CS 135 Winter 2016
Petrick

Assignment: 3

Due: Tuesday, Januar 26th, 9:00 pm
Language level: Beginning Student
Files to submit: `robot.rkt`, `area.rkt`, `cards.rkt`, `quaternion.rkt`
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).
- Do not forget to include the design recipe for every function *and* for every defined structure, as discussed in class.
- You must use *check-expect* (resp. *check-within*, where appropriate) for both examples and tests.
- You must use the **cond** special form, and are not allowed to use **if** in any of your solutions.
- **It is very important that the function names and structure field names match ours.** You must use the basic tests to be sure. 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.

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 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.

2. Write the function *robot* that moves a robot from one location to another. The function *robot* consumes four values: the initial robot position as a *Posn*, the initial direction the robot is facing (one of the four symbols 'north, 'east, 'south, or 'west), a turning action for the robot (one of the three symbols 'left, 'right, 'noturn) and a non-negative distance for the robot to move. On a left turn, the robot turns counterclockwise 90 degrees. On a right turn the robot turns clockwise 90 degrees. On no turn, the robot does not turn. After performing the turning action, the robot then moves forward the given distance to its new position. The function should produce a new *Posn* that records the robots new position after turning and moving the given from its initial position and orientation (you should assume that 'east is the positive x-direction and 'north is the positive y-direction, with 'west and 'south being the corresponding negative directions).

   Submit your code to this question as a file named `robot.rkt`.

3. A *Posn* is a built-in structure in Racket that allows working with points in two dimensions. For this question you will define a *3dPosn* structure that allows you to work with points in three dimensional space. The fields of a *3dPosn* are the coordinates for x, y and z.
   ;; A 3dPosn is a (make-3dposn Num Num Num)

   Write a function *area-triangle* that consumes three 3D coordinates (*3dPosn*s) and produces either the area of a triangle formed by the coordinates or 'undefined if the coordinates do not form a triangle. The contract for *area-triangle* is:
   ;; area-triangle: 3dPosn 3dPosn 3dPosn → (anyof Num 'undefined)

   One formula for the area of a triangle can be found at `www.mathworld.wolfram.com/TriangleArea.html` and is given to you below.

$$\text{Area} = 1/4 \times \sqrt{2b^2c^2 + 2c^2a^2 + 2a^2b^2 - a^4 - b^4 - c^4}$$

   where a, b and c represent the side lengths of the triangle. You may also implement another form of the formula for a triangle in 3 dimensions, provided it works. Note that the vertices consumed by this function may not form a proper triangle. If all three 3D coordinates are co-linear, a degenerate triangle is formed (area of the triangle will be 0). Your code needs to check for such special cases and produce 'undefined. Also, while the vertices consumed will be exact numbers, the area computed may be inexact and should be tested with *check-within* using 0.001 for precision.

   Submit your code to this question as a file named `area.rkt`.

4. A *Card* structure has the fields *rank* and *suit*:
   ;; A Card is a (make-card Nat Sym).

   A *Hand* structure has the fields *c1*, *c2*, and *c3*:
   ;; A Hand is a (make-hand Card Card Card).

   The *suit* of a *Card* is one of 'clubs, 'diamonds, 'hearts, or 'spades. The *rank* of a *Card* is an integer in the range 1 to 13, inclusive.

   Submit your code to this question as a file named `cards.rkt`.

   (a) Write structure definitions and the accompanying data definitions for a *Card* and a *Hand*. Use the field names as described above.

   (b) Write a function *better-card* which consumes two cards, and produces the *Card* which is the better of the two.

   The better card is either the card with the better *suit* (the suits are increasing from 'clubs (worst), 'diamonds (second-worst), 'hearts (second-best), or 'spades (best)) or, if the cards have the same *suit*, then the best card is the card with the largest/highest *rank*. If the two consumed *Card*s are the same (i.e., same *suit* and *rank*), return either one.

   (c) Write the function *hand-value* which consumes a *Hand* and produces a symbol indicating the best hand-value of the given *Hand* (where the arrangement/order of the three cards in the *Hand* can be altered). The hand-values are described in decreasing order (from best to worst) below:

   - 'straight-flush: all three cards are the same suit and their ranks are three consecutive integers;
   - 'flush: all three cards are the same suit;
   - 'straight: the ranks of the three cards are three consecutive integers;
   - 'three-of-a-kind: the ranks of the three cards are the same;
   - 'pair: the ranks of two of the cards are the same;
   - 'high-card: none of the previous outcomes are satisfied.

5. **BONUS QUESTION (5%)** Sir William Rowan Hamilton introduced in the 19th century the notion of quaternions. These are of the form $x_0 + x_1 i + x_2 j + x_3 k$, where x_0, x_1, x_2, x_3 are numbers and i, j, k are symbols. For two numbers a, b (both not equal to zero), the multiplication rules of i, j and k can be described by the following multiplication table.

	i	j	k
i	a	k	aj
j	$-k$	b	$-bi$
k	$-aj$	bi	$-ab$

Remark: As we do not have commutativity here, the order matters. You read this table as "row times column", not the other way around. For example, in this table you can deduce that the product $i \cdot k$ will be equal to aj.

The symbols i, j and k commute with any number, i.e. for every number c we have $ci = ic$, $cj = jc$ and $ck = kc$.

Example: Let us assume the classical quaternions given with $a = -1, b = -1$. Then we have

$$(2 + 3i + 0j + 0k) \cdot (5 + 6i + 0j + 0k) = -8 + 27i + 0j + 0k.$$

(You might recognize a certain relation to the multiplication we defined for *posn* structures above while trying to understand this example.)

You are provided the following structure for an element $x_0 + x_1 i + x_2 j + x_3 k$ in the quaternions.

```
(define-struct quaternion (cc ic jc kc))
;; A Quaternion is a (make-quaternion Num Num Num Num).
```

You can copy this definition into your code (but do not do "Copy&Paste", as mentioned in section 1.2 of the Style guide).

The quaternion structure, as you can see, contains 4 fields

- *cc* – representing the constant coefficient x_0.
- *ic* – representing the coefficient of i, i.e. x_1.
- *jc* – representing the coefficient of j, i.e. x_2.
- *kc* – representing the coefficient of k, i.e. x_3.

Write a function *quat-mult*, which consumes two numbers a, b and two elements *q1* and *q2* in the quaternions. This function shall produce the result of the multiplication *q1*·*q2*.

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

Submit your code to this question as a file named `quaternion.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.

Enhancements: *Reminder—enhancements are for your interest and are not to be handed in.*

The bonus question is not easy. If you are managing to solve it by yourself, you can pat yourself on the shoulder for accomplishing something which can be considered challenging for a first-year student.