Assignment: 6
Due: Tuesday, March 1st, 9:00 pm
Language level: Beginning Student with list abbreviations
Allowed recursion: Simple Recursion and Accumulative Recursion
Files to submit: movie-list.rkt, settheory.rkt, checksum.rkt, acadinteg-a06.txt
Warmup exercises: HtDP 6.2.5, 6.3.2, 17.1.2, 17.2.2, 17.3.1
Practice exercises: HtDP 6.5.2, 17.6.6

- 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.
- Of the built-in list functions, you may use only cons, first, second, rest, empty?, cons?, list, member?, and append. You may also use equal? to compare two lists.
- This assignment covers material up to Slide 31 of Module 10.
- Remember that basic tests are meant as sanity checks only; by design, passing them should not be taken as any indication that your code is correct, only that it has the right form.
- Unless the question specifically says otherwise, you are always permitted to write helper functions. You may use any constants or functions from any part of a question in any other part.
- Download the file acadinteg-a06.txt from the assignment page. Fill in your Quest userid in the provided place to indicate that you have read and understood the policy. We take this to be equivalent to signing the document, so save a copy for your own records.
- Solutions will be marked for both correctness [75%] and style [25%]. Follow the guidelines in the Style Guide.
You have decided to create your own movie rating system, and after some thinking you decide to create a structure (and an associated data definition) to hold information about a movie:

(define-struct movie (name director genre prod-year imdb-score my-score))

;; A Movie is a (make-movie Str Str Sym Nat Num Num)
;; requires: 0 <= imdb-score <= 10
;; 0 <= my-score <= 10

Since you’re interested in tracking a large collection of all the movies you’ve watched, you also come up with a data definition for a list of movies:

;; A MovieList (ML) is a (listof Movie)
;; requires: no two Movies have the same name, director, and year of production

(a) Design a template for Movie (named movie-template) and then design a template for MovieList (ML) (named movielist-template). You should NOT comment out these templates.

(b) Implement the functions below (It is highly recommended you do part (a) first as your templates will be very useful for understanding and solving the following questions). The list produced by any of the search-by-X functions should maintain the order of elements from the movie list in the argument.

• add-movie: consumes a movie list (ML) and a movie structure, in that order, and if there is an other movie with the same name, by the same director, and from the same year, it produces false; otherwise, adds it to the list of movies. The movie can be added anywhere in the list.

• remove-movie: consumes a movie list (ML) and movie name, director name, and year of production, in that order, and produces false if no movie in the list matches those requirements, otherwise removes the movie with fields that matches arguments.

• search-by-name: consumes a movie list (ML) and a movie name (Str), in that order, and produces a list of all the movies in your list with that name.

• search-by-director: consumes a movie list (ML) and a director name (Str) and produces a list of all the movies in your list with that director name.

• search-by-genre: consumes a movie list (ML) and a genre (Sym), in that order, and produces a list of all the movies in your list in that genre.

• search-by-year: consumes a movie list (ML) and a year (Nat), in that order, and produces a list of all the movies in your list made in that year.

• search-by-imdb-score: consumes a movie list (ML) and a score (Num), in that order, and produces a list of all the movies with IMDB scores equal to or higher than that.
• **search-by-my-score**: consumes a movie list (ML) and a score (Num), in that order, and produces a list of all the movies with your personal scores equal to or higher than that.

(c) Now, write a function name `search` that consumes a criteria list, which is a `(listof (anyof 'name 'director 'genre 'year 'imdb-score 'my-score))`, and a keyword list, which is a `(listof (anyof Str Sym Nat Num))` and is of the same length as the criteria list. This function should produce the list of movies that match the keywords in the specified criteria. **The list produced should maintain the order of elements from the movie list in the argument.** Below are some examples of search application. To help you understand this function better, you can see examples in file `movie-list-example.rkt`.

(d) Define a helper function that consumes a MovieList (ML) and produces the average personal score of those movies (or produces "No such movie found!" if the list of movies is empty). Then, define the following wrapper functions:

  - **average-by-name**: consumes a movie list (ML) a movie name (Str), in that order, and produces the average score of all the movies in your list with that name.
  - **average-by-director**: consumes a movie list (ML) a director name (Str), in that order, and produces the average score of all the movies in your list with that director name.
  - **average-by-genre**: consumes a movie list (ML) a genre (Sym), in that order, and produces the average score of all the movies in your list in that genre.
  - **average-by-year**: consumes a movie list (ML) a year (Nat), in that order, and produces the average score of all the movies in your list made in that year.
  - **general-average**: that consumes a movie list (ML) a criteria list, which is a `(listof (anyof 'name 'director 'genre 'year 'imdb-score 'my-score))`, and a keyword list, which is a `(listof (anyof Str Sym Nat Num))` and is of the same length as the criteria list, in that order. This function should produce the average score of movies that match the keywords in the specified criteria.

