- *shading*, a symbol, one of 'solid, 'striped, or 'open.

Write the structure definition for *setcard* and data definition for `SetCard` at the beginning of your solution.

a) Write a function called *valid-set?* that consumes three different `SetCard` structures and produces *true* when they form a set, and *false* otherwise.

b) Write a function called *complete-set* that consumes two different `SetCard` structures and produces a `SetCard` that would satisfy the requirements such that the three cards together (the two cards that are consumed and the one card that is produced) form a set. For example, if the function consumes (*make-setcard* 1 'purple 'diamond 'solid) and (*make-setcard* 3 'red 'oval 'striped) then the only card that could complete a set with these two cards would be (*make-setcard* 2 'green 'squiggle 'open).

   Good solutions can be created with a relatively small number of question/answer pairs and well chosen helper functions. Your solution must **not** simply generate a different constant for each possible pair of `SetCard` structures.

Place your solution in in the file `set.rkt`.

This concludes the list of questions for which you need to submit solutions. Don't forget to always check your email for the basic test results after making a submission.

---

4. **BONUS QUESTION (5%)**

   Structures are incredibly versatile. The fields of a structure can be any type and they can include other structures. Since functions in Racket can only produce a single result, structures can be used as a way to contain multiple data values within a single reference.

   Recall that a *posn* structure represents a point on the Cartesian plane. Suppose you have another structure, that contains three *posn* structures defined as follows:

   A *triangle* is a structure, of type `Triangle`, with three fields that appear in the following order:

   - *point1*, a *posn* representing one vertex of the triangle.
   - *point2*, a *posn* representing one vertex of the triangle.
   - *point3*, a *posn* representing one vertex of the triangle.

   Write a function called *triangle-area* that consumes a `Triangle` structure and produces a number representing the area of the shape if the three points actually form a triangle, and produces *false* if the three points are collinear (i.e. form a line) or if two or more of the points are the same. You may need to do some online research to find out how to calculate the area and/or how to determine if points are collinear.

   Include all appropriate structure and data definitions.

   As usual, this is an all-or-nothing type bonus question. No partial marks are awarded.

   Place your solution for the bonus question in in the file `triangle.rkt`