Assignment: 6
Due: Tuesday, March 10, 2020, 9:00pm
Language level: Beginning Student with List Abbreviations
Files to submit: debug-a06.rkt, pokemon.rkt, a06-bonus.rkt
Warmup exercises: HtDP 13.0.3, 13.0.4, 13.0.7, 13.0.8, 14.2.1, 14.2.2, 17.2.1, 17.3.1, 17.6.4, and 17.8.3
Practice exercises: HtDP 12.4.1, 12.4.2, 13.0.5, 13.0.6, 14.2.3, 14.2.4, 14.2.6, 17.1.2, 17.2.2, 17.6.2, 17.8.4, and 17.8.8

• Make sure you read the OFFICIAL A06 post on Piazza for the answers to frequently asked questions.
• Unless stated otherwise, all policies from Assignment 05 carry forward.
• This assignment covers material up to the end of Module 10, slide 21.
• Of the built-in list functions, you may use only cons, first, second, third, rest, empty?, cons?, list, member?, string->list, list->string, and length.
• You may only use the functions that have been discussed in the lecture slides, unless explicitly allowed or disallowed in the question. In particular, you may not use any of the following Racket functions: reverse, make-string, replicate, or list-ref. You may define your own versions of these functions as long as they are written using simple or accumulative recursion.
• Solutions will be marked for both correctness [80%] and style [20%]. Follow the guidelines in the Style Guide, pages 1-17 (skipping sections 3.7.2 and 3.7.4 for now).

1. [20% Correctness] Question 1
Consider the code in debug-a06.rkt, which may contain syntax, mathematical, and style errors. The code may also be difficult to read and/or maintain, lack helper functions and constants. Correct the code. Your solution MUST use member?.

2. [60% Correctness] Question 2
A poke-card is a structure that has the following fields:
• name - the name of the Pokemon, a string
• hp - the number of hit points, a natural number
• type - the primary type of the Pokemon, a symbol
• weakness - the type of Pokemon that will cause extra damage, a symbol
• attack - the base amount of damage this Pokemon can inflict, a natural number \( \geq 1 \)

Note: a card’s type cannot be the same as its weakness.

(a) Create a structure definition for a structure named 'poke-card' with the above field names. Then, create a data definition for a Poke-Card type that describes a poke-card structure with the above restrictions.

(b) Write a function \( \text{compute-damage} \) \( \text{card1} \) \( \text{card2} \) that consumes two Poke-Cards and computes the amount of damage \( \text{card1} \) will do to \( \text{card2} \). Note: if \( \text{card1} \)'s type is equal to \( \text{card2} \)'s weakness, then \( \text{card1} \) will deal double damage to \( \text{card2} \). And, if \( \text{card1} \)'s type is equal to \( \text{card2} \)'s type, then halve damage.

For example, if \( \text{card1} \) is a 'fire' type and \( \text{card2} \) is an 'ice' type with weakness 'fire', then \( \text{card1} \) does double damage to \( \text{card2} \).

Note: round-up non-Nat damage using the built-in \text{ceiling} \) function.

(c) Write a function \( \text{compute-attack} \) \( \text{card1} \) \( \text{card2} \) that consumes two Poke-Cards and produces the Poke-Card that results from \( \text{card1} \) attacking \( \text{card2} \).

Note: The produced card should be identical to \( \text{card2} \) except for \( \text{hp} \). If the damage is greater than the \( \text{hp} \), then set \( \text{hp} \) to 0.

(d) Write a function \( \text{compute-winner} \) \( \text{card1} \) \( \text{card2} \) that consumes two Poke-Cards and determines which Poke-Card is the winner if \( \text{card1} \) attacks first. This function should produce a three-element list: \( \langle \text{WINNER} \ \text{ORIGCARD} \ \text{CARD} \rangle \) where \( \text{WINNER} \) is (anyof 'you 'opponent), \( \text{ORIGCARD} \) is the winning card before the battle, and \( \text{CARD} \) is the winning card with \( \text{hp} \) reduced to reflect damage received during the battle.

Here is a helpful data definition you may use:

\[
\text{A Winner is a (list (anyof 'you 'opponent) Poke-Card Poke-Card)}
\]

You can assume that at least one of the cards will have positive \( \text{hp} \). As a result, the battle will never end in a draw.

Notes:

• the two cards attack in turn until one card reaches 0 \( \text{hp} \) (making the other card the winner)

Hint: break this problem into smaller sub-problems. You will find the format of the solution handy in later functions.

(e) Write a function \( \text{find-best-card} \) \( \text{opp-card} \) \( \text{cards} \) that consumes the opponent’s Poke-Card that they have played and a non-empty list of Poke-Cards (that you have) and produces a Poke-Card from your list that will win and have the most \( \text{hp} \) left after the battle, or, if no winning cards exist, produces the Poke-Card from your list that will result in the opponent’s card having the lowest \( \text{hp} \) after the battle. If there is more than one card in your hand that has the desired result, produce the one that occurs first.
Note: assume that the opponent attacks first and that there are no duplicate cards in your hand.

Hint: compute the winners between the opponent’s card and each of your cards as a list. Then search the produced list to find the desired information.

(f) Write a function (will-win? opponent player) that consumes two hands of cards—your opponent’s and yours and produces true if you will win if you repeatedly play your best available card. Each hand of cards contains only three cards. Assume your opponent plays first and that they play their cards in order.

The game is played as follows:

i. Your opponent plays the first card in their hand.
ii. You choose your best available card (according to find-best-card) to play first.
iii. Your opponent attacks first, followed by you until there is a winner.
iv. Both the winning and losing cards are removed from their respective hands.

The winner is the player that wins a minimum of 2 out of 3 matches.

Suggestions:

i. Use the functions that you created in the previous parts to help you.
ii. For each of the opponent’s cards, in order, determine which of your cards to play.
iii. After each “match”, remove the card you played from your hand. Then repeat the process, but this time your hand is smaller!
iv. Keep track of the winners in a list—0 means you lose, 1 means you win. If the sum of the list is greater than or equal to 2, then you win!
v. The strategy for battling in this question is not necessarily the optimal one.

Place your solutions in a file pokemon.rkt.

---

**Bonus Question [5%]**

Write a function (runlengths str) that consumes a string and produces a list of run lengths. A run length is the number of consecutive characters that are the same. For example, the run length of "aaabcddeeeea" is (list 3 1 1 2 5 1 1). It may be assumed that str is a non-empty string.

Put your solution in a file called a06-bonus.rkt.

---

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

The IEEE-754 standard, most recently updated in 2008, outlines how computers store floating-point numbers. A floating-point number is a number of the form

\[0.d_1d_2\ldots d_t \times \beta^e\]
where $\beta$ is the base, $e$ is the integer exponent, and the mantissa is specified by the $t$ digits $d_i \in \{0, \ldots, \beta - 1\}$. For example, the number 3.14 can be written

$$0.314 \times 10^1$$

In this example, the base is 10, the exponent is 1, and the digits are 3, 1 and 4. A computer uses the binary number system; the binary equivalent to the above example is approximately

$$0.11001001 \times 2^{10}$$

where $10_2$ is the binary integer representing the decimal number 2. Thus, computers represent such numbers by storing the binary digits of the mantissa (“11001001” in the example), and the exponent (“10” in the example). Putting those together, a ten-bit floating-point representation for 3.14 would be “1100100110”.

However, computers use more binary digits for each number, typically 32 bits, or 64 bits. The IEEE-754 standard outlines how these bits are used to specify the mantissa and exponent. The specification includes special bit-patterns that represent Inf, −Inf, and NaN.