Fall 2022
Becker, Kamal, Nasr Esfahani, Tompkins, Van Leeuwen, Vasiga

# Assignment: 03

| | |
|---|---|
| Due: | Tuesday, October 4, 2022 9:00 pm |
| Coverage: | Slide 14 of Module 06 |
| Language level: | Beginning Student |
| Files to submit: | `waterloo2-poker.rkt` |

- **Make sure you read the OFFICIAL A03 post on Piazza** for the answers to frequently asked questions.
- Unless otherwise specified, you may only use Racket language features we have covered up to the coverage point above (Slide 14 of Module 06). You may also use the builtin functions `second` and `member?` if you find them useful. As well, you have not seen recursion up to this point in the course, and it is not necessary nor recommended for this assignment.
- The names of functions we tell you to write, and symbols and strings we specify must match the descriptions in the assignment questions exactly. Any discrepancies in your solutions may lead to a severe loss of correctness marks. Basic test results will catch many, but not necessarily all of these types of errors.
- Policies from Assignment A02 carry forward, including:
  - For each function you are required to write, you are also required to submit the design recipe.
  - You are required to submit examples by **Friday, September 30 at 8:00AM** (before the due date of assignment A03).
  - More details about the design recipe grading are specified in the A02 Official Post on Piazza.

Here are the assignment questions you need to solve and submit.

1. In this question you will perform step-by-step evaluations of Racket programs, as you did in assignment one. Please review the instructions on stepping in A01.

   To begin, visit this web page:

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

   When you are ready, complete the three required questions under the "Module 6a: Lists" category, using the semantics given in class for Beginning Student.

2. You may be familiar with the card game *poker*. A poker hand is typically made up of five playing cards, and the most popular variants of poker use seven cards (where only five of the seven cards are used to make a hand). Three-card poker is also a popular game in some casinos.

   For this question, to keep things simple, we will be using a *two-card poker* variant known as *Waterloo*$^2$ *Poker*.

   *Waterloo*$^2$ *Poker* uses a standard (North American) 52-card deck. Each `Card` has a `Rank` and a `Suit`, and to represent them in Racket, we have provided some user-defined types:

```
;; A Rank is one of: 2, 3, 4, 5, 6, 7, 8, 9, 10, 'J, 'Q, 'K, 'A

;; A Suit is one of: 'C, 'D, 'H, 'S

;; A Card is a: (cons Rank (cons Suit empty))
```

   We have used short symbols to make your testing less onerous. For example, `'Q` is used to represent a *Queen*, and `'C` is used to represent *Clubs* so the following card: `(cons 'Q (cons 'C empty))` represents the playing card Queen of Clubs.

   The 52-card deck of `Card`s is composed of every possible combination of a `Rank` and `Suit` ($13 \times 4 = 52$).

   For now, we have defined a `Card` as a `list` with exactly two items. In Module 10 we will introduce structures, which can also be used to store multiple items. As we will see in that module, one advantage of using a structure is that you can give each field of the structure a meaningful name. However, we can achieve a similar effect with just lists by using a helper function to make your code easier to read:

```
;; suit: Card -> Suit
(define (suit c)
  (second c))
```

   The `Rank` of a `Card` can be either a number or a symbol. Fortunately, each `Rank` also has a corresponding `ordinality`, which will make comparing `Rank`s easier. For number ranks, the `ordinality` is the same as the number. The symbols `{'J, 'Q, 'K, 'A}` have the corresponding ordinalities of `{11, 12, 13, 14}`, respectively. Note that in some variants of poker, the `'A` (Ace) can have an `ordinality` of either 1 or 14, but in *Waterloo*$^2$ *Poker* the `'A` (Ace) always has an `ordinality` of 14 and there is no `Rank` with an `ordinality` of 1. The `ordinality` of a `Card` is simply the `ordinality` of the `Card`'s `Rank`.

