Winter 2025
Clarke

Assignment:	05
Due:	Tuesday, February 25, 2024 9:00 pm
Coverage:	L10
Language level:	Beginning Student with List Abbreviations
Allowed recursion:	Final version of the rules
Files to submit:	`sets.rkt`, `doudizhu.rkt`

Assignment policies:

- Make sure you read the official assignment post on **ed**.
- You may not use functions or language constructs from lectures after the "coverage" lecture listed above.
- Functions and symbols must be named **exactly** as they are written in the assignment questions. You may define helper functions, if needed.
- You must provide a purpose, contract, and appropriate test cases for all required functions, i.e. those we explicitly ask you to write.

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

1. **(46%)**: A data definition for a set of symbols is as follows:

```
;; A Set is a (listof Sym)
;; Requires: symbols must be unique
```

For example, both of these are a Set:

```
(list 'blue 'pizza 'idea)
empty ;; the empty set
```

Neither of these are a set (for different reasons):

```
(list 'blue 'pizza 'blue)
(list 'blue 1 'idea)
```

The order of symbols in the list doesn't matter. These are all the same set:

```
(list 'red 'green 'blue)
(list 'green 'blue 'red)
(list 'blue 'red 'green)
```

Remember that "`Set`" in a contract means you can assume that that argument is a Set. You do not need a separate "requires" statement.

In some cases, writing appropriate helper functions will greatly simplify your solutions. As always, you may use the solution for one part of the question to solve other parts of the question. You may re-use code from the slides.

Stick to the rules. Place your solutions to the following in `sets.rkt`.

(a) **(4%)**: Write a predicate that determines if an element is contained in a Set.

```
in?: Sym Set -> Bool
```

(b) **(4%)**: Write a function that adds an element to a Set.

```
add: Sym Set -> Set
```

(c) **(8%)**: Write a function that produces the union of two Sets.

```
union: Set Set -> Set
```

(d) **(8%)**: Write a function that produces the intersection of two Sets.

```
intersection: Set Set -> Set
```

(e) **(10%)** Write a predicate that determines if two sets are equal.

```
set=?: Set Set -> Bool
```

(f) **(12%)**: Write a predicate consumes anything and determines if it is a Set.

```
set?: Any -> Bool
```

2. **(54%)**: In this question, you will implement a function based on the game Dou Dizhu ("fight the landlord"). More questions based on this game will appear on future assignments; however, prior knowledge of the game is not required. We will provide all the necessary information in each question description as we go.

   Dou Dizhu is typically played by three people, with two of the players working together to defeat the third player (the "landlord"). While you don't need detailed knowledge of the game in order to complete this question, or the future questions, several sites explain the rules of the game, and free versions are available from the various app stores.

   - `https://www.pagat.com/climbing/doudizhu.html`
   - `https://gamerules.com/rules/dou-dizhu/`

The game uses an ordinary deck of playing cards, including the two jokers (a black joker and coloured "red" joker). Suits are ignored. All that matters is the value of the card: Ace, 2, 3, 4, ... The numbered cards are represented as their respective numbers. We will use the following symbols to represent the face cards and jokers: `'Ace`, `'Jack`, `'Queen`, `'King`, `'Black`, and `'Red`. In Dou Dizhu cards are valued from lowest to highest as follows:

`3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `'Jack`, `'Queen`, `'King`, `'Ace`, `2`, `'Black`, `'Red`

In this question, a `Card` is one of the numbers or symbols from the list above. Place your solutions to the following in `doudizhu.rkt`.

(a) **(2%)**: Write a function `card?` that consumes any value and produces `true` if the value is a valid `Card`; `false` otherwise.

(b) **(2%)**: Write a predicate `card=?` that consumes two `Card`s and produces `true` if they are the same card; `false` otherwise. Remember that `equal?` is not (and will never be) an allowed function in CS 135.

(c) **(2%)**: Write a predicate `card<?` that consumes two `Card`s and produces `true` if the first card has lower value than the second card according to the values above.

(d) **(16%)**: Write a function `sort-cards` that consumes a list of `Card` and produces a list of `Card` sorted in increasing Dou Dizhu order, as specified above. For example:

```
(check-expect (sort-cards
  (list 3 'King 6 7 'Jack 'Queen 2 7 3 7 3 'Ace 'Jack 2 3 4 5))
  (list 3 3 3 3 4 5 6 7 7 7 'Jack 'Jack 'Queen 'King 'Ace 2 2))
```

For the remainder of this question, we will call a sorted list of `Card` a `Hand`, even if it is not technically in a player's physical hand. `Hand` is just the name for a sorted list of `Card`.

(e) **(16%)**: Write a function `find-kind` that consumes a natural number `n` that is greater than or equal to 1 and a `Hand` (a sorted list of `Card`) and produces a `Hand` containing the card values with at least `n` occurrences in the consumed `Hand`. The produced list should be sorted with duplicates removed. For example:

```
(check-expect
  (find-kind
   3
   (list 3 3 3 3 4 5 6 7 7 7 'Jack 'Jack 'Queen 'King 'Ace 2 2))
  (list 3 7))
```

(f) **(16%)**: Write a function `remove-cards` that consumes two `Hand`s. The function removes the cards in the first `Hand` from the second `Hand`. Only one card should be removed from the second `Hand` for each card in the first `Hand`. It is okay if cards from the first `Hand` do not appear in the second `Hand`. For example:

```
(check-expect
  (remove-cards (list 4 5 6 7 8)
                (list 3 3 4 5 6 6 6 7 9 ))
  (list 3 3 6 6 9))
```

In answering these question you can assume all cards come a single deck of standard playing cards, e.g., there will be at most one black joker in the hand and no more than four jacks. You never need to explicitly check this condition. In a real game, a hand can never be larger than 20 cards, but you need not consider that limitation in your solutions, i.e., you never need to explicitly check the length of a list of cards.

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