Place your solutions in the file `movie-list.rkt`.

2. [7+7+7+7=28% Correctness] It’s possible to represent a set of numbers using a list, as long as the list does not contain any duplicate entries. Standard set-theoretic operations like the union, intersection and difference of two sets can then be written by comparing every element of one list against every element of the other.

   If we instead demand that sets are represented by sorted lists of numbers, then all of these operations can be computed more elegantly and more efficiently, by using an approach modeled on the example in the notes of merging two lists. With that in mind, we make the following data definition:
For example, if a set $A$ is represented by the list `(list 1 2 3 6)` and a set $B$ is represented by the list `(list 1 3 4)`, then we’d expect that the union of $A$ and $B$ would be `(list 1 2 3 4 6)`, the intersection would be `(list 1 3)`, the difference $A - B$ would be `(list 2 6)`, the difference $B - A$ would be `(list 4)`, and the symmetric difference of $A$ and $B$ (sort of like the exclusive-or: elements in one set or the other, but not both) would be `(list 2 4 6)`. In all cases, the results are sorted lists with no duplicates.

To begin, copy the data definition above into a new file called `settheory.rkt`.

(a) Write a function `union` that consumes two parameters of type `NumSet` and produces a single `NumSet` containing all the numbers that are in either of the consumed sets. You must base your solution on the form of the `merge` function in the course notes, making a single pass over the two lists. You must write a single simple recursive function—do not use any helper functions as part of your solution. The only built-in list functions you may use are `first`, `rest`, `empty?`, `cons?`, and `cons`.

(b) Write a function `intersection` that consumes two parameters of type `NumSet` and produces a single `NumSet` containing all the numbers that are in both of the consumed sets. The same constraints described in the previous sub-question apply here.

(c) Write a function `difference` that consumes two parameters of type `NumSet` and produces a single `NumSet` containing all the numbers that are in the first set but not the second one. For example, we would expect `(difference (list 1 3 4 5) (list 1 2 3))` to produce `(list 4 5)`. The same constraints described in the previous sub-questions apply here.

(d) Write a function `symmetric-difference` that consumes two parameters of type `NumSet` and produces a single `NumSet` containing all the numbers that are in one set or the other, but not both. Unlike the previous sub-questions, your solution should not call itself recursively; instead, use a combination of applications of some or all of `union`, `intersection` and `difference` to obtain the symmetric difference.

Note: you must not use any helper functions outside of the previous sub-questions.

Place your solutions in the file `settheory.rkt`.

3. **[10+10 = 20% Correctness]** Universal Product Codes (UPCs) are widely used to identify products for sale. Normally, a UPC \(^1\) consists of 12 digits, but we will extend this concept to consider codes of any length greater than or equal to two. In a UPC of length $n$, the $n$th digit is a checksum, which is computed based on the values of the other digits. The checksum helps identify common transcription errors.

The checksum digit is computed according to the following algorithm:

\(^1\)Technically, this is the UPC-A standard. There are others.
Take the sum of all the odd positions of the UPC (so the first digit, the third, and so on), not including the checksum digit. Multiply this result by 3.

Take the sum of all the even positions of the UPC, not including the checksum digit.

Add the two sums together, and take the result modulo 10 (that is, the remainder when divided by 10).

If the result is 0, the checksum digit is 0. Otherwise, if the result is \( x \), the checksum digit is \( 10 - x \).

For example, a UPC label where the first 11 digits (not including the checksum digit) are 66788809152 should have a checksum digit of 2, because
\[
3 \cdot (6 + 7 + 8 + 0 + 1 + 2) + (6 + 8 + 8 + 9 + 5) = 3 \cdot 24 + 36 = 72 + 36 = 108.
\]
Taking the modulo base 10 gives 8, and \( 10 - 8 = 2 \).

We will represent a UPC as a non-empty \((\text{listof Nat})\), where each element of the list is between 0 and 9. So the above UPC (including the checksum) would be represented as \((\text{list } 6 \ 6 \ 7 \ 8 \ 8 \ 0 \ 9 \ 1 \ 5 \ 2 \ 2)\).

Place your answers to this question in checksum.rkt.

(a) Write a function `valid-checksum-acc?` which consumes the representation of a UPC (including the checksum digit) and produces true if and only if the checksum digit is valid for this UPC. You should use accumulative recursion in your solution.

(b) Write a function `valid-checksum-simple?` which is the same as `valid-checksum-acc?` but uses only simple recursion.

**Hint:** Recurse on a natural number in addition to recursing on a list, so that you do not need to “toggle” a parameter (which would make your functions generative).

This concludes the list of questions for which you need to submit solutions. Always remember to check your basic test results by email or on MarkUs after making a submission.