A *Waterloo*$^2$ *Poker* `Hand` is a list with two unique `Card`s:

```
;; A Hand is a: (cons Card (cons Card empty))
;; requires: the two cards are not identical
```

**Important! A `Hand` is not a list of four items. It is a list of two `Card`s, and each `Card` is itself a list of length two.**

This question has been designed to help you become comfortable with working with lists.

Consider the following example:

```
(define ace-of-spades (cons 'A (cons 'S empty)))
(define ace-of-diamonds (cons 'A (cons 'D empty)))

(define good-hand (cons ace-of-spades (cons ace-of-diamonds empty)))
```

The value of `good-hand` is:

```
(cons (cons 'A (cons 'S empty)) (cons (cons 'A (cons 'D empty)) empty))
```

You should review this example carefully to make sure you understand a `Hand`.

In *Waterloo*$^2$ *Poker*, we need to determine the `strength` of a `Hand`, which will require some new terminology. In *Waterloo*$^2$ *Poker*:

- A *pair* is when both `Card`s have the same `Rank`.
  example: `good-hand`
- A *flush* is when both `Card`s have the same `Suit`.
  example: `(cons (cons 3 (cons 'H empty)) (cons (cons 'J (cons 'H empty)) empty))`
- A *straight* is when the `ordinality` of the `Card`s have a difference of exactly one.
  example: `(cons (cons 'J (cons 'S empty)) (cons (cons 'Q (cons 'D empty)) empty))`
- A *straight flush* is when the `Hand` is both a *straight* and a *flush*.
  example: `(cons (cons 'J (cons 'H empty)) (cons (cons 10 (cons 'H empty)) empty))`

In *Waterloo*$^2$ *Poker*, each `strength` is assigned a numerical value:

| Description | Strength |
|---|---|
| *straight flush* | 4 |
| *pair* | 3 |
| *straight* | 2 |
| *flush* | 1 |
| *none of the above* | 0 |

Note that the strengths in *Waterloo*$^2$ *Poker* are different than traditional 5-card poker, because the probability distributions are different.

When comparing two hands, the best `Hand` is the one with the larger `strength`. However, if two `Hand`s have the same `strength`, then the ordinalities of the `Hand`s must be compared.

Because each `Hand` has two `Cards`, it has a high `ordinality` (corresponding to the `Card` with the largest `ordinality`) and a low `ordinality`. Note that for a *pair*, the high and low ordinalities have the same value.

If two `Hand`s have the same `strength`, then the best `Hand` is the `Hand` with the largest high `ordinality`. If they have the same `strength` and the same high `ordinality`, then the best `Hand` is the `Hand` with the largest low `ordinality`. Two `Hand`s are equivalent if and only if they have the same `strength`, same high `ordinality` and the same low `ordinality`.

As a reminder, you may use the function you define in part (a) as a helper function for the function you define in part (b), *etc*.

(a) Write a function `ordinality` which consumes a `Card` $c$ and produces the ordinality of $c$.

(b) Write a function `strength` which consumes a `Hand` $h$ and produces the strength of $h$ as a number according to the table above (*i.e.,* one of: $0, 1, 2, 3, 4$).

(c) Write a predicate function `hand<?` which consumes two `Hand`s $h1$ and $h2$ and produces `true` if $h2$ is a better `Hand` than $h1$, and `false` otherwise.

(d) Write a function `winner` which consumes two `Hand`s $h1$ and $h2$ and produces `'hand1` if $h1$ is a better `Hand` than $h2$ or `'hand2` if $h2$ is a better `Hand` than $h1$, and `'tie` if both `Hand`s are equivalent.

(e) Normally in this course, you may assume that all arguments provided to a function will obey the contract. For example, in part (a) you should assume that the argument is a valid `Card`, and in parts (b)-(d) you should assume that each argument is a valid `Hand`. However, this part is an exercise in validating data.

Write a predicate function `valid-hand?` which consumes `Any` value $h$ and produces `true` if $h$ is a valid `Hand` that follows the data definition above, and `false` otherwise.