CPSC 326: Homework 8 *Due: Tuesday, Apr. 13*

Reading Assignment. Read the following sections in the textbook:

- Sections 8.3–8.7: Skim as needed
- Sections 8.8–8.11: Type Checking, Safety, Garbage Collection
- Sectoin 11.2, 11.3: Evaluation and Functional Programming

Programming Homework. The goal of this assignment is to write basic list functions in OCaml. You will write and test each of the functions twice: once *without* pattern matching in a file `hw8a.ml` and once *with* pattern matching in a file `hw8b.ml`. You must write each function "from scratch" as specified below and using only what we have discussed in class. Each function should use recursion and should not call similarly purposed functions that are already provided in OCaml. In particular:

- For `hw8a.ml`, besides `let` bindings, `let-in` expressions, `failwith`, if-then-else, and comparison and arithmetic operations, you can only use the functions `List.tl` (tail), `List.hd` (head), `List.length` (length), `@` (append), and `::` (cons).
- In `hw8b.ml`, you **cannot use** if-then-else, `List.tl`, and `List.hd`.

Finally, you must:

- follow the general style guide provided by OCaml (https://ocaml.org/learn/tutorials/guidelines.html)
- appropriately comment your code throughout including a file header with your name, file name, the date, and a brief description; and
- create sufficient test cases for your functions to ensure they work correctly, including error cases as needed.

The following ten functions must be implemented as stated above. If you have questions on how any of the following are supposed to work, please ask either during class or on piazza:

1. Write a function `my_rev` that takes a list and returns the reverse order of the list. Example: `my_rev [1; 2; 3]` should return `[3; 2; 1]`. Note that this function does not require guards.
2. Write a function `my_last` that takes a list and returns the last element of the list. Example: `my_last [1; 2; 3]` should return `3`. Calling `my_last` on an empty list should result in a failure ("Empty List"). Note that this function does not require guards.
3. 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"). Note that this function does not require guards.

4. Write a function `my_mem` that takes a value and a list and returns true if the value is in the list, and false otherwise. Examples: `my_mem 3 [1; 2; 3; 4]` should return true whereas `my_mem 3 [1; 2; 4; 5]` should return false.

5. Write a function `my_replace` that takes a pair of values and a list and returns a new list such that each occurrence of the first value of the pair in the list is replaced with the second value. Example: `my_replace (2,8) [1; 2; 3; 2]` should return `[1; 8; 3; 8]`.

6. Write a function `my_replace_all` that takes a list of pairs and a list of values and returns a new list where each occurrence of the first value in a pair is replaced by the second value in the pair. The replacement should occur in order of pairs. Examples: `my_replace_all [('a','b'), ('c','d')] ['a'; 'b'; 'c'; 'd']` should give `['b'; 'b'; 'd'; 'd']"` and `my_replace_all [(1,2); (2,3)] [1; 2; 3; 4]` should give `[3; 3; 3; 4]`. You can call `my_replace` from within `my_replace_all`. Note that you do not need guards to define this function.

7. Write a function `my_elem_sum` that takes a value and a list, and returns the sum of the given values in the list. Examples: `my_elem_sum 10 [15; 10; 25]` should return `10`, `my_elem_sum 3 [3; 2; 3; 2; 3; 4; 3]` should give `12` and `my_elem_sum 3 []` should give 0.

8. Write a function `my_rem_dups` that takes a list of values, and returns a copy of the original list with duplicate values removed. Examples: `my_rem_dups ['a'; 'b'; 'a'; 'c'; 'b'; 'a']` should return `['c'; 'b'; 'a']` and `my_rem_dups [10; 11; 13; 11; 12]` should return `[10; 13; 11; 12]`. Note you can call `my_mem` within your `my_rem_dups` function.

9. 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.

10. Write a `merge_sort` function that takes a list and returns a sorted copy. Your `merge_sort` function must be "self-contained", i.e., you can write helper functions, but they must be defined within `merge_sort`. In my implementation, e.g., I defined a `drop`, `take`, and `merge` function within my `merge_sort` function definition.

For testing, we'll use the same approach as in HW-7. For testing error cases, you can do the following to ensure an exception is thrown:

```
(* Question 2: my_last tests *)
assert_equal () (try my_last [] with _ -> ())
  "my_last []" ;;
```

Finally, you **must** write the function types in a comment for each problem. For example:

```
(* Question 2: my_last : 'a list -> 'a *)
let my_last xs =
  ...
```

Homework Submission. All homework must be submitted through GitHub Classroom. A link for each assignment will be posted on Piazza when the homework is assigned. Be sure all of your code is pushed by the due date (which you can double check for your repository using the GitHub website). Each programming assignment is worth **35 points**. The points are allocated based on the following.

- **Correct and Complete (25 points)**. Your code must correctly and completely do the requested tasks using the requested techniques. Note that for most assignments you will be provided a *partial* set of test cases to help you determine a *minimal level* of correctness. If your program fails any of the provided test cases you will only receive partial credit. *Note that passing the given test cases does not mean your work is complete nor correct.* Your assignment will also be graded with additional test cases (not provided to you) that will help the graders determine the extent of your solution and your final score. Note that for C++ code, correctness also implies properly handling the creation and deletion of dynamic memory (i.e., the absence of memory leaks).

- **Evidence and Quality of Testing (5 points)**. As part of your homework assignments you must develop additional test cases beyond those given to you to ensure your program is correct and complete. These test cases must be turned in with your assignment. You will be graded on the scope and quality of the additional test cases you provide.

- **Formatting and Comments (5 points)**. Your code must be formatted consistently and appropriately for the language used. For C++, you must follow the provided style guide (see the course webpage).You must also comment your code and test cases, which at a minimum must include a file heading (see examples provided), function comments, and meaningfully selected variable, class, and function names. See the assignment for style guides in other languages.