CPSC 326: Homework 7 *Due: Apr. 25*

Goals:

- Gain practice writing basic OCaml functions
- Gain practice using pattern matching and guards

Instructions:

1. Use the GitHub Classroom link (posted in Piazza) to copy the starter code into your own repository. Clone the repository in the directory where you will be working on the assignment.
2. Write the functions described below in `hw7a.ml` and `hw7b.ml`. As you write each function be sure to test each one.
3. Keep track of the test cases you used for each function (and include each test in your HW writeup).
4. Create a write up as a **pdf file** named `hw7-writeup.pdf`. For this assignment, your write up should provide a short description of the tests you used for each function and any challenges and/or issues you faced in finishing the assignment and how you addressed them.
5. Submit your files to your repository. Ensure all of your code and writeup is pushed to your GitHub repo. You can verify that your work has been submitted via the GitHub page for your repo.

Additional Requirements: Note that in addition to items listed below, details will also be discussed in class and in lecture notes. You must implement the functions below *from scratch* in `hw7a.ml` and `hw7b.ml`. You are implementing the same functions in `hw7a.ml` and `hw7b.ml`, however:

> *only* `if-then-else` *constructs are allowed as part of the function definitions in* `hw7a.ml` *(i.e., no pattern matching and guards) whereas only pattern-matching and guards are allowed as part of the function definitions in* `hw7b.ml` *(i.e., no* `if-then-else` *constructions).*

In addition:

- Only use the OCaml constructs we've discussed so far in class or as provided in the function description (if you go beyond what we've done, you'll receive no points for the question);
- In general, you are allowed to use comparison operators, logical operators, arithmetic operators, lists (but not the `List` module), cons (`::`), append (`@`), recursion, `failwith`, and let-in bindings.
- You may use the `head`, `tail`, and `length` helper functions in `hw7a.ml`, however, these are *not* allowed in `hw7b.ml`;
- You must format and indent your code.

The following functions must be implemented as stated above. If you have questions on how any of the following are supposed to work, please ask (preferably in piazza). Note that many of the examples use integer lists, but the functions can support other list types as well.

1. Write a function `my_min` that returns the smallest of a given *list* of values. Example: `my_min [7; 1; 9; 12; 10]` should return `1`. The function should report failure ("Empty List") when called on an empty list. Be careful with respect to efficiency, i.e., your implementation *must* be $O(n)$ for an n-element list.

2. Write a function `my_reverse` that takes a list and returns the reverse order of the list. Example: `my_rev [1; 2; 3]` should return `[3; 2; 1]`, `my_rev ['a']` should return `['a']`, and `my_rev []` should return `[]`.

3. Write a function `my_take` that takes an integer n and a list and returns a new list containing the first n elements of the input list. Examples: `my_take 0 [1; 2; 3]` should return `[]`, `my_take 1 [1; 2; 3]` should return `[1]`, and `my_take 2 [1; 2; 3]` should return `[1; 2]`. If $n \leq 0$ the empty list should be returned. If n is the length of the list or larger, then the entire list should be returned.

4. Write a function `my_drop` that takes an integer n and a list and returns a new list that is the same as the given list but with the first n elements removed. Examples: `my_drop 0 [1; 2; 3]` should return `[1; 2; 3]`, `my_drop 1 [1; 2; 3]` should return `[2; 3]`, and `my_drop 2 [1; 2; 3]` should return `[3]`. If $n \leq 0$ then the original list should be returned. If n is the length of the list or larger, then the empty list should be returned.

5. Write a function `my_set` that takes an integer index i, a value x, and a list and returns a copy of the given list with the i-th element changed to x. Examples: `my_set 0 4 [1; 2; 3]` should return `[4; 2; 3]`, `my_set 1 0 [1; 2; 3]` should return `[1; 0; 3]`, and `my_set 2 5 [1; 2; 3; 2]` should return `[1; 2; 5; 2]`. If the index i is invalid (i.e., too small or too large), then an "Invalid Index" exception should be raised. Your implementation cannot use the `my_take` and `my_drop` functions.

6. Write a function `my_init` that takes a list and returns a new list containing all of the elements of the input list except for the last element. Example: `my_init [1; 2; 3]` should return `[1; 2]`. Calling `my_init` on an empty list should result in a failure ("Empty List").

7. Write a function `kv_build` that takes a list of keys (of any type) and a list of values (of any type) and creates a corresponding list of key-value pairs. Examples: `kv_build ['a'; 'b'] [1; 2]` should return `[('a', 1), ('b', 2)]` and `kv_build [1; 2; 3; 2] [true; false; true; true]` should return `[(1, true), (2, false), (3, true), (2, true)]`. If one of the lists is smaller than the other, only that many (the smaller number of) key-value pairs should be returned.

8. Write a function `kv_key` that takes a key k and a list of key-value pairs and returns true if there is a pair whose key is k and false otherwise.

9. Write a function `kv_remove` that takes a key k and a list of key-value pairs and removes all pairs that have the key k.

10. Write a function `kv_collect` that takes a list of key-value pairs and returns the unique set of keys with their corresponding list of values. Example: `kv_collect [('a',1), ('b',2), ('a',3), ('c', 4)]` should return `[('a', [1,3]), ('b', [2]), ('c', [4])]`. Your function may (and should) call the `kv_remove` function. Hint: It is useful to create a local "helper" function using a let-in binding within `kv_collect`. For `hw7b.ml`, the let-in helper and the rest of `kv_collect` must not use if-then-else constructs.

Homework Submission and Grading. Your homework will be graded using the files you have pushed to your GitHub repository. Thus, you must ensure that all of the files needed to compile and run your code have been successfully pushed to your GitHub repo for the assignment. Note that this also includes your homework writeup. This homework assignment is worth a total of 20 points. The points will be allocated based on the correctness and completeness of your solution to the above functions. Note that you must also include evidence of testing and your writeup. *Assignments without a write up and without discussion of thorough testing will not receive a grade.*