Assignment: 9

Due: Tuesday, July 28 at 4:00 pm
Language level: Intermediate Student
Coverage: Modules 1–10

This assignment is worth 1.5 times the amount of assignments 2–8. To receive full marks you are required to use the design recipe for every function you write. You may include the examples given in the assignment in your submissions, but they will be ignored by the markers—you must develop a complete suite of examples and tests on your own. For your convenience, an interface file which contains the headers of the required functions is available on the course webpage.

Do not send any code files to course staff; they will not be accepted. Submissions must be made via MarkUs as described on the course webpage. After submission, check your basic test results to ensure your files were properly submitted. Solutions that do not pass the basic tests are unlikely to receive any correctness marks.

Remember, the solutions you submit must be **entirely your own work**.

1. These questions make use of the following data definitions:

```
;; A SortedList is a (listof Num)
;; requires: every number in the list is distinct and the numbers are sorted in ascending order

;; An As (association) is a (list Num Str), where
;;    the first element is known as the key
;;    the second element is known as the value

;; An AL (association list) is a (listof As)
;; requires: all keys are distinct

;; A Grade is a Nat which is less than or equal to 100
```

   To get practice with **local** expressions, all helper functions that you define must be local. If you are working on this assignment before this material has been covered, you can define helper functions like normal and later update them to be local once that material has been covered. It is recommended that you do this anyway, since this allows you to thoroughly test that your helper functions are behaving correctly before using them (as you have been previously taught to do).

   Except for the usage of **local**, your solutions should only use material from before module 10. In particular, question 1 must be completed **without using** abstract list functions.

(a) Write a Racket function *difference* which consumes two *SortedLists* and produces a *SortedList* containing all the elements of the first list which are not elements of the second list.

For example, (*difference* (*list* 1 2 3) (*list* 2 4)) should produce (*list* 1 3).

(b) Write a Racket function *intersection* which consumes two *SortedLists* and produces a *SortedList* containing all the elements which appear in both lists.

For example, (*intersection* (*list* 1 2 3) (*list* 2 4)) should produce (*list* 2).

(c) Write a Racket function *count-val* which consumes a string and an *AL* and produces the number of associations in the list whose value is equal to the given string.

For example, (*count-val* `"A"` (*list* (*list* 1 `"A"`) (*list* 2 `"B"`) (*list* 3 `"A"`) (*list* 4 `"C"`))) should produce 2.

(d) Write a Racket function *terminals* which consumes a list of nonempty strings and produces a string containing the last character of every string in the list. The ordering of the characters in the produced string should match the ordering of the strings that they were extracted from in the given list.

For example, (*terminals* (*list* `"abc" "de" "d" "de"`)) should produce `"cede"`.

(e) Write a Racket function *nonzero-average* which consumes a list of *Grades* and produces the average of the nonzero grades in the list. You can assume that there will be at least one nonzero grade in the list.

For example, (*nonzero-average* (*list* 0 50 90 0 100 80)) should produce 80.

2. Write each of the functions from question 1 again; this time you **must not use recursion** in the functions you write (both in the main functions and any helper functions). Instead, your solutions must use abstract list functions as discussed in module 10.

   Like in question 1, any helper functions that you define must be local. You may re-use the parts of the design recipe from your original solutions which apply to both questions (such as the purpose statements, contracts, examples, and tests).