Assignment 3

Due:	Tuesday, October 6th, 9:00 pm
Language level:	Beginning Student
Files to submit:	`wonderdiet.rkt`, `fourdigitenum.rkt`, `complexmath.rkt`, `quaternion.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).
- 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. Recall Assignment 2 Question 4 (A2Q4) which required several nutritional functions to be written. All of these functions required the same set of nutritional parameters–serving size, fat content, carb content and protein content. Looking at these functions from a software engineering point-of-view, the shared parameter set can be a source of bugs: it's easy to make a typo while copying the parameters when writing new functions, and it's difficult to ensure that each function maintains the same parameter set after they've been written. By encapsulating these common parameters in a structure, we can avoid some of these bugs. This question builds upon A2Q4 by encapsulating the nutrition parameters for several new functions.

   a) Define a new *Nutri-Info* type as a structure *nutri-info* with the following field names:
      (i) *serving-size*, a non-negative number representing the serving-size in grams.
      (ii) *fat*, a non-negative number representing the fat-content in grams.
      (iii) *carb*, a non-negative number representing the carbohydrate content in grams.
      (iv) *prot*, a non-negative number representing the protein content in grams.

      In order for a *Nutri-Info* to be valid, the sum of (ii)-(iv) must not exceed (i).

      You are not required to provide a template function.

   b) Write a predicate function *valid-nutri-info?* with a single parameter. The predicate shall determine if the parameter is a valid *Nutri-Info*. In other words, it shall check if the consumed value is a *nutri-info* structure and determine if the requirements of part a) are fulfilled by the numbers in the respective fields. You can assume that the fields in a provided *nutri-info* structure only contain numbers (*i.e.*, you do not need to check the type of each field).

   c) Write a function *higher-protein* that consumes two *Nutri-Info*s and produces the one with the higher protein content. If both have the same protein content, your function must produce the *Nutri-Info* with the smaller serving size. If both the protein content as well as the serving size are the same, simply produce the first *Nutri-Info*.

   d) Write a function *combine-nutri-info*, which consumes two *Nutri-Info*s and produces a new *Nutri-Info* that is the sum of both consumed values (i.e. each field is the sum of the respective fields in both consumed parameters).

   e) Dr. H.C. Charlatan has designed a potentially-lucrative fad diet targeting gullible people. Suppose someone eats a food (*make-nutri-info s1 f1 c1 p1*). Dr. H.C. Charlatan's claim is that to promote weight loss, any food (*make-nutri-info s2 f2 c2 p2*) eaten within the next hour must have all of the fat, carbohydrate and protein contents differ from the first food by at most 5g. In other words, $|f1 - f2| \leq 5$, $|c1 - c2| \leq 5$ and $|p1 - p2| \leq 5$.
      Charlatan's logic for this diet is not of concern here.[1]

      Write a function *good-combo?* that consumes two *Nutri-Info*s and determines if the second *Nutri-Info* can be eaten in the hour following the first *Nutri-Info*, according to Charlatan's diet.

[1] But feel free to be creative and write a "logical" explanation for this as a comment at the bottom of your file, clearly separated from your code. This is not a bonus question, but we may share the best explanations on Piazza.

Example: Assume you eat a yummy doughnut after a long day of trying to figure out its surface area. Suppose this doughnut weighs $52g$ with $12g$ fat, $22g$ carbs and $2g$ protein. Afterwards, you crave a small steak, serving size $85g$, $9g$ fat, $0g$ carbs and $23g$ protein. However, the diet does not allow you to eat this steak because of the difference in protein ($23 - 2 = 21 > 5$). On the other hand, you can eat a German Berliner (a sugar-glazed, jelly-filled bun) which weighs $65g$, contains $13g$ of fat, $24g$ of carbs and $4g$ of protein.

You may use portions of your solution from A02 or the posted solution to A02, but you must indicate so in your submission with a comment. Place your functions in in the file `wonderdiet.rkt`.

