CPSC 326: Homework Assignment 8 *Due: Thursday, Apr. 27*

The goal of this assignment is to gain practice writing basic OCaml functions. Note that you are free to use whatever IDE and machine you prefer for this assignment. However, to complete the assignment, you will need to download and install OCaml and/or use the the remote development server (`ada.gonzaga.edu`), which has OCaml installed already.

Overview:

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 (e.g., onto `ada` or your own machine).
2. Write the functions described below in `hw8.ml`.
3. Write multiple tests for each function to ensure correctness within the `hw8_tests.ml` file.
4. Submit your two program files. Be sure to add, commit, and push all assignment files to your GitHub repo. You can verify that your work has been submitted via the GitHub page for your repo.

Instructions: Implement the following functions *from scratch* in `hw8.ml` (i.e., without just calling functions provided by OCaml). In addition you must:

- 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);
- only use `if-then-else` constructs for conditionals (i.e., you *cannot* use pattern matching and guards);
- 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.

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 either during class or on piazza:

1. Implement a function `my_min x y` that evaluates to the minimum of the values `x` and `y`. For example, `my_min 2 3`, `my_min 4 2`, and `my_min 2 2` should each evaluate to `2`. You cannot use any other functions in your implementation.
2. Implement a function `my_median x y z` that evaluates to the middle value of `x`, `y`, and `z`. For example, `my_median 1 2 3`, `my_median 3 1 2`, and `my_median 2 3 2` should each evaluate to `2`. In addition, `my_median 2 2 2` and `my_median 2 1 2` should each also evaluate to `2`. You cannot use any other functions in your implementation.

3. Implement a function `my_triangle_area base height` that computes the triangle height given a base and a height value (as floats). You cannot use any other functions in your implementation.

4. Implement a function `my_circle_area radius` that computes the area of a circle given the radius (as a float). Note that to raise a float value to a power you can use the exponentiation operator `**`, e.g., `v ** 3.0` would cube the value `v`. You cannot use any other functions in your implementation.

5. Implement a function `my_midpoint (x1,y1) (x2,y2)` to compute the midpoint between two points. Assume the points are given as floats. For example `my_midpoint (1.,1.) (3.,5.)` would evaluate to `(2.,3.)`. Note this function uses 2-tuples of floats to represent points. You cannot use any other functions in your implementation.

6. Implement a function `my_manhattan_distance (x1,y1) (x2,y2)` to compute the "Manhattan" (i.e., "taxicab") distance between two points (defined as the absolute distance travelled in the x-axis plus the absolute distance travelled in the y-axis, as if you are walking city blocks to get from location 1 to location 2). Assume the points are given as floats. For example `my_manhattan_distance (1.,1.) (3.,4.)` and `my_manhattan_distance (3.,1.) (1.,4.)` would both evaluate to `5.`. To compute an absolute float value, you can use the OCaml `abs_float` function. You cannot use any other functions in your implementation.

7. Implement a function `my_euclidean_distance (x1,y1) (x2,y2)` to compute the "Euclidean" distance (i.e., the straight-line distance from location 1 to location 2) between two points. For example, `my_euclidean_distance (1.,1.) (4.,5.)` should evaluate to `5.`. To take the square root, you can use the OCaml `sqrt` function. You cannot use any other functions in your implementation.

8. Implement a function `my_range_sum v1 v2` that returns the sum of values starting at `v1` and ending at `v2`. For example, `my_range_sum 2 2` is `2`, `my_range_sum 2 3` is `5` (i.e., `2 + 3`), `my_range_sum 2 4` is `9` (i.e., `2 + 3 + 4`), and `my_range_sum 2 1` is `0`. You must write this as a recursive function. You cannot use any other functions in your implementation.

9. Implement a function `my_gcd x y` that returns the greatest common divisor of two integers `x` and `y`. You must use the recursive division-based version of Euclid's Algorithm. You can use the OCaml infix function `mod` in your function. You cannot use any other functions in your implementation.

10. Implement mutually recursive versions of even and odd functions `my_even x` and `my_odd x`. Your functions should implement the following:

```
bool even(int x) {
  if (x < 0)
    return false;
  else if (x == 0)
    return true;
  else
    return odd(x - 1);
}
bool odd(int x) {
  if (x < 0)
```

```
    return false;
  else if (x == 1)
    return true;
  else
    return even(x - 1);
}
```

For the following functions, you can use the list length, head, and tail functions, list creation (e.g., `[List.hd xs]`*), the cons and append operators, the failwith function as needed, and "normal" OCaml constructs (like let-in, comparison and arithmetic operators, etc.). However, no other helper functions or list operations are allowed.*

11. 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]`.

12. 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").

13. 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").

14. 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.

15. 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]`.

16. 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`.

17. 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.

18. 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.

19. Write a function `my_list_min` that returns the smallest of a given *list* of values. Example: `my_list_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.

Testing. For testing, we'll use our own "poor man's" unit testing approach (instead of a unit testing framework like OUnit). For each function, you will need to write multiple tests to ensure the function works correctly. In particular, your `hw8_test.ml` program should be structured as follows.

```
(*
   Name: <your-name-here>
   File: hw8_tests.ml
   Date: Spring 2023
   Desc: HW8 function unit tests
*)

open Hw8

let msg = "--- Running HW8 Tests --- ";;
print_endline msg;;

(* For equality assertions *)
let assert_equal v1 v2 msg =
  let cond = v1 = v2 in
  assert (if not cond then print_endline ("TEST FAILED: " ^ msg) ; cond)
;;

(* Question 1: my_min tests *)
assert_equal 1 (my_min 1 2) "1 = my_min 1 2";;
assert_equal 1 (my_min 2 1) "1 = my_min 2 1";;
assert_equal 1 (my_min 1 1) "1 = my_min 1 1";;

(* Question 2: my_median tests *)
assert_equal 2 (my_median 1 2 3) "2 = my_median 1 2 3";;
assert_equal 2 (my_median 3 2 1) "2 = my_median 3 2 1";;
...

...
```

To check that your program "passes" the tests, from the command line you will first need to compile your program:

```
ocamlopt -o hw8_tests hw8.ml hw8_tests.ml
```

You can then run the executable:

```
./hw8_tests
```

If your tests succeed you should not get any additional output (other than the “Running HW8 Tests” message.

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. This homework assignment is worth a total of **20 points**. In particular, you will receive one point per function and an additional point for your unit tests. Note that you do not need to create a writeup for this assignment.