3. Let us consider four-digit natural numbers. They appear for example in a personal identification number (PIN) for your VISA/Debit card. A *Four-Digit-Nat* can be represented by a structure with four fields (one for each digit). Then we have a minimal *Four-Digit-Nat* corresponding to 0000, and a maximal *Four-Digit-Nat* corresponding to 9999.

   A *Four-Digit-Nat* is a structure named *four-digit-nat*, which contains four fields:

   - e1 – the first digit (1000s).
   - e2 – the second digit (100s).
   - e3 – the third digit (10s).
   - e4 – the fourth digit (1s).

   a) Write the definition of a *Four-Digit-Nat*. You are not required to provide a template function.

   b) Write a function *next-num*, which consumes a *Four-Digit-Nat* and produces the next larger *Four-Digit-Nat*. If the consumed value is (*make-four-digit-nat* 9 9 9 9), your function shall produce (*make-four-digit-nat* 0 0 0 0).

   Please submit your code to this question as a file named `fourdigitenum.rkt`.

4. We learned about the *Posn* type in the lecture. Let us define some operations between *Posn*s in the two-dimensional plane.

   a) Given two *Posn*s (x_1, y_1) and (x_2, y_2). Then we define the product between these two *Posn*s in the following way:

   $$(x_1, y_1) \cdot (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1).$$

   Write a function *posn-mult*, which consumes two *Posn*s and produces a *Posn* that is the result of the multiplication as defined above.

   b) Given two *Posn*s (x_1, y_1) and (x_2, y_2). Then we define the division between these two *Posn*s in the following way:

   $$(x_1, y_1)/(x_2, y_2) = ((x_1 \cdot x_2 + y_1 \cdot y_2)/(x_2^2 + y_2^2), (y_1 \cdot x_2 - x_1 \cdot y_2)/(x_2^2 + y_2^2)).$$

   Write a function *posn-div*, which consumes two *Posn*s and produces the *Posn* that is the result of dividing the first parameter by the second, according to the above formula. You may assume that the user is not Chuck Norris and that they can not divide by zero, so ensure your contract enforces this by adding an appropriate requirement on the second parameter.

   c) Now we are going to see the benefit of defining – at first odd looking – operations on points in the two-dimensional plane: One can move a *Posn* (x, y) along the circle with radius $\sqrt{x^2 + y^2}$ and center $(0, 0)$ by an angle ϕ. This can be done quite easily using the multiplication operation: Just multiply the *Posn* by a *Posn* $(\cos\phi, \sin\phi)$. Write a function *rotate-along-circle* that consumes a number (angle) greater than or equal to 0 and less than 2π and a *Posn* (x, y). It shall produce a *Posn* moved by the angle along the circle with radius $\sqrt{x^2 + y^2}$.

   As we are dealing with inexact numbers here in general, you should use the *check-within* special form for your tests with an accepted margin of error of 0.001.

Please put your functions into a file called `complexmath.rkt`.

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.

Please put your solutions for this bonus question into a file called `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.*

Let us reflect on some of the questions:

Question 3: You will be surprised how often exercises similar to this one will occur in your programming career. But now let us stretch our thinking. Imagine you would not have a maximal value, i.e. all entries in this four-dimensional data-structure could be any non-negative integer. Clearly, the function *next-num* would be easy to implement in this case. But what about a function that finds the next smaller entry? Give it a thought.

Question 4: These – maybe strange looking – arithmetics on *posn* are not arbitrary. They are coming from so called "complex numbers". You may or may not have worked with complex numbers yet. You do not have to know anything about them other than the provided formulae given in the assignment text. But sit back for a moment and reflect what you have achieved in this exercise: You have written a very simple function that moves a point by a certain angle around a circle in a 2-dimensional plane. I dare you as an exercise to execute this task without using complex numbers or anything you have learnt in this assignment. You will quickly realise that things become more complicated. Think about what applications you might have concerning e.g. computer graphics. Summarising the take-home message: Never underestimate how math results can help you make your life as a programmer easier. They may look complicated to start with, but once you understand them, your tool-box of elegant tricks will be enormously expanded.

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.